Entrega Docker

Relizado por:

- David Massa Gallego
- Julio Martín García

Explicación Entrega 2

Enunciado

1. Utiliza un Vagrantfile para crear una máquina virtual donde puedas crear un cluster de kind: El Vagrantfile debe aprovisionar la instalación de docker las de kind y kubectl pueden realizarse manualmente
2. Dentro de la máquina Vagrant, crea un cluster de kind. Debe ser accesible desde el puerto 8085 de tu máquina local
3. Usando archivos de configuración, despliega una aplicación de Drupal dentro del cluster:

- Usa la imagen oficial más reciente
- Usa una base de datos MySQL desplegada en el cluster (usa la imagen oficial)
- Usa volúmenes persistentes para almacenar los datos de MySQL y Drupal
- En la documentación de la imagen de Drupal se indica los directorios donde se almacenan los datos

- Para el directorio /var/www/html/sites de Drupal, necesitarás usar initContainers para copiar los archivos de configuración

4. Elimina todos los pods y demuestra que los datos persisten

Introducción Entregable 2

En este entregable vamos a proceder a crear una máquina virtual con vagrant utilizando un Vagrantfile para ello. Posteriormente a esto crearemos un clúster donde implementaremos un par de pods:

1. Para el servicio de Drupal
2. Para la base de datos con MySQL

Una vez creado, podremos acceder a nuestro http://localhost:8085 como se nos exige en el enunciado y podremos disfrutar de los servicios que nos ofrece Drupal. Cuando lo configuremos y estemos ya dentro los eliminaremos para probar la persistencia de datos.

Explicación Vagrantfile

Hemos diseñado un Vagrantfile el cual es el siguiente:

```
Vagrant.configure("2") do |config|
  config.vm.define "entrega2-vs" do |vm_config|
    # Configuración básica de la máquina virtual
    vm_config.vm.hostname = "entrega2-vs"
    vm_config.vm.box = "ubuntu/focal64"
    vm_config.vm.provision "shell", path: "provisioning/entrega2_VS.sh"

    #Abrimos el puerto que usamos en la máquina de Vagrant
    config.vm.network "forwarded_port", guest: 8085, host: 8085

    # Sincronizar carpeta con los .yaml a la máquina de vagrant
    vm_config.vm.synced_folder "./files", "/home/vagrant/files"

    # Configuración de recursos
    vm_config.vm.provider "virtualbox" do |vb|
      vb.memory = "4096" # Incrementa la memoria a 4 GB
      vb.cpus = 2        # Incrementa el número de CPUs a 2
    end
  end
end
```

- Vamos a proceder a desglosarlo explicando la estructura de dicho archivo

1.

```
Vagrant.configure("2") do |config|
  config.vm.define "entrega2-vs" do |vm_config|
```

Aqui definimos la máquina virtual que vamos a usar asignándole el nombre de **entrega2-vs**.

2.

```
vm_config.vm.hostname = "entrega2-vs"
vm_config.vm.box = "ubuntu/focal64"
vm_config.vm.provision "shell", path: "provisioning/entrega2_VS.sh"
```

- `vm_config.vm.hostname = "entrega2-vs"`: Nombre del host de la máquina virtal **entrega2-vs**
- `vm_config.vm.box = "ubuntu/focal64"`: Imagen que usará Vagrant -> Ubuntu 20.04(focal64).
- `vm_config.vm.provision "shell", path: "provisioning/entrega2_VS.sh"`: Script de aprovisionamiento para ejecutar configuraciones adicionales al iniciar la máquina.

3.

`config.vm.network "forwarded_port", guest: 8085, host: 8085`: Abrimos el puerto 8085 de la máquina anfrition para redirigirlo al puerto 8085 de la máquina virtual que usaremos con Vagrant para poder acceder a los servicios en la máquina virtual desde el anfitrión usando: `localhost:8085`. 4. `vm_config.vm.synced_folder "./files", "/home/vagrant/files"`: Sincronizamos la carpeta **files** de la máquina anfritiona con la carpeta **/home/vagrant/files** dentro de la máquina virtual. lo que nos permite compartir archivos entre ambos entornos y los cuales usaremos para toda la configuración de Kubernetes. 5.

```
vm_config.vm.provider "virtualbox" do |vb|
      vb.memory = "4096" # Incrementa la memoria a 4 GB
      vb.cpus = 2        # Incrementa el número de CPUs a 2
    end
```

- Configuramos los recursos que nos harán falta en la máquina virtual.
- `vb.memory = "4096"`: Asignamos 4GB de RAM.
- `vb.cpus = 2`: Asignamos 2 CPUs.

Explicación Script de aprovisionamiento

- Aunque en el enunciado sólo se nos exige la instalación de Docker, también hemos procedido a meter en el Script de aprovisionamiento la instalación tanto de **kind** como de **Kubectl**. Vamos a proceder a explicar el Script que se encuentra en la carpeta ***provisioning*** con el nombre de ***entrega2_VS***:

```
#!/bin/bash

# Instalación de Docker
sudo apt-get update -y
sudo apt-get install -y apt-transport-https ca-certificates curl software-
properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
```

```
-
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"
sudo apt-get update -y
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
sudo systemctl start docker
sudo systemctl enable docker

# Añadir al usuario 'vagrant' al grupo de Docker para ejecutar comandos sin
sudo
sudo usermod -aG docker vagrant

# Verificar la instalación de Docker
docker --version

# Actualización de los paquetes que nos vienen de inicio.
sudo apt-get update -y && sudo apt-get upgrade -y

# Instalación Kind
[ $(uname -m) = x86_64 ] && curl -Lo ./kind
https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind

# Comprobación de la instalación de Kind
kind version

# Instalación de kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/kubectl

# Verificar la instalación de kubectl
kubectl version --client --output=yaml
```

1.

```
sudo apt-get update -y
sudo apt-get install -y apt-transport-https ca-certificates curl software-
properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
-
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"
sudo apt-get update -y
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
sudo systemctl start docker
sudo systemctl enable docker
```

En estas líneas de comandos hacemos diferentes cosas, las cuales son las siguientes:

- Actualizamos los paquetes e instalamos dependencias necesarias para Docker.
- Añadimos el repositorio oficial de Docker para Ubuntu 20.04.
- Instalamos Docker y sus componentes.
- Habilitamos y arrancamos el servicio de Docker.

2.

`sudo usermod -aG docker vagrant`: Añadir al usuario 'vagrant' al grupo de Docker para ejecutar comandos sin sudo.

3.

`sudo apt-get update -y && sudo apt-get upgrade -y`: Actualizamos los paquetes que nos vienen de inicio. 4.

```
# Instalación Kind
[ $(uname -m) = x86_64 ] && curl -Lo ./kind
https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind

# Comprobación de la instalación de Kind
kind version
```

Instalamos kind en nuestra máquina de Vagrant. Posteriormente a esto, verificamos que la instalación se ha realizado con éxito. 5.

```
# Instalación de kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/kubectl

# Verificar la instalación de kubectl
kubectl version --client --output=yaml
```

Instalamos kubectl en nuestra máquina de Vagrant. Posteriormente a esto, verificamos que la instalación se ha realizado con éxito. Gracias a esto podremos interactuar con los clústeres de Kubernetes.

Explicación clúster de configuramiento.

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
```

```
    extraPortMappings:
      - containerPort: 30010
        hostPort: 8085
        protocol: TCP
```

Este archivo configura un clúster local de Kubernetes utilizando Kind (Kubernetes in Docker).

- **kind: Cluster**: Indica que va ser un clúster de Kind.
- `apiVersion: kind.x-k8s.io/v1alpha4`: Especificamos la versión de API de configuración de Kind
- **nodes.role: control-plane**: Este nodo se define como nodo maestro, que será el encargado de la configuración del clúster.
- **nodes.role.extraPortMappings.containerPort: 30010**: Es el puerto dentro del nodo de Kubernetes que queremos exponer. En este caso lo usamos para exponer Drupal y poder acceder a este. Lo veremos más tarde cuando expliquemos el servicio de Drupal.
- **nodes.role.extraPortMappings.hostPort: 8085**: Es el puerto de nuestra máquina host que estará enlazado al puerto 30010 del nodo. Cuando accedemos a **localhost:8085** estamos redirigiéndonos al puerto 30010 del nodo, que es donde se encuentra el servicio de Drupal.
- **nodes.role.extraPortMappings.protocol: TCP**: Es el protocolo por excelencia para servicios webs y APIs.

Explicación deployments implementados.

drupal-deployment-vs.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: drupal-deployment-vs
spec:
  selector:
    matchLabels:
      app: drupal
  template:
    metadata:
      labels:
        app: drupal
    spec:
      initContainers:
      - name: init-drupal-setup
        image: drupal:latest
        command: ["/bin/sh", "-c"]
        args:
        - |
          #Crea directorio si no existe
          if [ ! -d /var/www/html/sites/default/files ]; then
            mkdir -p /var/www/html/sites/default/files
          fi
```

```
          #Copia archivo settings.php si no existe
          if [ ! -f "/var/www/html/sites/default/default.settings.php" ];
then
            cp
/opt/drupal/web/core/assets/scaffold/files/default.settings.php
/var/www/html/sites/default/default.settings.php
          fi
          #Crea settings.php desde default.settings.php y configuramos
variables de entorno de la configuración de la BD
          if [ ! -f "/var/www/html/sites/default/settings.php" ]; then
            cp /var/www/html/sites/default/default.settings.php
/var/www/html/sites/default/settings.php

            echo "Añadiendo configuración automática al settings.php..."
            cat <<EOL >> /var/www/html/sites/default/settings.php

          \$databases['default']['default'] = array (
            'database' => getenv('DRUPAL_DB_NAME'),
            'username' => getenv('DRUPAL_DB_USER'),
            'password' => getenv('DRUPAL_DB_PASSWORD'),
            'host' => getenv('DRUPAL_DB_HOST'),
            'port' => '',
            'driver' => 'mysql',
            'prefix' => '',
          );

          #Activamos los Trusted Host Settings, para evitar errores en
Drupal.
          \$settings['trusted_host_patterns'] = array(
            '^.*$', #Aquí tendría que ir los dominios que usaría nuestro
sitio web.
          );

          EOL
          fi

          chmod 777 -R /var/www/html/sites/default
        volumeMounts:
        - name: drupal-data-vs
          mountPath: /var/www/html/sites/default
      containers:
      - name: drupal-vs
        image: drupal:latest
        envFrom:
        - configMapRef:
            name: drupal-env-config
        ports:
        - containerPort: 80
        volumeMounts:
        - name: drupal-data-vs
          mountPath: /var/www/html/sites/default
      volumes:
      - name: drupal-data-vs
```

```
        persistentVolumeClaim:
          claimName: drupal-pvc
```

1.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: drupal-deployment-vs
spec:
  selector:
    matchLabels:
      app: drupal
  template:
    metadata:
      labels:
        app: drupal
```

- **kind**: tipo de recurso (Deployment).
- **apiVersion**: versión de este tipo de recurso (apps/v1).
- **metadata.name**: nombre del despliegue (drupal-deployment-vs).
- **selector.matchLabels**: selecciona aquellos Pods con una etiqueta (app:drupal) para que pertenezcan a este Deployment.
- **template.metadata.labels**: etiqueta (app:drupal) para los Pods que envuelven tu contenedor.

2.

```
spec:
      initContainers:
      - name: init-drupal-setup
        image: drupal:latest
        command: ["/bin/sh", "-c"]
        args:
        - |
          #Crea directorio si no existe
          if [ ! -d /var/www/html/sites/default/files ]; then
            mkdir -p /var/www/html/sites/default/files
          fi
          #Copia archivo settings.php si no existe
          if [ ! -f "/var/www/html/sites/default/default.settings.php" ];
then
            cp
/opt/drupal/web/core/assets/scaffold/files/default.settings.php
/var/www/html/sites/default/default.settings.php
          fi
          #Crea settings.php desde default.settings.php y configuramos
variables de entorno de la configuración de la BD
          if [ ! -f "/var/www/html/sites/default/settings.php" ]; then
            cp /var/www/html/sites/default/default.settings.php
```

```
/var/www/html/sites/default/settings.php

            echo "Añadiendo configuración automática al settings.php..."
            cat <<EOL >> /var/www/html/sites/default/settings.php

          \$databases['default']['default'] = array (
            'database' => getenv('DRUPAL_DB_NAME'),
            'username' => getenv('DRUPAL_DB_USER'),
            'password' => getenv('DRUPAL_DB_PASSWORD'),
            'host' => getenv('DRUPAL_DB_HOST'),
            'port' => '',
            'driver' => 'mysql',
            'prefix' => '',
          );

          #Activamos los Trusted Host Settings, para evitar errores en
Drupal.
          \$settings['trusted_host_patterns'] = array(
            '^.*$', #Aquí tendría que ir los dominios que usaría nuestro
sitio web.
          );

          EOL
          fi

          chmod 777 -R /var/www/html/sites/default
        volumeMounts:
        - name: drupal-data-vs
          mountPath: /var/www/html/sites/default
```

- **spec.initContainers**: Definimos un contenedor de inicialización para que este se ejecute proporcionándonos lo que le indicamos antes de que se ejecute el contenedor principal.
- **initContainers.name**: Nombre del contenedor de inicialización (init-drupal-setup)
- **initContainers.image**: Imagen Docker a usar para este contenedor (drupal:latest), en este caso, la última versión de Drupal.
- **initContainers.command**: Comando que ejecutará el contenedor (/bin/sh -c), iniciando un intérprete de comandos.
- **initContainers.args**: Argumentos para el comando especificado. Aquí se utiliza un script multilinea para realizar configuraciones específicas:
    - **Creación de directorio**: Comprueba si el directorio **/var/www/html/sites/default/files** no existe, y si no, lo crea.
    - **Copia de archivos predeterminados**: Si el archivo **default.settings.php** no existe, se copia desde la ubicación de recursos predeterminada de Drupal.
    - **Creación del archivo settings.php**:
        - Copia **default.settings.php** como **settings.php** si no existe.

    - Añade configuraciones automáticas al final del archivo **settings.php** para definir la base de datos y otros ajustes (como patrones de host de confianza).

- **chmod 777 -R /var/www/html/sites/default**: Cambia los permisos del directorio **sites/default** y sus contenidos. Esto puede provocar que luego salga un warning en Drupal en el caso de que lanzáramos la aplicación de forma real podríamos reconfigurar los permisos para quitarles y así nos quitaríamos el warning del cual se nos avisa.
- **initContainers.volumeMounts**: Monta un volumen en el contenedor:
- **initContainers.name**: Nombre del volumen a montar (drupal-data-vs).
- **initContainers.mountPath**: Ruta dentro del contenedor donde se monta el volumen (/var/www/html/sites/default), asegurando que los cambios persistan.

3.

```
containers:
      - name: drupal-vs
        image: drupal:latest
        envFrom:
        - configMapRef:
            name: drupal-env-config
        ports:
        - containerPort: 80
        volumeMounts:
        - name: drupal-data-vs
          mountPath: /var/www/html/sites/default
```

- **spec.containers.name**: Nombre del contenedor (drupal-vs).
- **spec.containers.image**: Imagen Docker utilizada (drupal:latest).
- **spec.containers.envFrom**: Lista de configuraciones de entorno para el contenedor:
    - **configMapRef**: Indica que se tomarán variables de entorno desde un ConfigMap:
        - **configMapRef.name**: Nombre del ConfigMap (drupal-env-config).
- **spec.containers.ports**: Especifica los puertos expuestos por el contenedor.
    - **ports.containerPort**: Número del puerto (80) que estará expuesto dentro del contenedor.
- **spec.containers.volumeMounts**: Define cómo se conectan los volúmenes al contenedor.
    - **volumeMounts.name**: Nombre del volumen (drupal-data-vs) que se montará en el contenedor.
    - **volumeMounts.mountPath**: Ruta dentro del contenedor donde se montará el volumen (/var/www/html/sites/default).

4.

```

```

```
volumes:
      - name: drupal-data-vs
        persistentVolumeClaim:
          claimName: drupal-pvc
```

- **spec.volumes.name**: Nombre del volumen (drupal-data-vs).
- **volumes.persistentVolumeClaim**: Conecta este volumen a un PersistentVolumeClaim (PVC).
    - **volumes.claimName**: Nombre del PVC (drupal-pvc) que gestiona el almacenamiento. Gracias a esto nuestros datos serán duraderos.

mysql-deployment-vs.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment-vs
spec:
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: MYSQL_ROOT_PASSWORD
        - name: MYSQL_DATABASE
          valueFrom:
            configMapKeyRef:
              name: mysql-config
              key: MYSQL_DATABASE
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: mysql-config
              key: MYSQL_USER
        - name: MYSQL_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: mysql-config
```

```
              key: MYSQL_PASSWORD
        ports:
        - containerPort: 3306
        volumeMounts:
        - name: mysql-data-vs
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-data-vs
        persistentVolumeClaim:
          claimName: mysql-pvc
```

1.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment-vs
spec:
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
```

- **kind**: tipo de recurso (Deployment).
- **apiVersion**: versión de este tipo de recurso (apps/v1).
- **metadata.name**: nombre del despliegue en cuestión (mysql-deployment-vs).
- **selector.matchLabels**: selecciona aquellos Pods con una etiqueta (app:mysql) para que pertenezcan a este Deployment.
- **template.metadata.labels**: etiqueta (app:mysql) para los Pods que envuelven tu contenedor.

2.

```
spec:
      containers:
      - name: mysql
        image: mysql:latest
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: MYSQL_ROOT_PASSWORD
        - name: MYSQL_DATABASE
          valueFrom:
            configMapKeyRef:
```

```
            name: mysql-config
            key: MYSQL_DATABASE
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: mysql-config
              key: MYSQL_USER
        - name: MYSQL_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: mysql-config
              key: MYSQL_PASSWORD
        ports:
        - containerPort: 3306
        volumeMounts:
        - name: mysql-data-vs
          mountPath: /var/lib/mysql
```

- **spec.containers.image**: Indica que se usará la última versión disponible de la imagen oficial de MySQL.
- **spec.containers.env**: Se define una lista de variables de entorno requeridas por MySQL para su configuración:
    - **MYSQL_ROOT_PASSWORD**: Define la contraseña del usuario root de MySQL.
        - **valueFrom.secretKeyRef**: Se toma el valor desde un Secret.
        - **name:mysql-secret**: Nombre del Secret que contiene la contraseña.
        - **key: MYSQL_ROOT_PASSWORD**: Clave específica dentro del Secret.
    - **MYSQL_DATABASE**: Nombre de la base de datos que se creará.
        - **valueFrom.configMapKeyRef**:Se toma el valor desde un ConfigMap.
        - **name: mysql-config**: Nombre del ConfigMap.
        - **key: MYSQL_DATABASE**: Clave específica dentro del ConfigMap.
    - **MYSQL_USER**: Nombre del usuario que se creará en la base de datos.
        - **valueFrom.configMapKeyRef**:Se toma el valor desde un ConfigMap.
        - **name**: mysql-config.
        - **key**: MYSQL_USER.
    - **MYSQL_PASSWORD**: Contraseña para el usuario definido anteriormente.
        - **valueFrom.configMapKeyRef**: Se toma el valor desde un ConfigMap.
        - **name**: mysql-config.
        - **key**: MYSQL_PASSWORD.
- **spec.containers.ports**: Especifica el puerto del contenedor, en este caso es el predeterminado por mySQL.

- **spec.containers.volumeMounts**: Conecta un volumen persistente al contenedor para almacenar datos de MySQL de forma persistente.

3.

```
volumes:
      - name: mysql-data-vs
        persistentVolumeClaim:
          claimName: mysql-pvc
```

- **spec.volumes.name**: Nombre del volumen (mysql-data-vs).
- **volumes.persistentVolumeClaim**: Conecta este volumen a un PersistentVolumeClaim (PVC).
    - **volumes.claimName**: Nombre del PVC (mysql-pvc) que gestiona el almacenamiento. Gracias a esto nuestros datos serán duraderos.

Explicación configMap (mysql-config.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  MYSQL_DATABASE: drupal
  MYSQL_USER: vs
  MYSQL_PASSWORD: vs
```

- Hemos creado un configMap para las variables de entorno de mysql, así podemos evitar que se hardcodeen credenciales y es una buena práctica que siempre tendríamos que implementar para mejorar la seguridad de lo que estemos desarrollando.

Explicación secret (mysql-secret.yaml)

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
stringData:
  MYSQL_ROOT_PASSWORD: example
```

- Una explicación similar a la anterior, lo único que los **secret** lo usamos explícitamente para guardar información sensible. En este caso, solo hemos guardado la contraseña Root de la BD. Igual que evitamos hardcodear credenciales como hemos mencionado previamente, en este caso contamos con

un plus extra y es que, aunque los datos no estén cifrados en un archivo **secret** se almacenan en base64 lo que evita la lectura directa.

Explicación servicios implementados

drupal-service-vs.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: drupal-service-vs
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      nodePort: 30010
  selector:
    app: drupal
```

- Dado a que los archivos de ahora en adelante no son tan extensos, en vez de desglosarlos vamos a hacer una explicación general de lo realmente importante.
- En este caso creamos un servicio que nos permite exponer aplicaciones que se ejecutan en Pods para que puedan ser accesibles dentro o fuera del clúster. En este caso contamos con que estamos exponiendo una aplicación de Drupal que es la que se nos exige en el enunciado.
- Es importante destacar que el tipo de servicio con el que contamos es **NodePort** lo que nos permite que la aplicación sea accesible desde fuera del clúster mediante el puerto de los nodos que van entre 30000–32767. En este caso hemos elegido el 30010.
- El puerto 80, es el puerto interno del servicio el cual será accesible dentro del clúster
- EL nodePort es 30010 que es el puerto que usaremos para exponer el servicio en cada nodo del clúster.

mysql-service-vs.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service-vs
spec:
  type: ClusterIP #Limito el acceso de MYSQL solo a que este dentro del
cluster.
  ports:
    - name: mysql
      port: 3306
  selector:
    app: mysql
```

- Este archivo **.yaml** está exclusivamente diseñado para exponer una base de datos MySQL únicamente dentro del clúster
- Es importante destacar que el tipo de servicio con el que contamos es **ClusterIP**, es el que viene por defecto con kubernetes. No nos interesa exponerlo fuera con **NodePort** ya que en este caso no queremos exponer la BD fuera del clúster.
- El puerto 3306, es el puerto interno del servicio el cual será accesible dentro del clúster. Es el puerto predeterminado de MySQL, lo que facilita la conexión entre los Pods internos del clúster.

Explicación PVCs implementados

drupal-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: drupal-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

- En este archivo creamos un PVC que es una solicitud de almacenamiento persistente el cual va ser utilizada en este caso por nuestro pod de Drupal, para que persista la información, lo cual nos permita que si borramos un pod los datos sigan ahí.
- Gracias a **ReadWriteOnce** nos permite que el volumen sea montado como lectura y escritura por un único nodo.
- Al final del archivo podemos ver como solicitamos **2Gi** de almacenamiento persistente del clúster

mysql-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

- La misma explicación de antes en todos los aspectos, sólo que en este caso persistirán la información de la BD la cual es MySQL como bien sabemos.

Explicación script.sh

```
#!/bin/bash
kubectl create configmap drupal-env-config --from-env-file=.env
kubectl apply -f mysql-secret.yaml
kubectl apply -f mysql-config.yaml
kubectl apply -f drupal-pvc.yaml
kubectl apply -f mysql-pvc.yaml
kubectl apply -f drupal-deployment-vs.yaml
kubectl apply -f mysql-deployment-vs.yaml
kubectl apply -f drupal-service-vs.yaml
kubectl apply -f mysql-service-vs.yaml
```

Dado a que se han sometido muchísimas pruebas para llevar acabo esta práctica y que todo finalmente fuera bien, diseñamos un script para ahorrarnos tiempo a la hora de ir haciendo pruebas dado a que continuamente se modificaban algo del los archivos **.yaml**. Gracias a este script hemos podido ahorrarnos tiempo y ser más eficientes.

Aquí contamos con todos los comandos necesarios para que podamos desplegar todo lo preparado con Kubernetes.

- `kubectl create configmap drupal-env-config --from-env-file=.env` : Comando para crear un configMap de Drupal a raíz del archivo .env donde almacenamos las variables del entorno.
- `kubectl apply -f mysql-secret.yaml`: Comando para crear el secret de mySQL.
- `kubectl apply -f mysql-config.yaml`: Comando para crear el configMap de las variables del entorno de mySQL. ***Desplegar volumenes persistentes***

```
kubectl apply -f drupal-pvc.yaml
kubectl apply -f mysql-pvc.yaml
```

- Desplegamos volúmenes persistentes de drupal y mysql.

Desplegar deployments

```
kubectl apply -f drupal-deployment-vs.yaml
kubectl apply -f mysql-deployment-vs.yaml
```

- Desplegamos deployments de drupal y mysql.

Desplegar servicios

```
kubectl apply -f drupal-service-vs.yaml
kubectl apply -f mysql-service-vs.yaml
```

- Desplegamos servicios de drupal y mysql.

Suponiendo que introdujeramos los comandos línea por línea, tendríamos que seguir estrictamente el que usamos en el script.

Explicación .env (variables del entorno)

```
DRUPAL_DB_HOST=mysql-service-vs
DRUPAL_DB_NAME=drupal
DRUPAL_DB_USER=vs
DRUPAL_DB_PASSWORD=vs
DRUPAL_SITE_NAME=pruebaVS
DRUPAL_SITE_MAIL=pruebaVSsitio@example.com
DRUPAL_ACCOUNT_NAME=vs
DRUPAL_ACCOUNT_PASS=vs
DRUPAL_ACCOUNT_MAIL=pruebaVScuenta@example.com
```

La usamos gracias a configMap por un comando el cual es el siguiente: `kubectl create configmap drupal-env-config --from-env-file=.env` . Con este comando creamos un configMap a raíz de el archivo oculto de entorno de variables que explicamos.

- Hemos conseguido implementar las variables del entorno para la configuración de la base de datos.
- Gracias a la profunda investigación que se ha hecho de esto, hemos conseguido dar con la herramienta llamada **Drush** la cual es una herramienta de línea de comandos ampliamente usada para gestionar sitios web basados en Drupal. Creemos que con esto se podría implementar las variables del entorno restantes, y así hacer una instalación web automatizada de Drupal, pero ha provocado muchos fallos a la hora de implementarlo con Kubernetes. Aún así no tenemos la certeza de que esto fuera cierto, pero intuimos que podría ser.

Conceptos aprendidos en este entregable

Explicación configMap de Kubernetes

Un configmap es un objeto de la API utilizado para almacenar datos no confidenciales en el formato clave-valor. Los Pods pueden utilizar los ConfigMaps como variables de entorno, argumentos de la linea de comandos o como ficheros de configuración en un Volumen.

Un ConfigMap te permite desacoplar la configuración de un entorno específico de una imagen de contenedor, así las aplicaciones son fácilmente portables.

Explicación Secrets de Kubernetes

Un Secret es un objeto que contiene una pequeña cantidad de datos confidenciales como contraseñas, un token, o una llave. Tal información podría ser puesta en la especificación de un Pod o en una imagen; poniendolo en un objeto de tipo Secret permite mayor control sobre como se usa, y reduce el riesgo de exposicición accidental.

Bibliografía

https://superuser.com/questions/535870/drupal-unable-to-create-files-on-var-www-html-drupal-directory
https://www.drupal.org/docs/administering-a-drupal-site/troubleshooting-drupal/fix-drupal-clean-urls-problems
https://kubernetes.io/es/docs/concepts/configuration/configmap/
https://kubernetes.io/es/docs/concepts/configuration/secret/ https://www.drush.org/13.x/
https://hub.docker.com//*drupal https://hub.docker.com/*/mysql