Ejercicio 1: Trabajar con Servicios Kubernetes

A. Crear un Servicio ClusterIP

1. **Desplegar un Pod**:
   - Primero, crea un Pod que será expuesto por el servicio. Aquí tienes un ejemplo simple utilizando nginx:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

   - Aplica el manifiesto con: `kubectl apply -f nginx-pod.yaml` .

> **Importante:** Crea un segundo archivo con las mismas configuraciones cambiando el nombre a `nginx-pod2` y el archivo a `nginx-pod2.yaml` y aplica el manifiesto usando: `kubectl apply -f nginx-pod2.yaml`

2. **Crear un Servicio ClusterIP**:
   - Ahora, crea un servicio ClusterIP para exponer el Pod:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

   - Aplica este manifiesto con: `kubectl apply -f nginx-service.yaml` .

3. **Obtener detalles del servicio**:

```
kubectl get service nginx-service
```

   - Este comando mostrará la dirección IP interna asignada al servicio.
4. **Conectar desde otro Pod**:
   - Primero, accede a otro al pod llamado `nginx-pod2` en el mismo clúster
   - Ejemplo: `kubectl exec -it nginx-pod2 -- /bin/bash`
5. **Probar la conectividad**:
   - Dentro del Pod, utiliza un comando como `curl` para probar la conectividad al servicio:
   - Ejemplo: `curl http://<IP-del-servicio-nginx-service>`
   - Reemplaza `<IP-del-servicio-nginx-service>` con la dirección IP que obtuviste en el paso 1.

Este método te permitirá verificar si el servicio ClusterIP está funcionando correctamente y es accesible desde otros Pods dentro del clúster.

B. Crear un Servicio NodePort

1. **Crear un Servicio NodePort**:
   - Modifica el tipo de servicio en tu manifiesto de servicio a `NodePort` o crea uno nuevo. Por ejemplo:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

   - Aplica el manifiesto con `kubectl apply -f nginx-nodeport.yaml` .
2. **Verificar el Servicio NodePort**:
   - Usa `kubectl get service nginx-nodeport` para obtener el puerto asignado.
   - Puedes acceder al servicio desde fuera del clúster utilizando la IP de cualquier nodo del clúster y el puerto asignado.

> **Nota:** al usar docker desktop, el nodo de Kubernetes es el mismo que la maquina local, entonces deberás acceder usando **localhost:NODEPORT**

C. Crear un Servicio LoadBalancer

1. **Crear un Servicio LoadBalancer**:
   - Modifica el tipo de servicio en tu manifiesto de servicio a `LoadBalancer`. Ejemplo:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-loadbalancer
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

   - Aplica el manifiesto con `kubectl apply -f nginx-loadbalancer.yaml`.
2. **Verificar el Servicio LoadBalancer**:
   - Usa `kubectl get service nginx-loadbalancer` para ver los detalles del servicio.
   - Este servicio asignará una dirección IP externa que puedes usar para acceder al servicio desde fuera del clúster.

D. Crear un Servicio ExternalName

1. **Crear un Servicio ExternalName**:
   - Este tipo de servicio redirige a un nombre DNS externo. Ejemplo:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  type: ExternalName
  externalName: example.com
```

   - Aplica el manifiesto con `kubectl apply -f external-service.yaml`.
2. **Revisa los Detalles del Servicio**:

- Utiliza `kubectl get service external-service` para ver los detalles del servicio `ExternalName`. Esto confirmará la configuración del servicio y mostrará el nombre DNS externo al que apunta.

3. **Probar la Resolución DNS desde un Pod**:
   - Accede a un Pod en tu clúster (cualquier Pod servirá) usando `kubectl exec -it <nombre-del-pod> -- /bin/bash` o `/bin/sh`.
   - Ejecuta `apt-get install -y dnsutils` para instalar `nslookup`.
   - Dentro del Pod, utiliza el comando `nslookup` seguido del nombre del servicio `ExternalName`. Por ejemplo, `nslookup external-service`.
   - Esto debería resolver al DNS externo configurado en el servicio `ExternalName`.

Ejercicio 2: Topología de los Servicios

Objetivo:
Comprender las prácticas recomendadas y los enfoques no recomendados en la topología de servicios en Kubernetes.

Topología de Servicios

1. **Crear aplicación para conectarse a un segundo servicio**
   - Crearemos una aplicación sencilla para conectarnos a `mongo` el cual estará en otro pod
   - ejecuta el comando `npm init -y` para crear el proyecto

```
const express = require('express');
const { MongoClient } = require('mongodb');

const app = express();
const port = 3000;
const url = 'mongodb://database-service:27017'; // Cambia 'database-service' por el nombr

app.get('/', async (req, res) => {
    try {
        const client = new MongoClient(url);
        await client.connect();
        res.send("Conectado a MongoDB");
        client.close();
    } catch (err) {
        res.status(500).send("Error al conectar a MongoDB: " + err.message);
    }
});

app.listen(port, () => {
    console.log(`Aplicación corriendo en http://localhost:${port}`);
});
```

- Instala los paquetes necesarios

```
npm install express
npm install mongob
```

2. **Crea imagen de la aplicación**
   - Crearás un archivo `Dockerfile` para construir la imagen y desplegarla a un repositorio, por ejemplo docker hub

```
FROM node:latest
WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install

COPY *.js ./

EXPOSE 3000

CMD ["node", "index.js"]
```

3. **Construye la imagen y despliegala a un repositorio**

```
docker build -t [TU USUARIO DOCKERHUB]/mongo-node .
docker push [TU USUARIO DOCKERHUB]/mongo-node
```

4. **Crea el manifiesto para desplegar los pods**

- Frontend `mongo-node.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-node
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mongo-node
  template:
    metadata:
      labels:
        app: mongo-node
    spec:
      containers:
      - name: mongo-node
        image: alien777/mongo-node:latest
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: mi-node-app-service
spec:
  type: LoadBalancer
  ports:
    - port: 8081
      targetPort: 3000
  selector:
    app: mongo-node
```

- Database pod `Database.yaml`

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: database
  labels:
    app: database
spec:
  containers:
  - name: db
    image: mongo
---
apiVersion: v1
kind: Service
metadata:
  name: database-service
spec:
  selector:
    app: database
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

- Aplica los manifiestos con `kubectl -f <nombre del manifiesto>.yaml apply`
- Verifica los servicios y accede al servicio de `mi-node-app-service`