Ejercicio 1: Trabajo con Labels y Selectors

Objetivo:

Comprender cómo utilizar labels y selectors para organizar y seleccionar recursos en Kubernetes.

Pasos:

1. **Crear Pods con Diferentes Labels**
   - Comando para crear un pod con un label específico:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    app: myapp
    env: development
spec:
  containers:
  - name: nginx
    image: nginx
```

   - Guarda esto como `pod1.yaml` y crea el pod con `kubectl apply -f pod1.yaml`.
   - Repite el proceso para crear varios pods con diferentes labels.
2. **Usar Selectors para Filtrar Pods**
   - Usa el comando `kubectl get pods -l <label>=<value>` para seleccionar pods basados en sus labels.
   - Por ejemplo: `kubectl get pods -l app=myapp` seleccionará todos los pods con el label `app=myapp`.

Ejercicio:

Crear al menos tres pods con diferentes sets de labels y luego usar selectores para filtrarlos de diferentes maneras.

Ejercicio 2: Gestión de Pods y sus Estados

Objetivo:

Entender el ciclo de vida de un pod y cómo Kubernetes maneja los estados de error.

Pasos:

1. **Crear un Pod y Observar su Ciclo de Vida**
   - Usa el siguiente manifiesto para crear un pod simple:

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

   - Aplica este manifiesto y observa el estado del pod con `kubectl get pod lifecycle-pod`.
2. **Simulación de Fallos**
   - Para simular un fallo, puedes cambiar la imagen del contenedor a una que no existe y aplicar el cambio.
   - Observa cómo el estado del pod cambia y cómo Kubernetes intenta reiniciar el contenedor.

Ejercicio:

Experimentar con diferentes formas de causar fallos en un pod y observar cómo Kubernetes reacciona.

Ejercicio 3: Implementación de Secrets y Config Maps

Objetivo:

Aprender a crear y utilizar Secrets y Config Maps en Kubernetes.

Pasos:

1. **Crear un Secret**
   - Primero, crea un secret. Por ejemplo, para guardar una contraseña:

```
kubectl create secret generic my-secret --from-literal=password=mypassword
```

2. **Crear un Config Map**
   - Crea un Config Map con un archivo de configuración o valores literales. Ejemplo:

```
kubectl create configmap my-config --from-literal=apiUrl=http://miapi.com --from-literal=
```

3. **Montar Secret y Config Map en un Pod**
   - Usa el siguiente manifiesto como ejemplo para montar el Secret y el Config Map en un Pod y llamalo `pod-secret.yaml` :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: nginx
    env:
      - name: PASSWORD
        valueFrom:
          secretKeyRef:
            name: my-secret
            key: password
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
  volumes:
  - name: config-volume
    configMap:
      name: my-config
```

4. **Verifica que el pod pueda acceder al secret y al config map**
   - Accede al contenedor y verifica la variable de entorno:

```
kubectl exec -it mypod -- /bin/sh
echo $PASSWORD
```

   - Aún dentro del contenedor, verifica los archivos del Config Map:

```
ls /etc/config
cat /etc/config/<nombre-del-archivo>
```

Ejercicio 3.1: Interactuando con secrets y ConfigMaps desde aplicaciones

Desarrollar la Aplicación Node.js

1. **Crear el Proyecto**:
   Inicia un nuevo proyecto Node.js usando `npm init -y` y crea un archivo `index.js`.
2. **Leer Variables de Entorno**:
   En tu aplicación, utiliza `process.env` para leer las variables de entorno. Ejemplo en `index.js`:

```
const express = require('express');
const app = express();

// Leer variables de entorno
const password = process.env.PASSWORD;
const apiUrl = process.env.API_URL;
const apiToken = process.env.API_TOKEN;

app.get('/', (req, res) => {
  res.send(`Password: ${password}, API URL: ${apiUrl}, API Token: ${apiToken}`);
});

const port = 3000;
app.listen(port, () => {
  console.log(`Servidor ejecutándose en http://localhost:${port}`);
});
```

3. **Añadir Express**:
   Si aún no lo has hecho, agrega Express a tu proyecto:

```
npm install express
```

4. **Crear Dockerfile**
   Crea tu dockerfile con lo siguiente:

```
FROM node:latest

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

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

5. **Construye la imagen**

```
docker build -t [TU USUARIO DOCKERHUB]/secretcm .
```

6. **Sube la imagen a dockerHub**

```
docker push [TU USUARIO DOCKERHUB]/secretcm
```

7. **Crear el Deployment y el service**

   Crea tu manifiesto de Kubernetes llamado `deployment.yaml` incluyendo montajes del Secret y del ConfigMap

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-node-secretcm
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mi-node-secretcm
  template:
    metadata:
      labels:
        app: mi-node-secretcm
    spec:
      containers:
      - name: mi-node-secretcm
        image: [TU USUARIO DOCKERHUB]/secretcm:latest
        ports:
        - containerPort: 3000
        env:
        - name: PASSWORD
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: password
        - name: API_URL
          valueFrom:
            configMapKeyRef:
              name: my-config
              key: apiUrl
        - name: API_TOKEN
          valueFrom:
            configMapKeyRef:
              name: my-config
              key: apiToken
---
apiVersion: v1
kind: Service
metadata:
  name: mi-node-app-service
spec:
  type: LoadBalancer
  ports:
```

```
    - port: 80
      targetPort: 3000
  selector:
    app: mi-node-secretcm
```

8. **Aplica el manifiesto**

```
kubectl -f deployment.yaml apply
```

9. **Revisa la aplicación ingresando a localhost y el puerto que este usa en este caso** `80`

Ejercicio 4: Uso de ReplicaSets

Objetivo:

Comprender cómo los ReplicaSets aseguran un número deseado de réplicas de un Pod.

Pasos:

1. **Crear un ReplicaSet**
   - Utiliza el siguiente manifiesto de ejemplo para crear un ReplicaSet:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: nginx
        image: nginx
```

   - Aplica este manifiesto y observa cómo Kubernetes crea tres réplicas del pod.

2. **Escalabilidad Manual de ReplicaSets**
    - Cambia el número de réplicas en el manifiesto y aplica el cambio.
    - Observa cómo el número de pods en el ReplicaSet aumenta o disminuye según la configuración.

Ejercicio:

Experimentar con diferentes configuraciones de ReplicaSet, incluyendo la modificación del número de réplicas y la observación de cómo Kubernetes gestiona los cambios.

Ejercicio 5: Despliegues con Deployments

Objetivo:

Aprender a crear y gestionar Deployments en Kubernetes, incluyendo la realización de actualizaciones.

Pasos:

1. **Crear un Deployment**
    - Usa el siguiente manifiesto para crear un Deployment:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

    - Aplica este manifiesto y observa cómo se crean los pods.
2. **Actualizar el Deployment**

- Actualiza la imagen del Deployment a una versión más reciente (por ejemplo, `nginx:1.16.1`).
- Aplica el cambio y observa el proceso de despliegue. Kubernetes debería realizar un rolling update sin tiempo de inactividad.

Ejercicio:

Realizar varios cambios en el Deployment, incluyendo actualizaciones de imagen y configuración, y observar cómo Kubernetes maneja estos cambios.

Ejercicio 6: Exploración de DaemonSets

Objetivo:

Entender el propósito de los DaemonSets y cómo se despliegan en un clúster de Kubernetes.

Pasos:

1. **Crear un DaemonSet**
   - Utiliza el siguiente manifiesto para crear un DaemonSet:

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-daemonset
spec:
  selector:
    matchLabels:
      name: mydaemon
  template:
    metadata:
      labels:
        name: mydaemon
    spec:
      containers:
      - name: mydaemon-container
        image: nginx
```

   - Aplica este manifiesto y observa cómo se crea un pod en cada nodo del clúster.

Ejercicio 7: Creación y Gestión de Jobs

Objetivo:

Aprender a crear y manejar Jobs en Kubernetes para ejecutar tareas que deben completarse una vez.

Pasos:

1. **Crear un Job**
   - Usa el siguiente manifiesto para crear un Job simple:

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  template:
    spec:
      containers:
      - name: my-job
        image: busybox
        command: ["sh", "-c", "echo Hello Kubernetes! && sleep 30"]
      restartPolicy: Never
```

   - Aplica este manifiesto y observa la ejecución del job con `kubectl get jobs`.
2. **Observar la Ejecución y el Estado Final del Job**
   - Usa `kubectl describe job my-job` para ver los detalles del Job y su estado.
   - Observa los logs del pod asociado al job con `kubectl logs <nombre-del-pod>`.

Ejercicio:

Crear Jobs con diferentes especificaciones y comandos para comprender cómo se ejecutan y completan en Kubernetes.

Ejercicio 8: Programación de CronJobs

Objetivo:

Entender cómo crear y gestionar CronJobs para realizar tareas en intervalos programados.

Pasos:

1. **Crear un CronJob**
   - Utiliza el siguiente manifiesto para crear un CronJob:

```
apiVersion: batch/v1
kind: CronJob
metadata:
 name: my-cronjob
spec:
 schedule: "*/1 * * * *"
 jobTemplate:
   spec:
     template:
       spec:
         containers:
         - name: my-cronjob
           image: busybox
           args:
           - /bin/sh
           - -c
           - date; echo Hello from the Kubernetes cluster
         restartPolicy: OnFailure
```

   - Este CronJob se ejecutará cada minuto. Aplica el manifiesto y observa la programación con `kubectl get cronjob`.
2. **Configurar y Observar la Ejecución Programada**
   - Usa `kubectl describe cronjob my-cronjob` para ver los detalles y el estado del CronJob.
   - Observa las ejecuciones programadas y sus resultados.

Ejercicio:

Experimentar con diferentes horarios y comandos en los CronJobs para entender su funcionamiento y aplicaciones prácticas.