04 - Despliegue K8s DevOps

Tipo	Area Temática	Duración
Guia	Automatización de despliegues en la nube	2 horas

Tema:

Despliegue de Aplicaciones Contenerizadas desde Pipelines

Objetivo de la Práctica

El objetivo de esta práctica es que implementes un pipeline de CI/CD en Azure DevOps para compilar una imagen Docker de una aplicación Node.js y desplegarla en un clúster de Kubernetes gestionado en AWS EKS. Utilizarás un pipeline de build para la compilación de la imagen y un pipeline de release para el despliegue.

Prerrequisitos

- Acceso a una cuenta de Azure DevOps.
- Acceso a un clúster de Kubernetes en AWS EKS.
- Conocimiento básico de Docker, Kubernetes, y pipelines de CI/CD.
- Instalación de herramientas necesarias: Docker, kubectl, AWS CLI, y Azure CLI.

Paso 1: Configuración del Repositorio de Código Fuente

1. **Crear un Repositorio en Azure Repos o GitHub**:
    - Si utilizas **Azure Repos**:

```
az repos create --name MyAppRepo
```

- Si utilizas **GitHub**, simplemente crea un nuevo repositorio en GitHub.

2. **Clonar el Repositorio a tu Máquina Local**:

```
git clone https://your-repo-url.git
cd MyAppRepo
```

3. **Añadir el Código de la Aplicación Node.js**:

- Añade una aplicación Node.js sencilla al repositorio. Puedes crear un archivo `app.js` con el siguiente contenido básico:

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

app.get('/', (req, res) => {
  res.send('Hello World from Node.js app!');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

- Añade un archivo `package.json` con las dependencias necesarias:

```
{
  "name": "nodejs-app",
  "version": "1.0.0",
  "description": "A simple Node.js app",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
```

```
    "express": "^4.17.1"
  }
}
```

4. **Crear un Dockerfile**:
   - En el directorio raíz de tu proyecto, crea un archivo `Dockerfile` con el siguiente contenido:

```
# Usar la imagen base de Node.js
FROM node:14

# Establecer el directorio de trabajo en el contenedor
WORKDIR /usr/src/app

# Copiar el archivo package.json e instalar dependencias
COPY package*.json ./
RUN npm install

# Copiar el código de la aplicación
COPY . .

# Exponer el puerto que usará la aplicación
EXPOSE 3000

# Comando para ejecutar la aplicación
CMD ["npm", "start"]
```

5. **Commit y Push de los Cambios**:

```
git add .
git commit -m "Add Node.js application and Dockerfile"
git push origin main
```

Paso 2: Configuración del Pipeline de Build en Azure DevOps

1. **Crear un Proyecto en Azure DevOps**:
   - Navega a Azure DevOps y crea un nuevo proyecto.
2. **Configurar un Pipeline de Build**:
   - Ve a la sección **Pipelines** y selecciona **New Pipeline**.
   - Conecta el pipeline a tu repositorio de código (Azure Repos o GitHub).
3. **Definir la Configuración del Pipeline de Build**:
   - En el archivo `azure-pipelines.yml` dentro de tu repositorio, define la configuración del pipeline para construir la imagen de Docker. Aquí tienes un ejemplo básico:

```
trigger:
  - main

pool:
  vmImage: 'ubuntu-latest'

steps:
  - task: NodeTool@0
    inputs:
      versionSpec: '14.x'
    displayName: 'Install Node.js'

  - script: |
      npm install
    displayName: 'Install dependencies'

  - task: Docker@2
    inputs:
      containerRegistry: '<nombre_de_tu_registro>'
      repository: 'nodejs-app'
```

```
      command: 'buildAndPush'
      Dockerfile: '**/Dockerfile'
      tags: |
        $(Build.BuildId)
```

4. **Guardar y Ejecutar el Pipeline de Build**:
   - Guarda y ejecuta el pipeline.
   - Verifica que la imagen Docker se construya y se envíe correctamente a tu registro de contenedores (Azure Container Registry o Amazon ECR).

Paso 3: Configuración de la Conexión entre Azure DevOps y AWS EKS

1. **Configurar AWS CLI**:
   - Asegúrate de tener configuradas tus credenciales de AWS en tu máquina local:

```
aws configure
```

2. **Obtener el kubeconfig para tu Clúster de EKS**:

```
aws eks --region <tu-región> update-kubeconfig --name <nombre-de-tu-cluster>
```

3. **Configurar un Servicio en Azure DevOps para Conectar con AWS**:
   - Ve a **Project settings** > **Service connections**.
   - Configura un nuevo servicio para AWS utilizando tus credenciales de AWS.

Paso 4: Configuración del Pipeline de Release para Desplegar la Aplicación

1. **Crear un Pipeline de Release en Azure DevOps**:

- Ve a **Pipelines** > **Releases** y selecciona **New Release Pipeline**.
- Configura un nuevo stage que desplegará la aplicación en EKS.

2. **Agregar un Artefacto desde el Pipeline de Build**:
   - En la sección **Artefacts**, selecciona el pipeline de build que creaste anteriormente como fuente de artefactos.
3. **Configurar la Tarea de Despliegue**:
   - Añade una tarea de **Kubernetes** en el stage de release para desplegar la aplicación en EKS.
   - Configura la tarea para utilizar el artefacto (imagen Docker) generado en el pipeline de build.
4. **Crear un Archivo de Manifesto para Kubernetes**:
   - Crea un archivo `deployment.yml` en tu repositorio con la siguiente estructura básica:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodejs-app-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nodejs-app
  template:
    metadata:
      labels:
        app: nodejs-app
    spec:
      containers:
      - name: nodejs-app-container
        image: <nombre_de_tu_registro>/nodejs-app:$(Build.
BuildId)
```

```
        ports:
        - containerPort: 3000
```

5. **Actualizar el Pipeline de Release**:
   - Configura la tarea de despliegue para aplicar el archivo `deployment.yml` al clúster de EKS:

```
steps:
  - script: |
      kubectl apply -f deployment.yml
    displayName: 'Deploy to Kubernetes'
```

6. **Ejecutar el Pipeline de Release**:
   - Ejecuta el pipeline de release para desplegar la aplicación en EKS.
   - Verifica el estado del despliegue utilizando `kubectl`:

```
kubectl get pods
```

7. **Acceder a la Aplicación**:
   - Si configuraste un servicio de tipo LoadBalancer en Kubernetes, podrás acceder a la aplicación a través de la IP pública asignada por AWS.

Paso 5: Solución de Problemas y Optimización

1. **Verificar Logs del Pipeline**:
   - Revisa los logs en Azure DevOps si hay algún problema durante la ejecución del pipeline.
2. **Resolver Problemas Comunes**:
   - **Problema de Conexión**: Asegúrate de que tus credenciales de AWS están configuradas correctamente en Azure DevOps.
   - **Fallos en el Despliegue**: Revisa la sintaxis del archivo YAML y asegúrate de que la imagen Docker se ha construido y subido correctamente.

3. **Optimizar el Pipeline**:
    - Considera añadir pasos de optimización como caché de dependencias o test paralelos para mejorar el tiempo de ejecución del pipeline.

Conclusión

Has completado la práctica de despliegue de una aplicación Node.js contenerizada desde pipelines en Azure DevOps. Ahora deberías sentirte más cómodo configurando pipelines de build y release, así como desplegando aplicaciones en un entorno de Kubernetes gestionado en AWS EKS.

Si encuentras problemas o necesitas ayuda adicional, no dudes en consultar la documentación oficial de Azure DevOps y AWS EKS.