Curso de desarrollo de software

Objetivo: Entender los principios fundamentales de DevOps, la historia detrás de su creación, y lo que DevOps es y no es.

Instrucciones:

1. **Lectura y reflexión:**

   Lee el texto proporcionado (Lectura 1), enfocándote en los siguientes puntos clave:
   - ¿Qué es DevOps?
   - Historia y antecedentes de DevOps.
   - Diferencias entre los equipos de desarrollo y operaciones en el pasado.
   - Principios fundamentales de DevOps (centrado en el cliente, equipos autónomos y multifuncionales, mejora continua, automatización).
   - Qué NO es DevOps.

2. **Preguntas de reflexión:**
   - **Pregunta 1:** ¿Por qué surgió la necesidad de DevOps en el desarrollo de software?
   - **Pregunta 2:** Explica cómo la falta de comunicación y coordinación entre los equipos de desarrollo y operaciones en el pasado ha llevado a la creación de DevOps.
   - **Pregunta 3:** Describe cómo el principio de mejora continua afecta tanto a los aspectos técnicos como culturales de una organización.
   - **Pregunta 4:** ¿Qué significa que DevOps no se trata solo de herramientas, individuos o procesos?
   - **Pregunta 5:** Según el texto, ¿cómo contribuyen los equipos autónomos y multifuncionales a una implementación exitosa de DevOps?

Aplicar los conceptos de DevOps en un entorno práctico, configurando un pipeline básico de CI/CD para un proyecto de software y experimentando con la automatización de procesos en un entorno local utilizando Docker.

Instrucciones:

1. Configuración del entorno

- Proyecto: Utilizaremos una aplicación web sencilla utilizando Node.js. Esta aplicación tendrá una API REST básica con un único endpoint que devuelve un mensaje de "Hello, World!".

- Paso a paso:

1. Inicializa el proyecto de Node.js:

```
mkdir devops-practice
cd devops-practice
npm init -y
```

2. Instala las dependencias necesarias:

```
npm install express jest
```

3. Crea la estructura del proyecto:

```
mkdir src tests
touch src/app.js tests/app.test.js
```

4. Implementa la API REST en src/app.js:

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
  res.send('Hello, World!');
});

const port = process.env.PORT || 3000;
app.listen(port, () => {
  console.log(Server running on port ${port});
});

module.exports = app;
```

5. Escribe un test básico en tests/app.test.js:

```
const request = require('supertest');
const app = require('../src/app');

describe('GET /', () => {
  it('should return Hello, World!', async () => {
```

```
        const res = await request(app).get('/');
        expect(res.statusCode).toEqual(200);
        expect(res.text).toBe('Hello, World!');
      });
    });
```

6. Configura el script de test en package.json:

```
{
  "name": "devops-practice",
  "version": "1.0.0",
  "scripts": {
    "test": "jest"
  },
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "jest": "^27.0.0",
    "supertest": "^6.1.3"
  }
}
```

2. Pipeline CI/CD

Parte 1: Configura integración continua (CI) con GitHub Actions

- Crea un archivo de configuración para GitHub Actions:

1. Crea la estructura para GitHub Actions:

```
mkdir -p .github/workflows
touch .github/workflows/ci.yml
```

2. Define el flujo de trabajo en .github/workflows/ci.yml:

```yaml
name: CI Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test
```

3. Sube el código a GitHub:

- Crea un nuevo repositorio en GitHub y empuja tu código:

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin <your-repo-url>
git push -u origin main
```

Parte 2: Configura entrega continua (CD) con Docker

- Crea un archivo Docker para contenerizar la aplicación:

1. Crea un archivo Dockerfile:

```
# Usa la imagen oficial de Node.js
FROM node:14

# Establece el directorio de trabajo en el contenedor
WORKDIR /app

# Copia los archivos package.json y package-lock.json
COPY package*.json ./

# Instala las dependencias
RUN npm install

# Copia el resto de los archivos de la aplicación
COPY . .

# Expone el puerto en el que la aplicación correrá
EXPOSE 3000
```

```
# Comando para iniciar la aplicación
CMD ["node", "src/app.js"]
```

2. Construye la imagen de Docker:

```
docker build -t devops-practice .
```

3. Corre el contenedor localmente:

```
docker run -p 3000:3000 devops-practice
```

- Automatiza el despliegue con GitHub Actions:

1. Actualiza el archivo .github/workflows/ci.yml para construir y desplegar la imagen de Docker:

```
name: CI/CD Pipeline
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2
```

```
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test

      - name: Build Docker image
        run: docker build -t devops-practice .

      - name: Run Docker container
        run: docker run -d -p 3000:3000 devops-practice
```

2. Verifica que la aplicación se despliegue correctamente localmente usando Docker:

- Abre un navegador web y accede a http://localhost:3000 para verificar que la aplicación esté funcionando.

3. Automatización

- Automatiza la configuración y gestión del entorno local usando Docker Compose:

1. Crea un archivo docker-compose.yml:

```
version: '3.8'

services:
  app:
```

```
build: .
ports:
  - "3000:3000"
environment:
  - NODE_ENV=production
```

2. Corre la aplicación usando Docker Compose:

```
docker-compose up --build -d
```

4. Documentación y evaluación

- Documenta el proceso seguido, desde la configuración del entorno hasta la creación del pipeline CI/CD.

- Evalúa la experiencia: Reflexiona sobre los beneficios de tener un pipeline automatizado y cómo esto reduce la fricción entre los equipos de desarrollo y operaciones.