Ejercicio Práctico 1: Evaluación de Conceptos DevOps y su Aplicación en Proyectos

Mauricio Barrios B. - Fecha: 10-03-2025

El motivo del presente informe es describir fundamentos de DevOps, la integración continua, la automatización de pruebas y el uso de contenedores en el ciclo de desarrollo, además de aplicar estos conceptos al proyecto del Hospital.

Cuestionario

**1. Fundamentos de DevOps**

**¿Qué es DevOps y cuál es su propósito principal?**

DevOps es una cultura y conjunto de prácticas que busca unificar el desarrollo de software (Dev) y las operaciones de TI (Ops) para acelerar el ciclo de vida de entrega de software. Su propósito principal es mejorar la colaboración, automatizar procesos, aumentar la eficiencia y garantizar entregas más rápidas y confiables.

**El modelo CAMS y su importancia en la cultura DevOps.**

CAMS es un modelo que representa los pilares de DevOps:

* Cultura (Culture): Colaboración entre equipos.
* Automatización (Automation): Eliminar tareas manuales.
* Medición (Measurement): Métricas para mejorar procesos.
* Compartir (Sharing): Transparencia y retroalimentación.

Su importancia radica en que proporciona un marco para implementar DevOps de manera efectiva.

**2. Integración y Entrega Continua**

**¿Cuál es la diferencia entre Integración Continua y Entrega Continua?**

* Integración Continua (CI):Automatiza la integración de cambios de código en un repositorio compartido varias veces al día, incluyendo pruebas automáticas
* Entrega Continua (CD):Extiende la CI al automatizar la preparación del código para su despliegue en producción (pero no lo despliega automáticamente).

**¿Qué beneficios aporta la Integración Continua al proceso de desarrollo de software?**

Algunos beneficios son:

* Detecta errores temprano.
* Reduce conflictos en el código.
* Acelera el tiempo de entrega.
* Mejora la calidad del software.
* Facilita la colaboración entre equipos.

**3. Contenedores y Docker**

**¿Qué es un contenedor y en qué se diferencia de una máquina virtual?**

Como su nombre lo indica, el contenedor empaqueta una aplicación con sus dependencias y bibliotecas en una unidad aislada que se ejecuta sobre el kernel del sistema operativo anfitrión. Algunas diferencias son que las máquinas virtuales incluyen un SO completo y requieren hipervisor, los contenedores son más ligeros, rápidos y consumen menos recursos.

**¿Cuáles son los beneficios del uso de Docker en entornos DevOps?**

Algunos beneficios son:

* Portabilidad ("funciona en mi máquina" ya no es un problema).
* Aislamiento de aplicaciones.
* Escalabilidad rápida.
* Integración con herramientas de CI/CD (ej. Jenkins, GitLab CI).
* Uso eficiente de recursos comparado con VMs.

**4. Pruebas y Automatización en CI/CD**

**¿Cuáles son los tipos de pruebas más importantes en un pipeline de CI/CD?**

Las pruebas más importantes son:

* Pruebas unitarias: validan componentes individuales.
* Pruebas de integración: comprueban interacción entre módulos.
* Pruebas de regresión: verifican que cambios no rompan funcionalidad existente.
* Pruebas de rendimiento: evalúan escalabilidad y velocidad.
* Pruebas de seguridad: por ejemplo, SAST/DAST.

**5. Infraestructura y Monitoreo en DevOps**

**¿Qué es Infraestructura como Código (IaC) y qué ventajas ofrece?**

IaC es la gestión de infraestructura (servidores, redes, etc.) mediante archivos de configuración versionables (ej. Terraform, Ansible), algunas de las ventajas son las siguientes:

* Consistencia y reproducibilidad de entornos.
* Automatización de despliegues.
* Reducción de errores manuales.
* Escalabilidad bajo demanda.

**¿Por qué es importante el monitoreo en DevOps?**

Importancia del monitoreo DevOps:

* Detección proactiva de fallos.
* Optimización de rendimiento.
* Cumplimiento de SLAs.

Algunas herramientas para el monitoreo DevOps:

* Prometheus.
* Grafana.
* ELK Stack (Elasticsearch, Logstash, Kibana).
* Datadog.

**6. Orquestación y Kubernetes**

**¿Cuál es el propósito de un orquestador de contenedores como Kubernetes?**

La importancia es que automatiza el despliegue, escalado y gestión de contenedores en clusters, asegurando:

* Alta disponibilidad.
* Balanceo de carga.
* Autorrecuperación (self-healing).
* Escalabilidad automática.

Kubernetes facilita escalabilidad en producción:

* Ajusta automáticamente el número de réplicas de contenedores según la demanda (Horizontal Pod Autoscaler).

Kubernetes Facilita escalabilidad en gestión:

* Distribuye cargas entre nodos.
* Gestiona actualizaciones sin downtime (rolling updates).
* Monitoriza el estado de los pods y los reinicia si fallan.

Informe Aplicado a Proyecto Hospital

**1. Descripción**

A continuación se describe el proceso de integración continua y despliegue continuo para proyecto desarrollo sitio web de un hospital, desarrollado en React JS y MySQL, cuyos requerimientos abordados han sido creación de sistema de autenticación de usuarios, mantenedor de doctores, mantenedor de pacientes y mantenedor de citas medicas.

**2.** **Aplicación de Integración y Entrega Continua**

La tecnología escogida para el Pipeline de CI/CD es GitHub Actions (debido a la familiaridad con la plataforma), para integrarlo, los pasos son los siguientes:

1. Crear workflows en `.github/workflows`:

- `ci.yml` para integración continua

- `cd.yml` para despliegue continuo

2. Configuración básica del workflow de CI:

```yaml

name: CI Pipeline

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install dependencies

run: npm install

- name: Run tests

run: npm test

build:

needs: test

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install dependencies

run: npm install

- name: Build project

run: npm run build

3. Configuración básica del workflow de CD:

```yaml

name: CD Pipeline

on:

push:

branches: [ main ]

jobs:

deploy:

runs-on: ubuntu-latest

needs: ci\_pipeline

steps:

- uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install dependencies

run: npm install

- name: Build project

run: npm run build

- name: Deploy to production

uses: some-deployment-action

with:

target: production-server

env:

DB\_HOST: ${{ secrets.DB\_HOST }}

DB\_USER: ${{ secrets.DB\_USER }}

DB\_PASS: ${{ secrets.DB\_PASS }}

**3.** **Pasos para integrar pruebas automatizadas en el pipeline de Github Actions**

Para integrar pruebas automatizadas en el pipeline:

1. Tipos de pruebas a incluir:

- Pruebas unitarias (Jest)

- Pruebas de componentes (React Testing Library)

- Pruebas E2E (Cypress)

2. Configuración de pruebas:

```yaml

- name: Run unit tests

run: npm test -- --coverage

- name: Run component tests

run: npm run test:components

- name: Run E2E tests

run: npm run test:e2e

env:

TEST\_ENV: ci

DB\_TEST\_URL: ${{ secrets.TEST\_DB\_URL }}

3. Umbrales de cobertura (en package.json):

```json

"jest": {

"coverageThreshold": {

"global": {

"branches": 80,

"functions": 80,

"lines": 80,

"statements": 80

}

}

}

**4.** **Uso de Contenedores y Orquestación**

Implementación de Docker en el proyecto Hospital (React JS + MySQL):

Estructura de Dockerización:

1. Frontend (React) - Dockerfile:

```dockerfile

# Build stage

FROM node:16 as builder

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

# Production stage

FROM nginx:alpine

COPY --from=builder /app/build /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

2. Backend (Node.js/API) - Dockerfile:

```dockerfile

FROM node:16

WORKDIR /app

COPY package\*.json ./

RUN npm install --production

COPY . .

EXPOSE 3000

CMD ["node", "server.js"]

```

3. MySQL - Dockerfile:

```dockerfile

FROM mysql:8.0

ENV MYSQL\_ROOT\_PASSWORD hospital\_root

ENV MYSQL\_DATABASE hospital\_db

ENV MYSQL\_USER hospital\_user

ENV MYSQL\_PASSWORD hospital\_pass

COPY init.sql /docker-entrypoint-initdb.d/

EXPOSE 3306

4. \*\*docker-compose.yml\*\*:

```yaml

version: '3.8'

services:

frontend:

build: ./frontend

ports:

- "80:80"

depends\_on:

- backend

backend:

build: ./backend

ports:

- "3000:3000"

environment:

DB\_HOST: db

DB\_USER: hospital\_user

DB\_PASS: hospital\_pass

DB\_NAME: hospital\_db

depends\_on:

- db

db:

build: ./database

ports:

- "3306:3306"

volumes:

- mysql\_data:/var/lib/mysql

volumes:

mysql\_data: …

**5.Ventajas de Docker y GitHub Actions:**

Ventajas de Docker:

* Consistencia: Mismo entorno en desarrollo, testing y producción
* Aislamiento: Cada servicio (frontend, backend, DB) corre en contenedores separados
* Portabilidad: Fácil despliegue en cualquier sistema con Docker
* Escalabilidad: Fácil replicación de contenedores

Ventajas de GitHub Actions:

* Alta disponibilidad: Auto-reparación de contenedores fallidos
* Gestión de configuraciones: Claves secreatas para credenciales de DB
* Actualizaciones sin downtime: Rolling updates para el frontend

**6.Estrategias para monitoreo de logs y métricas:**

Estrategias de monitoreo

Centralización de logs:

* Envío de logs de frontend (Nginx), backend (Node.js) y MySQL a un sistema central
* Uso de Filebeat/Fluentd para recolección

Métricas clave:

* Tiempos de respuesta de API
* Uso de CPU/memoria por contenedor
* Errores HTTP (4xx, 5xx)
* Consultas lentas a MySQL
* Disponibilidad de servicios

Alertas:

* Configurar umbrales para métricas críticas
* Notificaciones a Slack/Email para incidentes

Dashboard:

* Visualización unificada de métricas y logs
* Grafana para métricas, Kibana para logs

**7. Uso de herramientas de monitoreo**

ELK Stack (Elasticsearch, Logstash, Kibana):

Implementación:

* Logstash recibe logs de todos los servicios
* Elasticsearch indexa y almacena los logs
* Kibana para visualización y análisis

Beneficios:

* Búsqueda rápida en logs históricos
* Detección de patrones de error
* Análisis de tráfico por hora/día

Herramientas de monitoreo como Prometheus + Grafana:

Implementación:

* Prometheus recolecta métricas de Node.js, Nginx y MySQL
* Grafana para dashboards visuales
* Alertmanager para notificaciones

Beneficios:

* Monitoreo en tiempo real del rendimiento
* Detección temprana de cuellos de botella
* Capacidad de correlacionar métricas