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

Matías Espinoza

Contexto:
En este ejercicio práctico, los estudiantes deberán demostrar su comprensión de los fundamentos de DevOps, la integración continua, la automatización de pruebas y el uso de contenedores en el ciclo de desarrollo. Deberán responder preguntas teóricas y generar un informe explicando cómo aplicar estos conceptos en un proyecto real.

Parte 1: Preguntas Teóricas (3 puntos)

*1. **Fundamentos de DevOps (0.5 pts)***

¿Qué es DevOps y cuál es su propósito principal?

- DevOps es una cultura, filosofía y conjunto de prácticas que buscan integrar los equipos de Desarrollo (Dev) y Operaciones (Ops) para automatizar y mejorar el proceso de desarrollo y entrega de software.
- Su propósito principal es acortar el ciclo de vida del desarrollo, aumentar la frecuencia de las implementaciones, y proporcionar una entrega más confiable y rápida de software de alta calidad. DevOps busca fomentar la colaboración, la comunicación y la comprensión mutua entre los equipos involucrados en la creación y el mantenimiento del software.

Explica el modelo CAMS y su importancia en la cultura DevOps.

- El modelo CAMS es un acrónimo que representa los cuatro pilares fundamentales de la cultura DevOps:
    - **Cultura (Culture)**: Fomenta la colaboración, la confianza y la comunicación abierta entre los equipos de desarrollo y operaciones. Se busca romper los silos tradicionales y crear un sentido de responsabilidad compartida.
    - **Automatización (Automation)**: Implica la automatización de tantos procesos como sea posible en el ciclo de vida del desarrollo y la entrega, incluyendo la integración, las pruebas, la implementación y la infraestructura. Esto reduce los errores humanos, acelera los procesos y mejora la eficiencia. Herramientas de CI/CD como Jenkins, GitHub Actions y Bitbucket Pipelines son ejemplos de automatización.
    - **Medición (Measurement)**: Se refiere a la recopilación y análisis de métricas clave para monitorear el rendimiento del sistema, identificar problemas y tomar decisiones basadas en datos. El monitoreo continuo ayuda a asegurar la estabilidad del proyecto.
    - **Compartir (Sharing)**: Promueve el intercambio de conocimientos, herramientas y mejores prácticas entre los equipos. Esto incluye la retroalimentación continua y el aprendizaje colectivo.
- El modelo CAMS es importante porque define los valores y principios clave que sustentan una cultura DevOps exitosa. Al enfocarse en la colaboración, la automatización, la medición y el compartir, las organizaciones pueden lograr los beneficios de DevOps, como una entrega de software más rápida y confiable.

2. Integración y Entrega Continua (0.5 pts)

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

- **Integración Continua (CI)** es una práctica de desarrollo donde los desarrolladores integran su código en un repositorio compartido varias veces al día. Cada integración se verifica mediante una construcción automatizada (build) y pruebas automatizadas para detectar errores rápidamente. El objetivo principal de la CI es asegurar que los cambios de código se integren de manera eficiente y sin errores.
- **Entrega Continua (CD)** es una extensión de la Integración Continua que automatiza el proceso de liberar el software a un entorno de pruebas o de pre-producción después de que ha pasado todas las pruebas automatizadas. Sin embargo, la decisión final de cuándo implementar en producción sigue siendo manual en la Entrega Continua. En contraste, el Despliegue Continuo va un paso más allá y automatiza la implementación en producción si el código pasa todas las pruebas.
- **La Integración Continua (CI)** asegura que los cambios de código se integren de manera eficiente y sin errores, la **Entrega Continua (CD)** asegura que el código esté siempre listo para ser desplegado, mientras que el **Despliegue Continuo** hace automáticamente la Entrega Continua.

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

La Integración Continua aporta varios beneficios al proceso de desarrollo de software:

- **Detección temprana de errores:** Los errores se identifican y corrigen en las primeras etapas del ciclo de desarrollo, lo que reduce los costos y el tiempo de depuración.
- **Mejora la calidad del código:** Las pruebas automatizadas aseguran que el código cumpla con los estándares de calidad.
- **Reduce los problemas de integración:** La integración frecuente del código minimiza los conflictos y facilita la colaboración entre los desarrolladores.
- **Entrega de software más rápida y frecuente:** La automatización del proceso de construcción y pruebas permite liberar software de manera más rápida y con mayor confianza.
- **Mayor visibilidad del estado del proyecto:** El equipo tiene una visión clara del estado del código y de los posibles problemas.

3. Contenedores y Docker (0.5 pts)

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

- Un **contenedor** es un entorno ligero y portátil que encapsula una aplicación junto con todas sus dependencias (bibliotecas, archivos de configuración, etc.), asegurando que la aplicación se ejecute de manera consistente, rápida y confiable en cualquier entorno.
    - Los contenedores comparten el kernel del sistema operativo anfitrión y solo virtualizan el nivel del sistema operativo necesario para ejecutar la aplicación.
- Una **máquina virtual (VM)**, por otro lado, virtualiza todo el hardware.
    - Cada VM incluye un sistema operativo completo, sus propias bibliotecas y aplicaciones, lo que la hace mucho más pesada y consume más recursos que un contenedor.

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

- El uso de Docker en entornos DevOps ofrece varios beneficios:
    - **Independencia del entorno:** Las aplicaciones empaquetadas en contenedores Docker funcionan de manera consistente independientemente del sistema operativo subyacente. Esto elimina el problema de "funciona en mi máquina".

    - **Portabilidad:** Los contenedores son fácilmente transportables entre diferentes máquinas y entornos (desarrollo, pruebas, producción).
    - **Aislamiento:** Cada contenedor está aislado de los demás, lo que evita conflictos de dependencias entre aplicaciones.
    - **Escalabilidad:** Los contenedores pueden ser fácilmente replicados y escalados horizontalmente para manejar cargas de trabajo variables.
    - **Eficiencia:** Los contenedores son ligeros y consumen menos recursos que las máquinas virtuales, lo que permite una mayor densidad de aplicaciones en un mismo hardware.
    - **Estandarización:** Docker proporciona una forma estándar de empaquetar y distribuir aplicaciones.
    - **Mayor productividad:** Los entornos de desarrollo se pueden configurar rápidamente utilizando imágenes Docker preconstruidas.

4. Pruebas y Automatización en CI/CD (0.5 pts)

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

- En un pipeline de CI/CD, algunos de los tipos de pruebas más importantes incluyen:
    - **Pruebas unitarias**, que verifican el comportamiento de unidades individuales de código.
    - **Pruebas de integración**, que comprueban que los módulos funcionan juntos correctamente.
    - **Pruebas de rendimiento**, que evalúan el comportamiento del sistema bajo diferentes cargas.
    - **Pruebas de aceptación**, que validan que el software cumple con los criterios del cliente.
    - **Pruebas de humo**, que son pruebas rápidas para verificar las funcionalidades principales después de una nueva compilación.
- Las pruebas en CI/CD son continuas en el ciclo de desarrollo y buscan retroalimentación rápida.
- El rol de las pruebas automatizadas en un pipeline de CI/CD es asegurar la calidad del código en cada etapa del proceso de desarrollo, detectando errores de manera temprana y automática. Esto permite mantener la estabilidad del software y facilita la entrega frecuente de nuevas funcionalidades con mayor confianza.

Explica en qué consiste el desarrollo guiado por pruebas (TDD) y su impacto en CI/CD.

- **El Desarrollo Guiado por Pruebas (TDD)** es una metodología donde se escriben pruebas antes de desarrollar el código. El ciclo de TDD incluye escribir una prueba fallida, implementar la funcionalidad mínima para pasar la prueba, y luego refactorizar el código.
- **El TDD impacta positivamente en CI/CD** al asegurar la calidad del código desde el inicio, facilitando la detección temprana de errores, y proporcionando una base sólida para la automatización de pruebas en el pipeline.

5. Infraestructura y Monitoreo en DevOps (0.5 pts)

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

- La infraestructura como código (IaC) es la capacidad de aprovisionar y respaldar su infraestructura de computación a través de código en lugar de procesos y configuraciones manuales. Cualquier entorno de aplicaciones requiere muchos componentes de infraestructura, como sistemas operativos, conexiones de bases de datos y almacenamiento. Los desarrolladores deben

configurar, actualizar y mantener periódicamente la infraestructura para desarrollar, probar e implementar aplicaciones.
- La administración manual de la infraestructura lleva mucho tiempo y es propensa a errores, especialmente cuando se administran aplicaciones a escala. En este contexto se aprecian las ventajas que ofrece:
    - La infraestructura como código le permite definir el estado deseado de su infraestructura sin incluir todos los pasos para llegar a ese estado.
    - Automatiza la administración de la infraestructura para que los desarrolladores puedan centrarse en crear y mejorar las aplicaciones en lugar de administrar los entornos.
    - Las organizaciones utilizan la infraestructura como código para controlar los costos, reducir los riesgos y responder con rapidez a las nuevas oportunidades de negocio.

¿Por qué es importante el monitoreo en DevOps? Menciona al menos dos herramientas de monitoreo utilizadas en entornos CI/CD.

- El **monitoreo** es importante en DevOps porque permite detectar fallos en el comportamiento del sistema, asegurar la calidad, proporcionar retroalimentación rápida, y garantizar la estabilidad del producto.
    - El monitoreo constante del software y su rendimiento es uno de los cuatro principios clave de DevOps: **Medición**.
- Algunas herramientas para el monitoreo en entornos modernos de CI/CD son **Prometheus** y **Grafana**.

6. Orquestación y Kubernetes (0.5 pts)

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

- El propósito de un orquestador de contenedores como Kubernetes es automatizar el despliegue, la gestión, el escalado y la organización de múltiples contenedores en un entorno de producción.
- Maneja tareas complejas como el despliegue de aplicaciones en contenedores, el escalado automático, la gestión de la disponibilidad, el balanceo de carga, la configuración y el rollback.

Explica cómo Kubernetes facilita la escalabilidad y gestión de aplicaciones en producción.

- Kubernetes facilita la escalabilidad y gestión de aplicaciones en producción de varias maneras:
    - **Escalado automático:** Kubernetes puede aumentar o disminuir automáticamente el número de instancias de un contenedor en función de la demanda, asegurando que la aplicación pueda manejar picos de tráfico sin intervención manual.
    - **Balanceo de carga:** Distribuye el tráfico de red entre las diferentes instancias de un contenedor, mejorando la disponibilidad y el rendimiento de la aplicación.
    - **Autoreparación:** Si un contenedor falla, Kubernetes puede detectarlo y reemplazarlo automáticamente, asegurando la continuidad del servicio.
    - **Despliegues y rollbacks:** Facilita la implementación de nuevas versiones de la aplicación y permite revertir a versiones anteriores en caso de problemas.
    - **Gestión de la configuración:** Permite gestionar la configuración de las aplicaciones de manera centralizada.
    - **Abstracción de la infraestructura:** Oculta la complejidad de la infraestructura subyacente, permitiendo a los desarrolladores centrarse en la aplicación.

Parte 2: Informe Aplicado a un Proyecto (4 puntos)

1. Introducción (0.5 pts)

El proyecto consiste en el desarrollo de una plataforma web para la gestión de la información y servicios del hospital. Esta plataforma permitirá a los pacientes acceder a información sobre médicos disponibles por especialidad, agendar citas, y posiblemente acceder a sus historiales médicos.

Para asegurar una entrega eficiente, confiable y segura de esta aplicación crítica, se aplicará una cultura DevOps que busca integrar los equipos de Desarrollo y Operaciones, automatizando el proceso de desarrollo y despliegue. Se fomentará la colaboración continua, la automatización, y la retroalimentación continua para garantizar la calidad y estabilidad del software.

2. Aplicación de Integración y Entrega Continua (1 pt)

Se implementará un pipeline de Integración Continua (CI) donde los desarrolladores integrarán su código en un repositorio compartido de manera frecuente. Cada integración se verificará automáticamente mediante la compilación del código, la ejecución de pruebas automatizadas, y la detección temprana de errores. Para esto, se podría utilizar una herramienta como Jenkins, GitLab CI o GitHub Actions.

Configuración del Pipeline (ejemplo con GitHub Actions):

- **Checkout del código**: Se clonará el repositorio del proyecto al realizar un push o una pull request a ramas específicas como main o develop.
- **Configuración del entorno**: Se establecerá el entorno de ejecución necesario, incluyendo la versión de Node.js, según los componentes de la aplicación (ej., React en el frontend, posible backend en Node.js).
- **Instalación de dependencias**: Se instalarán las dependencias del proyecto utilizando los gestores de paquetes correspondientes (ej., npm install).
- **Ejecución de pruebas automatizadas**: Se ejecutarán pruebas unitarias para verificar la lógica de los componentes individuales y pruebas de integración para asegurar la correcta interacción entre diferentes partes del sistema. Esto podría incluir pruebas de los componentes de React o la lógica de manejo de citas.
- **Construcción de la aplicación**: Se construirá la aplicación, optimizando los recursos para el despliegue.
- **Construcción y publicación de la imagen Docker**: Se construiría una imagen Docker de la aplicación y se publicaría en un registro de contenedores.
- **Despliegue en entorno de pruebas:** Se desplegaría la nueva versión de la aplicación en un entorno de pruebas.

Integración de Pruebas Automatizadas: Se integrarán diversos tipos de pruebas, como pruebas unitarias y pruebas de integración. El Desarrollo Guiado por Pruebas (TDD) podría ser implementado para escribir pruebas antes del código, asegurando la calidad desde el inicio. La automatización de estas pruebas en el pipeline de CI permitirá una retroalimentación rápida sobre la calidad del código con cada cambio, reduciendo el riesgo de errores en producción.

3. Uso de Contenedores y Orquestación (1 pt)

Cómo implementarían Docker en el proyecto.

Se utilizará Docker para empaquetar cada componente de la aplicación del hospital (ej., frontend con React, posible backend, base de datos) en contenedores ligeros y portátiles.

Se creará un Dockerfile para cada servicio, definiendo el entorno específico necesario para su ejecución.

Si la aplicación requiere múltiples servicios (ej., frontend y backend interactuando con una base de datos), se utilizará docker-compose.yml para definirlos y gestionarlos de forma conjunta.

Ventajas de Docker y Kubernetes:

- **Docker:** Proporcionará independencia del entorno, asegurando que la aplicación funcione de manera consistente en los diferentes entornos de desarrollo, pruebas y producción, evitando problemas como "funciona en mi máquina".
    - Facilitará la portabilidad de los componentes entre diferentes máquinas y entornos. El aislamiento de los contenedores garantizará que los diferentes servicios de la aplicación no interfieran entre sí.
- **Kubernetes:** Se podría utilizar para orquestar los contenedores Docker en el entorno de producción. Kubernetes automatizará el despliegue, la gestión, el escalado y la autoreparación de los contenedores, asegurando la alta disponibilidad de la plataforma del hospital.
    - Facilitará el escalado automático de los servicios según la demanda (ej., más instancias del frontend durante horas en que el sistema se estresa por consultas de acceso a la información de médicos) y la distribución del tráfico para mejorar el rendimiento.

4. Monitoreo y Seguridad en DevOps (1.5 pts)

Estrategias para Monitorear Logs y Métricas:

Se implementarán estrategias de monitoreo continuo para asegurar la salud y el rendimiento de la plataforma del hospital en producción. Esto permitirá detectar fallos antes de que afecten a los usuarios (pacientes, personal médico), medir el rendimiento de la aplicación (ej., tiempos de carga de la lista de médicos, tiempo de respuesta al agendar citas), y prevenir problemas con alertas tempranas.

Se recopilarán logs de la aplicación y del sistema para rastrear eventos, errores y comportamientos inusuales. También se monitorizarán métricas clave del sistema (uso de CPU, memoria, red) y de la aplicación (latencia de las peticiones, tasa de errores HTTP).

Uso de Herramientas de Monitoreo (ELK Stack o Prometheus):

Se podría utilizar el ELK Stack (Elasticsearch, Logstash, Kibana) para la centralización, análisis y visualización de logs. Esto facilitará la identificación y diagnóstico de problemas que puedan surgir en la plataforma del hospital.

Alternativamente, se podría implementar Prometheus para la recopilación de métricas en tiempo real y Grafana para la creación de dashboards personalizados que permitan visualizar el rendimiento del sistema de manera clara.

Se configurarían alertas basadas en umbrales definidos para notificar al equipo de operaciones sobre posibles incidencias.