Consignas Trabajo Práctico Integrador – DevOps

Objetivo: El objetivo de este trabajo práctico es que los alumnos apliquen los conceptos y herramientas aprendidas en el curso de DevOps para crear un proceso de integración continua, entrega continua y despliegue automatizado de una aplicación utilizando herramientas y prácticas de DevOps.

Entregables:

- **Link al repositorio (público)**
- **El código en un archivo zip**
- **Informe del trabajo práctico integrador.**
- **PPT**

Grupos: Formar grupos de hasta 3 personas, o de forma individual.

Fecha máxima para anotar grupos:	**Fecha de entrega final:**
17/04	

Puntaje:

Dockerfile			**Docker compose**	**CI**			**CD**	**Monitoreo (new relic, Datadog,**	
Dockerfile (requerido)	*multi stage image*	*buenas prácticas*	*Buenas prácticas*	**Checks (build y unit test)**	*Flujo de mergeo*	**Publicacion dockerfile**	*Desplieque a render o un servicio*	**Dashboard (Hits, etc)**	*APM / Trazas*
1	1	0,5	0,5	0,5	1	2	1	1,5	1

[link al spreadsheet]

Parte 1: Creación de la Aplicación

Objetivo: Desarrollar una aplicación web básica y preparar su entorno de contenedores.

1. **Desarrollo de la Aplicación**:
   - **Lenguaje**: Elige un lenguaje de programación moderno y popular, como Node.js, Python, Go, Ruby, etc.
   - **Funcionalidad**: Endpoint básico de health check, endpoints adicionales que realicen una operación simple (por ejemplo, una operación CRUD básica con datos en memoria).
2. **Pruebas Unitarias**:
   - **Requisitos**: Añade al menos una prueba unitaria para tu aplicación. Esta prueba debería verificar una funcionalidad básica y asegurar que el código cumple con los requisitos esperados.
   - **Framework de Pruebas**: Utiliza un framework de pruebas apropiado para el lenguaje seleccionado (por ejemplo, Jest para Node.js, PyTest para Python,

etc.).

3. **Contenerización con Docker**:
    - **Dockerfile**: Crea un archivo Dockerfile que defina cómo se debe construir la imagen de Docker para tu aplicación.
    - **Multi-stage Builds**: Implementa una técnica de multi-stage build en tu Dockerfile para optimizar el tamaño de la imagen final, separando el proceso de compilación y el entorno de ejecución.
    - **Buenas Prácticas**: Asegúrate de seguir las mejores prácticas de Docker, cómo usar imágenes base ligeras y minimizar la cantidad de capas.
4. **Orquestación con Docker Compose**:
    - **docker-compose.yml**: Configura un archivo docker-compose que automatice el despliegue de tu aplicación, permitiendo levantar todos los servicios necesarios con un solo comando.
    - **Integración con Dockerfile**: El archivo docker-compose debe utilizar el Dockerfile previamente creado para construir la imagen de la aplicación.

Parte 2: Configuración del Repositorio y CI

Objetivo: Establecer un sistema de integración continua (CI) para asegurar la calidad y la coherencia del código.

1. **Gestión del Código Fuente**:
    - **Repositorio Git**: Crea un repositorio en GitHub para alojar el código de tu aplicación. Asegúrate de seguir una estructura de repositorio clara y organizada.
2. **Configuración del CI**:
    - **Archivo de Configuración de CI**: Genera un archivo de configuración específico para el sistema de CI que elijas (como GitHub Actions, Travis CI, CircleCI, etc.).
    - **Automatización de Tareas**:
        - **Instalación de Dependencias**: El pipeline de CI debe incluir un paso para instalar todas las dependencias de la aplicación.
        - **Ejecución de Pruebas**: Configura el pipeline para ejecutar las pruebas unitarias de la aplicación.
        - **Compilación o Análisis Estático**: Si el lenguaje lo requiere, añade un paso de build. En caso contrario, realiza una verificación con un linter o un análisis estático del código.
        - **Protección de la Rama Principal**: Implementa políticas en GitHub para proteger la rama principal (main), requiriendo aprobaciones de revisión antes de los merges.
3. **Construcción y Publicación de la Imagen Docker**:

- **Build y Push**: El pipeline de CI debe construir la imagen Docker usando el Dockerfile y, si todas las verificaciones pasan, subir la imagen resultante a un registro de Docker, como Docker Hub.

Parte 3: Despliegue Continuo (CD) (Opcional)

Objetivo: Configurar un proceso de despliegue automatizado que se active al realizar merges en el repositorio.

1. **Automatización del Despliegue**:
   - **Pipeline de CD**: Configura un pipeline de CD que despliegue automáticamente la aplicación a un entorno de producción o staging cada vez que se realiza un merge a la rama principal.
   - **Validaciones Previas al Despliegue**: Asegura que el despliegue solo ocurra si todas las verificaciones de CI (tests, lint, build) pasan correctamente.

Parte 4: Monitoreo y Análisis

Objetivo: Implementar herramientas de monitoreo y análisis para supervisar el rendimiento y la salud de la aplicación.

1. **Monitoreo de Aplicación**:
   - **Herramientas Gratuitas**: Utiliza herramientas como New Relic, DataDog o Sentry para monitorear la aplicación. Estas herramientas permiten capturar métricas de rendimiento y errores en tiempo real.
   - **Configuración Básica**: Configura estas herramientas para monitorear aspectos clave como el uso de CPU, memoria, tiempos de respuesta, y errores.

Parte 5: Documentación y Presentación

Objetivo: Documentar todo el proceso y presentar los resultados y aprendizajes.

1. **Informe Detallado**:
   - **Documentación del Proceso**: Crea un informe que describa detalladamente cada paso seguido en la configuración de CI/CD, el despliegue y el monitoreo.
   - **Evidencia Visual**: Incluye capturas de pantalla, explicaciones detalladas y enlaces a los recursos utilizados.
2. **Presentación Final**:
   - **Preparación de Diapositivas**: Desarrolla una presentación que resuma el proyecto, destacando los logros, los desafíos superados y los aprendizajes clave.

- **Duración de la Presentación**: La presentación no debe exceder los 15 minutos y debe ser clara y concisa, permitiendo a la audiencia entender el flujo de trabajo y los resultados obtenidos.