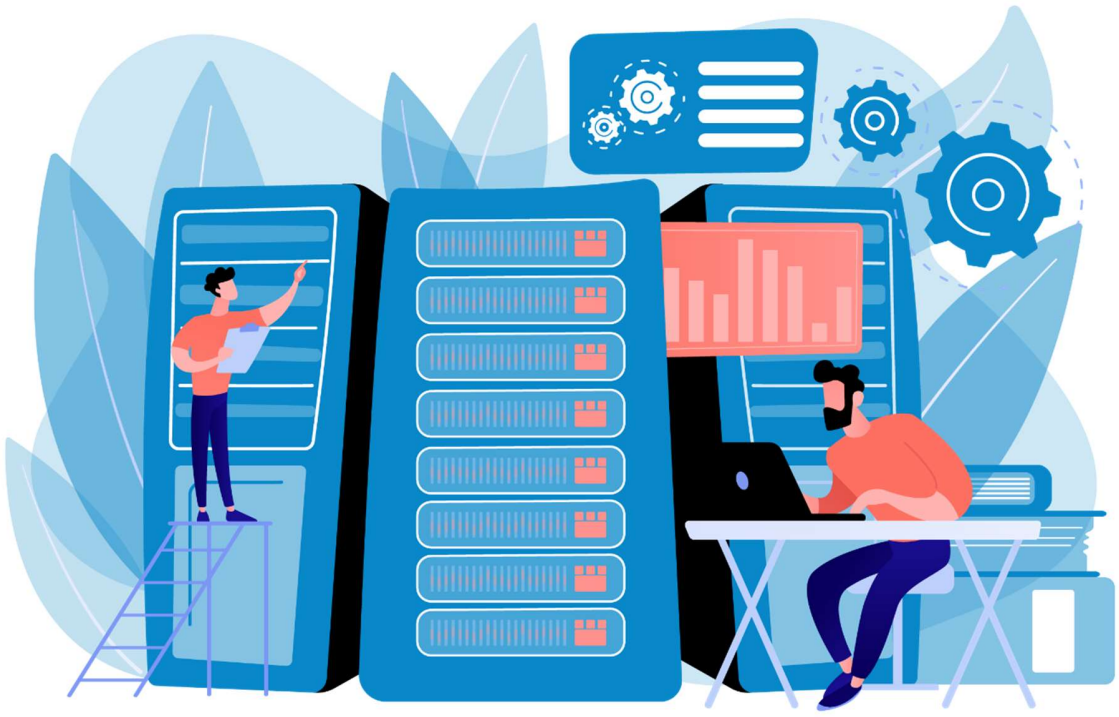
IES MONTE NARANCO

APUNTES DE TEORÍA

DEVOPS Y CLOUD COMPUTING



UT -4

INTEGRACIÓN Y ENTREGA CONTÍNUA (CI/CD)

Índice

1. Concepto y beneficios de CI/CD

1.1 ¿Qué es la Integración Continua (CI)?

La **Integración Continua (Continuous Integration)** es una práctica del desarrollo de software que consiste en integrar de manera frecuente el código generado por los desarrolladores en un repositorio común. Cada integración se valida automáticamente mediante compilaciones y pruebas.

Objetivos principales:

- Detectar errores de integración lo antes posible.
- Automatizar el proceso de construcción y pruebas.
- Mantener el repositorio siempre en un estado funcional.

1.2 ¿Qué es la Entrega y el Despliegue Continuos (CD)?

Entrega continua (Continuous Delivery): además de integrar el código automáticamente, se prepara el software para su despliegue en cualquier entorno con un solo clic.

Despliegue continuo (Continuous Deployment): va un paso más allá permitiendo que cada cambio aprobado pase directamente a producción sin intervención manual, siempre que supere todas las pruebas.

1.3 Beneficios del enfoque CI/CD

- Reducción de errores y fallos en producción.
- Entregas de software más rápidas y frecuentes.
- Mejora en la calidad del código.
- Aumento de la colaboración entre equipos.

- Trazabilidad completa del proceso de construcción y despliegue.
- Menor tiempo de recuperación ante fallos.

2. Herramientas para CI/CD y diseño de pipelines

2.1 Herramientas habituales de CI/CD

- Jenkins: plataforma muy configurable para automatización, con un amplio ecosistema de plugins.
- GitLab CI/CD: integrado en GitLab; permite diseñar pipelines usando archivos .gitlab-ci.yml.
- GitHub Actions: permite definir flujos de trabajo (workflows) directamente desde repositorios de GitHub.
- Azure DevOps Pipelines, Bitbucket Pipelines, CircleCI, Travis CI, entre otros.

2.2 Pipelines de CI/CD

Un **pipeline** es el conjunto de fases que recorre el código desde que se sube al repositorio hasta que acaba desplegado en un entorno.

Fases típicas de un pipeline:

1. **Build (compilación)**: generar artefactos ejecutables.
2. **Test**: realizar pruebas unitarias, de integración o funcionales.
3. **Package**: empaquetar la aplicación para su despliegue.
4. **Deploy**: desplegar en entornos como desarrollo, staging o producción.

2.3 Definición y configuración del pipeline

Los pipelines suelen definirse mediante archivos de configuración en formato YAML. Estos archivos indican:

- Qué tareas ejecutar.
- En qué orden.
- En qué condiciones.
- Sobre qué entorno o contenedor.

3. Pruebas automáticas en el pipeline

3.1 Importancia de las pruebas automatizadas

Las pruebas automáticas permiten validar que el software continúa funcionando correctamente tras cada cambio. Son clave para evitar regresiones y reducir fallos.

3.2 Tipos de pruebas automatizadas

- **Pruebas unitarias**: cada módulo o unidad del código se prueba de manera aislada.
- **Pruebas de integración**: comprueban que varios componentes funcionan correctamente en conjunto.
- **Pruebas funcionales o end-to-end**: simulan el flujo completo desde el punto de vista del usuario.
- **Pruebas de rendimiento**: verifican que la aplicación responde adecuadamente bajo carga.

3.3 Ejecución dentro del pipeline

En la mayoría de pipelines:

- Las pruebas unitarias se ejecutan en las primeras fases.
- Si alguna prueba falla, el pipeline se detiene automáticamente.
- Solo si todas las pruebas pasan se continúa con el empaquetado o despliegue.

Herramientas comunes:

- **JUnit, Mockito** (Java)
- **pytest** (Python)
- **Selenium** (pruebas end-to-end)
- **Postman/Newman** (APIs)

4. Estrategias de despliegue y rollback

4.1 Tipos de despliegue

- **Recreación (Recreate)**: apagar la versión antigua e iniciar la nueva. Rápido, pero con tiempo de inactividad.
- **Blue-Green Deployment**: se mantienen dos versiones activas (blue y green). El tráfico se redirige instantáneamente a la nueva versión.
- **Rolling Update**: se actualizan gradualmente las instancias, una a una.
- **Canary Release**: la nueva versión se lanza primero a un pequeño porcentaje de usuarios para comprobar su comportamiento antes del despliegue total.

4.2 Rollback

El **rollback** es el proceso de revertir un despliegue a una versión anterior cuando se detectan errores o comportamientos no deseados.

Estrategias de rollback:

- **Rollback automático en caso de fallo**: el sistema detecta errores y vuelve a la versión anterior sin intervención.
- **Rollback manual**: se ejecuta cuando el equipo detecta un problema tras el despliegue.
- En despliegues **blue-green o canary**, el rollback es especialmente rápido porque solo implica cambiar la ruta del tráfico.

4.3 Monitoreo post-despliegue

Una estrategia de despliegue eficaz incluye supervisión mediante:

- Logs
- Métricas
- Alarmas y alertas
- Paneles de observabilidad (Grafana, Prometheus, ELK)