CUDI
CENTRO UNIVERSITARIO DE INNOVACIÓN DE LA MATANZA

UTN
Tecnicatura Universitaria en Programación

Metodología de Sistemas II

Mejora continua en el desarrollo de software (CI/CD)

Prof. Rosalía Insaurralde

1. Mejora continua en el desarrollo de software

La mejora continua es un enfoque filosófico y técnico que busca mejorar procesos, herramientas, calidad del producto y colaboración entre personas de forma incremental, sistemática y permanente. Su origen se remonta a principios de gestión de calidad en la industria japonesa, y ha sido adoptado ampliamente en metodologías ágiles y DevOps.

En el contexto del desarrollo de software:

- Se analiza lo que funciona y lo que no después de cada entrega o iteración.
- Se introducen pequeñas mejoras técnicas, organizativas o de comunicación.
- El aprendizaje es parte del proceso, no algo externo o adicional.

Importancia de la mejora continua

El desarrollo de software es una actividad dinámica. Cambian los requisitos, tecnologías, equipos, herramientas. Un equipo que no mejora continuamente se estanca, acumula deuda técnica y pierde capacidad de innovación.

Ejemplos prácticos de mejora continua:

- Adoptar revisiones de código regulares para detectar errores antes de integrar.
- Agregar nuevas pruebas automatizadas para cubrir errores no detectados.
- Optimizar tiempos de ejecución del pipeline de integración.
- Recortar procesos burocráticos que generan demoras innecesarias.

2. Integración Continua CI (Continuous Integration)

La Integración Continua es una práctica de desarrollo en la que los desarrolladores integran su código frecuentemente (idealmente, varias veces al día) en un repositorio compartido, utilizando herramientas que automáticamente:

- Verifican que el código compile correctamente.
- Ejecutan pruebas automatizadas.
- Detectan errores de integración (conflictos, incompatibilidades, etc.).

Objetivos:

- Asegurar que el código compartido siempre esté en un estado funcional.
- Prevenir errores que se acumulan si los cambios se integran esporádicamente.
- Facilitar el feedback temprano y continuo sobre el estado del proyecto.

Características:

- Uso de sistemas de control de versiones (como Git) para centralizar el código.
- Automatización mediante scripts o herramientas CI (Jenkins, GitHub Actions, Travis CI, etc.).
- Cada *commit* o *pull request* desencadena un pipeline automático.
- Los errores se detectan inmediatamente después del cambio.

Ejemplo de flujo CI

- El desarrollador hace un commit y un push al repositorio remoto.
- Jenkins detecta el cambio y ejecuta:
  - Compilación del código.
  - Análisis estático (ej. revisión de estilo con ESLint o SonarQube).
  - Pruebas unitarias.
- Si todo es exitoso, el cambio puede ser revisado y aprobado.
- Si algo falla, el desarrollador recibe un aviso y lo corrige.

3. Entrega Continua
CD (Continuous Delivery)

La Entrega Continua extiende la CI automatizando el proceso de despliegue del software hacia entornos de prueba, pre-producción o incluso producción (en este caso, hablamos de *Continuous Deployment*).

Objetivos

- Tener el software siempre listo para ser entregado.
- Minimizar el tiempo entre desarrollo y disponibilidad del producto.
- Realizar despliegues rápidos, frecuentes, y confiables.

Diferencia entre *Delivery* y *Deployment*

- Continuous Delivery: el código está listo para producción, pero el despliegue se hace manualmente o con aprobación humana.
- Continuous Deployment: el código validado se despliega automáticamente sin intervención manual.

4. Herramientas de CI/CD

Jenkins

Jenkins

Jenkins es una herramienta de automatización de código abierto muy popular, especialmente utilizada para la Integración Continua (CI) y la Entrega Continua (CD) en el desarrollo de software. Permite automatizar tareas como la compilación, las pruebas y la implementación, lo que agiliza el proceso de desarrollo y reduce la posibilidad de errores.

Jenkins actúa como un servidor central que ejecuta tareas repetitivas de forma automática, basadas en eventos como la llegada de nuevo código al repositorio. Esto permite a los equipos de desarrollo detectar errores rápidamente, integrar código de manera más frecuente y desplegar software de forma más eficiente.

Ventajas:

- Extensible mediante plugins.
- Muy personalizable.
- Ideal para entornos empresariales complejos.

Desventajas:

- Requiere configuración y mantenimiento.
- La curva de aprendizaje puede ser elevada para proyectos pequeños.

4. Herramientas de CI/CD

GitHub Actions

GitHub Actions es una plataforma de automatización integrada en GitHub que permite a los desarrolladores automatizar flujos de trabajo de desarrollo de software, como la integración continua (CI) y la entrega continua (CD). Básicamente, permite crear flujos de trabajo (workflows) personalizados que se activan por eventos en tu repositorio de GitHub, como la creación de una solicitud de cambios o la fusión de ramas. Tiene *actions* predefinidas para construir, probar, desplegar, analizar código, etc.

Cómo funciona:

1.Eventos: Los flujos de trabajo se desencadenan por eventos en tu repositorio (por ejemplo, push, solicitud de extracción, creación de problema).

2.Flujos de trabajo: Son archivos YAML que definen los pasos a ejecutar en respuesta a un evento.

3.Trabajos: Un flujo de trabajo puede contener uno o más trabajos, que son conjuntos de pasos que se ejecutan en un entorno determinado (por ejemplo, un sistema operativo específico).

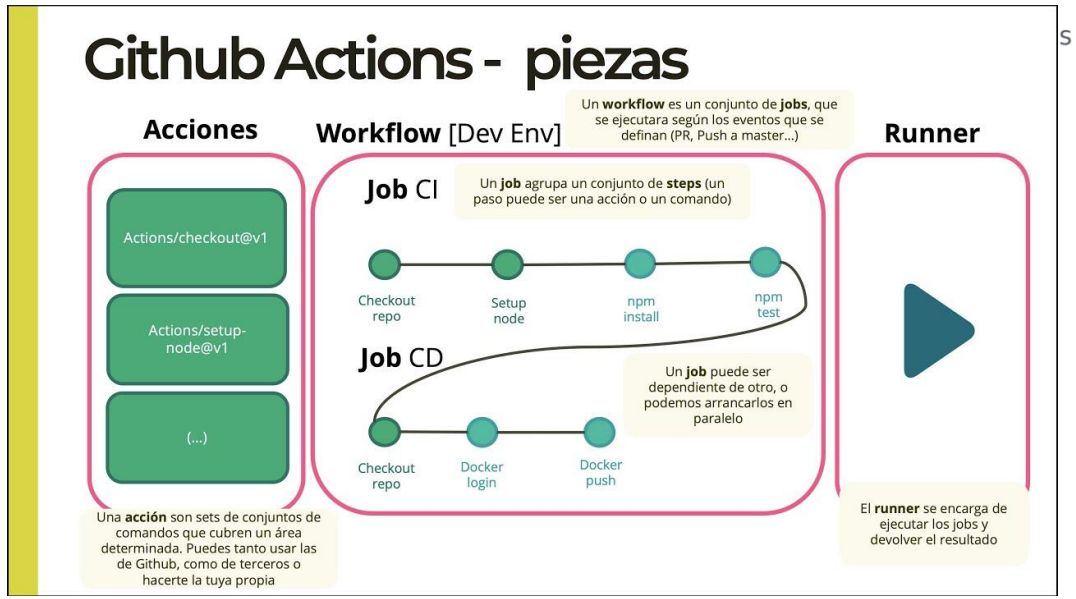
4.Pasos: Cada trabajo se compone de pasos, que pueden ser comandos o acciones.

5.Acciones: Son unidades reutilizables de código que realizan tareas específicas (por ejemplo, clonar el repositorio, ejecutar pruebas, desplegar la aplicación).

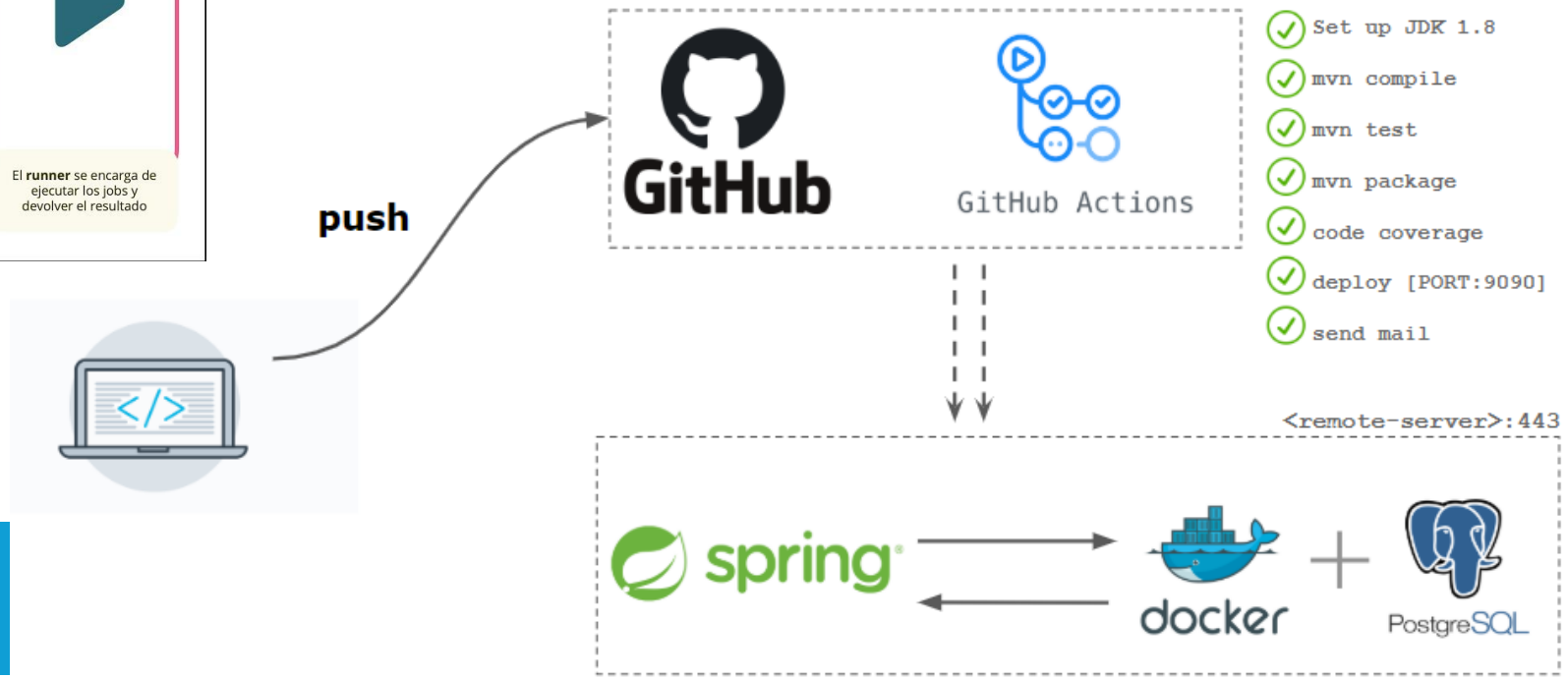
6.Runners: Son máquinas que ejecutan los pasos de los trabajos.

4. Herramientas de CI/CD

GitHub Actions



Ejemplo del potencial



4. Herramientas de CI/CD

GitHub Actions

GitHub Actions

Ejemplo de archivo YAML

```yaml
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm install -g bats
      - run: bats -v
```

https://docs.github.com/es/actions/tutorials/create-an-example-workflow

Ventajas de usar GitHub Actions:

- **Automatización:**
  Reduce la necesidad de realizar tareas manualmente, ahorrando tiempo y esfuerzo.
- **Integración:**
  Se integra directamente con GitHub, lo que facilita su uso y gestión.
- **Flexibilidad:**
  Permite personalizar los flujos de trabajo para adaptarse a las necesidades específicas de cada proyecto.
- **Comunidad:**
  Existe una gran comunidad y una amplia variedad de acciones pre-construidas disponibles para reutilizar.
- **Gratuito para código abierto:**
  Ofrece minutos de ejecución gratuitos para repositorios de código abierto y una cantidad generosa para repositorios privados.

4. Herramientas de CI/CD

ESLint

ESLint es una herramienta esencial para desarrolladores **JavaScript** que buscan escribir código más limpio, consistente y libre de errores.

Se utiliza para identificar problemas y patrones inconsistentes en el código, mejorando así la calidad y la mantenibilidad del mismo. ESLint ayuda a detectar errores, errores de estilo y otras anomalías antes de que el código se ejecute, lo que puede prevenir errores en tiempo de ejecución y mejorar la consistencia del código.

- **Análisis estático:**
ESLint analiza el código sin ejecutarlo, lo que permite detectar errores y problemas potenciales sin necesidad de ejecutar el programa.
- **Personalizable:**
ESLint es altamente configurable y permite a los usuarios definir reglas y estándares específicos para su proyecto, adaptándose a las necesidades del equipo y del proyecto.
- **Integración con IDEs:**
ESLint se puede integrar con editores de código como Visual Studio Code, lo que permite obtener retroalimentación en tiempo real mientras se escribe el código.
- **Comunidad activa:**
ESLint cuenta con una gran comunidad de usuarios y desarrolladores que contribuyen con reglas, complementos y soporte.
- **Mejora de la calidad del código:**
Al identificar patrones inconsistentes y errores potenciales, ESLint ayuda a mantener un código más legible, mantenible y libre de errores.
- **Previene errores:**
Al detectar problemas antes de que el código se ejecute, ESLint ayuda a prevenir errores en tiempo de ejecución y a reducir la cantidad de errores en el código.
- **Herramienta de linting estándar:**
ESLint se ha convertido en la herramienta de linting estándar para JavaScript, siendo utilizada por empresas como Microsoft, Airbnb y Netflix.

4. Herramientas de CI/CD

SonarQube

SonarQube es una plataforma de código abierto para el análisis estático de código fuente que evalúa la calidad y la seguridad del código, detectando errores, vulnerabilidades y problemas de estilo. Ofrece información en tiempo real sobre problemas de código directamente en el entorno de desarrollo.

- **Analiza la calidad del código:**
  Busca problemas como errores potenciales, vulnerabilidades de seguridad, código duplicado, falta de comentarios, complejidad excesiva, incumplimiento de estándares de codificación, y falta de pruebas unitarias.

Ejemplo:
https://ualmtorres.github.io/sonarqube-gitlab/

- **Proporciona retroalimentación:**
  Ofrece informes detallados sobre la calidad del código con sugerencias para mejorar.
- **Es una herramienta de código abierto:**
  Hay versiones disponibles para diferentes necesidades, incluyendo una edición comunitaria gratuita.
- **Admite múltiples lenguajes de programación:**
  A través de complementos, SonarQube puede analizar código en más de 20 lenguajes, incluyendo Java, JavaScript, Python, C#, PHP y C++.
- **Se integra con flujos de trabajo existentes:**
  Permite analizar el código en diferentes etapas del ciclo de desarrollo, incluso en la etapa de prueba y auditoría.
- **Apoya la inspección continua:**
  SonarQube puede integrarse con sistemas de integración continua como Jenkins, Azure DevOps, entre otros.
- **Ayuda a mejorar la seguridad del código:**
  Detecta vulnerabilidades de seguridad y ayuda a los desarrolladores a corregirlas.
- **Facilita la detección de "code smells":**
  Identifica patrones de código que podrían indicar problemas futuros de mantenibilidad.
- **Puede usarse para análisis de código generado por IA:**
  Permite asegurar la calidad del código generado por herramientas de inteligencia artificial.

5. Pipeline de CI/CD: etapas típicas

Un pipeline de CI/CD incluye múltiples etapas, que pueden variar según el proyecto.

Ejemplo típico:

Etapa	Objetivo
Checkout	Obtener el código fuente del repositorio
Build	Compilar el código (si es necesario)
Test	Ejecutar pruebas unitarias y de integración
Static Analysis	Verificar calidad del código (linter, cobertura)
Package	Generar paquetes listos para entrega
Deploy (opcional)	Enviar el software a un entorno (QA, staging, etc.)

6. Beneficios de la mejora continua

- **Detección temprana de errores**
  El código defectuoso no llega a etapas avanzadas del desarrollo.
- **Despliegue frecuente y predecible**
  La automatización reduce riesgos y permite entregas más rápidas.
- **Calidad y estabilidad del producto**
  Se refuerza la validación en cada cambio.
- **Mayor colaboración**
  Los equipos comparten la responsabilidad del código.
- **Reducción del costo de corrección**
  Es más barato corregir un error en desarrollo que en producción.

7. Riesgos si no se aplica mejora continua

- Se acumula *deuda técnica*.
- Se generan *entregas infrecuentes* y poco confiables.
- Mayor probabilidad de *fallos en producción*.
- *Falta de visibilidad* sobre la calidad del producto.
- *Desmotivación del equipo* por procesos rígidos o ineficientes.

**La mejora continua y la automatización mediante CI/CD no son solo prácticas técnicas:
son parte de una cultura de calidad, responsabilidad compartida y aprendizaje continuo. Implementarlas eficazmente transforma no solo el producto final, sino también la forma en que los equipos trabajan, colaboran y evolucionan.**

8. Cuadro comparativo de herramientas de CI/CD

Herramienta	Tipo / Licencia	Ventajas principales	Desventajas principales	Ideal para...
GitHub Actions	Integrado en GitHub, gratuito para proyectos públicos.	- Integración nativa con GitHub (repos, issues, PRs, secrets).- Facilidad de uso (no requiere servidores propios).- Marketplace con miles de acciones listas.- YAML simple, ejecución en runners Linux/Windows/macOS.- Buen soporte para proyectos open source y educativos.	- Limitado fuera del ecosistema GitHub.- Menos personalizable que Jenkins en entornos empresariales.- Ejecuciones gratuitas limitadas en repos privados.	Proyectos alojados en GitHub, cursos, startups o pequeños equipos que buscan agilidad.
Jenkins	Software libre (open source), autoalojado.	- Altamente configurable y extensible (más de 1800 plugins).- Compatible con cualquier VCS (Git, SVN, Mercurial).- Permite pipelines complejos y multiagente.- Control total del entorno y la seguridad.	- Requiere instalación y mantenimiento del servidor.- Interfaz menos intuitiva.- Actualizaciones de plugins pueden generar incompatibilidades.- Curva de aprendizaje alta.	Grandes organizaciones o entornos donde se necesita control total y personalización.
Travis CI	Servicio en la nube, gratuito para open source.	- Configuración simple (archivo .travis.yml).- Integración directa con GitHub.- Buen soporte para múltiples lenguajes y entornos.	- Plan gratuito limitado.- Menos soporte para repos privados.- Menos activo en desarrollo desde 2022.- Integración reducida fuera de GitHub.	Proyectos open source pequeños o medianos que buscan simplicidad.
CircleCI	SaaS (nube o self-hosted).	- Rápido y eficiente (paralelismo, caching inteligente).- Buen soporte para Docker y microservicios.- Configuración declarativa y flexible en YAML.- Integración con GitHub y Bitbucket.	- Interfaz más técnica.- Plan gratuito con límites de uso.- Aprendizaje inicial algo más complejo.- Algunas funciones avanzadas requieren pago.	Equipos DevOps o proyectos basados en contenedores que buscan rendimiento.
GitLab CI/CD	Integrado en GitLab (SaaS o self-hosted).	- Pipeline integrado con repositorios, issues y merge requests.- Excelente trazabilidad y control de versiones.- Permite runners propios o en la nube.- Compatible con Docker, Kubernetes, etc.	- Requiere usar GitLab o migrar repos.- Menos intuitivo para principiantes.- Algunos runners en la nube son pagos.	Organizaciones que usan GitLab como su plataforma principal de desarrollo.