Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Fases del ciclo de vida del software

Jesús Alberto Aréchiga Carrillo

22310439 6N

Profesor

José Francisco Pérez Reyes

Junio 2025

Guadalajara, Jalisco

Descripción general del pipeline de CI/CD

Durante décadas, los desarrolladores han intentado automatizar los elementos arduos y repetitivos de la codificación, para poder centrarse en la innovación y evitar el agotamiento. La CI/CD llegó como la solución perfecta, resolviendo problemas demasiados familiares asociados con la integración del nuevo código y las pruebas manuales.

La integración continua, la entrega continua y la implementación continua optimizan el proceso de combinar el trabajo de equipos separados en un producto cohesivo. La CI/CD proporciona un repositorio único para almacenar el trabajo y automatiza de manera consistente la integración y las pruebas continuas. Pero ¿qué es un pipeline de CI/CD y cómo funciona? Siga leyendo para descubrir las respuestas y obtener más información sobre las ventajas de los pipelines de CI/CD para ingenieros y empresas.

Pipelines de CI/CD definidos

Un pipeline de CI/CD es una serie de pasos que optimizan el proceso de entrega de software. A través de un enfoque de ingeniería de DevOps o fiabilidad del sitio, la CI/CD mejora el desarrollo de aplicaciones mediante la supervisión y la automatización. Esto es particularmente útil cuando se trata de integración y pruebas continuas, que generalmente son difíciles de realizar, requieren mucho tiempo y necesitan la creación de stubs y controladores.

Los pipelines automatizados pueden ayudar a prevenir los errores que resultan de los procesos manuales, permitir iteraciones rápidas del producto y proporcionar comentarios consistentes durante el proceso de desarrollo. Cada paso de un pipeline de CI/CD es un subconjunto de tareas agrupadas en las etapas del pipeline, que analizaremos en detalle más adelante en este artículo.

Integración continua

En el vertiginoso mundo tecnológico actual, los equipos de desarrollo deben ser capaces de trabajar simultáneamente en distintos elementos de una aplicación. Si los ingenieros tienen que esperar hasta el día de la fusión para integrar los cambios en la rama principal, el trabajo resultante es lento, laborioso y, francamente, aburrido. Debido a que todos hacen cambios de forma aislada, pueden ocurrir conflictos con otros miembros del equipo.

Cuando se practica la CI, se fusionan los cambios en un repositorio central de manera continua con la mayor frecuencia posible. Una compilación automatizada valida los cambios, con pruebas de unidad e integración que garantizan que la aplicación no colapse con cualquier cambio que se realice. Si las pruebas descubren un conflicto entre el código nuevo y el existente, la CI hace que la corrección de errores sea más rápida y frecuente.

Requisitos

- Pruebas automatizadas para mejoras, nuevas funcionalidades y correcciones de errores
- Fusionar los cambios con la mayor frecuencia posible, idealmente una vez al día
- Un servidor de integración continua para monitorear el repositorio y ejecutar pruebas en nuevas confirmaciones

Beneficios:

- Las pruebas automatizadas capturan las regresiones desde el principio, por lo que menos errores llegan a la producción
- Los problemas con la integración se resuelven más rápido, por lo que la creación de la versión es más fácil
- Los desarrolladores hacen menos cambio de contexto porque se les alerta de los errores tan pronto como colapsa la compilación
- Los servidores de CI ejecutan cientos de pruebas en segundos, lo que reduce los costos de las pruebas

Integración continua

La primera definición de CD, entrega continua, se basa en los principios de CI al automatizar el aprovisionamiento de la infraestructura y la implementación de la aplicación en entornos de prueba y producción.
En un pipeline de entrega continua, los cambios de código se crean, prueban y empaquetan automáticamente de manera que se puedan implementar en cualquier entorno y en cualquier momento. Se puede utilizar para activar implementaciones de manera manual, o se puede ampliar para incluir la implementación continua, donde las
implementaciones para clientes y usuarios finales también están automatizadas.

Requisitos

- Control de versiones para todos los archivos de código y configuración
- Un entorno de staging para probar nuevas versiones del software
- Un proceso de implementación automatizado y confiable

Beneficios:

- Entrega más rápida de nuevas funcionalidades y actualizaciones a los clientes
- Mejora de la fiabilidad y la calidad de las versiones de software
- Reversión más fácil de los cambios de código si es necesario
- Reducción del riesgo de error humano en el proceso de implementación
- Mayor colaboración entre los equipos de desarrollo y de operaciones

Implementación continua

La segunda definición de CD y la etapa final de un pipeline de CI/CD es la implementación continua. Los cambios de código se publican automáticamente para los usuarios finales después de completar con éxito las pruebas predefinidas. Tenga en cuenta que no hay una barrera manual antes de la producción, por lo que es esencial tener una automatización de pruebas hermética.

Para los desarrolladores, esto significa que los cambios en las aplicaciones en la nube podrían implementarse en cuestión de minutos, lo que facilita la recepción de comentarios de los usuarios finales y la actuación en consecuencia. Adoptar un enfoque de pipeline de CI/CD anula muchos de los riesgos asociados con la implementación. Esto se debe a que es más fácil lanzar cambios en lotes pequeños, en lugar de intentar implementar todos de una sola vez.

Requisitos:

- Un conjunto de pruebas de alta calidad
- Documentación que puede mantener el mismo ritmo que la producción
- Indicadores de funcionalidades (estos no son negociables para que pueda coordinar de manera efectiva con otros departamentos)

Beneficios:

- No es necesario pausar el desarrollo de nuevos lanzamientos, lo que agiliza todo el proceso
- Los lanzamientos son más fáciles de arreglar y menos riesgosos
- Las mejoras se hacen de manera continua, y estos aumentos en la calidad están claramente definidos para los clientes

Etapas del pipeline de CI/CD

Si bien un pipeline de CI/CD puede parecer trabajo adicional, es todo lo contrario. Simplemente se trata de un proceso que puede realizar para entregar nuevos productos de manera rápida y con menos problemas. Sin el pipeline automatizado, realizaría los mismos pasos de forma manual, lo que es más lento y menos

eficiente. A continuación, se presentan las etapas de un pipeline de CI/CD de DevOps. Una falla en cualquier etapa desencadena una notificación para alertar al ingeniero responsable. Si un producto aprueba todas las pruebas sin problemas, todos los miembros del equipo reciben una notificación después de cada implementación exitosa en producción.

Fuente

Por lo general, un pipeline es activado por un repositorio de código fuente. Los cambios de código activan una notificación en la herramienta de pipeline de CI/CD, que opera el pipeline correspondiente. Los flujos de trabajo iniciados por el usuario o programados automáticamente o los resultados de otros pipelines también pueden desencadenar un pipeline.

Compilación

Durante la fase de compilación, los ingenieros comparten el código que desarrollaron a través de un repositorio para crear una iteración ejecutable del producto. En términos generales, usaría Docker para implementar software nativo de la nube, y esta etapa del pipeline crea los contenedores de Docker necesarios. Si una aplicación no aprueba esta etapa, debe abordarla de inmediato porque sugiere que hay un error intrínseco con la configuración.

Prueba

Durante la prueba, se valida el código y tiene la oportunidad de observar cómo se comporta el producto. Es una red de seguridad esencial que evita que los errores lleguen a los usuarios finales. Como desarrollador, debe escribir las pruebas automatizadas, y cuanto más extenso sea su conjunto de pruebas, más rápido saldrá el producto al mercado y será menos probable que requiera una reimplementación.

Una falla en esta etapa deja al descubierto problemas que no se habían previsto al escribir el código. El objetivo de esta etapa es dar a los ingenieros una respuesta rápida, mientras la causa del problema está fresca en sus mentes, para que su estado de flujo no se desvíe de su rumbo.

Implementación

Una vez que se ha creado y probado una instancia ejecutable de todo el código, está lista para su implementación. Puede configurar su pipeline para implementar el código de acuerdo con un cronograma y elegir entre implementar productos a un grupo selecto de clientes o a todos ellos. Incluso puede automatizar el proceso de revertir a una versión anterior en caso de que se produzca un problema.

El resultado es que puede decidir qué funciona mejor y puede automatizarlo como parte de su pipeline de CI/CD.

Herramientas de pipeline de CI/CD

Existe una gran cantidad de herramientas de pipeline de CI/CD que se utilizan al implementar CI/CD en su proceso de DevOps, entre las que se incluyen:

- CI/CD de GitLab
- AutoDevOps

- ChatOps
- Docker
- Kubernetes
- OpenID Connect
- Bitbucket Cloud
- GitLab Runner
- Claves SSH
- Indicadores de funcionalidades
- UAT

¿Qué hace que un pipeline de CI/CD sea bueno?

No todos los pipelines de CI y CD se crean de la misma manera. Su objetivo es generar productos precisos y confiables rápidamente, con información exhaustiva a lo largo de todo el ciclo de desarrollo, por lo que la precisión, la confiabilidad y la velocidad son la base de un proceso eficaz. Veamos por qué:

- **Velocidad**: la integración continua debe ser rápida con retroalimentación instantánea, o el flujo se interrumpe y la productividad se reduce.
- **Precisión**: la automatización del proceso de implementación requiere una precisión milimétrica para evitar la interferencia humana.
- **Confiabilidad**: el pipeline debe ser confiable, con código de prueba hermético y salida estable.

Ejemplo de pipeline de CI/CD

A continuación, se muestra un breve ejemplo de un diagrama de pipeline de CI/CD:

- **Control de código fuente:** aloje el código en GitLab para integrar su aplicación con los principales software y servicios.
- **CI/CD:** use la CI/CD de GitLab para realizar la confirmación de todo el código, compilar y ejecutar las pruebas requeridas.
- **Implementar código en UAT:** configure la CI/CD de GitLab para implementar código en el servidor de UAT.
- **Implementar en producción:** repita el paso de CI/CD para implementar el código en UAT.

¿Por qué los líderes de TI deben usar los pipelines de CI/CD?

El uso de pipelines de CI/CD tiene una serie de ventajas claras, que resumimos a continuación:

- Simplifica la compilación y las pruebas

- Mejora la calidad y la consistencia del código
- Optimiza la comunicación
- Automatiza gran parte del proceso de entrega de software
- Inspira comentarios más rápidos
- Aumenta la visibilidad del producto
- Le permite corregir rápidamente los errores manuales
- Reduce los costos de mano de obra
- Acelera el ciclo de vida del desarrollo
- Facilita un ciclo de comentarios rápido entre los ingenieros y los clientes
- Las pruebas automatizadas ahorran dinero y minimizan los problemas para los usuarios finales

Cómo los pipelines de CI/CD respaldan a los equipos de DevOps

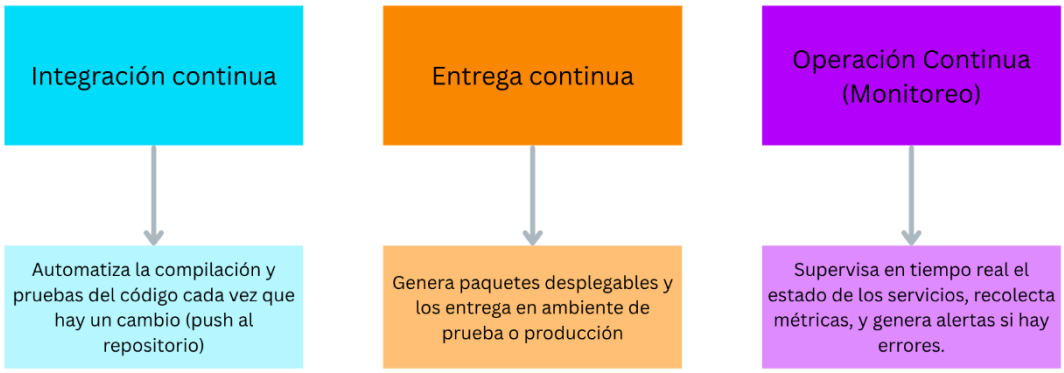
Un pipeline de integración continua mejora la calidad del código al garantizar que todos los cambios de código pasen por el mismo proceso. Los cambios de código se validan frente a otros cambios que se realizan en el mismo repositorio de código compartido. Las pruebas y compilaciones automatizadas disminuyen la posibilidad de errores humanos, lo que crea iteraciones más rápidas y un código de mejor calidad.

Concepto de Pipeline de CI/CD

Un **Pipeline de CI/CD (Integración y Entrega/Despliegue Continuas)** es una secuencia automatizada de pasos que permite compilar, probar, construir, desplegar y monitorear una aplicación de forma continua y eficiente. Este pipeline automatiza todo el **ciclo de vida del desarrollo de software**: desde que se escribe una línea de código hasta que se ejecuta en producción con monitoreo activo.

Fases del pipeline CI/CD:

1. **Integración Continua (CI)** Automatiza la compilación y pruebas del código cada vez que hay un cambio (push al repositorio).
2. **Entrega Continua (CD)** Genera paquetes desplegables (por ejemplo, imágenes Docker) y los entrega en ambientes de prueba o producción.
3. **Operación Continua (Monitoreo)** Supervisa en tiempo real el estado de los servicios, recolecta métricas, y genera alertas si hay errores.



Construcción del Pipeline CI/CD Completo

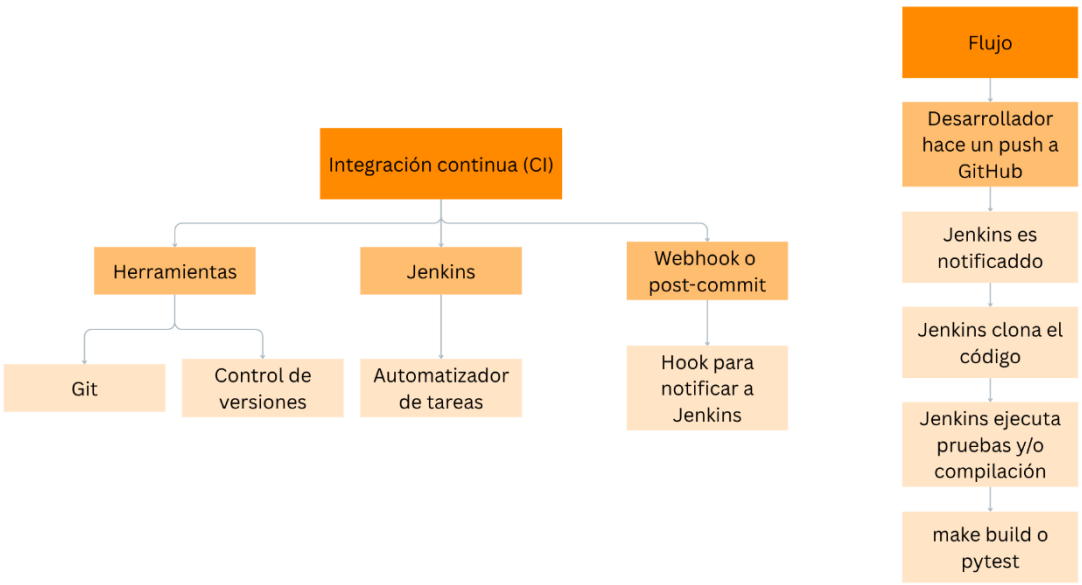
Integración Continua (CI)

Automatizar la compilación y pruebas cada vez que hay un cambio en el código.
Herramientas:

- **Git**: Control de versiones.
- **Jenkins**: Automatizador de tareas.
- **Webhook** o post-commit hook para notificar a Jenkins.

Flujo:

1. Desarrollador hace un push a GitHub.
2. Jenkins es notificado.
3. Jenkins clona el código.
4. Jenkins ejecuta pruebas y/o compilación (make build o pytest).



Jenkinsfile ejemplo (CI):

```
pipeline {
	agent any
	stages {
		stage('Checkout') {
			steps {
				git url: 'https://github.com/usuario/repo.git'
			}
		}
		stage('Build') {
			steps {
				sh 'make build'
			}
		}
		stage('Test') {
			steps {
				sh 'pytest tests/'
			}
		}
	}
}
```

Entrega Continua (CD)

Construir una imagen Docker y desplegarla automáticamente.
Herramientas:

- **Docker**: Para contenerizar la app.
- **Jenkins Docker Plugin**: Para manejar Docker desde Jenkins.

Archivos necesarios:

- Dockerfile
- app.py con Flask
- requirements.txt

Jenkinsfile extendido (CI + CD):

```
pipeline {
	agent any
	environment {
		IMAGE_NAME = 'miapp:latest'
	}
	stages {
		stage('Checkout') {
			steps { git url: 'https://github.com/usuario/repo.git' }
```

```
            }
            stage('Build') {
                steps { sh 'make build' }
            }
            stage('Test') {
                steps { sh 'pytest tests/' }
            }
                stage('Docker Build') {
                    steps {
                        sh 'docker build -t $IMAGE_NAME .'
                    }
                }
                stage('Deploy') {
                    steps {
                        sh 'docker stop miapp || true'
                        sh 'docker rm miapp || true'
                        sh 'docker run -d -p 5000:5000 --name miapp
                    $IMAGE_NAME'
                    }
                }
            }
        }
}
```

Operación Continua (Monitoreo)

Herramientas:

- **Prometheus**: Recolecta métricas del servicio.
- **Alertmanager**: Notifica fallas o umbrales superados.
- **Node Exporter** y **instrumentación Flask**.

Instrumentación básica en Flask

```python
from flask import Flask
from prometheus_client import Counter, generate_latest

app = Flask(__name__)
visit_counter = Counter('http_requests_total', 'Número total de peticiones')

@app.route("/")
def home():
    visit_counter.inc()
    return "App funcionando con monitoreo"

@app.route("/metrics")
def metrics():
    return generate_latest()

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Prometheus.yml

```yaml
scrape_configs:
  - job_name: 'miapp'
    static_configs:
      - targets: ['localhost:5000']
```

Resumen del Pipeline Unificado

```
[Push a GitHub]
       ↓
[Jenkins]
  - Checkout
  - Build
  - Test
  - Docker Build
  - Docker Deploy
       ↓
[App corriendo en contenedor Docker]
       ↓
[Prometheus monitorea /metrics]
       ↓
[Alertmanager notifica si hay errores]
```

Conclusión

El establecimiento de un pipeline de CI/CD completo, especialmente la incorporación de un pipeline de implementación continua, mejora significativamente la eficiencia y la confiabilidad de los proyectos de desarrollo de software. Al automatizar la ejecución de jobs desde cada confirmación de una rama hasta la implementación, este tipo de pipeline garantiza que cada cambio se pruebe e integre de manera exhaustiva. Esta automatización aprovecha las potentes funcionalidades de las herramientas de CI/CD para optimizar los procesos en todo el repositorio de código.

Esto no solo acelera el ciclo de desarrollo, sino que también ayuda a mantener altos estándares de calidad del código, lo que garantiza que cada confirmación contribuya de manera positiva a los resultados del proyecto.

Implementar un **Pipeline CI/CD completo** asegura:

1. Mayor **calidad y velocidad** en entregas.
2. Reducción de errores humanos.
3. Monitoreo proactivo del sistema.
4. Preparación para escalar a microservicios y Kubernetes.