

# Unidad 3: Entrega Continua y Despliegue Continuo (CD)

---

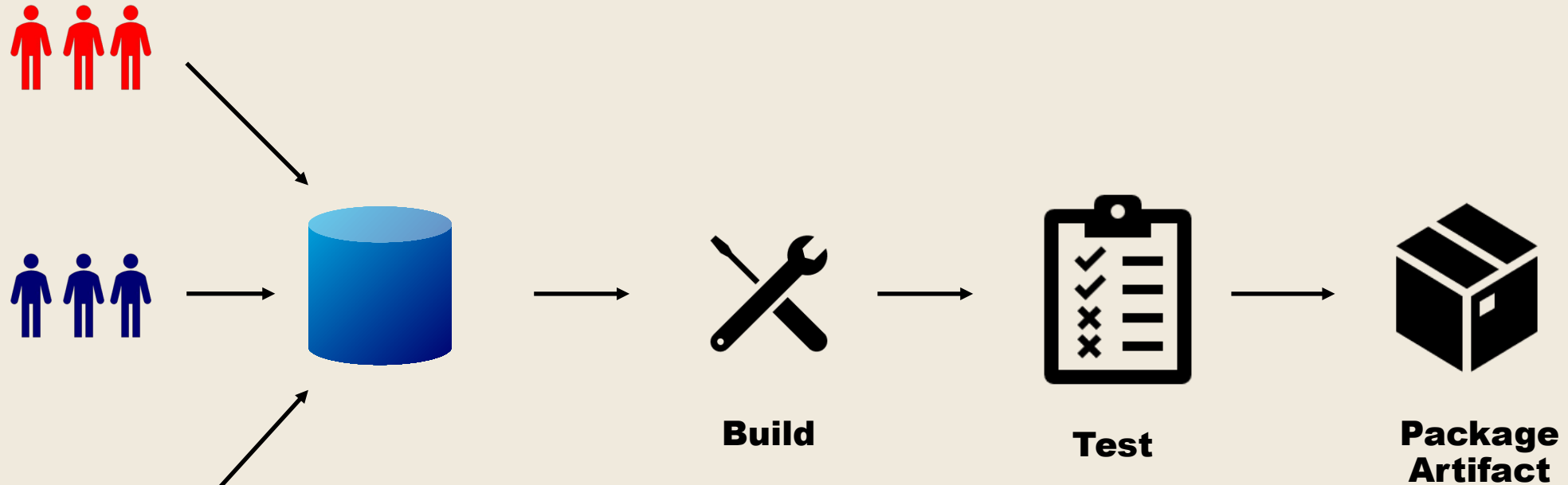
Llevando el Software Validado a los Usuarios

# Introducción a CD

---

# Punto de Partida

## Del CI a la Entrega



- Código integrado frecuentemente.
- Builds automatizadas y consistentes.
- Pruebas Unitarias y de Integración pasadas.
- Análisis Estático de Calidad y Seguridad OK.

### Resultado:

Un **artefacto confiable y versionado** listo para el siguiente paso.

# Punto de Partida

## Del CI a la Entrega

¿Y ahora qué? ¿Cómo hacemos llegar este artefacto a nuestros usuarios de forma rápida y segura?



**Package  
Artifact**

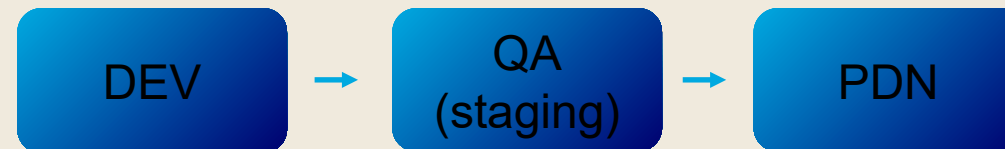
**¡Continuous Delivery / Continuous Deployment!**

# Conceptos clave de CD

---

# Separación de ambientes

La separación de ambientes en el desarrollo de software es **crear entornos aislados para cada etapa (desarrollo, pruebas, producción)** para **minimizar riesgos y asegurar la calidad del software antes de su lanzamiento**. Existe también un entorno específico llamado Staging que es una réplica casi exacta de producción para pruebas finales.



# Infraestructura como Código (IaC)

## Cimientos Automatizados para CD

### Necesidad:

Para que nuestros despliegues sean fiables y repetibles, **necesitamos que los entornos (Staging, Producción) sean consistentes. Aquí entra IaC.**

### Concepto clave:

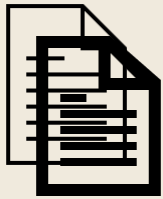
**Definir y gestionar toda la infraestructura** (servidores, redes, bases de datos, balanceadores, clusters K8s, funciones serverless) **usando código** (archivos de texto descriptivos) **y herramientas de automatización**, en lugar de configuración manual.

### Herramientas populares:

Terraform, AWS CloudFormation, Azure Resource Manager (ARM) / Bicep, Google Cloud Deployment Manager, Pulumi.

### Rol en CD:

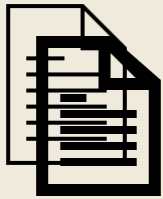
- **Consistencia de Entornos** (Staging – Producción).
- **Creación de Entornos Bajo Demanda** (entornos temporales para pruebas).
- **Versionado y Auditoría** (Se sabe que cambió, quien lo cambió y cuando).
- **Integración en Pipeline** (el pipeline de CD ejecuta terraform apply o create stack con aws por ejemplo).
- **Infraestructura Inmutable** (facilita crear infraestructura para cada reléase en lugar de modificar la existente).



# GitOps

## Cimientos Automatizados para CD

### Concepto clave:



**GitOps es un marco de trabajo operacional que utiliza Git como la única fuente de verdad para la infraestructura y las aplicaciones.** Automatiza la entrega continua y el despliegue de software, haciendo que los cambios sean predecibles y auditables. Se basa en los principios de IaC al utilizar código para definir el estado deseado.

Sin embargo, **GitOps va más allá: define cómo se aplican y mantienen esos cambios en producción** de forma continua y automatizada (por ejemplo, definiendo como se aplican las estrategias de despliegue que ya veremos).



# La Nube como habilitador de CD

## Motivante:

Implementar todos estos conceptos puede ser complejo on-premise. **Las nubes públicas ofrecen servicios que facilitan enormemente la construcción de pipelines de CI/CD.**

## Ventajas generales de la Nube para CI/CD:

- **Elasticidad:** Escalar recursos (agentes de build, entornos de prueba) bajo demanda.
- **Servicios Gestionados:** Evitan tener que instalar y mantener herramientas base (servidores CI, registros, orquestadores).
- **Integración:** Servicios diseñados para funcionar juntos dentro del ecosistema del proveedor. Se integran y proveen soluciones compatibles con herramientas DevOps.
- **Infraestructura como Código (IaC):** APIs y herramientas nativas para gestionar la infraestructura mediante código.



Google Cloud Platform




# Elementos de CD

---

# Automatización específica para CD

## Tareas clave a automatizar:

- **Orquestación de Despliegues:** Gestionar el flujo del artefacto a los entornos de Dev → Staging → Producción.
  - **Provisionamiento de Entornos (IaC):** Crear/actualizar infraestructura usando IaC (Terraform, CloudFormation, ARM, Pulumi).
  - **Gestión de Configuración (segura):** Aplicar configuraciones específicas por entorno (BDs, API Keys, Feature Flags) de forma segura (integrado con Vaults).
  - **Ejecución de Estrategias de Despliegue:** Implementar Blue/Green, Canary de forma automática.
  - **Ejecución de Pruebas Avanzadas:** Integrar y ejecutar tests E2E, Performance, DAST.
  - **Implementación de Rollbacks:** Disparar y ejecutar la reversión a una versión anterior estable.
- 

# Pruebas en CD

---

# Pruebas Clave en las etapas de CD

## Propósito:

**Ganar confianza** en que el artefacto **funcionará** correctamente en producción y **bajo condiciones reales**.

**La mayoría se realizan en el entorno de Staging** (un entorno con características similares o idénticas a producción en términos de infraestructura, integraciones, etc.)

**Otras se realizan después de la puesta en producción** para validar el correcto funcionamiento tras el despliegue.



# Pruebas Clave en las etapas de CD

## Tipos de pruebas esenciales para CD (staging):

- **Pruebas E2E / Aceptación:** (Ej. Cypress, Selenium) Simulan flujos de usuario completos (login -> compra -> pago). Verifican requisitos funcionales en el sistema integrado.
- **Pruebas de Contrato:** (Ej. Pact) Asegurar que las APIs cumplen los contratos esperados entre servicios (Microservicios). Aseguran que las APIs entre servicios mantienen la compatibilidad.
- **Pruebas de Rendimiento/Carga:** (Ej. k6, JMeter) Verificar comportamiento bajo carga esperada/pico. ¿Es rápida? ¿Escala bien? ¿Consume recursos eficientemente? **¡Fundamental!**
- **Pruebas de Seguridad Dinámicas (DAST):** (Ej. OWASP ZAP) Buscar vulnerabilidades en la aplicación en ejecución. Simulan ataques desde el punto de acceso del usuario.

## Validación post-producción:

- **Smoke Tests:** (Scripts simples, Postman/Newman) Chequeos básicos en Producción para verificar disponibilidad y funcionalidad crítica inmediatamente tras desplegar. (¿La app responde?).

**¡Su fallo indica un problema grave!**



# La Seguridad en CD

## Integrando la Seguridad en CD

### Necesidad:

La seguridad no es algo que se añade al final. **Debe estar integrada en todo el ciclo, incluyendo las fases de CD, complementando los chequeos de CI.**

### Puntos de control en CD:

- **(Post-CI) Escaneo de Artefactos/Imágenes:** Buscar vulnerabilidades conocidas en las dependencias y el sistema operativo base de las imágenes Docker ANTES de desplegarlas. (Herramientas: Trivy, Clair, Snyk Container, JFrog Xray).
- **Análisis de seguridad en IaC:** Escanear el código Terraform/CloudFormation/etc., en busca de malas configuraciones de seguridad (puertos abiertos, buckets públicos, permisos excesivos) ANTES de aplicar los cambios. (Herramientas: Checkov, tfsec, KICS, Terrascan)
- **Pruebas de seguridad dinámicas (DAST):** Ejecutar escaneos automatizados contra la aplicación ya desplegada en un entorno de Staging para encontrar vulnerabilidades en tiempo de ejecución (XSS, SQL Injection). (Herramientas: OWASP ZAP, Burp Suite integrado).
- **Gestión Segura de Secretos para Despliegue:** Asegurar que el pipeline obtiene los secretos necesarios para Producción (API keys, contraseñas BD) de forma segura desde un gestor centralizado (Vault, AWS Secrets Manager, Azure Key Vault, GCP Secret Manager) justo en el momento del despliegue, sin exponerlos antes.



# **Estrategias de Despliegue y Rollbacks**

---



# Herramientas de Orquestación de Despliegue (CD)

## Función Principal

Orquestar el proceso de release/deploy completo post-CI.

## Capacidades típicas:

- **Modelar pipelines de despliegue** (a menudo visualmente).
- **Gestionar múltiples entornos** (Dev, Staging, Prod).
- **Integrarse** con herramientas de CI, registros de artefactos, cloud providers, IaC.
- **Soportar diferentes estrategias de despliegue** (siguiente sección).
- **Gestionar aprobaciones manuales** (para Continuous Delivery).
- **Proporcionar visibilidad y dashboards** del estado de los despliegues.

## Ejemplos:

- **Plataformas Dedicadas:** Spinnaker, Argo CD (enfocado en K8s/GitOps), Harness, Octopus Deploy.
- **Funcionalidades en Suites DevOps:** GitLab CI/CD (Pipelines & Environments), GitHub Actions (Workflows & Environments), Azure Pipelines (Release Pipelines), AWS CodePipeline & CodeDeploy.



# Estrategias de Despliegue Avanzadas

## ¿Por qué?

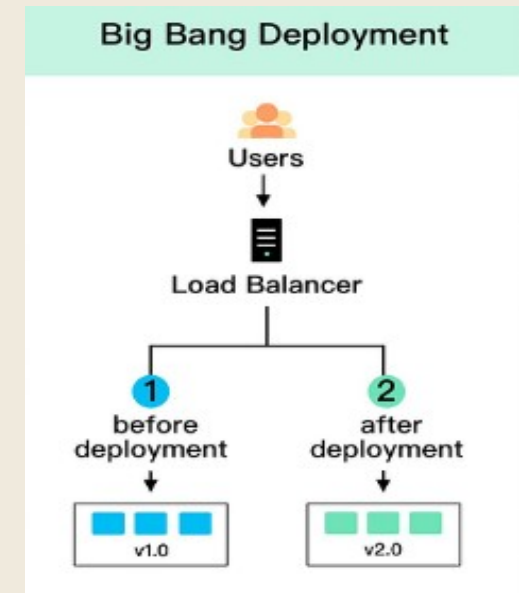
### El Riesgo del "Big Bang":

- Reemplazar la versión antigua por la nueva de golpe es simple pero arriesgado.
- Si la nueva versión tiene un bug crítico... ¡afecta a todos los usuarios instantáneamente!
- **Blast Radius (Radio de Impacto):** Es el alcance del daño si un despliegue sale mal. Queremos minimizarlo.

### Objetivo de las Estrategias:

Introducir **cambios en producción de forma controlada** para:

- **Limitar** el número de **usuarios expuestos** a posibles errores.
- **Permitir** la **detección temprana de problemas** antes de que afecten a todos.
- **Facilitar una reversión** (rollback) rápida y segura.



# Estrategia Base: Rolling Update

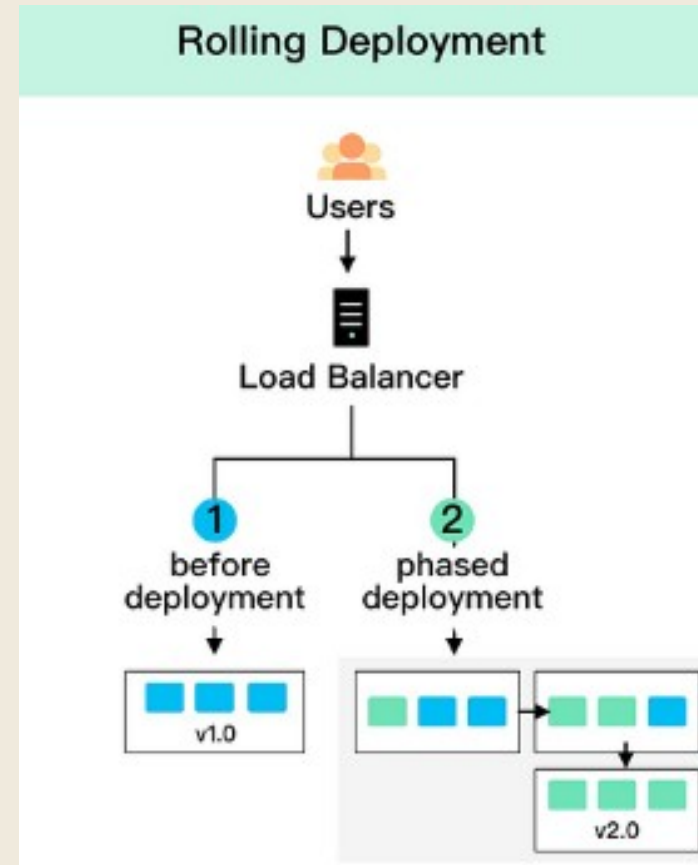
## (Actualización Continua)

### Concepto:

**Reemplazar gradualmente** las instancias (servidores, contenedores/pods) de **la versión antigua (v1)** por instancias de **la versión nueva (v2)**.

### ¿Cómo funciona?

1. Se inicia una instancia v2.
2. Cuando v2 está lista (health check OK), se elimina una instancia v1.
3. Se repite hasta que todas las instancias sean v2.



# Estrategia Base: Rolling Update

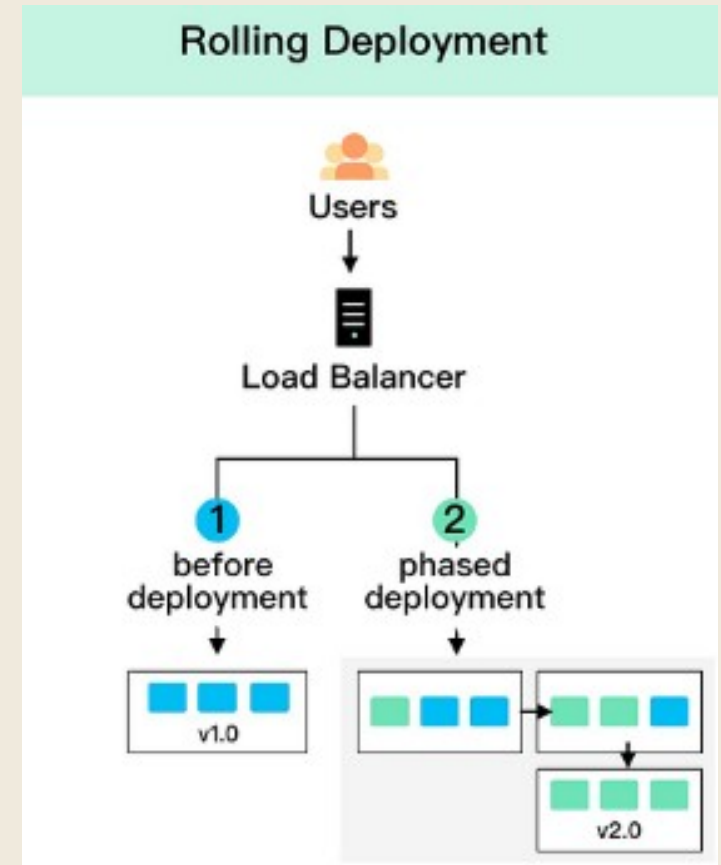
## (Actualización Continua)

### Ventajas:

- Generalmente **simple de configurar** (a menudo es la opción por defecto en orquestadores como Kubernetes).
- **No requiere** (normalmente) **infraestructura adicional** temporal.
- El **despliegue** de manera **gradual**.

### Desventajas:

- Durante el proceso, **coexisten instancias v1 y v2** sirviendo tráfico (puede causar problemas si no son compatibles).
- **El rollback puede ser lento** pues implica otro rolling update hacia atrás.
- Un **problema** podría **impactar** a un **número creciente de usuarios** antes de ser detectado y revertido.



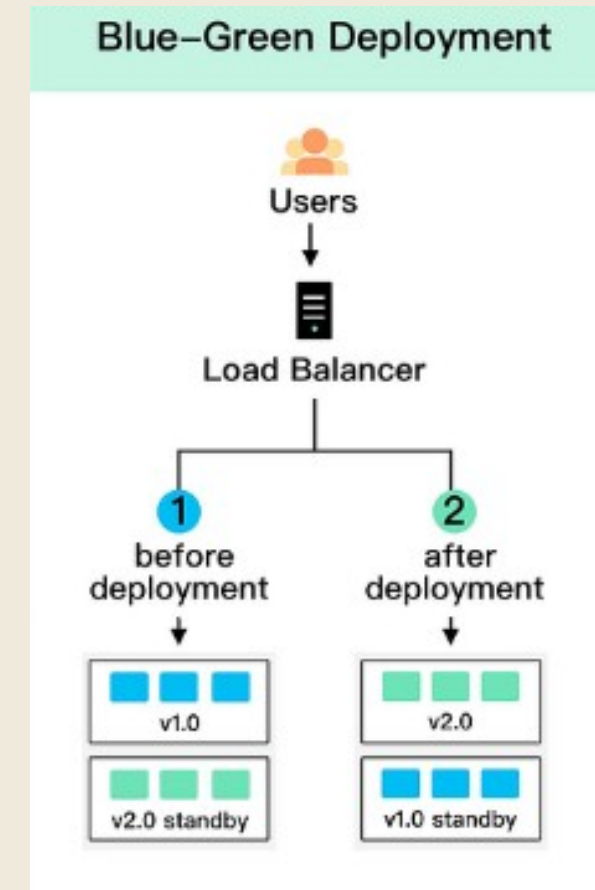
# Estrategia Avanzada Blue/Green Deployment

## Concepto:

Mantener dos entornos de producción idénticos y completos en paralelo: "Blue" (la versión actual activa) y "Green" (la nueva versión inactiva).

## ¿Cómo funciona?

1. **Desplegar la nueva versión completamente** en el entorno Green (mientras Blue sigue sirviendo todo el tráfico).
2. **Realizar pruebas exhaustivas en Green** (automatizadas, smoke manual si es necesario) usando una URL interna o cabecera especial.
3. **Cuando Green está validado**, cambiar el router/balanceador de carga para dirigir TODO el tráfico de usuarios de Blue a Green (¡este es el "switch"!): **Green se convierte en el nuevo Blue**.
4. **Mantener el antiguo entorno Blue (ahora inactivo) por un tiempo como backup inmediato para rollback**.



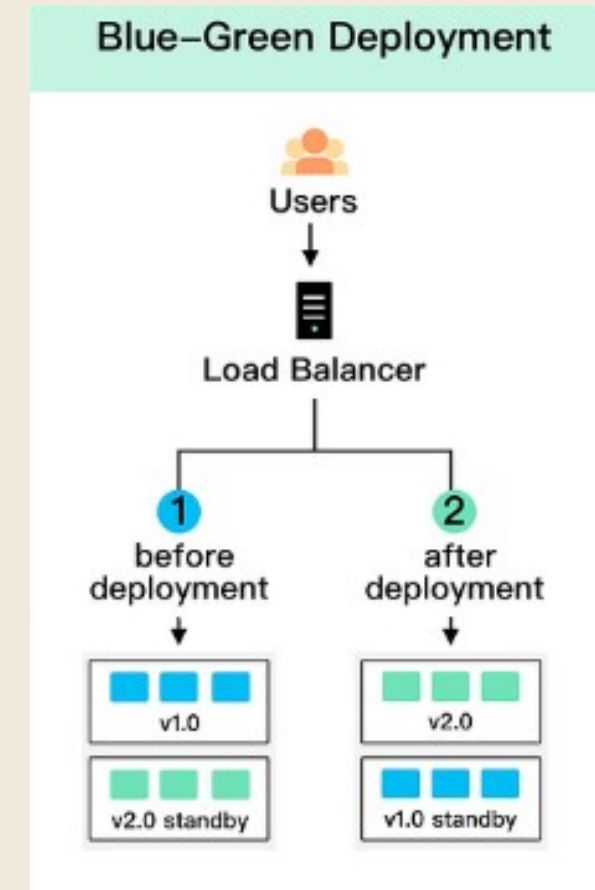
# Estrategia Avanzada Blue/Green Deployment

## Ventajas:

- **Rollback casi instantáneo:** Simplemente volver a apuntar el router al entorno Blue original.
- **Cero downtime** (idealmente) durante el cambio de tráfico.
- Los usuarios **nunca ven versiones mixtas**.
- Se pueden realizar pruebas completas en el entorno Green antes de exponerlo a usuarios.

## Desventajas:

- **Requiere** (al menos temporalmente) el **doble de infraestructura de producción** (puede ser costoso).
- **Puede ser complejo** gestionar cambios en la base de datos que requieran compatibilidad entre Blue y Green.
- Las sesiones de usuario largas pueden verse afectadas por el cambio.



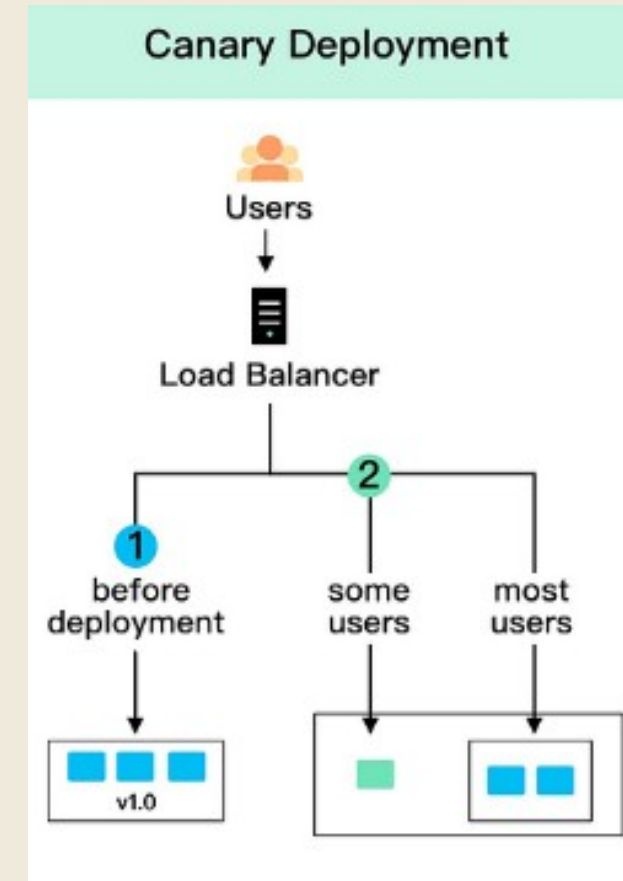
# Estrategia Avanzada Canary Release Deployment

## Concepto:

Introducir la nueva versión (v2) gradualmente, exponiéndola sólo a un pequeño porcentaje de usuarios/tráfico ("canarios") al principio.

## ¿Cómo funciona?

1. Desplegar v2 junto a la versión estable v1. Ambas activas.
2. Configurar el router/balancedor para enviar un pequeño % del tráfico (ej. 1%, 5%) a v2. El resto sigue yendo a v1.
3. **MONITOREAR INTENSIVAMENTE:** Observar de cerca las métricas, errores y logs específicos de v2.
4. Si las métricas de v2 son buenas y no hay errores, incrementar gradualmente el porcentaje de tráfico hacia v2 (10%, 25%, 50%, ..., 100%).
5. Si se detectan problemas en cualquier momento, revertir inmediatamente dirigiendo el 100% del tráfico de vuelta a v1 y parando/eliminando v2.





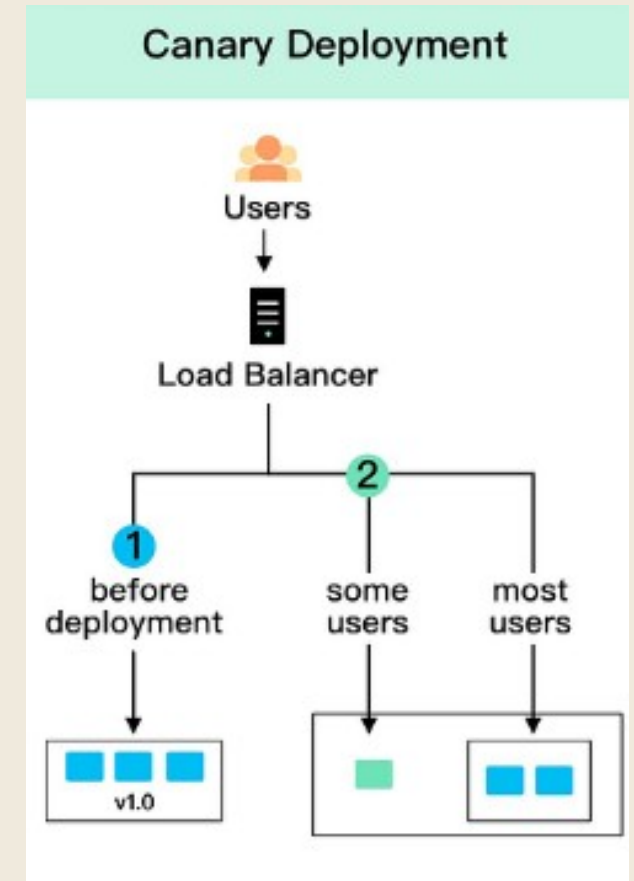
# Estrategia Avanzada Canary Release Deployment

## Ventajas:

- Prueba en producción real con usuarios reales pero con riesgo muy limitado inicialmente.
- Permite detectar problemas sutiles de rendimiento o bugs que no aparecieron en Staging.
- Feedback temprano del comportamiento real.

## Desventajas:

- Más complejo de implementar y gestionar (requiere enrutamiento de tráfico avanzado - L7).
- Necesita observabilidad muy madura para comparar el comportamiento de v1 y v2 y detectar problemas rápidamente.
- Puede complicar la gestión de sesiones de usuario o la consistencia de datos si no se maneja bien.





# Estrategia Avanzada Feature Flags

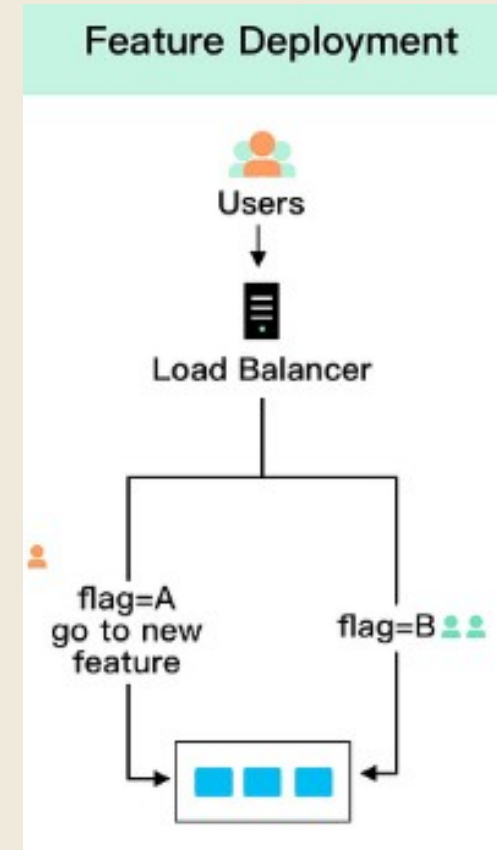
## Complemento

### Concepto:

Es una técnica que permite **activar o desactivar funcionalidades específicas de la aplicación en tiempo de ejecución**, sin necesidad de realizar un nuevo despliegue.

### ¿Cómo funciona?

- El código fuente incluye condicionales:  
→ if feature\_xyz\_enabled, then ...
- El estado de 'feature\_xyz' (activado/desactivado, para quién) se gestiona a través de un sistema de configuración externo.

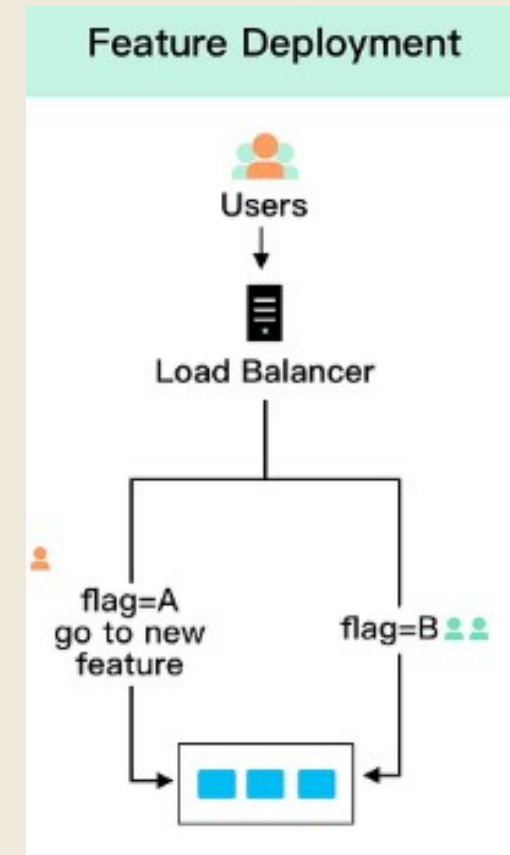


# Estrategia Avanzada Feature Flags

## Complemento

### Ventajas:

- **Desacoplar Despliegue de Lanzamiento:** Puedes desplegar código nuevo (incluso incompleto) a producción manteniéndolo desactivado hasta que esté listo para el lanzamiento. Reduce el riesgo del despliegue técnico.
- **Testing en Producción Controlado (Dark Launching):** Activar una nueva feature sólo para el equipo de desarrollo o un grupo beta en el entorno de producción.
- **Lanzamientos Graduales Basados en Segmentos:** Activar la feature progresivamente para diferentes grupos de usuarios (por país, tipo de suscripción, etc.).
- **"Kill Switch":** Desactivar rápidamente una funcionalidad que está causando problemas, sin necesidad de un rollback completo del código.
- **Puede combinarse** con Canary (activar flag sólo para canarios) o B/G (activar flags en Green antes del switch).



# Rollbacks Automatizados

## la red de seguridad final

Incluso con las mejores estrategias, algo puede salir mal.  
Necesitamos un plan B rápido y fiable: el rollback



### Objetivo:

Revertir el estado del sistema a la última versión estable conocida de forma rápida y, idealmente, automática cuando se detecta un problema grave post-despliegue.

### Disparadores:

- **Fallo de los Smoke Tests** ejecutados inmediatamente después del despliegue en producción.
- **Alertas críticas del sistema de Monitoreo** (ej. tasa de errores HTTP 5xx alta, latencia).
- **Degradación significativa de Métricas de Negocio clave** (Menos común automáticamente).
- **Disparo Manual a través de la herramienta de CD** ("botón de pánico") (Siempre disponible).

### Mecanismos:

- **Blue/Green:** Cambiar el router/LB de vuelta para apuntar al entorno Blue anterior. (¡Muy rápido!).
- **Rolling Update (K8s):** Usar el comando `kubectl rollout undo deployment <nombre-deployment>`. K8s gestiona la reversión gradual.
- **Canary:** Reconfigurar el router/LB para enviar 100% del tráfico a la versión estable v1. Escalar a cero (o eliminar) las instancias de la versión canary v2.
- **Serverless:** Re-apuntar el Alias (Lambda) o la configuración de tráfico (API Gateway) a la versión estable anterior.

# Validación Post-Despliegue:

## Smoke Tests & Health Checks

### Necesidad:

Justo después de desplegar en producción (o cualquier entorno), necesitamos una verificación rápida para confirmar que lo básico funciona

### Health Checks (Chequeos de Salud):



- **Propósito:** Verificaciones muy simples a nivel de proceso/infraestructura.
- **Ejemplos:** ¿El proceso de la aplicación está corriendo? ¿Responde el puerto HTTP? ¿Puede conectar a la BD? Una URL /health que devuelve {"status": "UP"}.
- **Uso:** Utilizados por Load Balancers (para sacar instancias malas del pool), Orquestadores (K8s Liveness/Readiness Probes para reiniciar/gestionar Pods) o inclusive tareas automatizadas periódicas.

### Smoke Tests (Pruebas de Humo):



- **Propósito:** Una suite pequeña y rápida de pruebas funcionales E2E que cubren la funcionalidad más crítica de la aplicación.
- **Ejemplos:** ¿Puedo cargar la página principal? ¿Puedo iniciar sesión? ¿Puedo realizar una búsqueda básica? ¿Puedo añadir un producto al carrito?
- **Ejecución:** Se lanzan contra el entorno real (ej. Producción) inmediatamente después de que el despliegue termina. ¡**CUIDADO! Si fallan, debe iniciarse un rollback inmediatamente manual o automático.**

**Observabilidad: Fundamental para la Confianza en CD**

---

# Observabilidad:

## Los Ojos y Oídos de Nuestro Pipeline CD

### Necesidad:

- Con despliegues frecuentes (potencialmente automáticos), **el tiempo para detectar problemas es crítico.**
- **Necesitamos feedback rápido y fiable** sobre la salud de la nueva versión.
- Es la **base para validar el éxito de un despliegue** (especialmente en Canary).
- **Permite tomar decisiones informadas** sobre si continuar un rollout o iniciar un rollback.
- Va más allá del monitoreo básico: **no solo saber si algo falla, sino poder entender por qué.**



### Pilares:

#### LOGS

(diario de la aplicación)

#### MÉTRICAS

(Latencia, tráfico, errores, saturación en: cpu, memoria, red, colas)

#### TRACING DISTRIBUIDO

¿En dónde está el problema?  
(Especialmente en MS, identifica cuellos de botella, errores en cascada, etc.)

### Herramientas:

**Recolección** (Prometheus, Telegraf, CloudWatch), **Almacenamiento** (Prometheus TSDB, InfluxDB, CloudWatch), **Visualización/Alertas** (Grafana, Datadog, Dynatrace, CloudWatch Dashboards/Alarms)

# Definiendo CD

---

# Definiendo Continuous Delivery

## Entrega Continua

### Definición:

Es una práctica donde los cambios de código que **pasan la fase de CI son automáticamente desplegados a un entorno similar a producción** (Staging, Pre-producción, UAT).

### Objetivo:

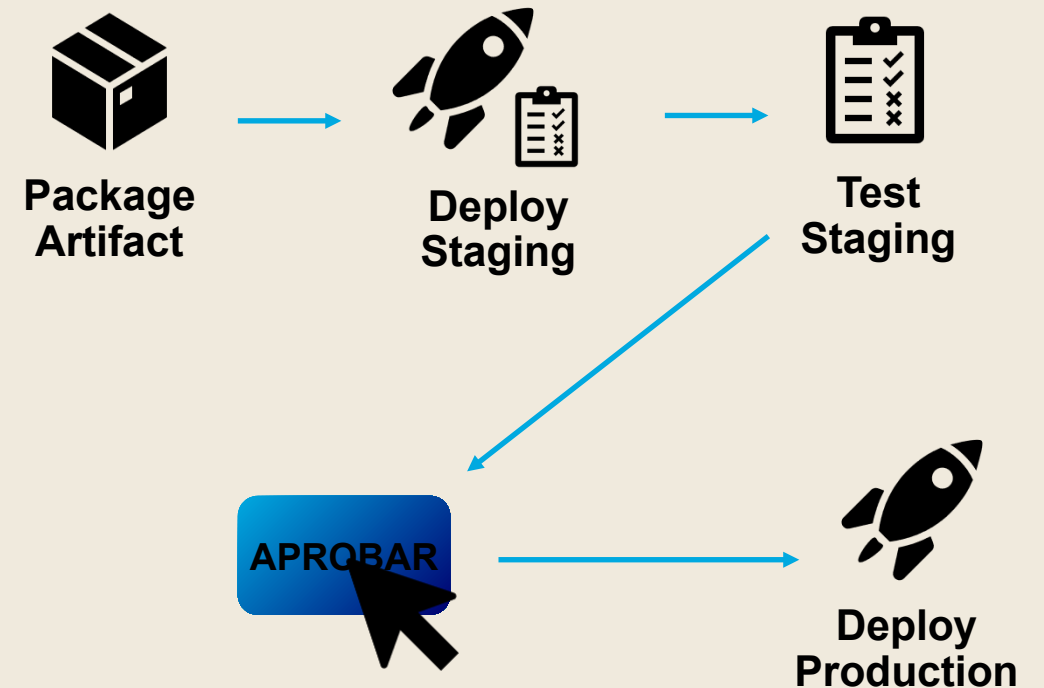
Asegurar que **siempre tengamos una versión del software probada en un entorno realista**, lista para ser liberada a producción en cualquier momento con **mínima preparación y riesgo**.

### Clave:

El **despliegue** final al entorno de Producción real **NO es automático**. Requiere una aprobación manual explícita (un "clic de botón").

### Estado:

**Siempre** hay una **versión del software** que podría ser liberada a producción con confianza y con solo un clic.

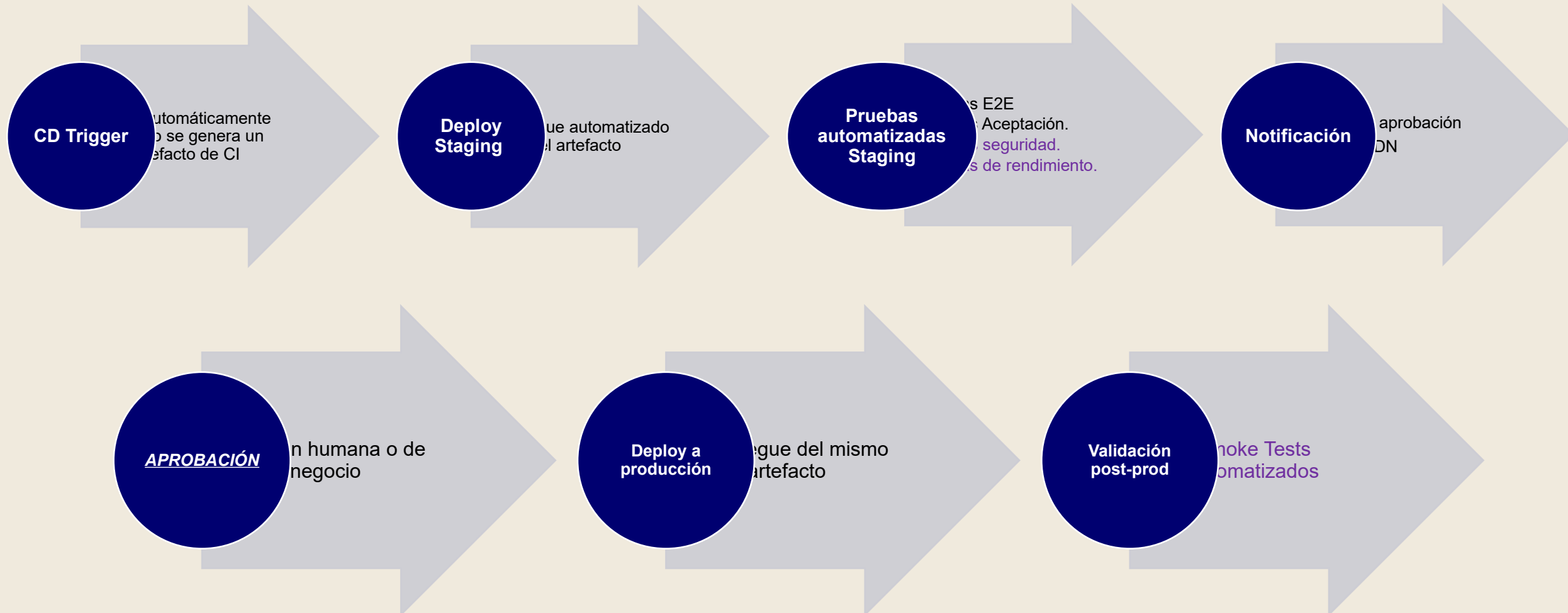




# Continuous Delivery

## Workflow

Pre requisito → Pipeline CI exitoso (Artefacto listo)



# Continuous Delivery

## Caso de Uso

### ¿Cuándo es adecuado?:

- Cuando se requieren **Pruebas de Aceptación de Usuario (UAT) manuales** antes de liberar.
- **Ventanas de lanzamiento** específicas controladas por negocio/marketing (lanzamientos coordinados).
- **Requerimientos regulatorios** (ej. Superintendencia Financiera en Colombia, sector salud), que exigen una revisión/aprobación final documentada.
- Como un **paso intermedio** para equipos que migran hacia una automatización más completa (Continuous Deployment).



# El siguiente nivel: Continuous Deployment

## Despliegue Continuo

### Definición:

Práctica donde **cada cambio que pasa todas las etapas automatizadas del pipeline** (CI y las etapas de prueba de CD) **se despliega directamente y automáticamente al entorno de Producción**.

### Objetivo:

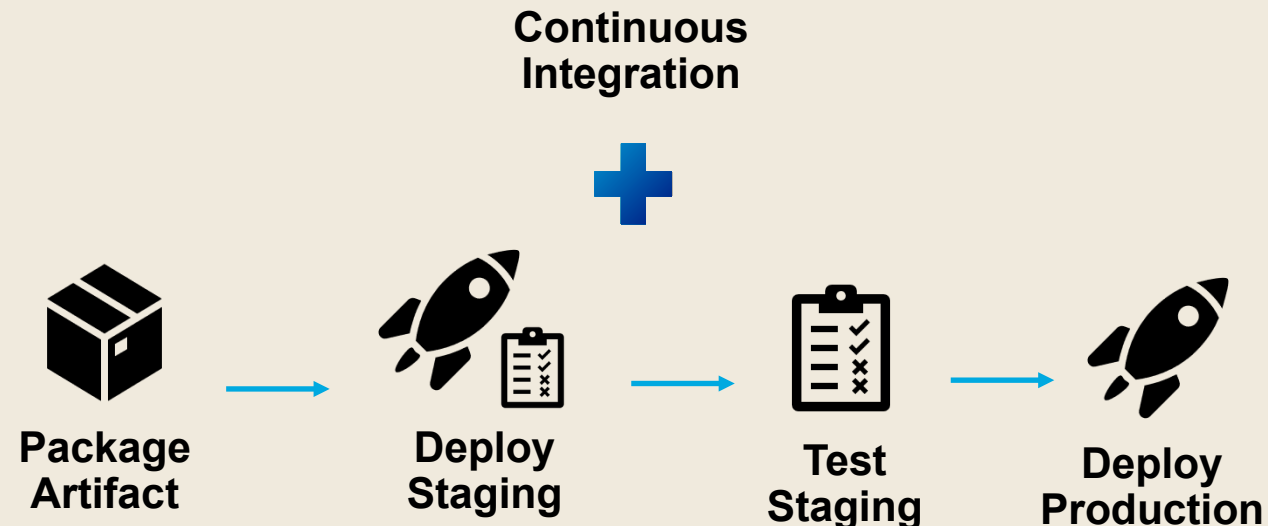
Minimizar radicalmente el tiempo desde que se escribe el código hasta que está en **producción** (Lead Time), permitiendo ciclos de feedback ultra rápidos y entrega de valor constante.

### Clave:

No hay intervención humana.

### Características:

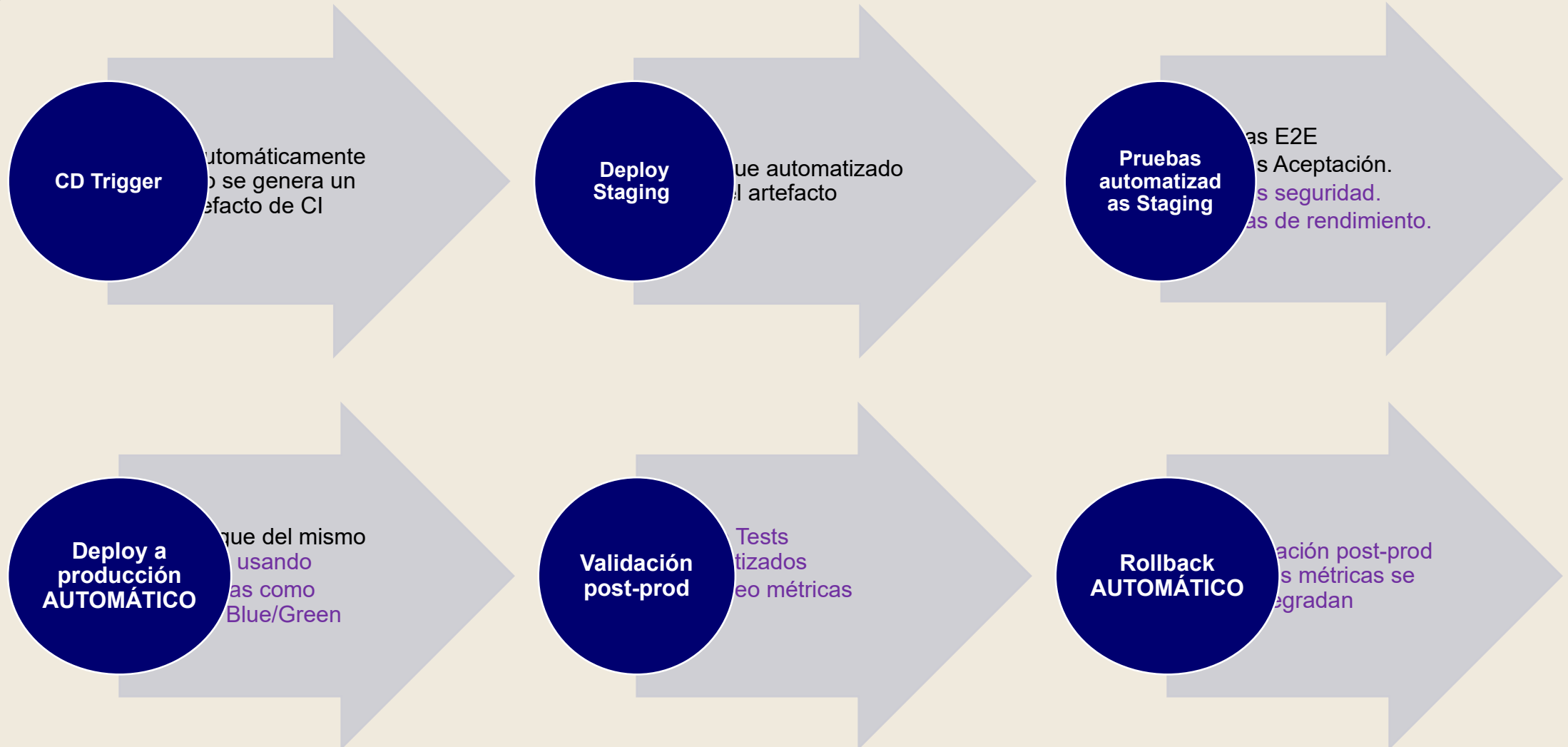
- No hay intervención humana en el proceso de despliegue a producción.
- **Confianza Absoluta:** Requiere una **fe muy alta en la red de seguridad automatizada** (pruebas exhaustivas, monitoreo robusto, rollbacks automáticos).



# Continuous Deployment

## Workflow

Pre requisito → Pipeline CI exitoso (Artefacto listo)



# Continuous Deployment

## Caso de Uso

### Requisitos indispensables:

- **Cultura de calidad** muy alta en el equipo.
- **Suite de pruebas automatizadas** extremadamente completa y fiable.
- **Monitoreo y observabilidad** maduros.
- **Infraestructura resiliente y automatizada (IaC).**
- Capacidad de realizar **rollbacks rápidos y automáticos** (¡y probados!).



High Confidence Testing Monitoring  
Advanced Monitoring

# Delivery vs. Deployment

## Las diferencias clave (tabla resumen)

Característica	Continuous Delivery	Continuous Deployment
Despliegue a Producción	Manual (Aprobación Requerida)	Automático (Tras pasar pipeline)
Intervención Humana	Sí (Decisión de liberar)	No (Idealmente)
Frecuencia de Release a Prod	Controlada (Días/Semanas/Bajo demanda)	Alta (Varias veces al día/Por commit)
Confianza Requerida	Alta en CI y Staging	Extrema en todo el pipeline + Monitoreo
Objetivo Principal	Listo para liberar en cualquier momento	Minimizar Lead Time al máximo
Riesgo Percibido	Menor (control humano final)	Mayor si la automatización falla

# **CD en Arquitecturas Modernas**

---

# CD con Contenedores Docker

## Portabilidad y Consistencia

### Motivante:

Los contenedores, popularizados por Docker, **son un pilar fundamental para CI/CD moderno.**

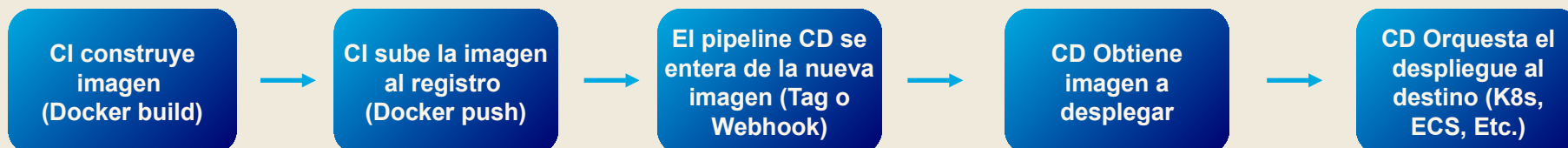
La Imagen Docker es un artefacto inmutable y autocontenido. Empaqueta la aplicación, sus dependencias, y la configuración del entorno de ejecución.

**Despliegues más rápidos, fiables y consistentes en todos los ambientes.**

### Registro de contenedores:

Es una **herramienta para almacenar y versionar las imágenes** (artefacto Docker). Es la fuente de verdad para el despliegue (usualmente se crea y almacena la imagen Docker que luego se despliega en los diferentes ambientes para prueba o puesta en producción).

### Pipeline CD con Contenedores (Simplificado):





# CD con Kubernetes (K8s)

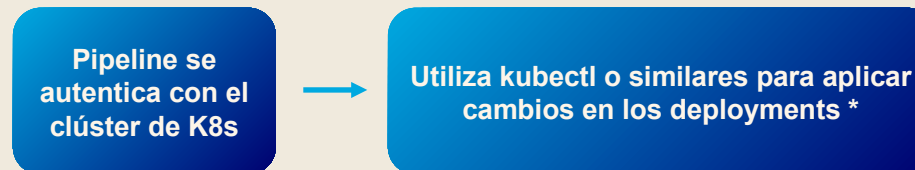
## Orquestación a escala

### ¿Qué es Kubernetes y cuál es su rol?:

Es la plataforma destino que recibe las instrucciones de despliegue del pipeline de CD. K8s se encarga de:

- **Iniciar contenedores** basados en la nueva imagen.
- **Gestionar el ciclo de vida** de los Pods (contenedores).
- **Manejar el enrutamiento de red** (Services).
- Realizar el **escalado**.
- Implementar la **estrategia de despliegue definida** (¡*Rolling Update* por defecto!).

### Interacción Pipeline CD <-> K8s API



\* Un deployment administra un conjunto de Pods para correr un flujo de aplicación.

# CD con Serverless

## FaaS

### ¿Cómo sería?:

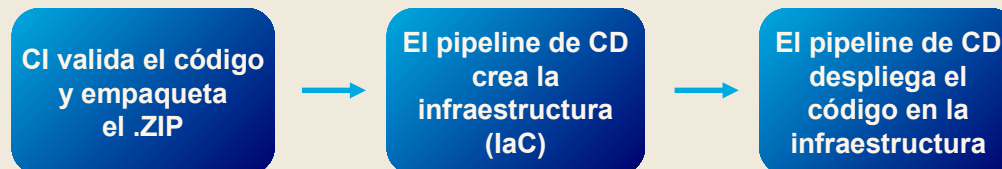
En arquitecturas Serverless (Functions as a Service - FaaS), no gestionamos servidores ni contenedores directamente.

**Desplegamos paquetes de código (funciones) y la configuración de la infraestructura** asociada (triggers, permisos, API Gateways), dejando que la plataforma cloud gestione la ejecución y el escalado.

### Artefacto típico:

**Un archivo ZIP con el código de la función y sus dependencias** + un archivo de **definición de infraestructura/aplicación** (ej. AWS SAM template.yaml, serverless.yml, Terraform main.tf, Cloudformation.yml).

### Pipeline CD con Serverless (Simplificado):



# Retos de CD en Microservicios

## Discusión

### Microservicio:

Un microservicio es una **arquitectura de software que descompone una aplicación en pequeñas piezas independientes**, cada una realizando una función específica y comunicándose con las demás.

Usualmente basados en contenedores.

### Retos que introducen:

- **Gestión de Dependencias:** Si el servicio A depende de una nueva API en el servicio B, ¿en qué orden los desplegamos?
- **Compatibilidad de APIs:** ¿Cómo aseguramos que un cambio en un servicio no rompe a sus consumidores? (Importancia de Pruebas de **Contrato**).
- **Pruebas E2E Complejas:** Validar un flujo completo que atraviesa múltiples servicios es difícil y potencialmente frágil.
- **Visibilidad y Debugging:** Si una transacción falla, ¿en qué servicio ocurrió el error? (Necesidad de Tracing Distribuido).
- **Consistencia de Datos:** Transacciones que abarcan múltiples servicios.
- **Complejidad del Pipeline:** ¿Un pipeline por servicio? ¿Un pipeline que orquesta varios?



# Contexto, Evolución y Medición de DevOps

---



# Contexto es Rey

## Adaptando el Pipeline CI/CD


### Motivación:

Ahora que hemos visto todos los componentes (CI, CD, Estrategias, Observabilidad, IaC, Seguridad), **entendemos que no existe un único pipeline 'perfecto'. Debe adaptarse.**

### Factores Clave:

- **Complejidad de la Aplicación:** Monolito vs. Microservicios.
- **Criticidad del Negocio:** E-commerce vs. Sistema de control aéreo/médico.
- **Madurez del Equipo/Organización:** Experiencia con automatización, cultura DevOps.
- **Requerimientos Regulatorios:** Industria financiera, salud, gobierno (ej. normativas locales).
- **Tolerancia al Riesgo:** ¿Cuánto impacto puede permitirse la empresa si un despliegue falla?

### Perfiles típicos:

- **Startup/Simple:** Prioriza velocidad. Pipeline corto, menos pruebas formales, despliegue manual/simple.
  - **Estable/SaaS:** Balance velocidad/seguridad. Pipeline más largo, pruebas robustas (incl. Perf/Sec), estrategias de despliegue automáticas (Canary/BG), observabilidad.
  - **Regulado/Crítico:** Prioriza seguridad/cumplimiento/estabilidad. Pipeline muy largo, pruebas exhaustivas, múltiples aprobaciones manuales, despliegues controlados, auditoría férrea.
- 

# NoOps vs NewOps

## El rol de Operaciones

### NoOps (visión futurista / aspiracional):

#### IDEA:

- Con suficiente automatización (Cloud, Serverless, CI/CD, IaC), la necesidad de un equipo de Operaciones tradicional se reduce drásticamente o desaparece.
- **Los desarrolladores gestionan todo el ciclo ("You build it, you run it").**

#### REALIDAD:

- A menudo es una simplificación excesiva.
- **La complejidad operativa no desaparece, se transforma** (en la plataforma, en la automatización, en la observabilidad, en los lineamientos y definiciones).

### NewOps (evolución práctica):

El rol de Operaciones **evoluciona**, no muere. **Pasa de tareas manuales y reactivas a ingeniería proactiva.**

#### ENFOQUE EN:

- **Construir y Mantener Plataformas Internas (Ingeniería de Plataformas):** Crear las herramientas, las automatizaciones y plataformas de autoservicio (incluyendo el pipeline CI/CD, Referencias de IaC y Cloud) que los desarrolladores usan.
- **Ingeniería de Fiabilidad del Sitio (SER – Ingeniería de Confiabilidad del Sitio):** Aplicar principios de ingeniería de software, reglas y lineamientos para hacer los sistemas productivos más fiables, escalables y eficientes (definir SLOs (service level objectives), gestionar incidentes, automatizar tareas operativas).
- **Maestría en Automatización y Observabilidad.**

**CD es un Habilitador Fundamental:**  
Libera tiempo de Ops para enfocarse en estas  
tareas de mayor valor.

# Impacto de CI-CD en las Métricas DevOps

## DORA

“Implementar CI/CD no es solo *'hacer cosas más rápido'*.”

Tiene un impacto medible en el rendimiento del equipo y la entrega de valor, como lo demuestran investigaciones como el reporte DORA (DevOps Research and Assessment)”

# Métricas DORA

Change  
Lead Time



**Plazo de entrega de los cambios:**  
Es el tiempo que tarda una confirmación o un cambio de código en implementarse correctamente en producción.



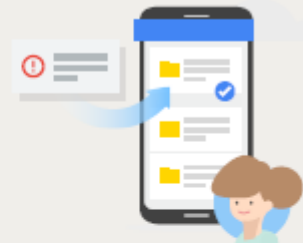
**Frecuencia de implementación:**  
Es la frecuencia con la que se implementan cambios de aplicaciones en la producción.

Deployment  
Frequency

Change  
Fail Rate



**Tasa de errores de los cambios:**  
Es el porcentaje de implementaciones que causan errores en la producción<sup>1</sup> y deben corregirse o revertirse.



**Tiempo de recuperación ante implementaciones con errores:**  
Es el tiempo necesario para recuperarse de una implementación con errores.

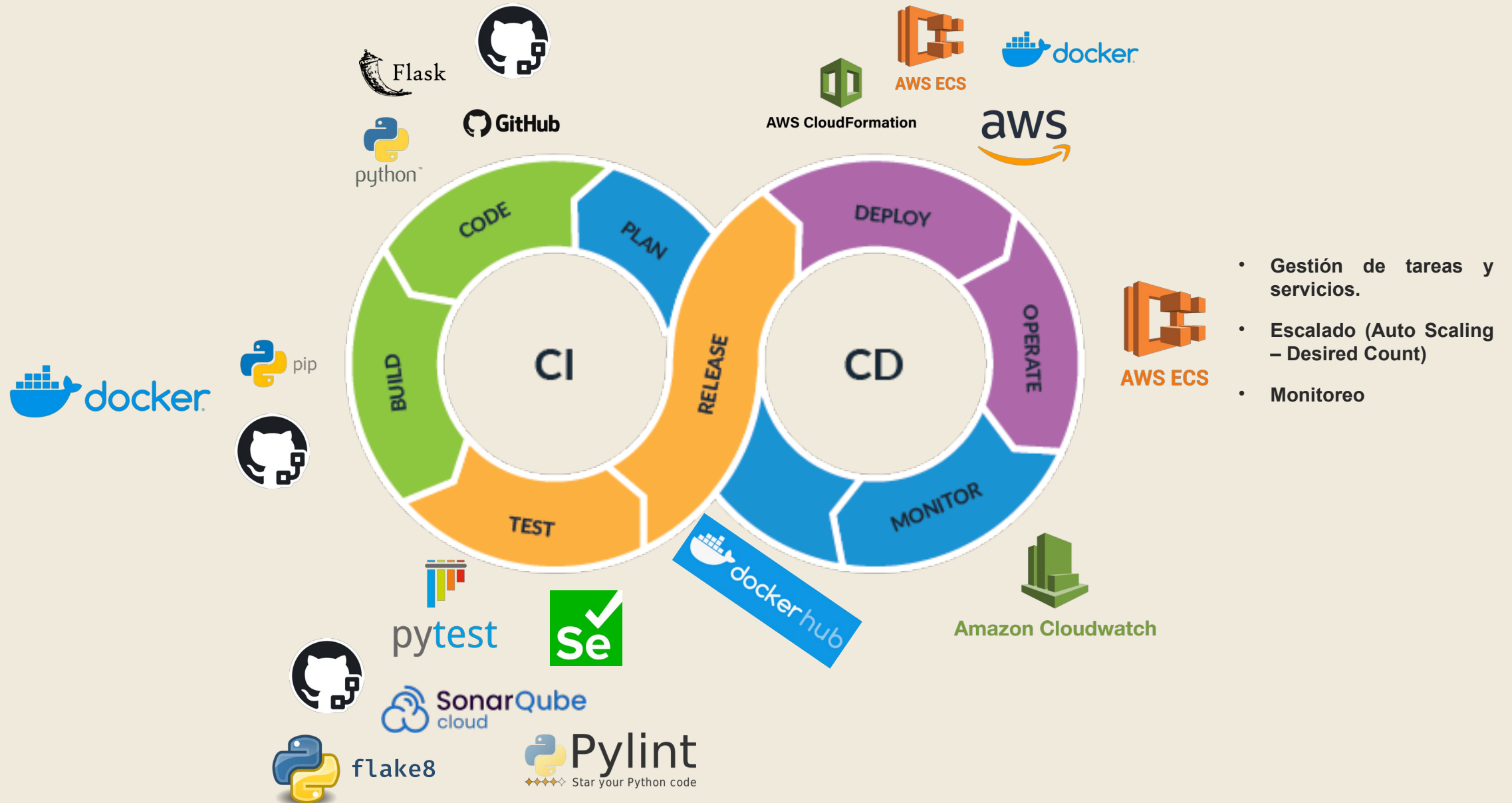
Time to  
Restore Service



# Taller CD

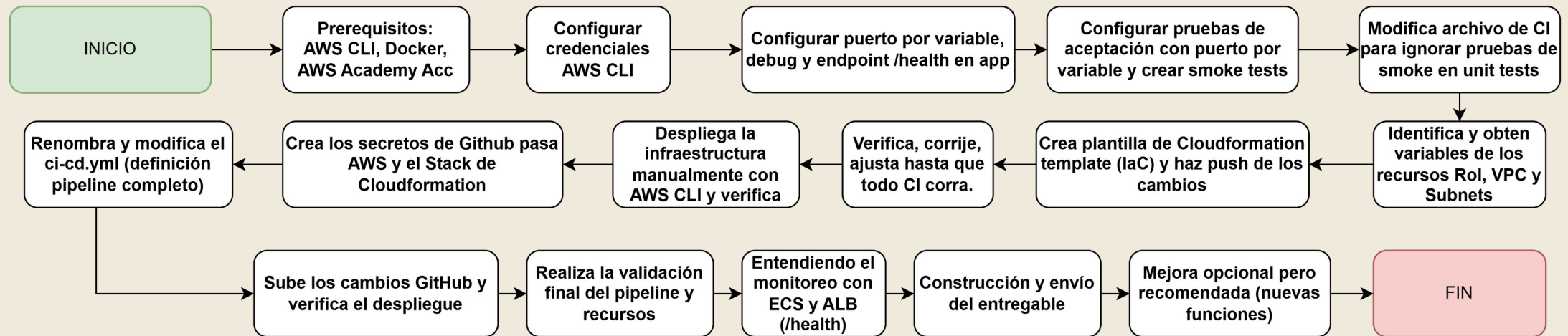
---

# Herramientas el taller de CI - CD



# TALLER GRUPAL ENTREGABLE 3

## Despliegue Continuo (CD) con Proveedores Cloud



**LA ENTREGA SE RECIBE HASTA EL SÁBADO 20 DE SEPTIEMBRE  
A LAS 11:59 PM**


# **Foro**

## **DORA, AI y GenAI en DevOps y sus empresas**

Lectura de recursos

**PREVIO A LA CLASE DEL VIERNES 26 SEPT**

- DORA – Accelerate State of DevOps (páginas 9 a 76).
  - DORA - Impact of Generative AI in SW Development (RECOMENDADO - OPCIONAL)
- 



**Proyecto final en grupos**  
Elegir un artefacto de software diferente al trabajado en clase que posea una **complejidad media** + implementar un **pipeline completo de CI/CD** que demuestre todo lo visto en clase **con al menos dos modificaciones o introducciones a nivel de pipeline o arquitectura** + **Presentación final**.

### **Presentación final:**

- **Explicación del artefacto de software** escogido (código a alto nivel y funcionalidades).
- **Explicación del diseño y los componentes del pipeline CI/CD** implementado (Flujograma visual con cada paso y herramienta utilizada en cada paso).
- **Explicación del diagrama arquitectura de despliegue** utilizada (si se usó ECS, ECR, ALB, Lambdas, etc.).
- **Explicación del(los) archivo(s) de configuración** del(los) pipeline(s) implementados.
- **Evidencia del(los) pipeline(s) ejecutado(s) exitosamente.**
- **Evidencia de las dos modificaciones o introducciones a nivel de pipeline o arquitectura.**
- **Demostración de App funcional** en cada ambiente + Infraestructura.
- **Socialización de desafíos y aprendizajes.**

### **Qué deben incluir los pipelines (mínimo):**

- **CI:** Checkout -> Build -> Pruebas (Unitarias) -> Release (Creación del artefacto a desplegar) .
- **CD:** IaC -> Deploy Infra ( $\geq 2$  Ambientes) -> Deploy App ( $\geq 2$  Ambientes) -> Pruebas Aceptación (Pre-Prod) -> Pruebas Humo (Prod) -> Rollback (ante fallo).

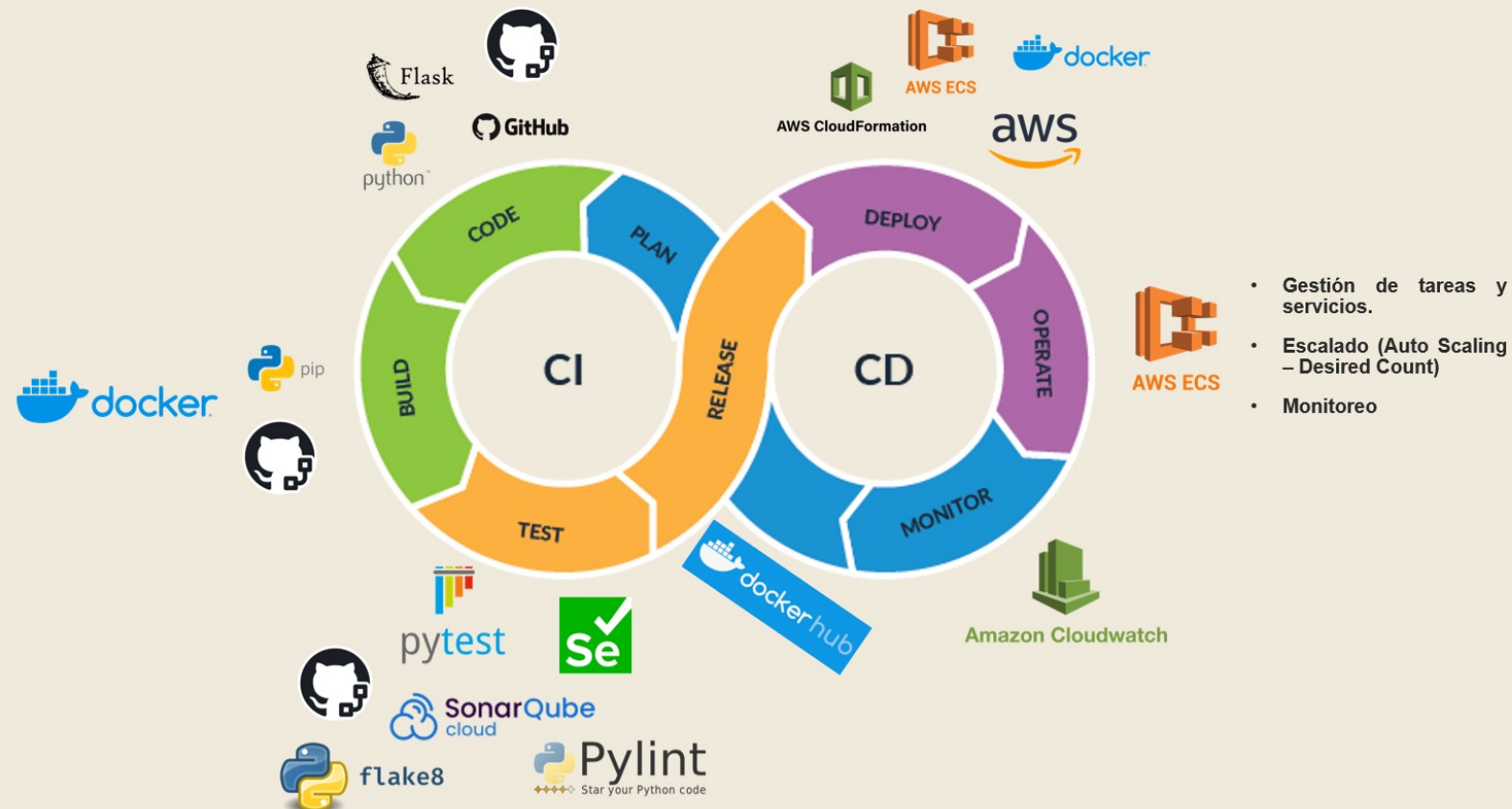
**Fecha de presentación: Sábado 27 de Septiembre**



# TALLER GRUPAL ENTREGABLE 3

## Despliegue Continuo (CD) con Proveedores Cloud

¿DUDAS? ¿PREGUNTAS? ¿INCONVENIENTES?



# Resumen Clave del Módulo

- CD extiende CI para automatizar la entrega (Delivery=Manual, Deployment=Auto).
- Pipeline CD incluye etapas de despliegue (IaC y Código), pruebas avanzadas (E2E, Perf, Sec) y validación.
- Estrategias (Rolling, B/G, Canary, Flags) y Rollbacks son clave para despliegues seguros.
- Observabilidad (Logs, Métricas, Trazas) es indispensable para la confianza y diagnóstico en CD.
- IaC automatiza entornos; Seguridad se integra en todo el pipeline CD.
- El pipeline se adapta al contexto (app, equipo, regulación).
- CI-CD impulsa NewOps/SRE y mejora drásticamente las métricas DORA



# Preguntas y Discusión

¿Hay algún concepto que les gustaría repasar?

¿Cómo creen que podrían empezar a aplicar DevOps en sus proyectos actuales [Medellín/Colombia/Su Empresa]?





# Referencias Clave

"Continuous Delivery" – Humble, Farley

"Accelerate" – Forsgren, Humble, Kim

"The DevOps Handbook" – Kim, Debois, Willis, Humble

"The Phoenix Project" / "The Unicorn Project" – Gene Kim et al.

Site Reliability Engineering & The SRE Workbook (Google/O'Reilly)

Observability Engineering – Charity Majors, Liz Fong-Jones, George Miranda

Infrastructure as Code – Kief Morris

<https://dora.dev/>



**Inspira  
Crea  
Transforma**

[www.eafit.edu.co](http://www.eafit.edu.co)

VIGILADA | MINEDUCACIÓN

**UNIVERSIDAD  
EAFIT**