DESARROLLO DE SOTWARE

# BIBLIOGRAFÍA

Learning DevOps - Second Edition

Modern DevOps Practices - Second Edition

Learning Continuous Integration with Jenkins - Third Edition

Continuous Delivery with Docker and Jenkins - Third Edition

Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines

Docker: Up & Running, 3rd Edition

The Ultimate Docker Container Book - Third Edition

Head First Software Architecture

Mastering Kubernetes - Fourth Edition

Kubernetes in Action

Terraform: Up and Running, 3rd Edition

Terraform Cookbook - Second Edition

Head First Design Patterns, 2nd Edition

Fundamentals of Software Architecture

Software Architecture in Practice, 4th Edition

Software Architecture with C# 12 and .NET 8 - Fourth Edition

Modern Data Architectures with Python

Software Architecture: The Hard Parts

Building Microservices, 2nd Edition

Practical Microservices Architectural Patterns: Event-Based Java Microservices with Spring Boot and Spring Cloud

Microservices with Spring Boot 3 and Spring Cloud - Third Edition

Microservice APIs

Microservices Patterns

Bootstrapping Microservices with Docker, Kubernetes, and Terraform

Designing Data-Intensive Applications

Practical GitOps: Infrastructure Management Using Terraform, AWS, and GitHub Actions

Istio in Action

Fluent Python, 2nd Edition

Full Stack Testing

The DevOps Handbook

Clean Code: A Handbook of Agile Software Craftsmanship

The Art of Agile Development, 2nd Edition

Essential Scrum: A Practical Guide to the Most Popular Agile Process

Infrastructure as Code, Patterns and Practices

The Complete Developer

Docker in Practice

Python for DevOps

The Linux DevOps Handbook

DevOps for the Desperate

## Cursos y videos

Web Services/Rest API Testing with SoapUI and Real-time Projects

Selenium Python Automation Testing from Scratch and Frameworks

## Empezando con DevOps

(cap. 1 de Learning DevOps - Second Edition)

El término DevOps Fue introducido en 2007-2009 por Patrick Debois, Gene Kim y John Willis, y representa la combinación de Desarrollo (Dev) y Operaciones (Ops). Ha dado lugar a un movimiento que aboga por unir a los desarrolladores y las operaciones dentro de equipos. Esto ofrece valor comercial agregado a los usuarios más rápidamente, lo que lo hace más competitivo en el mercado.

La cultura DevOps es un conjunto de prácticas que reducen las barreras entre los desarrolladores, que quieren innovar y entregar más rápido, y las operaciones, que quieren garantizar la estabilidad de los sistemas de producción y la calidad de los cambios que realizan en el sistema.

La cultura DevOps es también la extensión de los procesos ágiles (Scrum, XP, etc.), que permite reducir los tiempos de entrega y que ya involucra a desarrolladores y equipos comerciales; sin embargo, a menudo se ven obstaculizados por la no inclusión de Ops en los mismos equipos.

La comunicación y este vínculo entre Dev y Ops permite un mejor seguimiento de la producción de extremo a extremo e implementaciones más frecuentes y de mayor calidad, ahorrando dinero a la empresa.

Para facilitar esta colaboración y mejorar la comunicación entre Dev y Ops, existen varios elementos clave en los procesos que deben implementarse, como se muestra aquí:

* Más frecuentes Implementaciones de aplicaciones con integración y entrega continua (llamadas CI/CD).
* La implementación y automatización de pruebas unitarias y de integración, con un proceso centrado en el diseño basado en el comportamiento (BDD) o en el diseño basado en pruebas (TDD).
* E Implementación de un medio para recopilar comentarios de los usuarios.
* Monitoreo de aplicaciones e infraestructura.

**El movimiento DevOps se sustenta en tres ejes:**

* **La cultura de la colaboración**: *esta es la esencia misma de DevOps: el hecho de que los equipos ya no están separados por silos de especialización (un equipo de desarrolladores, un equipo de operaciones, un equipo de evaluadores, etc.). Sin embargo, estas personas se unen formando equipos multidisciplinares que tienen un mismo objetivo: entregar valor añadido al producto lo más rápido posible.*
* **Procesos**: *Para esperar un despliegue rápido, estos equipos deben seguir procesos de desarrollo a partir de metodologías ágiles con fases iterativas que permitan una mejor funcionalidad, calidad y retroalimentación rápida. Estos procesos no solo deben integrarse en el flujo de trabajo de desarrollo con integración continua, sino también en el flujo de trabajo de implementación con entrega e implementación continuas.* El proceso DevOps se divide en varias fases:

1. Planificación y priorización de funcionalidades
2. Desarrollo
3. Integración y entrega continuas
4. Despliegue continuo
5. Monitoreo continuo

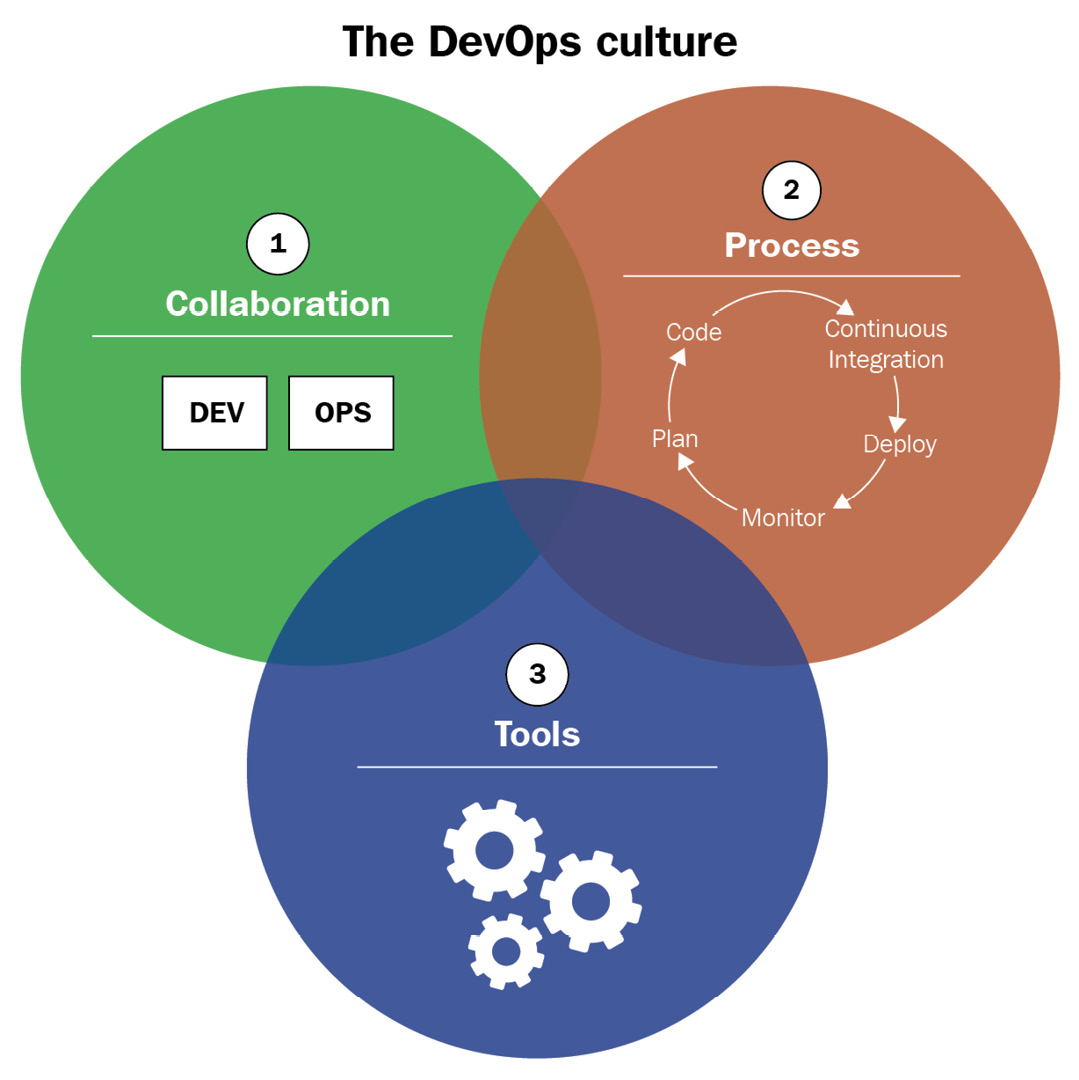
* **Herramientas**: La elección de herramientas y productos utilizados por los equipos es muy importante en DevOps. De hecho, cuando los equipos se separaron en Dev y Ops, cada equipo utilizó sus herramientas específicas (herramientas de implementación para desarrolladores y herramientas de infraestructura para Ops), lo que amplió aún más las brechas de comunicación.

Con equipos que unir el desarrollo y las operaciones, y con esta cultura de unidad, las herramientas que se utilizan deben ser “usables” y explotables por todos los miembros.

Los desarrolladores deben integrarse con las herramientas de monitoreo que utilizan los equipos de Ops para detectar problemas de rendimiento lo antes posible, y con las herramientas de seguridad proporcionadas por Ops para proteger el acceso a diversos recursos.

Ops, por otro lado, debe automatizar el proceso de creación y actualización de la infraestructura e integrar el código en un administrador de código. Esto se llama IaC, pero sólo se puede hacer en colaboración con desarrolladores que conocen la infraestructura necesaria para las aplicaciones. Las operaciones también deben integrarse en los procesos y herramientas de lanzamiento de aplicaciones.

El siguiente diagrama ilustra los tres ejes de la cultura DevOps: la colaboración entre Dev y Ops, los procesos y el uso de herramientas:



"DevOps es la unión de personas, procesos y productos para permitir la entrega continua de valor a nuestros usuarios finales".

Los beneficios de establecer una cultura DevOps dentro de una empresa son los siguientes:

Mejor colaboración y comunicación en equipos, lo que tiene un impacto humano y social dentro de la empresa.

Plazos de entrega más cortos para la producción, lo que resulta en un mejor rendimiento y satisfacción del usuario final.

Costos de infraestructura reducidos con IaC

Ahorro de tiempo significativo con ciclos iterativos que reducen los errores de aplicación y la automatización de herramientas que reducen las tareas manuales, por lo que los equipos se centran más en desarrollar nuevas funcionalidades con valor comercial agregado.

## Implementación de CI/CD y despliegue continuo

En las siglas de CI/CD, existen tres prácticas:

* Integración continua (CI)
* Entrega continua (CD)
* Despliegue continuo

### Integración continua (CI)

Siguiendo la definición dada por Martin Fowler, se mencionan tres cosas clave: miembros de un equipo , integrarse y lo más rápido posible :

*"La integración continua es una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo con frecuencia... Cada integración se verifica mediante una compilación automatizada (incluidas pruebas) para detectar errores de integración lo más rápido posible".*

**Es decir, CI es un proceso automático que le permite verificar la integridad del código de una aplicación cada vez que un miembro del equipo realiza un cambio.** Esta verificación debe realizarse lo más rápido posible.

Vemos la cultura devops en CI muy claramente, con espíritu de colaboración y comunicación, porque la ejecución de CI impacta a todos los integrantes en términos de metodología de trabajo y por ende de colaboración; Además, la CI requiere la implementación de procesos (branch, commit, pull request, revisión de código, etc.) con una automatización que se realiza con herramientas adaptadas a todo el equipo (Git, Jenkins, Azure DevOps, etc.) . Finalmente, la CI debe ejecutarse rápidamente para recopilar comentarios sobre la integración del código lo antes posible y, por lo tanto, poder ofrecer nuevas funciones más rápidamente a los usuarios.

### Implementación de CI

Por lo tanto, para configurar CI, es necesario tener un **Administrador de código fuente** (SCM) que centralizará el código de todos los miembros. Este administrador de códigos puede ser de cualquier tipo: Git, SVN o Team Foundation Version Control (TFVC). También es importante tener **un administrador de compilación automático (servidor CI**) que admita integración continua, como Jenkins, GitLab CI, TeamCity, Azure Pipelines, GitHub Actions, Travis CI y Circle CI.

*Cada miembro del equipo trabajará en el código de la aplicación diariamente, de forma iterativa e incremental (como en métodos ágiles y scrum). Cada tarea o característica debe separarse de otros desarrollos mediante el uso de ramas.*

*Regularmente, incluso varias veces al día, los miembros archivan o confirman su código y preferiblemente con pequeñas confirmaciones (troncales) que pueden corregirse fácilmente en caso de error. Este se integrará con el resto del código de la aplicación, con el resto de commits de los demás miembros.*

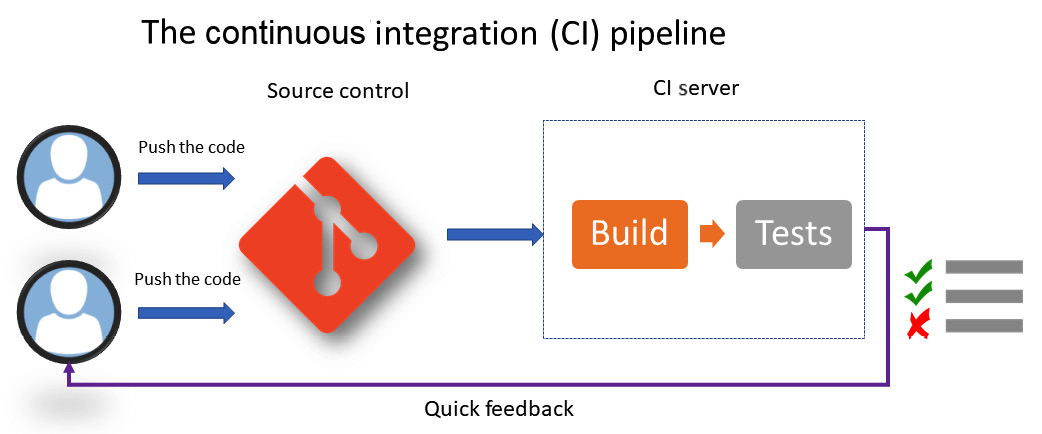
*La integración de todas las confirmaciones es el punto de partida del proceso de CI.*

Este proceso, que ejecuta el servidor de CI, debe automatizarse y activarse en cada confirmación. El servidor recuperará el código y luego hará lo siguiente:

* Creará el paquete de la aplicación: compilación, transformación de archivos, etc.
* Realizar pruebas unitarias (con cobertura de código)

*Este proceso de CI debe optimizarse lo antes posible para que pueda ejecutarse rápidamente y para que los desarrolladores puedan recopilar comentarios rápidos sobre la integración de su código; por ejemplo, el código que se ha archivado y no se compila o cuya ejecución de prueba falla puede afectar y bloquear a todo el equipo.*

Con un proceso de CI optimizado y completo, el desarrollador puede solucionar rápidamente sus problemas y mejorar su código o discutirlo con el resto del equipo y confirmar su código para una nueva integración.



### Entrega continua (CD)

Una vez que se ha completado la integración continua, el siguiente paso es implementar la aplicación automáticamente en uno o más entornos de producción, lo que se llama puesta en escena. **Este proceso se llama entrega continua (CD).**

El CD a menudo comienza con un paquete de aplicación preparado por CI, que se instalará en función de una lista de tareas automatizadas. Estas tareas pueden ser de cualquier tipo: descomprimir, detener y reiniciar el servicio, copiar archivos, reemplazar configuración, etc. La ejecución de pruebas funcionales y de aceptación también se puede realizar durante el proceso de CD.

*A diferencia de CI, CD tiene como objetivo probar la aplicación completa con todas sus dependencias. Esto es muy visible en aplicaciones de microservicios compuestas por varios servicios y API; CI solo probará el microservicio en desarrollo, mientras que una vez implementado en un entorno de prueba, será posible probar y validar toda la aplicación, así como las API y microservicios que la componen.*

*En la práctica, hoy en día, es muy común vincular CI con CD en un entorno de integración; es decir, la CI se implementa al mismo tiempo en un entorno*. Esto es necesario para que los desarrolladores no sólo puedan ejecutar pruebas unitarias sino también verificar la aplicación en su conjunto (UI y funcional) en cada confirmación, junto con la integración de los desarrollos de los demás miembros del equipo.

Es importante que el paquete que se genera durante la CI, que también se implementará durante el CD, sea el mismo que se instalará en todos los entornos, y este debería ser el caso hasta la producción. Sin embargo, puede haber transformaciones de archivos de configuración que difieran según el entorno, pero el código de la aplicación (binarios, DLL, imágenes de Docker y JAR) debe permanecer sin cambios.

*Este carácter inmutable del código es la única garantía de que la aplicación que se verifica en un entorno será de la misma calidad que la versión que se implementó en el entorno anterior, y también la misma que se implementará en el siguiente entorno. Si se van a realizar cambios (mejoras o correcciones de errores) en el código después de la verificación en uno de estos entornos, una vez realizado, las modificaciones deberán pasar por el ciclo de CI y CD nuevamente.*

Las herramientas configuradas para CI/CD se utilizan a menudo con otras soluciones, como se muestra a continuación:

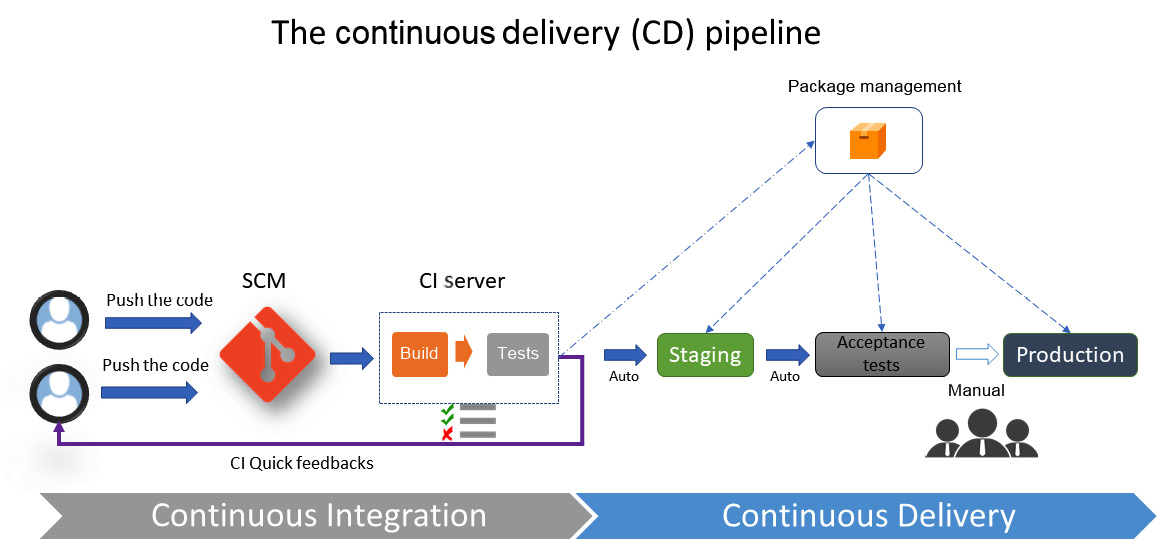
**Un administrador de paquetes**: Constituye el espacio de almacenamiento de los paquetes generados por CI y recuperados por CD. Estos administradores deben admitir feeds, versiones y diferentes tipos de paquetes. Hay varios en el mercado, como Nexus, ProGet, Artifactory y Azure Artifacts.

**Un administrador de configuración**: esto le permite administrar los cambios de configuración durante el CD; la mayoría de las herramientas de CD incluyen un mecanismo de configuración con un sistema de variables.

En CD, la implementación de la aplicación en cada entorno de prueba se activa de la siguiente manera:

* **Puede activarse automáticamente**, tras una ejecución exitosa en un entorno anterior; por ejemplo, podemos imaginar un caso en el que la implementación en el entorno de preproducción se activa automáticamente cuando las pruebas de integración se han realizado con éxito en un entorno dedicado.
* **Puede activarse manualmente**, para entornos sensibles como el entorno de producción, previa aprobación manual por parte de la persona responsable de validar la funcionalidad adecuada de la aplicación en un entorno.

Lo que es importante en un proceso de CD es que la implementación en el entorno de producción (es decir, en el usuario final) la activan manualmente los usuarios aprobados.



El diagrama anterior muestra claramente que el proceso de DC es una continuación del proceso de CI. Representa la cadena de pasos del CD, que son automáticos para entornos de ensayo, pero manuales para implementaciones de producción. También muestra que el paquete es generado por CI y almacenado en un administrador de paquetes, y que es el mismo paquete que se implementa en diferentes entornos.

### Despliegue continuo

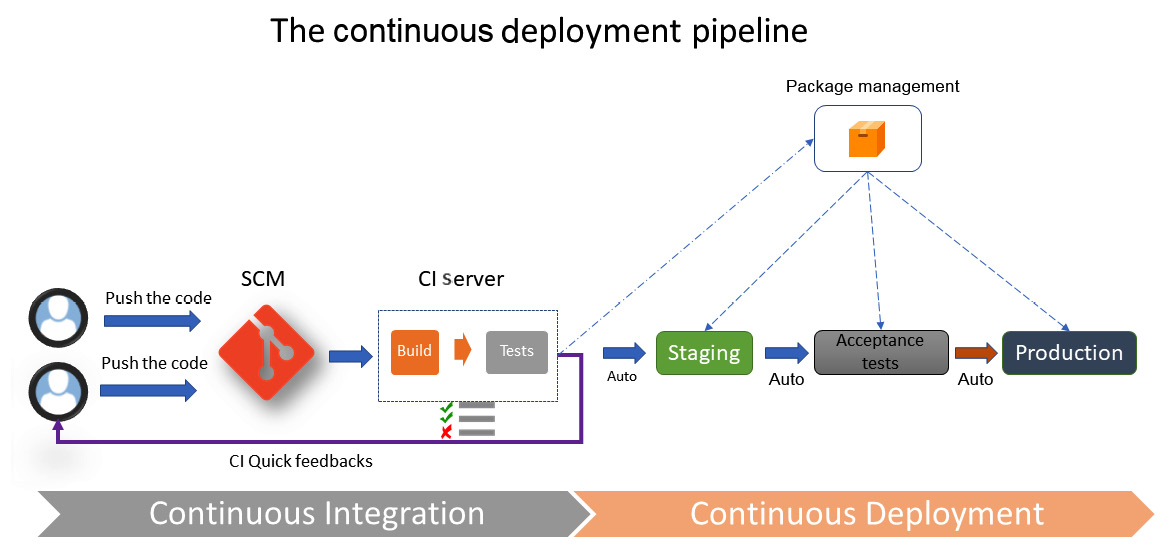
La implementación es una extensión de CD, pero esta vez, con un proceso que automatiza todo el proceso de CI/CD desde el momento en que el desarrollador envía su código a la implementación en producción a través de todos los pasos de verificación.

*Esta práctica rara vez se implementa en las empresas porque requiere cubrir una variedad de pruebas (unitarias, funcionales, de integración, de rendimiento, etc.) para la aplicación*. La ejecución exitosa de estas pruebas es suficiente para validar la funcionalidad adecuada de la aplicación con respecto a todas estas dependencias; sin embargo, también le permite implementar automáticamente en un entorno de producción sin necesidad de ninguna acción de aprobación.

El proceso de implementación continua también debe tener en cuenta todos los pasos para restaurar la aplicación en caso de un problema de producción.

La implementación se puede implementar mediante el uso de técnicas de alternancia de funciones (o indicadores de funciones), lo que implica encapsular las funcionalidades de la aplicación en funciones y activar sus funciones bajo demanda, directamente en producción, sin tener que volver a implementar el código de la aplicación.

Otra técnica consiste en utilizar una infraestructura de producción azul-verde, que consta de dos entornos de producción, uno azul y otro verde. Primero, nos desplegamos en el entorno azul, luego en el verde; esto asegurará que no se requiera tiempo de inactividad.



El diagrama anterior es casi el mismo que el de CD, pero con la diferencia de que muestra la implementación automatizada de un extremo a otro.

*Por lo tanto, los procesos de CI/CD son una parte esencial de la cultura DevOps, y la CI permite a los equipos integrar y probar la coherencia de su código y obtener comentarios rápidos con regularidad*. El CD se implementa automáticamente en uno o más entornos de prueba y, por lo tanto, ofrece la posibilidad de probar toda la aplicación hasta que se implemente en producción.

## Comprender las prácticas de IaC

**IaC es una práctica que consiste en escribir el código de los recursos que componen una infraestructura.**

Esta práctica comenzó a surtir efecto con el auge de la cultura DevOps y con la modernización de la infraestructura de la nube. De hecho, los equipos de operaciones que implementan infraestructuras manualmente se toman el tiempo para realizar cambios en la infraestructura debido al manejo inconsistente y al riesgo de errores. Además, con la modernización de la nube y su escalabilidad, la forma en que se construye la infraestructura requiere revisar las prácticas de aprovisionamiento y cambio adaptando un método más automatizado.

**IaC es el proceso de escribir el código de los pasos de aprovisionamiento y configuración de los componentes de la infraestructura, lo que ayuda a automatizar su implementación de manera repetible y consistente.**

### Los beneficios de la IaC

Los beneficios de IaC son los siguientes:

* La estandarización de la configuración de la infraestructura reduce el riesgo de errores.
* El código que describe la infraestructura está versionado y controlado en un administrador de código fuente.
* El código está integrado en canales de CI/CD.
* Las implementaciones que realizan cambios en la infraestructura son más rápidas y eficientes.
* Hay una mejor gestión, control y reducción de costos de infraestructura.

IaC también aporta beneficios a un equipo de DevOps al permitir que Ops sea más eficiente en términos de tareas de mejora de infraestructura, en lugar de perder tiempo en la configuración manual. También brinda a los desarrolladores la posibilidad de actualizar sus infraestructuras y realizar cambios sin tener que solicitar más recursos de operaciones.

IaC también permite la creación de entornos efímeros de autoservicio que brindarán a los desarrolladores y evaluadores más flexibilidad para probar nuevas funciones de forma aislada e independiente de otros entornos.

### Lenguajes y herramientas de IaC

#### Tipos de secuencias de comandos

Estos son scripts en Bash, PowerShell u otros que utilizan los diferentes clientes (SDK) proporcionados por el proveedor de la nube; por ejemplo, puede crear secuencias de comandos para el aprovisionamiento de una infraestructura de Azure con la CLI de Azure o Azure PowerShell.

El problema con estos lenguajes y herramientas es que requieren muchas líneas de código. Esto es porque necesitamos gestionar los diferentes estados de los recursos manipulados, y es necesario escribir todos los pasos de creación o actualizar la infraestructura deseada.

Sin embargo, estos lenguajes y herramientas pueden ser muy útiles para tareas que automatizan acciones repetitivas a realizarse sobre una lista de recursos (selección y consulta), o que requieren procesamiento complejo con cierta lógica a realizarse sobre recursos de infraestructura, como un script que automatiza la eliminación de máquinas virtuales que llevan una determinada etiqueta.

#### Tipos declarativos

Estos son lenguajes en las que basta con escribir el estado del sistema o infraestructura deseado en forma de configuración y propiedades. Este es el caso, por ejemplo, de Terraform y Vagrant de HashiCorp, Ansible, la plantilla Azure ARM, Azure Bicep (https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/bicep -descripción general), PowerShell DSC, Puppet y Chef. Todo lo que el usuario tiene que hacer es escribir el estado final de la infraestructura deseada; la herramienta se encargará de aplicarlo.

## La topología IaC

En una infraestructura en la nube, IaC se divide en varias tipologías:

* **Implementar y aprovisionar la infraestructura**
* **Configuración y plantillas del servidor**
* **Contenedorización**
* **Configuración y despliegue en Kubernetes.**

### Implementar y aprovisionar la infraestructura

**El aprovisionamiento es el acto de crear instancias de los recursos que componen la infraestructura**. Éstos pueden ser de plataforma como servicio (PaaS) y tipos de recursos sin servidor, como una aplicación web, una función de Azure o Event Hub, pero también toda la parte de la red que se gestiona, como VNet, subredes, tablas de enrutamiento o Azure Firewall. Para los recursos de máquinas virtuales, el paso de aprovisionamiento solo crea o actualiza el recurso de nube de la máquina virtual, pero no su contenido.

Existen diferentes herramientas de aprovisionamiento que podemos utilizar para esto, como Terraform, la plantilla ARM, la capacitación en la nube de AWS, la CLI de Azure, Azure PowerShell y también Google Cloud Deployment Manager. Por supuesto, hay muchos más, pero es difícil mencionarlos a todos.

### Configuración del servidor

*Este paso se refiere a la configuración de máquinas virtuales, como el refuerzo, los directorios, el montaje del disco, la configuración de la red (firewall, proxy, etc.) y la instalación del middleware.*

Hay diferentes herramientas de configuración, como Ansible, PowerShell DSC, Chef, Puppet y SaltStack.

*Para optimizar los tiempos de aprovisionamiento y configuración de los servidores, también es posible crear y utilizar modelos de servidores, también llamados imágenes, que contienen toda la configuración (hardening, middleware, etc.) de los servidores. Mientras aprovisionamos el servidor, indicaremos la plantilla a utilizar. Así, en unos minutos tendremos un servidor configurado y listo para ser utilizado.*

También existen muchas herramientas IaC para crear plantillas de servidor, como Aminator (utilizado por Netflix) y HashiCorp Packer.

### Infraestructura inmutable con contenedores

**La contenerización consiste en implementar aplicaciones en contenedores en lugar de implementarlas en máquinas virtuales.**

Hoy en día está muy claro que la tecnología de contenedor a utilizar es Docker y que una imagen de Docker se configura con código en un Dockerfile . Este archivo contiene la declaración de la imagen base, que representa el sistema operativo que se utilizará, el middleware adicional que se instalará en la imagen, solo los archivos y binarios necesarios para la aplicación y la configuración de red de los puertos. A diferencia de las máquinas virtuales, se dice que los contenedores son inmutables*; la configuración de un contenedor no se puede modificar durante su ejecución.*

### Configuración y despliegue en Kubernetes

Kubernetes es un orquestador de contenedores: es la tecnología que más encarna IaC porque de la forma en que implementa contenedores, la arquitectura de red (equilibrador de carga, puertos, etc.) y la administración de volúmenes, así como también cómo protege la información confidencial, todo lo cual se describe en los archivos de especificación YAML.

IaC, al igual que el desarrollo de software, requiere que implementemos prácticas y procesos que permitan que el código de la infraestructura evolucione y se mantenga.

Entre estas prácticas se encuentran las de desarrollo de software, así:

* Tener buenos principios de nomenclatura.
* No sobrecargues el código con comentarios innecesarios.
* Utilice funciones pequeñas.
* Implementar manejo de errores

Sin embargo, hay prácticas más específicas que merecen más atención:

* **Todo debe estar automatizado en el código** : Al realizar IaC es necesario codificar y automatizar todos los pasos de aprovisionamiento y no dejar fuera del código los pasos manuales que distorsionan la automatización de la infraestructura, lo que puede generar errores. Y si es necesario, no dudar en utilizar varias herramientas como Terraform y Bash con los scripts de la CLI de Azure.
* **El código debe estar en un administrador de control de fuente**: el código de infraestructura también debe estar en un SCM para ser versionado, rastreado, fusionado y restaurado y, por lo tanto, tener una mejor visibilidad del código entre Dev y Ops.
* **El código de infraestructura debe estar con el código de la aplicación**: en algunos casos, esto puede resultar difícil, pero si es posible, es mucho mejor colocar el código de infraestructura en el mismo repositorio que el código de la aplicación. Esto es para garantizar que tengamos una mejor organización del trabajo entre desarrolladores y operaciones, quienes compartirán el mismo espacio de trabajo.
* **Separación de roles y directorios** : Es bueno separar el código de la infraestructura según el rol del código. Esto le permite crear un directorio para aprovisionar y configurar máquinas virtuales y otro que contendrá el código para probar la integración de la infraestructura completa.
* **Integración en un proceso CI/CD**: Uno de los objetivos de IaC es poder automatizar el despliegue de la infraestructura. Entonces, desde el inicio de su implementación, es necesario configurar un proceso CI/CD que integre el código, lo pruebe y lo implemente en diferentes entornos. Algunas herramientas, como Terratest, le permiten escribir pruebas en código de infraestructura. Una de las mejores prácticas es integrar el proceso CI/CD de la infraestructura en el mismo proceso que la aplicación.
* **El código debe ser idempotente**: la ejecución del código de implementación de infraestructura debe ser idempotente; es decir, debería ser ejecutable automáticamente a voluntad. Esto significa que los guiones deben tener en cuenta el estado de la infraestructura al ejecutarla y no generar un error si el recurso a crear ya existe, o si un recurso a eliminar ya ha sido eliminado. El código de la infraestructura, una vez completamente automatizado, debe permitir que la infraestructura de la aplicación sea construida y destruida.
* **Para ser utilizado como documentación:** El código de la infraestructura debe ser claro y debe poder servir como documentación. La documentación de la infraestructura lleva mucho tiempo redactarse y, en muchos casos, no se actualiza a medida que la infraestructura evoluciona.
* **El código debe ser modular:** en infraestructura, los componentes suelen tener el mismo código; la única diferencia es el valor de sus propiedades. Además, estos componentes se utilizan varias veces en las aplicaciones de la empresa. Por lo tanto, es importante optimizar los tiempos de escritura del código factorizándolo con módulos (o roles, para Ansible) que serán llamados como funciones. Otra ventaja de utilizar módulos es la capacidad de estandarizar la nomenclatura de recursos y el cumplimiento de algunas propiedades.
* **Tener un entorno de desarrollo:** El problema de IaC es que es difícil probar su código de infraestructura en desarrollo en entornos que se utilizan para la integración, así como probar la aplicación, porque cambiar la infraestructura puede tener un impacto. Por lo tanto, es importante tener un entorno de desarrollo, incluso para IaC, que pueda verse afectado o incluso destruido en cualquier momento.

Por supuesto, la lista completa de buenas prácticas es más larga que ésta; todos los métodos y procesos de las prácticas de ingeniería de software también son aplicables.

Por lo tanto, IaC, al igual que los procesos CI/CD, es una práctica clave de la cultura DevOps que permite implementar y configurar una infraestructura escribiendo código. Sin embargo, la IaC sólo puede ser eficaz con el uso de herramientas adecuadas y la implementación de buenas prácticas.

<https://learn.microsoft.com/en-us/devops/>

<https://www.puppet.com/resources/history-of-devops-reports>

# La forma moderna de DevOps

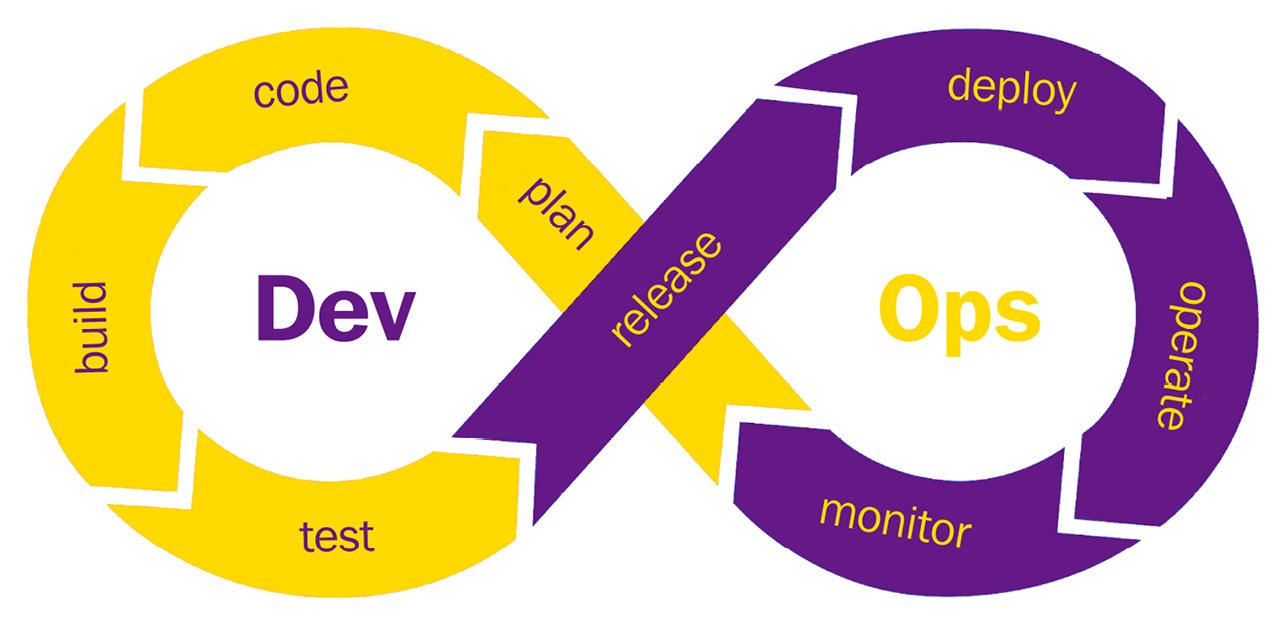
(cap. 1 de Modern DevOps Practices - Second Edition)

## ¿Qué es DevOps?

**Es un conjunto de principios y prácticas, así como una filosofía, que fomentan la participación de los equipos de desarrollo y operaciones en todo el ciclo de vida del desarrollo de software, incluido el mantenimiento y las operaciones del software.** Para implementar esto, las organizaciones administran varios procesos y herramientas que ayudan a automatizar el proceso de entrega de software para mejorar la velocidad y agilidad, reducir el tiempo del ciclo de liberación de código a través de canalizaciones de integración y entrega continuas (CI/CD) y monitorear las aplicaciones que se ejecutan en producción.

*Un equipo de DevOps debe asegurarse de que, en lugar de tener un conjunto claro de grupos aislados que se ocupan del desarrollo, las operaciones y el control de calidad, tengan un solo equipo que se encargue de todo el ciclo de vida del SDLC, es decir, el equipo creará, implementará y monitoreará el software*. El equipo combinado posee toda la aplicación en lugar de determinadas funciones. Eso no significa que la gente no tenga especialidades, pero la idea es asegurar que los desarrolladores sepan algo sobre operaciones y que los ingenieros de operaciones sepan algo sobre desarrollo. El equipo de control de calidad trabaja mano a mano con desarrolladores e ingenieros de operaciones para comprender los requisitos comerciales y los diversos problemas que enfrentan en el campo. Con base en estos aprendizajes, deben asegurarse de que el producto que están desarrollando cumpla con los requisitos comerciales y resuelva los problemas encontrados en el campo.

En un equipo de desarrollo tradicional, la fuente del trabajo pendiente es la empresa y sus arquitectos. Sin embargo, para un equipo de DevOps, hay dos fuentes de su trabajo pendiente diario: la empresa y sus arquitectos y los clientes y los problemas que enfrentan mientras operan su aplicación en producción; por lo tanto, en lugar de seguir un camino lineal de entrega de software, las prácticas de DevOps generalmente siguen un bucle infinito, como se muestra en la siguiente figura:



Para garantizar una interoperabilidad fluida entre personas con diferentes habilidades, DevOps se centra en gran medida en la automatización y las herramientas. DevOps tiene como objetivo garantizar que intentemos automatizar tareas repetibles tanto como sea posible y centrarnos en cosas más importantes. Esto garantiza la calidad del producto y una entrega rápida. DevOps se centra en las personas, los procesos y las herramientas, dando la mayor importancia a las personas y la menor a las herramientas. Generalmente utilizamos herramientas para automatizar procesos que ayudan a las personas a alcanzar los objetivos correctos.

### Integración continua (CI)

*CI es una práctica de desarrollo que implica fusionar con frecuencia cambios de código de varios desarrolladores en un repositorio compartido*, generalmente varias veces al día. Esto garantiza que sus desarrolladores combinen código periódicamente en un repositorio central donde se ejecutan compilaciones y pruebas automatizadas para proporcionar comentarios en tiempo real al equipo. Esto reduce significativamente el tiempo del ciclo y mejora la calidad del código. Este proceso tiene como objetivo minimizar los errores dentro del código al principio del ciclo en lugar de más tarde durante las fases de prueba. Detecta problemas de integración tempranamente y garantiza que el software siempre permanezca en un estado liberable.

### Entrega continua ( CD )

*El CD se trata de enviar su software probado en su entorno de producción cuando esté listo*, por lo tanto, una canalización de CD incorporará sus cambios en paquetes y ejecutará pruebas de integración y del sistema en ellos. Una vez que haya probado exhaustivamente su código, podrá implementar cambios automáticamente (o previa aprobación) en sus entornos de prueba y producción; por lo tanto*, CD pretende tener el último conjunto de artefactos probados listos para implementar.*

### Infraestructura como código (IaC)

*IaC es una práctica en software de desarrollo que implica administrar y aprovisionar recursos de infraestructura, como servidores, redes y almacenamiento, utilizando código y archivos de configuración en lugar de procesos manuales.* IaC trata la infraestructura como software, lo que permite a los equipos definir y gestionar los recursos de la infraestructura de una manera programable y controlada por versiones. Con la llegada de las máquinas virtuales, los contenedores y la nube, la infraestructura tecnológica se ha vuelto virtual en gran medida. Esto significa que podemos construir infraestructura a través de plantillas y llamadas API. Con herramientas modernas, también podemos construir infraestructura en la nube de forma declarativa. Esto significa que ahora puede crear IaC, almacenar el código necesario para construir la infraestructura dentro de un repositorio de código fuente como Git y utilizar una canalización de CI/CD para girar y administrar la infraestructura.

### Configuración como código (CaC)

*CaC es una práctica en desarrollo de software y administración de sistemas que implica gestionar y aprovisionar ajustes de configuración utilizando códigos y sistemas de control de versiones. Trata los ajustes de configuración como artefactos de código, lo que permite a los equipos definir, almacenar y gestionar la configuración de forma programática y reproducible. Históricamente, los servidores solían construirse manualmente desde cero y rara vez se modificaban. Sin embargo, con una infraestructura elástica implementada y un énfasis en la automatización, la configuración también se puede gestionar mediante código. CaC va de la mano con IaC para crear una infraestructura escalable y tolerante a fallas para que su aplicación pueda ejecutarse sin problemas.*

### Monitoreo y registro

*Supervisión y el registro son prácticas esenciales en el desarrollo y las operaciones de software que implican capturar y analizar datos sobre el comportamiento y el rendimiento de las aplicaciones y sistemas de software.* Proporcionan información sobre el estado, la disponibilidad y el rendimiento del software, lo que permite a los equipos identificar problemas, solucionarlos y tomar decisiones informadas para mejorar. El monitoreo y el registro se incluyen en la observabilidad, que es un área crucial para cualquier equipo de DevOps, es decir, saber cuándo su aplicación tiene problemas y excepciones mediante el monitoreo y clasificarlos mediante el registro.

### Comunicación y colaboración

Comunicación y colaboración son aspectos cruciales de las prácticas de DevOps. Promueven el trabajo en equipo eficaz, el intercambio de conocimientos y los flujos de trabajo optimizados en todo el desarrollo, las operaciones y otras partes interesadas involucradas en el ciclo de vida de la entrega del software. La comunicación y la colaboración hacen que un equipo de DevOps funcione bien. Atrás quedaron los días en que la comunicación se realizaba a través de correos electrónicos. En cambio, los equipos de DevOps modernos gestionan su trabajo pendiente mediante tickets y herramientas ágiles, realizan un seguimiento de sus artículos de conocimiento y otros documentos usando un wiki, y se comunican instantáneamente usando herramientas de chat y mensajería instantánea (IM).

### Comprender las aplicaciones modernas nativas de la nube

Cuando decimos nativas de la nube, hablamos de aplicaciones creadas para ejecutarse de forma nativa en la nube. **Una aplicación nativa de la nube está diseñada para ejecutarse en la nube aprovechando al máximo las capacidades y beneficios de la nube utilizando los servicios de la nube tanto como sea posible.**

Estas aplicaciones son inherentemente escalables, flexibles y resistentes (tolerantes a fallos). Dependen en gran medida de los servicios en la nube y la automatización.

Algunas de las características de una aplicación moderna nativa de la nube son las siguientes:

* **Arquitectura de microservicios** : las aplicaciones modernas nativas de la nube suelen seguir la arquitectura de microservicios. **Los microservicios son aplicaciones que se dividen en múltiples partes más pequeñas y poco acopladas con funciones empresariales independientes.** Los microservicios independientes se pueden escribir en diferentes lenguajes de programación según la necesidad o la funcionalidad específica. Estas piezas más pequeñas pueden escalarse de forma independiente, su funcionamiento es flexible y su diseño es resistente.
* Contenedorización: Las aplicaciones suelen utilizar contenedores para ejecutarse. Los contenedores proporcionan un entorno consistente, portátil y liviano para que se ejecuten las aplicaciones, asegurando que tengan todas las dependencias y configuraciones necesarias agrupadas. Los contenedores pueden ejecutarse igual en todos los entornos y plataformas en la nube.
* **DevOps y automatización**: las aplicaciones nativas de la nube utilizan en gran medida prácticas y herramientas modernas de DevOps y por lo tanto, dependen en gran medida de la automatización. Esto agiliza el desarrollo, las pruebas y las operaciones de su aplicación. La automatización también produce escalabilidad, resiliencia y consistencia.
* **Orquestación dinámica:** las aplicaciones nativas de la nube están diseñadas para escalar y están inherentemente destinadas a ser tolerante a fallas. Estas aplicaciones suelen ser efímeras (transitorias); por lo tanto, las réplicas de los servicios pueden aparecer y desaparecer según sea necesario. Para gestionar estos servicios se utilizan plataformas de orquestación dinámica como Kubernetes y Docker Swarm. Estas herramientas ayudan a ejecutar su aplicación bajo demandas y patrones de tráfico cambiantes.
* **Uso de servicios de datos nativos de la nube**: las aplicaciones nativas de la nube generalmente utilizan servicios de datos administrados en la nube, como almacenamiento, bases de datos, almacenamiento en caché y sistemas de mensajería para permitir comunicación entre múltiples servicios.

## DevOps moderno versus DevOps tradicional

El tradicional enfoque de DevOps implicó establecer un equipo DevOps formado por miembros de desarrollo, control de calidad y operaciones y que trabaja para crear un mejor software más rápido. Sin embargo, si bien habría un enfoque en la automatización de la entrega de software, herramientas de automatización como Jenkins, Git y otros fueron instalados y mantenidos manualmente. Esto generó otro problema, ya que ahora teníamos que gestionar otro conjunto de infraestructura de TI. Finalmente todo se redujo a infraestructura y configuración, y la atención se centró en automatizar el proceso de automatización.

Con la llegada de los contenedores y el reciente auge en el panorama de la nube pública, el enfoque moderno de DevOps entró en escena, lo que implicaba automatizar todo. Desde el aprovisionamiento de infraestructura hasta la configuración de herramientas y procesos, hay código para todo. Entonces, ahora tenemos IaC , CaC , infraestructura inmutable y contenedores .

Es importante señalar que la distinción entre DevOps moderno y DevOps tradicional no es estrictamente binaria, ya que las organizaciones pueden adoptar diversas prácticas y tecnologías a lo largo de un espectro. El enfoque moderno de DevOps generalmente se centra en aprovechar las tecnologías de la nube, la automatización, la contenedorización y los principios de DevSecOps para mejorar la colaboración, la agilidad y la eficiencia del desarrollo y la implementación de software.

## La necesidad de contenedores

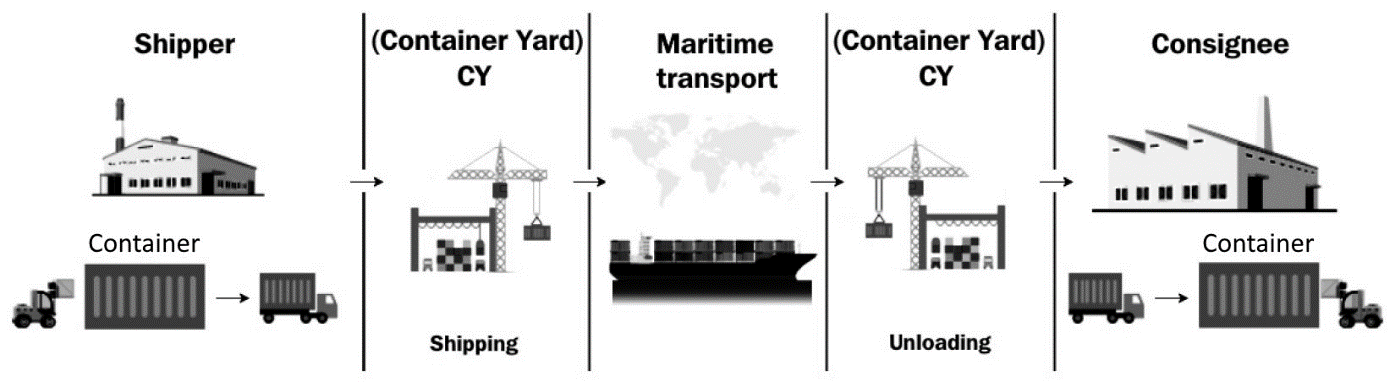
Los contenedores están de moda últimamente y por una excelente razón. Resuelven el problema más crítico de la arquitectura informática: **ejecutar software distribuido y confiable con una escalabilidad casi infinita en cualquier entorno informático.**

Han permitido una disciplina completamente nueva en ingeniería de software: *los microservicios*. También han introducido el concepto de paquete una vez implementado en cualquier lugar en tecnología. Combinados con la nube y las aplicaciones distribuidas, los contenedores con tecnología de orquestación de contenedores han dado lugar a una nueva palabra de moda en la industria (nativa de la nube) que cambia el ecosistema de TI como nunca antes.

Los contenedores derivan su sustantivo por el “envíoainers”. Explicaré los contenedores utilizando una analogía con los contenedores de envío para una mejor comprensión. Históricamente, debido a las mejoras en el transporte, muchas cosas se movían a través de múltiples geografías. Dado que se transportaban diversas mercancías en diferentes modos, la carga y descarga de mercancías era un problema enorme en cada punto de transporte. Además, con el aumento de los costos laborales, no era práctico para las compañías navieras operar a gran escala manteniendo los precios bajos.

Además, provocaba daños frecuentes a los artículos y las mercancías solían extraviarse o mezclarse con otros envíos porque no había aislamiento. Era necesario contar con una forma estándar de transporte de mercancías que proporcionara el aislamiento necesario entre envíos y permitiera una fácil carga y descarga de mercancías. La industria naviera ideó los contenedores como una solución elegante a este problema.

Ahora, los contenedores de envío han simplificado muchas cosas en la industria del transporte marítimo. Con un contenedor estándar podemos enviar mercancías de un lugar a otro con sólo mover el contenedor. El mismo contenedor puede utilizarse en carreteras, cargarse en trenes y transportarse en barcos. Los operadores de estos vehículos no necesitan preocuparse por lo que hay dentro del contenedor la mayor parte del tiempo. La siguiente figura muestra gráficamente todo el flujo de trabajo para facilitar la comprensión:



De manera similar, ha habido problemas con la portabilidad del software y la gestión de recursos informáticos en la industria del software. En un ciclo de vida de desarrollo de software estándar, una pieza de software se mueve a través de múltiples entornos y, a veces, numerosas aplicaciones comparten el mismo sistema operativo. Puede haber diferencias en la configuración entre entornos, por lo que el software que pudo haber funcionado en un entorno de desarrollo puede no funcionar en un entorno de prueba. Algo que funcionó en la prueba puede que tampoco funcione en producción.

Además, cuando tienes varias aplicaciones ejecutándose en una sola máquina, no hay aislamiento entre ellas. Una aplicación puede agotar los recursos informáticos de otra y eso puede provocar problemas de tiempo de ejecución.

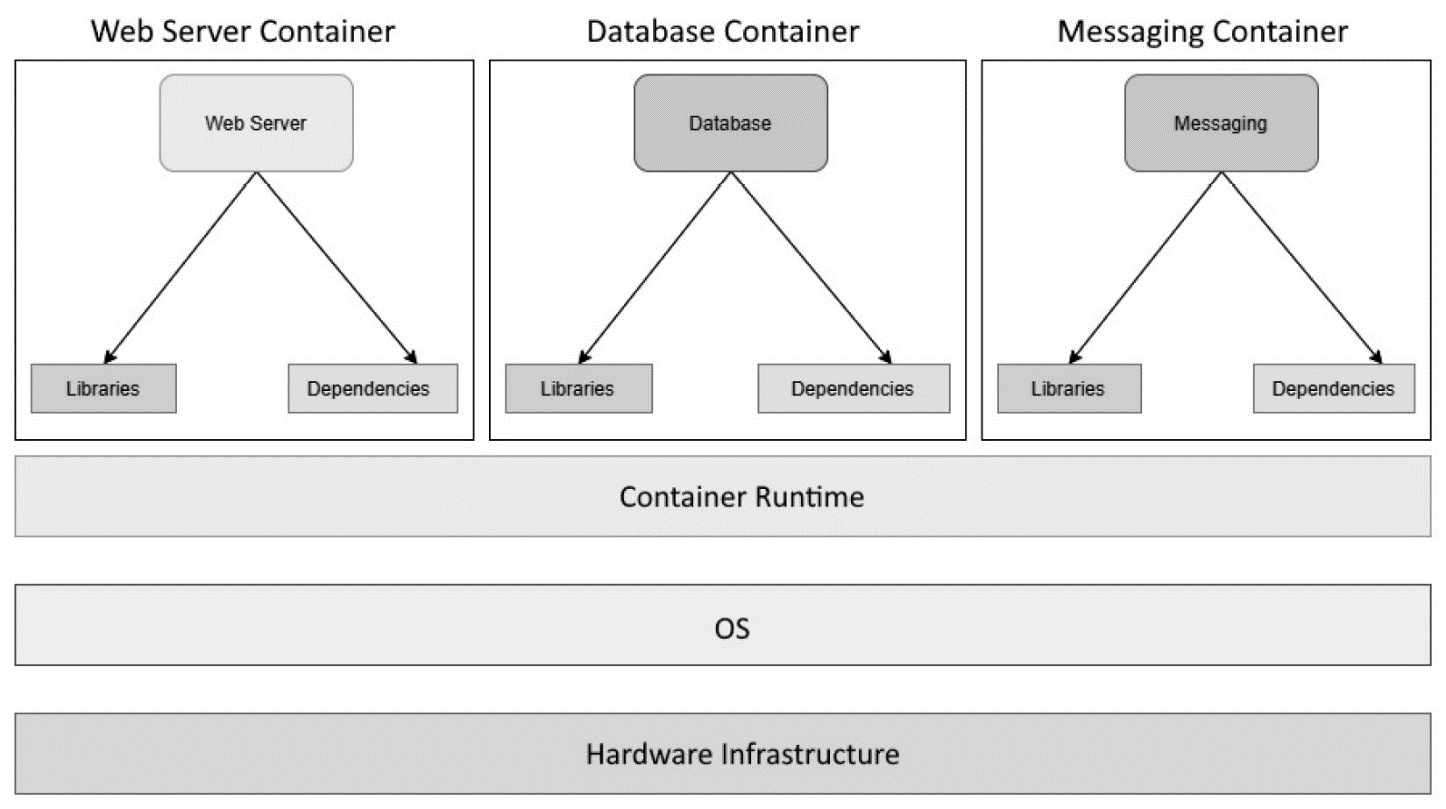
Es necesario volver a empaquetar y reconfigurar las aplicaciones en cada paso de la implementación, por lo que requiere mucho tiempo y esfuerzo y, a veces, es propenso a errores.

*En la industria del software, los contenedores resuelven estos problemas al proporcionar aislamiento entre la aplicación y la gestión de recursos informáticos, lo que proporciona una solución óptima a estos problemas.*

El mayor desafío de la industria es proporcionar aislamiento de aplicaciones y administrar dependencias externas con elegancia para que puedan ejecutarse en cualquier plataforma, independientemente del sistema operativo o la infraestructura. El software está escrito en numerosos lenguajes de programación y utiliza varias dependencias y marcos. Esto lleva a un escenario llamado la matriz del infierno.

### Arquitectura de contenedores

En la mayoría de los casos, puedes visualizar contenedores como mini máquinas virtuales, al menos así lo parecen; pero, en realidad, *son sólo programas informáticos que se ejecutan dentro de un sistema operativo.*

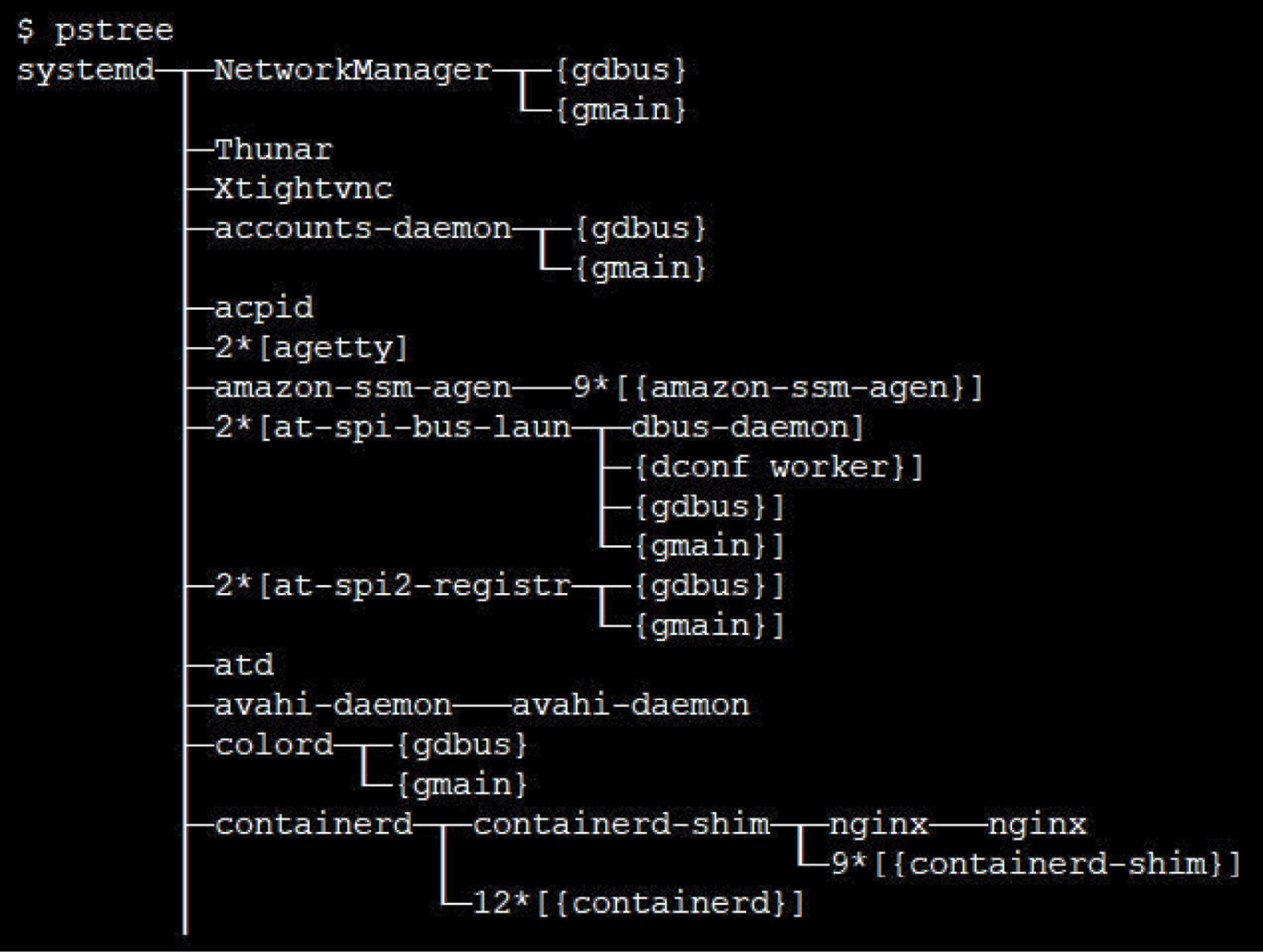


Como podemos ver, tenemos la infraestructura informática justo en la parte inferior, formando la base, seguida por el sistema operativo host y un tiempo de ejecución del contenedor (en este caso, Docker) que se ejecuta encima. Luego tenemos múltiples aplicaciones en contenedores que utilizan el tiempo de ejecución del contenedor, ejecutándose como procesos separados en el sistema operativo host usando *espacios de nombres y cgroups .*

Como habrás notado, no tenemos una capa de sistema operativo invitado, que es algo que tenemos con las máquinas virtuales. *Cada contenedor es un programa de software que se ejecuta en el espacio de usuario del Kernel y comparte el mismo sistema operativo y el tiempo de ejecución asociado y otras dependencias, con solo las bibliotecas y dependencias necesarias dentro del contenedor*. *Los contenedores no heredan las variables de entorno del sistema operativo. Tienes que configurarlos por separado para cada contenedor.*

*Los contenedores replican el sistema de archivos y, aunque están presentes en el disco, están aislados de otros contenedores. Esto hace que los contenedores ejecuten aplicaciones en un entorno seguro. Un sistema de archivos de contenedor separado significa que los contenedores no tienen que comunicarse de un lado a otro con el sistema de archivos del sistema operativo, lo que resulta en una ejecución más rápida que las máquinas virtuales.*

*Los contenedores fueron diseñados para usar espacios de nombres de Linux para proporcionar aislamiento y cgroups para ofrecer restricciones en el consumo de CPU, memoria y E/S de disco;* esto significa que si enumera los procesos del sistema operativo, verá el proceso contenedor ejecutándose junto con otros procesos, como se muestra a continuación:



### Redes de contenedores

**Los usuarios son entidades de red separadas dentro del sistema operativo**. Los tiempos de ejecución de Docker utilizan controladores de red para definir la red entre contenedores y son redes definidas por software.Los trabajos en red de contenedores se realizan mediante el uso de software para manipular las *iptables del host , conectarse con interfaces de red externas, crear redes de túneles y realizar otras actividades para permitir conexiones hacia y desde contenedores.*

Si bien existen varios tipos de configuraciones de red que puede implementar con contenedores, es bueno conocer algunas de las más utilizadas

**None :** esto es una red completamente aislada y sus contenedores no pueden comunicarse con el mundo externo. Se les asigna una interfaz loopback y no pueden conectarse con una interfaz de red externa. Puede utilizar esta red para probar sus contenedores, preparar su contenedor para uso futuro o ejecutar un contenedor que no requiera ninguna conexión externa, como el procesamiento por lotes.

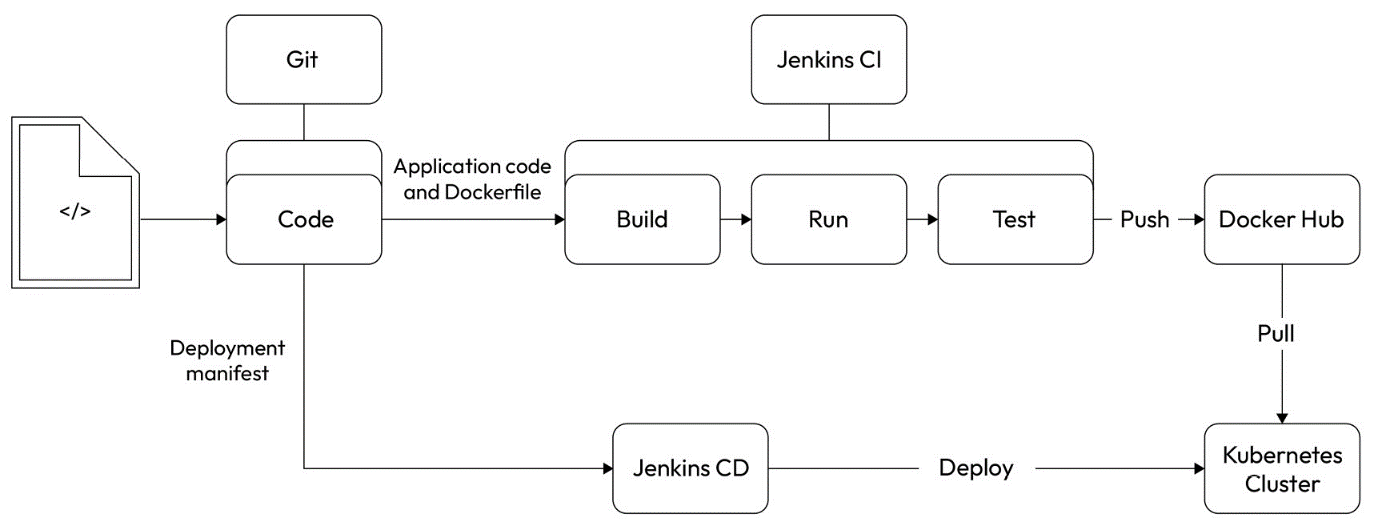
**Puente**: La red puente es el tipo de red predeterminado en la mayoría de los tiempos de ejecución de contenedores, incluido Docker, y utiliza la interfaz docker0 para los contenedores predeterminados. La red puente manipular I*Ptables* para proporcionar traducción de direcciones de red (NAT) entre el contenedor y la red host, permitiendo la conectividad de la red externa. Tampoco genera conflictos de puertos, lo que permite el aislamiento de la red entre contenedores que se ejecutan en un host, por lo tanto, puede ejecutar varias aplicaciones que utilicen el mismo puerto de contenedor dentro de un único host. Una red puente permite que los contenedores dentro de un único host se comuniquen utilizando las direcciones IP del contenedor. Sin embargo, no permiten la comunicación con contenedores que se ejecutan en un host diferente. Por lo tanto, no debe utilizar la red puente para configuración de clústeres (usando múltiples servidores en conjunto para ejecutar sus contenedores).

**Anfitrión**: La red utiliza el espacio de nombres de red de la máquina host para todos los contenedores. Es similar a ejecutar múltiples aplicaciones dentro de su host. Si bien una red host es fácil de implementar, visualizar y solucionar problemas, es propensa a problemas de conflictos de puertos. Si bien los contenedores utilizan la red host para todas las comunicaciones, no tienen el poder de manipular las interfaces de la red host a menos que se ejecuten en modo privilegiado. La red de host no utiliza NAT, por lo que es rápida y se comunica a velocidades básicas.; por lo tanto, puede utilizar la red de host para optimizar el rendimiento. Sin embargo, dado que no tiene aislamiento de red entre contenedores, desde el punto de vista de la seguridad y la administración, en la mayoría de los casos, debe evitar el uso de la red host.

**Base**: Expone las interfaces de red del host directamente a los contenedores. Esto significa que puede ejecutar sus contenedores directamente en las interfaces de red en lugar de utilizar una red puente. Hay varias redes subyacentes, siendo las más notables *MACvlan e IPvlan*. MACvlan le permite asignar una dirección MAC a cada contenedor para que su contenedor parezca un dispositivo físico. Esto es beneficioso para migrar su pila existente a contenedores, especialmente cuando su aplicación necesita ejecutarse en una máquina física. MACvlan también proporciona un aislamiento completo de la red de su host, por lo que puede utilizar este modo si tiene un requisito de seguridad estricto. MACvlan tiene limitaciones ya que no puede funcionar con conmutadores de red con una política de seguridad que no permita la suplantación de MAC. También está limitado al límite de direcciones MAC de algunas tarjetas de interfaz de red, como Broadcom, que solo permite 512 direcciones MAC por interfaz.

**Superposición**: no confundas overlay con underlay: aunque parezcan antónimos, no lo son. Las redes superpuestas permiten la comunicación entre contenedores en diferentes máquinas host a través de un túnel de red.; por lo tanto, desde la perspectiva de un contenedor, parecen interactuar con contenedores en un único host, incluso cuando están ubicados en otro lugar. Supera las limitaciones de la red puente y es especialmente útil para la configuración de clústeres, especialmente y cuando se utiliza un orquestador de contenedores como *Kubernetes o Docker Swarm*.

## Contenedores y prácticas modernas de DevOps



Los contenedores encajan muy bien en las prácticas modernas de CI/CD, ya que ahora dispone de una forma estándar de crear e implementar aplicaciones, independientemente del lenguaje en el que codifique. No tiene que gestionar software costoso de creación e implementación, ya que obtiene todo con contenedores.

Los contenedores rara vez se ejecutan por sí solos y es un estándar práctico en la industria para conectar a ellos un contenedor orquestador como Kubernetes o usar *una plataforma de contenedor como servicio CaaS)*como AWS ECS y EKS , Google Cloud Run y ​​Kubernetes Engine , Azure ACS y AKS , Oracle OCI y OKE , entre otros. Una plataforma popular de *función como servicio (FaaS )*como AWS Lambda , Google Functions , Azure Functions y Oracle Functions en la que también se ejecutan contenedores en el fondo; entonces, aunque ellos quizás tengan abstraído el mecanismo subyacente, es posible que ya esté utilizando contenedores sin saberlo.

Como los contenedores son livianos, puede crear partes más pequeñas de aplicaciones en contenedores para administrarlas de forma independiente. Combine eso con un orquestador de contenedores como Kubernetes y obtendrá una arquitectura de microservicios distribuidos que se ejecuta con facilidad. Estas piezas más pequeñas pueden luego escalarse, repararse automáticamente y liberarse independientemente de las demás, lo que significa que puede lanzarlas a producción más rápido que antes y de manera mucho más confiable.

También puedes conectar una *malla de servicios (componentes de infraestructura que le permiten descubrir, enumerar, administrar y permitir la comunicación entre múltiples componentes (servicios) de su aplicación de microservicios*) como Istio en la parte superior, y *obtendrá funciones avanzadas de Operaciones como gestión del tráfico, seguridad y observabilidad con facilidad. Entonces puedes hacer cosas interesantes como implementaciones blue/green y pruebas A/B , pruebas operativas en producción con espejo de tráfico , geolocalización-enrutamiento basado , y mucho más.*

Como resultado, las empresas grandes y pequeñas están adoptando contenedores más rápido que nunca y el campo está creciendo exponencialmente. Según businesswire.com , el mercado de contenedores de aplicaciones muestra un crecimiento compuesto del 31% anual y alcanzará los 6.900 millones de dólares en 2025. El crecimiento exponencial del 30,3% anual en la nube, que se espera alcance más de 2.400 millones de dólares en 2025, también ha contribuido a esto.

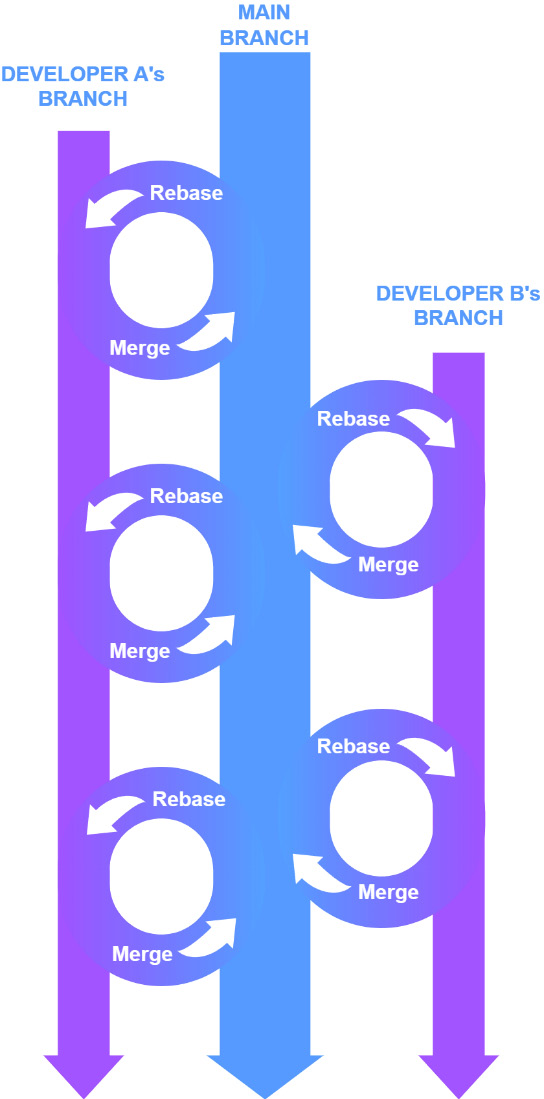
# El qué, cómo y por qué de la integración continua

(cap. 1 de Learning Continuous Integration with Jenkins - Third Edition)

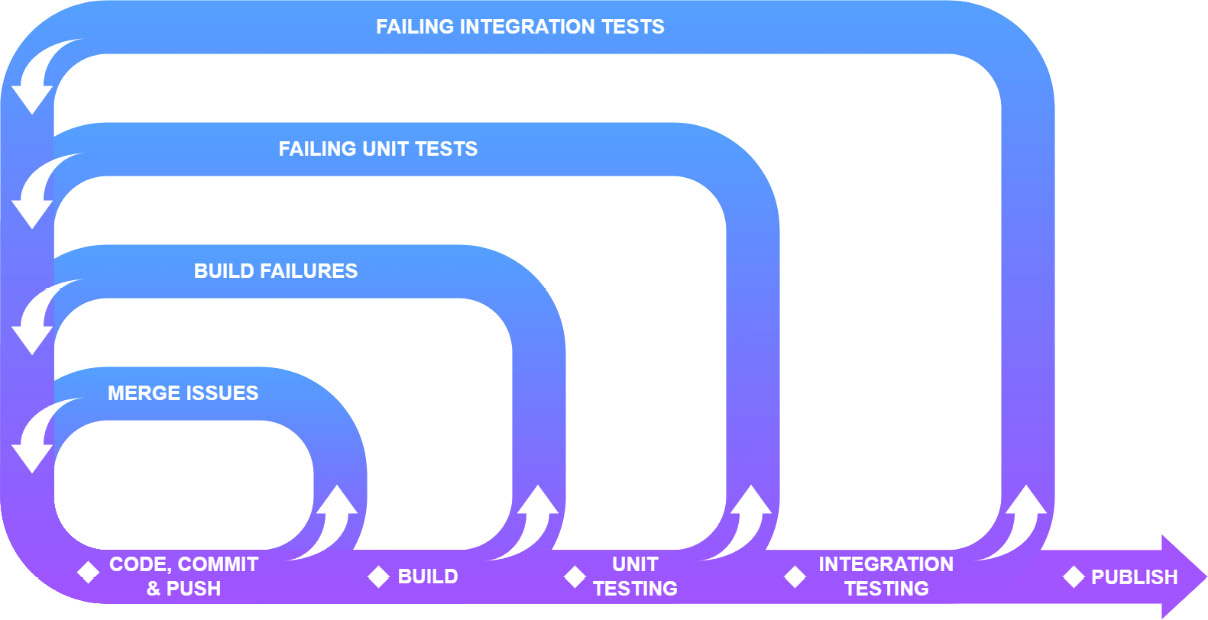
## ¿Qué es la integración continua?

*CI es una práctica de desarrollo de software en la que los desarrolladores integran periódicamente sus cambios de código en el área de trabajo compartida del proyecto e inspeccionan sus cambios en busca de problemas de compilación y otros problemas relacionados con la calidad.*

La integración es el acto de enviar su código modificado a la rama principal del proyecto (la posible solución de software). Esto se hace técnicamente fusionando su rama de funciones con la principal rama en un desarrollo basado en tronco (una estrategia de rama basada en Git) y, además, fusionando la rama de características en el desarrollo de la bifurcación en un desarrollo basado en Gitflow Workflow (otra estrategia de bifurcación basada en Git), como se muestra en la siguiente figura:



Con CI es necesario detectar y resolver los problemas encontrados durante y después de la integración lo antes posible en el ciclo. Esto se puede entender en la Figura siguiente, que muestra varios problemas encontrados durante un único ciclo de CI:



Cuando un desarrollador confirma y envía (fusiona) sus cambios de código a la rama remota, puede experimentar dificultades de fusión, si no errores de compilación, pruebas unitarias fallidas o pruebas de integración fallidas.

Puede surgir un problema de fusión si los desarrolladores no extraen (reorganizan) de forma rutinaria el código de la rama troncal remota en su rama local. De manera similar, es posible que el código no se pueda compilar debido a problemas de compilación, y que las pruebas fallidas indiquen un defecto**, lo cual es algo positivo.** Si ocurre alguno de estos problemas, el desarrollador debe ajustar el código para corregirlo.

## Principios clave de la integración continua

### Validar cada cambio de código

### Fallar rápido

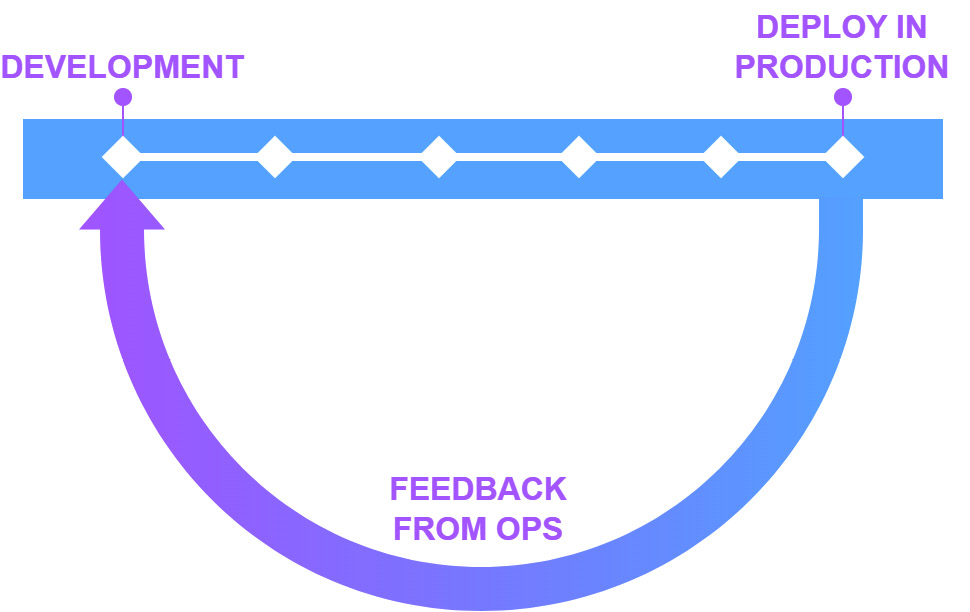
Entendamos la idea de fallar rápido de una manera un poco diferente. El concepto de fallo rápido defiende que todos losLas etapas de la canalización, como la validación de problemas de fusión, la verificación de problemas de compilación, las pruebas unitarias, las pruebas de integración, el análisis de código estático, etc., deben realizarse de forma secuencial para un cambio de código particular en lugar de simultáneamente. Esto se debe a que crear el código no tiene sentido si hay problemas de fusión; de manera similar, ejecutar pruebas unitarias no tiene sentido si la compilación falla, y así sucesivamente. En otras palabras, deténgase tan pronto como descubra que hay un problema e intente otra cosa.

### Automatización

*Los actos de automatización como catalizador. Desde encontrar fallas hasta publicar exitosamente su aplicación, la automatización lo acelera todo. Automatice todo lo que pueda, ya sea compilación de código, pruebas unitarias, pruebas de integración, análisis de código estático, empaquetado de la aplicación o servicio terminado, etc.*

### Retroalimentación continua

*El proceso de desarrollo de software es análogo a la línea de montaje de una fábrica*. La entrega de nuevas funciones y actualizaciones fluye de izquierda a derecha. El extremo izquierdo del proceso es donde se lleva a cabo el desarrollo y el extremo derecho es donde su aplicación opera en producción. Como se muestra en el siguiente diagrama, considere una línea de ensamblaje con un único circuito de retroalimentación que fluye desde el equipo de operaciones al equipo de desarrollo:



Debido a que en tal caso hay un único flujo de información que retrocede, los desarrolladores nunca sabrán si sus modificaciones han producido una aplicación que funciona bien hasta que se inicie la aplicación. Otra cosa a tener en cuenta es que el tiempo que tardan los comentarios en llegar a los desarrolladores es igual al tiempo que lleva llevar una función desde el desarrollo hasta la implementación.

Uno de esos métodos con ciclos de retroalimentación limitados es el modelo en cascada de desarrollo de software. En este modelo, la primera retroalimentación ocurre cuando se integra todo el código, la segunda cuando se prueba el código y la final cuando se pone en producción. No se puede hacer nada tangible rápidamente cuando el equipo de operaciones se queja de una falla en la producción cuyo código fue creado un año antes. Necesitamos comentarios más inmediatos.

Y ese es el objetivo de la retroalimentación continua: *tener cada vez más ciclos de retroalimentación en nuestro ciclo de vida de desarrollo de software que estén más cerca entre sí y sucedan más rápido.* Un ejemplo de ello es la práctica de revisión por pares, que a menudo ocurre durante todo el proceso de desarrollo. Durante la revisión por pares, un desarrollador senior evalúa las modificaciones propuestas para afirmar si los cambios funcionarán según lo previsto, como se muestra en la siguiente figura:

Las aportaciones continuas de todas las fases del ciclo de vida del desarrollo de software deben proporcionarse con la mayor claridad posible. Además, si su proceso de desarrollo de software está muy automatizado, ayudará aún más a lograr una retroalimentación continua.

## Cómo practicar la integración continua

### Usando una herramienta de control de versiones

Un sistema de control de versiones ayuda al equipo en el control de cambios en el código fuente del proyecto. Lo hace monitoreando cada cambio de código realizado en el código fuente, lo que da como resultado un historial detallado de *quién modificó qué* en un momento dado. Permite a los desarrolladores trabajar en sus propios espacios de trabajo, proporcionando aislamiento y, al mismo tiempo, ofreciendo un medio para la colaboración y la resolución de disputas. Lo más importante es que le ayuda a mantener una única fuente de verdad para su proyecto.

Idealmente, todo lo necesario para entregar la aplicación de software debería estar controlado por versiones y rastreado, incluidas las dependencias de software que se rastrean de una manera ligeramente diferente utilizando un administrador de repositorio binario (una herramienta DevOps para administrar todos los artefactos, binarios, archivos y contenedores). imágenes a lo largo del ciclo de vida de desarrollo de software de su aplicación). Incluso puedes ampliar esta idea de código de versiones para administrar y sincronizar configuraciones de infraestructura e implementaciones de aplicaciones. Esto es conocido como *GitOps .*

Una herramienta de control de versiones ofrece características como etiquetas para marcar hitos importantes en el ciclo de vida del desarrollo de software y ramas para permitir el desarrollo independiente que, en última instancia, se fusionará con el lugar de origen.

### Desarrollo paralelo

La ramificación permite el **desarrollo paralelo**. Eso permite a los desarrolladores en un equipo trabajar en versiones individuales del código al mismo tiempo. Para entender esto, considere un proyecto de desarrollo de software que tiene todo su código fuente almacenado en un repositorio Git con una única rama principal/maestra. Supongamos que es necesario ofrecer dos funciones nuevas y se eligen dos desarrolladores diferentes del equipo para trabajar en ellas de forma independiente, sin interferir con el trabajo de los demás. La solución es que usarían sucursales. Cada desarrollador crearía su propia rama y trabajaría en ella en soledad. Cuando han completado su trabajo y la canalización de CI que se ejecuta en sus ramas separadas está en verde, los cambios se fusionan nuevamente en la rama principal.

Aunque no existen restricciones sobre cómo se pueden utilizar las sucursales, se recomienda emplear ciertos métodos de sucursales estándar de la industria. Gitflow Workflow y, más recientemente, el flujo de trabajo basado en troncales son dos de esas estrategias de ramificación.

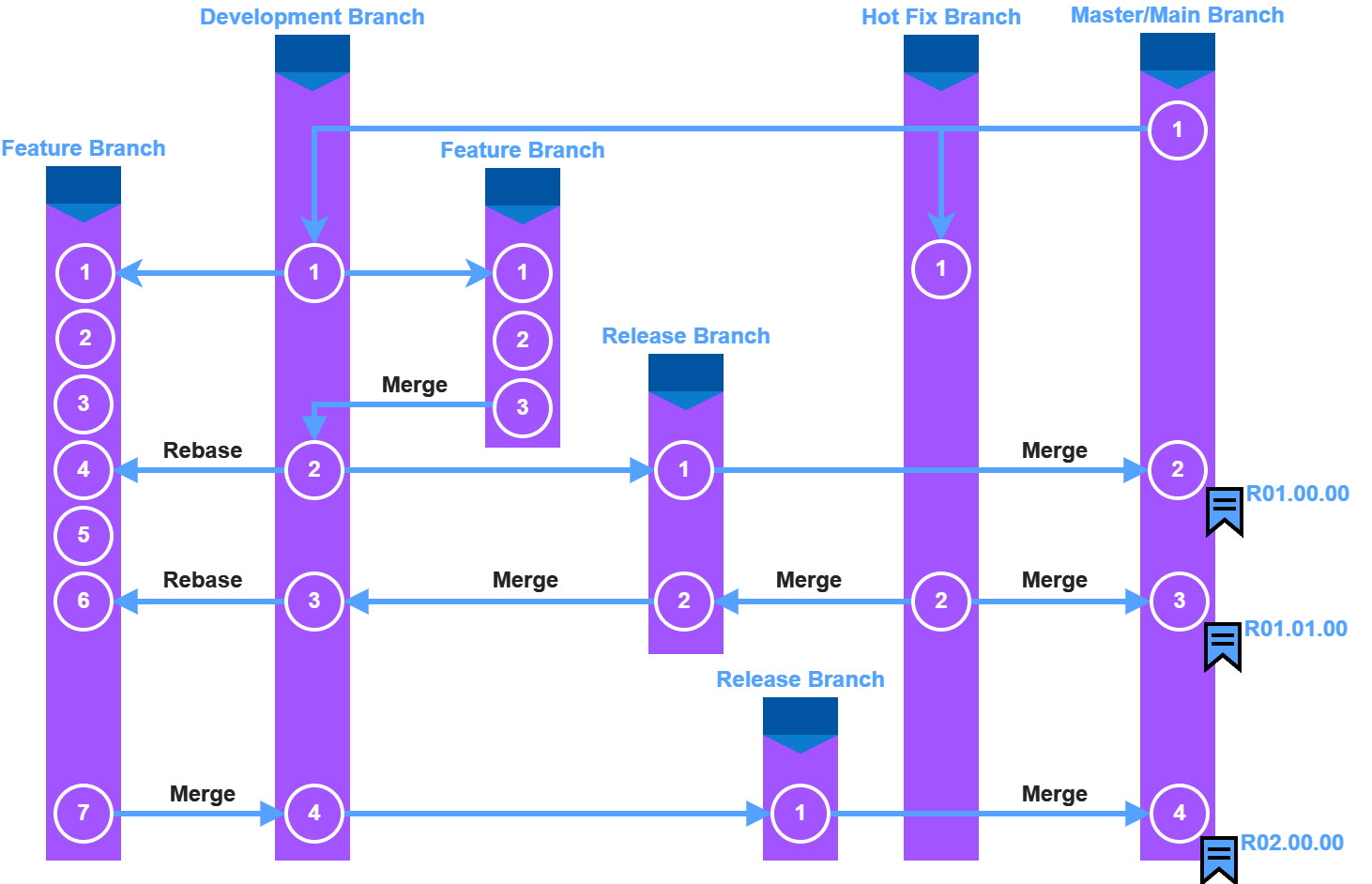
*Gitflow es un paradigma de desarrollo más restrictivo en el que no todos tienen el privilegio de realizar modificaciones en la fuente principal. Esto mantiene los errores al mínimo y la calidad del código al más alto nivel.*

Por otro lado, debido a que todos los desarrolladores tienen acceso al código principal, el desarrollo basado en troncales es una estrategia de ramificación más vulnerable. Sin embargo, permite a los equipos iterar rápidamente.

#### Flujo de trabajo de GitFlow

El flujo de trabajo de Gitflow es una metodología de ramificación que utiliza múltiples ramas para organizar el código fuente. En el flujo de trabajo que se muestra en el siguiente diagrama, la rama principal/maestra se mantiene limpia y solo contiene código liberable y listo para enviar. Todo el desarrollo se lleva a cabo en las ramas de funciones, y la rama de desarrollo actúa como una ubicación central para integrar todas las funciones.

La figura muestra un flujo de trabajo de Gitflow. Como puede ver, tenemos ramas de lanzamiento que se sacan de la rama de desarrollo cuando hay una liberación estable. Todas las correcciones de errores para una versión se realizan en la rama de versión. También hay una rama de revisión que se crea desde la rama principal/maestra cuando se requiere una revisión:



### Flujo de trabajo basado en troncales

La técnica de ramificación basada en troncales alienta el uso de la rama principal/maestra como ubicación única para rastrear todos los hitos importantes del ciclo de vida de desarrollo de software. Aún puedes crear pequeñas ramas de funciones temporales desde la rama principal/maestra, pero debes fusionarlas en la rama principal/maestra lo antes posible.

Alguien que ha estado en la industria de TI durante más de una década, le resultará divertido ver que el flujo de trabajo basado en troncales vuelve a ser popular. Cuando el control de versiones y la CI eran conceptos nuevos, el flujo de trabajo basado en troncales era lo que usaban la mayoría de los equipos; luego, Gitflow Workflow se hizo popular por buenas razones y permaneció así durante mucho tiempo, solo para ser derrocado nuevamente por el flujo de trabajo basado en troncales.

En modelo de bifurcación basado en troncales, solo se requiere una canalización de CI/CD que rastree la rama principal/maestra. Cuando se produce una confirmación, se activa la canalización de CI/CD, lo que da como resultado un candidato liberable. Cualquier confirmación con la rama main/master podría ser una posible versión candidata, una revisión o una nueva característica.

Debido a que todos pueden fusionar cambios en la rama principal/maestra, se requiere un procedimiento estricto de revisión del código. Esta metodología se adapta mejor a proyectos de software contemporáneos de alto rendimiento donde todo el proceso de CI/CD está automatizado de principio a fin. Los proyectos de desarrollo de software basados ​​en microservicios, en particular, pueden beneficiarse enormemente del flujo de trabajo basado en troncales, ya que tienen un proceso de CI/CD más rápido. Los proyectos basados ​​en monolíticos, por otro lado, se beneficiarían más de Gitflow Workflow.

Porque el modelo de ramificación carece de ramas de características; en su lugar, se pueden usar indicadores de características. Los indicadores de funciones, también conocidos como alternancia de funciones, son un enfoque de desarrollo de software que le permite publicar programas de software que tienen funciones inactivas que pueden activarse y desactivarse durante el tiempo de ejecución.

### Verificar cada confirmación de código para detectar errores de compilación

**El primer y más importante paso de cualquier proceso de CI es verificar si los cambios en el código son compilables**. Básicamente, intentamos ver si podemos generar un binario funcional que pueda probarse e implementarse. Si el código base está en un lenguaje interpretado, verificamos si todas las dependencias están presentes y empaquetadas juntas. No tiene sentido avanzar más en el proceso de CI si los cambios en el código no se compilan.

El proceso de CI recomienda compilar cada cambio de código, activar una notificación (bucle de retroalimentación) y fallar toda la ejecución del proceso en caso de error. Como tal, la duración de la ejecución de la construcción (tiempo de construcción) juega un factor crítico para determinar la efectividad del proceso de CI. Si el tiempo de construcción es largo (digamos, varias horas), resulta poco práctico construir cada cambio de código en una infraestructura limitada por el costo y la disponibilidad. La automatización y la CI no tienen valor si los procesos tardan horas en completarse.

*Modularizar su aplicación es una forma eficaz de contrarrestar las compilaciones de software que tardan más en construirse.*

### Realizar pruebas unitarias

Recuerda el Principio de retroalimentación continua: “tener cada vez más ciclos de retroalimentación en nuestro ciclo de vida de desarrollo de software que están más cerca y son más rápidos”. Las pruebas unitarias son una de esas retroalimentaciónes.

Sin pruebas unitarias, el siguiente nivel de pruebas es la etapa de pruebas de integración de larga duración, seguida de la etapa de control de calidad del proceso de entrega/implementación continua. Normalmente, su compilación tardará algún tiempo en llegar al equipo de ingeniería de control de calidad, quienes generalmente plantearán un defecto si identifican un problema. En resumen, si se utilizan pruebas unitarias, se puede evitar un ciclo de retroalimentación prolongado.

Las pruebas unitarias le permiten probar más rápido la parte comprobable más pequeña de su programa. Si su código fuente está bien escrito, las pruebas unitarias son fáciles de escribir y de resolver cuando fallan.

#### Cobertura de código

La cobertura de código es una métrica de CI importante y se define como la proporción de código cubierto por pruebas unitarias. Tener una puntuación alta de cobertura de código es esencial, ya que detectar y resolver defectos en el control de calidad o en la producción requiere mucho tiempo y es costoso.

Cualquier fragmento de código que carezca de una prueba unitaria coincidente es un posible riesgo; por lo tanto, es fundamental tener una mayor puntuación de cobertura de código. También vale la pena señalar que el porcentaje de cobertura del código no garantiza la funcionalidad del código cubierto por las pruebas unitarias.

El porcentaje de cobertura del código se calcula durante las pruebas unitarias. Casi todos los lenguajes de programación tienen una herramienta de cobertura. Sin embargo, la mayoría de los hallazgos se cargan en una herramienta de análisis de código estático como *SonarQube,* donde se muestran en un hermoso panel, junto con otros datos relacionados con la calidad del código.

### Análisis de código estático

El análisis de código estático, a veces conocido como prueba de caja blanca, es un tipo de prueba de software que busca características estructurales en el código. Determina cómo de resistente o mantenible es el código. El análisis de código estático se realiza sin ejecutar programas. Se diferencia de las pruebas funcionales, que exploran las características funcionales del software y son dinámicas.

El análisis de código estático es la evaluación de las estructuras internas del software. No cuestiona la funcionalidad del código. Una herramienta de análisis de código estático como SonarQube viene con un panel para mostrar varias métricas y estadísticas de cada ejecución del proceso de CI. El análisis de código estático se activa cada vez que se ejecuta una compilación después de la prueba unitaria. SonarQube admite muchos lenguajes, como Java, C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, etc.

#### Métricas de calidad

*Cuando se ejecuta análisis de código estático en su código fuente, la herramienta SonarQube genera un informe que contiene métricas importantes sobre la calidad de su código*. Estas métricas se agrupan en secciones explícitas: complejidad, duplicaciones, problemas, mantenibilidad, confiabilidad, seguridad, tamaño y pruebas. Cada informe de análisis de código estático se publica en su propio panel específico del proyecto en SonarQube, rama por rama; sin embargo, las mediciones se muestran colectivamente para el proyecto como una puntuación o un número, o mediante un gráfico de serie temporal.

La seguridad es otra sección de métrica de calidad clave. Incluye métricas como vulnerabilidades (el número total de problemas de seguridad descubiertos). *¿Recuerda los fallos de seguridad de Apache Log4j? Aquí es donde lo detectarías y lo detendrías antes de que avance más.*

#### Perfiles y reglas de calidad.

Los perfiles de calidad son colecciones de reglas que elije aplicar durante el análisis de código estático. Esto se puede hacer desde el panel de SonarQube. Los perfiles de calidad son específicos del lenguaje, por lo que normalmente verás más de un perfil de calidad asignado a un proyecto en SonarQube.

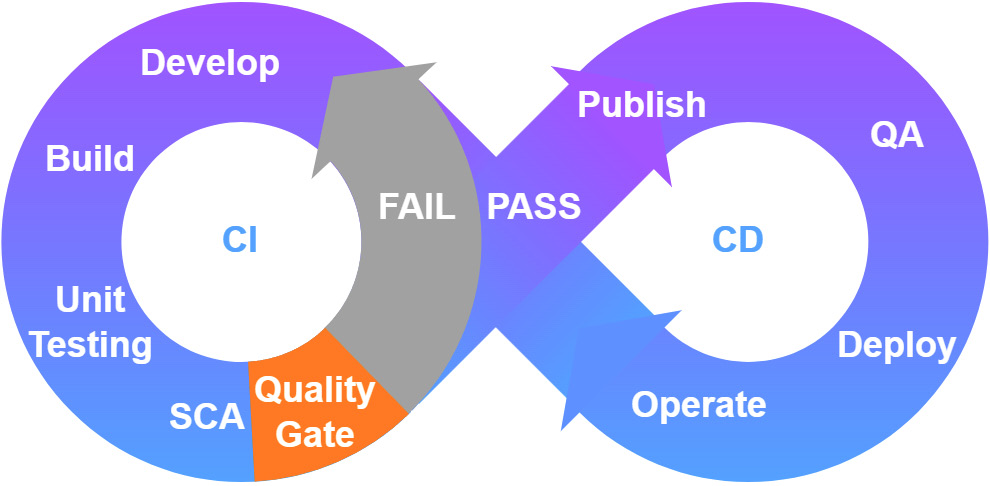
Los problemas que ves para una ejecución de análisis de código estático en particular se generan en función de las reglas de SonarQube que infringieron. Las reglas de SonarQube se clasifican de la siguiente manera:

|  |  |
| --- | --- |
| Tipo de regla | Sección de métricas de calidad |
| Code smell | Mantenibilidad |
| Errores | Fiabilidad |
| Vulnerabilidades | Seguridad |
| Punto de acceso de seguridad (security hotspot) | Seguridad |

### Puertas de calidad

Hasta ahora, hemos analizado métricas, perfiles y reglas de calidad.; pero, ¿cómo podemos convertir esta idea en acción? La respuesta son puertas de calidad*. La puerta de calidad sirve como punto de control a través del cual debe pasar el flujo de una tubería*. Las puertas de calidad son, en la práctica, combinaciones de condiciones específicas. *Un ejemplo de una condición de puerta de calidad puede ser el siguiente: el número de errores es inferior a 10 y el porcentaje de cobertura del código es superior al 95 %.*

Las tuberías de CI se pueden configurar para aprobar o reprobar según los resultados del control de calidad. Los equipos de desarrollo normalmente deciden y aplican los controles de calidad. El siguiente diagrama muestra las diversas etapas de una canalización de CI/CD, enfatizando la función crucial de Quality Gate (puertas de calidad) como un punto de control fundamental donde el código debe coincidir con los estándares de calidad establecidos antes de que pueda proceder a la implementación.



*Si el código no cumple con estos estándares en Quality Gate, se devuelve para desarrollo y mejora adicionales, garantizando que solo el software de alta calidad y probado minuciosamente avance a la fase de implementación continua para su lanzamiento y operación.*

### Gestión de dependencias de software

Las dependencias o bibliotecas de software, como npm, Maven y Conda, por mencionar algunas, permiten a los desarrolladores crear aplicaciones más rápidamente; sin embargo, el uso de estos marcos de código abierto presenta dos desafíos principales: administración y seguridad.

La herramienta Artifactory le ayuda a gestionar dependencias y a comprobar si hay vulnerabilidades. Sirve como una ventanilla única para acceder a todos los diferentes paquetes dependientes necesarios para construir su proyecto.

Esto a menudo se hace creando tantos repositorios remotos como sea necesario dentro de Artifactory, cada uno de los cuales está vinculado a su fuente de paquete apropiada en Internet. Con una URL interna de Artifactory, un repositorio de paquetes remoto en Artifactory le permite enumerar todos los paquetes accesibles desde Internet. Todo lo que debe hacer es hacer referencia a estas bibliotecas utilizando las URL de Artifactory correctas al escribir su código. Dado que su herramienta de control de versiones (Git) rastrea estas URL de Artifactory, puede fácilmente generar una *lista de materiales (BOM),* que se utiliza para crear una versión particular de su aplicación de software. Un repositorio remoto es sólo una referencia a la fuente real: nunca se descarga nada más al servidor Artifactory excepto los metadatos.

#### Análisis de composición de software (SCA)

Otra ventaja de utilizar un administrador de repositorio binario como Artifactory es *la herramienta de seguridad incorporada que escanea las bibliotecas de paquetes de código abierto en busca de vulnerabilidades y verifica las bibliotecas de código abierto en busca de violaciones de licencia.*

Herramientas tales como *JFrog Xray,* parte de la suite JFrog, escanea sus bibliotecas de paquetes en busca de componentes OSS y vulnerabilidades de seguridad y crea un informe antes de que ingresen su código fuente. SCA también se puede incorporar como una etapa separada en su proceso de CI utilizando otras herramientas SCA como Black Duck.

## Orquestación mediante una herramienta CI/CD

El corazón del sistema de CI es una herramienta de CI/CD: su función es orquestar sin problemas otras herramientas DevOps para lograr un flujo de trabajo de CI/CD totalmente automatizado; por tanto, una buena herramienta CI/CD debe proporcionar, como mínimo, las siguientes características:

* **Integración perfecta con diversas herramientas y tecnologías de software.**
* **Potente sintaxis de canalización para crear una automatización sólida**
* **Capacidades intuitivas de visualización y notificación de canalizaciones.**

### Integración perfecta y más amplia

Varias etapas de un proceso de CI se programan utilizando sus herramientas y tecnologías correspondientes; por ejemplo, para crear código escrito en Java, necesita las bibliotecas Maven y la herramienta de compilación Maven. Del mismo modo, para crear un código escrito en C++ o C#, necesita paquetes NuGet, la herramienta MS Build, etc.

De manera similar, para implementar una aplicación creada en una infraestructura determinada, el equipo podría emplear una herramienta de administración de configuración como Ansible o Chef, o Helm si se trata de una aplicación en contenedores. Y la lista continúa si continuamos incluyendo todas las demás etapas del ciclo de vida del desarrollo de software.

Debido a que existe una amplia variedad de herramientas y tecnologías, se espera que una buena herramienta de CI/CD se integre bien con tantas de ellas como sea posible.

**Jenkins es una de esas herramientas versátiles que ha resistido la prueba del tiempo.** Su ecosistema de complementos le permite integrarse con casi todas las herramientas enumeradas en la tabla periódica de DevSecOps**:**

****

### Automatización a través de ductos

Los canales de automatización se construyeron por primera vez manualmente en el servidor CI/CD usando parámetros basados ​​en GUI.; pero, a medida que los equipos de desarrollo crecieron y proliferaron los canales de automatización, a los ingenieros les resultó cada vez más difícil mantenerlos, porque el único método para realizar copias de seguridad y versionar los canales era crear una copia de seguridad de toda la herramienta CI/CD. *Freestyle Job de Jenkins y Build de TeamCity son dos de esos ejemplos.*

Pero gracias al movimiento de canalización como código, las canalizaciones de automatización ahora se pueden definir como código, lo que hace posible controlar sus versiones junto con el código fuente. Y no sólo los canales de CI/CD; describir su infraestructura completa como código también es una realidad.

Realizar así el código le permite compartir, examinar y crear procesos de automatización más largos. Además, para reducir el código de canalización redundante, incluso es posible componer componentes de canalización reutilizables como bibliotecas utilizando la función Bibliotecas compartidas en Jenkins.

*El código de canalización, en particular el código de canalización de Jenkins, es un DSL (abreviatura de lenguaje específico de dominio) construido sobre parte superior del lenguaje de programación Groovy*. La sintaxis del código de canalización puede ser escrita o declarativa y siempre sigue una estructura. Los pipelines con sintaxis escrita son más flexibles y extensibles que sus homólogos declarativos, pero este último es más sencillo de escribir. Sin embargo, para fines de CI/CD, siempre se recomienda utilizar la sintaxis de canalización declarativa.

### Visualización y notificación de tuberías

Si la ejecución de la canalización falla por algún motivo, es posible que desee saber en qué etapa o paso falló. Cuando encuentre dónde falló, querrá saber por qué falló. *Y eso es exactamente para lo que está pensada la función de visualización de canalizaciones en cualquier herramienta CI/CD.*

Un buen sistema de visualización de tuberías le ayuda a identificar la etapa o paso real fallido de una tubería de forma intuitiva. También segrega los registros de la canalización por etapas y pasos, por lo que no es necesario desplazarse por todo el registro de la canalización.

Por otro lado, un buen sistema de notificación le permite obtener comentarios rápidos sobre el estado de la ejecución de su canalización. Esto se hace agregando activadores de notificación dentro del código de su canalización como paso posterior a cada etapa; por lo tanto, si la canalización comienza a fallar, recibirá inmediatamente un correo electrónico, un mensaje en la aplicación de chat o incluso un ticket en el panel de Jira.

### Paralelización mediante compilaciones distribuidas

Ya sabemos que una canalización se divide en etapas, por ejemplo, las etapas de compilación, prueba unitaria y SCA, seguidas de la publicación en un repositorio binario.

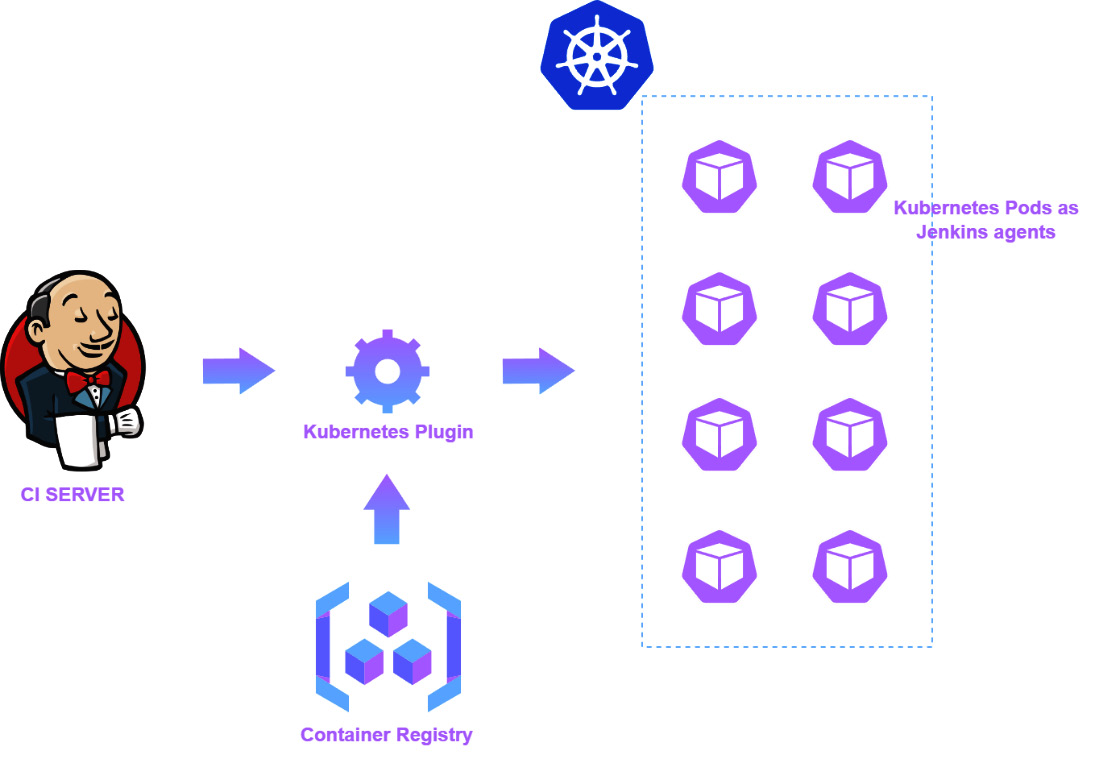
Estas etapas, particularmente en Jenkins, pueden ejecutarse en secuencia, en paralelo o una combinación de ambas. Además, cada etapa de una canalización requiere una máquina en la que ejecutarse, que puede ser una máquina virtual en VMware o la nube, o un contenedor que opera en un host Docker o un clúster de Kubernetes.

Cuando pones la CI en práctica, cada confirmación en tu repositorio de código fuente activa una canalización en Jenkins. Como resultado, en un momento dado, hay varias etapas de varios tramos de tuberías que compiten por una máquina sobre la cual operar. Como es de esperar, se requiere una granja de máquinas para completar todas las etapas.

En Jenkins, las etapas de la canalización se ejecutan en un grupo de agentes de compilación. Se **denominan granja de compilación** y están conectados al servidor Jenkins en lugar de ser el propio servidor Jenkins. La granja de compilación puede ser un proveedor de nube, un equipo VMware local, un host Docker o un clúster de Kubernetes. **Esto se llama arquitectura de compilación distribuida de Jenkins**.



El diagrama anterior es un ejemplo de una granja de compilación de Jenkins creada con instancias AWS EC2 que se escalan automáticamente según demanda. De manera similar, el siguiente es un ejemplo de una granja de compilación de Jenkins creada utilizando un clúster de Kubernetes.

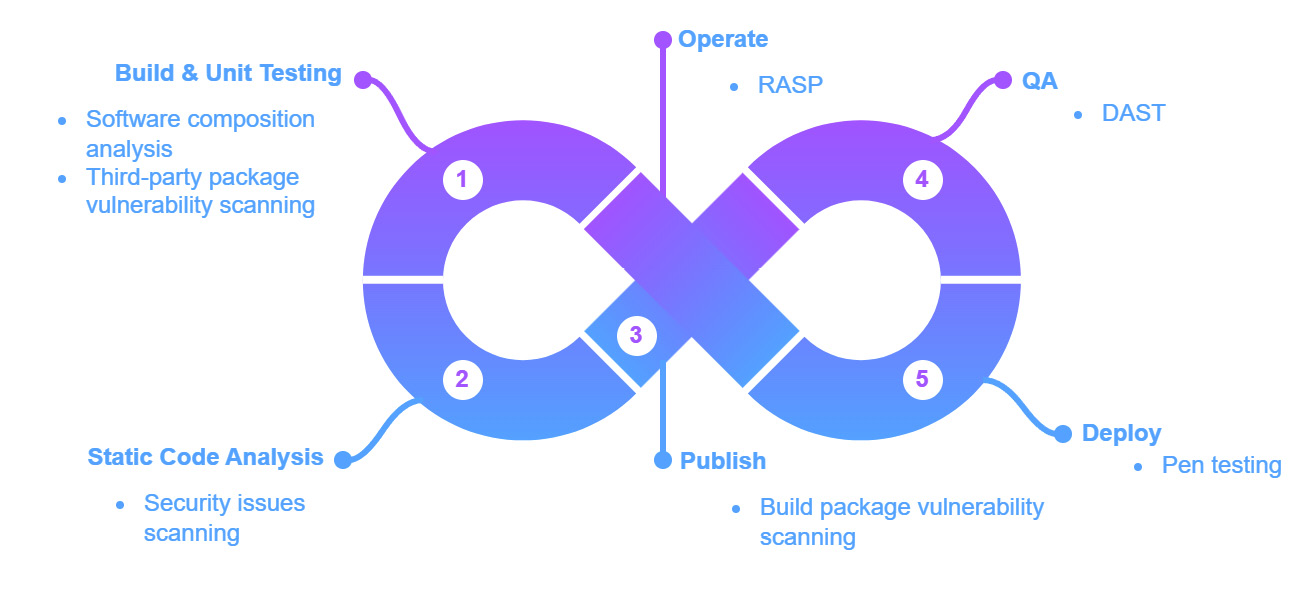


### Seguridad continua – DevSecOps

La cultura de pensar en la seguridad a lo largo del ciclo de vida del desarrollo de software como un esfuerzo colaborativo, en contraposición a algo que al final hace el equipo de seguridad, se conoce como **seguridad continua**, también conocida popularmente como DevSecOps (abreviatura de desarrollo, seguridad y operaciones

*La metodología de seguridad continua fomenta la adición de pruebas de seguridad no solo al desarrollar la aplicación, sino también al realizar comprobaciones de control de calidad en su código fuente durante la fase de CI.*

Es razonable para nosotros sugerir que si ampliamos este concepto, también debemos someter el código de infraestructura y tubería a controles de seguridad:



### Escaneando dependencias de terceros

Da igual como elijas llamarlos (dependencias de software, bibliotecas de software o paquetes binarios de terceros), los desarrolladores los utilizan con frecuencia para acelerar el proceso de desarrollo de sus aplicaciones.

Estas bibliotecas de terceros son uno de esos puntos de entrada para que programas maliciosos ingresen a su sistema. Sin embargo, la práctica de CI de utilizar un administrador de repositorio binario, como Artifactory, le permite evitar la descarga de paquetes de terceros cuestionables identificándolos y etiquetándolos directamente en el repositorio de paquetes mediante el uso de una herramienta adicional, JFrog Xray, que forma parte de la misma suite que Artifactory.

# Presentación de Docker

(cap. 2 de Continuous Delivery with Docker and Jenkins - Third Edition)