# Parte 3: La Primera Forma - FLUJO

CertiProf®

Professional Knowledge

www.certiprof.com

## 3.0 Las Prácticas Técnicas del Flujo

### ¿Por qué la Primera Forma?

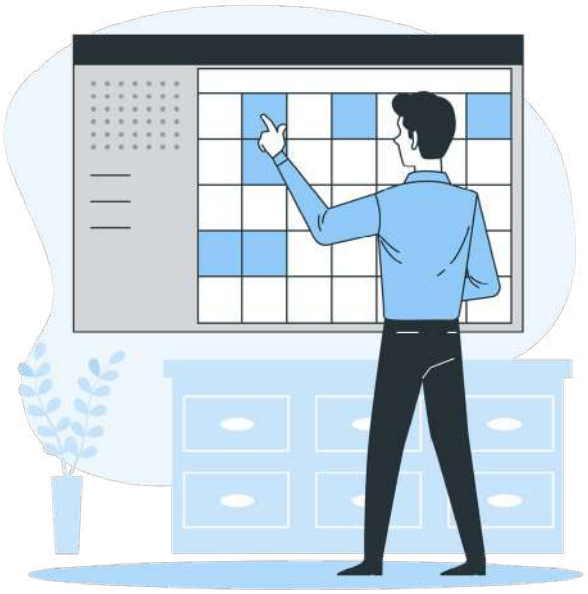
Nuestro objetivo es crear las prácticas técnicas y la arquitectura necesarias para permitir y sostener el rápido flujo de trabajo desde Desarrollo a Operaciones sin causar caos e interrupciones en el entorno de producción o en nuestros clientes. Esto significa que tenemos que reducir el riesgo asociado al despliegue y la liberación de los cambios en la producción. Para ello, aplicaremos un conjunto de prácticas técnicas conocidas como entrega continua. Los enfoques principales a ser cubiertos son:

- La creación de la base de nuestro pipeline de despliegue
- Habilitación de pruebas automatizadas rápidas y fiables
- Habilitar y practicar la integración y las pruebas continuas
- Automatización, habilitación y arquitectura para lanzamientos de bajo riesgo

Estas prácticas reducen el tiempo de espera para obtener entornos similares a los de producción, permite la realización de pruebas continuas con una rápida retroalimentación sobre el trabajo realizado, permite a los equipo desarrollar, probar y desplegar su código en producción de forma segura e independiente, y hace que los despliegues y lanzamientos de producción sean una parte rutinaria del trabajo diario.

## Agenda

- 3.1 La creación de la base de nuestro pipeline de despliegue
- 3.2 Habilitación de pruebas automatizadas rápidas y fiables
- 3.3 Desarrollo Basado en Hipótesis
- 3.4 Habilitar y practicar la integración y las pruebas continuas
- 3.5 Automatización, habilitación y arquitectura para lanzamientos de bajo riesgo
- 3.6 Arquitectura para lanzamientos de Bajo riesgo



## 3.1 Las Prácticas Técnicas Pipeline de Despliegue

### Crear los cimientos de nuestra línea de despliegue

Debatir cómo construir los mecanismos que nos permitirán crear entornos bajo demanda, ampliar el uso del control de versiones a todos los integrantes del flujo de valor, hacer que la infraestructura sea más fácil de reconstruir que de reparar y garantizar que los desarrolladores ejecuten su código en entornos similares a los de producción a lo largo de cada etapa del ciclo de vida del desarrollo de software.

En esta sección más adelante explicaremos:

1. Permitir la creación bajo demanda de entornos de desarrollo, prueba y producción
2. Crear nuestro único repositorio de la verdad para todo el sistema
3. Hacer que la infraestructura sea más fácil de reconstruir que de reparar
4. Modificar nuestra definición de desarrollo "hecho" para incluir la ejecución en entornos similares a los de producción

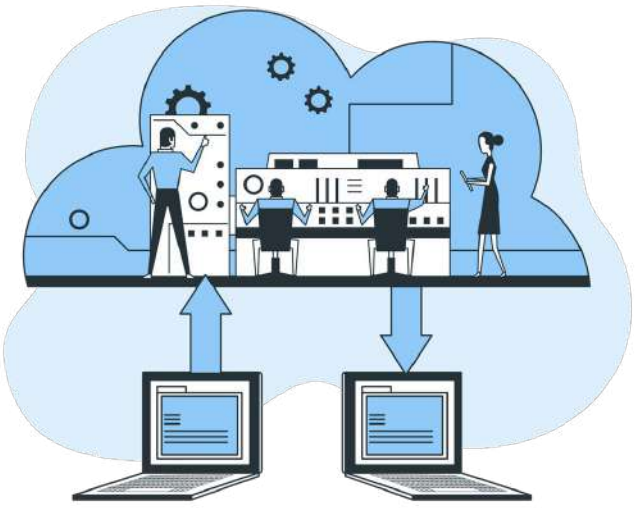
## Flujo

### Introducción

Debemos crear un flujo rápido y confiable de Dev a Ops para garantizar que siempre usamos entornos de producción en cualquier estado del flujo de valores.

Estos entornos deben ser creados de forma automatizada, idealmente bajo demanda desde scripts e información de configuración almacenados en control de versiones y totalmente autónomo, sin ningún trabajo manual requerido de Operaciones.

El objetivo es garantizar que podamos recrear todo el entorno de producción basado en el control de versiones.



*El Mapeo del Flujo de Proceso es el punto de partida para la empresa que desea elaborar un plan bien estructurado para mejorar la productividad, rentabilidad, calidad, reducción de desperdicios y reducción de lead time.*

## 3.1 El Pipeline de Despliegue

### Pipeline de despliegue

- Cómo reducir el desperdicio
- Optimice el flujo de valor
- Controles de versión compartida
- Adaptación "Definición de Hecho"
- Automatice la Construcción y Configuración de entornos

Se crea una instancia del pipeline de despliegue cada vez que se realiza un cambio en una aplicación y permite los siguientes puntos:

- ☑ Reduce el tiempo de espera para obtener entornos de producción
- ☑ Permite pruebas continuas que dan a todos la respuesta rápida sobre su trabajo
- ☑ Permite que pequeños equipos desarrollan, prueban y despliegan con seguridad independientemente el código en producción
- ☑ Hace implementaciones de producción y libera una parte rutinaria del trabajo diario

### 2.1.1 Evaluación de técnicas, infraestructura cómo código y contenedores

Abordaremos cómo crear mecanismos que nos permitan:

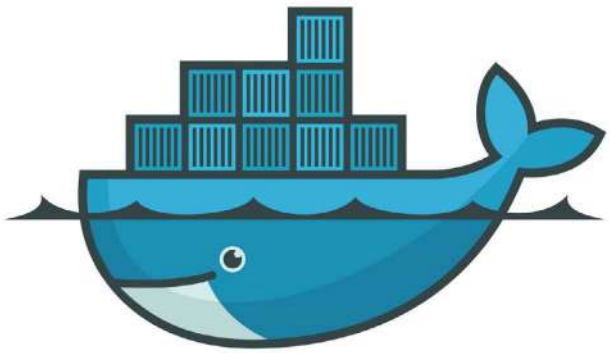
- Crear entornos bajo demanda
- Ampliar el uso del control de versiones para todos en el flujo de valor
- Hacer que la infraestructura sea más fácil de reconstruir que reparar
- Garantizar que los desarrolladores ejecuten su código en entornos de producción a lo largo de cada etapa del ciclo de vida del desarrollo de software

*El objetivo de una pipeline es automatizar el proceso de entrega de software en producción de forma rápida, al mismo tiempo que garantiza su estabilidad, calidad y resiliencia.*

## Infraestructura como código

Es el enfoque del proceso de gestión y aprovisionamiento a través de la lectura de archivos de definición de máquina en lugar de configuración de hardware físico o herramientas de configuración interactiva.

Las configuraciones deben estar en un sistema de control de versiones. Puede utilizar secuencias de comandos o definiciones declarativas en lugar de procesos manuales.

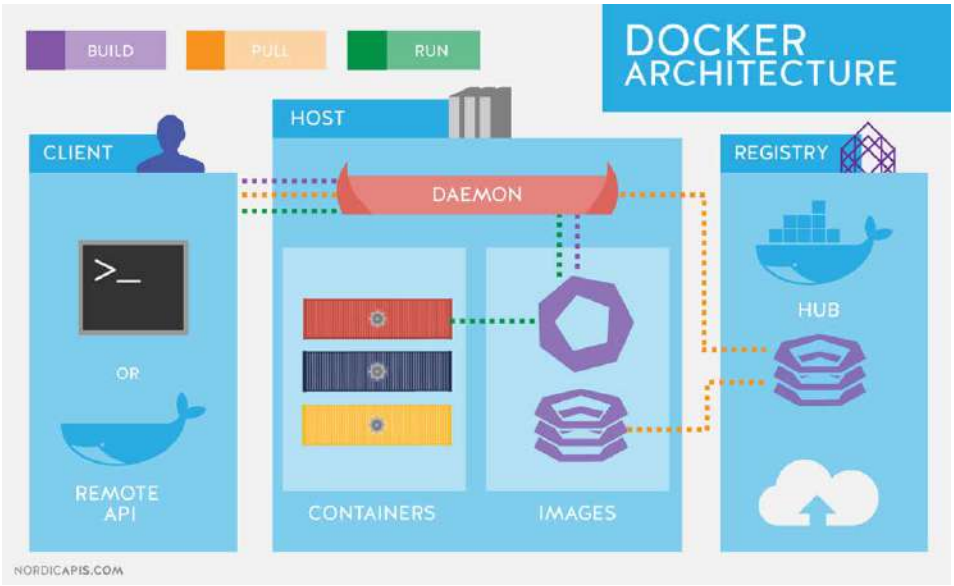


## Contenedores

Proporcionan una alternativa ligera a las máquinas virtuales y permiten a los desarrolladores trabajar con entornos y pila de DEV en forma idéntica.

Contenerización es una alternativa ligera a la virtualización completa de la máquina que implica encapsular una aplicación en un contenedor con su propio entorno operativo.

## Docker Architecture



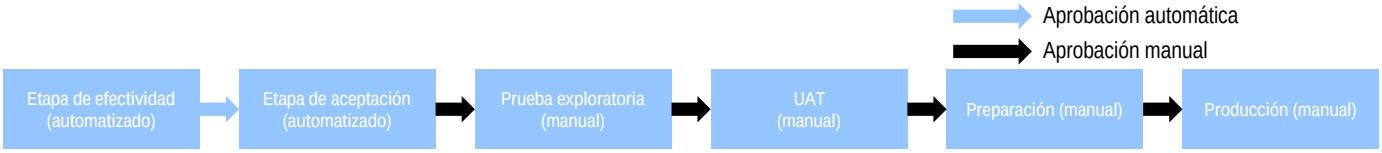

*Fuente: https://compbcn.es/docker-en-pocas-palabras/*

## 3.1 El Pipeline de Despliegue

Al definir cuidadosamente todos los aspectos del entorno antes del tiempo, es posible crear nuevos entornos rápidamente, y garantizar que estos entornos sean estables, confiables, consistentes y seguros.

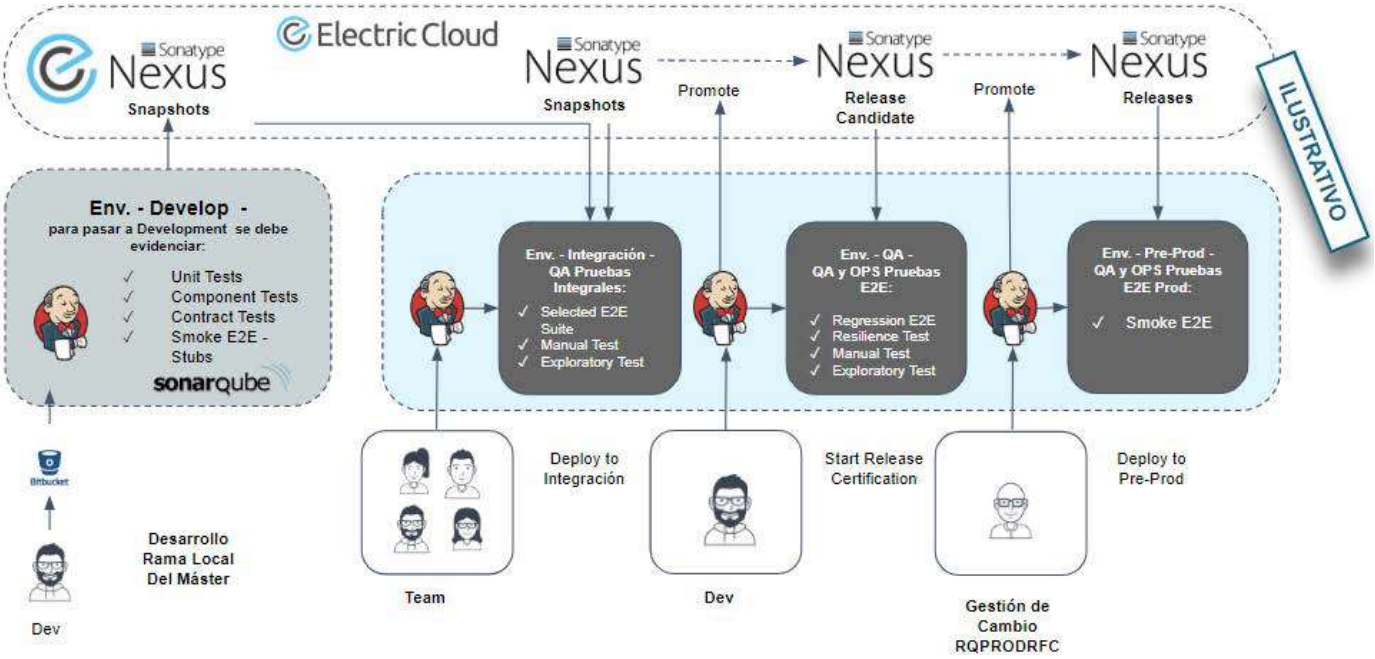
- La **Operación se beneficia** al crear nuevos entornos rápidamente, refuerza la consistencia y reduce el trabajo manual tedioso y propenso a errores
- El **Desarrollo se beneficia** al reproducir todas las partes necesarias del entorno de producción para crear, ejecutar y probar su código en sus estaciones de trabajo
- El **Desarrollo puede reproducir**, diagnosticar y corregir defectos rápidamente, aislados con seguridad de servicios de producción y otros recursos compartidos
- El **Desarrollo puede experimentar** cambios en los entornos, así como la infraestructura como un código, creando aún más conocimiento compartido entre Desarrollo y Operaciones

## 2.1.2 Mejor solución para optimizar el flujo de valor




- Definición de "Hecho"
- Cualquier provisión de entornos similares al de producción bajo demanda
- Reducir el riesgo de producción
- Las operaciones pueden hacer que los desarrolladores sean mucho más productivos
- Todos los artefactos de producción en el control de versiones. Hay una "única fuente de la verdad“
- Infraestructura de producción con foco más en reconstruir que reparar

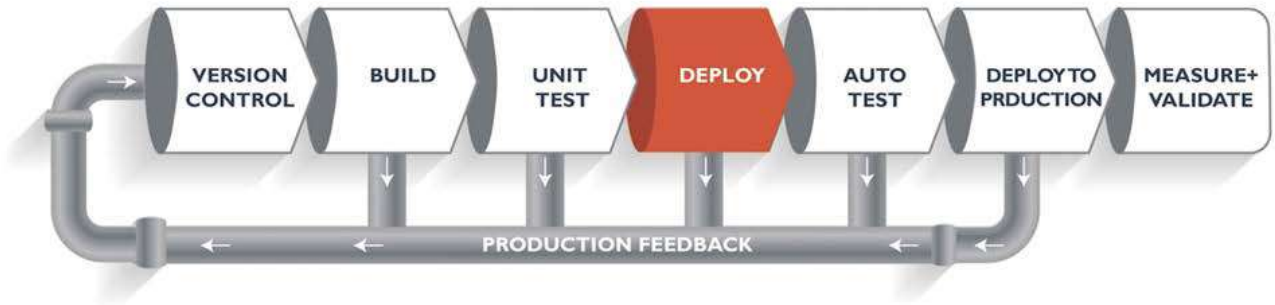
## Principios de DevOps

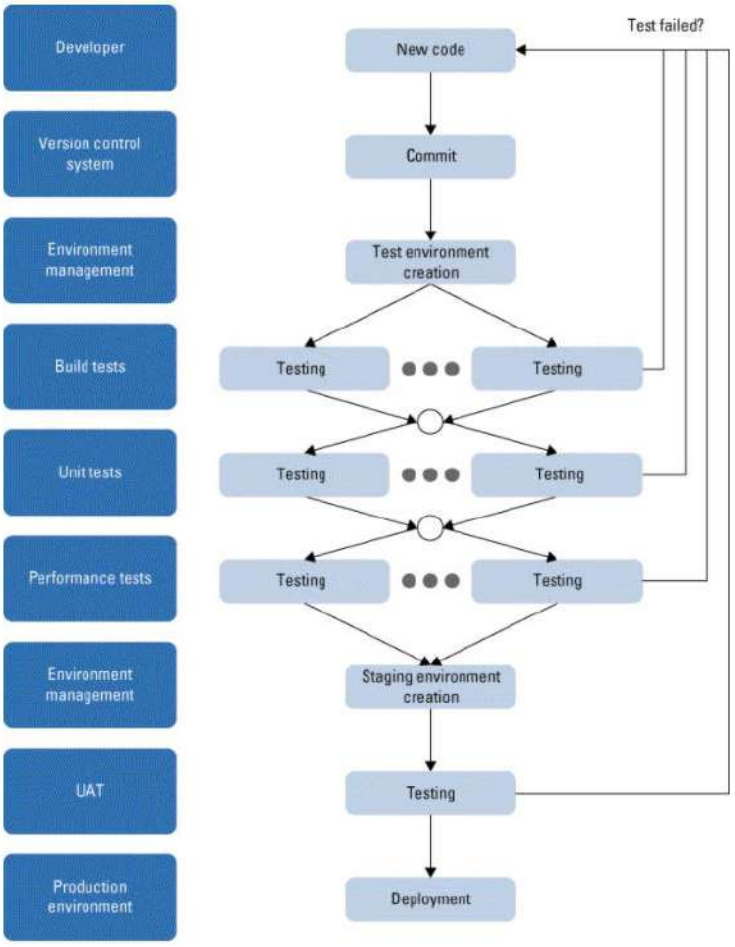



## Pipeline de Despliegue

Se usa el deployment pipeline para ayudar a la revisión de la cadena de valor, esto es la transición más automatizada de los cambios a través de todos los pasos de la cadena de valor, comenzando desde el punto 'El desarrollo está completo', hasta 'Implementado en operaciones'.

El pipeline ayuda a lidiar con tareas importantes de DevOps. Primero, ahorra recursos al no comenzar los siguientes pasos antes de que se completen los anteriores. Segundo, garantiza la calidad del producto: los cambios que no funcionan según lo requerido, no alcanzan el entorno de producción. Tercero, acelera la entrega de cambios en el entorno de producción al maximizar la automatización de cada paso. Y cuarto, constantemente deja registros en los logs de auditoría, lo que proporciona datos valiosos para su optimización.






Ayuda a lidiar con 4 tareas de DevOps:

- Ahorra recursos al no empezar una etapa antes de finalizar la otra
- Asegura la calidad del producto, ya que los cambios que no se comporten como lo esperado no alcanzan producción
- Acelera la entrega de cambios a producción automatizando cada paso
- Deja registros y logs constantes lo que permite el monitoreo de los cambios realizados y permiten la medición de cada etapa proveyendo data para la optimización

## 3.1 El Pipeline de Despliegue

### 2.1.3 Repositorio de control de versión compartida para la integridad

Todos los archivos y configuraciones de su aplicación deben estar en control de versiones; se convierte en el único repositorio de confianza que contiene el estado deseado y preciso del sistema.

Permite reproducir repetidamente y de forma confiable todos los componentes de nuestro sistema de software de trabajo, que incluye nuestras aplicaciones y entorno de producción, así como todos nuestros entornos de preproducción.

- Todos los códigos y dependencias de la aplicación
- Cualquier script utilizado para crear esquemas de base de datos, referencia de aplicación de datos, etc
- Todas las herramientas de creación de entorno y artefactos descritos en el paso anterior
- Cualquier archivo utilizado para crear contenedores
- Todas las pruebas automatizadas de soporte y cualquier script de prueba manual
- Cualquier script que admita paquetes de código, implementación, migración de base de datos y aprovisionamiento de entorno
- Todos los artefactos del proyecto
- Todos los archivos de configuración de la nube
- Cualquier otro script o información de configuración necesaria para crear infraestructura que soporta varios servicios

### 2.1.4. Definición de Hecho (DoR) para DevOps

El objetivo es asegurar que el Desarrollo y el Control de Calidad estén integrando rutinariamente el código con entornos similares al de producción a intervalos cada vez más frecuentes a lo largo del proyecto.

Si se expande la definición de "Hecho" además de sólo la funcionalidad de código correcta, al final de cada intervalo de desarrollo, se integra, prueba, se trabaja con el potencial despliegue del código, demostrados en un entorno similar al de producción.

### 2.1.5 Herramientas se pueden utilizar para automatizar la construcción y la configuración del entornos

Una de las principales causas contributivas de las implementaciones de software caótico, disruptivo y catastrófico, es la primera vez que la aplicación se comporte en el entorno de producción con un conjunto de datos reales durante la liberación.

- En algunos casos, los equipos de desarrollo pueden haber solicitado entornos de prueba en las etapas iniciales del proyecto
- Largos tiempos de espera (lead time) para el aprovisionamiento de entornos de pruebas
- Entornos con falta de datos adecuados
- Entornos de prueba mal configurados o diferentes de la operación

Queremos que los desarrolladores ejecuten sus códigos en entornos similares al de producción en sus propias estaciones de trabajo, creadas bajo demanda y en servicios de autoservicio.

- Ofrecer un mecanismo de provisión, que crea todos nuestros entornos, como desarrollo, prueba y producción

Para ello, es necesario definir y automatizar la creación de nuestros entornos conocidos y buenos, de manera estable, segura y de riesgo reducido.
Los requerimientos deben ser incorporados en el proceso automatizado de construcción del entorno.

Usar la automatización para cualquiera o todos los siguientes (en lugar de construcciones manuales de entornos y sus configuraciones):

- Copiando un entorno virtualizado
- Construcción automatizada de entorno “metal crudo”
- Uso de herramientas de administración de configuración ("infraestructura como código")
- Uso de herramientas automáticas de configuración del sistema operativo
- Montar un entorno desde un conjunto de imágenes o contenedores virtuales
- Subir un nuevo entorno en una nube pública, nube privada u otras PaaS

## 3.2 Las Prácticas Técnicas Pruebas Automatizadas

### Permitir pruebas automatizadas rápidas y fiables (1 / 2 )

Repasar las prácticas de integración continua necesarias para crear las prácticas de pruebas automatizadas que garanticen que los desarrolladores obtengan rápidamente información sobre la calidad de su trabajo. Esto es aún más importante a medida que aumentamos el número de desarrolladores y de ramas en las que trabajan en el control de versiones.

En esta sección explicaremos:
- Construir, probar e integrar continuamente nuestro código y entornos
- Construir un conjunto de pruebas de validación automatizadas rápido y fiable
- Detectar los errores lo antes posible en nuestras pruebas automatizadas
- Garantizar que las pruebas se ejecuten rápidamente (en paralelo, si es necesario)
- Escribir nuestras pruebas automatizadas antes de escribir el código ("desarrollo dirigido por pruebas")

## Permitir pruebas automatizadas rápidas y fiables (2 / 2 )

En esta sección explicaremos :

- Automatizar tantas pruebas manuales como sea posible
- Integrar las pruebas de rendimiento en nuestro conjunto de pruebas
- Integrar las pruebas de requisitos no funcionales en nuestro conjunto de pruebas
- Tirar de la cuerda de andon cuando se rompa la tubería de despliegue
- Por qué tenemos que tirar de la cuerda de andon

## 3.2 Pruebas Automatizadas

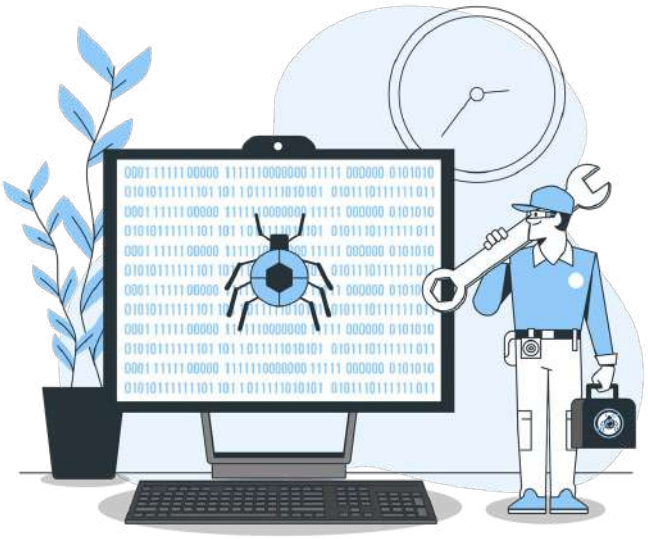
Las pruebas automatizadas aborda otro problema significativo e inquietante.

Sin pruebas automatizadas, cuanto más código se escribe, más tiempo y dinero son necesarios para probar el código, y en la mayoría de los casos, es un modelo no escalable para cualquier organización de tecnología.

Con la automatización de pruebas, es posible evitar las actividades manuales y repetitivas que sobrecargan tanto el presupuesto como el cronograma de una empresa. Además, todavía existe la posibilidad de crear pruebas más amplias, elaboradas y que estén de acuerdo con las funcionalidades y exigencias de sus productos.

Las pruebas en DevOps se pueden dividir de la siguiente manera:

- **Unitarios:** permiten la reducción de las unidades de las clases, aplicaciones y validaciones de tamaño de campos
- **Integrados**: fomentan la integración de las aplicaciones existentes en los sistemas
- **Visuales:** aseguran el funcionamiento del layout y de los elementos estáticos de una aplicación
- **Funcionales**: garantizan un buen desempeño de las funcionalidades
- **Performance**: hacen que los releases atiendan a las especificaciones previamente definidas y los comparan con resultados anteriores



Crear calidad en el producto, desde las etapas iniciales con pruebas automatizadas en el trabajo diario del desarrollador.

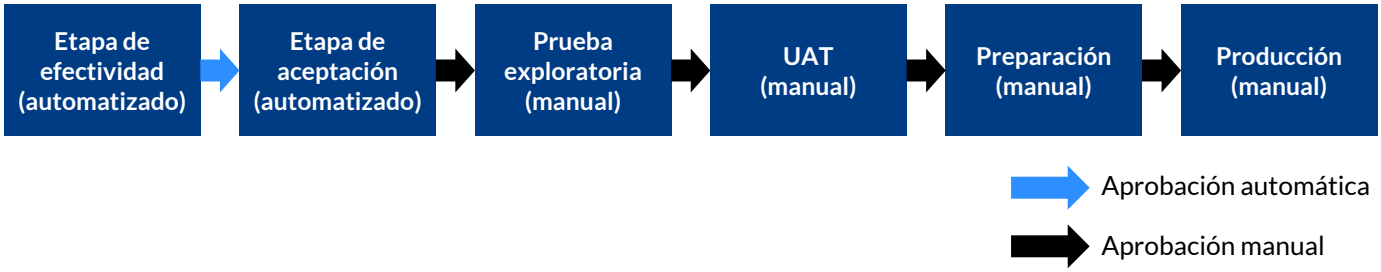
Así, se crea un loop de feedback rápido que ayuda a los desarrolladores a encontrar y corregir problemas rápidamente, cuando hay menos restricciones (por ejemplo, tiempo, recursos).

Los procesos automatizados de construcción y prueba se vuelven críticos por los siguientes motivos:

- El proceso de construcción y prueba se puede ejecutar todo el tiempo
- Entender todas las dependencias necesarias para construir, empaquetar, ejecutar y probar nuestro código
- El empaquetado de la aplicación permite la instalación repetitiva de código y configuraciones en un entorno
- Se puede optar por empaquetar nuestras aplicaciones en contenedores
- Los entornos pueden volverse más parecidos a la producción de una manera consistente y repetible

El pipeline de despliegue válida después de cada cambio, que el código se integra con éxito en un entorno de producción.

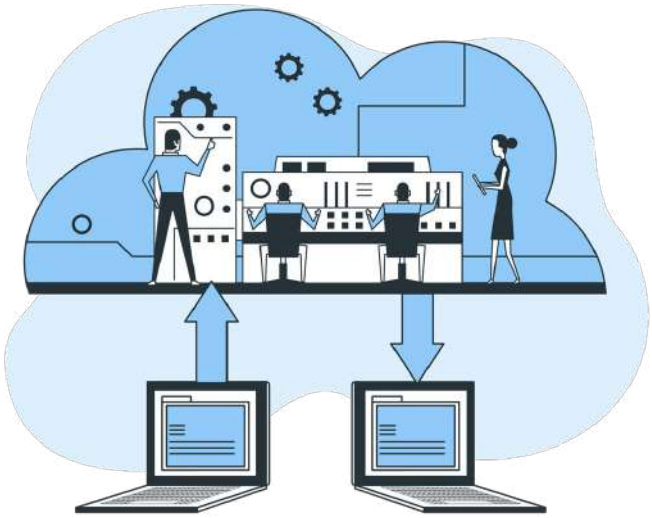
Se convierte en la plataforma a través de la cual los analistas de pruebas solicitan y certifican compilaciones durante pruebas de aceptación y pruebas de usabilidad, y donde ejecutarán validaciones automatizadas de rendimiento y seguridad.




En general, las pruebas automatizadas se encuadran en una de las siguientes categorías, desde el más sencillo al más complejo de implementar:

- Pruebas de unitarias
- Pruebas de integración
- Pruebas de aceptación

Cualquier error debe ser encontrado lo más temprano posible.



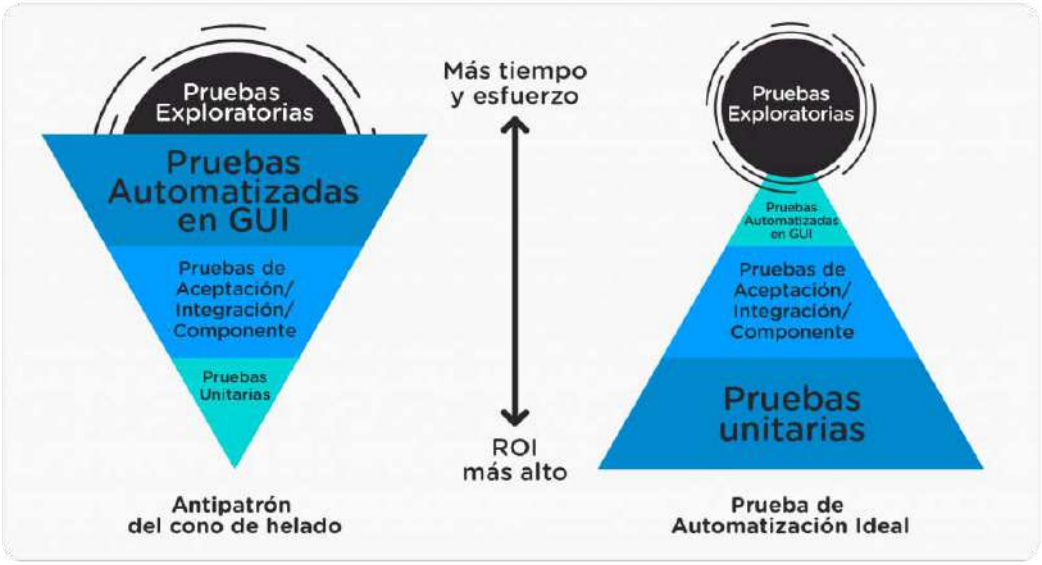
- ☑ Si la mayoría de nuestros errores se encuentran en nuestras pruebas de aceptación e integración, el feedback que proporcionamos a los desarrolladores es mucho más lento que en las pruebas unitarias

Por lo tanto, siempre que encontremos un error con una prueba de aceptación o integración, debemos crear unas pruebas unitarias que puedan encontrar el error más rápido, más temprano y más barato.

Una señal que tenemos una arquitectura fuertemente acoplada:

- Muchas veces tenemos consecuencias inesperadas en otros módulos diferentes al módulo que estamos desplegando
- Escribir y mantener pruebas de unitarias y de aceptación es difícil y costoso

En ese caso, necesitaremos crear un sistema más débilmente acoplado para que los módulos puedan ser probados independientemente.



Una de las maneras más eficaces para las pruebas automatizadas confiables es utilizar técnicas como:

- Desarrollo orientado a pruebas (TDD)
- Desarrollo orientado a pruebas de aceptación (ATDD)

Es cuando empezamos todos los cambios en el sistema, primero escribiendo una prueba automatizada que valida el comportamiento esperado y después escribimos el código que pasará por las pruebas.

Kent Beck (1990) en Extreme Programming, define tres etapas:

1. **Asegúrese de que las pruebas fallen.** "Escriba una prueba para el siguiente bit de funcionalidad que desea agregar". Check-in
2. **Asegúrese de que las pruebas sean satisfactorias**. "Escriba el código funcional hasta que la prueba pase". Check-in
3. **Refactorizar el código nuevo y antiguo para que esté bien estructurado**. Asegúrese de que las pruebas sean satisfactorias. Check-in de nuevo

## 3.3 Desarrollo Orientado por Pruebas

### Automatice tantas Pruebas Manuales como sea posible

- ☑ Aunque las pruebas se pueden automatizar, la creación de calidad no puede. Tener personas que realizan pruebas que se deben automatizar es un desperdicio de potencial humano
- ☑ Así, habilitamos a todos nuestros analistas de pruebas (lo que, por supuesto, incluye desarrolladores)
- ☑ En trabajos de actividades de alto valor que no se pueden automatizar, cómo probar o mejorar el proceso de prueba en sí
- ☑ Un pequeño número de pruebas confiables y automatizadas es casi siempre preferible a un gran número de pruebas manuales automáticas o no confiables
- ☑ Comenzamos con un pequeño número de pruebas automatizadas confiables y añadimos a ellos a lo largo del tiempo

### Desafíos de automatización de pruebas:

1. Arquitectura de automatización de pruebas
2. Paradigmas de automatización de pruebas
3. Costo de la automatización y mantenimiento de las pruebas
4. Profesionales calificados
5. Entorno de prueba
6. La garantía de la calidad
7. Expectativa de que el retorno de inversión en automatización sea de corto plazo

## Integre las pruebas de performance en su grupo de pruebas

El objetivo es crear y ejecutar pruebas de performance automatizadas que validen la performance en toda la pila de aplicaciones (código, base de datos, almacenamiento, red, virtualización, etc.) como parte del pipeline de despliegue para detectar problemas precozmente, cuando las correcciones son más baratas y más rápidas.

Las aplicaciones y los entornos se comportan bajo una carga de producción, podemos hacer un trabajo mucho mejor en la planificación de la capacidad, así como detectar condiciones como:

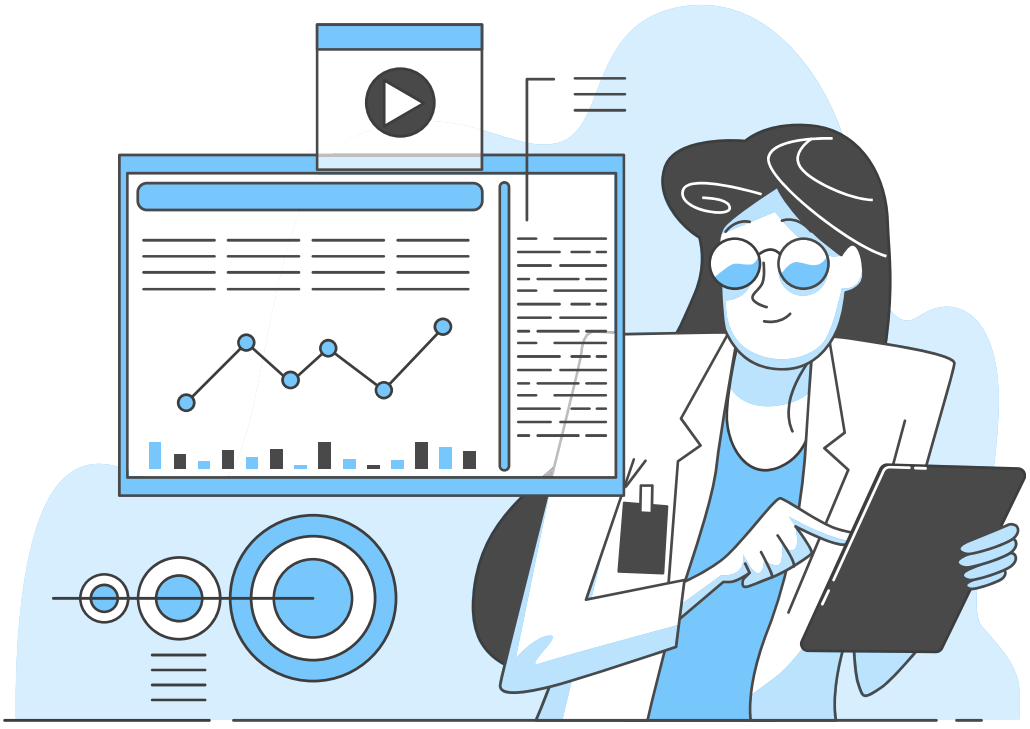
- Cuando los tiempos de consulta de base de datos crecen de forma no lineal (por ejemplo, olvidamos de activar la indexación de la base de datos y la carga de la página pasa de cien milisegundos a treinta segundos)
- Cuando un cambio de código hace que el número de llamadas de base de datos, uso de almacenamiento o tráfico de red aumenta diez veces

## Integre pruebas no funcionales en su grupo de pruebas

Es necesario validar todos los demás atributos relevantes del sistema, llamados requisitos no funcionales, que incluyen:

- Disponibilidad
- Escalabilidad
- Capacidad
- Seguridad

…y así sucesivamente.



### Tire de la cuerda ANDON cuando el pipeline de despliegue se rompe

Para mantener el pipeline de despliegue en un estado verde, crearemos una cuerda Andon virtual, similar al físico en el Sistema Toyota de Producción.

Siempre que alguien introduzca un cambio que hace que nuestra creación o pruebas automatizadas fallen, ningún nuevo trabajo puede entrar en el sistema hasta que el problema se corrija. Y si alguien necesita ayuda para resolver el problema, puede traer la ayuda que necesite.

## 3.4 Integración Continua

### Habilitar y practicar la integración continua

Tras el uso exhaustivo del control de versiones, la integración continua es una de las prácticas más críticas que permiten el rápido flujo de trabajo en nuestro flujo de valor, permitiendo a muchos equipos de desarrollo desarrollar, probar y entregar valor de forma independiente.

En esta sección explicaremos:

1. El desarrollo en lotes pequeños y lo que sucede cuando confirmamos el código al tronco con poca frecuencia
2. Adoptar prácticas de desarrollo basadas en el tronco

## Trabajar en BRANCH

Desarrollo en BRANCH en control de versiones: Se creó principalmente para permitir a los desarrolladores trabajar en diferentes partes del sistema de software en paralelo, sin el riesgo de que los desarrolladores individuales **verificaron** cambios que podrían desestabilizar el trabajo o incluso de introducir errores en el TRUNK (o maestro o mainline).

Desventajas:

- Más esfuerzo en BRANCH, más difícil integrar y combinar los cambios de todos en el tronco
- Integrar estos cambios se vuelve exponencialmente más difícil
- Los problemas de integración resultan en una cantidad significativa de retrabajo
- Si se realiza al final del proyecto (tradicionalmente es así), toma mucho más tiempo
- Espiral descendente: cuando la fusión de código es "dolorosa", tendemos a hacerlo con menos frecuencia haciendo las futuras mezclas aún peores

La integración continua fue diseñada para resolver este problema, haciendo que la combinación en el trunk una parte del trabajo diario de todos.

La integración continua resuelve una variedad sorprendente de problemas. Los principales objetivos de la integración continua son encontrar e investigar bugs más rápidamente, mejorar la calidad del software y reducir el tiempo que tarda en validar y lanzar nuevas actualizaciones de software.

***"Sin pruebas automatizadas, la integración continua es la manera más rápida de obtener una gran pila de basura electrónica que nunca se compila o se ejecuta correctamente".***

*La integración continua es una práctica de desarrollo de software de DevOps en la que los desarrolladores, a menudo, juntan sus cambios de código en un repositorio central. Después de esto, se ejecutan creaciones y pruebas.*

Gary Gruver, director de ingeniería de la división HP LaserJet Firmware, donde crean el software embarcado para todos sus escáneres, impresoras y dispositivos multifunción.

El equipo estaba compuesto por cuatrocientos desarrolladores distribuidos en Estados Unidos, Brasil e India. A pesar del tamaño del equipo, se estaban moviendo muy lentamente. Durante años, no pudieron proporcionar nuevos recursos tan rápidamente como los negocios necesitaban.

Gruver describió: "El marketing llegaba a nosotros con un millón de ideas para deslumbrar a nuestros clientes y les decíamos: 'Quítate de tu lista, elige las dos cosas que te gustaría obtener en los próximos seis o doce meses".

Sólo estaban concluyendo dos lanzamientos al año, con la mayor parte del tiempo dedicado a la portabilidad de código para soportar nuevos productos. Gruver estimó que sólo el 5% de su tiempo se gastó en la creación de nuevos recursos - el resto del tiempo fue trabajo no productivo asociado a la deuda técnica, como la gestión de varias ramas de código y pruebas manuales, como se muestra a continuación:

- 20 % en la planificación detallada (su baja productividad y largos períodos de tiempo se asignaron erróneamente a una estimación defectuosa y, en espera de una respuesta mejor, se pidió más detalles)
- 25 % en la portabilidad de código, todos mantenidos en branches de código separados
- 10 % en la integración del código entre branch de desarrollador
- 15 % en el llenado de pruebas manuales

Gruver y su equipo crearon una meta de acelerar el tiempo gastado en innovación y nuevas funcionalidades por un factor de diez. El equipo esperaba que ese objetivo pudiera ser alcanzado por medio de:

- Integración continua y desarrollo basado en TRUNK
- Inversión significativa en la automatización de pruebas
- Creación de un simulador de hardware para que las pruebas puedan ejecutarse en una plataforma virtual
- La reproducción de errores de prueba en las estaciones de trabajo del desarrollador
- Una nueva arquitectura para soportar la ejecución de todas las impresoras de un build y release comunes

**Las estrategias de ramificación o de uso de BRANCH**

**1) Optimizar para la productividad individual:**

Cada desarrollador trabaja en su propio branch privado.

Todos trabajan independientes y nadie obstaculiza el trabajo del otro; sin embargo, la fusión o mezcla se convierte en una pesadilla.

La colaboración se vuelve difícil - el trabajo de cada uno tiene que ser meticulosamente integrado con el trabajo de todos para ver hasta la parte más pequeña del sistema completo.

**2) Optimizar la productividad del equipo:**

Todos trabajan en la misma área común. No hay branch, sólo un trunk de desarrollo largo e ininterrumpido.

- Verificar el código con frecuencia reduce el tamaño de lote para el trabajo
- Cuanto más registran el código en el tronco, más cerca del ideal teórico del flujo de pieza única
- Ejecutar todas las pruebas automatizadas y recibir alertas de cambios de interrupción
- Con la detección de problemas de mezcla aún pequeños, podemos corregirlos más rápidamente
- Método confirmaciones bloqueadas: sólo es posible realizar la confirmación en el pipeline si el cambio enviado se aprueba en todas las pruebas automatizadas antes de ser mezclada en el trunk
- El control de versiones se convierte en un mecanismo integral de cómo el equipo se comunica entre sí

## Adopte prácticas de desarrollo basadas en el TRUNK

Una medida importante para las grandes mezclas es instituir prácticas de integración continua y de desarrollo basado en TRUNK:

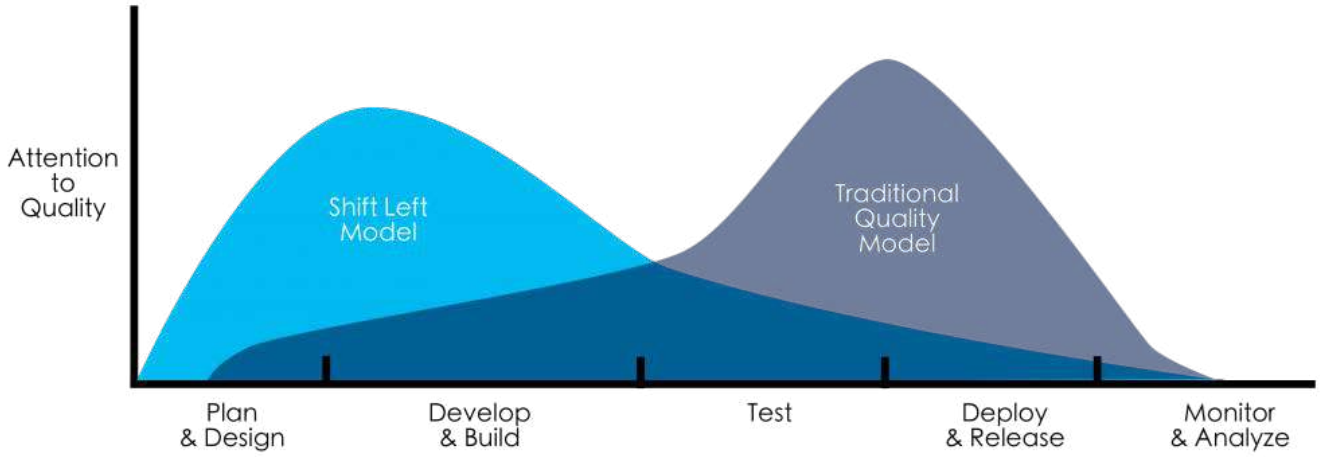
- Los desarrolladores registran su código en el tronco al menos una vez al día

Verificar el código a menudo reduce el tamaño del lote para el trabajo realizado por todo nuestro equipo de desarrolladores en un solo día.

Cuanto más los desarrolladores registran el código en el tronco, menor el tamaño del lote y más próximos estamos del ideal teórico del flujo de pieza única.

*Definición de "listo": "Al final de cada intervalo de desarrollo, debemos tener código integrado, probado, funcional y potencialmente utilizable, demostrado en un entorno similar a la producción, creado a partir del trunck usando un proceso de un clic y validado con pruebas automatizado".*

## Sponsored By: Shift Left Concept



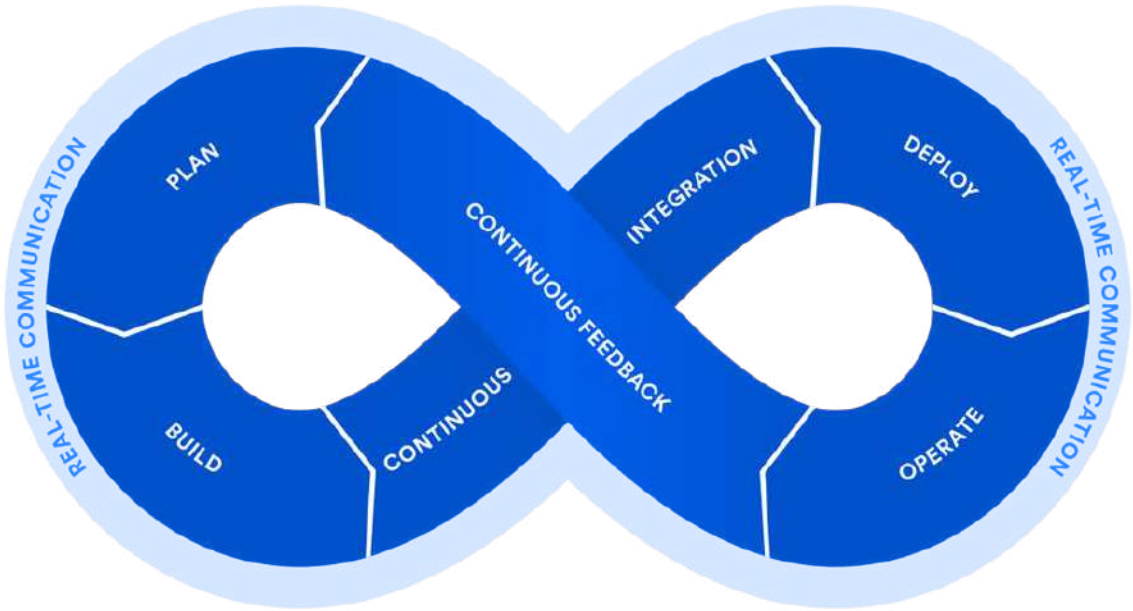

*Fuente: https://jaxenter.com/devops-shifting-left-172792.html*

## Automatice el proceso de Despliegue

- Documentar las etapas del proceso de Despliegue, en el mapeo del flujo de valor
- Simplifique y automatice el mayor número posible de etapas manuales, como:
  - ☑ Empaquetado del código en las formas adecuadas para la implementación
  - ☑ Creación de imágenes o contenedores de máquina virtual preconfigurados
  - ☑ Automatización de la implementación y configuración de middleware
  - ☑ Copiar paquetes o archivos a servidores de producción
  - ☑ Reiniciar servidores, aplicaciones o servicios
  - ☑ ....

## Automatice el proceso de Despliegue

- Intente remodelar para eliminar etapas
- También intente reducir los plazos de entrega y el número de transferencias
- Involucre desarrolladores en la automatización y optimización del proceso de Despliegue
- Ponga el equipo de desarrollo a trabajar en estrecha colaboración con las Operaciones



## 3.5 Las Prácticas Técnicas Releases

### Automatice y habilite las liberaciones de alto riesgo

En este capítulo, reducimos la fricción asociada a los despliegues de producción, asegurando que se puedan realizar con frecuencia y fácilmente, ya sea por Operaciones o por Desarrollo. Para ello, ampliaremos nuestro pipeline de despliegue.

En lugar de limitarse a integrar continuamente nuestro código en un entorno similar al de producción, habilitaremos la promoción a producción de cualquier compilación que supere nuestro proceso automatizado de pruebas y validación, ya sea bajo demanda (es decir, pulsando un botón) o automáticamente (es decir, cualquier compilación que supere todas las pruebas se despliega automáticamente).

## 3.5 Lanzamiento de Alto Riesgo

El principal tema que debemos preocuparnos no es la forma, sino los resultados: las implementaciones deben ser eventos de "presionar un botón" de bajo riesgo que podemos ejecutar bajo demanda.

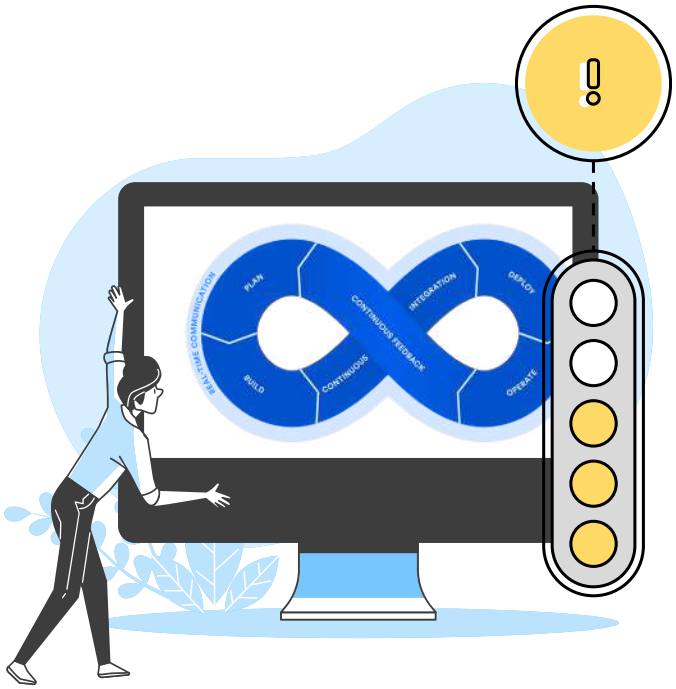
- Trabajando en pequeños lotes en el TRUNK, y siempre el código se mantiene en un estado liberable, podemos liberar bajo demanda "presionando un botón" durante el horario comercial normal, estamos haciendo una entrega continua
- Implementando buenos builds en producción regularmente a través del Auto-Servicio - desplegando a producción **al menos una vez al día por desarrollador**, o automáticamente todos los cambios que un desarrollador completa - es cuando nos estamos comprometiendo al despliegue continuo


*Definida de esta manera, **la entrega continua** es el requisito previo para el el despliegue continuo, así como la **integración continua** es un requisito previo para la **entrega continua**.*

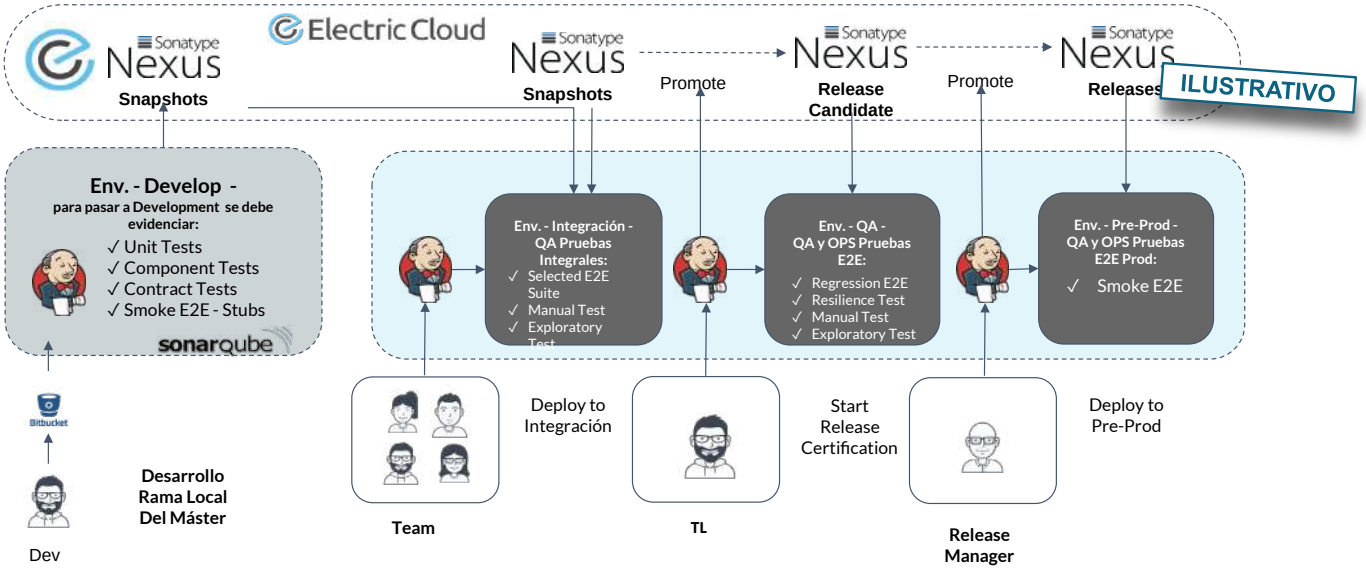
## Automatice el proceso de Despliegue

Los requisitos para el pipeline de despliegue incluyen:

- ☑ Despliegue de la misma manera en todos los entornos
- ☑ Prueba de humo de nuestras implementaciones
- ☑ Garantizar el abastecimiento y mantenimiento de entornos consistentes (para Dev, QA, Soporte y Producción)




## CI / CD Pipelines para Servicios



## 3.5 Lanzamiento de Alto Riesgo

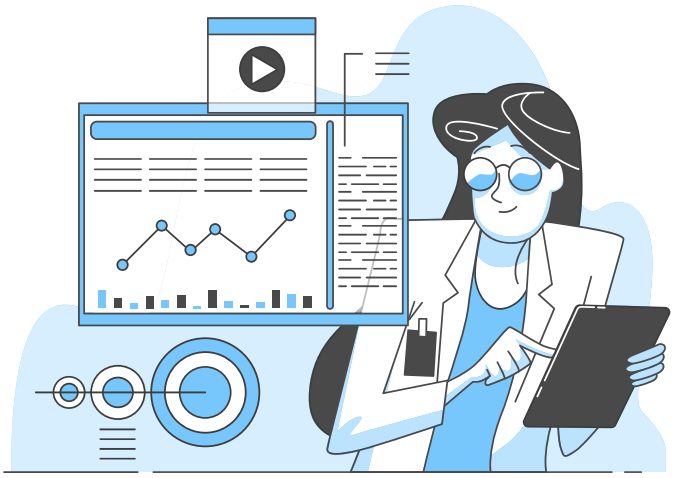
### Automatice el proceso de Despliegue

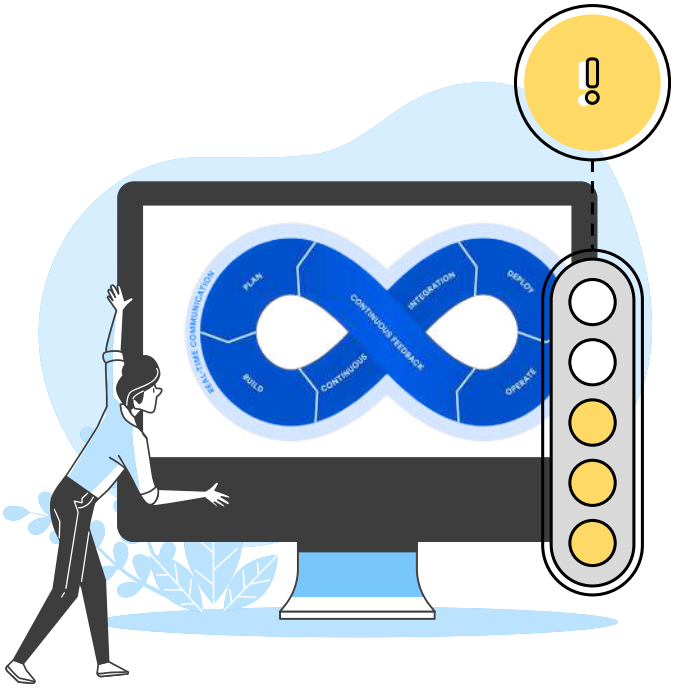
**Proporciona implementaciones automatizadas en forma de autoservicio:**

Para habilitar mejor el flujo rápido, deseamos un proceso de promoción de código que pueda ser ejecutado por Desarrollo o Operaciones, idealmente sin ninguna etapas manuales o transferencias. Esto afecta a los pasos siguientes:

- Build
- Prueba
- Despliegue

*La capacidad de los desarrolladores de auto implantar código en producción, ver rápidamente los clientes satisfechos cuando sus recursos funcionan y corregir rápidamente cualquier problema sin tener que abrir un ticket en Operaciones.*






Proporcione implementaciones automatizadas en forma de autoservicio:

- Build
- Prueba
- Despliegue

## 3.5 Las Prácticas Técnicas Releases

### Automatice y habilite las liberaciones de bajo riesgo (2 / 2 )

En esta sección explicaremos:

1. Automatizar nuestro proceso de despliegue
2. Habilitar las implantaciones automatizadas de autoservicio
3. Integrar el despliegue de código en el proceso de despliegue
4. Desvincular las implantaciones de los lanzamientos
5. Patrones de despliegue basados en el entorno
6. El patrón de despliegue azul-verde
7. Cómo hacer frente a los cambios en las bases de datos
8. Los patrones de liberación del sistema inmune canario y del clúster
9. Patrones basados en la aplicación para permitir liberaciones más seguras
10. Implantar interruptores de funciones
11. Realizar lanzamientos oscuros
12. Estudio de la entrega continua y el despliegue continuo en la práctica

## 3.5 Lanzamiento de Bajo Riesgo

### Arquitectura para liberaciones de bajo riesgo

En este capítulo, describiremos los pasos que podemos dar para invertir la espiral descendente, revisaremos los principales arquetipos arquitectónicos, examinaremos los atributos de las arquitecturas que permiten la productividad de los desarrolladores, la capacidad de prueba, la capacidad de despliegue y la seguridad, así como evaluaremos las estrategias que nos permiten migrar de forma segura desde cualquier arquitectura actual que tengamos a una que permita alcanzar mejor nuestros objetivos organizativos.

En esta sección explicaremos :

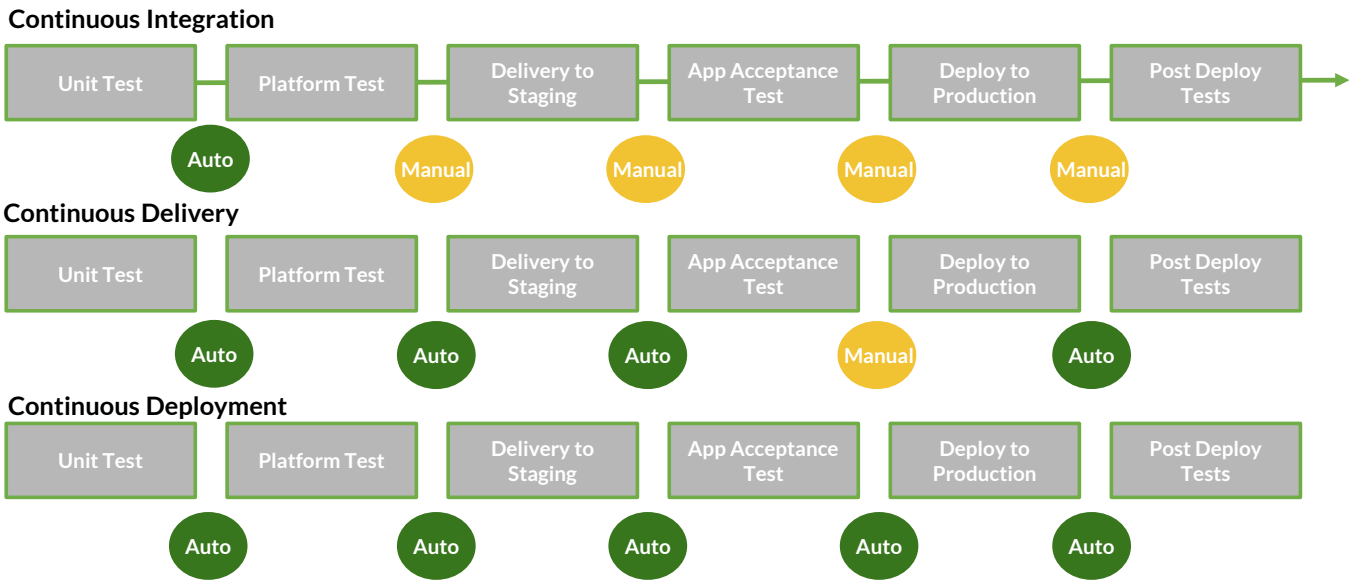
1. Una arquitectura que permite la productividad, la comprobabilidad y la seguridad
2. Arquetipos arquitectónicos: monolitos vs. Microservicios
3. Utilizar el patrón de aplicación estrangulador para evolucionar con seguridad nuestra arquitectura empresarial

## Automatice el proceso de Despliegue

### Integre el código compilado en el pipeline de despliegue

Posterior al proceso de compilación de código automatizado, es posible incluirlo en el pipeline de despliegue.

- ☑ Garantizar que los paquetes sean adecuados para su despliegue en producción
- ☑ Mostrar rápidamente la preparación de los entornos de producción
- ☑ Proporcionar un método de autoservicio accionado por un botón
- ☑ Grabar automáticamente, qué comandos, en qué máquinas, cuándo, quién autorizó y cuál fue la salida
- ☑ Ejecutar una prueba de humo
- ☑ Proporcionar feedback rápido al implantador




## Desacoplar los despliegues de las liberaciones / releases

Necesitamos desacoplar nuestros despliegues de producción de nuestras liberaciones de aspectos o funcionalidad.

En la práctica, los términos despliegue y liberación se utilizan frecuentemente de forma intercambiable. Sin embargo, son dos acciones distintas que sirven a dos propósitos muy diferentes:

- Despliegue(Deployment)
- Liberación (release)

***Los despliegues (deployment)*** *es la instalación de una versión específica del software en un entorno determinado.*

***Liberación (release)*** *es cuando ofrecemos una funcionalidad (o conjunto) a todos nuestros clientes o a un segmento de clientes*

**Automatice el proceso de Despliegue**

**Proporcione implementaciones automatizadas en forma de autoservicio:**

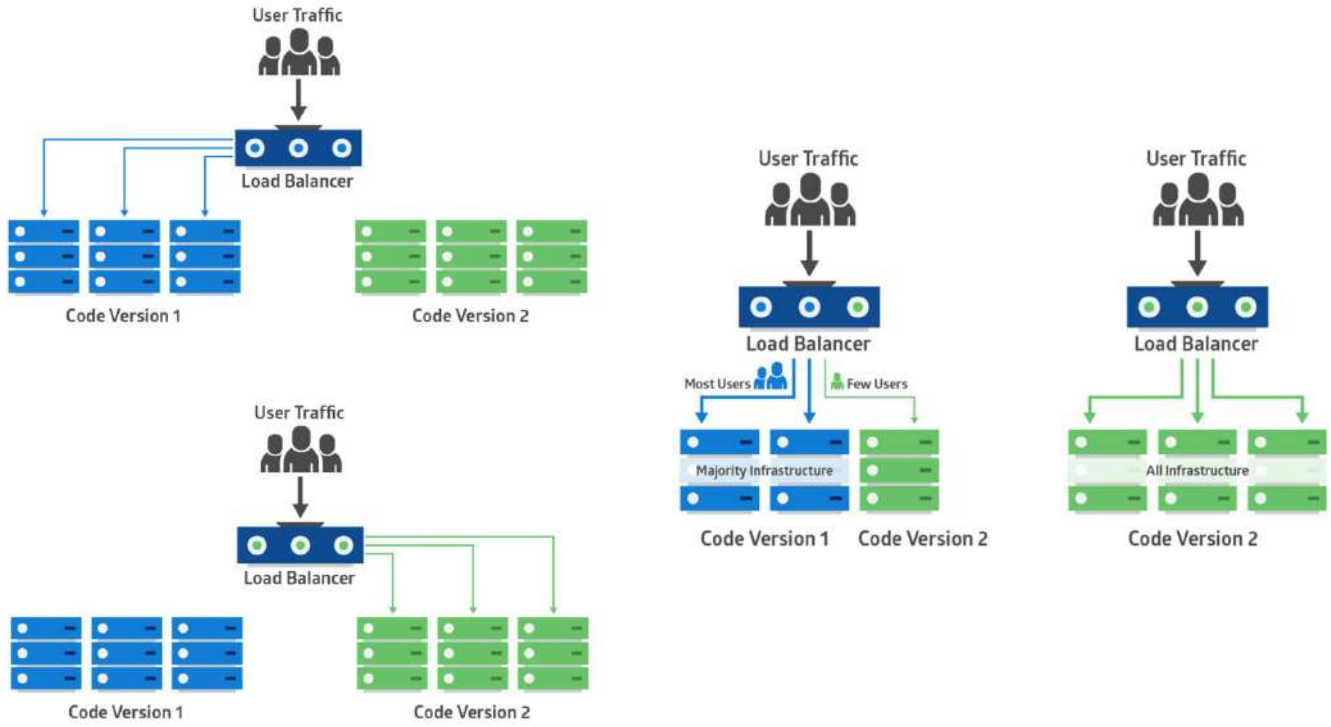
- Build
- Prueba
- Despliegue
- Liberación (Operate)

## Desacoplar los despliegues de las liberaciones / releases

Hay dos grandes categorías de estándares de lanzamiento que podemos utilizar:

- **Estándares de versión basados en el entorno**: dos o más entornos en los que desplegamos, pero sólo un entorno está recibiendo tráfico de clientes activos (por ejemplo, configurando nuestros equilibradores de carga). El nuevo código se despliega en un entorno no activo y el lanzamiento se ejecuta moviendo el tráfico a ese entorno
- **Estándares de versión basados en aplicaciones:** es donde modificamos nuestra aplicación para que podamos liberar y exponer selectivamente la funcionalidad específica de la aplicación a través de pequeños cambios en la configuración. Por ejemplo, podemos desplegar nuevas funcionalidades que exponen progresivamente nuevas funcionalidades en producción
- **Estándares de versión basados en el entorno:** dos o más entornos en los que desplegamos, pero sólo un entorno está recibiendo tráfico de clientes activos (por ejemplo, configurando nuestros equilibradores de carga). El nuevo código se despliega en un entorno no activo y el lanzamiento se ejecuta moviendo el tráfico a ese entorno

- **Estándares de versión basados en aplicaciones**: es donde modificamos nuestra aplicación para que podamos liberar y exponer selectivamente la funcionalidad específica de la aplicación a través de pequeños cambios en la configuración. Por ejemplo, podemos desplegar nuevas funcionalidades que exponen progresivamente nuevas funcionalidades en producción




*Fuente: https://dev.to/mostlyjason/intro-to-deployment-strategies-blue-green-canary-and-more-3a3*

## Estándares de versión basados en el entorno

Desacoplar despliegues de nuestros lanzamientos cambia drásticamente como trabajamos.

- Se evita despliegues en el medio de la noche o en los fines de semana
- Así, los despliegues son durante el horario de trabajo típico
- De esta manera, se disminuye significativamente el riesgo asociado a las liberaciones de producción y reducir el lead time de el despliegue

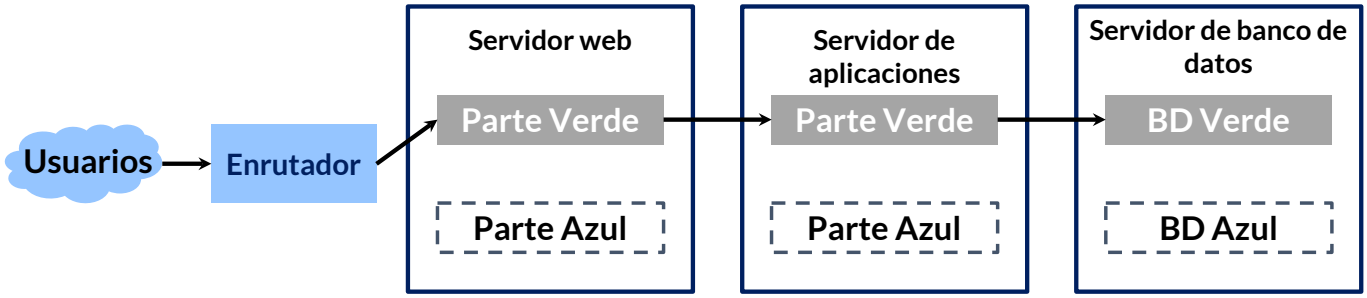
## Estándar de liberación Azul-verde:

El más simple de los tres estándares se llama el despliegue azul-verde. En este estándar, hay dos entornos de producción: azul y verde. En cualquier momento, sólo uno de ellos está atendiendo al tráfico de clientes, el entorno verde está en vivo.

## Lidiando con los cambios en la base de datos

Tener dos versiones de nuestra aplicación en producción crea problemas cuando dependen de una base de datos común. Hay dos enfoques generales para resolver este problema:

- Cree dos bases de datos (por ejemplo, una base de datos azul y otra verde)
- Desacoplar cambios de base de datos de cambios de aplicaciones




*Este estándar también es comúnmente llamado estándar de expansión/contrato. No cambiamos (mutate) objetos de base de datos, como columnas o tablas. En vez de eso, primero hemos expandido, añadiendo nuevos objetos, y luego contratamos removiendo los antiguos.*

## Estándar de liberación Canario

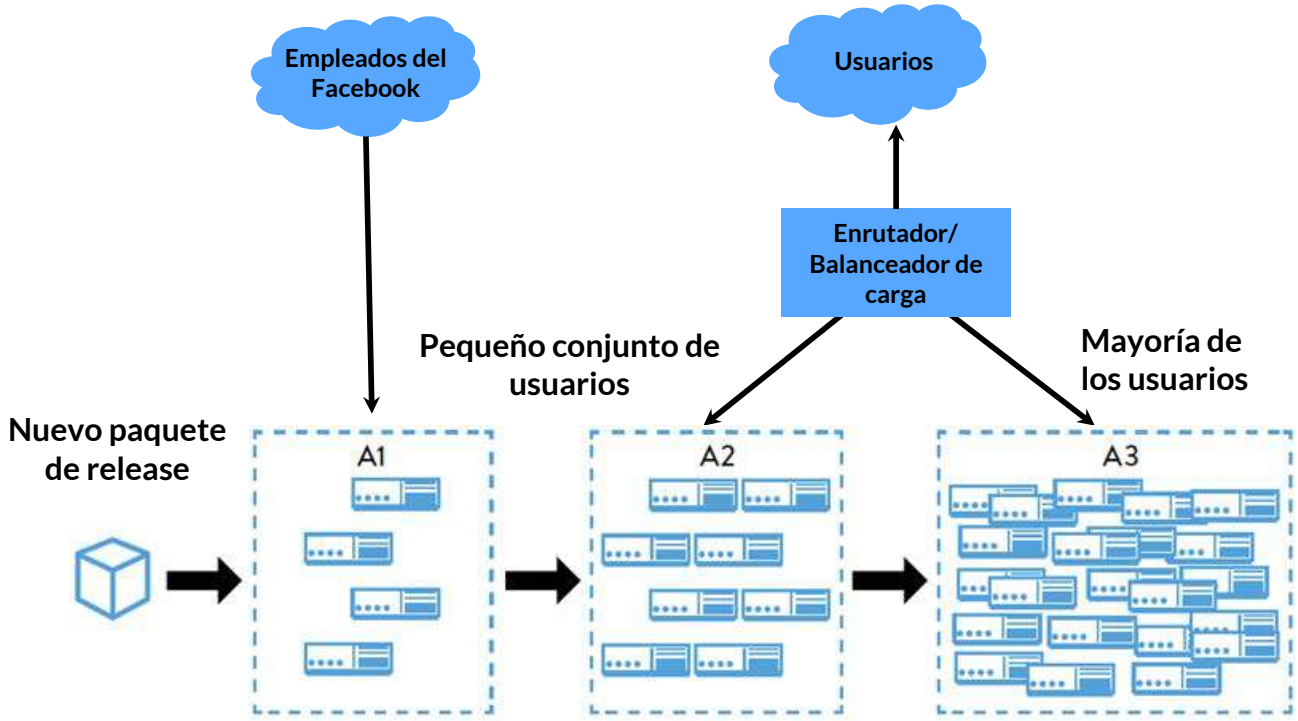
Automatiza el proceso de lanzamiento de promoción para entornos más grandes y críticos, con confirmaciones planificadas.

Se monitorea el rendimiento del software en cada entorno.

Cuando algo parece estar equivocado, retrocedemos; de lo contrario, se despliega en el siguiente entorno.

*El término liberación de canario viene de la tradición de mineros de carbón, que llevaban canarios enjaulados a las minas para proporcionar detección precoz de niveles tóxicos de monóxido de carbono. Si hubiera mucho gas en la cueva, él mataría a los canarios antes de matar a los mineros, alertándolos para evacuar.*

Ejemplo muestra los grupos de entornos que creó Facebook para soportar este estándar de release:




El sistema inmune de clúster

Expande el estándar de release canario conectando el sistema de monitoreo de producción con el proceso de release y automatizando la reversión del código.

Hay dos beneficios significativos:

1. Protegemos contra defectos que son difíciles de encontrar
2. Reducimos el tiempo necesario para detectar y responder al desempeño degradado

Aumenta la flexibilidad en la forma en que lanzamos nuevos recursos con seguridad para nuestros clientes, generalmente en una base por característica o funcionalidad.

Hay una necesidad de exigir mayor disciplina y participación del Desarrollo.

Despliega las funcionalidades selectivamente.

Mecanismo para habilitar y deshabilitar selectivamente recursos sin requerir un despliegue de código de producción. **Las alternancias de recursos** también nos permiten hacer lo siguiente:

- ☑ Retroceder fácilmente
- ☑ Degradar el rendimiento graciosamente
- ☑ Aumentar nuestra resiliencia a través de una arquitectura orientada a servicios

*Proporciona el mecanismo para habilitar y deshabilitar selectivamente recursos sin requerir despliegues de código de producción.*

## Desplegar Dark Releases/ Lanzamientos Ocultos

Las alternancias de recursos permiten desplegar recursos en la producción sin hacerlos accesibles a los usuarios, permitiendo una técnica conocida como lanzamiento oscuro.

Despliegues con toda funcionalidad en la producción:

- Realizamos pruebas de esta funcionalidad
- Todavía es invisible para los clientes

Para cambios grandes o arriesgados, muchas veces lo hacemos por semanas antes del lanzamiento en la producción, lo que nos permite probar con seguridad las cargas previstas de producción.

## 3.6 Arquitectura de Bajo Riesgo

### Arquitectura evolucionista

El desafío es cómo seguir migrando de la arquitectura que tenemos para la arquitectura que necesitamos.

Migración:

- Buscar áreas de mayor retorno posible clasificando las páginas del sitio por los ingresos producidos
- Así, elegir las áreas de mayor ingreso

**Técnica de estrangulamiento** - en lugar de "extraer y reemplazar" servicios antiguos por arquitecturas que ya no soportan las metas organizacionales, se coloca la funcionalidad existente detrás de una API y se evita hacer más cambios.


### Arquitectura que permita productividad, testeabilidad y seguridad

Arquitectura débilmente acoplada:

- Interfaces bien definidas que refuerzan cómo los módulos se conectan
- Promueve la productividad y la seguridad

Permite que equipos pequeños, productivos y de dos pizzas puedan hacer pequeños cambios que puedan ser desplegados de manera segura e independiente.

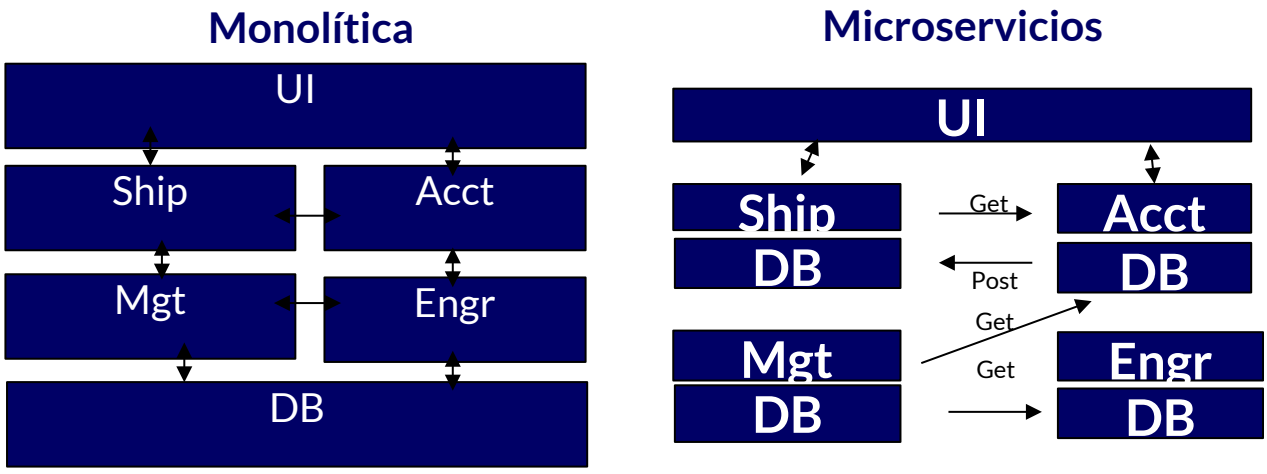
Esta arquitectura orientada a servicios permite:

- Pequeños equipos trabajen en unidades más pequeñas y más sencillas de desarrollo
- Que cada equipo puede desplegar de forma independiente, rápida y segura

## Arquitectura Monolítica versus Microservicios

En algún momento de su historia, la mayoría de las organizaciones de DevOps fue perjudicada por arquitecturas monolíticas y fuertemente acopladas que, a pesar de ser extremadamente exitosas en ayudarlas a alcanzar el ajuste del producto/mercado, colocaron al grupo en riesgo de falla organizacional.

- Las arquitecturas monolíticas no son inherentemente malas - en realidad, por lo general, son la mejor opción para una organización al principio del ciclo de vida de un producto




## Utilice el patrón estrangulador para evolucionar la arquitectura con seguridad

Si hemos determinado que nuestra arquitectura actual está muy acoplada, podemos empezar a separar con seguridad partes de la funcionalidad de nuestra arquitectura existente.

Al hacer esto, habilitamos a los equipos que soportan la funcionalidad desacoplada a desarrollar, probar y desplegar independientemente su código en producción con autonomía y seguridad, además de reducir la entropía arquitectónica.

Creando aplicaciones estranguladoras, evitamos simplemente reproducir la funcionalidad existente en alguna nueva arquitectura o tecnología - generalmente, nuestros procesos de negocio son mucho más complejos de lo necesario debido a las idiosincrasias de los sistemas existentes, que acabamos reproduciendo.