

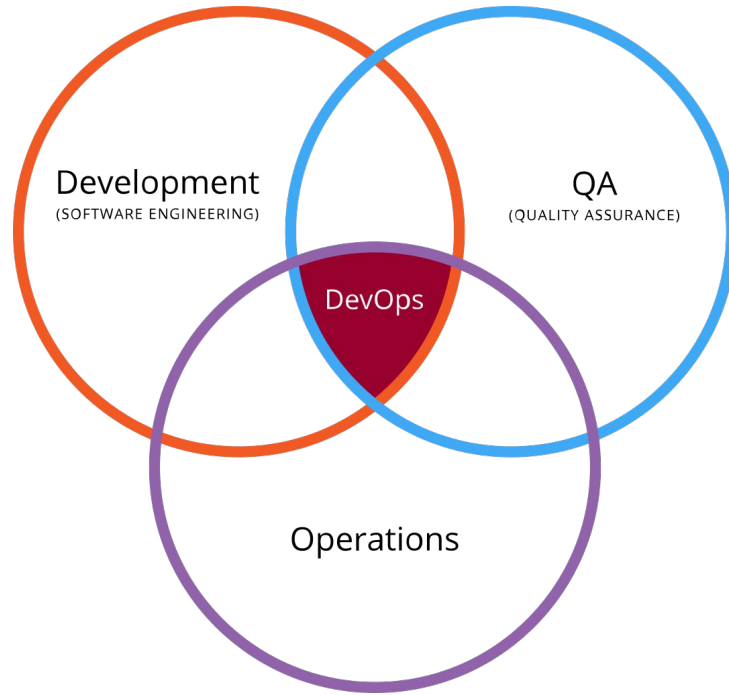
# **Integración Despliegue y Entrega Continua**

**+DEVOPS**

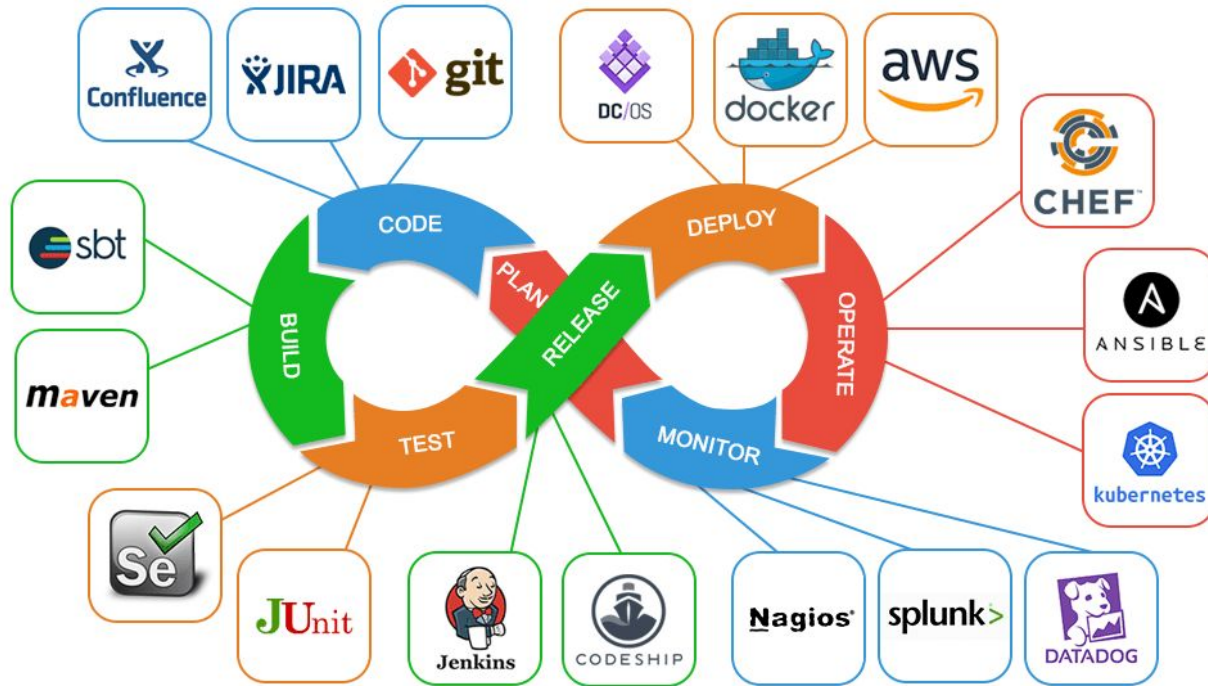
# DEVOPS

Desarrollo + Operaciones

# DEVOPS



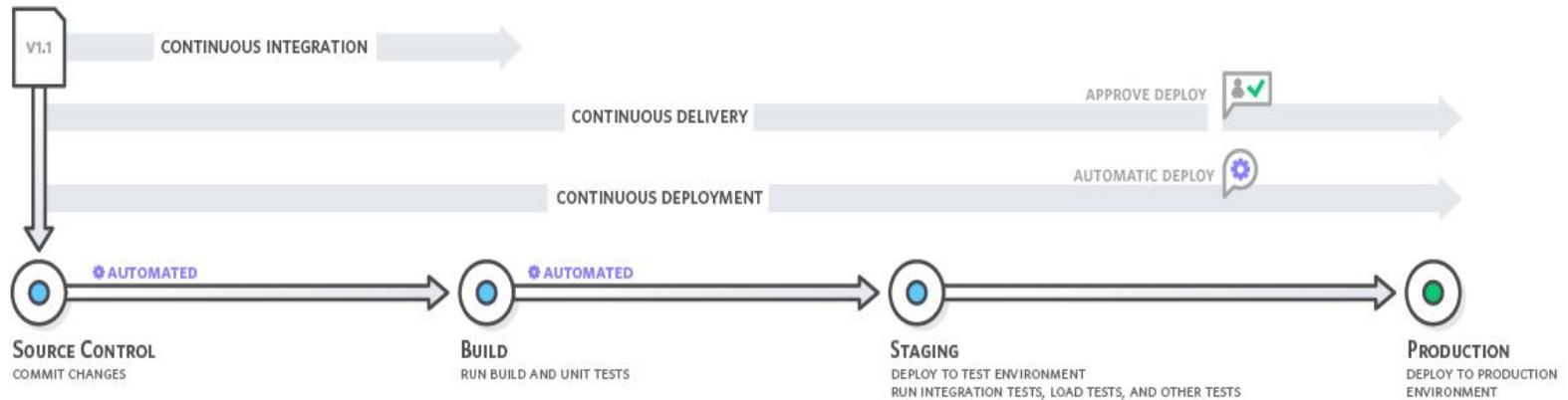
# DEVOPS-Herramientas



# Práctica de Desarrollo

Hay que definir un conjunto de etapas (pipeline), de fases por las que va pasando el software y que serán automatizadas.

# Pipeline



# ¿Qué es la Integración Continua?

Es una **práctica de desarrollo de software** mediante la cual los desarrolladores **combinan los cambios** en el código en un repositorio central de forma **periódica**, tras lo cual se ejecutan pruebas automáticas y nuevas versiones.

# Entrega Continua

Todo cambio subido al repositorio y que hayan pasado los test son compilados, probados y verificados.



# Despliegue Continuo

Pasar el sistema a producción

# Pipeline

Un ejemplo de un pipeline podría ser que con cada subida de código al repositorio de control de versiones este se descargue y compile.

Si está todo correcto, que se ejecuten una serie de pruebas unitarias, o se despliegue el código a otro entorno para hacer pruebas de sistema.

# Filosofía

Si algo te cuesta, debes hacerlo más a menudo y poco a poco, para que con el tiempo te vaya costando cada vez menos esfuerzo.

# Detectar fallos rápidamente

Cuanto más tiempo pasa desde que se introduce un error hasta que se detecta y se resuelve, más nos cuesta solucionar ese error.

# Principios (Martin Fowler)

- Un único repositorio
- Automatizar la compilación del proyecto
- Sistema realiza sus propias pruebas
- Integración diaria (mantener código estable)
- Línea principal operativa
- Reparación de errores inmediata

# Principios (Martin Fowler)

- Integración rápida (dos fases - una de pruebas)
- Pruebas en una réplica del entorno de producción
- Acceso fácil al último ejecutable
- Buena comunicación (saber estado del proceso)
- Despliegue automático en diferentes entornos

# Ventajas - Desventajas

Ventajas	Inconvenientes
Detección temprana de errores	Cambio de procesos habituales
Comunicación constante	Precisa servidores y entornos adicionales
Evita una integración final de gran envergadura	Elaboración de un plan de pruebas propio
Registro exacto de las modificaciones	Puede derivar en demoras cuando varios desarrolladores intentan integrar sus códigos al mismo tiempo
Disponibilidad constante de una versión funcional y actualizada	
Fomenta el trabajo minucioso	

# Componentes

**Control de versiones:** Git, svn

**Manejador dependencias:** Apache Maven, Apache Ant, scripts shell o programas batch

**Software Integración Continua:** Jenkins

**Software de reportes:** Sonar



# Estrategia control versiones

¿Desarrollaremos en distintas ramas?  
¿tendremos una rama de integración  
que vigilará el servidor de integración  
continua? ¿mantendremos una única  
rama? ¿una rama por tarea?

# YAML

YAML es un formato para guardar objetos de datos con estructura de árbol. Sus siglas significan YAML Ain't Markup Language (YAML no es otro lenguaje de marcas).

# YAML

# vs

# JSON

```
doe: "a deer, a female deer"  
ray: "a drop of golden sun"  
pi: 3.14159  
xmas: true  
french-hens: 3  
calling-birds:  
  - huey  
  - dewey  
  - louie  
  - fred
```

```
{  
  "doe": "a deer, a female deer",  
  "ray": "a drop of golden sun",  
  "pi": 3.14159,  
  "xmas": true,  
  "french-hens": 3,  
  "calling-birds": [  
    "huey",  
    "dewey",  
    "louie",  
    "fred"  
  ]  
}
```

# Jenkins-pipeline

Un pipeline se compone de distintas etapas (**stages**) secuenciales que ejecutan tareas (**steps**) que son lanzadas en nodos de trabajo (**nodes**)

# Step

Son las tareas ó comandos que ejecutados de forma secuencial implementan la lógica de nuestro flujo de trabajo.

Por ejemplo la descarga de un repositorio  
`git checkout master`

# Node

Los nodos son agrupaciones de tareas o steps que comparten un workspace.

# Workspace

El directorio de trabajo (workspace) es compartido por los steps del nodo, de forma que steps de un nodo pueden acceder a ficheros/directorios generados por steps de ese mismo nodo.

# Stage

Son las etapas lógicas en las que se dividen los flujos de trabajo de Jenkins. Cada **stage** puede estar compuesto por un conjunto de **steps**, y cada **node** puede contener un conjunto de **stages**.



# Resumen

Node	agrupa	Stages
Stages	agrupa	Steps