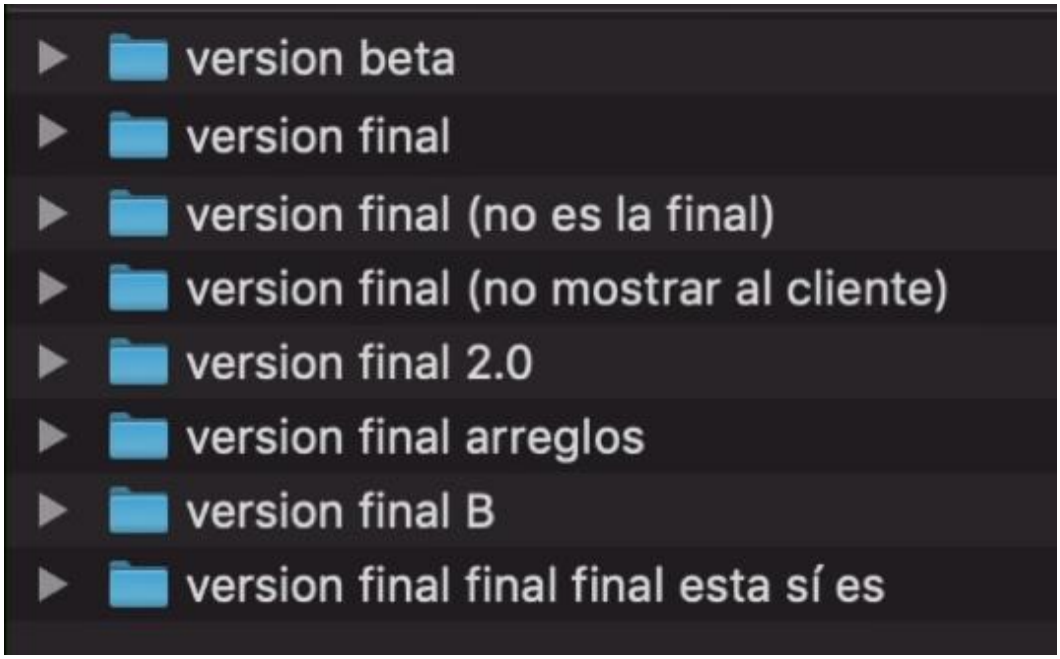
1. INTRODUCCIÓN

Cuando el ser humano vivía en cavernas, no se había inventado la rueda y el fuego solo se producía cuando caía un rayo en un árbol, el manejo de las versiones de los proyectos software tenía este aspecto:



A día de hoy, cuando empezamos a programar, todavía gestionamos las versiones de nuestros primeros programas de esta forma caótica.

Sin embargo, al trabajar con un proyecto de software con cierto grado de complejidad, surge la necesidad de controlar de forma ordenada su crecimiento y evolución. Se hace necesario mantener un historial de cambios y disponer de algún mecanismo que permita volver a alguna versión anterior de un archivo si un cambio reciente hizo que las cosas dejaran de funcionar, o comparar versiones anteriores con la actual para detectar qué parte es la que está produciendo el problema. Los sistemas de control de versiones buscan resolver estos inconvenientes, junto con agregar funcionalidades adicionales que faciliten la tarea para el o los desarrolladores, como por ejemplo, la libre modificación de un proyecto sin miedo a destruir trabajo anterior mediante la creación automática de copias de trabajo del proyecto original, almacenar sólo las modificaciones efectuadas recientemente sobre los archivos, las que serán enviadas al repositorio con un nuevo número de revisión o conocer quién y en qué momento ha modificado un archivo.

Uno de los primeros sistemas de control de versiones de éxito fue **CVS**.

2. CVS

Su funcionamiento era muy simple: el desarrollador se conectaba con el servidor CVS y le pedía la última versión disponible del proyecto, con lo cual podía ver qué cambios se han realizado respecto a su versión local y los conflictos que pudiera ocasionar el código que ha estado realizando con el que ya está disponible en el servidor. Surgió en 1986 y desde 2008 no se publican nuevas versiones.

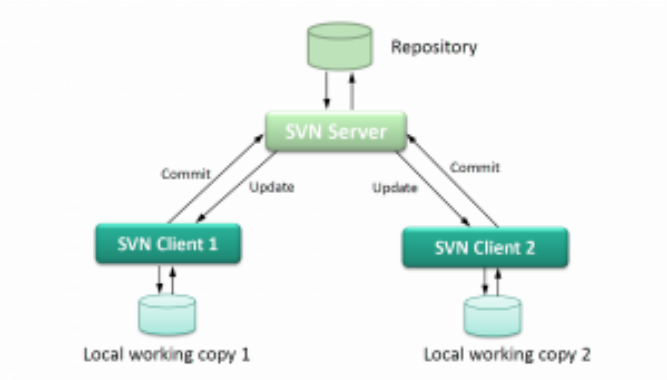
CVS utiliza la arquitectura del servidor cliente: esto significa que un servidor almacena una versión actual (o versiones) de un proyecto en particular, además de guardar su historial. Luego, el cliente se conecta al servidor como un medio para "retirar" una copia del proyecto que se ha completado antes de su conexión al servidor. El cliente puede entonces trabajar en esta copia del proyecto y luego verificar los cambios que ha realizado más adelante. Además de permitir que un cliente se registre en una copia de un determinado proyecto, CVS permite que varios clientes trabajen y se registren en el mismo proyecto al mismo tiempo. Los clientes podían modificar los archivos dentro de su propia copia de trabajo del proyecto y enviar estas modificaciones al servidor.

3. SUBVERSION

Subversion (también conocido como **SVN**) es una actualización directa de CVS. También es una tecnología de código abierto y se ha utilizado en múltiples proyectos, como Apache Software Foundation o Free Pascal.

SVN es un sistema de control de versiones centralizado, es decir, existe un repositorio que dispone de todos los archivos que forman nuestro proyecto, así como del historial de cambios realizados sobre ellos.

En los sistemas centralizados, los distintos desarrolladores, que deben trabajar en red, tienen que conectarse a un repositorio central para adquirir archivos y acceder a sus modificaciones a lo largo del tiempo. Una vez adquiridos los archivos, es posible que varios desarrolladores puedan realizar modificaciones al mismo tiempo, desde distintos ordenadores. Gracias a Subversion no hay riesgo que los cambios realizados por un desarrollador borren los cambios introducidos por otros programadores que estén trabajando a la vez en el mismo archivo.



Aunque Subversion es un sistema de control de versiones relativamente sencillo, en comparación con otros competidores, cubre la mayor parte de las

funcionalidades necesarias para un flujo correcto en el desarrollo colaborativo del software:

- Trabajar con archivos y directorios. Hace seguimiento a los directorios del mismo modo que lo hace sobre los archivos.
- Las copias, eliminaciones y renombrados de los archivos son versionados.
- Es capaz de almacenar metadatos, con información arbitraria, sobre los archivos o directorios del proyecto.
- La revisión del software se hace por commit y no para cada archivo o directorio particular.
- Sencilla creación de ramas o tags. Se pueden fusionar las ramas mediante un sistema de tracking que ofrece asistencia durante la administración de los cambios.
- Bloqueo de archivos. Aunque no es necesario bloquear los archivos para poder editar su código, es posible realizarlo si un desarrollador lo desea.
- Gracias a la resolución interactiva de los conflictos, es posible mezclar el código de los archivos versionados de manera sencilla, tanto desde la línea de comandos como diversos programas de interfaz gráfica.

Algunas de sus principales ventajas respecto a CVS son:

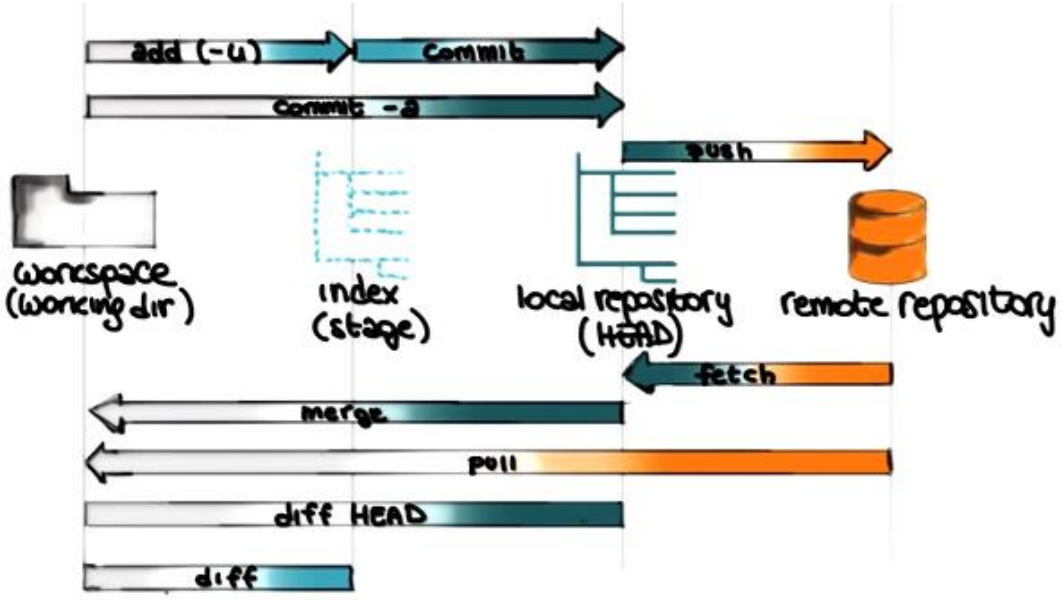
- Se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos).
- Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).
- Mejoras en el sistema de resolución de conflictos.
- Mejoras en velocidad de funcionamiento.

La principal desventaja de los sistemas de control de versiones centralizados, además de su menor flexibilidad, es que, si ese servidor sufre una caída, nadie podrá subir los cambios versionados de aquello en lo que ha estado trabajando ni podrá obtener otra versión del proyecto para empezar un nuevo desarrollo.

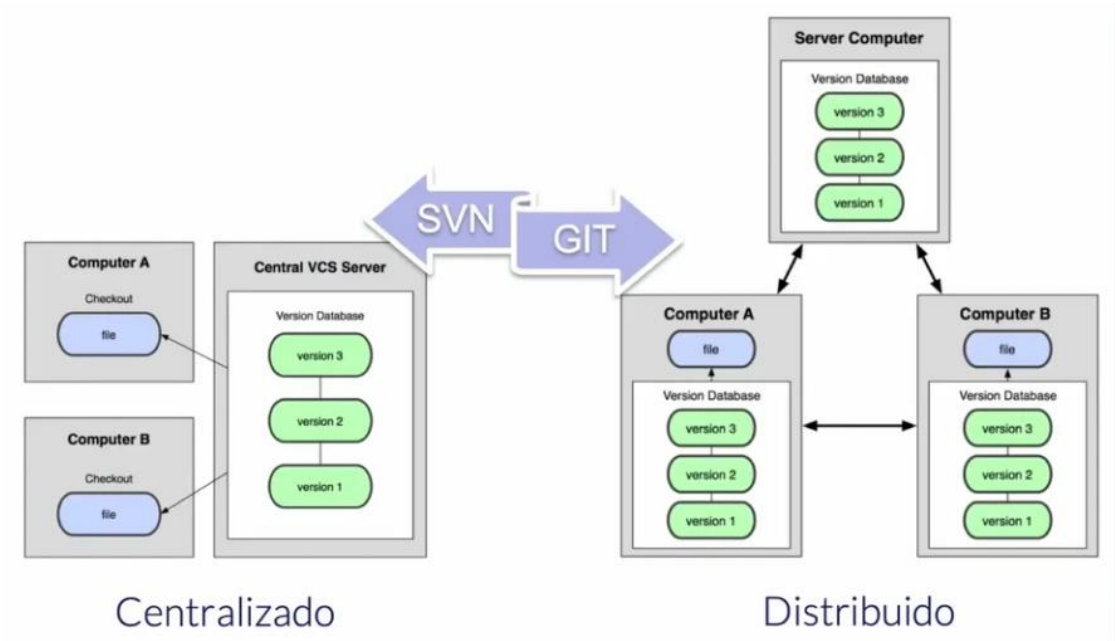
4. GIT

Un sistema de control de versiones distribuido desarrollado por Linus Torvalds en 2005. Originalmente, la comunidad de desarrolladores de Linux utilizaba BitKeeper, un sistema de control de versiones distribuido y comercial, aunque gratuito para proyectos de código abierto. Sin embargo, el hecho de ser un software propietario y ciertas desavenencias con su creador, llevaron a Linus Torvalds diseñar e implementar su propio sistema de control de versiones. Tan sólo un día después ya había desarrollado las bases de GIT para poder autogestionar el propio código de GIT en GIT y, una semana después, ya tenía una primera versión.

GIT es un sistema distribuido, lo que significa que los desarrolladores pueden commitear cambios sin tener que estar todos conectados a un servidor central, además también facilita el branching y el merging. GIT es en parte muy popular gracias a GitHub.



5. DIFERENCIA ENTRE SVN Y GIT



Subversion es un software libre que permite el control de versiones, permite el acceso al repositorio a través de redes ofreciendo la posibilidad de trabajar desde diferentes equipos, consiguiendo de esta manera la colaboración entre distintos miembros del proyecto. La última versión estable es la 1.13.0 y se lanzó en octubre de 2019. Se basa en un sistema centralizado, la principal ventaja de un sistema centralizado es su simplicidad, pero por el contrario no nos permite tener más de un repositorio central sobre el que trabajar, no podemos realizar confirmaciones (commit) si no estamos conectados al repositorio central y si

estamos trabajando en un equipo de trabajo con muchos usuarios, la colaboración se complica.

Git se basa en un sistema distribuido, siendo perfecto para el trabajo colaborativo dada su enorme flexibilidad, ya que cada usuario dispone de edición a su propio repositorio y tiene capacidad de lectura de los repositorios de los otros usuarios.

Dada su arquitectura, es posible trabajar sin conexión al repositorio central (trabajo offline), se facilita la colaboración, podemos tener tantos repositorios externos como queramos, la mayoría de las operaciones son locales por lo que el tiempo de ejecución de las mismas se reduce considerablemente y su instalación y configuración es muy sencilla en nuestro espacio de hosting.

Sus principales desventajas son que su curva de aprendizaje es alta y los flujos de trabajo (workflow) entre colaboradores pueden ser algo más complejos.

Vamos a recurrir a un ejemplo concreto para entender mejor la diferencia entre ambos sistemas. Supongamos que somos un desarrollador que está fuera de la oficina y no tenemos conexión a Internet:

- Con Subversion, no podremos conectarnos al repositorio central y, por lo tanto, no podremos realizar confirmaciones (commit) ni tener un control local de versiones del código fuente.
- Con Git, nuestra copia local es un repositorio y podemos hacer confirmaciones (commit) sobre éste y tener todas las ventajas del control de código fuente. Cuando volvamos a tener conexión con el repositorio central, podremos realizar confirmaciones sobre él.

¿Cuándo debemos emplear uno u otro?

Subversion es ideal cuando:

- Tenemos equipos de trabajo pequeños.
- Conocemos a todos los involucrados en el desarrollo del proyecto.
- Cuando no necesitamos un repositorio complejo.
- No se tiene una infraestructura para mantener varios repositorios.

Git es ideal cuando:

- Cuando tenemos equipos muy medianos a grandes.
- No conocemos a todos los involucrados en el desarrollo del proyecto.
- Cuando mantener es necesario mantener una jerarquía de confianza en las personas que afectan el repositorio.
- Cuando trabajamos en proyectos Open Source. Creamos una rama del proyecto principal, hacemos nuestros cambios sobre esa rama sin afectar a la versión en producción y generamos una petición al responsable del proyecto (pull request) para que revise las modificaciones y las integre en producción si lo considera oportuno.

En resumen:

- Mientras que SVN tiene la ventaja de que es más sencillo de aprender, Git se adapta mejor para desarrolladores que no están conectados continuamente al repositorio central.
- Git es más rápido en ejecución que SVN y características avanzadas, como la creación de ramas de trabajo (branching) y la combinación de ramas (merging) están mejor definidas.

Git no es mejor que SVN, sólo trabajan de forma diferente. Si necesitamos un control de código fuente offline y tenemos la disposición de gastar más tiempo en aprender un sistema de control de versiones, Git es nuestra elección.

Si tenemos un sistema de control de código fuente estrictamente centralizado y estamos iniciándonos en el control de versiones, la simplicidad de SVN nos facilitará nuestro flujo de trabajo.

6. GITHUB y GITLAB

GitHub es una de las primeras plataformas online que permitió hospedar proyectos utilizando el sistema de control de versiones GIT, se lanzó en 2008 y está escrito en Ruby on Rails.

La comunidad de desarrolladores de código abierto comenzó a utilizar rápidamente GitHub como plataforma para desarrollar y compartir proyectos de código abierto, lo que lo convirtió en un éxito prácticamente instantáneo.

En 2018 se estimaba que GitHub contaba con 28 millones de usuarios activos y hospedaba alrededor de 57 millones de repositorios.

Pero el éxito no solo trae consigo cosas buenas…el 28 de febrero de 2018, GitHub fue víctima del segundo mayor ataque de denegación de servicio (DDoS) de la historia. Llegando a recibir un tráfico de 1.35 terabits por segundo.

En junio de ese mismo año, Microsoft anunció la compra de GitHub por un importe de 7500 millones de dólares.

La compra originó mucha controversia entre la comunidad de desarrolladores y usuarios de GitHub, que temían que perdiera su condición de código libre. Esto hizo que muchos terminaran buscando alternativas en otras plataformas, principalmente GitLab, en la que Google comenzó a invertir grandes cantidades de dinero después de la compra de GitHub por parte de Microsoft. Al igual que GitHub, es una plataforma destinada a alojar proyectos desarrollados bajo el control de versiones GIT.

Fue lanzado en 2011 y desde el principio GitLab quiso distinguirse de GitHub.

GitLab nos permite almacenar el código de nuestros proyectos en la nube al igual que GitHub y, además, nos permite instalar su servicio en una maquina local. De esta manera, aquellas organizaciones o personas celosas de almacenar sus proyectos en la nube, podrán disponer de todas las funcionalidades ofrecidas por GitLab sin necesidad de confiar su código a terceros. Como contrapartida, seremos responsables de garantizar que el servicio este siempre disponible y no haya caídas.

Además, en sus modalidades de pago, proporciona servicios que cubren todo el ciclo de vida de DevOps, una metodología de trabajo que cada vez gana mayor importancia.

7. ENTORNO ACTUAL DE DESARROLLO

En el proceso actual de creación de software, herramientas que hemos visto en este tema, como Junit o Git, no se usan de manera aislada, sino que se su uso se encuentra integrado junto con muchas otras.

El creciente éxito del desarrollo de software ágil, llevó a que las organizaciones quieran lanzar a producción versiones de su software más rápido y con mayor frecuencia. Para que esto pueda ocurrir, tiene que pasar por todas las etapas del ciclo de vida de la aplicación, es decir, se debe realizar el análisis y diseño de la solución, se debe codificar y probar de forma aislada, se tiene que integrar con el resto de modificaciones, se tiene que implantar en los distintos entornos para que se realicen las pruebas oportunas hasta que finalmente es validado y liberado a producción. Así surge **DevOps** (development and operations).

7.1 DevOps

El termino **DevOps** se introduce por primera vez en la conferencia Agile 2008 Toronto, en una charla sobre "Infraestructura Ágil" y es una práctica de ingeniería de software que tiene como objetivo unificar el desarrollo de software (Dev) y la operación del software (Ops). El principal objetivo de DevOps es lograr reducir al máximo el tiempo que va desde el desarrollo del software, hasta su puesta en producción, pasando por la integración, las pruebas, la liberación, la administración de la infraestructura, etc.

Para reducir este tiempo necesario en cada uno de estos pasos la máxima automatización de cada uno de ellos, de forma que la intervención humana sea mínima y cuando realmente sea necesaria. En este sentido hay que destacar las técnicas de integración continua, entrega continua y despliegue continuo.

7.2 Integración continua

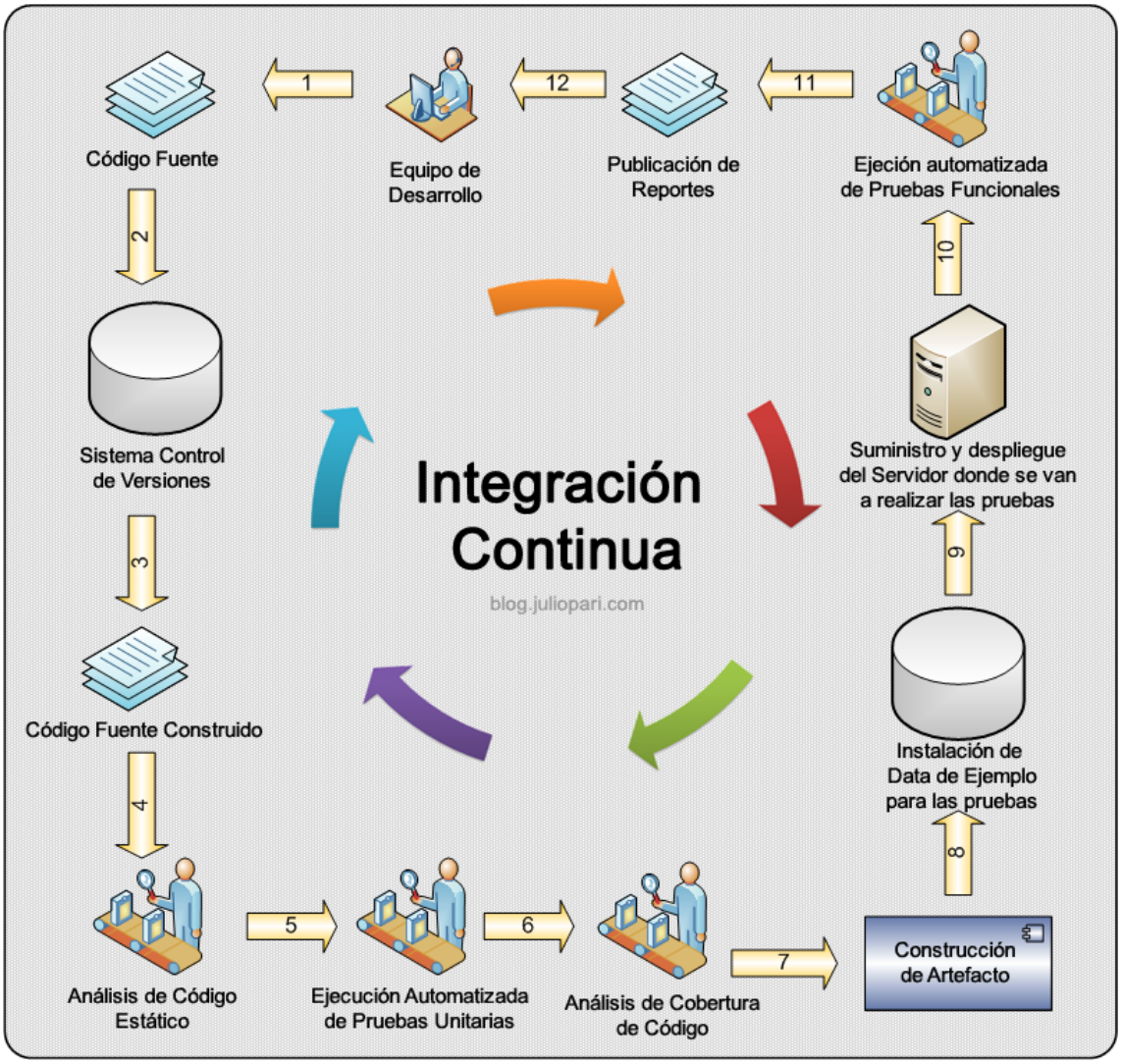
Es un modelo que se basa en compilar, realizar el análisis de código estático y ejecutar pruebas automáticas de todo el proyecto lo más a menudo posible de cara a la detección temprana de errores.

El proceso se ejecuta de forma periódica, o bien según ciertos disparadores y suele ser de la siguiente forma:

- Se descarga del control de versiones las últimas fuentes con los cambios implementados a un directorio de trabajo.
- Se realiza la compilación de estas fuentes.
- Se lanza el análisis estático y de cobertura de las fuentes.
- Se instala en el entorno (compartido) de Desarrollo.
- Se lanzan las pruebas automáticas.
- Se generan los informes.

Si alguno de estos pasos falla se da por errónea la ejecución y se notifica a los responsables oportunos.

Este enfoque permite la identificación temprana de errores ya que no se ha salido de la propia fase de desarrollo cuando se están encontrando posibles fallos o carencias en el código, por lo que es mucho más sencillo proceder a la subsanación de estos y se evita los cambios de última hora antes de la entrega.



7.3 Entrega continua

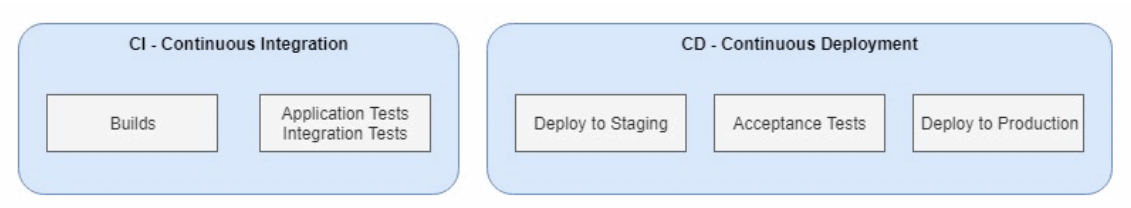
Es el siguiente paso de la integración continua. Su objetivo no es otro que el de proporcionar una manera ágil y fiable de poder entregar o desplegar los nuevos cambios a los clientes. Esto se consigue automatizando el proceso de despliegue, aunque esto ocurrirá al hacer clic sobre la acción de despliegue.

7.4 Despliegue continuo

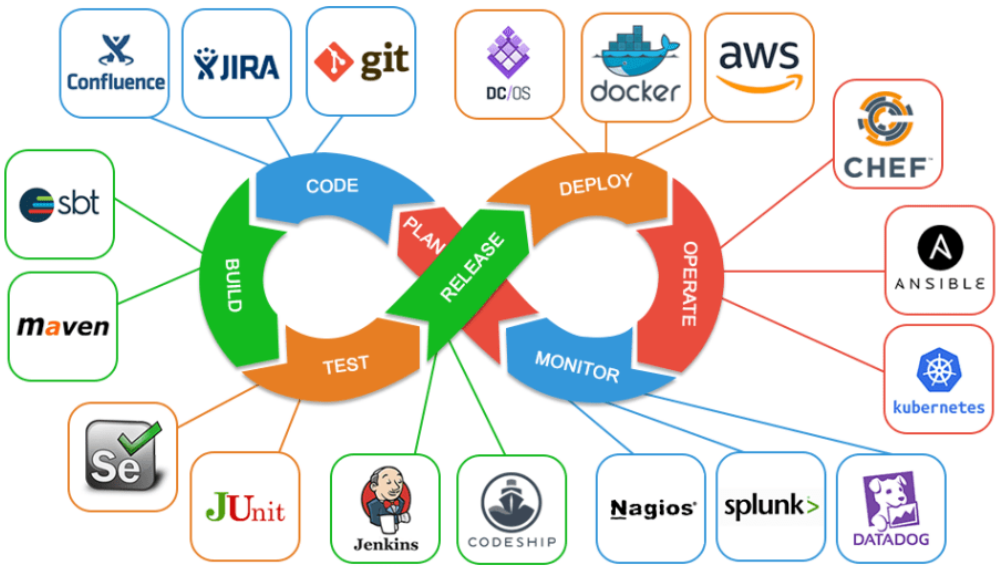
El objetivo a diferencia de la entrega continua es que no exista intervención humana a la hora de realizar el despliegue de nuestro Software en producción.

El resumen de CI + Continuous Delivery se puede ver en la siguiente imagen:

El resumen de CI + Continuos Deployment se puede ver en la siguiente imagen:



Por último vamos a mencionar las principales herramientas utilizadas en la práctica de DevOps:



Estas herramientas se pueden clasificar en una serie de categorías según su función. Una posible clasificación sería la siguiente:

1. Colaborativas: Jira, Confluence, Slack, Trello.
2. Gestión y revisión de código: Git, SVN, Mercurial.
3. Integración y despliegue continuo: Bamboo, Jenkins.
4. Automatización de pruebas: Selenium, JUnit.
5. Automatización de releases: IBM Urbancode, AWS CodeDeploy
6. Gestión de configuraciones: Puppet, Chef, Salt, Ansible.

7. Monitorización: Nagios, Datadog, Dynatrace

8.Gestión de infraestructuras: Terraform, Vagrant, AWS Cloud Formation.

9. Provisionamiento y empaquetado: Docker, Kubernetes, Google/Azure/AWS

10. Computación serveless: AWS Lambda, GCP Functions, Azure WebJobs