TEMA 097. MODELOS DE INTEGRACIÓN CONTINUA. HERRAMIENTAS Y SUS APLICACIONES

Actualizado a 23/01/2022

1 INTEGRACIÓN CONTINUA

La integración continua (CI) no trata de una herramienta sino de una metodología de desarrollo. Consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible **para poder detectar fallos cuanto antes**. Entendemos por integración: la compilación, ejecución de pruebas y análisis de la calidad del código.

El proceso de integración continua tiene como objetivo principal comprobar que cada actualización del código fuente no genera problemas en una aplicación en desarrollo, y de esta manera poder detectar errores lo antes posible.

A menudo la integración continua está asociada con las metodologías de desarrollo ágil, por el hecho de que favorece al proceso de entrega. Sin sistemas de CI, sería difícil y caro entregar software en iteraciones cortas.

1.1 BENEFICIOS

- Detección de problemas de integración de forma continua, evitando el caos de última hora cuando se acercan las fechas de entrega (The Master Build and Checking in)
- Disponibilidad constante de una versión para pruebas, demos o lanzamientos anticipados. (Single Source Point)
- Ejecución inmediata de las pruebas unitarias (Self-Testing Code)
- Monitorización continua de las métricas de calidad del proyecto. (The more often The Better)
- Disminución de errores humanos a través de automatización de tareas. (Automated Build Scripts)
- Posibilidad de reducir el tiempo para liberar cambios a producción. (Less Time to market)

Artículo - Martin Fowler - Continuous Integration ,
https://www.martinfowler.com/articles/continuousIntegration.html

1.2 TIPOS DE HERRAMIENTAS

Para ayudar a implementar la metodología de integración continua existen distintas herramientas, cada una centrada en una o varias funcionalidades.

Un proyecto de integración continua necesita, al menos, los siguientes componentes:

1. Una **herramienta de gestión y construcción de proyectos** para gestionar la configuración y compilar el código, si fuera necesario. Hay muchas soluciones de construcción diferentes Apache Ant, Maven basado en XML, Gradle basado en Groovy o Kotlin DSL, Rake, MSBuild en .NET, por mencionar algunas.
2. Un **sistema de control de versiones o SCM System Control Management**: es la herramienta que almacena todos los cambios realizados tanto en la estructura de directorios como en el contenido de los ficheros, con el fin de tener un histórico navegable. Ofrece persistencia en un entorno dedicado. Es imprescindible cuando más de una persona trabaja con los mismos archivos, ofreciendo una trazabilidad del cambio: quién, cuándo, qué, ... Además, permite trabajar con distintas ramas, generar releases, etc.

   En este grupo encontramos dos tipos de SCM:

a. **SCM centralizados**: Subversion (SVN), ClearCase, Perforce, Concurrent Versions System (CVS), Microsoft Visual SourceSafe
b. **SCM distribuidos** como Git y Mercurial. A diferencia de los anteriores, permiten trabajar de forma descentralizada y ofrecen al usuario su propio sistema de control de versiones. Son más complejos que los anteriores ya que se tiene que interactuar con el sistema de control central y el individual, pero ofrecen más funciones y su uso está altamente extendido. Existen distintos productos Software con funcionalidades de plataformas de software colaborativo o forja basados en Git: Github, Gitlab, BitBucket, Gerrit

3. Un **repositorio de artefactos** donde almacenar archivos binarios o paquetes compilados (.jar, .war, .ear, paquetes npm, …) generados del código construido con una herramienta de Construcción. (1). Ejemplos de Repositorios de Artefactos: Nexus, Artifactory, Apache Archiva.

4. Un **servidor de integración continua**: es el orquestador de CI, permite planificar e implementar multitud de tareas, simplificando los procesos involucrados en el ciclo de vida de desarrollo de un proyecto. Es decir, permite programar, por ejemplo, la construcción del software y la ejecución de las pruebas, mostrando los resultados de forma centralizada y realizando acciones en base a los resultados. Ejemplos: Jenkins, Hudson, Bamboo. Destacar además que la tendencia actual es que las herramientas basadas en Git incorporen módulos con funcionalidad CI como: Github Actions, GitLab CI o Bitbucket Pipelines.
5. Una **herramienta para el análisis estático de la calidad del código**. Ejemplo: SonarQube, SonarCloud, Sonarlint (En local)
6. Herramientas de pruebas
    a. Pruebas unitarias Junit, TestNG,
    b. Pruebas de Regresión Selenium, Cypress, (opcional)
    c. Pruebas de integración de Servicios Web Soapui, Postman (opcional)

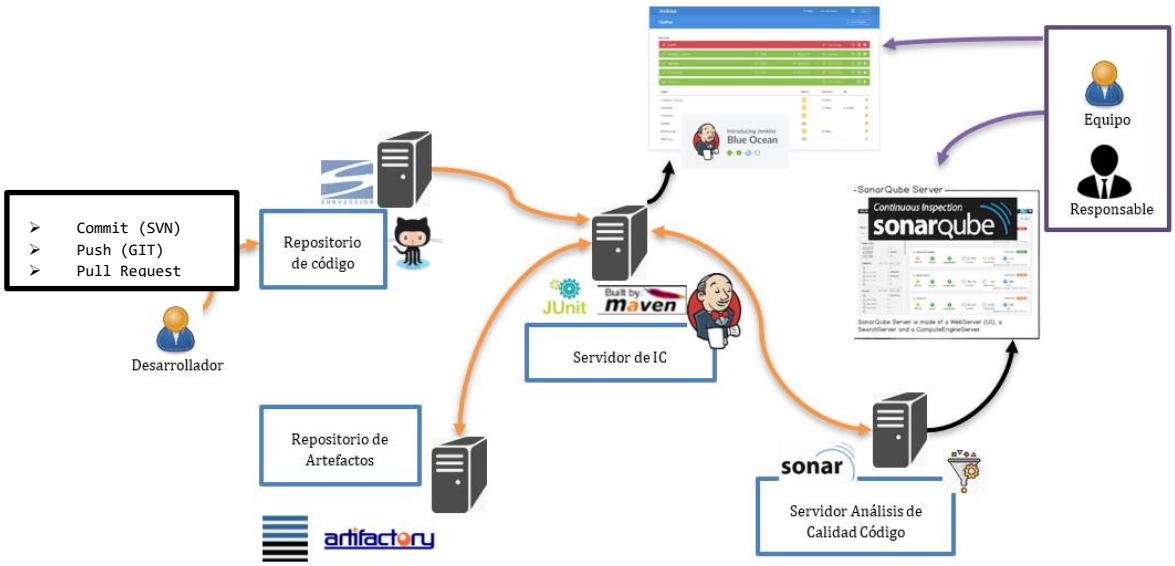

Imagen 1 Interacción entre herramientas IC y el equipo de desarrollo

2 CONCEPTOS IC

2.1 JOBS

Un Job o trabajo en Jenkins es el elemento que se utiliza para configurar tareas. La ejecución de un Job se denomina build o construcción. No confundir con la fase build de construcción de nuestro proyecto.

Los Jobs del servidor de CI, se pueden ejecutar manualmente, periódicamente y ejecutar automáticamente a través de disparadores o triggers, por ejemplo: Acción COMMIT al subir un desarrollador código a GIT.

2.2 PIPELINE

Un Job puede estar implementado a través de un pipeline o conjunto de pasos y tareas. Un Pipeline permite definir los pasos del ciclo de desarrollo a ejecutar. Este término también es usado en DevOps haciendo referencia a los pasos de un proceso de desarrollo y de operación. Los pipelines tienen una serie de stages o estadios y estos permiten una serie de steps o pasos. Cada Step permite incorporar scripts y programar un conjunto de acciones diferenciadas.

La imagen muestra la vista de un pipeline de un job ya ejecutado, build #5. En ella, el stage Checkout, contiene el conjunto de acciones para descargar el código del repositorio SCM al servidor de Jenkins. El stage Compilar y test, realiza las acciones de compilación usando Maven y realizar los test a través de Junit. El stage Sonarqube, lanzará el análisis de código estático y stage Post Actions, realizará acciones de notificación, envío de emails o avisos.

Imagen 2 Resultado de la ejecución de un pipeline en Jenkins

2.3 QUALITY GATES

El concepto de Quality Gate está asociado a la herramienta de análisis estático de código SonarQube. A través de ella, se puede definir una serie de controles de calidad, en base a los resultados obtenidos del análisis de código. Cuando se configura en el proceso de Integración continua, se pueden configurar unos mínimos bloqueantes. Si no se supera un mínimo de calidad de código la ejecución del pipiline puede configurarse para que se pare en ese punto.

Ejemplos de puertas de calidad:

- un mínimo del 80% de cobertura de pruebas sobre el código.
- un máximo de 10 días de deuda técnica.

2.4 EJEMPLO DE SECUENCIA TÍPICA DE TRABAJO

1. Un desarrollador se sube el código al repositorio de control de versiones GIT (*push*), SVN (*commit*)
2. El paso anterior dispara un Job en Jenkins.
3. En el entorno de Jenkins se invocará a un comando Maven para realizar la construcción del software (en función del comando invocado Maven ejecuta distintas

fases: clean, compile, install, test, deploy, ...) Para realizar la compilación se descargarán las librerías o dependencias del repositorio de artefactos (artifactory).

4. Para la fase de test Jenkins dispone de un plugin de JUnit para la realización de pruebas unitarias.
5. Jenkins también tiene un plugin para lanzar un análisis estático en SonarQube. Analizando el código para determinar su calidad, complejidad ciclomática, nº de comentarios o partes de código que no se ejecutan.
6. Jenkins realiza el despliegue en el servidor donde se van a realizar las pruebas.
7. Se realiza la ejecución automatizada de pruebas funcionales (Selenium).
8. Una vez validado todo, se sube al repositorio de artefactos la nueva versión recién validada.
9. Jenkins posee un historial de ejecuciones, para ver quién lo realizó y cuáles archivos fueron manipulados.
10. Si el resultado no es el esperado en alguno de los pasos o hay errores, Jenkins notificará al equipo de desarrollo, por email o cualquier otro medio (slack,…), para que lo solucione lo antes posible.

En la siguiente figura podemos observar un sistema de integración continua sencillo. El objetivo que perseguimos es que cada vez que un programador haga un *push* en el repositorio remoto (en este caso, Git) y suba código a la rama master, Jenkins lo detecte, compile la aplicación, llame a Sonar para ejecutar los test y realizar el análisis de la calidad del código y, si todo ha ido bien, publique un WAR del proyecto en Nexus.

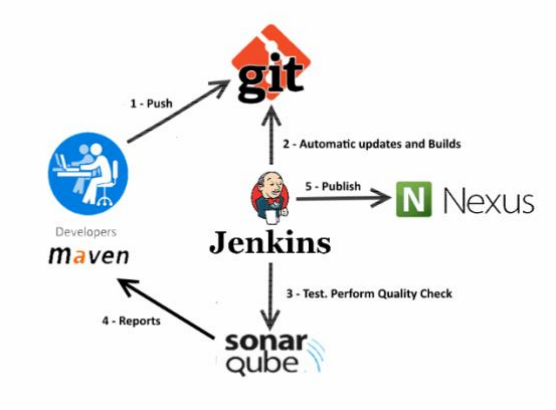

Imagen 3 secuencia típica IC

3 CI/CD - INTEGRACIÓN CONTINUA / ENTREGA CONTINUA / DESPLIEGUE CONTINUO

Podríamos decir que la implantación de CI tiene diferentes niveles de madurez, son también consideradas prácticas DevOps:

- **Integración continua (*Continuous Integration*)[CI]:**
  Se automatiza la integración con el control de versiones, la compilación, construcción, ejecución de tests unitarios, análisis estático de código, etc. para asegurar que el código subido al control de versiones no rompe nada. No llega a realizar tareas de despliegue, ni manuales ni automáticas.

- **Entrega Continua (*Continuous Delivery*) [CD]:**

  Va algo más allá, porque se automatiza el despliegue en un entorno como preproducción/staging y se automatiza el conjunto de pruebas de aceptación. Este paso puede realizarse a través de una

acción humana. Estamos en disposición de promocionar el código a un entorno, pero no se realiza de forma automática, sino que requiere de una aprobación.

- **Despliegue Continuo *(Continuous Deployment) [CD]:***
- Automatiza todos los pasos. Si se pasan todas las pruebas automatizadas en el anterior entorno, entonces se termina desplegando automáticamente en el entorno de Producción.
- Tratan de automatizar todo el proceso de entrega del software al usuario, eliminando toda acción manual o intervención humana
- La calidad de las pruebas (tanto unit testing pero sobre todo funcionales E2E) tiene que ser muy confiable para llegar a este nivel de madurez.
- Este tipo de Despliegue Continuo suele ir acompañado de despliegue Canary, para que, si se detectan problemas, se pueda hacer el rollback rápidamente, impactando al menor número de usuarios.

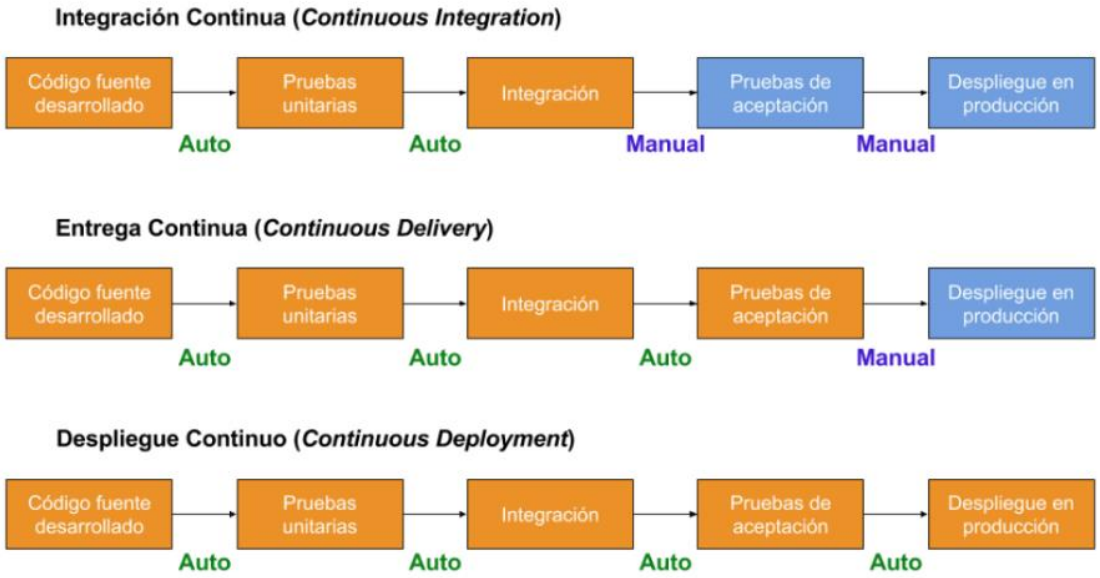

Imagen 4 Comparativa CI/CD

3.1 CaC Configuration as Code

El paradigma *As Code* se basa en ser capaz de reproducir y restaurar un entorno a través de instrucciones ya establecidas a través de instrucciones y automatización, gestionadas como código.

Configuration as Code trata de establecer la configuración de una determinada herramienta o entorno, a través de instrucciones fácilmente interpretables por un humano y que pueden ser ejecutadas de forma automática. La configuración como código se podrá versionar y controlar a través de una herramienta de control de versiones, siendo muy sencillo generar una *release* o hacer un *rollback* de una configuración determinada. Jenkins implementa esta estrategia a través de Jenkins Configuration as Code y utiliza archivos *JenkinsFile* con una sintaxis yaml.

4 DEVOPS

DevOps (acrónimo inglés de development -desarrollo- y operations -operaciones-) es un conjunto de prácticas que agrupan el desarrollo de software (Dev) y las operaciones de TI (Ops). Su objetivo es hacer más rápido el ciclo de vida del desarrollo de software y proporcionar una entrega continua de alta calidad.

La entrega continua y DevOps tienen objetivos comunes y a menudo se usan en conjunto, pero hay diferencias, DevOps también se centra en el cambio de la organización para adquirir más colaboración y eliminar silos y problemas entre equipos como los que pueda haber entre Equipos de desarrollo y Equipos de sistemas/operación.

4.1 IAC INFRASTRUCTURE AS CODE

Asociado a DevOps, se encuentra el concepto **Infraestructura como código (IaC):** Trata las infraestructuras como un software y esta configuración se versiona a través de las herramientas de SCM.

4.2 SERVIDORES MUTABLES E INMUTABLES

El concepto de servidor inmutable significa que, una vez montados los servidores, estos no se cambian sobre el entorno. Si tenemos que cambiar de servidor lo que haremos será montar ese nuevo servidor en un nuevo entorno. Gracias a IaC realizar esta operación será relativamente rápido y seguro.

Como desventaja, cada herramienta utiliza su propio DSL (Domain Specific Language), lenguajes declarativos o dialectos, basados en YAML, JSON y derivados del JSON para la definición de la infraestructura.

4.3 HERRAMIENTAS IAC

- **Terraform** es una herramienta declarativa de aprovisionamiento y orquestación de infraestructura que permite automatizar el aprovisionamiento de todos los aspectos de su infraestructura empresarial basada en la nube y en las instalaciones
- **Chef** es una de las herramientas de gestión de la configuración más populares que las organizaciones utilizan en sus procesos continuos de integración y entrega. Chef con muchos proveedores de servicios de nube como AWS, Microsoft Azure, Google Cloud Platform, OpenStack.
- **Puppet** es otra herramienta popular de administración de configuración. Usando Puppet, puedes definir el estado final deseado de tu infraestructura y exactamente lo que quieres que haga. Luego Puppet automáticamente refuerza el estado deseado y arregla cualquier cambio incorrecto. Puppet se integra con los principales proveedores de cloud computing como AWS, Azure, Google Cloud y VMware, lo que le permite automatizar en varias nubes.
- **Ansible** es una herramienta de automatización de infraestructuras creada por Red Hat. Ansible modela su infraestructura describiendo cómo se relacionan sus componentes y el sistema entre sí, en lugar de gestionar los sistemas de forma independiente.
- **AWS CloudFormation** es un servicio Amazon que ayuda a modelar y configurar recursos de Amazon Web Services, Permite crear una plantilla que describa todos los recursos de AWS que desea (como instancias Amazon EC2 o instancias de base de datos de Amazon RDS) y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.
- **Pulumi** es un framework de código abierto que permite crear y gestionar infraestructura y aplicaciones sobre proveedores de nube pública utilizando lenguajes de programación comunes y ampliamente extendidos, como Typescript, Javascript, Python, Go y C#
- **Vagrant –** permite creación y configuración de entornos virtualizados. Originalmente se desarrolló para VirtualBox y sistemas de configuración tales como Chef, Salt y Puppet. Sin embargo desde la versión 1.1 Vagrant es capaz de trabajar con múltiples proveedores, como VMware, Amazon EC2, LXC,

DigitalOcean, etc.Aunque Vagrant se ha desarrollado en Ruby se puede usar en multitud de proyectos escritos en otros lenguajes, tales como PHP, Python, Java, C# y JavaScript

5 DEVSECOPS

Es un aumento de DevOps, incluyendo prácticas de seguridad a lo largo de todo el ciclo. El modelo tradicional de equipo de seguridad centralizado debe adoptar un modelo federado que permita a cada equipo de Delivery tener en cuenta los controles de seguridad correctos en sus prácticas de DevOps.

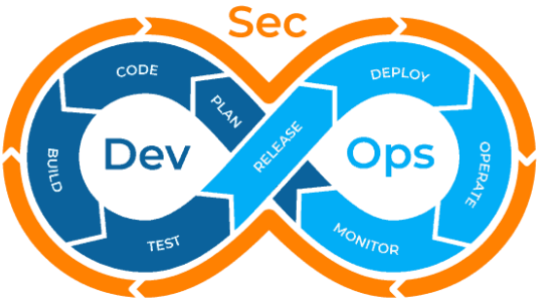

Imagen 5 DevSecOps

Asociado a DevSecOps, existe el concepto Seguridad como código (SaC): cuando se incorporan la seguridad al desarrollo y las operaciones de herramientas informáticas o aplicaciones.

Los beneficios de la implementación de DevSecOps son:

- Desarrollo de software seguro desde el diseño.
- Mayor velocidad y agilidad para los equipos de seguridad. Que también participan en el ciclo DevOps, evitando Silos y trabajando desde el inicio.
- Mejor comunicación entre equipos.
- Capacidad de respuesta rápida a los cambios.
- Identificación anticipada de posibles ataques y vulnerabilidades.

Imagen 6 Tareas de Seguridad en DevOps

5.1 HERRAMIENTAS

Existen muchas herramientas que dan soporte a las funcionalidades DevSecOps, se indican las más relevantes:

PLAN	CODE	BUILD	REPOS	CI	TEST	DEPLOY & RELEASE	RUNTIME	OPERATE	MONITOR
JIRA SLACK TAIGA REDMINE	SVN GITHUB GITLAB BITBUCKET	GRADLE MAVEN ANT MSBUILD	NEXUS ARTIFACTORY	JENKINS BAMBOO CIRCLECI GITLAB ACTIONS GITHUB ACTIONS AMIGO GO CD NEVERCODE CODESHIP	JUNIT SONARQUBE SOAPUI JMETER SELENIUM CYPRESS	CLOUDBEES JENKINS RUNDECK KUBERNETES ANSIBLE PUPPET CHEF CAPISTRANO OPENSTACK OPENNEBULA CLOUDSTACK	OPENSHIFT DOCKER	ANSIBLE PUPPET CHEF KUBERNETES	APPDYNAMICS SPLUNK BLUEPILL MCOLLECTIVE GRAHITE GRAFANA GRAYLOG CFENGINE ZABHIS NAGIOS
SEC									
JIRA	GITROD CHECKMARX		SONATYPE NESSUS TANIUM	PLUGINS DEPENDENCY CHECK – JENKINS SPLUNK	VULNERABILITIES (SONARQUBE) OWASP RULES (SONARQUBE) FINDBUGS	INSPECT BEAKER		OWASP ZAP	FORTYFY SPLUNK FIREEYE METASPOILT EVIDENT.IO

Tabla 1 Herramientas DEVSECOPS

6 SRE

SER o Site Reliability Engineering es un concepto creado por Ben Treynor en Google para implementar la filosofía DevOps y fomentar la fiabilidad, responsabilidad e innovación de los productos.

SRE es un sistema en el que el desarrollo controla las operaciones, donde el entorno se divide en los componentes más básicos de la pila de TI y se implementa con las mejores prácticas integradas en el hardware.

La principal diferencia entre DevOps y SRE, es que SRE se maneja más operativamente desde arriba hacia abajo, y está gobernado por el desarrollador o el equipo de desarrollo, en lugar del equipo de operaciones. Los desarrolladores tienen más control sobre los procesos de monitoreo y mantenimiento del software, dos trabajos que normalmente son administrados por el equipo de operaciones.

DevOps tiene como objetivo cerrar la brecha entre el desarrollo y las operaciones al alinear culturalmente sus tareas, objetivos e iniciativas; SRE coloca al equipo de desarrollo a la cabeza de toda la iniciativa.

DevOps y SRE comparten los mismos objetivos finales:

- Hacer cambios incrementales de forma rápida y eficiente.
- Reducir el número de silos de organización.
- Tener una cultura de trabajo flexible, abierta y adaptable.
- Utilice la automatización siempre que sea posible
- Para monitorear el rendimiento y mejorar cuando sea necesario.

7 ESTRATEGIAS DE DESPLIEGUE

Esta sección como tal no está dentro del Concepto de Integración Continua, pero está íntimamente relacionado. De cara a los despliegues se pueden seguir varias estrategias para maximizar el éxito.

7.1 BLUE/GREEN DEPLOYMENT

En esta estrategia se tiene un entorno de producción oculto, en el que se despliega la nueva versión del código. Tras las pruebas en el entorno oculto, se procede a mover todo el tráfico de usuario mediante el balanceador de carga. El beneficio es un Zero un despliegue sin pérdida de servicio (*Zero Downtime)*

7.2 DOG FOODING

Técnicamente similar al anterior, pero se deja acceder a las nuevas funcionalidades a los empleados internos. Para recoger feedback interno antes de abrir a usuarios finales.

7.3 CANARY DEPLOYMENT

En vez de abrir las nuevas funcionalidades a todos los usuarios finales, se hace de forma progresiva y controlada. Por ejemplo, se abre la nueva funcionalidad a sólo el 5% de usuarios finales y toma feedback antes de seguir abriendo progresivamente al resto de usuarios.

8 HERRAMIENTAS IC

A continuación, vamos a describir con mayor detalle algunas de las herramientas más populares que se utilizan a lo largo del ciclo de integración continua.

8.1 CONTROL DE VERSIONES

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante.

Entre dos revisiones sólo guardan el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco.

No sólo sirve como repositorio de código (que es el uso típico), también puede ser usado para guardar los archivos de tests, archivos configuración de los servidores, … cualquier archivo sobre el que se quiera tener un control de trazabilidad de cambios.

8.1.1 GIT

Git es un sistema de control de versiones **distribuido** cuyo principal objetivo es ayudar en el desarrollo de cualquier tipo de aplicación manteniendo una gran cantidad de código de un gran número de programadores diferentes.

Esta herramienta fue impulsada por Linux Torvalds y el equipo de desarrollo del Kernel de Linux, que al igual que este sistema operativo, lo lanzaron como código abierto.

Git ofrece:

- Auditoría completa del código, sabiendo en todo momento quién ha tocado algo, cuándo y qué.
- Control sobre cómo ha ido cambiando nuestro proyecto con el paso del tiempo.
- Volver uno o más pasos hacia atrás de forma rápida.
- Control de versiones del proyecto por medio de etiquetas.
- Seguridad, ya que todas las estructuras internas de datos irán *"hasheadas"* con el algoritmo SHA1.

8.1.2 SUBVERSION (SVN)

Subversión o SVN es un sistema de control de versiones centralizado que se asemeja a un sistema de ficheros. En los últimos SVN ha ido perdiendo mucha cuota de mercado para dejar a GIT convertirse en un standard de facto.

8.1.3 GITHUB

No hay que confundir GITHUB con GIT. GitHub.com es una empresa que ofrece un GIT en la nube como SaaS. Github ofrece funcionalidades más allá del control de versiones, como ciertas funcionalidades de Integración Continua.

8.1.4 GITLAB

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto. Se puede usar en modo nube y on-premise.

8.2 SERVIDORES DE INTEGRACIÓN CONTINUA

El servidor de integración continua es el elemento clave de un sistema de integración continua, ya que va a coordinar la ejecución de las otras herramientas. A continuación, nombramos brevemente alguno de ellos.

- **Jenkins**: software de integración continua open source. Es uno de los más usados y surgió como un *branch* del proyecto Hudson.
- **Bamboo**: es una solución propietaria de integración continua y despliegue que reúne compilaciones, pruebas y versiones automatizadas en un solo flujo de trabajo. Muy bien integrado con herramienta JIRA de Atlassian.
- **CruiseControl**: aplicación de código abierto basado en Java que cada cierto tiempo, o cuando hay cambios en el gestor de versiones (por ejemplo CVS o Subversion), hace una compilación y ejecuta tests (más cualquier otra cosa que esté configurada en Ant o Maven) y una vez acaba presenta el resultado en HTML, por correo electrónico, RSS, Jabber/XMPP, etc.
- **Continuum**: sistema complementario de Apache Maven que ejecuta compilaciones de acuerdo con una calendarización configurable. De manera semejante a CruiseControl.
- **Hudson**: alternativa a CruiseControl. Puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como también *shell scripts* o procesos por lotes Windows.
- **Solano CI:** solución de integración continua ofrecida como SaaS. Propietario.

Para **proyectos .Net**

- **CruiseControl.Net**: la versión de CruiseControl para el mundo .Net implementado en Framework.net y de código abierto.
- **Team Foundation Build**: es la solución integrada en Visual Studio en Team Foundation server. Permite configurar una compilación para cada check-in, compilaciones de la rama o ramas que se deseen, la ejecución de las pruebas del listado de pruebas de proyectos de Visual Studio, el etiquetado del código fuente en relación a esta compilación, etc.

8.3 HERRAMIENTAS DE CONSTRUCCIÓN DE PROYECTOS

El objetivo de estas herramientas es la construcción del software: descarga de dependencias, compilado del código, paquetizado, etc.

8.3.1 APACHE ANT

Apache Ant es una herramienta usada en programación para automatizar las tareas repetitivas y mecánicas como la construcción de paquetes y de proyectos. Ant utiliza un fichero en formato xml denominado *build.xml*. Maven y Gradle están tomando más relevancia.

8.3.2 MAVEN

Maven es un framework para la gestión y construcción de proyectos software de la Apache Software Foundation.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Cada proyecto tiene la información para su ciclo de vida en el descriptor XML, por defecto el fichero pom.xml.

Maven está construido usando una arquitectura basada en plugins que le permite utilizar cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, … y cualquier lenguaje.

Las partes del ciclo de vida principal del proyecto Maven son:

- **compile**: genera los ficheros *.class* compilando fuentes .java.
- **test**: ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
- **package**: genera el fichero .jar con los *.class* compilados.
- **install**: copia el fichero .jar a un directorio de nuestro ordenador donde Maven deja todos los .jar (repositorio local). De esta forma, esos .jar pueden utilizarse en otros proyectos Maven en el mismo ordenador.
- **deploy**: copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto Maven con acceso a ese servidor remoto.

Los objetivos perseguidos con Maven son:

- Facilitar el proceso de construcción.
- Proporcionar un sistema de construcción uniforme. Todos los proyectos Maven funcionan de la misma forma, por lo que el esfuerzo de aprendizaje sólo se hace una vez.
- Proporcionar información útil sobre el proyecto: lista de cambios desde el control de versiones, dependencias transitivas, informes de la ejecución de pruebas unitarias, etc.
- Ayudar a utilizar las “mejores prácticas” de desarrollo.
- Permitir introducir nuevos servicios de forma sencilla.
  https://maven.apache.org/

8.3.3 GRADLE

Gradle es un sistema de automatización de construcción de código de software que construye sobre los conceptos de Apache Ant y Apache Maven e introduce un lenguaje específico del dominio (DSL) basado en Groovy en vez de la forma XML utilizada por Apache Maven para declarar la configuración de proyecto.

Gradle utiliza un grafo acíclico dirigido ("DAG") para determinar el orden en el que las tareas pueden ser ejecutadas. Gradle fue diseñado para construcciones multi-proyecto las cuales pueden crecer para ser bastante grandes, y da apoyo a construcciones incrementales determinando inteligentemente qué partes del árbol de construcción están actualizadas, de modo que cualquier tarea dependiente a aquellas partes no necesitarán ser reejecutada.

Los plugins iniciales están principalmente centrados en el desarrollo y despliegue en Java, Groovy y Scala, pero existen más lenguajes en proceso de incorporación.

https://gradle.org/

8.4 REPOSITORIOS DE ARTEFACTOS

Los repositorios de artefactos juegan un papel importante en los ciclos de vida de integración continua. Son útiles tanto para subir y compartir librerías de terceros (que podemos utilizar en nuestros proyectos como dependencias) como para publicar los artefactos que generemos de nuestra aplicación y que posteriormente serán desplegados en cualquier entorno (preproducción, producción,…)

Los Repositorios de Artefactos también soportan el concepto de versiones (para un artefacto puede haber diferentes versiones). Pero no confundir con un software de Control de Versiones como GIT.

Los repositorios de artefactos como Nexus o Artifactory se utilizan para no tener que depender de algo externo siempre y poder mantener controladas las librerías propias de la organización, como puede verse en la siguiente figura:

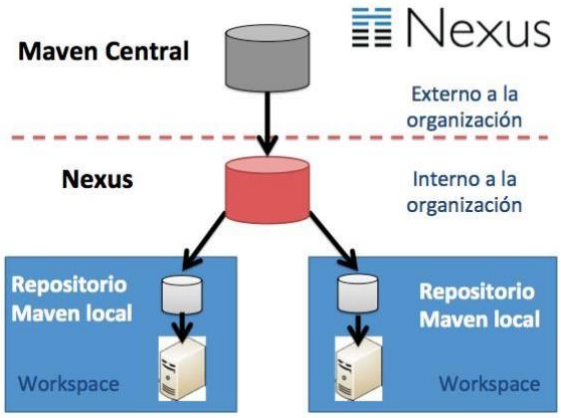

Imagen 7 Interacción repositorio Central, Local, Externo e interno a la organización

Este tipo de repositorios también se utilizan como cachés. La idea de tener un Nexus o Artifactory, es que sean repositorios de librerías internos en la organización. Al estar en una red local, el acceso al Nexus interno será más rápido que tener que conectarse a Maven Central para descargar las librerías.
Además, ese Nexus es compartido por toda la organización y el equipo de desarrollo.

Los repositorios de artefactos más extendidos son: **NEXUS / ARTIFACTORY / Archiva**

8.5 SONARQUBE

Herramienta de análisis de código en base a un conjunto de reglas adoptadas por la comunidad. El análisis nos indica la calidad de nuestro código, a través de reglas de calidad, nos indica evidencias de tipo: bugs, code smells y vulnerabilities, para cada evidencia encontrada ofrece posibles soluciones. También, califica la calidad del código, valor que se actualiza continuamente tras cada modificación del código del proyecto, por lo que puede evolucionar tanto positiva como negativamente. Esto permite supervisar de forma continua la calidad del código del software de cada proyecto, y analizar tendencias.

Sonar evalúa aspectos como la complejidad ciclomática del código (número de caminos independientes). Asimismo, informa sobre deuda técnica, código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, posibles errores y diseño del software.
SonarQube ofrece una versión Community, open source, y versiones de pago que incluyen más lenguajes a analizar y funcionalidades, SonarQube Developer, SonarQube Enterprise y SonarQube DataCenter.
Además, se ofrece en modo SasS, a través de SonarCloud, que es gratuita para proyectos bajo licencias open source.
Ofrece un plugin Sonarlint instalable en los principales IDEs que permite revisar el estado del análisis desde el entorno local del desarrollador.

8.6 SELENIUM

Es un framework para la automatización de navegadores, que puede emplearse para la realización de cualquier actividad repetitiva que requiera de la interacción a través del navegador, como pueden ser los test de aplicaciones web o ciertas tareas administrativas. En un entorno de integración continua se utilizará para automatizar las pruebas de regresión de aplicaciones web que requieran del uso del navegador, permitiendo reducir el esfuerzo en realizar las pruebas con diferentes configuraciones y tipos de navegadores.

Selenium provee una herramienta de grabar/reproducir para crear pruebas sin usar un lenguaje de scripting para pruebas (Selenium IDE). Incluye también un lenguaje específico de dominio para pruebas (Selenese) para escribir pruebas en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python.

En el caso de **Aplicaciones Nativas para Móvil**, existen otros frameworks de automatización de interfaz como son Espresso o Apium.

8.7 CYPRESS

Cypress es un framework de testing todo en uno. Es rápido, fácil de usar y permite ejecutar pruebas sobre cualquier aplicación web.
En Cypress las pruebas se ejecutan dentro del navegador controlándolo a través de un proceso backend en NodeJS. Además, funciona a nivel de red permitiendo la intercepción de todo el tráfico entrante y saliente de nuestra aplicación. Como las pruebas se ejecutan dentro del navegador, dispone del acceso nativo a todas las APIs como window, document, etc. Además, la ejecución es muchísimo más rápida que en otras herramientas alternativas.
Su instalación es muy simple, solo se necesita agregar una sola dependencia con NPM. Sin perder tiempo en configuración, conexión con un servidor remoto, etc, como requería Selenium

Usa frameworks y librerías front-end conocidos como Mocha, Chai, Sinon, Lodash, jQuery, etc. A diferencia de otras herramientas Cypress dispone de una interfaz gráfica que permite ver de forma

interactiva cada uno de los pasos y las acciones ejecutadas durante la prueba, el estado de la aplicación, la duración de la prueba, los test fallidos o pasados.
Además de otras ventajas como:

- Espera automática de los elementos (DOM sin cargar por completo, framework sin terminar de arrancar, petición AJAX sin completar, etc.). Lo cual minimiza los falsos positivos en las pruebas fallidas.
- Grabación automática del navegador en la ejecución de las pruebas. Lo cual permite ver en qué estado estaba la aplicación en cada uno de los pasos
- Spies, stubs y mocks que además de las pruebas E2E permite hacer pruebas unitarias
- Intercepción de las peticiones AJAX / XHR para hacer aserciones o crear fixtures o mocks
- Capturas de pantalla y videos automática
- Depuración nativa con instrucción debugger
- Velocidad, la ejecución pruebas comienzan tan rápido como se cargue la aplicación
- Documentación muy completa
- Mensajes de error claros
- Extensible a través de plugins que hace esta herramienta aún más potente. Plugins para Redux para hacer aserciones y Cucumber para implementar BDD.