INTEGRACIÓN CONTINUA CON JENKINS

Curso

Contenido

Introducción a Jenkins

Jenkins es un servidor de integración continua, gratuito, open-source y actualmente uno de los más empleados para esta función.

Esta herramienta, proviene de otra similar llamada Hudson, ideada por Kohsuke Kawaguchi, que trabajaba en Sun. Unos años después de que Oracle comprara Sun, la comunidad de Hudson

decidió renombrar el proyecto a Jenkins, migrar el código a Github y continuar el trabajo desde ahí. No obstante, Oracle ha seguido desde entonces manteniendo y trabajando en Hudson.

La base de Jenkins son las tareas, donde indicamos qué es lo que hay que hacer en un build. Por ejemplo, podríamos programar una tarea en la que se compruebe el repositorio de control de versiones cada cierto tiempo, o el mismo servidor de control de versiones notifique a Jenkins que hay cambios realizados y cuando un desarrollador quiera subir su código al control de versiones, este se compile y se ejecuten las pruebas.

Si el resultado no es el esperado o hay algún error, Jenkins notificará al desarrollador, al equipo de QA, por email o cualquier otro medio, para que lo solucione. Si el build es correcto, podremos indicar a Jenkins que intente integrar el código y subirlo al repositorio de control de versiones.

Pero la cosa no queda ahí. Una de las cosas buenas que tiene Jenkins es que además de poder ayudarte a integrar el código periódicamente, puede actuar como herramienta que sirva de enlace en todo el proceso de desarrollo.

Desde Jenkins podrás indicar que se lancen métricas de calidad y visualizar los resultados dentro de la misma herramienta. También podrás ver el resultado de los tests, generar y visualizar la documentación del proyecto o incluso pasar una versión estable del software al entorno de QA para ser probado, a pre-producción o producción.

Otra de las ventajas es que posee un historial de cambios realizados por build o versión, saber quién lo realizo y cuales archivos fueron manipulados, y sus comentarios al respecto.

Por si todo esto no fuera suficiente, se trata de una herramienta que se puede instalar en plataformas windows, linux o mac. Su instalación y posterior mantenimiento es muy sencillo, como es habitual con las herramientas desarrolladas en Java. No necesita ninguna base de datos. La interfaz es una interfaz web, desde la que accederemos a todos los controles y ajustes que necesitemos. No hay necesidad de ajustar manualmente ningún XML más, aunque si deseamos hacerlo, también podemos hacerlo así.

Y para finalizar las principales tareas (pero no las únicas) de las que se encargará Jenkins son:

- Monitoreo constante de los cambios en servidor de control de versiones.
- Compilar el código.
- Ejecutar las pruebas.
- Notificación de errores que se hayan detectado tras las ejecuciones de pruebas, por ejemplo, vía mail, twitter, chat, etc.
- Generación y publicación de binarios.
- Ejecución de métricas de calidad y visualización de los resultados.
- Generación de documentación asociada a un proyecto.

REFERENCIAS:

- HTTPS://WWW.MARTINFOWLER.COM/ARTICLES/CONTINUOUSINTEGRATION.HTML
- HTTPS://WWW.ATLASSIAN.COM/CONTINUOUS-DELIVERY/CI-VS-CI-VS-CD
- HTTPS://STACKOVERFLOW.COM/QUESTIONS/28608015/CONTINUOUS-INTEGRATION-VS-CONTINUOUS-DELIVERY-VS-ONTINUOUS-DEPLOYMENT
- HTTPS://ABOUT.GITLAB.COM/2014/09/29/GITLAB-FLOW/

¿Por qué la integración continua?

Existe una creciente exigencia desde las Áreas de Negocio y que abarca a todos los sectores para hacer los desarrollos informáticos de una forma muy rápida; deben ser puestos en producción, probar los resultados, el efecto en los clientes y volver a modificar para ajustar o entregar nuevas funcionalidades.

Estas necesidades han fomentado la utilización, cada vez mayor, de modelos tipo "AGILE" (Scrum, Kanban, Lean…), que permiten realizar desarrollos en equipos reducidos, gestionados de formar independiente y orientados a la entrega de funcionalidades nuevas en ciclos muy cortos.

Hay un hecho diferencial que ha hecho posible que esta nueva forma de trabajo sea aplicable de forma sencilla: la aparición de Herramientas para Integración Continua.

Para poder realizar ciclos de desarrollos en plazo de días es preciso hacer las cosas de forma totalmente diferente, no se puede mantener el control y garantizar que todo se hace con la calidad adecuada sin las Herramientas precisas.

La Integración continua es una práctica de desarrollo descrita por Martín Fowler según la cual, en lugar de trabajar de forma independiente, y cada componente del equipo por separado para, y, finalmente, integrar el resultado (con los consiguientes problemas de incompatibilidades que hay que resolver) los equipos de trabajo pueden estar realizando integraciones de forma continua. La integración se garantiza por las herramientas de construcción automatizada, asegurando que la integración en correcta y realizando las pruebas también de forma automatizada para detectar posibles errores de integración de la forma más rápida posible. Esta forma de trabajo permite a un equipo de trabajo desarrollar software de forma rápida y a la vez garantizando que se mantiene la coherencia entre los trabajos realizados por los distintos miembros del equipo.

- La mayor ventaja es evitar el caos de última hora. Un fallo en un sistema acabado no evaluado supone analizar todas las piezas que lo forman. Testeando periódicamente conseguimos señalar fallos en el momento de cometerlos, asegurando el resultado final.
- Disponemos en todo momento de una visión clara de la fase en la que nos encontramos, ya que revisamos cada tarea realizada.
- Posibilidad de testar tanto piezas aisladas, como la integración de varios componentes.
- Disponemos de métricas y datos relativos al proyecto desde el momento en el que comenzamos a evaluar.
- Estas ventajas se pueden resumir en un mayor control de la información y la anulación de errores en las fases finales del proyecto.

En resumen, la integración continua nos permite realizar un análisis continuo y una detección precoz de los errores de un proyecto, mediante un punto de vista Ágil. Implementar este tipo de proceso es muy recomendable en todo tipo de modelos de negocio.

Requisitos de la integración continua

Ahora ya disponemos de una visión más concreta de lo que es la integración continua, pero nos vendrá bien repasar cuáles son los elementos indispensables para hacerlo correctamente.

- ## Repositorio de código

El repositorio de código es la herramienta fundamental que permite que el equipo trabaje de forma totalmente sincronizada, pero a la vez sencilla. Cada uno trabaja en su parte y puede actualizar los cambios sin necesidad de que otros desarrolladores tengan que esperar por estos y viceversa.

Desgraciadamente todavía hay muchas empresas que aún no poseen una herramienta de este tipo. Para aquellos que aún no utilicen un repositorio de código cabe destacar que actualmente existen una gran cantidad de soluciones gratuitas de calidad, como pueden ser Subversion, Git o Mercurial por mencionar algunas. Esta es una pieza fundamental y será prácticamente imposible realizar integración continua si no se dispone de ella.

Este tipo de herramientas marcan un antes y un después en el trabajo cotidiano de un equipo de desarrollo de software. Pero además es el pivote fundamental de la integración continua, que no es posible sin un repositorio de código.

- ## Sistema de construcción automatizado

Muchos equipos utilizan complejos entornos de desarrollo para implementar y compilar los proyectos. Esto en sí mismo no es malo, pero debemos poder construir el proyecto en cualquier máquina sin necesidad de dichos entornos. No todas las máquinas en las que vayamos a instalar nuestro proyecto tendrán el entorno de desarrollo con el que trabajamos, incluso en algunos casos estos pueden tener licencias relativamente caras, por lo que no sería una opción viable.

Hay muchas soluciones de construcción diferentes en función de los lenguajes de programación, en Java están Ant o **Maven**, en Ruby está Rake, solo por mencionar algunas. La condición es que debe ser una herramienta muy sencilla que se pueda ejecutar en cualquier máquina y que consiga automatizar no solo el proceso de compilación, sino la ejecución de los tests de nuestra aplicación e incluso el despliegue en el servidor de aplicaciones llegado el caso.

- ## Commits diarios

No todos los requisitos se basan en tener un determinado tipo de herramienta, algunos se basan en saber cómo utilizar una determinada herramienta. Una vez que tenemos un control de versiones hay que recordar que el almacenamiento de los cambios debe ser diario, idealmente un desarrollador debe hacer incluso varias subidas al repositorio de código al día.

Si tenemos un repositorio de código, pero cada desarrollador sube sus cambios cada dos semanas seguimos en el mismo problema de antes, la integración del proyecto se retrasa durante todo ese tiempo, perdiendo toda opción de obtener información inmediatamente. Además, retrasar la integración tanto tiempo genera más conflictos entre los ficheros editados por los desarrolladores. Debido a que con cada actualización del repositorio se hace una integración del proyecto completo, cada mínimo cambio que no desestabilice el proyecto debe ser subido para que se compruebe que todo sigue funcionando correctamente.

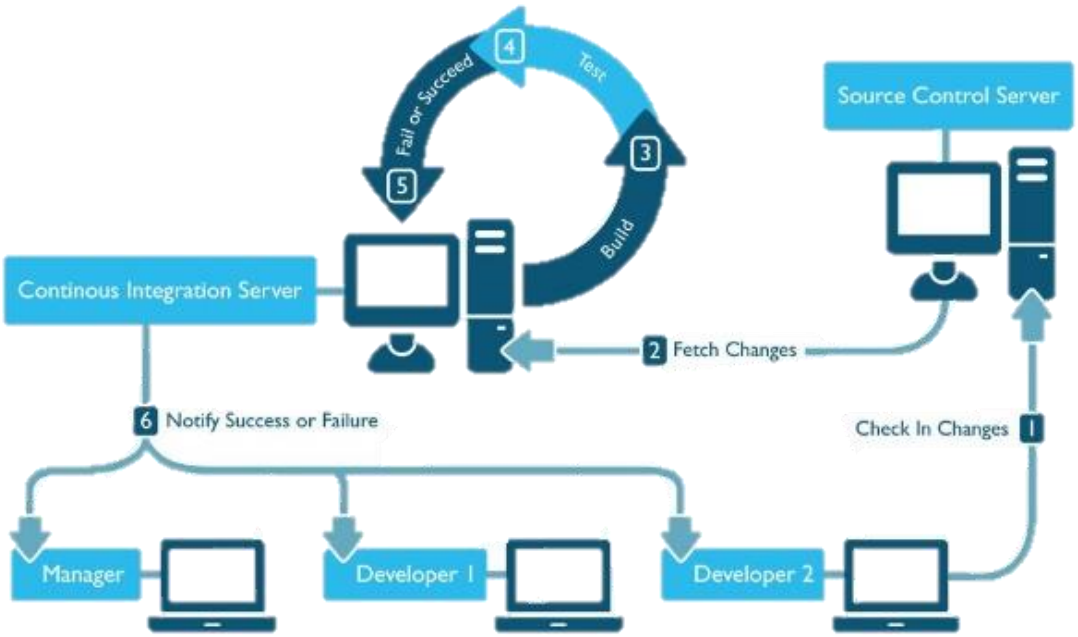
- ## Pruebas automatizadas

Hemos hablado todo el tiempo de comprobar que el proyecto funciona correctamente y al mismo tiempo de automatizar todo el proceso lo mejor posible. Para poder comprobar que el proyecto funciona de forma automática necesitamos pruebas automatizadas.

Existen muchas maneras diferentes de incluir tests en el proyecto, personalmente creo que TDD es la manera más adecuada, pero si el equipo no tiene experiencia y crea las pruebas después de haber realizado la implementación tampoco es un método inválido. La premisa fundamental es que toda nueva funcionalidad debe tener una batería de pruebas que verifique que su comportamiento es correcto.

- Servidor de integración

Esta es la pieza más polémica, mucha gente cree que no es absolutamente necesaria para hacer integración continua correctamente, por ejemplo, algunos equipos hacen la integración de forma manual con cada subida al repositorio de código. En mi opinión es un paso tan sencillo y barato de automatizar que no merece la pena ahorrárselo. Montar algún servidor de integración, como por ejemplo **Jenkins**, es gratuito y tan sencillo como desplegar un fichero en un servidor. Por contra las ventajas son grandísimas. Para empezar el proceso está totalmente automatizado, lo que evita el error humano. Y por otro lado reduce la cantidad de trabajo, ya que tenemos una solución prefabricada sin necesidad de tener que crear nosotros mismos complejas soluciones caseras.



Un paso más allá -

Con los pasos descritos hasta el momento ya tendríamos un proceso bastante completo y que a buen seguro mejorará enormemente la calidad de nuestro producto. Pero podemos ir un poco más allá y utilizar el servidor de integración para que haga ciertas tareas relativamente complejas por nosotros.

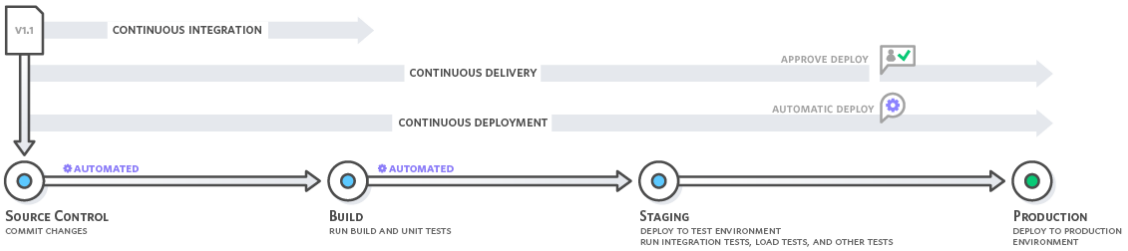
Entrega Continua o Continuous Delivery

La entrega continua es una práctica de desarrollo de software mediante la cual se crean, prueban y preparan automáticamente los cambios en el código y se entregan para la fase de producción. Amplía la integración continua al implementar todos los cambios en el código en un entorno de pruebas y/o de producción después de la fase de creación. Cuando la entrega continua se

implementa de manera adecuada, los desarrolladores dispondrán siempre de un artefacto listo para su implementación que se ha sometido a un proceso de pruebas estandarizado.

Despliegue Continuo o Continuous Deployment

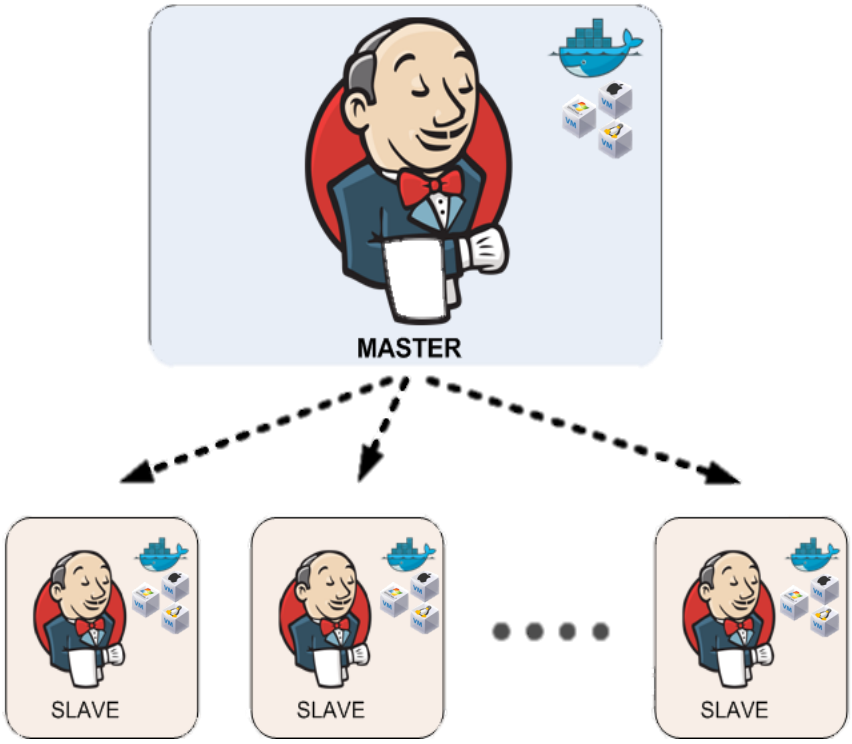
Otra operación que podemos automatizar, y que el servidor de integración podría hacer por nosotros, es la realización de pruebas de sistema sobre la aplicación. Imaginemos que estamos desarrollando una aplicación web, podríamos crear tests con alguna herramienta de grabación de la navegación, como por ejemplo Selenium, y lanzarlos con cada construcción que haga el servidor. Es un tipo de prueba que requiere mucho tiempo y no sería viable que se lancen con cada compilación del desarrollador, pero para el servidor de integración no habría ningún problema. Este es solo un ejemplo más de la cantidad de cosas que puede hacer un servidor de integración continua por nosotros, y que nos ayudará a mantener un producto estable y testeado de manera totalmente automática.



REFERENCIAS:

- HTTPS://WWW.MARTINFOWLER.COM/ARTICLES/CONTINUOUSINTEGRATION.HTML
- HTTPS://WWW.ATLASSIAN.COM/CONTINUOUS-DELIVERY/CI-VS-CI-VS-CD
- HTTPS://STACKOVERFLOW.COM/QUESTIONS/28608015/CONTINUOUS-INTEGRATION-VS-CONTINUOUS-DELIVERY-VS-ONTINUOUS-DEPLOYMENT
- HTTPS://ABOUT.GITLAB.COM/2014/09/29/GITLAB-FLOW/

Distribución de construcciones en diferentes sistemas



Jenkins permite distribuir la carga de trabajo. Además, permite asignar nodos de ejecución a nivel de Job. Es habitual encontrar la palabra esclavos (slaves) o nodos (nodes) para referirse a los agentes. En Jenkins estos conceptos son sinónimos.

Para que una máquina sea reconocida como nodo esclavo, necesita ejecutar un programa agente específico que establecerá una comunicación bidireccional con el nodo maestro.

Hay diferentes formas de establecer una conexión entre el nodo maestro y el nodo esclavo: Conexión SSH, Conexión JNLP-HTTP, Conexión JNLP-TCP.

Todas las conexiones ofrecen la configuración maestro/esclavo y tienen ventajas e inconvenientes. La selección de una de ellas debe estar basada en la arquitectura existente y las restricciones de conectividad que pueda haber

Conexión a través de SSH

Esta configuración es la escogida en la mayoría de los casos ya que permite establecer una comunicación estable entre maestro y esclavo. Inicialmente optamos por SSH para comunicar los dos nodos pero en Windows, SSH no aparece disponible de forma nativa y para utilizarlo hay que instalar el servicio a través de alguna herramienta que instale un servidor SSH en Windows, como por ejemplo sshd con cygwin, FreeSSHd,..

- Ventajas:
  - Conexión segura.
  - Gestión del nodo esclavo desde el maestro.
  - Actualización automática a nuevas versiones.
- Inconvenientes:
  - Uso de usuario/contraseña para la conexión.
  - Requiere la instalación de un servidor ssh en Windows.

Conexión vía Java Web Start (JNLP)

En este caso, la comunicación es establecida arrancando el agente del nodo esclavo mediante Java Web Start (JNPL). Con esta conexión el programa Java Web Start tiene que estar lanzado en el esclavo de forma manual o como un servicio. La conexión se realiza por JNLP/TCP utilizando un

puerto libre aleatorio, el puerto TCP a utilizar en este tipo de conexiones puede ser fijado a nivel de administración del nodo maestro de Jenkins.

- Ventajas:
  - Solución más sencilla de implementar.
  - Elimina el tráfico de red necesario para crear/gestionar el servicio desde el nodo maestro al esclavo.
- Inconvenientes:
  - Requiere que el programa Java Web Start esté arrancado en el nodo esclavo, pero una vez instalado puede ser configurado como servicio automático en el nodo esclavo.
  - Cuando se realiza una actualización en el master se debe actualizar manualmente los slaves.

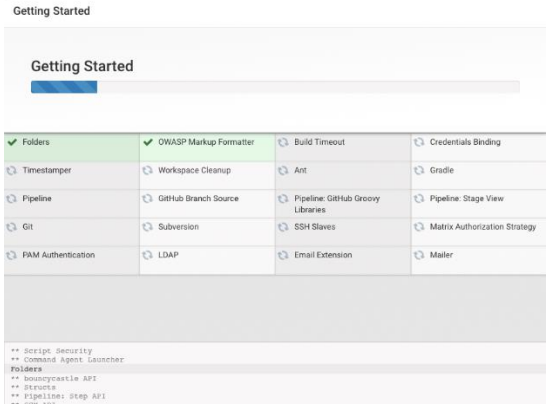
Configurando Jenkins

Instalación

Existen varias formas de instalar Jenkins, como standalone application corriendo en un servidor Java embebido, puede correr en cualquier contenedor de Servlets como Apache Tomcat, Glassfish, etc. También puede instalarse a partir de un contenedor de Docker.

Ejemplo Docker:

```
docker run -d -p 8080:8080 -p 50000:50000 -v /home/nobleprog/jenkins_home:/var/jenkins_home jenkins/jenkins:latest
```



REFERENCIAS:

- https://jenkins.io/doc/book/installing

Administrar Sistema

El área de administración de Jenkins se divide en las siguientes secciones

- Configurar el sistema:
  - Permite configurar variables globales, rutas, opciones de plugins, etc.
- Configuración global de la seguridad
  - Permite definir la estrategia de seguridad aplicada al sistema, quienes y a qué tienen acceso los usuarios.

- Configure Credentials
    - Permite definir los proveedores de seguridad para la gestión de credenciales.
- Global Tool Configuration
    - Permite configurar las tools que se utilizaran a nivel global con Jenkins. JDK, Maven, Git, etc.
- Actualizar configuración desde el disco duro
    - Permite descartar todos los datos cargados en memoria y actualizar todo nuevamente desde los ficheros del sistema. Es útil cuando se modifican ficheros de configuración directamente en el disco duro.
- Administrar Plugins
    - Permite instalar, desinstalar, actualizar, activar y desactivar los plugins que extienden la funcionalidad de Jenkins.
- Información del sistema
    - Muestra toda la información del sistema en cuanto a propiedades, variables de entorno definidas y el detalle de las versiones de todos los plugins instalados.
- System Log
    - Permite ver el log general del sistema.
- Estadísticas de Carga
    - Permite monitorear el uso de recursos de Jenkins.
- Jenkins CLI
    - Permite descargar una herramienta para gestionar Jenkins desde línea de comandos.
- Consola de scripts
    - Permite ejecutar en una consola online scripts de tipo Groovy.
- Administrar Nodos
    - Permite crear, eliminar, gestionar y monitorear los nodos de Jenkins.
- Gestión de Usuarios
    - Crear, eliminar o editar los usuarios registrados en Jenkins.
- Managed Files
    - Permite gestionar archivos estáticos (principalmente de configuración) que pueden ser utilizados para las tareas ejecutadas en Jenkins.
- Preparar Jenkins para apagar el contenedor
    - Permite apagar el sistema de forma segura finalizando todas las tareas en curso.

Instalación de complementos de Jenkins

Los complementos de Jenkins o plugins permiten extender la funcionalidad del sistema mejorando funcionalidades o dejándonos realizar nuevas acciones específicas para el usuario.

Al momento de la instalación de Jenkins nos da la opción de instalar complementos de los más utilizados, a estos los divide en las siguientes categorías los miles de plugins que tiene disponible en su catálogo oficial:

- Organization and Administration
- Build Features
- Build Tools
- Build Analysis and Reporting
- Pipelines and Continuous Delivery
- Source Code Management
- Distributed Builds
- User Management and Security
- Notifications and Publishing

En la sección de "Gestión de Plugins" del área administrativa de Jenkins nos da las opciones de:

- Actualizar plugins instalados
- Buscar en instalar plugins desde el catálogo oficial de Jenkins o de un catálogo privado previamente configurado
- Checkear y/o eliminar plugins instalados actualmente

El catálogo oficial de Jenkins se divide en las siguientes categorías para que la búsqueda o exploración de estos sea más sencilla:

- .NET Development
- Android Development
- Build Parameters
- Cloud Providers
- Columnas de la lista
- Database
- Deployment
- DevOps
- Groovy-related
- iOS Development
- Library plugins (for use by other plugins)
- Misceláneo (administrative-monitor)
- Misceláneo (aws)
- Misceláneo (ca-apm)
- Misceláneo (cmp)
- Misceláneo (codebuild)
- Misceláneo (credential)
- Misceláneo (emailext)
- Misceláneo (file)
- Misceláneo (logging)
- Misceláneo (notification)
- Misceláneo (npm)
- Misceláneo (performance)
- Misceláneo (pipeline)
- Misceláneo (plugin-external)
- Misceláneo (plugin-misc)
- Misceláneo (plugin-post-build)
- Misceláneo (plugin-test)
- Misceláneo (queue)
- Misceláneo (rocketchatnotifier)
- Misceláneo (spot)
- Misceláneo (spotinst)
- Misceláneo (textfile)
- Misceláneo (website)
- Plugins de integración con sitios y herramientas externas

- Plugins de interfaz de usuario
- Plugins de notificación
- Plugins de repositorios de software
- Plugins lanzadores de tareas
- Plugins para decorar páginas
- Plugins para el control de nodos
- Plugins para generar informes
- Plugins para la administración de clusters y trabajos distribuidos
- Plugins para la gestión de usuarios y autenticación
- Plugins para la interfaz de línea de comandos
- Plugins para Maven
- Plugins para subir o desplegar los paquetes generados
- Plugins que añaden acciones de post-ejecutión
- Plugins que añaden tareas relacionadas con la ejecución
- Plugins relacionados con la forma de ejecutar trabajos
- Plugins relacionados con la gestión repositorios
- Python Development
- Ruby Development
- RunConditions for use by the Run Condition plugin
- Scala Development
- Security
- Sin categoría
- Testing
- Varios
- Views

Construyendo con Jenkins

La principal tarea de Jenkins es la de construir **tareas(o jobs)**, para ello nos brinda la posibilidad de crear tareas de distinto tipo de acuerdo a la necesidad que tenga el usuario y el proyecto con el que se esté trabajando. Existen muchos plugins que extienden en cantidad y funcionalidad los distintos tipos de tareas.

Freestyle Project

Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.

Maven Project

Ejecuta un proyecto maven. Jenkins es capaz de aprovechar la configuracion presente en los ficheros POM, reduciendo drásticamente la configuración.

Pipeline

Utilizando pipeline y Jenkins, podemos definir el ciclo de vida completo de una aplicación (descargar código, compilar, test, desplegar, etc.) mediante código

Multibranch Pipeline

Es igual al tipo pipeline pero con la diferencia que nos permite crear una tarea por cada branch que se encuentre en un repositorio.

¿Qué son los pipelines de Jenkins?

Son un conjunto de complementos que nos permiten definir nuestros flujos de integración/entrega continua (delivery pipeline) en Jenkins.

Los jobs y todo su contenido ya no son definidos mediante la interfaz de Jenkins. Estos son desarrollados en Groovy.

Jenkinsfile

El pipeline del proyecto se declara en un fichero, se almacena y se versiona junto con el código en un fichero comúnmente llamado Jenkinsfile. En este fichero definimos principalmente las fases (stages) de que las consta nuestro flujo.

Stages (Fases)

Las fases de nuestro flujo pueden ser muy distintas según el proyecto y las tecnologías con las que trabajemos.

Por ejemplo, cuando hablamos de fases en un flujo de Continuous Deployment, estamos hablando de cada una de las etapas que tenemos desde que el desarrollador sube su código hasta que este es desplegado automáticamente.

Un proyecto java común puede estar compuesto por fases (stages) como: clonado de código, compilado, ejecución de tests, análisis estático, generación y subida de artefacto,construcción de imagen docker y despliegue.

A parte de las fases mencionadas anteriormente podemos añadir cualquier código que necesitemos como funciones, parámetros, configuraciones…

Mantenibilidad

El Jenkinsfile que compone nuestro flujo se almacena junto al código. Con ello, se vuelve mucho más mantenible, ya que es modificable por cualquier desarrollador que tenga acceso al repositorio.

También ganamos todo el potencial que nos proporcionan Github, Bitbucket o cualquier herramienta similar, como ver el histórico de cambios.

La mantenibilidad y versionado es uno de los principales pilares cuando hablamos de Pipelines, pero algunos ya teníamos esto sin necesidad de ellos.

¿Qué ocurre si usamos un job típico de Jenkins, que llame a un script que a su vez ejecute cada uno de los comandos necesarios de nuestro flujo? Que si almacenamos este script junto al código tenemos la misma mantenibilidad y control de versionado que conseguimos con los Jenkinsfile.

En los Jenkinsfiles no sólo almacenamos comandos necesarios. A parte de esto, podemos añadir muchos más elementos como: gestión de credenciales, ficheros, uso de plugins, que antes pertenecían más a la propia configuración del job.

Flexibilidad

Jenkins ofrece gran variedad de plugins que ayudan con las tareas más comunes. Los pipelines son compatibles con la gran mayoría de ellos y nos aportan una gran mejoría.

Al ser programáticos permiten controlar mejor los escenarios y añadir nuevas variantes que eran imposibles hasta ahora.

Esto tiene un punto negativo. Muchas veces es mucho más engorroso definir una una tarea usando un plugin desde un Jenkinsfile, de lo que lo era usando el mismo plugin desde la interfaz de Jenkins.

Visibilidad

Ofrecen una visualización completa de forma atractiva e intuitiva dándonos un feedback rápido del proceso. De un vistazo puede detectarse si hay algún problema y en qué parte de nuestro pipeline ha sucedido.

También ganamos gran visibilidad de los resultados a la hora de ver los logs. Podemos acceder a los de cada fase de forma individual sin tener que sumergirnos en los largos logs de salida de la consola. Gracias a esto ahorramos mucho tiempo a la hora localizar un error en la ejecución.

Para acceder a los logs solo hay que situarse sobre el stage del que queramos consultaros y hacer un clic.

Ventajas:

- Control de versiones.
- Evitamos el riesgo de pérdida de código y configuración.
- Reutilización de código, parámetros, visualización, plugins…

Desventaja:

- Necesidad de mayores conocimientos técnicos.

Tipos de Pipelines

Hay dos tipos de pipeline: scripted y declarative. Nos centraremos en el declarativo, por ser más novedoso y completo, a continuación una tabla comparativa:

SCRIPTED PIPELINE	DESCRIPTED PIPELINE
Modelo de programación imperativo	Modelo de programación declarativo
Entorno de programación con todas las funciones	Sintaxis más simple y pragmática para la creación de Jenkins pipeline
Gran flexibilidad y extensibilidad	Limitado a lo que puede hacer un usuario

SCRIPTED PIPELINE	DESCRIPTED PIPELINE
Se declara y ejecuta mediante nodos	Se declara y ejecuta mediante el comando pipeline y los diferentes stages, stage, steps y step
No hay muchas limitaciones para usuarios expertos y requerimientos complejos	La mejor opción para el desarrollo de estrategias de CI y CD con Pipelines
En común: – Usan Groovy – Pueden utilizar librerías compartidas – Pueden utilizar script y shell – Se guardan en un fichero .jenkinsfile	

Sintaxis Básica de un Pipeline Declarativo

- **Pipeline {}** Identificamos dónde empieza y termina el pipeline así como los pasos que tiene
- **Agent.** Especificamos cuando se ejecuta el pipeline. Uno de los comandos más utilizados es any, para ejecutar el pipeline siempre y cuando haya un ejecutor libre en Jenkins.
- **Stages.** Bloque donde se definen una serie de estados a realizar dentro del pipeline.
- **Stage.** Bloque que define una serie de tareas realizadas dentro del pipeline, por ejemplo: Build. test, deploy, etc. Podemos utilizar varios plugins en Jenkins para visualizar el estado o el progreso de estos estados.
- **Steps.** Son todos los pasos a realizar dentro de un stage. Podemos definir uno o varios pasos.
- **Step.** Es una tarea simple dentro del pipeline. Fundamentalmente es un paso donde se le dice a Jenkins qué hacer en un momento específico o paso del proceso. Por ejemplo, para ejecutar un comando en shell podemos tener un paso en el que tengamos la línea sh “ls” para mostrar el listado de ficheros de una carpeta.

Ejemplo pipeline:

```
pipeline {
        agent any
        stages {
                stage('Build') {
                        steps {
                                sh 'make'
                        }
                }
                stage('Test'){
                        steps {
                                sh 'make check'
                                junit 'reports/**/*.xml'
                        }
                }
                stage('Deploy') {
                        steps {
                                sh 'make publish'
                        }
                }
        }
}
```