



# Jenkins

## Integracion Continua con Jenkins

Victor Herrero Cazurro



# Contenidos

1. Integración Continua	1
2. ¿Que es Integracion Continua (Continuous Integration)?	1
2.1. Ventajas de IC	2
2.2. Costes de IC	2
3. ¿Que es Entrega Continua (Continuous Delivery)?	2
3.1. Ventajas de EC	3
3.2. Costes de EC	3
4. ¿Que es despliegue Continuo (Continuous Deployment)?	3
4.1. Ventajas de DC	4
4.2. Costes de DC	4
5. Entornos de despliegue	4
6. Buenas practicas para Integracion Continua	5
7. Jenkins	5
7.1. Introducción	5
7.2. Instalación	6
7.2.1. Instalacion en un contenedor Docker	8
7.3. Configuración	8
7.4. Seguridad y Gestion de usuarios	10
7.5. Jobs	12
7.5.1. Resultado de la ejecución	15
7.5.2. Variables disponibles en los Jobs	15
7.6. Pipeline	16
7.7. Pipeline Declarativo	20
7.7.1. Agent	20
7.7.2. Post	22
7.7.3. Environment	23
7.7.4. Options	24
7.7.5. Parametros	26
7.7.6. Triggers	26
7.7.7. Tools	27
7.7.8. Input	28
7.7.9. When	29
7.7.10. Parallel	31
7.8. Pipeline Script	33
7.8.1. Descarga del repositorio	33
7.9. Comandos comunes	33
7.9.1. Impresion en el log	33



7.9.2. Ejecucion de un script .....	34
7.9.3. Archivado de ficheros .....	34
7.10. Plugins .....	34
7.10.1. Maven Plugin .....	34
7.10.2. Plugin Sonarqube .....	36
7.10.3. Cobertura Plugin .....	37
7.10.4. Deploy To Container Plugin .....	37
7.10.5. Copy Artifact plugin .....	38
7.10.6. Disk Usage Plugin .....	38
7.10.7. Backup Plugin .....	38
7.10.8. Dependency Graph Viewer Plugin .....	38
7.10.9. Maven Release Plug-in .....	39
7.10.10. Plugin Job DSL .....	39
7.10.11. Plugin Project Template .....	39
7.10.12. Plugin Pipeline .....	39
7.10.13. Otros Plugin .....	39
7.11. Scripting con Jenkins CLI (Deprecated) .....	39
7.12. Consola de Script Integrada .....	41
7.13. API de acceso remoto .....	42
7.14. Ejecución parametrizada .....	43
7.15. Tarea Multiconfiguración .....	44
7.16. Dependencias entre proyectos .....	45
7.17. Ejecución Distribuida .....	45



## 1. Integración Continua

## 2. ¿Que es Integración Continua (Continuous Integration)?

Metodología de desarrollo de software propuesto inicialmente por **Martin Fowler** que permite evaluar el estado de un proyecto en un momento determinado, determinando si los últimos cambios introducidos en el software han introducido además problemas que hagan que el software este inestable.

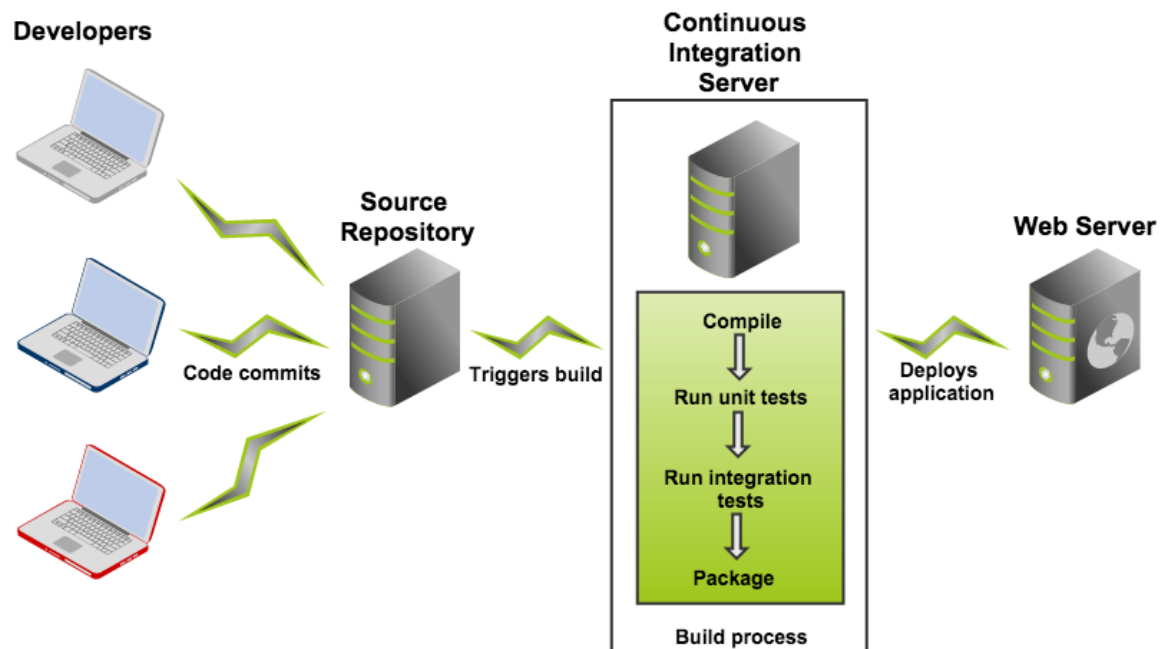
El proceso normalmente es automatico, desencadenandose su ejecución de forma planificada o bien por aparecer una nueva version del software.

Las versiones del software se extraerán de un sistema gestor de versiones.

La evaluación del estado del proyecto se basará en pruebas de compilación, pruebas unitarias y pruebas de integración.

El objetivo principal de estos sistemas es la detección temprana de problemas en los nuevos desarrollos.

### Continuous Integration



## 2.1. Ventajas de IC

- Llegan menos errores a producción por el descubrimiento temprano.
- No existen problemas en crear versiones del software, ya que al hacerse periódicamente, los problemas ya han sido detectados.
- Los desarrolladores sufren menos cambios de contexto, ya que al verificarse el estado del proyecto nada más subir sus cambios, si existen errores, son avisados antes incluso de que hayan cambiado a otra tarea.
- Los costos temporales por la ejecución de las pruebas se reducen drásticamente: el servidor de su CI puede ejecutar cientos de pruebas en cuestión de segundos y de forma automática.
- El equipo de calidad se puede dedicar a otras tareas, como definición de procedimientos, o profundizar en las pruebas ya que no han de dedicar su tiempo con las pruebas que sean automatizadas.

## 2.2. Costes de IC

- Se precisan pruebas automatizadas para cada nueva característica, mejora o corrección de errores.
- Servidor de integración continua y repositorio de versiones de software.
- Commit muy frecuentes al repositorio de versiones de software, recomendable al menos una vez al día.

## 3. ¿Que es Entrega Continua (Continuous Delivery)?

Consiste en automatizar la generación del software entregable.

El concepto está asociado a las metodologías ágiles de desarrollo, por lo que en principio está asociado a iteraciones breves de desarrollo que provocan software funcional.

Evidentemente no se puede llegar a automatizar la entrega de un producto si este no tiene la calidad suficiente, por lo que todo lo comentado en referencia a la importancia de las pruebas para la IC es igualmente aplicable a la EC, si las pruebas son confiables, suponen una garantía de calidad, y por tanto es posible publicar el software cuando las pasan, si estas pruebas se pasan con gran frecuencia, se puede llegar a un escenario en el que en cualquier momento se



puede entregar el software actualizado.

La entrega continua no siempre significa entregar, sino tener la posibilidad de hacerlo rápidamente.

### 3.1. Ventajas de EC

- Se elimina la complejidad de la generación del entregable, por lo que no hay que dedicarle días a la preparación.
- Al poder entregar mas rapido, se obtiene un feedback antes.
- Los cambios pequeños son mas facilmente implementables, no existe la sensación de riesgo por el cambio.

### 3.2. Costes de EC

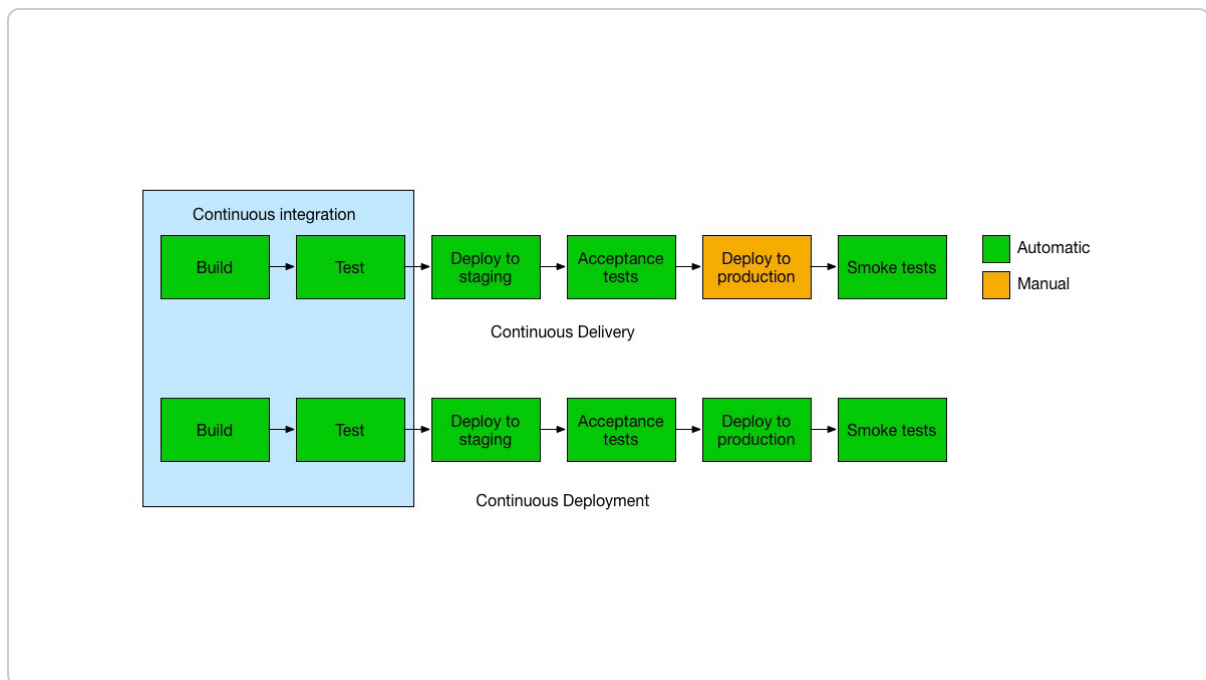
- Se necesita buenas bases de integración continua, con test que cubran gran cantidad de código, todo lo que no se cubra, se incluya en el entregable sin validar.
- Los despliegues deberían ser automatizados, con interacción humana para que se produzca, pero formado por un proceso automatico.
- Se deberán emplear perfiles para que funcionalidades que no estén completas no sean accesibles por el cliente final, aunque el código desarrollado ya esté en el despliegue.

## 4. ¿Que es despliegue Continuo (Continuous Deployment)?

Es un paso más aplicado al concepto de Entrega Continua, en el que si se llega a desplegar el software en producción, sin ningún tipo de intervención humana, la única forma de evitar una nueva release es que fallen los test.

Es una forma rápida de tener la última versión del software en funcionamiento y así poder tener el feedback del usuario final.





### 4.1. Ventajas de DC

- Puede desarrollarse más rápido ya que no es necesario detener el desarrollo para generar una release. Los despliegues se lanzan automáticamente con cada cambio.
- Las Releases son menos arriesgadas y más fáciles de parchear en caso de problemas.
- Los clientes ven un flujo continuo de mejoras y la calidad aumenta todos los días, en lugar de cada mes, trimestre o año.

### 4.2. Costes de DC

- de nuevo los test serán vitales, sin unos buenos test, se desplegará un software de poca calidad.
- La documentación se debe mantener al día.
- El uso de perfiles que permitan activar funcionalidades será obligatorio, dado que el software llega si o si a producción.

## 5. Entornos de despliegue

- **Local:** Equipo del desarrollador
- **Desarrollo:** Servidor donde se realizan los test unitarios



- **Integration:** Servidor donde se realizan los test de integracion.
- **QA:** Servidor donde un perfil cliente realiza pruebas completas de la aplicación siguiendo un plan de pruebas.
- **Stage:** Servidor clon de Production.
- **Production:** Servidor donde esta la Aplicacion en uso por los clientes finales.

## 6. Buenas practicas para Integracion Continua

Según Martin Fowler, se deben seguir las siguientes buenas practicas

- Mantener el codigo versionado con un SCM (Git, SVN, CVS, ...→).
- Programar un servicio de IC para que ejecute las pruebas cada vez que haya un cambio en el repositorio de fuentes de forma automatizada (Jenkins, Bamboo, ...→)
- Crear Test que permitan tener confianza en el codigo generado (JUnit, Selenium, SoapUI, ...→), sobre todo para las partes mas criticas y para las correcciones, ya que han demostrado su vulnerabilidad.
- Commits diarios al SCM, para que el servidor de CI, ofrezca una imagen real del desarrollo.
- Aviso inmediato al autor/equipo del commit que introduce una inestabilidad, para que el problema se arregle tan pronto como se encuentra.
- Mantener una construcción rapida, que el proceso de CI no sea vital, no quiere decir que pueda tardar mucho en ejecutarse.
- Pruebas sobre un clon de producción.
- Obtención facil de las construcciones (Releases, snapshot) a traves de servidores de artefactos.
- Visibilidad del proceso de IC y de los reportes para todos los miembros del equipo.

## 7. Jenkins

### 7.1. Introducción

Servidor de Integracion Continua (CI), basado en Hudson.





Creado por Kohsuke Kawaguchi. Está liberado bajo licencia MIT.

Jenkins tiene la posibilidad de ser extendido mediante Plugins, existiendo multitud de ellos disponibles, mas información [aquí](#)

## 7.2. Instalación

Desde la [pagina oficial](#) se puede realizar la descarga de Jenkins en multiples modalidades.

Si se descarga el **war**, este puede ser desplegado en el servidor de aplicaciones deseado.

Tambien se puede auto ejecutar, ya que lleva embebido un Jetty.

```
java -jar jenkins.war
```

Si se desea cambiar el puerto donde escucha **Jenkins**, que por defecto es el **8080**

```
java -jar jenkins.war --httpPort=8081
```

Otra opción es por ejemplo el instalador de Jenkins para Windows, que crea un servicio de Windows para poder manejar Jenkins.

Cuando se arranca el servicio por defecto Jenkins escucha en **localhost:8080**

Independientemente de como se ejecute, **Jenkins** necesita un directorio de trabajo, que por defecto tiene los siguientes valores para las distintas plataformas con un usuario **admin**

- **Windows 7** → **C:\Users\admin\.jenkins**
- **Windows XP** → **C:\Documents and Settings\admin\.jenkins**
- **Linux** → **/home/admin/.jenkins**

Si se desea cambiar, lo unico que habrá que hacer será definir la variable de entorno **JENKINS\_HOME** con la ubicación deseada.

Si por ejemplo se despliega en un **Tomcat**, este puede ser el encargado de definir dicha variable para su ejecución, para ello se ha de crear un fichero **jenkins.xml** en el directorio **\$CATALINA\_BASE/conf/localhost**, con el siguiente contenido



```
<Context docBase="../../jenkins.war">  
  <Environment name="JENKINS_HOME" type="java.lang.String" value=  
    "/data/jenkins" override="true"/>  
</Context>
```

También se puede cambiar a nivel de la **JVM**

```
java -jar -DJENKINS_HOME=D:\utilidades\jenkins jenkins.war
```

En el proceso de instalación, se pide la definición de un usuario administrador

Getting Started

×

## Create First Admin User

Usuario:

Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

Continue as admin

Save and Finish

Para el curso se establecerá **admin/admin**.



## 7.2.1. Instalación en un contenedor Docker

La gente de Jenkins, ha creado unas imágenes para el uso de Jenkins desde un contenedor Docker, se puede encontrar esas imágenes en [DockerHub](#).

Los más habituales serían

```
> docker run --name jenkins -p 8080:8080 -p 50000:50000 -v  
/var/run/docker.sock:/var/run/docker.sock jenkins/jenkins:lts  
  
> docker run --name jenkins-blue-ocean -p 8080:8080 jenkinsci/blueocean
```

## 7.3. Configuración

La zona de configuración se accede a través del enlace **Administrar Jenkins**

Desde aquí se puede entre otras cosas acceder a

- Configurar el sistema
- Configurar herramientas
- Instalar Plugins

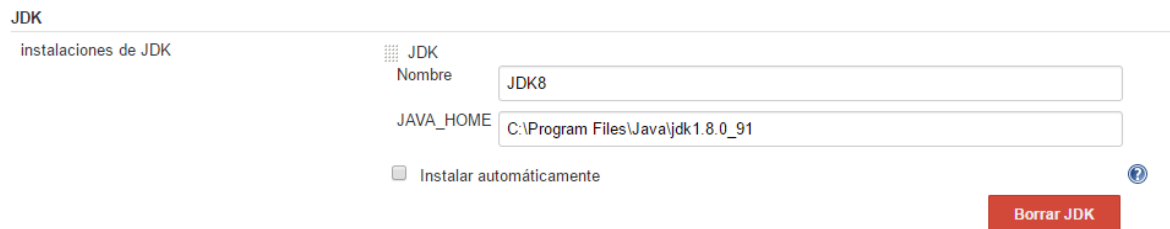


- Consola de Scripts
- Gestión de usuarios

Ya vienen instalados unos cuantos plugins, que habrá que configurar, como son

- Maven
- Git
- SVN
- CVS
- Docker

Lo primero será configurar la JDK, para ello entrar en **Administrar Jenkins** → **Configuración del sistema** → **JDK**



JDK

instalaciones de JDK

JDK

Nombre

JDK8

JAVA\_HOME

C:\Program Files\Java\jdk1.8.0\_91

☐ Instalar automáticamente

Borrar JDK

Lo siguiente será configurar **Maven**, para ello entrar en **Administrar Jenkins** → **Configuración del sistema** → **Maven**



Maven

instalaciones de Maven

Maven

Nombre

Maven 3.3.9

MAVEN\_HOME

D:\utilidades\apache-maven-3.3.9


☐ Instalar automáticamente

Borrar Maven

Añadir Maven

Listado de instalaciones de Maven en este sistema

Lo siguiente a configurar puede ser el servidor de SMTP que se desea emplear para las notificaciones de los eventos



Notificación por correo electrónico

Servidor de correo saliente (SMTP)

localhost

sufijo de email por defecto

@my-proyecto-jenkins-ci.com

☐ Probar la configuración enviando un correo de prueba

Avanzado...

De forma nativa **Jenkins** soporta **CVS** y **SVN**, pero no **Git**, por lo que si se quiere trabajar con **Git**, habrá que instalar un plugin, el **Git Plugin**.



## Integración Continua con Jenkins

<input type="checkbox"/>	<a href="#">CMVC Plugin</a> This plugin integrates <a href="#">CMVC</a> to Hudson.	0.3
<input type="checkbox"/>	<a href="#">Darcs Plugin</a> This plugin integrates <a href="#">Darcs</a> version control system to Jenkins. The plugin requires the Darcs binary (darcs) to be installed on the target machine.	0.3.5
<input type="checkbox"/>	<a href="#">Dimensions Plugin</a> This plugin integrates Hudson with <a href="#">Dimensions</a> , the Serena SCM solution.	0.8.1
<input type="checkbox"/>	<a href="#">File System SCM</a> Use File System as SCM.	1.10
<input checked="" type="checkbox"/>	<a href="#">Git Plugin</a> This plugin allows use of <a href="#">GIT</a> as a build SCM. Git 1.3.3 or newer is required.	1.1.6
<input type="checkbox"/>	<a href="#">Harvest Plugin</a> This plugin allows you to use <a href="#">CA Harvest</a> as a SCM.	0.4

Una vez instalado y para el correcto funcionamiento de **Git** desde **Jenkins**, se ha de configurar el plugin, para ello se ha de acceder a **Administrar Jenkins-Configurar el sistema-Git Plugin** y allí añadir el nombre de usuario y el mail.

Jenkins > Configuración

Directorio raíz de Jenkins  ?

System Admin e-mail address  ?

**SSH Server**

SSH Port ☐ Arreglado:  ☒ Aleatoria ☐ Desactivar ?

**GitHub**

GitHub Servers  ?

**GitHub Enterprise Servers**

**Plugin de tiempo máximo de ejecución > Acción del paso de ejecución**

☐ Habilitar acción del paso de ejecución ?

**Git plugin**

Global Config user.name Value  ?

Global Config user.email Value  ?

Create new accounts base on author/committer's email ☐ ?

**Subversion**

Subversion Workspace Version  ?

Nombre de exclusión "revprop"  ?

**Línea de comandos**

Ejecutable para la línea de comandos (shell)  ?

**Extended E-mail Notification**

SMTP server  ?

Default user E-mail suffix  ?

Default Content Type  ?

## 7.4. Seguridad y Gestión de usuarios

Por defecto Jenkins permite acceder en modo anónimo a todas las tareas, pudiendo ver la información asociada a ellas, aunque no se permite iniciar la construcción.

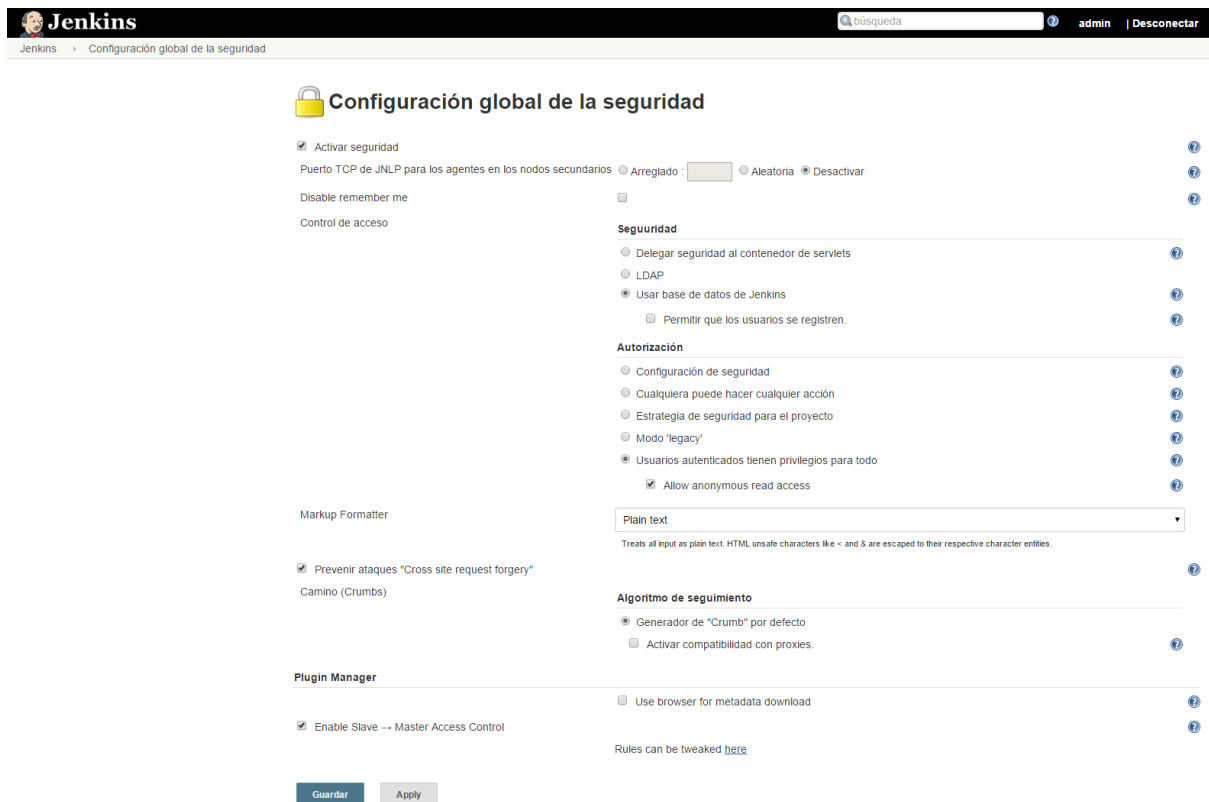
La seguridad se puede activar en **Administrar Jenkins/Configuración Global de la Seguridad**, donde se puede elegir la forma de autenticar



## Integración Continua con Jenkins

- Contenedor de servlets
- LDAP
- Base de datos de Jenkins

También se puede gestionar la autorización, ya que por defecto todos los usuarios autenticados tienen permisos para hacer de todo, pero se pueden establecer planes para todo Jenkins o para los proyectos.



Jenkins > Configuración global de la seguridad

### Configuración global de la seguridad

☒ Activar seguridad

Puerto TCP de JNLP para los agentes en los nodos secundarios: ☐ Arreglado:  ☐ Aleatoria ☒ Desactivar

Disable remember me: ☐

Control de acceso

#### Seguridad

☐ Delegar seguridad al contenedor de servlets

☐ LDAP

☒ Usar base de datos de Jenkins

☐ Permitir que los usuarios se registren.

#### Autorización

☐ Configuración de seguridad

☐ Cualquiera puede hacer cualquier acción

☐ Estrategia de seguridad para el proyecto

☐ Modo 'legacy'

☒ Usuarios autenticados tienen privilegios para todo

☒ Allow anonymous read access

Markup Formatter: Plain text

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

☒ Prevenir ataques "Cross site request forgery"

Camino (Crumbs)

#### Plugin Manager

☒ Enable Slave -> Master Access Control


Use browser for metadata download

Rules can be tweaked [here](#)

De establecerse otros criterios de seguridad, será conveniente dar de alta usuarios, para ello se ha de acceder a la sección **Administrar Jenkins/Gestión de usuarios**




# Integración Continua con Jenkins


 Jenkins


admin | Desconectar

Jenkins > Usar base de datos de Jenkins

ACTIVAR AUTO. REFRESCO



 Volver al Panel de control

 Administrar Jenkins

 Crear un usuario

## Usuarios

Estos usuarios pueden entrar en Jenkins. Este es un subconjunto de [esta lista](#), que también incluyen usuarios creados automáticamente porque hayan hecho 'commits' a proyectos. Los usuarios creados automáticamente no tienen acceso directo a Jenkins.

ID usuario	Nombre
 admin	admin 

Página generada: 27-abr-2016 21:28:56 CEST Jenkins ver. 2.0

## 7.5. Jobs

Representan los trabajos que se pretenden automatizar, luego deberán ejecutar los siguientes pasos

- Descarga de fuentes desde el SCM.
- Compilación del código.
- Ejecución de las pruebas.
- Validación de informes.

Es normal que se delegue en una herramienta de gestión de ciclo de vida del proyecto, como Maven, ANT o Gradle el control de este proceso, aunque existen otras opciones, para crear una tarea de estas características, se ha de seleccionar **Crear un proyecto de estilo libre**.



# Integración Continua con Jenkins

The screenshot shows the Jenkins 'Enter an item name' dialog box. At the top, there's a search bar and 'admin | Desconectar' link. Below the search bar, there's a text input field for the item name, with a 'Required field' label. The main area lists several item types with icons and descriptions:

- Crear un proyecto de estilo libre**: 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.
- Pipeline**: Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Crear un proyecto multi-configuración**: Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en multiples entornos, ejecutar sobre plataformas concretas, etc.
- External Job**: Este tipo de tareas te permite registrar la ejecución de un proceso externo a Jenkins, incluso en una máquina remota. Está diseñado para usar Jenkins como un panel de control de tu sistema de automatización. Para más información consulta esta [página](#).
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- GitHub Organization**: Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- Multibranch Pipeline**: Creates a set of Pipeline projects according to detected branches in one SCM repository.

Below these options, there's a section 'if you want to create a new item from other existing, you can use this option:' with a 'Copy from' dropdown menu and a 'Type to autocomplete' input field. At the bottom, there's an 'OK' button.

Lo primero en la creación de la tarea, será definir el origen del código, es decir la conexión con el SCM.

The screenshot shows the Jenkins 'Configurar el origen del código fuente' configuration page. The page has a breadcrumb 'Jenkins > Proyecto >' and a navigation bar with tabs: 'General', 'Configurar el origen del código fuente' (selected), 'Disparadores de ejecuciones', 'Entorno de ejecución', 'Ejecutar', and 'Acciones para ejecutar después.'.

Under the 'Configurar el origen del código fuente' tab, there are several sections:

- General**: Contains checkboxes for 'Desactivar la ejecución' and 'Lanzar ejecuciones concurrentes en caso de ser necesario'. An 'Avanzado...' button is on the right.
- Configurar el origen del código fuente**: This section has radio buttons for 'Ninguno', 'Git', and 'Subversion' (selected). Below this is a 'Módulos' section with fields for 'Repository URL' (https://Victor-Portatit.8443/svn/EjemploReleasePlugin/trunk), 'Credentials' (desarrollador/\*\*\*\*\* (Usuario desarrollador para SVN) with an 'Add' button), 'Local module directory' (.), 'Repository depth' (infinity), and 'Ignore externals' (checked). There are 'Add module...' and 'Add additional credentials...' buttons. Below this is a 'Check-out Strategy' dropdown set to 'Use 'svn update' as much as possible' with a description: 'Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.' At the bottom of this section is a 'Navegador del repositorio' dropdown set to '(Auto)' and another 'Avanzado...' button.
- Disparadores de ejecuciones**: Contains checkboxes for 'Lanzar ejecuciones remotas (ejem: desde 'scripts')', 'Build after other projects are built', and 'Build when a change is pushed to GitHub'.

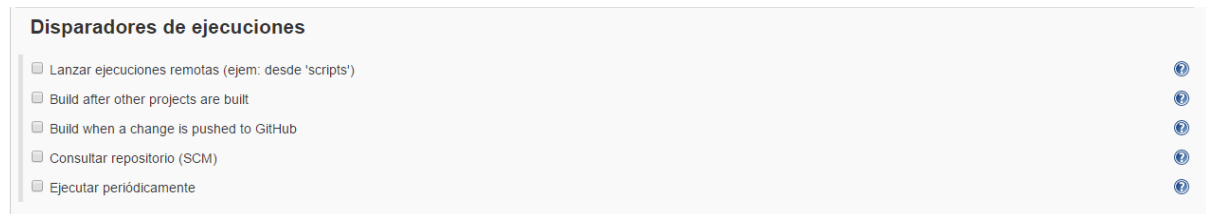
At the bottom, there are 'Guardar' and 'Apply' buttons.

Se podrá definir un disparador (Trigger) que inicie la ejecución de la tarea, hay varios tipos

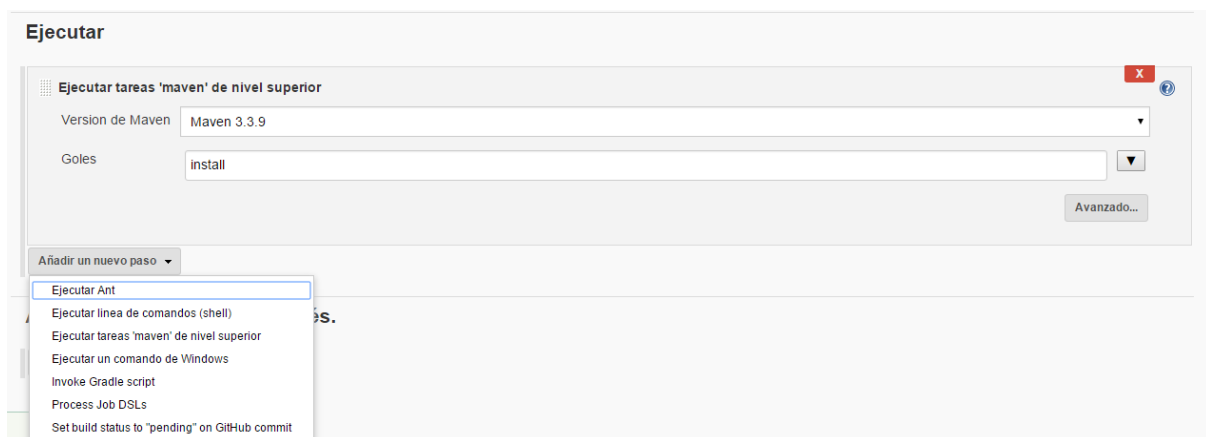




- Ejecución temporal empleando una expresión de Cron.
- Comprobar periódicamente si el estado del SCM no ha cambiado, y si cambia construir, se emplea una expresión de Cron.
- Construir cuando haya cambios en Github, este SCM, permite definir un Hook, que establece una comunicación bidireccional entre Github y Jenkins, pudiendo Github indicar cuando hay cambios para que Jenkins construya.
- Construir cuando otros proyectos se construyan.

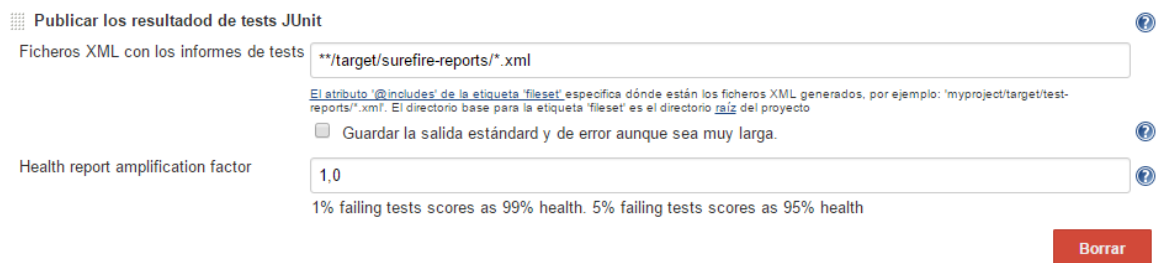


Habrá que definir una tarea o conjunto de tareas a realizar una vez se tenga el código fuente, una de las más habituales es una tarea Maven.



Se pueden definir pasos posteriores a la tarea, como por ejemplo la **publicación de los resultados de los test de JUnit**

Acciones para ejecutar después.



También se puede configurar el archivado de los artefactos producidos por la tarea



Guardar los archivos generados

Ficheros para guardar

Avanzado...

Borrar

## O la publicación de los Javadoc generados

Publicar Javadoc

Directorio para los javadoc

Directorio relativo al 'workspace' del proyecto, ejemplo: 'myproject/build/javadoc'

☐ Conservar los javadoc para todas las ejecuciones correctas

Borrar

### 7.5.1. Resultado de la ejecución

La ejecución de la tarea, se mostrará con un círculo de color

- azul. La ejecución ha ido bien.
- amarillo. Ha habido un problema con los Test o con la Cobertura.
- rojo. Ha habido un error en ejecución.

La ejecución de la tarea puede ofrecer como resultado

- Sol (0/5)
- Nubes (1-2/5)
- Lluvia (3-4/5)
- Tormenta (5-5)

Todo +						
S	W	Name ↓	Último éxito	Último fallo	Última duración	
		<a href="#">CleanDailyReleases</a>	2 días 12 Hor (#12)	N/D	10 Seg	
		<a href="#">Deploy Daily Build</a>	4 Hor 47 Min (#104)	6 días 0 Hor (#100)	2 Min 13 Seg	
		<a href="#">Liquibase-Update</a>	4 Hor 48 Min (#123)	4 Hor 48 Min (#123)	9,9 Seg	
		<a href="#">Liquibase-Update-SQL</a>	4 Hor 48 Min (#95)	1 Mes 17 días (#42)	31 Seg	
		<a href="#">Power Desk Web</a>	4 Hor 59 Min (#93)	21 Hor (#91)	10 Min	
		<a href="#">Restore-DB-IC</a>	4 Hor 48 Min (#96)	2 Mes 7 días (#7)	2,2 Seg	

### 7.5.2. Variables disponibles en los Jobs

A disposición de la configuración de los Jobs, existen una serie de variables definidas por Jenkins.



- **BUILD\_NUMBER** → El número de ejecución actual.
- **BUILD\_ID** → La fecha empleada para identificar la ejecución actual en formato **YYYY-MM-DD\_hh-mm-ss**.
- **JOB\_NAME** → Nombre de la tarea, por ejemplo **game-of-life**.
- **BUILD\_TAG** → Etiqueta que permite identificar la actual ejecución de la tarea, el formato es **jenkins-\${JOB\_NAME}-\${BUILD\_NUMBER}**, por ejemplo **jenkins-game-of-life-2010-10-30\_23-59-59**.
- **EXECUTOR\_NUMBER** → Número que identifica el ejecutor que ejecuta la tarea actual.
- **NODE\_NAME** → Nombre del nodo esclavo que ejecuta la tarea, de ser "", es que se ejecuta en master.
- **NODE\_LABELS** → Lista de etiquetas asociadas con el Nodo que ejecuta la tarea.
- **JAVA\_HOME** → Variable de entorno **JAVA\_HOME**, que se emplea para actualizar la variable de entorno **PATH** con el valor **\$JAVA\_HOME/bin**.
- **WORKSPACE** → Path absoluto del workspace.
- **HUDSON\_URL** → URL del servidor Jenkins.
- **JOB\_URL** → URL del Job, por ejemplo **<http://ci.acme.com:8080/jenkins/gameof-life>**.
- **BUILD\_URL** → URL de la ejecución, por ejemplo **<http://ci.acme.com:8080/jenkins/game-oflife/20>**.
- **SVN\_REVISION** → Revision para Jobs con repositorio SVN.
- **CVS\_BRANCH** → Rama para proyectos CVS.

Para acceder a estas variables desde Maven, basta con indicar entre llaves la variable **\${JOB\_URL}**.

Para scripts de Groovy, haríamos

```
def env = System.getenv()
println env['BUILD_NUMBER']
```

## 7.6. Pipeline

Es un plugin de Jenkins que permite definir un flujo, en un DSL propio basado en



Groovy, con las tareas a realizar por Jenkins sobre un repositorio.

image::pipeline.png

Existen dos maneras de definir el flujo

- Declarativo: Basado en el componente **pipeline**



```

pipeline {
    agent any

    stages {
        stage('Preparation') { // for display purposes
            steps {
                //Descarga del repositorio a local
                git 'D:\\Cursos\\2018-06-Bamboo-CLE
ed1\\RepositorioDemo'
                //Alternativamente se puede hacer
                //checkout scm

                // La herramienta Maven, con nombre M., debe estar
                configurada e global configuration.
                mvnHome = tool 'M3'
            }
        }
        stage('Build') {
            steps {
                // Ejecucion de tarea maven, que aunque lanza los Test,
                no los evalua, se hará en la siguiente etapa
                if (isUnix()) {
                    sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore
clean package"
                } else {
                    bat("/"${mvnHome}\\bin\\mvn" -Dmaven.test.failure.ignore
clean package/)
                }
            }
        }
        stage('Results') {
            steps {
                //Se evaluan los resultados de los test, haciendo
                fallar el buil, sino estan correctos
                junit '**/target/surefire-reports/TEST-*.xml'
                //Se almacena el artefacto generado
                archiveArtifacts 'target/*.war'
            }
        }
    }
}

```



- Script: Basado en el componente **node**

```
node {
    def mvnHome

    stage('Preparation') { // for display purposes
        //Descarga del repositorio a local
        git 'D:\\Cursos\\2018-06-Bamboo-CLE ed1\\RepositorioDemo'
        //Alternativamente se puede hacer
        //checkout scm

        // La herramienta Maven, con nombre M., debe estar configurada e
        global configuration.
        mvnHome = tool 'M3'
    }

    stage('Build') {
        // Ejecucion de tarea maven, que aunque lanza los Test, no los
        evalua, se hará en la siguiente etapa
        if (isUnix()) {
            sh "${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean
package"
        } else {
            bat("/"${mvnHome}\\bin\\mvn" -Dmaven.test.failure.ignore clean
package/)
        }
    }

    stage('Results') {
        //Se evaluan los resultados de los test, haciendo fallar el buil,
        sino estan correctos
        junit '**/target/surefire-reports/TEST-*.xml'
        //Se almacena el artefacto generado
        archiveArtifacts 'target/*.war'
    }
}
```

Con ambos se consigue el mismo efecto.

Se puede definir el **pipeline** visualmente sobre la aplicación web asociada a Jenkins o bien tenerlo definido en un fichero **Jenkinsfile** en el SCM, esta ultima es la opcion recomendada dado que permite cambiar los comportamientos de jenkins sin interaccionar con la aplicación web, unicamente comitando al SCM, estando ademas



el flujo asociado a los fuentes del proyecto.

Para escribir el script que representa el pipeline, Jenkins ofrece una ayuda dinámica que ofrece ejemplos de uso de los distintos plugins instalados en Jenkins, para acceder a esa ayuda se ha de invocar <http://localhost:8080/pipeline-syntax>

Así mismo se ofrece un listado de las variables de entorno disponibles en el pipeline en el servicio <http://localhost:8080/pipeline-syntax/globals#env> y de variables disponibles en la actual ejecución del job en <http://localhost:8080/pipeline-syntax/globals#currentBuild>

También se proporciona un compendio de ejemplos de uso de steps de Jenkins en <https://jenkins.io/doc/pipeline/examples/>

## 7.7. Pipeline Declarativo

Los Pipelines declarativos se organizan en los siguientes componentes principales:

- pipeline: declara el ámbito del flujo
- agent: Reserva un ejecutor y un workspace para el pipeline.
- stages: Agrupan todas las etapas del flujo.
- stage: Da nombre a una etapa del flujo
- steps: define los pasos que componen una etapa.

### 7.7.1. Agent

Reserva un ejecutor y un workspace para el pipeline.

Realiza la descarga de los fuentes del SCM.

Es obligatorio.

Se puede incluir a nivel del **pipeline** o en cada **stage**.

Acepta como valores: **any**, **none**, **label**, **node**, **docker** o **dockerfile**.



### Ejemplo de agent docker

```
agent {  
  docker {  
    image 'maven:3.3.9-jdk-8'  
    args "--entrypoint=''" //Es necesario para evitar un error  
  }  
}
```

#### NOTE

La seccion de **args** definida en el anterior error, es necesaria para evitar el siguiente error

```
java.io.IOException: Failed to run top  
'506c3f7e22182431d37233d14ed90c0d1f50e91347cbffc26fc659a7  
7fddbaae'. Error: Error response from daemon: Container  
506c3f7e22182431d37233d14ed90c0d1f50e91347cbffc26fc659a77  
fddbaae is not running
```

Mas informacion [aquí](#)

#### NOTE

No poner los argumentos --rm y --name, ya que el plugin se encarga de borrar el contenedor

### Ejemplo de agent dockerfile

```
agent {  
  // Equivalent to "docker build ./docker/  
  dockerfile {  
    filename 'Dockerfile'  
    dir 'docker'  
  }  
}
```

Los parametros que acepta el bloque **agent** son:

- label
- customWorkspace
- reuseNode





Para poder emplear contenedores de docker desde un jenkins ejecutandose en un contenedor de docker, se necesita lanzar el comando compartiendo el **sock**

### NOTE

```
docker run -p 8080:8080 -v  
/var/run/docker.sock:/var/run/docker.sock --name jenkins-  
docker jenkins/jenkins:lts
```

### 7.7.2. Post

Permite la definicion de tareas a ejecutar despues de ejecutar el **pipeline** o un **stage**.

Es condicional, admitiendo las siguientes condiciones

- **always**
- **changed**: solo ejecuta el bloque si hay cambios en el estado del stage o del pipeline con respecto a la anterior ejecucion
- **fixed**: solo ejecuta el bloque cuando el stage o pipeline terminan **success** y el anterior build termino **failure** o **unstable**.
- **regression**: solo ejecuta el bloque cuando el stage o pipeline terminan **failure**, **unstable** o **abort** y el anterior build termino **success**.
- **aborted**: solo ejecuta el bloque cuando el stage o pipeline terminan **abort** (las etapas abortadas se representan en gris)
- **failure**: solo ejecuta el bloque cuando el stage o pipeline terminan **failure** (las etapas abortadas se representan en rojo)
- **success**: solo ejecuta el bloque cuando el stage o pipeline terminan **success** (las etapas abortadas se representan en azul o verde)
- **unstable**: solo ejecuta el bloque cuando el stage o pipeline terminan **unstable**, esto puede ser porque hayan fallado test o existan violaciones de codigo (las etapas abortadas se representan en amarillo)
- **cleanup**: Se ejecuta al final del resto de post condiciones.



```

pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'
      }
    }
  }
  post {
    always {
      echo 'I will always say Hello again!'
    }
  }
}

```

### 7.7.3. Environment

Permite definir variables de entorno en forma de pares clave-valor a nivel del pipeline o de un stage.

```

pipeline {
  agent any
  environment {
    CC = 'clang'
  }
  stages {
    stage('Example') {
      environment {
        AN_ACCESS_KEY = credentials('jenkins')
      }
      steps {
        sh 'printenv'
        echo("env.AN_ACCESS_KEY_USR =
'${env.AN_ACCESS_KEY_USR}'")
        echo("env.AN_ACCESS_KEY_PSW =
'${env.AN_ACCESS_KEY_PSW}'")
      }
    }
  }
}

```



Dentro del bloque **environment**, se puede acceder a credenciales predefinidas en el ámbito de **jenkins**, definiéndose para el anterior caso, tres variables de entorno:

**AN\_ACCESS\_KEY**, **AN\_ACCESS\_KEY\_USR** y **AN\_ACCESS\_KEY\_PSW**.

Debiendo de existir una credencial de tipo **User y Password** con **id** igual a **jenkins**.

Las variables de entorno son accesibles con

**`${env.<NOMBRE_DE_LA_VARIABLE>}`**

### 7.7.4. Options

Permite definir opciones particulares para un **pipeline**.

Estas opciones pueden ser provistas por **plugins**

Las opciones disponibles son:

- **buildDiscarder**: Permite indicar el número de ejecuciones de las que almacenará los artefactos y los logs.

```
options {  
    buildDiscarder(logRotator(numToKeepStr: '1'))  
}
```

- **checkoutToSubdirectory**: Permite indicar la descarga de los fuentes en un subdirectorio

```
options {  
    checkoutToSubdirectory('foo')  
}
```

- **disableConcurrentBuilds**: Deshabilita la ejecución concurrente

```
options {  
    disableConcurrentBuilds()  
}
```

- **newContainerPerStage**: En **agent docker** o **\*dockerfile**, permite indicar el uso de agentes distintos para cada **stage**
- **overrideIndexTriggers**:



- **preserveStashes**
- **retry**: Permite definir el número de intentos, si el pipeline termina en **failure**

```
options {  
    retry(3)  
}
```

- **skipDefaultCheckout**: Permite omitir la descarga inicial del SCM

```
options {  
    skipDefaultCheckout()  
}
```

- **skipStagesAfterUnstable**: Permite dejar de ejecutar los **stages** posteriores a uno que acabe con estado **unstable**

```
options {  
    skipStagesAfterUnstable()  
}
```

- **timeout**: Permite definir un tiempo máximo de ejecución del **pipeline**, llegado ese momento, si el pipeline no ha terminado se aborta.

```
options {  
    timeout(time: 1, unit: 'HOURS')  
}
```

- **timestamps**: Permite añadir a todas las líneas de log generadas, la hora a la que son generadas.

```
options {  
    timestamps()  
}
```

Para los **stage** únicamente se tienen disponibles las opciones: **retry**, **timeout**, **timestamps** y **skipDefaultCheckout**.



### 7.7.5. Parametros

Permite definir parametros disponibles en la ejecucion del pipeline.

Existen dos tipos

- string
- booleanParam

```
pipeline {
    agent any
    parameters {
        string(name: 'PERSON', defaultValue: 'Mr Jenkins', description:
'Who should I say hello to?')
        booleanParam(name: 'DEBUG_BUILD', defaultValue: true,
description: '')
    }
    stages {
        stage('Example') {
            steps {
                echo "Hello ${params.PERSON}"
            }
        }
    }
}
```

Se acceden con **`${params.<NOMBRE_DEL_PARAMETRO>}`**

Cuando se define, se muestra una opcion de ejecucion nueva **build with parameters**

[ pipeline build with parameters 1 ] | *pipeline\_build\_with\_parameters\_1.png*

Al pulsar en ella se accede a un formulario donde se puede rellenar los valores de los parametros.

[ pipeline build with parameters 2 ] | *pipeline\_build\_with\_parameters\_2.png*

### 7.7.6. Triggers

Permite definir las formas por las que el **pipeline** es ejecutado de forma automatizada.



Las posibilidades son:

- **cron**: Permite definir una condición temporal empleando una expresión **cron**

```
triggers {  
    cron('H */4 * * 1-5')  
}
```

- **pollSCM**: Permite definir el intervalo de tiempo que transcurra entre consultas de Jenkins al SCM en busca de nuevos cambios.

```
triggers {  
    pollSCM('H */4 * * 1-5')  
}
```

- **upstream**: Permite asociar la ejecución del build a la finalización de otros builds. Acepta nombres de jobs separados por comas y estados en los que han de terminar

```
triggers {  
    upstream(upstreamProjects: 'job1,job2', threshold:  
        hudson.model.Result.SUCCESS)  
}
```

### 7.7.7. Tools

Permite añadir al **PATH** del agente una herramienta.

Se soportan: maven, jdk y gradle



```

pipeline {
  agent any
  tools {
    maven 'M3'
  }
  stages {
    stage('Example') {
      steps {
        sh 'mvn --version'
      }
    }
  }
}

```

Será necesario que en la configuracion de **jenkins**, se haya definido la herramienta maven llamada **M3**, para hacerlo se accede al menú **Manage jenkins / global tool configuration**

### 7.7.8. Input

Permite interrumpir la ejecucion de un **stage** del pipeline a la espera de una validacion manual de un usuario de jenkins.

Se pueden definir los siguientes parametros:

- **message**: Requerido. Mensaje que visualizará el usuario que tiene que aceptar o cancelar el **stage**
- **id**: Opcional. Por defecto se le da el nombre del **Stage**.
- **ok**: Opcional. Texto a mostrar en el boton de **Aprobar**.
- **submitter**: Opcional. Lista de usuarios o grupos que tienen permiso para **Aprobar**. Por defecto puede cualquier usuario.
- **submitterParameter**: Opcional. Nombre de una variable de entorno con el nombre del usuario que interacciona con el formulario.
- **parameters**: Opcional. Lista de parametros a mostrar al usuario que interacciona con el formulario.



```

pipeline {
  agent any
  stages {
    stage('Example') {
      input {
        message "Should we continue?"
        ok "Yes, we should."
        submitter "alice,bob"
        parameters {
          string(name: 'PERSON', defaultValue: 'Mr Jenkins',
description: 'Who should I say hello to?')
        }
      }
      steps {
        echo "Hello, ${PERSON}, nice to meet you."
      }
    }
  }
}

```

Cuando un **stage** esta parado esperando la interaccion del usuario, se puede acceder al formulario pasando el raton por encima del **stage**

[ pipeline build with input 1 ] | *pipeline\_build\_with\_input\_1.png*

O bien accediendo al build, que aparece una nueva opcion \*\*

[ pipeline build with input 2 ] | *pipeline\_build\_with\_input\_2.png*

### 7.7.9. When

Permite definir condiciones de ejecucion para los **stage**.

Se debe definir al menos una condicion, de definirse varias, todas han de cumplirse para que se ejecute el **stage**.

Se proporcionan las siguientes opciones para definir las condiciones:

- **branch**: Define una condicion basada en el nombre del **branch** a contruir. Solo valido en los **build** de tipo **multibranch**.

```
when { branch 'master' }
```





- **buildingTag**: Se cumple cuando existe la variable de entorno **TAG\_NAME**.

```
when { buildingTag() }
```

- **changelog**: Se cumple cuando el changelog del SCM, cumple con una expresion regular definida.

```
when { changelog '.*^\\[DEPENDENCY\\] .+$' }
```

- **changeset**: Se cumple cuando el changet del SCM contiene alguno de los ficheros indicados

```
when { changeset "**/*.js" }
```

Por defecto es **case insensitive**, se puede cambiar con la propiedad **caseSensitive**

```
when { changeset glob: "ReadMe.*", caseSensitive: true }
```

- **changeRequest**: Se cumple cuando el **build** viene motivado por un **change request (pull request)**

```
when { changeRequest() }
```

Se puede parametrizar con: id, target, branch, fork, url, title, author, authorDisplayName, and authorEmail.

```
when { changeRequest target: 'master' }
```

- **environment**: Se cumple si existe la variable de entorno indicada y su valor tambien es el indicado.

```
when { environment name: 'DEPLOY_TO', value: 'production' }
```

- **equals**: Permite definir una condicion de igualdad sobre variables de ejecucion.



```
when { equals expected: 2, actual: currentBuild.number }
```

- **expression:** Permite definir una expresión de Groovy booleana

```
when { expression { return params.DEBUG_BUILD } }
```

- **tag:** Se cumple si la variable de entorno **TAG\_NAME** cumple el patrón definido. Si no hay patrón, es equivalente a **buildingTag()**

```
when { tag "release-*" }
```

- **not:** Permite negar las anteriores condiciones

```
when { not { branch 'master' } }
```

- **allOf:** Se cumple si todas las condiciones se cumplen. Se debe definir al menos una.

```
when { allOf { branch 'master'; environment name: 'DEPLOY_TO', value: 'production' } }
```

- **anyOf:** Se cumple si alguna de las condiciones se cumple

```
when { anyOf { branch 'master'; branch 'staging' } }
```

### 7.7.10. Parallel

Permite la ejecución de **stage** en paralelo.

Se puede indicar que el **stage** principal que contiene los **stage** secundarios a ejecutar en paralelo falle cuando falle alguna de las tareas indicando **failfast true**

```
pipeline {
    agent any
    stages {
        stage('Non-Parallel Stage') {
            steps {
```



```

        echo 'This stage will be executed first.'
    }
}
stage('Parallel Stage') {
    when {
        branch 'master'
    }
    failFast true
    parallel {
        stage('Branch A') {
            agent {
                label "for-branch-a"
            }
            steps {
                echo "On Branch A"
            }
        }
        stage('Branch B') {
            agent {
                label "for-branch-b"
            }
            steps {
                echo "On Branch B"
            }
        }
        stage('Branch C') {
            agent {
                label "for-branch-c"
            }
            stages {
                stage('Nested 1') {
                    steps {
                        echo "In stage Nested 1 within Branch
C"
                    }
                }
                stage('Nested 2') {
                    steps {
                        echo "In stage Nested 2 within Branch
C"
                    }
                }
            }
        }
    }
}

```



```
}
}
}
```

## 7.8. Pipeline Script

Los Pipelines script se organizan en los siguientes componentes principales:

- **node**: declara el uso de un nodo de ejecución, creando un directorio workspace para trabajar con los ficheros descargados del SCM.
- **stage**: Da nombre a una etapa del flujo

### 7.8.1. Descarga del repositorio

Es lo primero que se ha de hacer, dependiendo del tipo de **Build**, se podrá predefinir el **scm** fuera del pipeline, en los **multibranch**, o dentro del pipeline, en los **basicos**.

Para descargar cuando ya esta definido el **scm**, se puede invocar

- **checkout**: Descarga los fuentes del SCM (necesario para los bloques **node**)

Sino esta definido, se trendran los comandos

- **git**
- **svn**
- **cvs**

## 7.9. Comandos comunes

### 7.9.1. Impresion en el log

Se proporciona el comando **echo** que permite escribir en la traza

```
stage('Example Test') {
    agent { docker 'openjdk:8-jre' }
    steps {
        echo 'Hello, JDK'
        sh 'java -version'
    }
}
```



### 7.9.2. Ejecución de un script

Se proporcionan dos comandos para lanzar script propietarios del sistema operativo

- sh para linux

```
sh ....
```

- bat para windows

```
bat ....
```

Estos comandos interrumpen la ejecución del pipeline si el retorno del comando es distinto de 0.

### 7.9.3. Archivado de ficheros

```
archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
```

#### NOTE

La tarea **archiveArtifacts** es para realizar un archivado local de jenkins, no sustituye a Nexus, Artifactory, ...→

## 7.10. Plugins

### 7.10.1. Maven Plugin

Se ha de configurar Maven en Jenkins, para ello se ha de acceder a **Administrar Jenkins/Global Tool Configuration** y allí crear una nueva configuración de Maven, indicando o bien **MAVEN\_HOME**, o bien que se descargue la versión de Maven deseada.



# Integración Continua con Jenkins

Jenkins > Global Tool Configuration

Name: Default

Path to Git executable: git.exe

☐ Instalar automáticamente

[Delete Git](#)

[Add Git](#)

description

**Gradle**

instalaciones de Gradle

[Añadir Gradle](#)

Listado de instalaciones de Gradle en este sistema

**Ant**

instalaciones de Ant

[Añadir Ant](#)

Listado de instalaciones de Ant en este sistema

**Maven**

instalaciones de Maven

☐ Maven

Nombre: Maven 3.3.9

MAVEN\_HOME: D:\utilidades\apache-maven-3.3.9

☒ Instalar automáticamente

**Instalar desde Apache**

Versión: 3.3.9

[Borrar un instalador](#)

[Añadir un instalador](#)

[Añadir Maven](#)

Listado de instalaciones de Maven en este sistema

[Save](#) [Apply](#)

Página generada: 27-abr-2016 20:59:31 CEST [Jenkins ver. 2.0](#)

## NOTE

Una vez configurado Maven, se ha de asegurar que los proyectos emplean esta configuración, en versiones de Jenkins ocurre que se selecciona la versión de Maven por defecto y de esta forma no funciona la construcción

Jenkins > Proyecto >

General Configurar el origen del código fuente **Disparadores de ejecuciones** Entorno de ejecución Ejecutar Acciones para ejecutar después.

**Disparadores de ejecuciones**

☐ Lanzar ejecuciones remotas (ejem: desde 'scripts')

☐ Build after other projects are built

☐ Build when a change is pushed to GitHub

☐ Consultar repositorio (SCM)

☐ Ejecutar periódicamente

**Entorno de ejecución**

☐ Delete workspace before build starts

☐ Abortar la ejecución si se atasca

☐ Add timestamps to the Console Output

☐ Use secret text(s) or file(s)

**Ejecutar**

**Ejecutar tareas 'maven' de nivel superior**

Version de Maven: Maven 3.3.9

Goles: (Por defecto) Maven 3.3.9

[Avanzado...](#)

[Añadir un nuevo paso](#)

**Acciones para ejecutar después.**

[Añadir una acción](#)

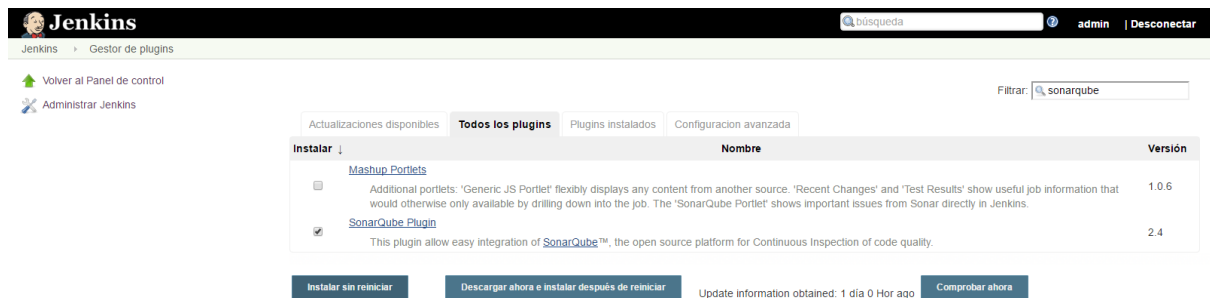
[Guardar](#) [Apply](#)

Página generada: 27-abr-2016 21:01:54 CEST [REST API](#) [Jenkins ver. 2.0](#)

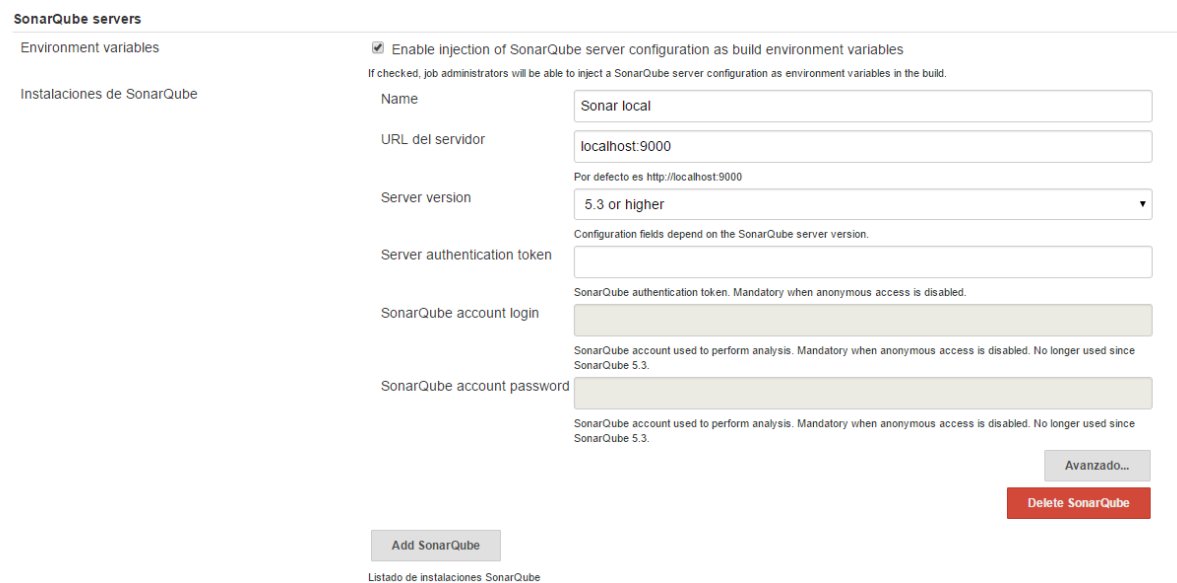


## 7.10.2. Plugin Sonarqube

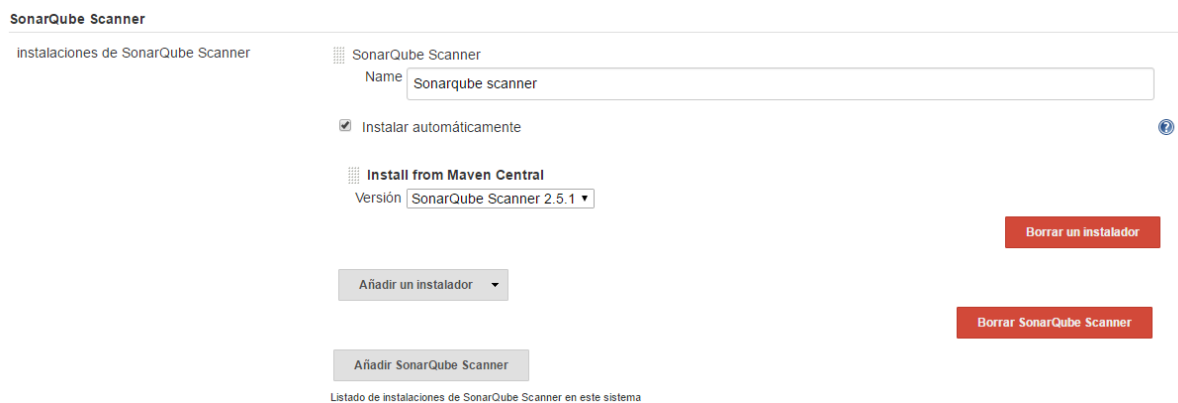
Es un pugin que permite conectar Jenkins con Sonar.



Se ha de configurar el servidor Sonar en **Administrar Jenkins/Configurar el sistema**



Se ha de configurar el Sonarqube Scanner en **Global Tool Configuration**



Este plugin proporciona un nuevo ejecutable a incluir en la ejecución de la tarea.



## Ejecutar

Ejecutar tareas 'maven' de nivel superior

Version de Maven

Maven 3.3.9

Goles

install

Avanzado...

Execute SonarQube Scanner

Task to run

JDK

(Inherit From Job)

JDK to be used for this sonar analysis

Path to project properties

Analysis properties

Additional arguments

JVM Options

Añadir un nuevo paso

## 7.10.3. Cobertura Plugin

Plugin que permite visualizar los resultados del analisis estatico de código que realiza Cobertura, así como la definición de los limites en los cuales se considera una Cobertura aceptable.

☒ Publish Cobertura Coverage Report

Cobertura xml report pattern

\*\*/target/site/cobertura/coverage.xml

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use **\*\*/target/site/cobertura/coverage.xml**). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root. Cobertura must be configured to generate XML reports for this plugin to function.

Consider only stable builds

☐

Include only stable builds, i.e. exclude unstable and failed ones.

Coverage Metric Targets

Conditionals		☀️ 98	☁️ 75	🟡 75
Lines	Delete	☀️ 98	☁️ 75	🟡 75
Methods	Delete	☀️ 100	☁️ 80	🟡 80
Packages	Delete	☀️ 100	☁️ 95	🟡 95
Add				

Configure health reporting thresholds.  
 For the ☀️ row, leave blank to use the default value (i.e. 80).  
 For the ☁️ and 🟡 rows, leave blank to use the default values (i.e. 0).

## 7.10.4. Deploy To Container Plugin

Es un **Plugin**, que permite desplegar una aplicación empresarial en un servidor de aplicaciones, como Tomcat, JBoss, WebSphere, Weblogic, ... ↗





☒ Deploy war/ear to a container

WAR/EAR files

Container

Manager user name

Manager password

Tomcat URL

### 7.10.5. Copy Artifact plugin

Permite copiar uno o varios ficheros de un **Job** a otro.

**Build**

**Copy artifacts from another project**

Project name

Which build

☐ Stable build only

Artifacts to copy

Target directory

☒ Flatten directories ☐ Optional

### 7.10.6. Disk Usage Plugin

Permite monitorizar el uso del disco.

### 7.10.7. Backup Plugin

Aunque el Backup de Jenkins es fácil de realizar, basta con hacer el backup de la carpeta **JENKINS\_HOME**, este plugin facilita la tarea, permitiendo configurar que partes del directorio se van a guardar, ya que por ejemplo la carpeta de **workspace** es una carpeta innecesaria a la hora del backup y que puede ocupar bastante, ya que contiene el proyecto entero.

### 7.10.8. Dependency Graph Viewer Plugin

Permite visualizar las dependencias configuradas entre los **Jobs** definidos en **Jenkins**.

Este plugin emplea **graphviz**, el cual habrá que tener instalado en el equipo.



### 7.10.9. Maven Release Plug-in

Permite publicar una release empleando el plugin de release de **Maven**, siendo configurado por **Jenkins**

### 7.10.10. Plugin Job DSL

Este plugin, permite definir la tarea como un script DSL de Groovy, se puede encontrar un tutorial que crea una tarea a partir de una tarea de tipo Job DSL [aquí](#)

### 7.10.11. Plugin Project Template

Permite reutilizar las configuraciones de un proyecto en otro

#### NOTE

Dentro de la solución de pago de **CloudBees**, se proporcionan plugins para crear plantillas no solo de **Jobs**, sino también incluso de **Builds**

### 7.10.12. Plugin Pipeline

Permite definir un script con las fases de un Job.

El Script se escribe en groovy

### 7.10.13. Otros Plugin

## 7.11. Scripting con Jenkins CLI (Deprecated)

Descargar el siguiente jar

```
http://localhost:8080/jnlpJars/jenkins-cli.jar
```

Ejecutar el comando **login**, para que CLI recuerde el login hasta que se cierre la sesión.

```
java -jar jenkins-cli.jar -s http://localhost:8080 login --username  
admin --password admin
```

Ejecutar el comando **groovy** indicando el path de un fichero Groovy, para ejecutar



scripts de **Groovy**.

```
java -jar jenkins-cli.jar -s http://localhost:8080 groovy  
fichero_script.groovy
```

Un script de Groovy de ejemplo, que recorre los ficheros de la instalación, indicando aquellos de gran tamaño podría ser.

```
root = jenkins.model.Jenkins.instance.getRootDir()  
count = 0  
size = 0  
maxsize=1024*1024*32  
root.eachFileRecurse() { file ->  
    count++  
    size+=file.size();  
    if (file.size() >maxsize) {  
        println "Thinking about deleting: ${file.getPath()}"  
    }  
}  
println "Space used ${size/(1024*1024)} MB Number of files ${count}"
```

Otro script de Groovy de ejemplo, que recorre los **Jobs** creados en Jenkins, comprobando si la última construcción correcta es del año en curso.



```

def warning='<font color=\'red\'>[ARCHIVE]</font> '
def now=new Date()

for (job in hudson.model.Hudson.instance.items) {
    println "\nName: ${job.name}"
    Run lastSuccessfulBuild = job.getLastSuccessfulBuild()
    if (lastSuccessfulBuild != null) {
        def time = lastSuccessfulBuild.getTimestamp().getTime()
        if (now.year.equals(time.year)){
            println("Project has same year as build");
        }else {
            if (job.description.startsWith(warning)){
                println("Description has already been changed");
            }else{
                job.setDescription("${warning}${job.description}")
            }
        }
    }
}
}

```

Ejecutar el comando **logout**, para que CLI olvide el login.

```
java -jar jenkins-cli.jar -s http://localhost:8080 logout.
```

En [esta](#) pagina se tiene un conjunto de scripts de ejemplo de tareas que se pueden hacer.

## 7.12. Consola de Script Integrada

En la administración de Jenkins, hay una consola integrada, que permite ejecutar scripts de Groovy.



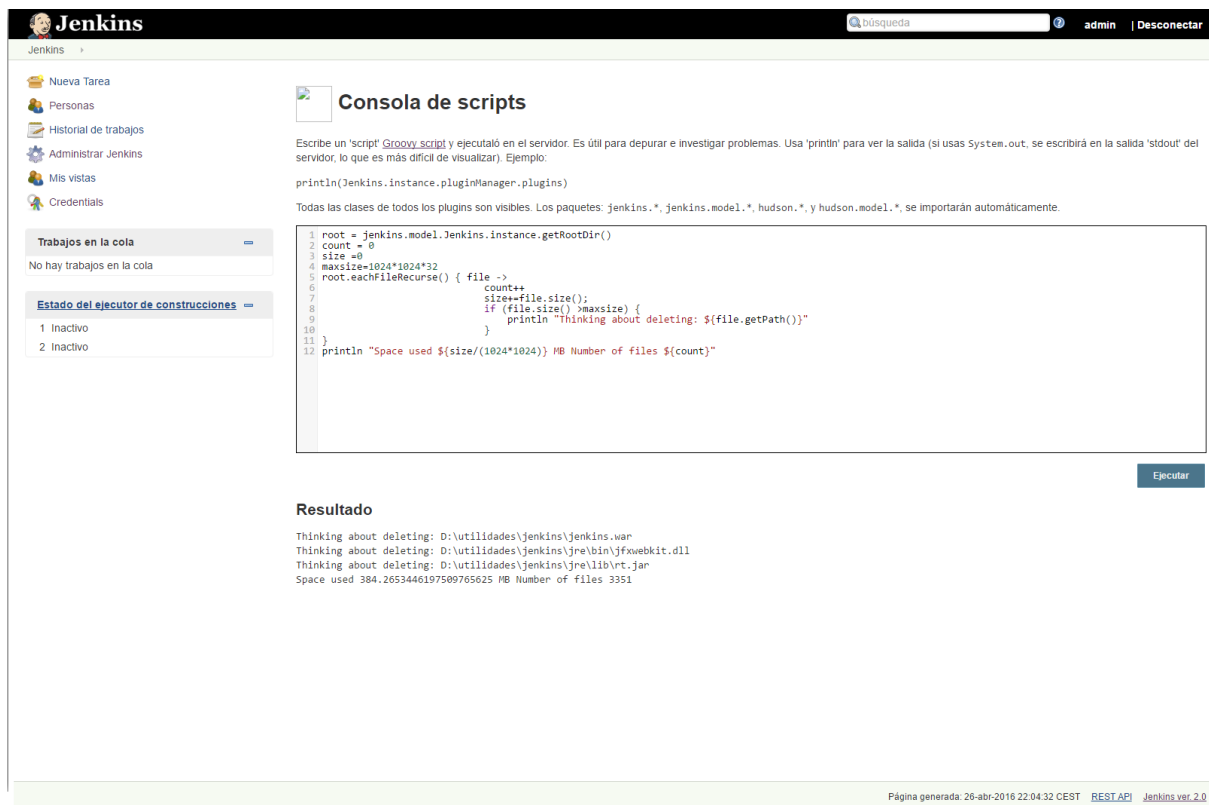
# Integración Continua con Jenkins



The screenshot shows the Jenkins administration interface. On the left sidebar, there are links for 'Nueva Tarea', 'Personas', 'Historial de trabajos', 'Administrar Jenkins', 'Mis vistas', and 'Credentials'. The main content area is titled 'Administrar Jenkins' and lists various administrative tasks. The 'Consola de scripts' option is highlighted with a red box. Below it, there is a description of the console script functionality.

**Administrar Jenkins**

- Configurar el Sistema**: Configurar variables globales y rutas.
- Configuración global de la seguridad**: Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización)
- Global Tool Configuration**: Configure tools, their locations and automatic installers.
- Actualizar configuración desde el disco duro**: Descartar todos los datos cargados en memoria y actualizar todo nuevamente desde los ficheros del sistema. Útil cuando se modifican ficheros de configuración directamente en el disco duro.
- Administrar Plugins**: Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.
- Información del sistema**: Muestra información del entorno que puedan ayudar a la solución de problemas.
- System Log**: El log del sistema captura la salidad de la clase java.util.logging en todo lo relacionado con Jenkins.
- Estadísticas de Carga**: Comprobar la utilización de los recursos y comprobar si es necesario añadir nuevos nodos para la ejecución de tareas.
- Jenkins CLI**: Accede y administra Jenkins desde la consola, o desde scripts.
- Consola de scripts**: Ejecutar script para la administración, diagnóstico y solución de problemas.
- Administrar Nodos**: Añadir, borrar, gestionar y monitorizar los nodos sobre los que Jenkins ejecuta tareas.
- Gestión de credenciales**: Crear/borrar/modificar las credenciales que pueden ser usadas por Jenkins y por las tareas que se ejecutan en él para conectar con servicios de terceros
- Acerca de Jenkins**: Eche un vistazo a la información sobre la versión y la licencia.
- Datos antiguos**: Scrub configuration files to remove remnants from old plugins and earlier versions.
- Gestión de usuarios**: Crear/borrar/editar usuarios que puedan utilizar Jenkins
- In-process Script Approval**: Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.
- Preparar Jenkins para apagar el contenedor**: Detener la ejecución de nuevas tareas para que el sistema pueda apagarse de manera segura.



The screenshot shows the Jenkins 'Consola de scripts' page. The page title is 'Consola de scripts'. Below the title, there is a description of the console script functionality. A Groovy script is pasted into the text area, and the 'Ejecutar' button is clicked. The output of the script is displayed in the 'Resultado' section.

**Consola de scripts**

Escribe un 'script' [Groovy script](#) y ejecutalo en el servidor. Es útil para depurar e investigar problemas. Usa 'println' para ver la salida (si usas System.out, se escribirá en la salida 'stdout' del servidor, lo que es más difícil de visualizar). Ejemplo:

```
println(Jenkins.instance.pluginManager.plugins)
```

Todas las clases de todos los plugins son visibles. Los paquetes: jenkins.\*, jenkins.model.\*, hudson.\*, y hudson.model.\*, se importarán automáticamente.

```
1 root = Jenkins.instance.getRootDir()
2 count = 0
3 size = 0
4 maxSize = 1024 * 1024 * 32
5 root.eachFileRecurse() { file ->
6     count++
7     size += file.size()
8     if (file.size() > maxSize) {
9         println "Thinking about deleting: ${file.getPath()}"
10    }
11 }
12 println "Space used ${size/(1024*1024)} MB Number of files ${count}"
```

**Resultado**

```
Thinking about deleting: D:\utilidades\jenkins\jenkins.war
Thinking about deleting: D:\utilidades\jenkins\jre\bin\jfxwebkit.dll
Thinking about deleting: D:\utilidades\jenkins\jre\lib\rt.jar
Space used 384.2653446197509765625 MB Number of files 3351
```

## 7.13. API de acceso remoto

Desde el API de acceso remoto, se puede entre otras cosas,

- Lanzar un build de un tarea.



- Deshabilitar/Habilitar una tarea
- Borrar una tarea.

Se puede acceder desde

```
http://localhost:8080/job/<Nombre del Job>/api/
```

### 7.14. Ejecución parametrizada

Se pueden definir variables en la construcción de las tareas en Jenkins, que permitan cambiar el comportamiento de la construcción en cada momento.

Para ello se ha de definir el parametro en la sección inicial **Esta ejecución debe parametrizarse**.

☒ Esta ejecución debe parametrizarse ?

**Parámetro de cadena** ?

Nombre  ?

Valor por defecto  ?

Descripción ?

[Plain text] [Visualizar](#)

[Borrar](#)

[Añadir un parámetro](#) ▼

Una vez definido el parametro, este se puede incluir en cualquier zona de la configuración, empleando \$

**Proyecto**

Fichero POM raíz  ?

Goles y opciones  ?

[Avanzado...](#)

Lo mas habitual con **Tareas Maven** es emplear los parametros para seleccionar el **profile** de Maven

```
mvn clean install -P produccion
```

Se puede lanzar la Tarea parametrizada de forma remota, indicando



```
http://localhost:8080/job/MiTarea/buildWithParameters?GOAL=clean
```

Los parámetros empleados en cada una de las ejecuciones de la tarea, se almacenan en la propia Tarea



## 7.15. Tarea Multiconfiguración

Este tipo de proyectos incluyen la **Matriz de Configuración**, que permite definir un parámetro de configuración, con los posibles valores que puede tomar, y por cada uno de los valores definidos, se creará una **SubTarea**.

Por defecto las **SubTareas** se ejecutarán de forma paralela, pero en ocasiones esto no será recomendable, ya que pueden necesitar el mismo recurso de forma simultánea, y el código puede no contemplar la concurrencia (porque no tenga sentido, son en realidad el mismo proyecto corriendo con distintas configuraciones), en este caso, se puede marcar **Run each configuration sequentially**, que ejecutará las **Subtareas** de forma secuencial.

Si se define más de un **Eje** (variable), se ejecutarán todas las posibles combinaciones con los valores de los **Ejes**, sino se desea que se ejecuten todas las posibles combinaciones, se deberá definir un **Filtro de combinación**

Los **Filtros de combinación** definen lo que se ha de cumplir para que se cree una **SubTarea**

```
(browser=="firefox") || (browser=="iexplorer" && os=="windows") ||  
(browser=="chrome" && os != "linux")
```

Los **Ejes** definidos, se pasan como parámetros Maven a la construcción (-D<nombre del parametro>=<valor del parametro>), además de poder ser empleados en la configuración de la **Tarea de Jenkins**, ya que son parámetros de Jenkins (\$<nombre





del parametro>).

## 7.16. Dependencias entre proyectos

Dentro de la configuración de una **Tarea**, se puede indicar que se ejecute otra **Tarea** al finalizar la actual, para ello se acude **Acciones a ejecutar después** y se incluye una referencia al proyecto hijo.


Acciones para ejecutar después.

---

 Ejecutar otros proyectos 

Proyectos a ejecutar

☒ Trigger only if build is stable  
☐ Lanzar incluso si el resultado de la ejecución fué inestable.  
☐ Lanzar incluso si la ejecución acabó con errores.

 **Borrar**

Las **Tareas Hijas**, se ejecutarán dependiendo del resultado de la ejecución de la **Tarea Padre** y de la configuración establecida, pudiendo ser esta:

- Lanzar solo si la ejecución es estable.
- Lanzar aunque la ejecución no sea estable.
- Lanzar aunque la ejecución haya finalizado con errores.

## 7.17. Ejecución Distribuida


Se pueden definir nodos secundarios sobre los que delegar la ejecución de las tareas, para ello, se ha de definir el nodo secundario en el nodo principal desde **Administrar Jenkins→Administrar Nodos→Nuevo nodo**

Lo primero es indicar el tipo de nodo, solo podrá ser **Pasivo**.

Nombre del nodo

☒ **Secundario pasivo**  
 Añadir un esclavo pasivo a Jenkins. Es llamado 'pasivo' porque Jenkins no provee ningún tipo de integración de alto nivel con estos esclavos, como pueda ser aprovisionamiento dinámico. Selecciona este tipo si no hay ningún otro tipo mas adecuado. Por ejemplo cuando se añaden maquinas físicas o virtuales gestionadas desde fuera de Jenkins, etc.

☐ Copiar un nodo existente  
 Copiar desde



Una vez definido, se ha de configurar indicando:

- Numero de ejecutores.








- Directorio Raíz remoto.
- Cuando usar.
  - Usar tanto como sea posible
  - Usar solo con tareas asociadas directamente a el.
- Modo de ejecución.
  - Arrancar agente remotos Linux, via SSH.
  - Arrancar con un comando desde el nodo principal.
  - Ejecutar empleando JNLP
  - Permitir al esclavo que se inicie como servicio windows
- Disponibilidad.
  - Mantener el nodo en linea todo lo que sa posible.
  - Poner en linea cuando se necesite.
  - Programar cuando esta en linea.

En Windows se suele emplear la opcion de **Modo de ejecucion** la de **Ejecutar empleando JNLP**, para arrancarlo, se ha de ejecutar

```
java -jar slave.jar -jnlpUrl http://localhost:8081/computer/<Nombre de esclavo>/slave-agent.jnlp
```

Donde el fichero **slave.jar** esta en **%JENKINS\_HOME%\war\WEB-INF\slave.jar**.

Una vez arrancado, se vera como sincronizado

S	Nombre ↓	Arquitectura	Diferencia entre los relojes	Espacio de disco libre	Espacio de intercambio libre	Espacio temporal libre	Tiempo de respuesta
	<a href="#">Esclavo</a>	Windows 10 (amd64)	Sincronizados	N/A	4,49 GB	142,94 GB	4041ms 
	<a href="#">principal</a>	Windows 10 (amd64)	Sincronizados	142,94 GB	4,49 GB	142,94 GB	0ms 
Data obtained		12 Min	12 Min	12 Min	12 Min	12 Min	12 Min

[Actualizar el estado](#)

El siguiente paso será configurarlo para que se ejecuten las tareas en el, por un lado habra que configurar el **Nodo** con etiquetas, que definan para que se ha de emplear.



## Integración Continua con Jenkins

Nombre	<input type="text" value="Esclavo"/>	?
Descripción	<input type="text"/>	?
Número de ejecutores	<input type="text" value="1"/>	?
Directorio raíz remoto	<input type="text" value="C:\slave"/>	?
Etiquetas	<input type="text" value="performance integration-test"/>	?

Y por otro, en las tareas, activando la opción **Restringir dónde se puede ejecutar este proyecto**, indicar las etiquetas que indicaran en que nodo se ha de ejecutar la tarea.

image::jenkins\_nodo\_esclavo\_seleccion\_de\_nodo\_en\_tarea\_por\_etiqueta.png[]

En este campo, se pueden emplear expresiones booleanas como las siguientes

```
performance //Nodos con la etiqueta performance

!performance //Nodos sin la etiqueta performance

linux && postgres //Nodos con las etiquetas linux y postgres

"Windows 7" || "Windows XP" //Nodos con las etiquetas "Windows 7" o
"Windows XP"

windows -> "Windows 7" //Si existe la etiqueta "windows", debe existir
la etiqueta "Windows 7"

windows <-> "Windows 7" //Si existe la etiqueta "windows", debe existir
la etiqueta "Windows 7", pero sino existe windows, tampoco puede
existir "Windows 7"
```

include::Jenkins\12-Ejecucion\_Distribuida\_Con\_Docker.adoc[ ]

