

# Integracion Continua con Jenkins

Victor Herrero Cazurro

# Table of Contents

1. Integración Continua .....	1
1.1. Introducción .....	1
1.2. Buenas practicas para la CI .....	1
1.3. ¿Porque Integracion Continua? .....	2
2. Jenkins .....	2
2.1. Introducción .....	2
2.2. Instalación .....	2
2.3. Configuración .....	4
2.4. Seguridad y Gestion de usuarios .....	7
2.5. Tareas (Jobs) .....	9
2.6. Plugins .....	12
2.7. Scripting con Jenkins CLI .....	17
2.8. Consola de Script Integrada .....	18
2.9. API de acceso remoto .....	19
2.10. Ejecución parametrizada .....	20
2.11. Tarea Multiconfiguración .....	21
2.12. Dependencias entre proyectos .....	21
2.13. Ejecución Distribuida .....	22
3. Calidad estática del código .....	24
3.1. Calidad del Código .....	24
3.2. Análisis Estático del Código .....	24
3.3. PMD .....	25
3.4. Checkstyle .....	26
3.5. Findbugs .....	27
3.6. Cobertura .....	28
3.7. Jacoco .....	28
4. Sonarqube .....	32
4.1. Introducción .....	32
4.2. Instalación .....	33
4.3. Conceptos .....	34
4.4. Organizacion .....	34
4.5. Carga de datos .....	35
4.6. Gestion de Usuarios y Seguridad .....	36
4.7. Cuadro de mando .....	36

# 1. Integración Continua

## 1.1. Introducción

Modelo propuesto inicialmente por [Martin Fowler](#) que consiste en automatizar un proceso sobre los proyectos, comprendiendo este proceso los siguientes pasos

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

Este proceso persigue la vigilancia del código, para la detección temprana de errores.

Para esto se utilizan aplicaciones como

- Bamboo
- Continuum
- Hudson
- Jenkins
- CruiseControl
- Team Foundation Server

## 1.2. Buenas practicas para la CI

Según Martin Fowler, se deben seguir las siguientes buenas practicas

- Mantener el código versionado con un SCM (Git, SVN, CVS, ...).
- Automatizar la construcción (Jenkins, Bamboo, ...)
- Crear Test que permitan tener confianza en el código generado (JUnit, Selenium, SoapUI, ...).
- Commits diarios al SCM, para que el servidor de CI, ofrezca una imagen real del desarrollo.
- Cada commit debería forzar un build en el servidor de CI.
- Aviso inmediato al autor/equipo del commit que introduce una inestabilidad.
- Mantener una construcción rápida, que el proceso de CI no sea vital, no quiere decir que pueda tardar mucho en ejecutarse.
- Pruebas sobre un clon de producción.
- Obtención fácil de las construcciones (Releases, snapshot) a través de servidores de artefactos.

- Visibilidad del proceso de IC y de los reportes para todos los miembros del equipo.

## 1.3. ¿Porque Integracion Continua?

Normalmente en los proyectos las entregas son el punto mas caliente, no solo porque supone culminar un trabajo, sino porque que se llega a ellas con poca información del estado del proyecto.

- ¿Cuanto se tarda en desplegar?
- ¿Tiene defectos la aplicación?
- ¿Cuántos tests han pasado?¿De que tipo son?
- ¿Cómo es el código de robusto?

Aplicando la IC, se puede obtener este tipo de onformación desde el principio y tener una visión de la evolución.

## 2. Jenkins

### 2.1. Introducción

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

Creado por Kohsuke Kawaguchi. Esta liberado bajo licencia MIT.

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

### 2.2. 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>
```

Tambien 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**.

## 2.3. 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
- Gestion de usuarios


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

- Maven
- Git
- SVN
- CVS

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

## JDK


### instalaciones de JDK

JDK	
Nombre	<input type="text" value="JDK8"/>
JAVA_HOME	<input type="text" value="C:\Program Files\Java\jdk1.8.0_91"/>
<input type="checkbox"/> Instalar automáticamente	
<input type="button" value="Borrar JDK"/>	

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


## Maven

### instalaciones de Maven

Maven	
Nombre	<input type="text" value="Maven 3.3.9"/>
MAVEN_HOME	<input type="text" value="D:\utilidades\apache-maven-3.3.9"/>
<input type="checkbox"/> Instalar automáticamente	
<input type="button" value="Borrar Maven"/>	
<input type="button" value="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)	<input type="text" value="localhost"/>	
sufijo de email por defecto	<input type="text" value="@my-proyecto-jenkins-ci.com"/>	
<input type="checkbox"/> Probar la configuración enviando un correo de prueba		 <input data-bbox="1316 1137 1428 1164" type="button" value="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**.

<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

Dirección URL de Jenkins  ?

System Admin e-mail address  ?

**SSH Server**

SSHD 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  ?

## 2.4. Seguridad y Gestion de usuarios

Por defecto Jenkins permite acceder en modo anonimo 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

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

Tambien 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.

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

## 2.5. Tareas (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**.

Jenkins

admin | Desconectar

Enter an item name

» Required field

**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 múltiples 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.

if you want to create a new item from other existing, you can use this option:

Copy from

OK

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

Jenkins > Proyecto >

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

☐ Desactivar la ejecución  
☐ Lanzar ejecuciones concurrentes en caso de ser necesario

Avanzado...

### Configurar el origen del código fuente

☐ Ninguno  
☐ Git  
☒ Subversion

Módulos

Repository URL:

Credentials:  Add

Local module directory:

Repository depth:

Ignore externals: ☒

Add module...

Additional Credentials:

Check-out Strategy:

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.

Navegador del repositorio:

Avanzado...

### Disparadores de ejecuciones

☐ Lanzar ejecuciones remotas (ejem: desde 'scripts')  
☐ Build after other projects are built  
☐ Build when a change is pushed to GitHub

Guardar Apply

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

### 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

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.

### Ejecutar

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

Version de Maven:

Goals:

Avanzado...

Añadir un nuevo paso ▾

- Ejecutar Ant
- Ejecutar línea de comandos (shell)
- Ejecutar tareas 'maven' de nivel superior
- Ejecutar un comando de Windows
- Invoke Gradle script
- Process Job DSLs
- Set build status to "pending" on GitHub commit

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.

### Publicar los resultados de tests JUnit

Ficheros XML con los informes de tests

**\*\*/target/surefire-reports/\*.xml**

El atributo '@includes' de la etiqueta 'files' especifica dónde están los ficheros XML generados, por ejemplo: 'myproject/target/test-reports/\*.xml'. El directorio base para la etiqueta 'files' es el directorio [raíz](#) del proyecto

☐ Guardar la salida estándar y de error aunque sea muy larga.

Health report amplification factor

**1,0**

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Borrar

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

### Guardar los archivos generados

Ficheros para guardar

**\*\*/target\*.jar**

Avanzado...

Borrar

O la publicación de los **Javadoc** generados

### Publicar Javadoc

Directorio para los javadoc

**target/site/apidocs**

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

☐ Conservar los javadoc para todas las ejecuciones correctas

Borrar

## 2.5.1. 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	

## 2.6. Plugins

### 2.6.1. 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.

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](#)



Una vez configurado Maven, se ha de asegurar que los proyectos emplean esta configuración, en versiones de Jenkins ocurre que se selecciona la version 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:

Goles:

### Acciones para ejecutar después.

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

## 2.6.2. Plugin Sonarqube

Es un pugin que permite conectar Jenkins con Sonar.

Jenkins

Jenkins > Gestor de plugins

[Volver al Panel de control](#) [Administrar Jenkins](#)

Filtrar:

Actualizaciones disponibles   **Todos los plugins**   Plugins instalados   Configuración avanzada

Instalar ↓	Nombre	Versión
<input type="checkbox"/>	<a href="#">Mashup Portlets</a> Additional portlets: 'Generic JS Portlet' flexibly displays any content from another source. 'Recent Changes' and 'Test Results' show useful job information that would otherwise only be available by drilling down into the job. The 'SonarQube Portlet' shows important issues from Sonar directly in Jenkins.	1.0.6
<input checked="" type="checkbox"/>	<a href="#">SonarQube Plugin</a> This plugin allow easy integration of <a href="#">SonarQube™</a> , the open source platform for Continuous Inspection of code quality.	2.4

Update information obtained: 1 día 0 Hor ago

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

## SonarQube servers

Environment variables

☒ Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

Instalaciones de SonarQube

Name

Sonar local

URL del servidor

localhost:9000

Server version

5.3 or higher

Por defecto es http://localhost:9000

Configuration fields depend on the SonarQube server version.

Server authentication token

SonarQube account login

SonarQube account password

SonarQube authentication token. Mandatory when anonymous access is disabled.

SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.

SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.

Add SonarQube

Listado de instalaciones SonarQube

Avanzado...

Delete SonarQube

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

### SonarQube Scanner

instalaciones de SonarQube Scanner

SonarQube Scanner

Name

Sonarqube scanner

☒ Instalar automáticamente

Install from Maven Central

Versión

SonarQube Scanner 2.5.1

Añadir un instalador

Añadir SonarQube Scanner

Listado de instalaciones de SonarQube Scanner en este sistema

Borrar un instalador

Borrar SonarQube Scanner

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

### 2.6.3. 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

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			<input type="text" value="98"/>		<input type="text" value="75"/>		<input type="text" value="75"/>
Lines	Delete		<input type="text" value="98"/>		<input type="text" value="75"/>		<input type="text" value="75"/>
Methods	Delete		<input type="text" value="100"/>		<input type="text" value="80"/>		<input type="text" value="80"/>
Packages	Delete		<input type="text" value="100"/>		<input type="text" value="95"/>		<input type="text" value="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).

### 2.6.4. 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

## 2.6.5. 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

## 2.6.6. Disk Usage Plugin

Permite monitorizar el uso del disco.

## 2.6.7. Backup Plugin

Aunque el Backup de Jenkins es facil de realiza, 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.

## 2.6.8. Dependency Graph Viewer Plugin

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

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

## 2.6.9. Maven Release Plug-in

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

### 2.6.10. 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](#)

## 2.7. Scripting con Jenkins CLI

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 sesion.

```
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 fichers de la instalación, indicando aquellos de gran tamaño podria 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.
```

## 2.8. Consola de Script Integrada

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

Jenkins

admin

Desconectar

busqueda

ACTIVAR AUTO REFRESCO

Nueva Tarea

Personas

Historial de trabajos

Administrar Jenkins

Mis vistas

Credentials

Trabajos en la cola

No hay trabajos en la cola

Estado del ejecutor de construcciones

1 Inactivo

2 Inactivo

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.

Jenkins

busqueda

admin

Desconectar

Nueva Tarea

Personas

Historial de trabajos

Administrar Jenkins

Mis vistas

Credentials

Trabajos en la cola

No hay trabajos en la cola

Estado del ejecutor de construcciones

1 Inactivo

2 Inactivo

Consola de scripts

Escribe un 'script' Groovy script y ejecútalo 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}"
```

Ejecutar

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

## 2.9. 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/Proyecto/api/
```

## 2.10. 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 parametros empleados en cada una de las ejecuciones de la tarea, se almacenan en la propia Tarea



## 2.11. Tarea Multiconfiguración

Este tipo de proyectos incluyen la **Matriz de Configuración**, que permite definir un parametro 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 simultanea, 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 mas 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 parametros Maven a la construcción, además de poder ser empleados en la configuración de la **Tarea** de **Jenkins**.

## 2.12. 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 despues** y se incluye una referencia al proyecto hijo.

#### Ejecutar otros proyectos

Proyectos a ejecutar

MiTarea

- ☒ 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 condifugración 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.

## 2.13. 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

Esclavo-Windows

#### ☒ 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

OK

Una vez definido, se ha de configurar indicando:

- Numero de ejecutores.
- Directorio Raiz 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 empelando 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.

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"
```

## 3. Calidad estática del código

### 3.1. Calidad del Código

Decimos que un código tiene calidad, cuando tenemos facilidad de mantenimiento y de desarrollo.

¿Como podemos hacer que nuestro código tenga mas calidad? Consiguiendo que nuestro código no tenga partes que hagan que:

- Se reduzca el rendimiento.
- Se provoquen errores en el software.
- Se compliquen los flujos de datos.
- Lo hagan mas complejo.
- Supongan un problema en la seguridad.

Tendremos dos técnicas para mejorar el código fuente de nuestra aplicación y, con ello, el software que utilizan los usuarios como producto final:

- Test. Son una serie de procesos que permiten verificar y comprobar que el software cumple con los objetivos y con las exigencias para las que fue creado.
- Análisis estático del código. Proceso de evaluar el software sin ejecutarlo.

### 3.2. Análisis Estático del Código

Es una técnica que se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo.

La idea es que, en base a ese código fuente, podamos obtener información que nos permita mejorar la base de código manteniendo la semántica original.

Esta información nos vendrá dada en forma de sugerencias para mejorar el código.

Emplearemos herramientas que incluyen

- Analizadores léxicos y sintácticos que procesan el código fuente.
- Conjunto de reglas que aplicar sobre determinadas estructuras.

Si nuestro código fuente posee una estructura concreta que el analizador considere como "mejorable" en base a sus reglas nos lo indicará y nos sugerirá una mejora.

Se deberían realizar análisis estáticos del código cada vez que se crea una nueva funcionalidad, así como cuando el desarrollo se complica, nos cuesta implementar algo que supuestamente debe ser sencillo.

### 3.3. PMD

Detecta patrones de posibles errores que pueden aparecer en tiempo de ejecución, por ejemplo

- Código que no se puede ejecutar nunca porque no hay manera de llegar a él.
- Código que puede ser optimizado.
- Expresiones lógicas que puedan ser simplificadas.
- Malos usos del lenguaje, etc
- También incluye detección de CopyPaste (CPD)

La pagina de referencia [aquí](#)

Los patrones que se emplean se encuentran catalogados en distintas categorías, se pueden consultar [aquí](#)

Se pueden añadir nuevas reglas o configurar las que ya se incluyen en caso de que esto fuera necesario.

Se dispone de un plugin de Maven para la generación de reportes

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>3.6</version>
      <configuration>
        <linkXref>true</linkXref>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

Este plugin también puede ser configurado como plugin de la fase de Test

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>3.6</version>
      <configuration>
        <failOnViolation>true</failOnViolation>
        <failurePriority>2</failurePriority>
        <minimumPriority>5</minimumPriority>
      </configuration>
      <executions>
        <execution>
          <phase>test</phase>
          <goals>
            <goal>pmd</goal>
            <goal>cpd</goal>
            <goal>cpd-check</goal>
            <goal>check</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Con los parametros

- **failOnViolation** → le indicamos que si hay fallos, haga que la fase de Test falle, supeditamos el exito de los Test al analisis de PMD
- **failurePriority** → le indicamos a partir de que prioridad se considera fallo, las prioridades van de 1 a 5, siendo 1 la maxima y 5 la menor, si se define por ejemplo 2, solo se consideran las reglas con prioridad 1 y 2.
- **minimumPriority** → Minima prioridad de las reglas a evaluar.

Y los goal

- **pmd** → Ejecuta las reglas de pmd
- **cpd** → Ejecuta las reglas de cpd
- **cpd-check** → Chequea los resultados de cpd
- **check** → Chequea los resultados de pmd

## 3.4. Checkstyle

Inicialmente se desarrolló con el objetivo de crear una herramienta que permitiese comprobar que el código de las aplicaciones se ajustase a los estándares dictados por **Sun Microsystems**.

Posteriormente se añadieron nuevas capacidades que han hecho que sea un producto muy similar a PMD. Es por ello que también busca patrones en el código que se ajustan a categorías muy similares a las de este analizador.

La página de referencia [aquí](#)

Dispone de un plugin de Maven

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.9.1</version>
    </plugin>
  </plugins>
</reporting>
```

## 3.5. Findbugs

Es un producto de la Universidad de Maryland que, como su nombre indica, está especializado en encontrar errores.

Tiene una serie de categorías que catalogan los errores

- malas prácticas
- mal uso del lenguaje
- internacionalización
- posibles vulnerabilidades
- mal uso de multihilo
- rendimiento
- seguridad, ...

La página de referencia [aquí](#) y la descripción de los bugs [aquí](#)

Hay un plugin de maven

```

<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>findbugs-maven-plugin</artifactId>
      <version>2.5.2</version>
    </plugin>
  </plugins>
</reporting>

```

## 3.6. Cobertura

La Cobertura, representa la cantidad de código que cubren las pruebas realizadas sobre el código.

Existe un plugin de Maven que permite realizar la medición de la cobertura, presentando el resultado en informes **html** y **xml**.

```

<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>cobertura-maven-plugin</artifactId>
      <version>2.5.2</version>
      <configuration>
        <formats>
          <format>xml</format>
          <format>html</format>
        </formats>
      </configuration>
    </plugin>
  </plugins>
</reporting>

```

## 3.7. Jacoco

Formado por las primeras sílabas de **Java Code Coverage**, es otro plugin de cobertura.

La página de referencia se encuentra [aquí](#)

```

<build>
  <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.5.201505241946</version>
      <executions>
        <execution>

```

```

<id>pre-unit-test</id>
<goals>
  <goal>prepare-agent</goal>
</goals>
<configuration>
  <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
  <destFile>${project.build.directory}/jacoco-ut.exec</destFile>
  <!-- Establece la propiedad que contiene la ruta del agente
Jacoco para las pruebas unitarias-->
  <propertyName>surefireArgLine</propertyName>
</configuration>
</execution>
<execution>
  <id>post-unit-test</id>
  <phase>test</phase>
  <goals>
    <goal>report</goal>
  </goals>
  <configuration>
    <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
    <dataFile>${project.build.directory}/jacoco-ut.exec</dataFile>
    <!-- Establece la ruta donde se genera el reporte para pruebas
unitarias -->
    <outputDirectory>${project.reporting.outputDirectory}/jacoco-
ut</outputDirectory>
  </configuration>
</execution>
<execution>
  <id>pre-integration-test</id>
  <phase>pre-integration-test</phase>
  <goals>
    <goal>prepare-agent</goal>
  </goals>
  <configuration>
    <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
    <destFile>${project.build.directory}/jacoco-it.exec</destFile>
    <!-- Establece la propiedad que contiene la ruta del agente
Jacoco para las pruebas de integracion -->
    <propertyName>failsafeArgLine</propertyName>
  </configuration>
</execution>
<execution>
  <id>post-integration-test</id>
  <phase>post-integration-test</phase>
  <goals>
    <goal>report</goal>
  </goals>
  <configuration>

```

```

        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
        <dataFile>${project.build.directory}/jacoco-it.exec</dataFile>
        <!-- Establece la ruta donde se genera el reporte para pruebas
de integracion -->
        <outputDirectory>${project.reporting.outputDirectory}/jacoco-
it</outputDirectory>
    </configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>

```

Se ha de configurar igualmente que el agente sea ejecutado en las distintas fases de **test** e **integration-test**, indicando en el plugin con **argLine** la ubicacion del agente de **jacoco** que debera generar los datos del analisis.



```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.12.4</version>
      <configuration>
        <argLine>${surefireArgLine}</argLine>
        <excludes>
          <exclude>**/integracion/*.java</exclude>
        </excludes>
        <includes>
          <include>**/unitarias/*.java</include>
        </includes>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>2.8</version>
      <configuration>
        <argLine>${failsafeArgLine}</argLine>
        <excludes>
          <exclude>**/unitarias/*.java</exclude>
        </excludes>
        <includes>
          <include>**/integracion/*.java</include>
        </includes>
      </configuration>
      <executions>
        <execution>
          <id>pasar test integracion</id>
          <phase>integration-test</phase>
          <goals>
            <goal>integration-test</goal>
          </goals>
        </execution>
        <execution>
          <id>validar pruebas integracion</id>
          <phase>verify</phase>
          <goals>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

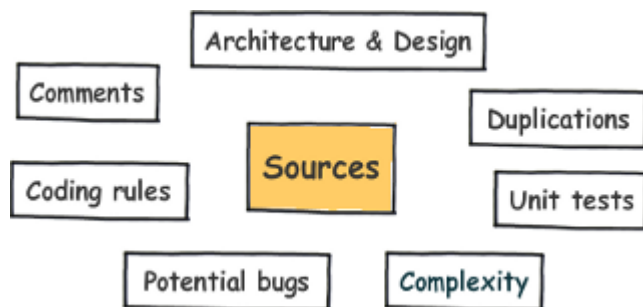
Finalmente se pueden añadir los resultados del análisis al sitio añadiendo

```
<reporting>
  </plugins>
  <plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.7.5.201505241946</version>
  </plugin>
</plugins>
</reporting>
```

## 4. Sonarqube

### 4.1. Introducción

Herramienta que centraliza otras herramientas que analizan la calidad estática del código de un proyecto. Cubre 7 ejes principales de la calidad del software



Ofrece información sobre

- Cobertura
- Complejidad ciclomatica.
- Buenas practicas.

A traves de herramientas como

- Checkstyle
- PMD
- FindBugs

Hay disponible una demo con APIs conocidas [aquí](#)

También hay un grupo español, que ofrece información [aquí](#)

Algunos de los plugins mas interesantes de Sonar

- PDF Export. Plugin que permite generar pdf con la info de Sonar
- Motion Chart. Plugin que permite mostrar graficos en movimiento con la evolucion de las metricas

- Timeline. Plugin que visualiza el historico de las metricas
- Sonargraph. Plugin enables you to check and measure the overall coupling and the level of cyclic dependencies
- Taglist. Plugin handles Checkstyle ToDoComment rule and Squid NoSonar rule and generates a report.

## 4.2. Instalación

Descargar la distribución de [aquí](#)

Configurar la base de datos en el fichero

```
SONAR_HOME/conf/sonar.properties
```

Estos son los posibles valores para mysql, de no configurarse se empleará una base de datos Derby, no recomendable para entornos de producción.

```
sonar.jdbc.url:
jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8
sonar.jdbc.driverClassName: com.mysql.jdbc.Driver
sonar.jdbc.validationQuery: select 1
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
```

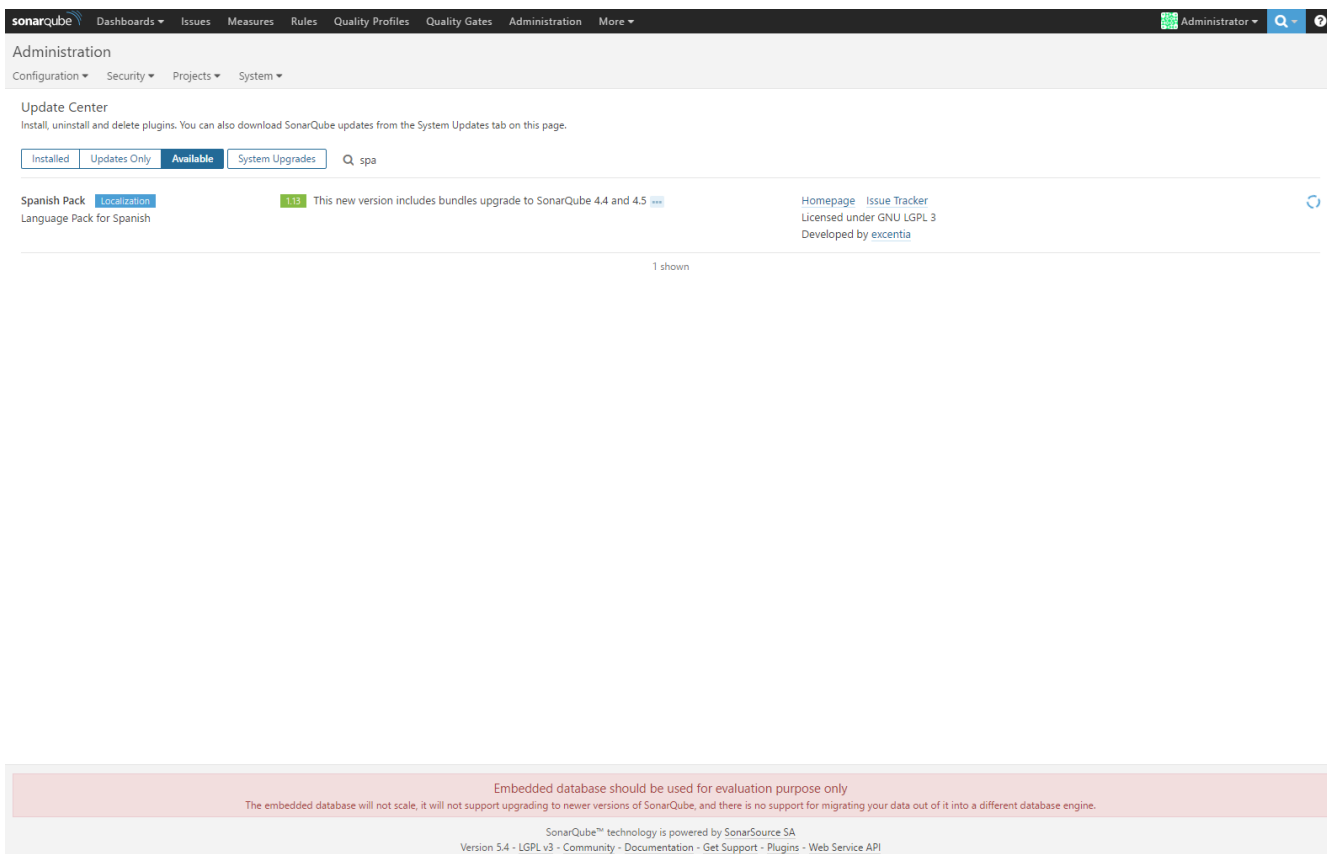
Arrancar el servidor con el comando para el sistema operativo correspondiente que se encuentra en

```
SONAR_HOME/bin/<sistema operativo>/<comando>
```

Esta opción arranca Sonar en el puerto **9000**, el puerto tambien se puede configurar en el anterior fichero de configuración.

Existe un usuario administrador creado por defecto con **admin/admin**

Se puede instalar un plugin para el idioma español, para ello acceder a **Administration/System/Update Center/Available Plugins** y buscar el **Spanish Pack**



## 4.3. Conceptos

**Deuda tecnica:** Es un calculo basado únicamente en **reglas y evidencias**. La deuda técnica en Sonar, se calcula con la metodología SQALE (Software Quality Assessment based on Lifecycle Expectations). Se mide en dias.

**Reglas:** Representan aquellos puntos que se desean vigilar en los proyectos. Se pueden definir con un perfil de calidad. Se pueden obtener mas reglas a partir de nuevos Plugins. Se pueden definir nuevas reglas basadas en plantillas.

**Evidencias:** Son los incumplimientos de las Reglas de calidad que se presentan en el código. Tendrán asociada una severidad. Con las evidencias se pued hacer: Comentar, Asignar, Planificar, Confirmar, Cambiar Severidad, Resolver y Falso Positivo. Todas estas tareas se pueden realizar de forma individual o conjunta. Se pueden definir evidencias manuales.

## 4.4. Organizacion

La interface web de sonar, se divide en tres partes

- Menu superior
- Menu lateral
- Zona de visualizacion de datos

En el menu superior aparecen los siguientes items

- Cuadros de mando para volver en cualquier momento a la página de inicio

- Proyectos para acceder al listado completo de proyectos, vistas, desarrolladores, etc. o para acceder de forma rápida a proyectos recientemente accedidos
- Medidas, permite definir consultas sobre las medidas, se pueden guardar para visualizarlas en un cuadro de mando.
- Evidencias para acceder al servicio de evidencias
- Reglas para acceder a la página de reglas
- Perfiles navegar y gestionar perfiles de calidad
- Configuración para acceder a la configuración del sistema (acceso restringido a administradores de sistema)
- Conectarse / <Nombre> para conectarse con tu usuario. Dependiendo de tus permisos de usuario, tendrás acceso a diferentes servicios y cuadros de mando. Autenticarte en el sistema te permitirá tener acceso a tu propia interfaz web personalizada. Desde aquí puedes modificar tu perfil y desconectarte.
- Buscar un componente: proyecto, fichero, vista, desarrollador, etc. para acceder rápidamente a él. Pulsa 's' para acceso directo a la caja de búsqueda.

El menú lateral ira cambiando sus opciones dependiendo del área en la que nos encontremos, de los permisos de usuario y de las extensiones que se hayan incorporado en la instalación. Proporciona acceso a diferentes cuadros de mando y servicios.

## 4.5. Carga de datos

Para cargar los datos de un proyecto, se puede hacer de varias formas, la mas habituales son

- A través de un plugin de Maven.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>sonar-maven-plugin</artifactId>
  <version>2.7</version>
</plugin>
```

Y su goal

```
mvn sonar:sonar
```

Para la seleccion de un perfil de Sonar a emplear, se ha de indicar el parametro **sonar.profile**

```
-Dsonar.profile="Mi Perfil"
```

Si se desea publicar la cobertura en Sonar, se ha de seguir los pasos que se pueden encontrar en <https://github.com/SonarSource/sonar-examples/blob/master/projects/lenguajes/java/code-coverage>

- A través del plugin de jenkins sonarqube.

## 4.6. Gestion de Usuarios y Seguridad

Se pueden añadir nuevos usuarios, grupos, definir permisos a nivel global o de proyecto, desde **Administration/Security/Users**

## 4.7. Cuadro de mando

Los cuadros de mando, son los componentes principales de Sonar, ya que son los que nos permiten configurar que información de que proyecto queremos visualizar y como visualizarlo.

Se organizan en columnas, pudiendo seleccionar entre 5 distribuciones distintas.

Se dividen en **Widget**, habiendo **Widget** orientados a distintos propositos, si se busca un Widget concreto se pueden filtrar los Widget mostrados por categorias.

Los Widget a parte de mostrar información, permiten acceder a vistas mas avanzadas, ya que en general los Widget ofrecen resúmenes de la información.

Existen Widget que ofrecen información sobre

- Tamaño de los ficheros
- Bloques duplicados
- Mala distribución de la complejidad
- Código Spaguetti
- Falta de pruebas unitarias
- Cumplimiento de estándares y defectos potenciales
- Contabilización de comentarios
- Eventos en cuanto a la calidad.
- Treemap
- Evidencias y deuda técnica.
- Pirámide de deuda técnica. Muestra la deuda técnica organizada de abajo arriba por prioridad en su resolución.
- Resumen de deuda técnica. Ofrece un Ratio entre lo que se necesita invertir para solventar la deuda técnica y lo que se necesita invertir para crear el proyecto desde cero.

Mi Cuadro de mando

Volver al cuadro de mando

Category: **Cualquier** Filters History Hotspots Issues Technical Debt Tests

Alertas

Mostrar alertas del proyecto.

Añade un widget

Descripción

Muestra información general relativa a un proyecto

Añade un widget

Bienvenido

Mensaje de bienvenida para proporcionar enlaces a los recursos más valiosos como documentación y soporte

Añade un widget

Documentación y Comentarios

Informa sobre comentarios y documentación del código

Añade un widget

Cobertura de código

Muestra los resultados de la ejecución de tests y de su cobertura

Añade un widget

Duplicados

Informa sobre copiar/pegar y duplicados en el código

Añade un widget

Cobertura tests de integración

Informa de la cobertura de código de los tests de integración

Añade un widget

Eventos

Muestra los eventos ocurridos en la vida de un proyecto como versiones o alertas.

Añade un widget

Complejidad

Muestra la complejidad total, media y su distribución.

Añade un widget

Evidencias y deuda técnica.

Muestra información de las evidencias y la deuda técnica.

Añade un widget

Search:

Embedded database should be used for evaluation purpose only

The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA  
Version 5.4 - LGPL v3 - Community - Documentation - Get Support - Plugins - Web Service API

37