



Jenkins

Jenkins
Guía de uso

Manuel Tomás Ortega Sánchez

Version 1.1.0 2019-12-09

Table of Contents

1. Introducción	1
1.1. Beneficios	1
1.2. Integración continua (CI)	1
1.3. Entrega continua (CD)	1
1.4. Historia	2
1.5. Plugins	2
2. Instalación	4
2.1. Tipos de compilaciones	4
2.2. Sobre instalaciones en ámbito Windows	4
2.3. Lab: Instalación de Jenkins (.war)	5
3. Administración de Jenkins	10
3.1. Administración de flujos de trabajo (Pipelines)	10
3.2. Administración de plugins	10
3.3. Actualizando Jenkins	11
3.4. Mantenimiento del sistema	11
4. Configuración Básica	12
4.1. Opciones de menú básicas	12
4.2. Configuración de la JDK	13
4.3. Configuración de Maven	13
4.4. Sistema de control de versiones	14
4.5. Configuración descriptiva básica del servidor	14
5. Tareas (Jobs)	15
5.1. Resultado de la ejecución	17
5.2. Variables disponibles en los Jobs	18
5.3. Lab: Construcción de un proyecto Java Maven (Pipeline)	19
6. Proyectos Multiconfiguración	23
6.1. Lab: Proyecto multiconfiguración	23
7. Plugins	25
7.1. Maven Plugin	25
7.2. Plugin Sonarqube	26
7.3. Cobertura Plugin	28
7.4. Copy Artifact plugin	28
7.5. Backup Plugin	29
7.6. Dependency Graph Viewer Plugin	29
7.7. Maven Release Plug-in	29
7.8. Plugin Job DSL	29
7.9. Plugin Project Template	29
7.10. Plugin Deploy To Container	30

7.11. Ansible	42
7.12. Pipelines	54
7.13. Plugin Blue Ocean	70
8. Scripting con Jenkins CLI	85
8.1. Lab: Conectando a Jenkins con cliente SSH	85
9. Consola de Script Integrada	91
10. API de acceso remoto	92
10.1. Lab: Acceso API Remoto	92
11. Ejecución Distribuida	103
11.1. Modelo Master - Minion :)	103
11.2. Lab: Ejecución distribuída	104
12. Roles de usuarios	112
12.1. Tipos de roles	112
12.2. Lab: Roles de usuarios	112
13. Notificaciones	120
13.1. Lab: Notificaciones mediante e-mail	120

Chapter 1. Introducción

Jenkins es un servidor de automatización de código abierto, escalable y altamente eficiente que ejecuta un ecosistema de plugins que le otorgan funcionalidad

Está escrito en lenguaje Java y utiliza complementos diseñados para la Integración Contínua, implementación y automatización de tareas

1.1. Beneficios

Desarrollar y probar software puede llegar a disparar los costes de manera significativa en un proyecto, por requerir ciertos procesos demasiado tiempo para llevarlos a cabo

Personas y empresas están continuamente buscando la automatización de procesos en el área del desarrollo de software que nos hagan más eficientes la cadena de producción final :)

Jenkins permite el uso de funciones y complementos para automatizar el flujo de trabajo y pasar por las fases de integración continua (CI) y de entrega continua (CD)

Mediante la implementación de esas fases, podremos poner en marcha secuencias de compilación, testeo y armado de nuestros sistemas, como también la posterior puesta en marcha de los mismos en los entornos que necesitemos

1.2. Integración continua (CI)

Mediante la implementación de una fase de CI, podríamos llevar a cabo las siguientes acciones:

- Pruebas Unitarias
- Pruebas de Integración
- Pruebas de compilación
- Pruebas de despliegue
 - Por rama...
 - Por funcionalidades...
 - ...
- Análisis de la calidad de código
- Etc.

1.3. Entrega continua (CD)

Mediante la implementación de una fase de CD, podríamos llevar a cabo las siguientes acciones:

- Regeneración / puesta en marcha de los entornos de pre-producción, sqa, producción, etc.
- Empaquetamiento de nuestra aplicación en una tecnología de contenedores como Docker, Rocket, etc.

- Ejecutar instrucciones de alguna herramienta de aprovisionamiento para ejecutar el despliegue, como podría ser Ansible, Puppet, etc.
- Ejecutar instrucciones de alguna herramienta de orquestación de contenedores, como podría ser Docker Swarm, Kubernetes, Apache Mesos, etc.

Perseguimos objetivos como:

- Capacidad de obtener versiones entregables para todo tipo de usuarios finales, de la forma más segura y rápida posible
- Fácil despliegue a pedido, ya sea una pequeña aplicación 'Hello World', o de un gran sistema distribuído

1.4. Historia

Antes de que desapareciera Sun Microsystems, existía un proyecto basado en Java con licencia MIT, cuyo nombre era Hudson, el proyecto nace en torno a 2005

El desarrollador principal de la herramienta fue Kohsuke Kawaguchi, empleado de Sun Microsystems

A raíz de la compra de Sun Microsystems por parte de Oracle en el año 2010, surgen ciertas discrepancias en la comunidad, y nace el proyecto a modo de Fork, rebautizado como Jenkins, ya que Oracle no permitió ceder el nombre de Hudson al proyecto :p

Creado por Kohsuke Kawaguchi. Esta liberado bajo licencia MIT.

1.5. Plugins

Inicialmente al instalar Jenkins, el propio sistema nos sugiere una serie de plugins a instalar por defecto, que son los más usados de ámbito general.

Getting Started

Getting Started

✓ Ant Plugin	✓ OWASP Markup Formatter Plugin	✓ build timeout plugin	✓ Folders Plugin	** Token Macro Plugin Jenkins build timeout plugin Folders Plugin ** Credentials Plugin ** Structs Plugin ** Pipeline: Step API ** Plain Credentials Plugin Credentials Binding Plugin Email Extension Plugin ** SSH Credentials Plugin ** Jenkins Git client plugin ** SCM API Plugin Jenkins Git plugin Jenkins Gradle plugin LDAP Plugin Jenkins Mailer Plugin Matrix Authorization Strategy Plugin PAM Authentication plugin ** JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) plugin ** Durable Task Plugin ** Pipeline: API ** Pipeline: Supporting APIs ** Pipeline: Job ** Pipeline: REST API Plugin ** JavaScript GUI Lib: Handlebars bundle plugin ** JavaScript GUI Lib: Moment.js bundle plugin Pipeline: Stage View Plugin Jenkins SSH Slaves plugin ** MapDB API Plugin Jenkins Subversion Plug-in Timestampper ** Pipeline: Build Step ** JavaScript GUI Lib: ACE Editor bundle plugin ** - required dependency
✓ Credentials Binding Plugin	✓ Email Extension Plugin	✓ Git plugin	✓ Gradle plugin	
✓ LDAP Plugin	✓ Mailer Plugin	✓ Matrix Authorization Strategy Plugin	✓ PAM Authentication plugin	
✓ Pipeline: Stage View Plugin	✓ SSH Slaves plugin	✓ Subversion Plug-in	✓ Timestampper	
🔗 Pipeline	🔗 GitHub Organization Folder Plugin	🔗 Workspace Cleanup Plugin		

El secreto de la potencia de Jenkins, radica en sus plugins, ya que estos nos van a permitir extender las funcionalidades del sistema para adaptarlo a nuestras necesidades

Podemos consultar los plugins existentes en la siguiente URL, existen multitud de ellos disponibles:

URL: <https://plugins.jenkins.io/>

Chapter 2. Instalación

Desde la URL oficial podemos llevar a cabo su instalación bajo diferentes modalidades, accedemos a la URL: <https://jenkins.io/download/>

2.1. Tipos de compilaciones

Jenkins dispone de 2 tipos de compilaciones del producto:

- **Weekly**
 - Compilaciones realizadas semanalmente
 - Ofrecen correcciones de errores y nuevas características rápidamente a los usuarios y desarrolladores de complementos que los necesitan
 - Orientada para propósitos de prueba y de desarrollo
 - No recomendado para ámbito de producción
- **LTS**
 - Compilación Long-Term-Support
 - Concepto de versión muy parecido al LTS de la distribución Linux Ubuntu
 - Lanzadas cada 3 meses
 - Cambia con menos frecuencia y solo recibe correcciones de errores importantes, incluso si dicha línea de lanzamiento se retrasa en términos de características.
 - Este sería el tipo de versión recomendada para su puesta en producción

2.2. Sobre instalaciones en ámbito Windows

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\<username>\.jenkins**
- **Linux → /home/<username>/.jenkins**

Si se desea cambiar, lo único 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 podría cambiar a nivel de la **JVM**

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

2.3. Lab: Instalación de Jenkins (.war)

Mediante este laboratorio, vamos a proceder a llevar a cabo la instalación de Jenkins en modo standalone, instalación simple de un único nodo mediante un .war

2.3.1. Descarga del .war

Desde la URL oficial podemos llevar a cabo su instalación bajo diferentes modalidades, accedemos a la URL: <https://jenkins.io/download/>

El **.war** que nos descargamos, podría ser desplegado en un servidor de aplicaciones que necesitemos.

Indicamos la siguiente URL en el navegador para proceder a su descarga:

```
http://mirrors.jenkins.io/war-stable/latest/jenkins.war
```

En nuestro home de usuario, vamos a crear una carpeta con el nombre **software**:

```
$ mkdir /home/jenkins/software
```

Dejamos dentro el .war descargado

2.3.2. Puesta en marcha

Antes de poner en marcha el propio jenkins, vamos a ajustar la variable de entorno JENKINS_HOME, con un valor de ruta específica para esta instancia.

Ejecutamos en la consola la siguiente instrucción:

```
$ export JENKINS_HOME=/home/jenkins/software/jenkins_home_master/
```

A continuación, vamos a auto - ejecutar Jenkins, ya que lleva embebido un Jetty internamente, ejecutamos el siguiente comando:

```
$ java -jar jenkins.war --httpPort=8081 --webroot=/home/jenkins/software/jenkins_web_root_master/
```

- **--httpPort=8081**

- Indicamos el puerto de operación, mediante este puerto nos conectaremos vía web mediante el navegador... <http://localhost:8081>

- **--webroot=...**

- Indicamos la ruta de contexto, donde esta instancia de jenkins tendrá disponibles archivos referentes a la sesión web, etc.



Por defecto cuando ejecutamos Jenkins, este opera en el puerto 8080, si queremos sobre-escribir dicho comportamiento y ejecutar Jenkins en un puerto diferente, en nuestro caso, en el 8081, indicamos el parámetro **--httpPort**

2.3.3. Arranque inicial

Una vez puesto en marcha el sistema, Jenkins generará una clave de administrador de primer acceso aleatoria, podemos obtener dicha clave de 2 formas:

- Observando la traza de la consola, debe de aparecer algo parecido a esto:

```
...
*****
*****
*****
```

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

97522efcaa524d9488c5636d4fbc2f24

This may also be found at: /home/jenkins/software/jenkins_home_master/secrets/initialAdminPassword

...

- Accediendo al archivo **initialAdminPassword**, que Jenkins de forma automática habrá generado en el home de nuestro usuario

A continuación, accedemos mediante el navegador a la siguiente URL: <http://localhost:8081>

Observaremos que la instalación inicial nos pide un password para poder acceder, introducimos la clave que hemos observado en la traza

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

D:\utilidades\jenkins\secrets\initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

.....



Continue

Una vez creado, nos pedirá la instalación de plugins, (podremos instalar más adelante otros plugins si necesitamos), instalamos los plugins por defecto que Jenkins nos sugiera:



Getting Started

✓ Ant Plugin	✓ OWASP Markup Formatter Plugin	✓ build timeout plugin	✓ Folders Plugin	** Token Macro Plugin Jenkins build timeout plugin Folders Plugin ** Credentials Plugin ** Structs Plugin ** Pipeline: Step API ** Plain Credentials Plugin Credentials Binding Plugin Email Extension Plugin ** SSH Credentials Plugin ** Jenkins Git client plugin ** SCM API Plugin Jenkins Git plugin Jenkins Gradle plugin LDAP Plugin Jenkins Mailer Plugin Matrix Authorization Strategy Plugin PAM Authentication plugin ** JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) plugin ** Durable Task Plugin ** Pipeline: API ** Pipeline: Supporting APIs ** Pipeline: Job ** Pipeline: REST API Plugin ** JavaScript GUI Lib: Handlebars bundle plugin ** JavaScript GUI Lib: Moment.js bundle plugin Pipeline: Stage View Plugin Jenkins SSH Slaves plugin ** MapDB API Plugin Jenkins Subversion Plug-in Timestampper ** Pipeline: Build Step ** JavaScript GUI Lib: ACE Editor bundle plugin ** - required dependency
✓ Credentials Binding Plugin	✓ Email Extension Plugin	✓ Git plugin	✓ Gradle plugin	
✓ LDAP Plugin	✓ Mailer Plugin	✓ Matrix Authorization Strategy Plugin	✓ PAM Authentication plugin	
✓ Pipeline: Stage View Plugin	✓ SSH Slaves plugin	✓ Subversion Plug-in	✓ Timestampper	
⌚ Pipeline	⌚ GitHub Organization Folder Plugin	⌚ Workspace Cleanup Plugin		

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

Getting Started

Create First Admin User

Usuario:

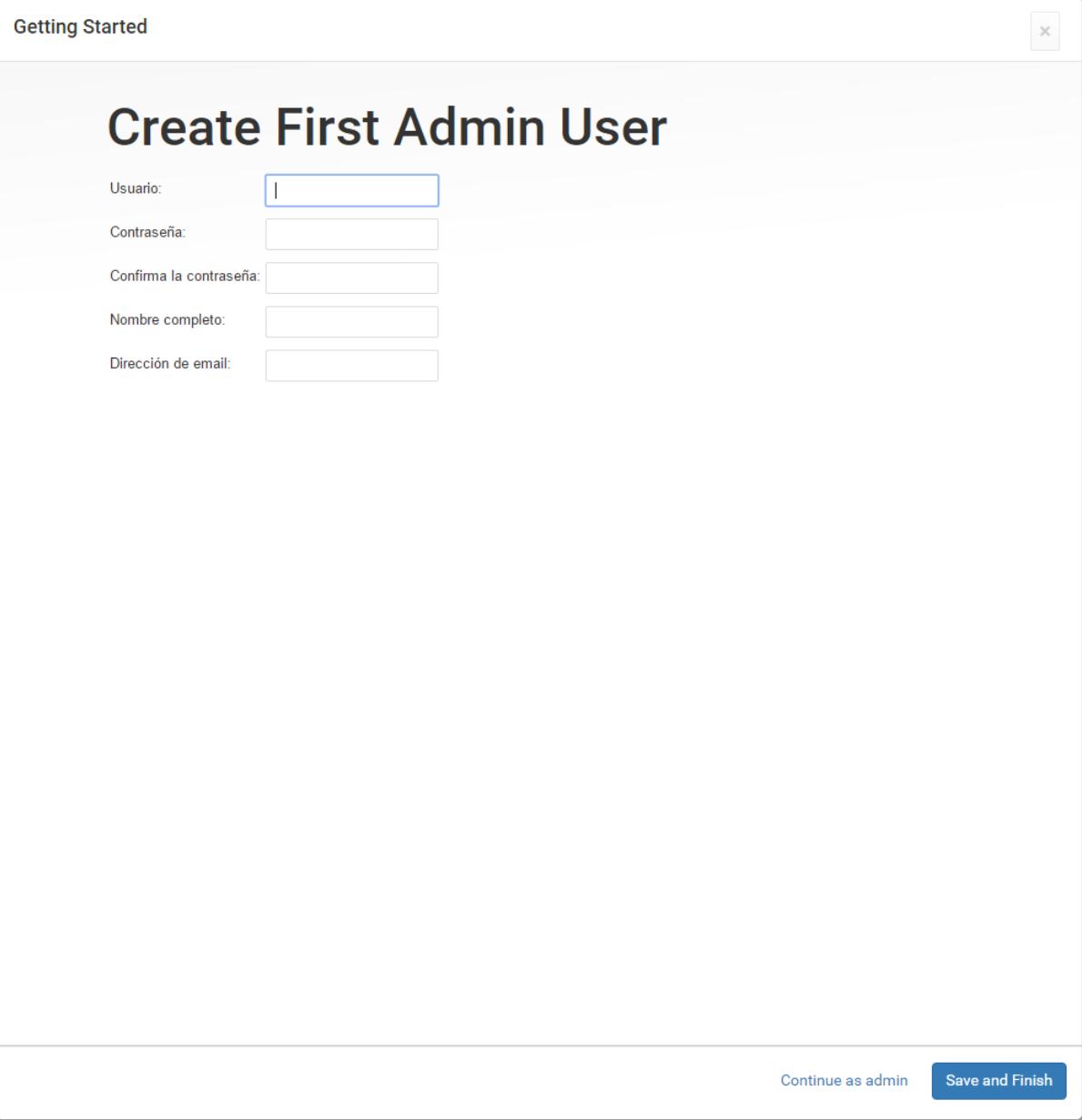
Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

[Continue as admin](#) [Save and Finish](#)



Vamos a ajustar las siguientes credenciales:

- **Username:**
 - admin
- **Password:**
 - admin
- **Confirm password:**
 - admin
- **Full name:**
 - <Tu nombre completo>
- **E-mail address:**
 - <Tu correo electrónico completo>

Chapter 3. Administración de Jenkins

El papel de jenkins se integra en los circuitos de integración continua (CI) y entrega continua (CD), de forma que podemos llevar a cabo automatización de tareas repetitivas relacionadas con las secuencia en nuestros proyectos de software

Además de implementar fases de compilación en integración continua (CI), y fases de puesta en marcha de entornos como pre, sqa, pro, etc. (CD), deberemos de llevar a cabo una administración en diferentes ámbitos:

- Administración de flujos de trabajo (Pipelines)
- Administración de plugins

3.1. Administración de flujos de trabajo (Pipelines)

Jenkins nos puede permitir automatizar los flujos de trabajo mediante el uso de lo que conocemos como **pipelines**, haciendo que nuestro día a día en desarrollo sea más divertido :D

Los flujos de trabajo, podremos regularlos mediante un archivo de secuencia, denominado **Jenkinsfile**, cuya custodia del archivo se encontrará en el mismo repositorio de código fuente, formando parte de este

Así como el mantenimiento del flujo de trabajo, que podremos implementarlo mediante un lenguaje DSL declarativo que provee el propio Jenkins, así como scripting directo en lenguaje Groovy

3.2. Administración de plugins

Adicionalmente al mantenimiento de los diferentes flujos de trabajo que en cada proyecto podremos implementar, también existe una labor de administración/mantenimiento de los plugins que tenga jenkins instalados o que deseemos instalar

La administración de plugins es esencial para poder explotar todo el potencial que Jenkins nos puede otorgar

Cuando llevamos a cabo la instalación básica de Jenkins, quizás este no se ajusta a nuestras necesidades, ya que necesitará de la instalación/configuración de plugins que nos vengan bien en nuestros proyectos :)

Los plugins en Jenkins se utilizan esencialmente para mejorar la funcionalidad que de base trae el propio sistema

Puntos clave a tener en cuenta cuando estudiamos la instalación de un plugin:

- **Siempre debemos de leer la documentación previa del plugin :D**
 - Debemos de evaluar si el plugin se ajusta a los objetivos que perseguimos
- **Estadísticas de uso y frecuencia de actualización**

- Debemos de echar un vistazo a la frecuencia de uso que tiene el plugin y si este es actualizado con frecuencia, para intentar no tener plugins que están obsoletos o que han dejado de mantenerse
- **Preguntarme... ¿El plugin es compatible con la versión de Jenkins que dispongo?**
 - Muchas veces estamos tan ofuscados buscando un plugin que cubra nuestras necesidades, que pasamos por alto este dato, y nos encontramos con problemas de funcionamiento derivados de la no compatibilidad del plugin con nuestra versión de Jenkins

Podremos encontrar información relativa a los plugins en la siguiente URL:
<https://plugins.jenkins.io/>

3.3. Actualizando Jenkins

Jenkins tiene detrás una comunidad bastante activa, debido a esto con bastante frecuencia se lanzan actualizaciones tanto de plugins como del propio sistema Jenkins, tanto con arreglos de errores como de nuevas características

Es muy recomendable mantener siempre el servidor lo más actualizado posible

El proceso de actualización no sólo implica llevar a cabo actualizaciones en el propio host, existen diversas prácticas que cada DevOps y Desarrollador deben de seguir para mantener siempre disponibilidad en el servicio de (CI) y (CD)

3.4. Mantenimiento del sistema

Se trata de un período de tiempo designado previamente por el personal técnico, durante el cual, el mantenimiento del sistema podría acarrear interrupción del servicio

Para intentar que los tiempos de mantenimiento sean lo más cortos posibles, cada programador debe de establecer los tiempos de mantenimiento, que por ejemplo, podría darse 1 vez a la semana

Este tiempo de mantenimiento, podría planificarse para el fin de semana con el fin de interferir lo mínimo posible en la fase de entrega continua (CD)

Posibles efectos de tener la maquinaria de integración continua (CI) y entrega continua (CD) detenida:

- Retrasos en la entrega de funcionalidades del sistema/comprobación de fallos
- Penalizaciones por retrasos en entregas planificadas
- Impactos en servicios o plataformas que ya estuvieran operando, si no es sincronizada su puesta en servicio de forma coordinada

La versión de Jenkins que estemos operando, podremos consultarla desde el dashboard principal de control, abajo a la derecha

Chapter 4. Configuración Básica

Vamos a establecer cierta configuración básica inicial, para acceder a la zona de configuración

4.1. Opciones de menú básicas

Desde el dashboard principal, podemos observar algunas opciones directas en el menú lateral izquierdo

- **New Item**
 - Construcción de un nuevo proyecto en Jenkins basado en pipelines, de estilo libre, etc.
- **People**
 - Muestra todos los usuarios disponibles en Jenkins
- **Build History**
 - Muestra todos los históricos de tareas de construcción que ha llevado a cabo Jenkins a modo de histórico
- **Manage Jenkins**
 - Muestra una lista de todas las configuraciones relacionadas con el servidor de Jenkins
- **My Views**
 - Podemos crearnos vistas personalizadas para mostrar información más acorde a nuestro gusto
- **Credentials**
 - Muestra todas las credenciales por usuario que se encuentran en el servidor Jenkins
- **New View**
 - Permite crear nuevas vistas personalizadas

Si accedemos a **Jenkins > Manage Jenkins**, vamos a poder entre otras cosas acceder a opciones de configuración como:

- Configurar el sistema
- Configurar herramientas
- Instalar Plugins
- Consola de Scripts
- Gestión de usuarios
- Etc.

Por otra parte, desde el panel principal de Jenkins, accedemos a **Jenkins > Manage Jenkins > Configure System**

4.2. Configuración de la JDK

Para poder utilizar proyectos basados en tecnología Java, una operación básica, es indicar la ruta donde se encuentra la JDK, en nuestro caso la opción de configuración se muestra en la sección **Jenkins > Manage Jenkins > Configure System > Global Tool Configuration > JDK**

Indicamos la siguiente configuración:

- **JDK Name:**
 - JDK 1.8
- **JAVA_HOME:**
 - /usr/java/jdk1.8.0_191-amd64

4.3. Configuración de Maven

Ahora, vamos a configurar **Maven**, para que también Jenkins tenga acceso a la herramienta, accedemos a la opción **Jenkins > Manage Jenkins > Global Tool Configuration > Maven**

Indicamos al siguiente configuración:

- **Maven Name:**
 - Maven 3
- **MAVEN_HOME:**
 - /usr/local/apache-maven-3.6.3

4.4. Sistema de control de versiones

Al haber instalado los plugins sugeridos por defecto en la fase de instalación, de forma "nativa" Jenkins soporta los siguientes sistemas de control de versiones:

- SVN
- Git

A continuación, vamos a llevar a cabo cierta configuración para **Git**, de forma que podamos realizar ciertas tareas de una forma cómoda.

Accedemos desde el panel de control principal de Jenkins a **Jenkins > Manage Jenkins > Configure System > Git Plugin**

Indicamos la siguiente configuración:

- **Global Config user.name Value:**
 - Nombre de usuario que deseamos que Jenkins pudiera usar para fijar commits
- **Global Config user.email Value:**
 - Email que deseamos que Jenkins pudiera usar para fijar commits

4.5. Configuración descriptiva básica del servidor

Típicamente quizás también queramos añadir cierta información para el servidor en cuestión, como una descripción sobre el servidor, compañía, etc.

Desde el dashboard principal, hacemos clic en la opción **add description**

Indicamos por ejemplo lo siguiente:

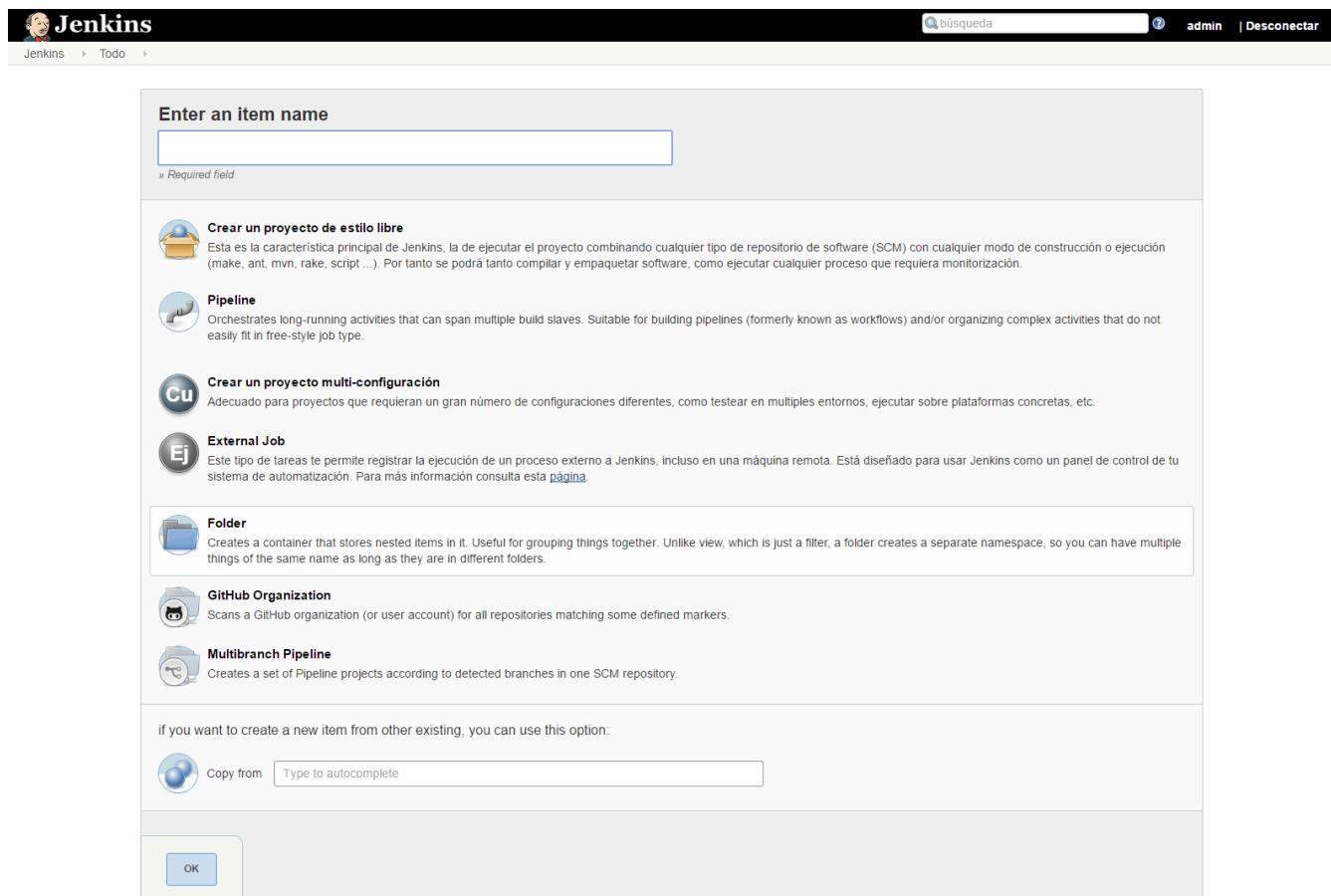
- Company: Curso Jenkins
- Server: Principal
- Department: Ingeniería

Chapter 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**.



The screenshot shows the Jenkins 'Create New Item' dialog box. At the top, there is a field labeled 'Enter an item name' with a note '(Required field)'. Below this, there is a list of job types:

- Create a free-style project**: Descripción: 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**: Descripción: 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.
- Create a multi-configuration project**: Descripción: Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.
- External Job**: Descripción: 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**: Descripción: 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**: Descripción: Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- Multibranch Pipeline**: Descripción: 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

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

Configurar el origen del código fuente

- Ninguno
- Git
- Subversion**

Módulos

Repository URL: <https://Victor-Portatil:8443/svn/EjemploReleasePlugin/trunk>

Credentials: desarrollador/***** (Usuario desarrollador para SVN)

Local module directory: .

Repository depth: infinity

Ignore externals:

Add module...

Additional Credentials: Add additional credentials...

Check-out Strategy: Use 'svn update' as much as possible

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: (Auto)

Avanzado...

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.

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**

Guardar

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

Ejecutar

The screenshot shows the 'Execute' configuration page for a Jenkins job. At the top, it says 'Ejecutar tareas 'maven' de nivel superior'. Below that, 'Version de Maven' is set to 'Maven 3.3.9' and 'Goles' is set to 'install'. A 'Avanzado...' button is visible. A dropdown menu titled 'Añadir un nuevo paso' is open, showing options like 'Ejecutar Ant', 'Ejecutar linea de comandos (shell)', 'Ejecutar tareas 'maven' de nivel superior', etc. The 'Ejecutar Ant' option is currently selected.

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.

This section shows actions to be taken after the build. It includes:

- Publicar los resultados de tests JUnit**:
 - Ficheros XML con los informes de tests: `**/target/surefire-reports/*.xml`
 - Descripción: El atributo '@includes' de la etiqueta 'fileset' especifica dónde están los ficheros XML generados, por ejemplo: 'myproject/target/test-reports/*.xml'. El directorio base para la etiqueta 'fileset' es el directorio raíz del proyecto.
 - Guardar la salida estándar y de error aunque sea muy larga.
- Health report amplification factor**: `1,0`

At the bottom right is a red 'Borrar' button.

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

This section shows artifact archiving settings:

- Guardar los archivos generados**:
 - Ficheros para guardar: `**/target*.jar`

Buttons for 'Avanzado...' and 'Borrar' are at the bottom right.

O la publicación de los Javadoc generados

This section shows JavaDoc publication settings:

- Publicar Javadoc**:
 - Directorio para los javadoc: `target/site/apidocs`
 - Descripción: Directorio relativo al 'workspace' del proyecto, ejemplo: 'myproject/build/javadoc'
 - Conservar los javadoc para todas las ejecuciones correctas

At the bottom right is a red 'Borrar' button.

5.1. Resultado de la ejecución

La ejecución de la tarea, se mostrará con un circulo 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.

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']
```

5.3. Lab: Construcción de un proyecto Java Maven (Pipeline)

Mediante este laboratorio, crearemos un proyecto Java Maven2 que estará compuesto por un único módulo, el módulo será de tipo JAR y se tratará de una aplicación de consola que ejecute un texto por consola al ser invocada

Cubriremos las siguientes fases de lanzamiento:

- Compile
- Test
- Package

5.3.1. Creación de la carpeta para proyectos Java

Nos situamos en el directorio /home/jenkins-ci/projects y creamos una carpeta llamada **java** para nuestros proyectos Java

5.3.2. Creando el proyecto

Vamos a crear una carpeta para el proyecto java que vamos a crear, en nuestro caso, a dicha carpeta la llamaremos **app-backend**

Creamos la carpeta /home/jenkins-ci/projects/java/app-backend/

Vamos a crear nuestro módulo maven para trabajar con él, nos situamos en nuestra carpeta **app-backend**, que acabamos de crear y ejecutamos la siguiente instrucción:

```
$ mvn archetype:generate -DgroupId=com.tutorial.jenkins -DartifactId=jenkins-maven-project
```

- Indicará que elijamos el formato a aplicar, en principio dejamos por defecto la sugerencia que nos indique, ya que será el formato básico de maven
- Indicamos la versión del arquetipo, ahora mismo la más moderna, indicamos **8: 1.4**, debe de aparecer seleccionado por defecto, pulsamos intro
- Indicamos la versión del módulo maven, indicamos **1.0-SNAPSHOT**, debe de aparecer seleccionado por defecto, pulsamos intro
- Nos confirma la información con la que se creará el módulo, groupId, artifactId, etc.

Únicamente pulsamos **Y** para confirmar

Si todo ha ido bien, debería de aparecer en la consola **BUILD SUCCESS**

5.3.3. Inicializando el proyecto con el control de versiones (GIT)

A continuación, estando situados en el directorio **/home/jenkins-ci/projects/java/app-backend/** e inicializamos el repositorio con Git, ejecutando la siguiente instrucción:

```
$ git init
```

```
Initialized empty Git repository in /home/jenkins-ci/projects/java/app-backend/.git/
```

5.3.4. Creando nuestro archivo .gitignore

Una de las primeras tareas al crear nuestro proyecto Java Maven, es indicar los archivos a ignorar en los commits, en nuestro caso indicamos ignorar la carpeta **target** en su conjunto, creamos en el raíz del proyecto un archivo con nombre **.gitignore** y agregamos el siguiente contenido:

```
$ jenkins-maven-project/target
```

5.3.5. Creando nuestro archivo de pipeline Jenkinsfile

Ahora, vamos a llevar a cabo la implementación de nuestro pipeline, de forma que podamos llevar a cabo la ejecución de 3 fases secuenciales (build, test, package):

```
pipeline {
    agent {
        node {
            label 'master'
        }
    }

    options {
        timestamps()
    }

    stages {
        stage('Clean WorkSpace') {
            steps {
                cleanWs()
            }
        }

        stage('Prepare Environment') {
            steps {
                git branch: '$BRANCH_NAME', url: 'https://github.com/tomas-ortega/jenkins-maven-project.git'
            }
        }

        stage('Compile Project') {
```

```

        steps {
            sh 'mvn compile -f jenkins-maven-project/pom.xml'
        }
    }

    stage('Unit Testing') {
        steps {
            sh 'mvn test -f jenkins-maven-project/pom.xml'
        }
    }

    stage('Package') {
        steps {
            sh 'mvn package -f jenkins-maven-project/pom.xml'
        }
    }
}
}

```

Subimos los cambios a nuestro repositorio de código fuente preferido, GitHub, GitLab, etc.

5.3.6. Creando nuestro proyecto en Jenkins

Desde el dashboard principal, vamos a la opción **New Item**, indicamos el tipo **Multibranch Pipeline**, y le indicamos como nombre al job **multibranch-tutorial-pipeline**

Indicamos las siguientes configuraciones a las secciones:

Branch Sources > Add Source > Git

- **Project Repository**
 - Indicamos la URL donde tengamos el repositorio
- **Credentials**
 - Si hemos alojado el proyecto en un repo tipo GitHub, indicamos none
 - Si hemos hecho el repo privado, y queremos que Jenkins se pueda autenticar contra el repo, lo mejor sería optar por una estrategia de clave pública - privada por RSA (SSH)

Build Configuration > Mode

- By Jenkinsfile

Build Configuration > Script Path

- Jenkinsfile
- Aquí podremos elegir el nombre del archivo que hayamos indicado, en principio la comunidad nombra al archivo Jenkinsfile, y no sería mala idea adoptar dicho nombre :)

 La configuración de repositorio en esta parte, es para propósitos de detección de cambios en el repositorio

Si queremos implementar secuencia de compilación con código que nos hemos

descargado, tendremos que especificarlo en la secuencia del pipeline

Chapter 6. Proyectos Multiconfiguración

En ocasiones, nos puede interesar crear un proyecto de tipo multi-configuration

Este tipo de proyecto, resulta útil cuando para un mismo proyecto tenemos diferentes configuraciones, los pasos para llevar a cabo la compilación final, son los mismos, pero únicamente se difiere con ciertas configuraciones

Por ejemplo, queremos llevar a cabo una secuencia de compilación para nuestro proyecto, y deseamos comprobar si la compilación se realiza bien con una JDK 1.8 oficial de oracle y paralelamente con una OpenJDK 1.8

6.1. Lab: Proyecto multiconfiguración

Para este laboratorio, vamos configurar en el nodo de Jenkins, 2 versiones distintas de la JDK, para que estén disponibles en todos los proyectos que requieran de la JDK para llevar a cabo su compilación

La idea, es que podamos crear un proyecto de configuración múltiple, donde simultáneamente la secuencia que queramos implementar, sea ejecutada en paralelo en ambas JDK

6.1.1. Configurando las versiones de las JDK disponibles

Desde el dashboard principal de Jenkins, accedemos a **Manage Jenkins > Global Tool Configuration > JDK**

Vamos a registrar la versión de Oracle de la JDK:

Add JDK

- **Name**
 - JDK_1_8
- **JAVA_HOME**
 - /home/jenkins-ci/software/jdk1.8.0_152

Y a continuación, registramos la versión Open de la JDK:

Add JDK

- **Name**
 - OPEN_JDK_1_8
- **JAVA_HOME**
 - /usr/lib/jvm/java-1.8.0-openjdk

Pulsamos **Save** y listo

6.1.2. Creando nuestro proyecto multi configuration

Desde el dashboard principal de Jenkins, accedemos a la opción **New Item > Multi-configuration project** y le indicamos de nombre **multi-config-jenkins-project**

A continuación, vamos a la sección **Configuration Matrix > Add axis**

Aquí podremos indicar tantas configuraciones paralelas deseemos, cada vez que indiquemos una nueva entrada de Axis, significará una configuración paralela que se deberá de tener en cuenta

En nuestro caso, indicamos **JDK** y seleccionamos los dos checks, **JDK_1_8** y **OPEN_JDK_1_8**

A continuación, accedemos a la sección de **Build > Add build step > Execute shell**

Indicamos este script, que nos indicará por la consola, la versión de la JDK que en cada ejecución se está teniendo en cuenta

```
#!/bin/bash  
  
java -version
```

6.1.3. Comprobando las ejecuciones

Una vez lanzada la ejecución del trabajo, observaremos en el detalle del job, que aparecen las dos configuraciones **JDK_1_8** y **OPEN_JDK_1_8**

Accedemos al detalle de la configuración **JDK_1_8**, accedemos al **Build History > #1 > Console Output**, observamos la siguiente traza

```
java version "1.8.0_152"  
Java(TM) SE Runtime Environment (build 1.8.0_152-b16)  
Java HotSpot(TM) 64-Bit Server VM (build 25.152-b16, mixed mode)
```

Efectivamente, observamos ejecución contra un entorno de Oracle JDK

Ahora, accedemos al detalle de la configuración **OPEN_JDK_1_8**, accedemos al **Build History > #1 > Console Output**, observamos la siguiente traza

```
openjdk version "1.8.0_232"  
OpenJDK Runtime Environment (build 1.8.0_232-b09)  
OpenJDK 64-Bit Server VM (build 25.232-b09, mixed mode)
```

Y en este punto, comprobamos que esta ejecución se ha llevado a cabo contra otra configuración de otra JDK, concretamente, contra la OpenJDK

Chapter 7. Plugins

7.1. 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.

The screenshot shows the Jenkins Global Tool Configuration interface. Under the 'Maven' section, there is a configuration for 'Maven'. It includes fields for 'Nombre' (Name) set to 'Maven 3.3.9', 'MAVEN_HOME' set to 'D:\utilidades\apache-maven-3.3.9', and a checked checkbox for 'Instalar automáticamente' (Install automatically). Below this, there is a dropdown menu labeled 'Instalar desde Apache' (Install from Apache) with 'Versión' (Version) set to '3.3.9'. At the bottom of the Maven section, there are 'Save' and 'Apply' buttons. To the right of the Maven section, there are 'Borrar un instalador' (Delete an installer) and 'Borrar Maven' (Delete Maven) buttons. The top of the page shows a navigation bar with 'Jenkins > Global Tool Configuration'.

i 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

Página generada: 27-abr-2016 21:01:54 CEST REST API Jenkins ver. 2.0

7.2. Plugin Sonarqube

Es un plugin que permite conectar Jenkins con Sonar.

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
Server authentication token	Configuration fields depend on the SonarQube server version.
SonarQube account login	SonarQube authentication token. Mandatory when anonymous access is disabled.
SonarQube account password	SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.
Avanzado...	
Delete SonarQube	
Add SonarQube	

Listado de instalaciones SonarQube

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

SonarQube Scanner

instalaciones de SonarQube Scanner

SonarQube Scanner	Name	Sonarqube scanner
<input checked="" type="checkbox"/> Instalar automáticamente	?	
Install from Maven Central		
Versión SonarQube Scanner 2.5.1		
		Borrar un instalador
Añadir un instalador		Borrar SonarQube Scanner
Añadir SonarQube Scanner		

Listado de instalaciones de SonarQube Scanner en este sistema

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	

7.3. Cobertura Plugin

Plugin que permite visualizar los resultados del análisis estático de código que realiza Cobertura, así como la definición de los límites 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	98	75	75
Methods	100	80	80
Packages	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).

7.4. Copy Artifact plugin

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

Build

Copy artifacts from another project

Project name: gameoflife-default/com.wakaleo.gameoflife\$gameoflife-web 

Which build: Latest successful build  

Stable build only

Artifacts to copy: **/*.war 

Target directory: 

Flatten directories Optional 

Delete

7.4.1. Disk Usage Plugin

Permite monitorizar el uso del disco.

7.5. 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.

7.6. 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.

7.7. Maven Release Plug-in

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

7.8. 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 [aqui](#)

7.9. Plugin Project Template

Permite reutilizar las configuraciones de un proyecto en otro



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

7.10. Plugin Deploy To Container

En este tema vamos a ver como instalar y configurar el plugin de **Deploy to Container** de Jenkins y Tomcat sobre el que vamos a desplegar un proyecto.

Durante el proceso de CI/CD se irán pasando a través de distintas fases (construcción, testing, calidad...) hasta llegar a la última que será la fase de despliegue en la que si nuestra aplicación ha pasado satisfactoriamente por todas las fases anteriores, finalmente se desplegará en un servidor y de esta forma hacerla accesible a los usuarios para que la empiecen a usar.

Este plugin nos permite desplegar nuestras aplicaciones sobre los servidores de aplicaciones Tomcat, JBoss y Glassfish una vez que el job en el que vayamos a usar este plugin se haya ejecutado correctamente.

7.10.1. Lab: Deploy to Container

En este laboratorio, vamos a ver como usar el plugin de Jenkins de **Deploy to Container** para desplegar un proyecto WAR en un servidor Tomcat.

Instalar Plugin Deploy To Container

Empezamos por instalar el plugin, y para ello vamos a la página de **Administrar Jenkins > Gestor de plugins**, buscamos **deploy to container** en la pestaña de **Todos los plugins** y lo instalamos.



Instalación de Tomcat

Vamos a proceder a descargar Tomcat, nos situamos en la carpeta **/home/jenkins/software** y ejecutamos el siguiente comando:

```
$ wget http://apache.uvigo.es/tomcat/tomcat-9/v9.0.34/bin/apache-tomcat-9.0.34.tar.gz
```

Seguidamente, procedemos a descomprimir, ejecutando en consola el siguiente comando:

```
$ tar -xvf apache-tomcat-9.0.34.tar.gz
```

Configuración de Tomcat

Antes de poder usar el plugin, tenemos que realizar algunos cambios en la configuración de Tomcat para que todo funcione a la perfección.

Vamos a empezar por añadir un usuario con unos roles a tomcat. Para ello tenemos que buscar un archivo **conf/tomcat-users.xml** donde vamos a añadir las siguientes etiquetas **XML**.

- **manager-script**: rol que va a permitir hacer el deploy desde Jenkins.
- **manager-gui**: rol que nos permite entrar a la página de Tomcat y ver los proyectos que hay.

/conf/tomcat-users.xml

```
<tomcat-users>
<!-- ... -->
<role rolename="manager-script" />
<role rolename="manager-gui" />
<user username="admin" password="admin" roles="manager-gui, manager-script" />
</tomcat-users>
```

A continuación, vamos a cambiar el puerto por defecto con el que se levanta el servidor Tomcat.

Por defecto es el **8080**, y vamos a cambiarlo al **8085**.

Vamos al archivo **conf/server.xml** y buscamos el siguiente fragmento de configuración, modificamos el valor del elemento **port**:

/conf/server.xml

```
<!-- A "Connector" represents an endpoint by which requests are received
     and responses are returned. Documentation at :
      Java HTTP Connector: /docs/config/http.html
      Java AJP Connector: /docs/config/ajp.html
      APR (HTTP/AJP) Connector: /docs/apr.html
      Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
-->
<Connector port="8085" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```



Después de realizar este tipo de cambios en los archivos de configuración de Tomcat tenemos que reiniciar el servidor

Una vez realizados los cambios, vamos a poder levantar el servidor con los comandos:

```
$ ./bin/startup.sh
$ ./bin/catalina.sh start
```

Y podemos pararlo con los siguientes comandos:

```
$ ./bin/shutdown.sh
$ ./bin/catalina.sh stop
```

Crear API REST para desplegar

Una vez que tenemos Tomcat, vamos a crear una API REST con Spring Boot para desplegar en Tomcat con el plugin.

Empezamos por entrar en la página <https://start.spring.io/> de donde nos vamos a descargar un proyecto inicial ya configurado con las opciones que se muestran a continuación.

The screenshot shows the Spring Initializr interface with the following configuration:

- Project:** Maven Project
- Language:** Java
- Spring Boot:** 2.2.5
- Project Metadata:** Group: com.example, Artifact: EjemploRest
- Options:**
 - Name: EjemploRest
 - Description: Demo project for Spring Boot
 - Package name: com.example.EjemploRest
 - Packaging: War
 - Java version: 8

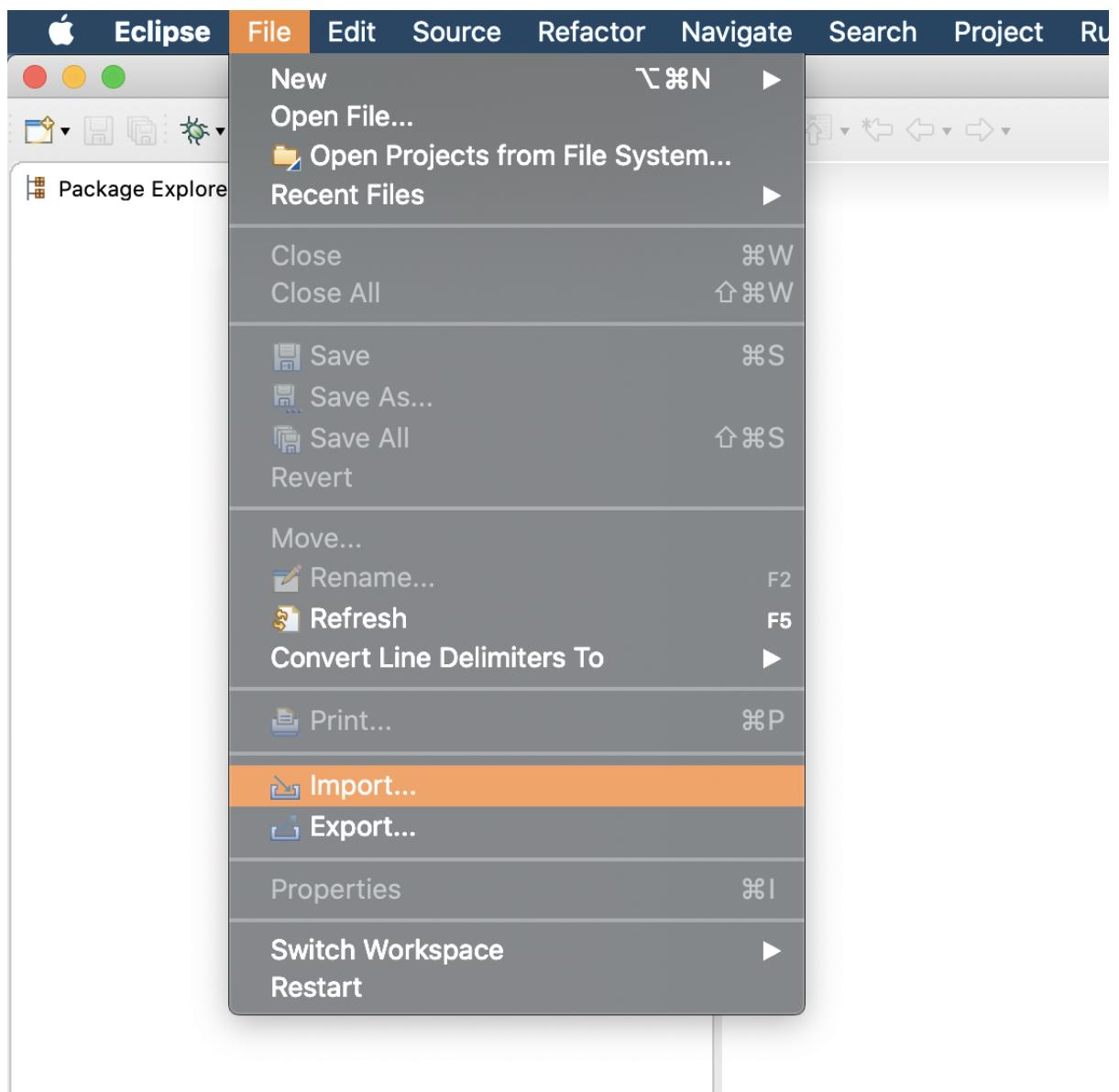
The screenshot shows the Spring Initializr interface with the following selected dependencies:

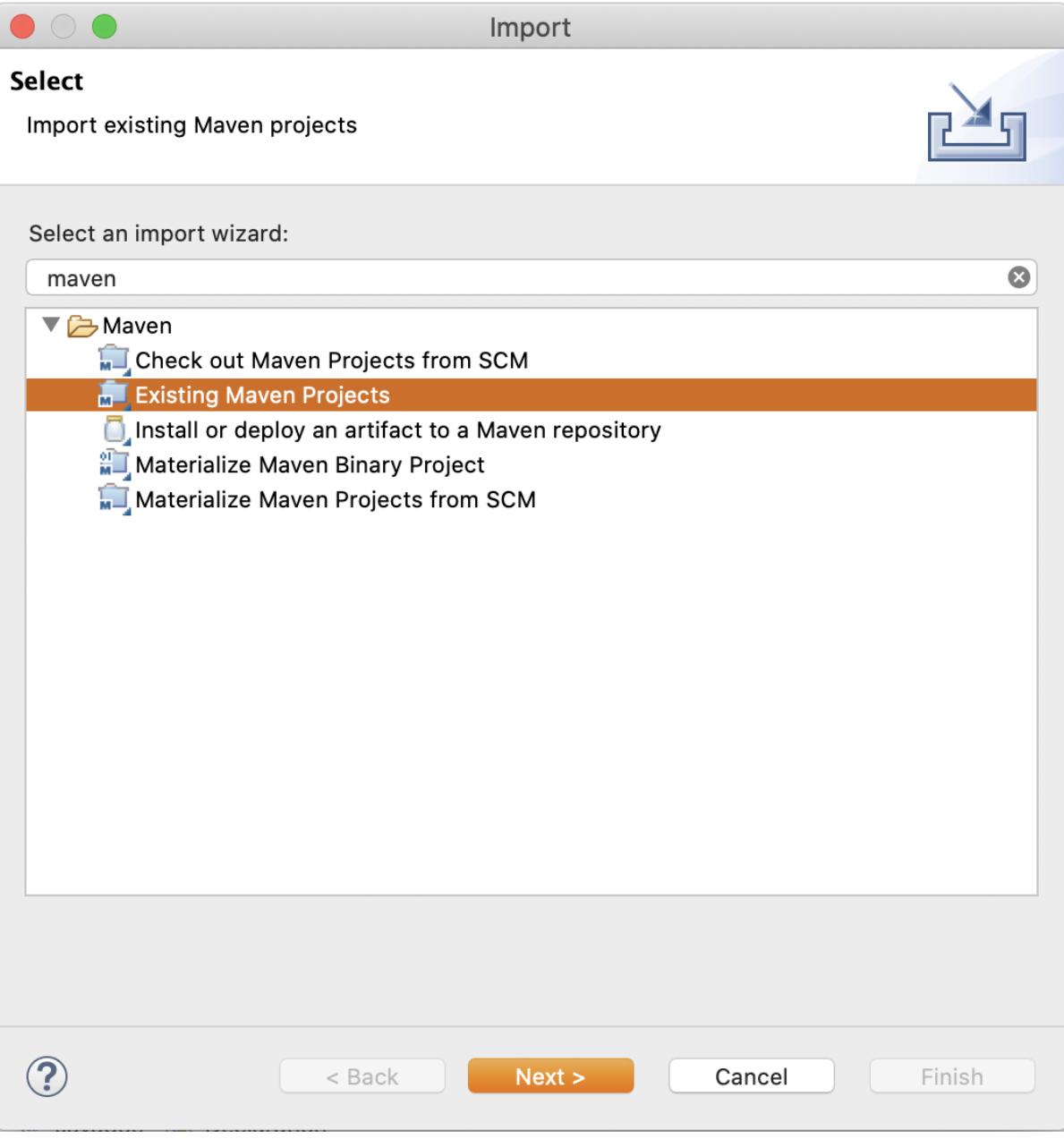
- Spring Web
- Spring Data JPA
- H2 Database

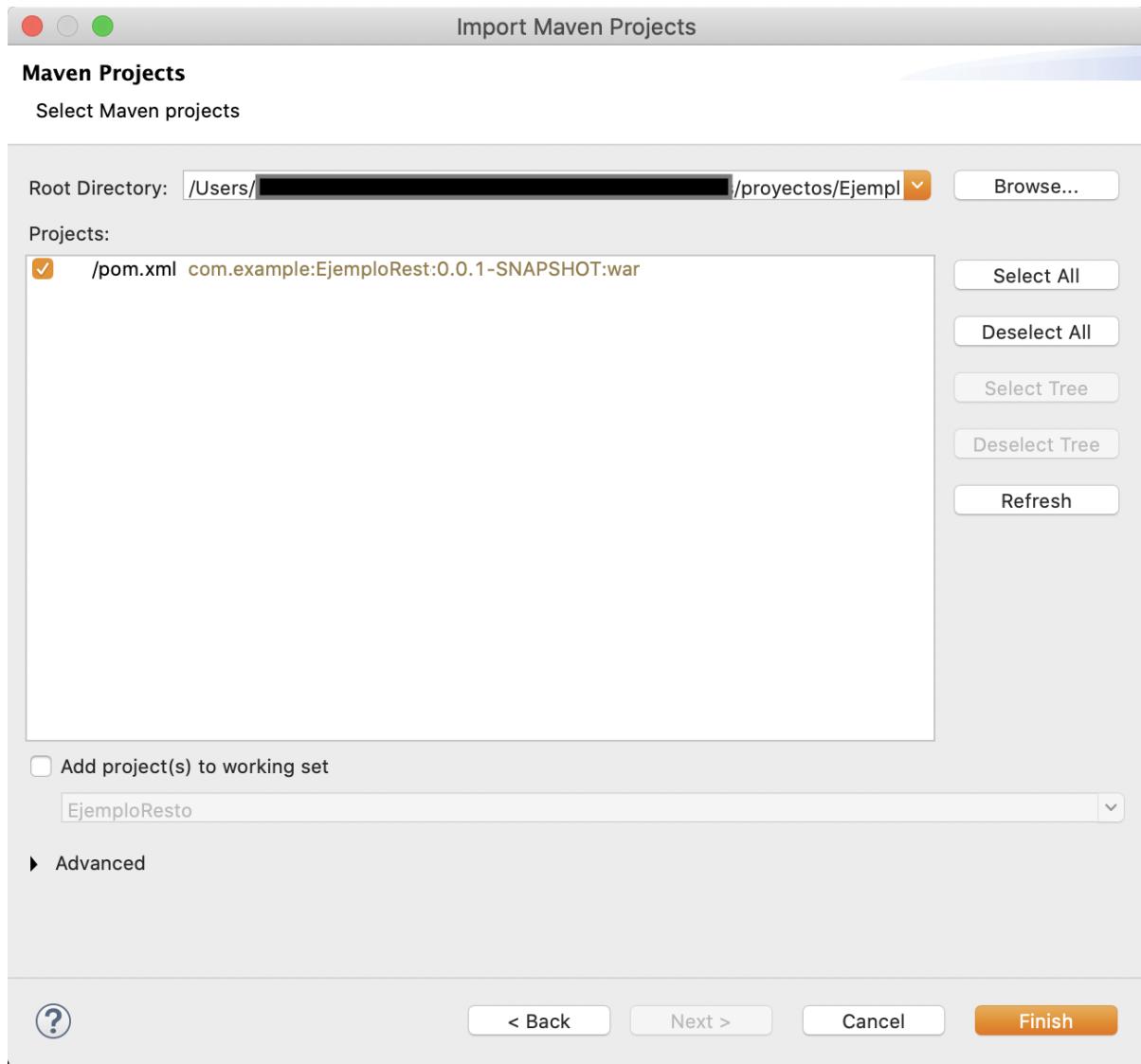
At the bottom, there are buttons for "Generate" and "Explore".

Una vez que hemos seleccionado las opciones mostradas, podemos darle al botón de generar y se descargará un proyecto ya configurado listo para empezar a trabajar en nuestra aplicación.

Abrimos el eclipse e importamos el proyecto siguiendo los pasos que se muestran a continuación:







Una vez importado, vamos a empezar a crear nuestra API REST.

Empezamos creando el modelo **Oferta** para crear los objetos de tipo Oferta con sus propiedades.

/src/main/java/com/example/EjemploRest/entities/Oferta.java

```
package com.example.EjemploRest.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotBlank;

@Entity
@Table(name="ofertas")
public class Oferta {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @NotBlank
    private String titulo;
    private String descripcion;
    private int salario;
    private String ciudad;
```

```

private String empresa;

public Oferta() {
    super();
}

public Oferta(int id, String titulo, String descripcion, int salario, String ciudad, String empresa) {
    super();
    this.id = id;
    this.titulo = titulo;
    this.descripcion = descripcion;
    this.salario = salario;
    this.ciudad = ciudad;
    this.empresa = empresa;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getTitulo() {
    return titulo;
}

public void setTitulo(String titulo) {
    this.titulo = titulo;
}

public String getDescripcion() {
    return descripcion;
}

public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}

public int getSalario() {
    return salario;
}

public void setSalario(int salario) {
    this.salario = salario;
}

public String getCiudad() {
    return ciudad;
}

public void setCiudad(String ciudad) {
    this.ciudad = ciudad;
}

public String getEmpresa() {
    return empresa;
}

public void setEmpresa(String empresa) {
    this.empresa = empresa;
}

@Override
public String toString() {

```

```

        return "Oferta [id=" + id + ", titulo=" + titulo + ", descripcion=" + descripcion + ", salario=" + salario + ",
ciudad=" + ciudad + ", empresa=" + empresa + "]";
    }
}

```

Ahora vamos a crear una interface `OfertaDAO` que extiende de `JPARepository` y que nos proporcionará los métodos más usados a la hora de interactuar con la BBDD y que usaremos en el servicio que vamos a crear después.

`/src/main/java/com/example/EjemploRest/persistence/OfertaDAO.java`

```

package com.example.EjemploRest.persistence;

import org.springframework.data.jpa.repository.JpaRepository;
import com.example.EjemploRest.entities.Oferta;

public interface OfertaDAO extends JpaRepository<Oferta, Integer>{
}

```

Lo siguiente es crear el servicio `OfertasService` que va a hacer de intermediario entre el controlador y la BBDD.

`/src/main/java/com/example/EjemploRest/services/OfertasService.java`

```

package com.example.EjemploRest.services;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.EjemploRest.entities.Oferta;
import com.example.EjemploRest.persistence.OfertaDAO;

@Service
public class OfertasService {

    @Autowired
    private OfertaDAO ofertaDao;

    public Oferta getOferta(int id) {
        return ofertaDao.findById(id).get();
    }

    public List<Oferta> getOfertas() {
        return ofertaDao.findAll();
    }

    public Oferta insertOferta(Oferta oferta) {
        return ofertaDao.save(oferta);
    }
}

```

```

public Oferta updateOferta(Oferta oferta) {
    return ofertaDao.save(oferta);
}

public void deleteOferta(int id) {
    ofertaDao.deleteById(id);
}
}

```

Y por último vamos a crear el controlador **OfertasController** donde recibiremos las peticiones desde el cliente y mandaremos las respuestas a dichas peticiones.

/src/main/java/com/example/EjemploRest/controllers/OfertasController.java

```

package com.example.EjemploRest.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.example.EjemploRest.entities.Oferta;
import com.example.EjemploRest.services.OfertasService;

@CrossOrigin(
    origins="*",
    allowedHeaders="*",
    methods= {
        RequestMethod.GET,
        RequestMethod.POST,
        RequestMethod.PUT,
        RequestMethod.DELETE
    }
)
@RestController
public class OfertasController {

    @Autowired
    private OfertasService ofertasService;

    @GetMapping(path="ofertas", produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<List<Oferta>> getOfertas() {
        List<Oferta> ofertas = ofertasService.getOfertas();
        return new ResponseEntity<List<Oferta>>(ofertas, HttpStatus.OK);
    }

    @GetMapping(path="ofertas/{id}", produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Oferta> getOferta(@PathVariable("id") int id) {
        Oferta oferta = ofertasService.getOferta(id);
        return new ResponseEntity<Oferta>(oferta, HttpStatus.OK);
    }
}

```

```

    @PostMapping(path="ofertas", produces=MediaType.APPLICATION_JSON_VALUE, consumes=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Oferta> createOferta(@RequestBody Oferta ofertaNueva) {
        Oferta oferta = ofertasService.insertOferta(ofertaNueva);
        return new ResponseEntity<Oferta>(oferta, HttpStatus.CREATED);
    }

    @PutMapping(path="ofertas/{id}", produces=MediaType.APPLICATION_JSON_VALUE, consumes=MediaType.
    APPLICATION_JSON_VALUE)
    public ResponseEntity<Oferta> createOferta(@RequestBody Oferta ofertaActualizada, @PathVariable("id") int id) {
        ofertaActualizada.setId(id);
        Oferta oferta = ofertasService.updateOferta(ofertaActualizada);
        return new ResponseEntity<Oferta>(oferta, HttpStatus.OK);
    }

    @DeleteMapping(path="ofertas/{id}", produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Integer> deleteOferta(@PathVariable("id") int id) {
        ofertasService.deleteOferta(id);
        return new ResponseEntity<Integer>(id, HttpStatus.OK);
    }
}

```

Con esto ya tendríamos la API REST creada y podríamos usar Postman para interactuar con ella y ver que funciona correctamente.

Crear Job para desplegar un WAR en Tomcat

Una vez que tenemos la configuración anterior y hemos levantado el servidor, vamos a crear un nuevo job al que le indicaremos el repositorio donde se encuentra en proyecto del que queremos generar un WAR para desplegarlo en el Tomcat.

Desde el menú principal de Jenkins, accedemos a **New Item > Freestyle project**, indicamos como nombre **test-deploy-to-container**

Una vez dentro del Job, vamos a configurar lo siguiente:

Vamos a la sección de **Source Code Management > Git**

- **Repository URL**
 - URL del repositorio donde tengamos alojado el código fuente
- **Credentials**
 - Para nuestro uso didáctico, accederemos al repositorio sin contraseña
 - None
- **Branch Specifier (blank for 'any')**
 - */master
- **Repository browser**
 - (Auto)

Vamos a la sección de **Build > Add build step > Invoke top-level Maven targets**

- **Maven Version**

- Maven 3
 - Seleccionamos la versión de Maven que previamente hemos configurado en Jenkins
- **Goals**
 - clean package

Vamos a la sección de **Post-build Actions** > **Add post-build action** > **Deploy war/ear to a container**

- **WAR/EAR files**

- Indicamos la ruta donde se encuentran los archivos generados
- En nuestro caso indicamos /target/*.war

- **Context path**

- Path sobre el que se tiene que desplegar la aplicación, es decir, que si ponemos /api-rest, luego para entrar a la aplicación entraremos a **mi-app.com/api-rest/datos** por ejemplo
- En nuestro caso indicamos /api

- **Containers**

- Los contenedores donde queremos realizar el despliegue
- En nuestro caso indicamos **Add Container** > **Tomcat 9.x Remote**

- **Credentials**

- Usuario y contraseña con la que acceder al servidor.
- Pulsamos sobre **Add** > **Jenkins** e introducimos las siguientes credenciales

Jenkins Credentials Provider: Jenkins

Add Credentials

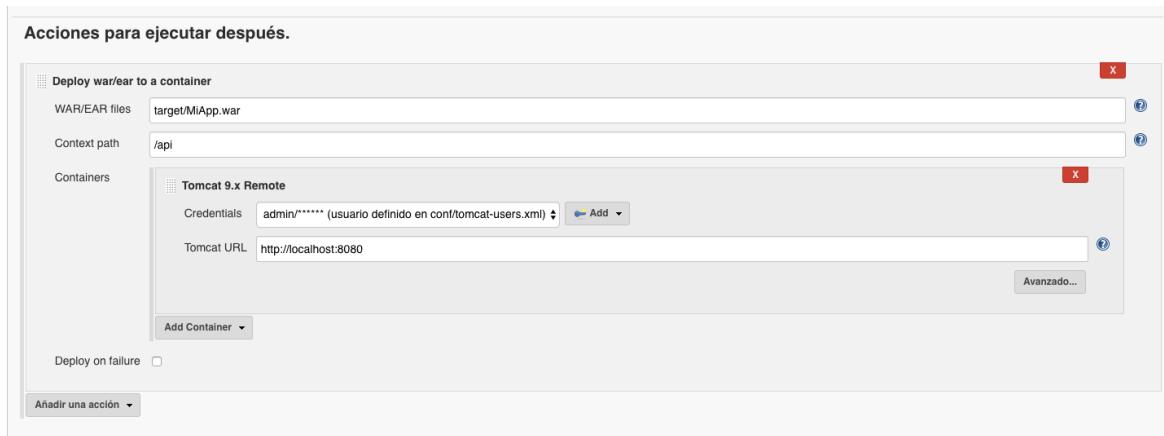
Domain	Global credentials (unrestricted)
Kind	Username with password
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	admin
Password
ID	user-tomcat
Description	usuario definido en conf/tomcat-users.xml

Add **Cancel**

- **Tomcat URL**

- La url donde se encuentra levantado el servidor
- En nuestro caso indicamos <http://localhost:8085>

Vamos a llenar los campos como se muestran en la siguiente imagen.



Una vez finalizada la configuración, guardamos y construimos el job, y si todo va bien deberíamos de poder acceder a la aplicación en <http://localhost:8085/api>



El sistema se ha desplegado con una base de datos en memoria, si accedemos mediante el navegador al endpoint por GET <http://localhost:8085/api/ofertas>, visualizaremos una lista de ofertas de trabajo vacías, para que nos saliera información, deberíamos de invocar primero al método PUT para cargar algunos datos

Despliegue con Jenkins en un contenedor de Docker

En el caso de que tengamos nuestro Jenkins en un contenedor de Docker, tenemos que tener en cuenta que el Tomcat no va a ser accesible a través de <http://localhost:PUERTO> desde Jenkins porque no están en el mismo host y al ejecutar el job nos dará un error como el siguiente:

```
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /var/jenkins_home/workspace/deploy-war-to-tomcat/target/EjemploRest.war to container Tomcat 9.x Remote with
context /api
ERROR: Build step failed with exception
org.codehaus.cargo.container.ContainerException: Failed to redeploy [/var/jenkins_home/workspace/deploy-war-to-tomcat/target/EjemploRest.war]
        at org.codehaus.cargo.container.tomcat.internal.AbstractTomcatManagerDeployer.redeploy(AbstractTomcatManagerDeployer.java:188)
        at hudson.plugins.deploy.CargoContainerAdapter.deploy(CargoContainerAdapter.java:81)
        at hudson.plugins.deploy.CargoContainerAdapter$DeployCallable.invoke(CargoContainerAdapter.java:167)
        at hudson.plugins.deploy.CargoContainerAdapter$DeployCallable.invoke(CargoContainerAdapter.java:136)
        at hudson.FilePath.act(FilePath.java:1075)
        at hudson.FilePath.act(FilePath.java:1058)
        at hudson.plugins.deploy.CargoContainerAdapter.redeployFile(CargoContainerAdapter.java:133)
        at hudson.plugins.deploy.PasswordProtectedAdapterCargo.redeployFile(PasswordProtectedAdapterCargo.java:95)
        at hudson.plugins.deploy.DeployPublisher.perform(DeployPublisher.java:113)
        at hudson.tasks.BuildStepCompatibilityLayer.perform(BuildStepCompatibilityLayer.java:78)
        at hudson.tasks.BuildStepMonitor$3.perform(BuildStepMonitor.java:45)
        at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:741)
        at hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSteps(AbstractBuild.java:690)
        at hudson.model.Build$BuildExecution.post2(Build.java:186)
        at hudson.model.AbstractBuild$AbstractBuildExecution.post(AbstractBuild.java:635)
        at hudson.model.Run.execute(Run.java:1881)
        at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:43)
        at hudson.model.ResourceController.execute(ResourceController.java:97)
        at hudson.model.Executor.run(Executor.java:428)
Caused by: java.net.ConnectException: Connection refused (Connection refused)
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
```

Para solucionarlo, tenemos que buscar la IP de nuestra máquina donde tenemos el Tomcat levantado. Una vez que la sabemos la URL que tendremos que poner será http://MI_IP:PUERTO como

aparece a continuación:



Y por último, Tomcat tiene capado los despliegues remotos, por lo que tendremos que habilitarlo yendo al archivo `/tomcat/webapps/manager/META-INF/context.xml` donde comentaremos la etiqueta `Valve` como se muestra a continuación:

`/tomcat/webapps/manager/META-INF/context.xml`

```
<Context antiResourceLocking="false" privileged="true" >

<!--
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
      allow="127\\.\\d+\\.\\d+\\.\\d+|::1|0:0:0:0:0:0:1" />
-->

<Manager sessionAttributeValueClassNameFilter=
  "java\\.lang\\.\\(?:Boolean|Integer|Long|Number|String)|org\\.apache\\.catalina\\.filters\\.CsrfPreventionFilter\\$LruCache\\(\\?:\\$1
  \\?|java\\.util\\.\\(?:Linked)?HashMap\\)">
</Context>
```

Ahora ya deberíamos de poder ejecutar el job que realiza el despliegue en el Tomcat desde un Jenkins que se encuentra en un contenedor de Docker.

7.11. Ansible

- Ansible permite automatizar la instalación de software en múltiples plataformas
- Para ello, Jenkins cuenta con un plugin de Ansible muy simple que permite tres cosas
 - Gestionar vaults
 - Ejecuciones Ad-hoc (comando ansible)
 - Ejecuciones de Playbooks

Updates	Available	Installed	Advanced	
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	Ansible plugin Invoke Ansible Ad-Hoc commands and playbooks.	1.0		Uninstall
<input checked="" type="checkbox"/>	Credentials Plugin This plugin allows you to store credentials in Jenkins.	2.3.0		Uninstall
<input checked="" type="checkbox"/>	Plain Credentials Plugin Allows use of plain strings and files as credentials.	1.5		Uninstall
<input checked="" type="checkbox"/>	SSH Credentials Plugin Allows storage of SSH credentials in Jenkins	1.18		Uninstall

7.11.1. Builds

- En la sección de build de cualquier job aparecerán las tres opciones

The screenshot shows the Jenkins 'Build Environment' configuration screen. In the 'Build' section, a dropdown menu titled 'Add build step' is open, showing various options like 'Conditional step (single)', 'Invoke Ansible Ad-Hoc Command', 'Invoke Ansible Playbook', and 'Invoke Ansible Vault'. These last three options are specifically highlighted with a red box.

7.11.2. Ad-Hoc

- Los comandos Ad-hoc permiten ejecutar módulos individuales que permiten realizar una sola tarea puntual

Build

Invoke Ansible Ad-Hoc Command

Ansible installation	Ansible
Host pattern	all
Inventory	<input checked="" type="radio"/> Do not specify Inventory <input type="radio"/> File or host list <input type="radio"/> Inline content
Module	ping
Module arguments or command to execute	
Credentials	- none - <button>Add</button>
Vault Credentials	- none - <button>Add</button>
become	
sudo	



- Debido a un bug existente en Jenkins, no es posible ejecutar este componente en un pipeline y no es accesible directamente

- Permite también elegir los hosts afectados, el inventario, que módulo se va a ejecutar
- Permite definir credenciales de acceso, seleccionar credenciales de vault, cambiar de usuario con become, hacer un sudo para ejecutar el comando e incluso cambiar el número de procesos en paralelo que ejecutarán el módulo

7.11.3. Playbook

- Permite la ejecución de un playbook, que es un conjunto de módulos ansible orquestados

Build

Invoke Ansible Playbook

Ansible installation: Ansible

Playbook path: mi-playbook.yml

Inventory:

- Do not specify Inventory
- File or host list
 - File path or comma separated host list: inv/development
- Inline content

Host subset: all

Credentials: - none - [Add](#)

Vault Credentials: - none - [Add](#)

become
 sudo

[Advanced...](#)

- Permite también elegir los hosts afectados, el inventario, que módulo se va a ejecutar
- Permite definir credenciales de acceso, seleccionar credenciales de vault, cambiar de usuario con become, hacer un sudo para ejecutar el comando e incluso cambiar el número de procesos en paralelo que ejecutarán el módulo

7.11.4. Vault

- Permite la gestión de claves para Ansible

Build

Invoke Ansible Playbook

Ansible installation: Ansible

Playbook path: mi-playbook.yml

Inventory:

- Do not specify Inventory
- File or host list
 - File path or comma separated host list: inv/development
- Inline content

Host subset: all

Credentials: - none - [Add](#)

Vault Credentials: - none - [Add](#)

become
 sudo

[Advanced...](#)

- Permite encriptar o desencriptar con sencillez como una fase independiente

- Usa credenciales de Jenkins de tipo user/password o de tipo textfile exclusivamente

7.11.5. Lab: Ejecución de un playbook en Jenkins

- Para ello, vamos a crear un repositorio local
- Creamos una carpeta nueva en nuestro directorio actual

```
[ansible@ansible-master ~]$ sudo mkdir ejemploPlaybook
[ansible@ansible-master ~]$ cd ejemploPlaybook/
```

- Inicializamos un nuevo repositorio git
- El objetivo es crear aqui nuestros playbooks y lanzarlos con Jenkins después de hacer un commmit

```
[ansible@ansible-master ejemploPlaybook]$ git init
Initialized empty Git repository in /home/ansible/ejemploPlaybook/.git/
```

- Cramos un fichero readme y realizamos nuestro primer commit

```
[ansible@ansible-master ejemploPlaybook]$ echo "# Repositorio de Playboooks" > README.md
[ansible@ansible-master ejemploPlaybook]$ git add .
[ansible@ansible-master ejemploPlaybook]$ git commit -m "First commit"
[master (root-commit) 055e283] First commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

- 
- En caso de que el usuario de Jenkins no tenga acceso a la ruta local, lo movemos a un directorio con acceso.
 - En nuestro caso, el dueño es un usuario llamado Ansible, movemos el directorio a un path publico accesible y agregamos como grupo dueño el grupo de jenkins

```
[ansible@ansible-master ~]$ sudo mv ejemploPlaybook/ /ejemploPlaybook
[ansible@ansible-master ~]$ sudo chown -R ansible.jenkins /ejemploPlaybook/
```

- Ahora vamos a crear nuestro playbook base
- Creamos un fichero playbook.yml con el siguiente contenido

playbook.yml

```
---
- name: Ejecutar debug
  hosts: ansible_masters
  become: yes
```

```

vars:
  clave: valor
tasks:
  - name: Instalar httpd
    yum:
      name: httpd
      state: installed
  - debug:
      msg: "Uso de variables clave = {{ clave }}"

```

- En este caso, estamos instalando el servicio httpd en la maquina local donde se ejecute, y mostrar un mensaje por consola con el contenido de una variable
- Creamos ahora un directorio llamado inventarios
- Creamos un fichero llamado hosts.dev con el siguiente contenido

hosts.dev

```

[ansible_masters]
10.0.2.15

```

- Este es el inventario que usaremos con la máquina local
- Ahora agregamos el contenido a un nuevo commit

```

[ansible@ansible-master ejemploPlaybook]$ git add .
[ansible@ansible-master ejemploPlaybook]$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   inventarios/hosts.dev
    new file:   playbook.yml

[ansible@ansible-master ejemploPlaybook]$ git commit -m "Agregado inventario"
[master 2b55498] Agregado inventario
 2 files changed, 15 insertions(+)
 create mode 100644 inventarios/hosts.dev
 create mode 100644 playbook.yml

```

7.11.6. Ejecución de comando Ad-hoc

- Ahora vamos a crear una nueva tarea de Jenkins llamada Masters
- Será de tipo Freestyle

Enter an item name

Masters

» Required field



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Bitbucket Team/Project

Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.



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.

- Como **Source Code Management** usamos la ruta al directorio absoluto agregando file://

- En nuestro caso será [file:///ejemploPlaybook](#)

Source Code Management

None

Git

Repositories

Repository URL

Credentials

- none -



Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any')



Add Branch

Repository browser

(Auto)



Additional Behaviours

- Como build seleccionaremos un step llamado **Invoke Ansible Ad-Hoc Command**

- Host pattern: ansible_masters
 - Inventory: inventarios/hosts.dev
 - module: ping

Invoke Ansible Ad-Hoc Command

Ansible installation	Ansible
Host pattern	ansible_masters
Inventory	<input type="radio"/> Do not specify Inventory <input checked="" type="radio"/> File or host list File path or comma separated host list: inventarios/hosts.dev
Module	ping
Module arguments or command to execute	
Credentials	- none -
Vault Credentials	- none -
<input type="checkbox"/> become	
<input type="checkbox"/> sudo	
Advanced...	



- El inventario debe tener las credenciales de las máquinas a las que se van a acceder
- Si no debemos indicar manualmente las credenciales de acceso
- También permite ignorar la comprobación de fingerprint en conexiones ssh

- Pulsamos en avanzado

- Deshabilitamos el key check
- Habilitamos los colores

<input type="checkbox"/> become	
<input type="checkbox"/> sudo	
Number of parallel processes	5
Disable the host SSH key check	<input checked="" type="checkbox"/>
Unbuffered stdout	<input checked="" type="checkbox"/>
Colorized stdout	<input checked="" type="checkbox"/>
Extra Variables	Add Extra Variable
Additional parameters	

- Guardamos y ejecutamos la tarea pulsando en **Build now**

The screenshot shows the Jenkins interface with the 'Console Output' tab selected. The output window displays the command-line logs of an Ansible playbook run. The logs show the playbook being started by user 'admin' and running as 'SYSTEM'. It details the fetch of changes from a local repository and an upstream source, the configuration of remote origins, and the execution of a playbook named 'file:///ejemploPlaybook'. The output concludes with a successful ping to a host and the message 'Finished: SUCCESS'.

```

Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Masters
No credentials specified
> /usr/bin/git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> /usr/bin/git config remote.origin.url file:///ejemploPlaybook # timeout=10
Fetching upstream changes from file:///ejemploPlaybook
> /usr/bin/git -version # timeout=10
> /usr/bin/git fetch --tags --progress - file:///ejemploPlaybook +refs/heads/*:refs/remotes/origin/*
> /usr/bin/git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> /usr/bin/git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 2b5549843d17d093911d500dc98a972059c46c4 (refs/remotes/origin/master)
> /usr/bin/git config core.sparsecheckout # timeout=10
> /usr/bin/git checkout -f 2b5549843d17d093911d500dc98a972059c46c4 # timeout=10
Commit message: "Agregado inventario"
> /usr/bin/git rev-list --no-walk 2b5549843d17d093911d500dc98a972059c46c4 # timeout=10
[Masters] $ /usr/local/bin/ansible ansible_masters -i inventarios/hosts.dev -m ping -f 5
10.0.2.15 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
Finished: SUCCESS

```

- En el caso de que no aparezca en color las entradas de Ansible, se puede activar instalando el plugin **ANSI Color**
- En el proyecto, en Build Environment hay que seleccionar el plugin de ANSI Color.
- Ahora aparecerán correctamente los colores

The screenshot shows the 'Build Environment' configuration section for a Jenkins job. It includes several checkboxes for build-related options and a specific checkbox for 'Color ANSI Console Output' which is checked. Below this, there is a dropdown menu set to 'xterm' under the heading 'ANSI color map'. There are also other unselected checkboxes for generating release notes, inspecting build logs for Gradle scans, and using Ant.

7.11.7. Uso de Playbook

- Ahora vamos a agregar un step con un playbook en ejecución que tendrá que definir la directiva become para instalar software como root
- Para ello editamos el proyecto y agregamos un nuevo step de tipo **Invoke Ansible Playbook**
 - Host subset: ansible_masters
 - Inventory (File or hosts list): inventarios/hosts.dev
 - become: yes
 - Playbook path: playbook.yml

Invoke Ansible Playbook

Ansible installation	Ansible
Playbook path	playbook.yml
Inventory	<input type="radio"/> Do not specify Inventory <input checked="" type="radio"/> File or host list File path or comma separated host list inventarios/hosts.dev
Host subset	ansible_masters
Credentials	- none - Add
Vault Credentials	- none - Add
<input checked="" type="checkbox"/> become	
become user	
<input type="checkbox"/> sudo	
Advanced...	

- Seleccionamos colorized en avanzado
- Seleccionamos disable hosts ssh key check

<input type="checkbox"/> sudo	
Tags to run	
Tags to skip	
Task to start at	
Number of parallel processes	5
Disable the host SSH key check	<input checked="" type="checkbox"/>
Unbuffered stdout	<input checked="" type="checkbox"/>
Colorized stdout	<input checked="" type="checkbox"/>
Extra Variables	Add Extra Variable
Additional parameters	

- Construimos el proyecto

Jenkins > Masters > #4

Build Information

Delete build #4'

Git Build Data

No Tags

Open Blue Ocean

Previous Build

```

Fetching changes from the remote Git repository
> /usr/bin/git config remote.origin.url file:///ejemploPlaybook # timeout=10
Fetching upstream changes from file:///ejemploPlaybook
> /usr/bin/git --version # timeout=10
> /usr/bin/git fetch -t --tags --progress -- file:///ejemploPlaybook +refs/heads/*:refs/remotes/origin/* # timeout=10
> /usr/bin/git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> /usr/bin/git config core.sparsecheckout # timeout=10
> /usr/bin/git checkout -f 2b5549843d717d093911d500dc98a972059c46c4 (refs/remotes/origin/master)
Checking out Revision 2b5549843d717d093911d500dc98a972059c46c4 (refs/remotes/origin/master)
> /usr/bin/git checkout -f 2b5549843d717d093911d500dc98a972059c46c4 # timeout=10
Commit message: "Agregado inventario"
> /usr/bin/git rev-list --no-walk 2b5549843d717d093911d500dc98a972059c46c4 # timeout=10
[Masters] $ /usr/local/bin/ansible ansible_masters -i inventarios/hosts.dev -m ping -f 5
10.0.2.15 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
[Masters] $ /usr/local/bin/ansible-playbook playbook.yml -i inventarios/hosts.dev -l ansible_masters -f 5
PLAY [Ejecutar debug] ****
TASK [Gathering Facts] ****
ok: [10.0.2.15]
TASK [Instalar httpd] ****
ok: [10.0.2.15]
TASK [debug] ****
ok: [10.0.2.15] => {
    "msg": "Uso de variables clave = valor"
}
PLAY RECAP ****
10.0.2.15 : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
Finished: SUCCESS

```

7.11.8. Vault

- Vamos a pensar que la variable que utilizamos es una variable que no debería estar en un repositorio en plano
- Para ello vamos a encriptar la variable con el vault de Ansible
- Abrimos una shell y encriptamos la variable clave con contenido valor_secreto

```
[ansible@ansible-master ejemploPlaybook]$ ansible-vault encrypt_string 'valor-secreto' --name 'clave'
New Vault password:
Confirm New Vault password:
clave: !vault |
$ANSIBLE_VAULT;1.1;AES256
3132663533065333835623730333133623837386665316135313030663630633065316530363763
6639356363653032646432643031643738393133323334340a393562363163323631353637336438
3833643038363734643131623337383731386532633233331656135393231656137393464316464
333263646333862360a3435323837313361633162336633333833313139656633313030356334
6637
Encryption successful
```

- Como contraseña usaremos ansible
- Modificamos el playbook y cambiamos la entrada clave por la salida del vault

playbook.yml

```
---
- name: Ejecutar debug
  hosts: ansible_masters
  become: yes
  vars:
    clave: !vault |
$ANSIBLE_VAULT;1.1;AES256
```

```

31326635333065333835623730333133623837386665316135313030663630633065316530363763
6639356363653032646432643031643738393133323334340a393562363163323631353637336438
3833643038363734643131623337383731386532633233331656135393231656137393464316464
3332636463333862360a34353238373133616331623366333333833313139656633313030356334
6637

```

tasks:

- name: Instalar httpd

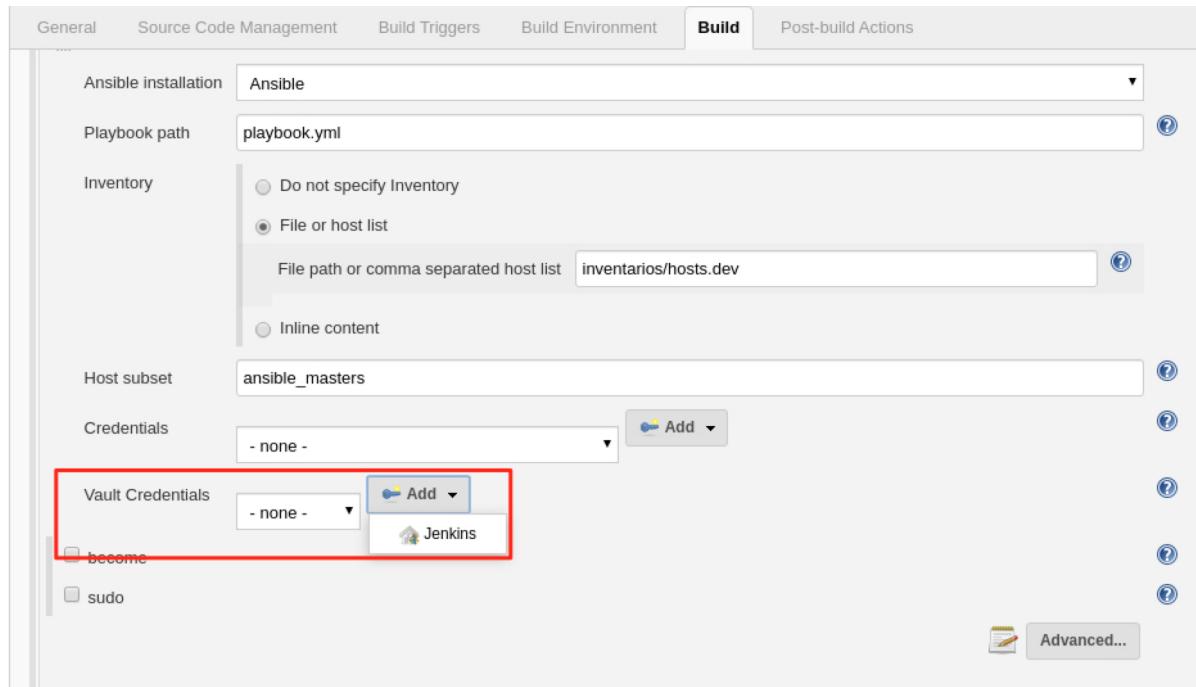

```
yum:
    name: httpd
    state: installed
```
- debug:


```
msg: "Uso de variables clave = {{ clave }}"
```

- Volvemos a generar el commit

```
[ansible@ansible-master ejemploPlaybook]$ git commit -am "Agregado secreto"
[master 7dcce71] Agregado secreto
 1 file changed, 7 insertions(+), 1 deletion(-)
```

- Ahora vamos a modificar la tarea de playbook
- Vamos a agregar una credencial de Vault



- Usamos un **Secret text** ya que solo son compatibles secret text y secret file
- Como secret usamos la clave que utilizamos, en nuestro caso ansible
- Como id y description usamos clave-vault
- Guardamos

The screenshot shows the Jenkins 'Add Credentials' dialog. It has fields for Domain (set to 'Global credentials (unrestricted)'), Kind ('Secret text'), Scope ('Global (Jenkins, nodes, items, all child items, etc)'), Secret ('.....'), ID ('clave-vault'), and Description ('clave-vault'). At the bottom are 'Add' and 'Cancel' buttons.

- Seleccionamos nuestra nueva credencial
- Guardamos y volvemos a lanzarlo

The screenshot shows the Jenkins build configuration screen under the 'Build' tab. It includes sections for 'Invoke Ansible Playbook' (Ansible installation set to 'Ansible', Playbook path 'playbook.yml', Inventory set to 'File or host list' with path 'inventarios/hosts.dev'), 'Host subset' ('ansible_masters'), 'Credentials' (empty dropdown), 'Vault Credentials' ('clave-vault' selected), and checkboxes for 'become' and 'become user'.

- Ahora comprobamos como el valor secreto se muestra en el log

7.12. Pipelines

- Se trata de un conjunto de plugins que permiten introducir pipelines de integración continua en jenkins
- Existe un gran número de herramientas de diseño de pipelines

7.12.1. Prerequisitos

- Es necesario al menos Jenkins 2.x
- Instalación de los plugins de pipeline, que vienen por defecto en los plugins sugeridos en la instalación de Jenkins

7.12.2. Elementos

- Los jobs de jenkins no son necesarios por defecto
- Ahora se permite el uso de Groovy para generar el script de pipelining

Jenkinsfile

- Fichero declarado, almacenado y versionado en el código del proyecto
- En ella se definen las fases del mismo
- Se puede agregar código groovy para realizar las operaciones y condiciones que deseemos

Stages

- Son las fases del pipeline
- Intenta imitar a los jobs concretos
- Permite definir compilación, clonado, test o ejecución

7.12.3. Pipeline declarativo

- Permite un lenguaje de dominio más sencillo que scripting de Groovy
- Todas comienzan con un bloque de tipo pipeline

```
pipeline {  
    /* Código de pipeline */  
}
```

- Existe un generador de directivas declarativas
 - <http://url-jenkins/directive-generator>
- A continuación describiremos las secciones más comunes

Agent

- Especifica donde ejecutar el pipeline completo o la stage concreta
- Permite elegir la instancia para cada sección o en general
- Es obligatoria su definición
- Opciones
 - any: En cualquier agente disponible

```

pipeline {
    agent any
    /* Código de pipeline */
}

```

- none: No hay agente global, eso indica que debemos definir los agentes localmente en cada stage

```

pipeline {
    agent none
    /* Código de pipeline */
}

```

- label: En cualquier agente que posea la etiqueta que indiquemos

```

pipeline {
    agent {
        label 'mi-etiqueta'
    }
    /* Código de pipeline */
}

```

- node: Similar a label pero permite definir opciones adicionales

```

pipeline {
    agent {
        node {
            label 'mi-etiqueta'
            /* Opciones avanzadas de nodo */
        }
    }
    /* Código de pipeline */
}

```

- docker: Ejecución del pipeline o stage en un contenedor provisionado dinámicamente

```

pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            label 'testing-jenkins-slave'
            args '-v /var/lib/jenkins/workspace:/mnt -w /mnt'
        }
    }
    /* Código de pipeline */
}

```

```
}
```

- Existen otras como kubernetes o Dockerfile que permiten crear nodos nuevos sin imagen o crearlas en un cluster de kubernetes

Postcondiciones

- Existen una serie de condiciones a ejecutar cuando el agente termina
 - always: Ejecución de fase pase lo que pase
 - changed: Si el pipeline ha cambiado o tiene un status distinto de ejecución que la ejecución anterior
 - fixed: Solo si fallo el anterior o es inestable
 - regression: Si falla el actual y el anterior fue correcto
 - aborted: si tiene status aborted (normalmente al pararlo manualmente via web)
 - failure: Si tiene status failed, se ejecuta
 - success: Si tiene status success, se ejecuta
 - unstable: Si las pruebas ejecutadas fallan, pasa a unstable y se ejecuta la tarea
 - unsuccessful: Si no posee status success (es decir cualquier otro) se ejecuta
 - cleanup: Permite ejecutar en cada condicion de post, esta tarea de cleanup

```
pipeline {  
    agent any  
    stages {  
        stage('Primera fase') {  
            steps {  
                echo 'Ejecutando primera fase'  
            }  
        }  
    }  
    post {  
        always {  
            echo 'Al final siempre ejecuto esta pase lo que pase'  
        }  
    }  
}
```

Stages

- Secuencia de una o mas directivas stage
- Es requerido

```
pipeline {  
    agent any
```

```

stages {
    stage('Fase uno') {
        steps {
            echo 'Primer step ejecutado'
        }
    }
}

```

Stage

- Permite tener una serie de steps asociados
- La idea es que haya un conjunto de steps asociados
- Puede tener un agent asociado

Steps

- Permite definir los pasos de un stage.
- Puede haber mas de un step en un stage
- Intenta definir de forma lógica los pasos
- Existe una gran cantidad de steps asociados a plugins
- Cada plugin agrega sus steps
- No todos los steps se pueden ejecutar en formato declarativo
- Una lista completa de las opciones es:
 - <https://jenkins.io/doc/pipeline/steps/>

Lock

En lugar de intentar limitar el número de compilaciones simultáneas de un trabajo usando un stage, ahora podemos utilizar el concepto de "Recursos bloqueables" con la instrucción **lock**.

Cuando utilizamos la cláusula **lock**, conseguimos limitar la concurrencia a una sola compilación como máximo en paralelo, proporcionando una flexibilidad mucho mayor en la designación de la concurrencia limitada.

```

stage('Build') {
    doSomething()
    lock('myResource') {
        echo "locked build"
    }
}

```

```

lock('myResource') {
    stage('Build') {

```

```

    echo "Building"
}
stage('Test') {
    echo "Testing"
}
}

```

Milestone

La instrucción **milestone** podemos decir que es la última pieza del rompecabezas para reemplazar la funcionalidad originalmente pensada para el escenario y agrega aún más control para manejar las construcciones concurrentes de un trabajo.

La instrucción lock limita el número de compilaciones que se ejecutan simultáneamente en una sección del pipeline, mientras que el bloque milestone, garantiza que las compilaciones más antiguas de un job no sobrescriban una compilación posterior.

Las compilaciones simultáneas del mismo trabajo no siempre se ejecutan a la misma velocidad.

Dependiendo de la red, el nodo utilizado, los tiempos de compilación, los tiempos de prueba, etc., siempre es posible que una compilación más moderna se complete más rápido que una compilación anterior.

Por ejemplo, suponemos que tenemos 1 Job, que recibe de forma simultánea 3 peticiones de construcción:

- Build 1 is triggered
- Build 2 is triggered
- El lanzamiento de la construcción 2 puede que transcurra más rápido que la ejecución del primer lanzamiento, esto puede provocar que la ejecución del Build2 sobre-escriba o altere datos que el Build1 espera tener consistentemente en otro estado.

En lugar de permitir que la Build 1 continúe y posiblemente sobrescriba el artefacto más moderno que se ha producido en el Build 2, se puede usar el **milestone** para cancelar Build1

```

stage('Build') {
    milestone()
    echo "Building"
}
stage('Test') {
    milestone()
    echo "Testing"
}

```

Cuando se utiliza alguna sentencia que produce un bloqueo, como un mensaje de entrada, se pueden acumular fácilmente compilaciones simultáneas, ya sea esperando la entrada del usuario o esperando que un recurso se libere.

El milestone eliminará automáticamente todos los trabajos más antiguos que están esperando en estos cruces.

```
milestone()  
input message: "Proceed?"  
milestone()
```



Podemos encontrar más información al respecto sobre stage, lock y milestone en la web <https://jenkins.io/blog/2016/10/16/stage-lock-milestone/>

Environment

- Permite definir claves y valores para todos los steps
- Existe un método específico llamado credentials() que permite acceder alas credenciales de Jenkins
 - Permite acceder a credenciales de tipo:
 - Secret text
 - Secret file
 - Username y Password
 - SSH con private key

```
pipeline {  
    agent any  
    environment {  
        CLAVE = 'valor'  
    }  
    stages {  
        stage('Fase uno') {  
            environment {  
                CLAVE_FADE_UNO = 'Valor de fase uno'  
            }  
            steps {  
                echo 'Primer step ejecutado. Mostrando variables'  
                sh 'printenv'  
            }  
        }  
    }  
}
```

- Si queremos usar las credenciales

```
pipeline {  
    agent any  
    environment {
```

```

        CLAVE = 'valor'
    }
    stages {
        stage('Fase uno') {
            environment {
                CLAVE_FASE_UNO = 'Valor de fase uno'
                CREDENCIALES_FASE_UNO = credentials('user-pass-jenkins')
            }
            steps {
                echo 'Primer step ejecutado. Mostrando variables'
                sh 'printenv'
                sh 'echo "Usuario : $CREDENCIALES_FASE_UNO_USR"'
                sh 'echo "Pass : $CREDENCIALES_FASE_UNO_PSW"'
                sh 'echo "Credenciales $CREDENCIALES_FASE_UNO"'
            }
        }
    }
}

```

Options

- Permite definir opciones específicas
- Por ejemplo:
 - buildDiscarder
 - disableConcurrentBuilds
 - checkoutToSubdirectory
 - disableResume
 - quietPeriod
 - retry
 - skipStagesAfterUnstable
 - parallelsAlwaysFailFast
 - timeout
 - timestamps

```

pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS')
    }
    stages {
        stage('Fase uno') {
            steps {
                echo 'Primer step ejecutado'
            }
        }
    }
}

```

```
    }  
}
```

Parameters

- Parámetros que se pueden enviar para ejecutar las tareas
- Permite mostrar via web un formulario con los parametros necesarios
- tipos
 - string
 - text
 - booleanParam
 - choice
 - password

```
pipeline {  
    agent any  
    parameters {  
        string(name: 'NOMBRE', defaultValue: 'Ruben', description: 'Cual es tu nombre?')  
        text(name: 'APELLIDO', defaultValue: '', description: 'Y tu apellido?')  
        booleanParam(name: 'BOOLEANO', defaultValue: true, description: 'Ejemplo booleano')  
        choice(name: 'ELECCION', choices: ['Primero', 'Segundo', 'Tercero'], description: 'Elige')  
        password(name: 'PASSWORD', defaultValue: 'secreto', description: 'Indica la contraseña')  
    }  
    stages {  
        stage('Example') {  
            steps {  
                echo "Hello ${params.NOMBRE}"  
                echo "Biography: ${params.APELLIDO}"  
                echo "Toggle: ${params.BOOLEANO}"  
                echo "Choice: ${params.ELECCION}"  
                echo "Password: ${params.PASSWORD}"  
            }  
        }  
    }  
}
```

input

- Permite definir parámetros de entrada obligando a introducirlos en fases concretas
- Se encuentra dentro de un stage y permite parar la ejecución esperando un input concreto
- Posee los siguientes atributos
 - message: Mensaje a mostrar
 - id: Identificador opcional
 - ok: Texto del botón del formulario
 - submitter: listas de usuarios o grupos que no pueden usar el input

- submitterParameter: Parametros para usar en los campos submitter
- parameters: Lista de parametros a preguntar

```
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."
            }
        }
    }
}
```

Triggers

- Permite por medio de pollSCM, cron y upstream ejecutar este pipeline

```
pipeline {
    agent any
    triggers {
        cron('0 0 * * *')
    }
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

- Este ejemplo se dispara todos los dias a las 00:00

when

- Condicional para una stage
- Es obligatorio que posea al menos una condición
- condiciones
 - branch
 - buildingTag

- changelog
- changeset
- changeRequest
- environment
- equals
- expression
- tag
- not
- allOf
- anyOf
- triggeredBy

```

pipeline {
    agent any
    stages {
        stage('Fase inicial') {
            steps {
                echo 'Primera fase'
            }
        }
        stage('Fase intermedia') {
            when {
                branch 'master'
                andOf {
                    environment name: 'DESPLIEGUE', value 'produccion'
                    environment name: 'DESPLIEGUE', value 'preproduccion'
                }
            }
            steps {
                echo 'Ejecutando fase intermedia'
            }
        }
        stage('Fase final') {
            when {
                branch 'master'
                environment name: 'DESPLIEGUE', value 'produccion'
            }
            steps {
                echo 'Ejecutando fase final de produccion'
            }
        }
    }
}

```

7.12.4. Lab: Creación de un pipeline con jenkinsfile

- Vamos a realizar una prueba sencilla de un pipeline de jenkins
- Para ello vamos a crear distintos pipelines que permitan utilizar algunos de los recursos asociados
- Para ello, primero vamos a crear un proyecto en local para poder hacer más rápidamente las pruebas con jenkins

```
[ansible@ansible-master ~]$ sudo mkdir /projeto-jenkinsfile
[ansible@ansible-master ~]$ sudo chown -R ansible.jenkins /projeto-jenkinsfile
[ansible@ansible-master ~]$ cd /projeto-jenkinsfile
[ansible@ansible-master projeto-jenkinsfile]$ echo "Proyecto Jenkinsfile" > README.md
[ansible@ansible-master projeto-jenkinsfile]$ git init
Initialized empty Git repository in /home/ansible/projeto-jenkinsfile/.git/
```

- En el repositorio, creamos el fichero Jenkinsfile, en la raiz

Jenkinsfile

```
pipeline {
    agent any
    stages {
        stage('Primera fase') {
            steps {
                echo 'Ejecutando primera fase'
                sh 'sleep 10'
            }
        }
        stage('Segunda fase') {
            steps {
                echo 'Ejecutando segunda fase'
                sh 'sleep 10'
            }
        }
        stage('Tercera fase') {
            steps {
                echo 'Ejecutando tercera fase'
                sh 'sleep 10'
            }
        }
    }
}
```

- Salvamos los cambios en el repositorio

```
[ansible@ansible-master projeto-jenkinsfile]$ git commit
[ansible@ansible-master projeto-jenkinsfile]$ git commit -m "Primer commit"
[master (root-commit) 6100bda] Primer commit
```

```
2 files changed, 24 insertions(+)
create mode 100644 README.md
create mode 100644 Jenkinsfile
```

- Ahora abrimos jenkins y vamos a crear un proyecto multibranch

Enter an item name

Ejemplo con Jenkinsfile

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Bitbucket Team/Project
Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.

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 Type to autocomplete

OK

- Agregamos un nuevo repositorio git

General

Display Name:

Description:

[Plain text] [Preview](#)

Branch Sources

Add source: [Bitbucket](#), [Git](#) (highlighted), [GitHub](#), [Mercurial](#), [Single repository & branch](#), [Subversion](#)

Script Path: Jenkinsfile

Relative location within the checkout of your Pipeline script. Note that it will always be run inside a Groovy sandbox. Default is [Jenkinsfile](#) if left empty. (just use `checkout scm` to retrieve sources from the same location as is configured here)

(from [Pipeline: Multibranch](#))

Scan Multibranch Pipeline Triggers

Periodically if not otherwise run

Orphaned Item Strategy

Jobs for removed SCM heads (i.e. deleted branches) can be removed immediately or kept based on a desired retention strategy. By default, jobs will be removed as soon as Jenkins determines their associated SCM head no longer exists. As an example, it may be useful to configure a different retention strategy to be able to examine build results of a branch after it has been removed.

Discard old items

Days to keep old items:

if not empty, old items are only kept up to this number of days

Max # of old items to keep:

if not empty, only up to this number of old items are kept

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

- Definimos como repositorio el path del directorio creado

- file//proyecto-jenkinsfile



- file// indica que es una ruta de la máquina actual

- Por último, guardamos y vemos como se dispara el script dejandonos ver en que stage está operando

Jenkins > Ejemplo con Jenkinsfile > master >

[Up](#) [Status](#) [Changes](#) [Build Now](#) [View Configuration](#) [Full Stage View](#) [Open Blue Ocean](#) [Pipeline Syntax](#)

Pipeline master

Full project name: Ejemplo con Jenkinsfile/master

[Recent Changes](#)

Stage View

Declarative: Checkout SCM	Primera fase	Segunda fase	Tercera fase
204ms	10s	10s	10s
204ms	10s	10s	10s

Average stage times: (Average full run time: ~36s)

Dec 03 15:25 No Changes

Permalinks

- [Last build \(#1\), 3 sec ago](#)

Uso de variables

- Vamos a usar variables de entorno para poder reutilizarlas mas adelante

Jenkinsfile

```
pipeline {
    agent any
    environment {
        SERVIDOR = 'localhost'
    }
    stages {
        stage('Primera fase') {
            steps {
                echo 'Ejecutando primera fase'
                sh 'sleep 10'
                sh 'echo Servidor : $SERVIDOR'
            }
        }
        stage('Segunda fase') {
            environment {
                SERVIDOR = '127.0.0.1'
            }
            steps {
                echo 'Ejecutando segunda fase'
                sh 'sleep 10'
                sh 'echo Servidor : $SERVIDOR'
            }
        }
        stage('Tercera fase') {
            steps {
                echo 'Ejecutando tercera fase'
                sh 'sleep 10'
                sh 'echo Servidor : $SERVIDOR'
            }
        }
    }
}
```

- Ahora realizamos un commit y comprobamos la salida de la ejecución

```
[ansible@ansible-master proyecto-jenkinsfile]$ git commit -am "Environment"
[master cc5e713] Environment
1 file changed, 2 insertions(+), 2 deletions(-)
```

Jenkins > Ejemplo con Jenkinsfile > master > #5

```
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Primera fase)
[Pipeline] echo
Ejecutando primera fase
[Pipeline] sh
+ sleep 10
[Pipeline] sh
+ echo Servidor : localhost
Servidor : localhost
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Segunda fase)
[Pipeline] withEnv
[Pipeline] {
[Pipeline] echo
Ejecutando segunda fase
[Pipeline] sh
+ sleep 10
[Pipeline] sh
+ echo Servidor : 127.0.0.1
Servidor : 127.0.0.1
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Tercera fase)
[Pipeline] echo
Ejecutando tercera fase
[Pipeline] sh
+ sleep 10
[Pipeline] sh
+ echo Servidor : localhost
Servidor : localhost
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

- Ahora vamos a agregar unas variables para que se puedan llenar en ejecución
- Para ello modificamos el Jenkinsfile

Jenkinsfile

```
pipeline {
    agent any
    environment {
        SERVIDOR = 'localhost'
    }
    parameters {
        string(name: 'NOMBRE', defaultValue: 'Ruben', description: 'Cual es tu nombre?')
        text(name: 'APELLIDO', defaultValue: '', description: 'Y tu apellido?')
        booleanParam(name: 'BOOLEANO', defaultValue: true, description: 'Ejemplo booleano')
        choice(name: 'ELECCION', choices: ['Primero', 'Segundo', 'Tercero'], description: 'Elige')
        password(name: 'PASSWORD', defaultValue: 'secreto', description: 'Indica la contraseña')
    }
    stages {
        stage('Primera fase') {
            steps {
                echo 'Ejecutando primera fase'
                sh 'sleep 10'
                sh 'echo Servidor : $SERVIDOR'
            }
        }
        stage('Segunda fase') {
```

```

environment {
    SERVIDOR = '127.0.0.1'
}
steps {
    echo 'Ejecutando segunda fase'
    sh 'sleep 10'
    sh 'echo Servidor : $SERVIDOR'
}
}
stage('Tercera fase') {
    steps {
        echo 'Ejecutando tercera fase'
        sh 'sleep 10'
        sh 'echo Servidor : $SERVIDOR'
    }
}
stage('Evaluacion de variables') {
    steps {
        echo 'Ejecutando evaluacion de variables'
        sh 'sleep 10'
        sh 'echo Servidor : ${SERVIDOR}'
        sh "echo Nombre : ${params.NOMBRE}"
        sh "echo Apellido : ${params.APELLIDO}"
        sh "echo Selector : ${params.ELECCION}"
        sh "echo Contraseña : ${params.PASSWORD}"
    }
}
}
}
}

```

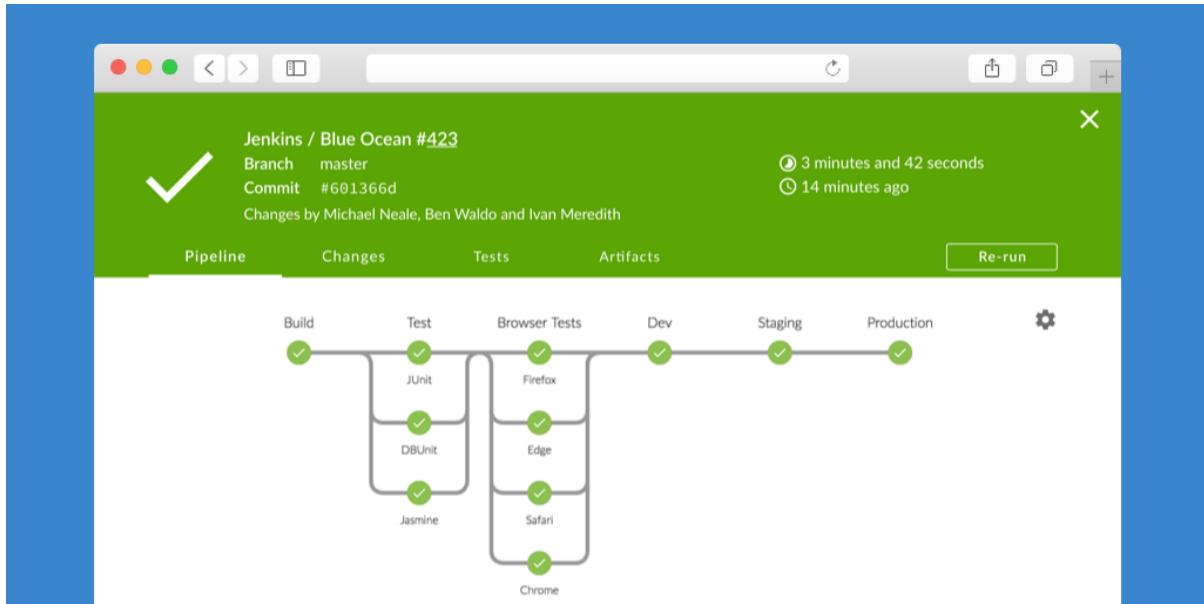
- Como podemos comprobar, es muy sencillo generar stages y steps

7.13. Plugin Blue Ocean

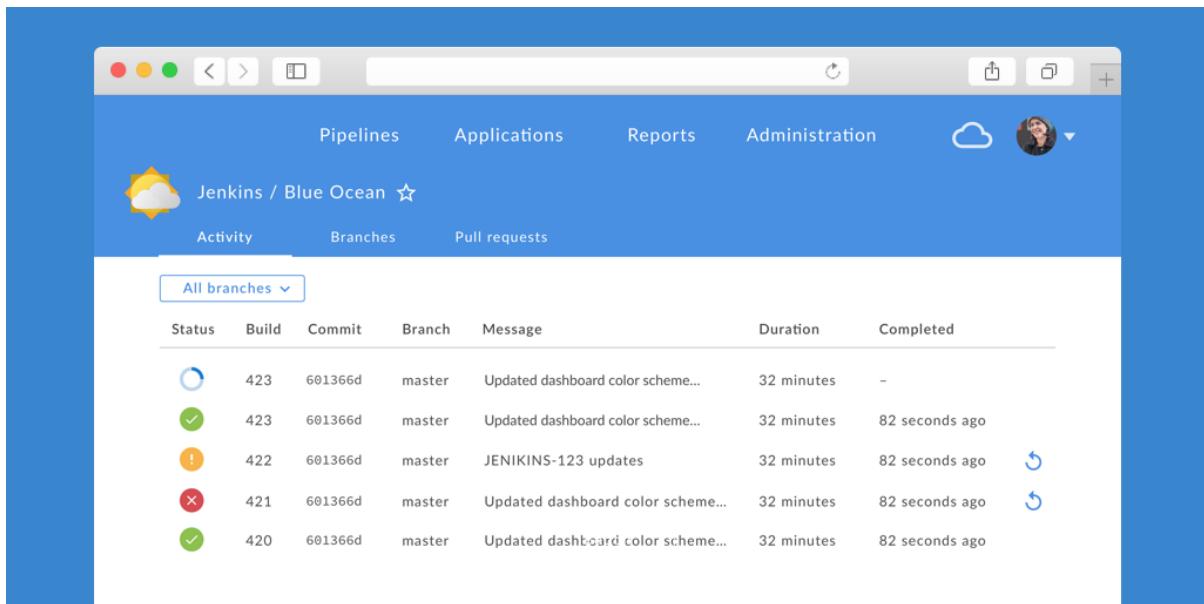
- Se trata de un plugin que permite cambiar la experiencia de Jenkins
- Permite modelar y presentar un proceso de entrega de software mostrando lo que es realmente importante, sin salir del core de jenkins
- Permite mostrar de forma paralela la misma información que el UI de Jenkins clásico
- Se trata de un plugin Open Source muy sencillo de gestionar

7.13.1. Gestión de Pipelines

- Permite mostrar de forma sencilla tanto ejecuciones de tareas como la salida de los pipelines

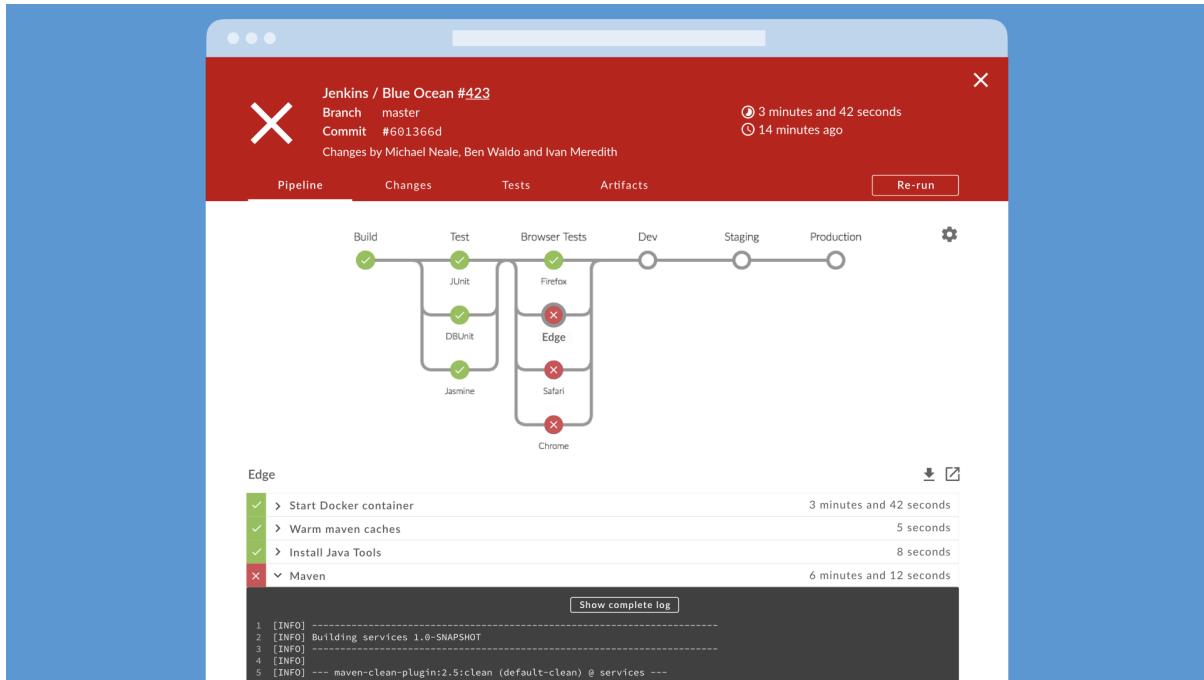


- Permite comprobar la salud de cada pipeline de forma sencilla



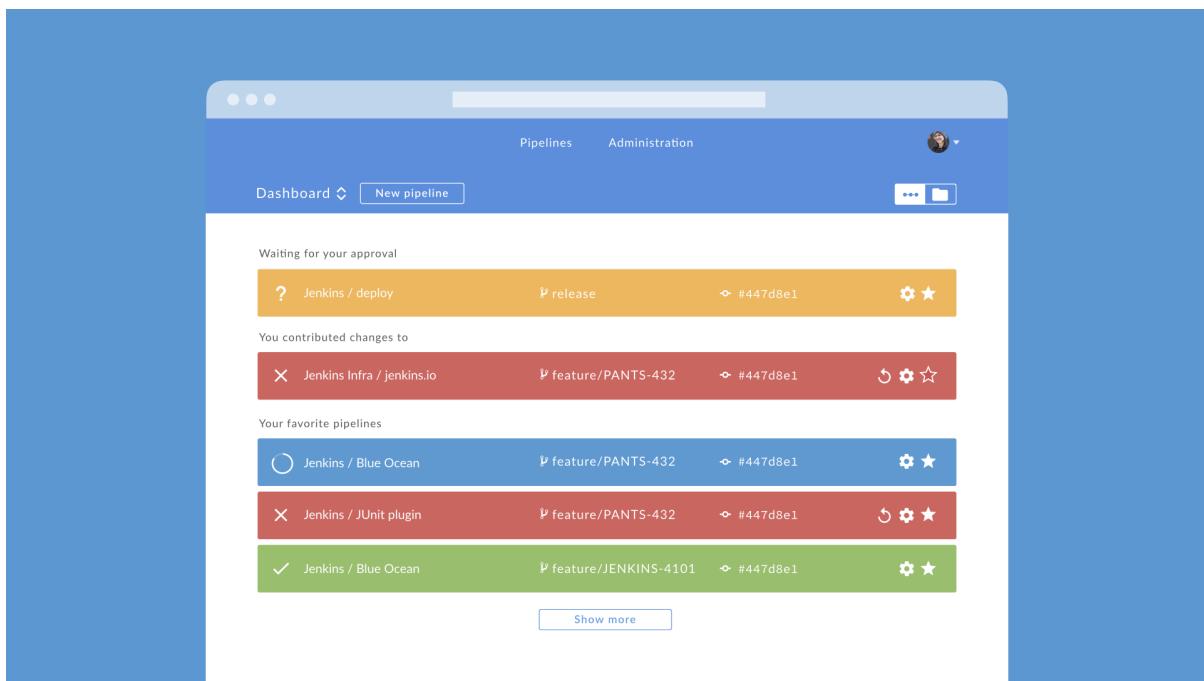
7.13.2. Ramas y Pull Requests

- Los pipelines suelen utilizar múltiples ramas de Git
- Con poseer un Jenkinsfile en un repositorio de Git, es descubierto por Blue Ocean y ejecutado por jenkins
- Permite mostrar en todo momento en que rama estamos, el commit actual y la ejecución del mismo
- Se muestra fácilmente los errores del pipeline



7.13.3. Dashboards

- Permite el uso de dashboards personalizados por el usuario para mostrar lo que mas necesitamos



7.13.4. Lab: Uso de Blue Ocean

- Para el uso del plugin de Blue Ocean es necesario instalar varios plugins

Instalación

- Para ello vamos a la pestaña de plugins disponibles e instalamos los plugins de Blue Ocean
- Es obligatorio el plugin de Pipeline Editor, el core y el plugin de blue Ocean, aunque para poder usar todo su potencial es recomendable instalar todos los plugins de Blue Ocean

Updates	Available	Installed	Advanced	Name ↓	Version	Previously installed version	Uninstall
				Apache HttpComponents Client 4.x API Plugin	4.5.10-	2.0	<button>Uninstall</button>
				Bundles Apache HttpComponents Client 4.x and allows it to be used by Jenkins plugins.			
				Autofavorite for Blue Ocean	1.2.4		<button>Uninstall</button>
				Automatically favorites multibranch pipeline jobs when user is the author			
				Bitbucket Branch Source Plugin	2.6.0		<button>Uninstall</button>
				Allows to use Bitbucket Cloud and Bitbucket Server as sources for multi-branch projects. It also provides the required connectors for Bitbucket Cloud Team and Bitbucket Server Project folder (also known as repositories auto-discovering).			
				Bitbucket Pipeline for Blue Ocean	1.21.0		<button>Uninstall</button>
				BlueOcean Bitbucket pipeline creator			
				Blue Ocean	1.21.0		<button>Uninstall</button>
				BlueOcean Aggregator			
				Blue Ocean Core JS	1.21.0		<button>Uninstall</button>
				The Jenkins Plugins Parent POM Project			
				Blue Ocean Executor Info	1.21.0		<button>Uninstall</button>
				Show what executors are doing what.			
				Blue Ocean Pipeline Editor	1.21.0		<button>Uninstall</button>
				The Blue Ocean Pipeline Editor is the simplest way for anyone wanting to get started with creating Pipelines in Jenkins			
				Branch API Plugin	2.5.4		<button>Uninstall</button>
				This plugin provides an API for multiple branch based projects.			
				Common API for Blue Ocean	1.21.0		<button>Uninstall</button>
				This plugin is a part of Blue Ocean UI			
				Config API for Blue Ocean	1.21.0		<button>Uninstall</button>
				BlueOcean Analytics Tools plugin			
				Credentials Plugin	2.3.0		<button>Uninstall</button>
				This plugin allows you to store credentials in Jenkins.			
				Dashboard for Blue Ocean	1.21.0		<button>Uninstall</button>
				Blue Ocean Dashboard			
				Design Language	1.21.0		<button>Uninstall</button>

Repositorio

- Antes de comenzar, necesitamos un nuevo repositorio de git
- Para ello nos dirigimos a nuestra cuenta de GitLab

gitlab.com/dashboard/projects

Projects Groups More

Free Trial of GitLab.com Gold Try all GitLab has to offer for 30 days. No credit card required. Start your trial

Projects

Your projects 15 Starred projects 0 Explore projects Filter by name... Last updated

All Personal

- Creamos un nuevo repositorio de nombre BlueOceanJenkins y lo inicializamos con un fichero README.md

New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), [among other things](#).

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

To only use CI/CD features for an external repository, choose [CI/CD for external repo](#).

Information about additional Pages templates and how to install them can be found in our [Pages getting started guide](#).

Tip: You can also create a project from the command line. [Show command](#)

Project name
BlueOceanJenkins

Project URL
https://gitlab.com/ ecosistemas

Project slug
blueoceanjenkins

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)

Description format

Visibility Level

- Private Project access must be granted explicitly to each user.
- Public The project can be accessed without any authentication.

Initialize repository with a README Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project **Cancel**

- Copiamos la url de acceso al proyecto

Ruben > BlueOceanJenkins > Details

Project 'BlueOceanJenkins' was successfully created.

B BlueOceanJenkins [Project ID: 15756822](#)

- 1 Commit | 1 Branch | 0 Tags | 133 KB Files

Clone with SSH
git@gitlab.com:ecosistemas/b

Clone with HTTPS
https://gitlab.com/ecosistem

Auto DevOps

It will automatically build, test, and deploy your application based on changes.

Learn more in the [Auto DevOps documentation](#)

Enable in settings

master blueoceanjenkins / +

Initial commit Ruben authored just now 5dc19bea

README Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster Set up CI/CD

Name	Last commit	Last update
README.md	Initial commit	just now

<< Collapse sidebar

Inicialización

- Ahora nos dirigimos a Jenkins
- Nada mas ir a la página principal, después de instalar los plugins, vamos a la opción Blue Ocean:

The Jenkins dashboard displays the Pipelines section. The table shows the following data:

S	W	Name	Last Success	Last Failure	Last Duration	Fav
		ansible-masters	6 days 23 hr - log	N/A	4.9 sec	
		Ejemplo con Jenkinsfile	5 days 0 hr - log	N/A	0.38 sec	
		EjemploJob	5 days 23 hr - #14	7 days 3 hr - #8	3 min 14 sec	

Icon: S M L

Legend: Atom feed for all Atom feed for failures Atom feed for just latest builds

[New View](#)

[Build Queue](#) No builds in the queue.

[Build Executor Status](#) 1 Idle 2 Idle

Minimize all open windows and show the desktop

Page generated: Dec 8, 2019 4:58:04 PM CET REST API Jenkins ver. 2.206

- Si ya tenemos Pipelines, aparecerán en la lista inicial
- Pulsamos New Pipeline

Jenkins Pipelines

NAME	HEALTH	BRANCHES	PR	
ansible-masters		1 passing	-	
Ejemplo con Jenkinsfile		1 passing	-	
EjemploJob		-	-	

New Pipeline

- Seleccionamos la opción de código como Git
- Nos pedirá la URL y pegamos la url ssh que copiamos de nuestro repositorio
- Debajo aparecerá la clave SSH que debemos copiar al servidor de GitLab para autorizar su acceso



Where do you store your code?



Bitbucket Cloud



Bitbucket Server



Github



Github Enterprise



Git



Connect to a Git repository

Any repository containing a Jenkinsfile will be built automatically. Not sure what we are talking about? [Learn more about Jenkinsfiles.](#)

Repository URL - Please enter a valid URL.

git@gitlab.com:ecosistemas/blueoceanjenkins.git

You need to register this public SSH key with your Git server to continue - [learn more.](#)

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQDT4KkvoFnICOc3bGtww
0H2xOg0Ara3Jq5kdnWAyDGymrGdOjFyAzE0uc5ZXcZjUpTyFOO4/ghl
HUDOeLuNFZvKY2HqJQOGuvylVE1vCPo6Bcb6N8RQ7panmcLCgpmY
+Im06z/YMvnCyzI5AUBFKNx21jDoukyVam9CBoTcUkS+KUn72YSAlz
ybDThQeqAI23pgqPX9dZYT9zldAiA428gTFJQAI0zbVR334Enoy+Dm7l
TbbyuaaBD4jwz3fV0pnWEAV4iCer3efESXqbE6FKdFen2d9tu7e3A3y5l
JO+N710ZJaQy9mEZRinNv5KOJVofvPE/Z+Uf2VdzV5mis3
admin@localhost:8888
```

[Create Pipeline](#)

[Copy to clipboard](#)

- Abrimos Gitlab y pulsamos arriba a la derecha, en el icono de usuario, en la sección de Settings

- Luego seleccionamos la opción del menú derecho **SSH Keys**

- Copiamos la clave SSH y usamos de titulo Blue Ocean

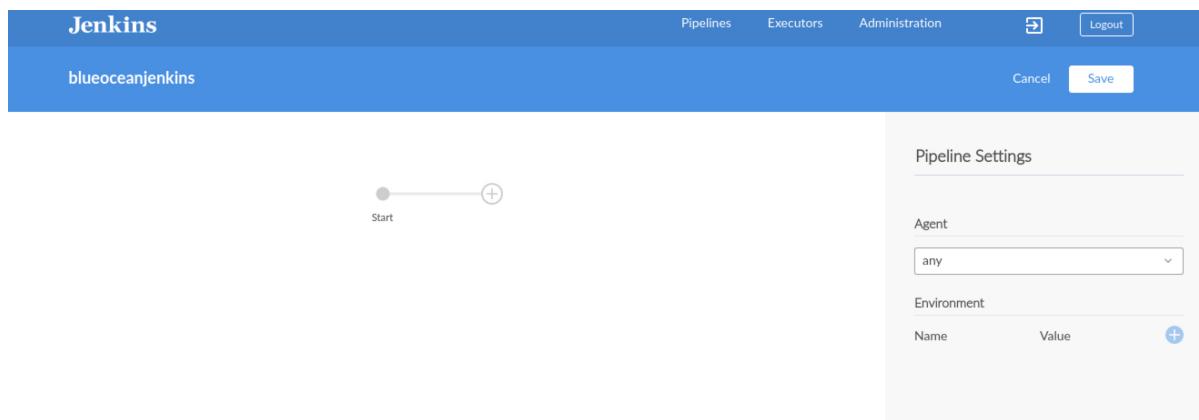
The screenshot shows the GitLab user settings interface. On the left, a sidebar lists various options like Profile, Account, Billing, Applications, Chat, Access Tokens, Emails, Password, Notifications, SSH Keys (which is currently selected), GPG Keys, Preferences, Active Sessions, Authentication log, and Pipeline quota. The main content area is titled 'User Settings > SSH Keys'. It contains a section for adding an SSH key, with instructions to generate or use an existing key. A text input field contains a public SSH key, and a title field is set to 'Blue Ocean SSH Key'. Below this, it shows two keys listed under 'Your SSH keys (2)'. The first key is 'admin@localhost:8088' with a last used date of '3 days ago'. The second key is 'ansible@ansible-master' with a creation date of 'Created 3 days ago'. A note at the bottom says 'You need to register this public SSH key with your Git server to continue - learn more.'

- Volvemos a la url del pipeline y pulsamos en **Create Pipeline**

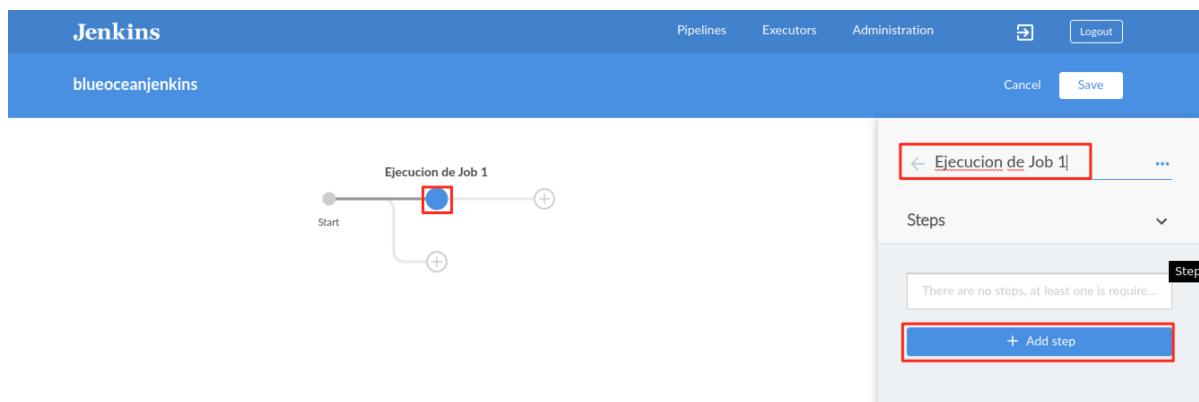
The screenshot shows the Jenkins Pipeline configuration process. Step 1: 'Where do you store your code?' offers choices: Bitbucket Cloud, Bitbucket Server, GitHub, GitHub Enterprise, and Git. 'Git' is selected and highlighted with a blue border. Step 2: 'Connect to a Git repository'. It says 'Any repository containing a Jenkinsfile will be built automatically. Not sure what we are talking about? Learn more about Jenkinsfiles.' Below is a 'Repository URL - Please enter a valid URL.' input field containing 'git@gitlab.com:ecosistemas/blueoceanjenkins.git'. A large text area below shows the same SSH key as the previous screenshot. At the bottom are 'Create Pipeline' and 'Copy to clipboard' buttons, with 'Create Pipeline' being highlighted with a red box.

Creación de pipeline

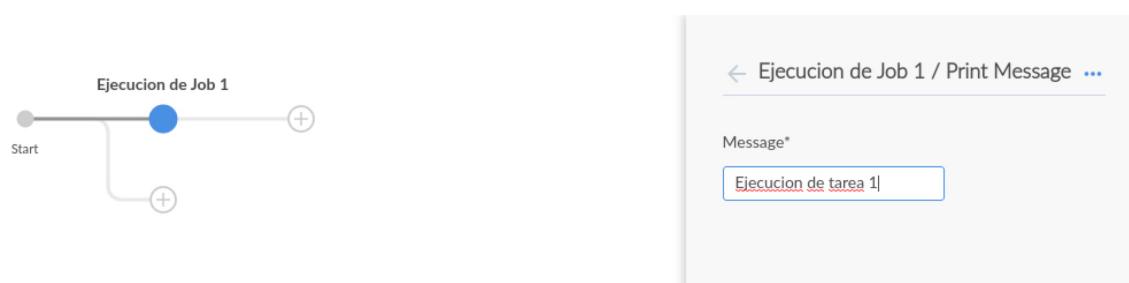
- Para crearlo, nos aparece un formato muy simple, donde en el centro está el diagrama de stages/steps y a la derecha la sección de formulario para completar las acciones



- Agregamos un nuevo nodo y lo llamamos: **Ejecucion de Job 1**
- Pulsamos luego un nuevo Step para indicar la acción de este stage



- Buscamos en la lista la acción **Print message** y escribimos "Ejecucion de tarea 1"



- Agregamos otro step y buscamos la opción sleep y rellenamos a 5 segundos

← Ejecucion de Job 1 / Sleep ...

Time*

Unit

- Pulsamos ahora en el nodo inferior, indicando que vamos a ejecutar otra tarea en paralelo a la primera



- Llamamos al stage **Ejecución en paralelo a Job 1**

← Ejecucion en paralelo a Job 1 ...

Steps

There are no steps, at least one is required...

+ Add step

- Agregamos el step **Print Message** con el mensaje "Ejecutando en paralelo a job 1"

← Ejecucion en paralelo a Job 1 / Print Message ...

Message*

- Ahora en la sección superior a los dos nodos, pulsamos e indicamos como nombre Fase 1

← Fase 1 ...

Settings

Name	Value

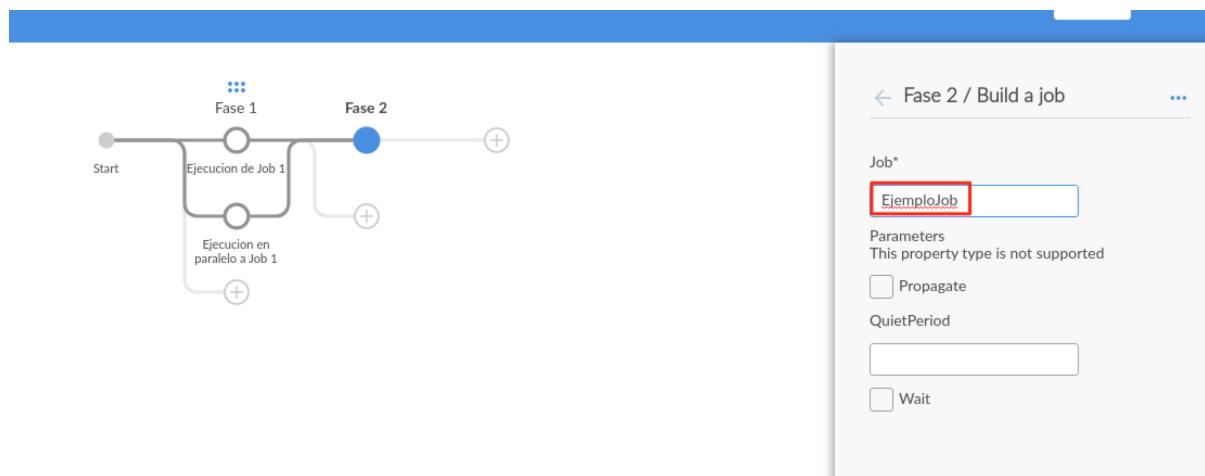
Environment

Name	Value

- Seleccionamos un nuevo nodo y lo llamamos Fase 2
- Agregamos como tarea **Build a Job**



- El job lo llamamos EjemploJob y pulsamos save



- Ahora guardamos el Pipeline como commit en la rama master

Save Pipeline

Saving the pipeline will commit a Jenkinsfile to the repository.

Description

Prueba de Jenkins lanzando dos tareas en paralelo y una tercera tras finalizar las dos primeras

Commit to master
 Commit to new branch
 master

Save & run **Cancel**

- En la primera ejecución solo subirá el commit

- Ahora pulsamos en ejecutar y vemos el pipeline en ejecución pero fallará en la fase 2

- Para que funcione, debe existir un Job con nombre EjemploJob. Para ello salimos de Blue Ocean pulsando el aspa superior derecha.
- Creamos en el menú de jenkins un nuevo job

- En el job accedemos a la sección build y escribimos un echo diciendo que todo está correcto

Build

The screenshot shows the Jenkins build configuration interface. A single step is defined: 'Execute shell' with the command 'echo "Todo correcto!"'. Below the command, there's a link to 'See the list of available environment variables'. At the bottom, there are 'Save' and 'Apply' buttons.

- Probamos el job para ver si funciona correctamente

The screenshot shows the Jenkins console output for build #1 of the 'EjemploJob'. The log shows the command 'echo "Todo correcto!"' was executed and printed 'Todo correcto!'. The status is listed as 'SUCCESS'.

- Volvemos a ir a Blue Ocean a la sección **Branches** de los pipelines

The screenshot shows the Blue Ocean pipeline interface. In the 'Branches' section, there is one branch named 'master' which is currently 'Replayed #1' and completed 4 minutes ago.

- Pasamos el ratón por la fila y pulsamos ejecutar

The screenshot shows the Blue Ocean pipeline interface during execution. The pipeline consists of 'Fase 1' and 'Fase 2'. 'Fase 1' is completed with a green checkmark. 'Fase 2' is currently executing, indicated by a blue progress bar. The status bar at the top shows 'blueoceanjenkins < 3' and 'Started by user admin'. The pipeline navigation bar includes 'Pipeline', 'Changes', 'Tests', 'Artifacts', and 'Logout'.

- Ahora si que se ejecuta correctamente
- Si accedemos al repositorio veremos como nos ha generado un Jenkinsfile con las tareas a ejecutar

The screenshot shows the BlueOcean Jenkins interface for a project named "BlueOceanJenkins". The left sidebar contains links for Project overview, Repository (Commits, Branches, Tags, Contributors, Graph, Compare, Charts), Issues (0), Merge Requests (0), CI / CD, Operations, Packages, and Wiki.

The main area displays the Jenkinsfile content:

```
1 pipeline {
2     agent any
3     stages {
4         stage('Fase 1') {
5             parallel {
6                 stage('Ejecucion de Job 1') {
7                     steps {
8                         echo 'Ejecucion de tarea 1'
9                         sleep 5
10                    }
11                }
12            }
13            stage('Ejecucion en paralelo a Job 1') {
14                steps {
15                    echo 'Ejecutando en paralelo a job 1'
16                }
17            }
18        }
19    }
20
21
22    stage('Fase 2') {
23        steps {
24            build 'EjemploJob'
25        }
26    }
27
28
29 }
```

At the top right of the main area, there are buttons for Edit, Web IDE, Replace, Delete, and download options.

Chapter 8. Scripting con Jenkins CLI

Jenkins posee una interfaz de comandos que permite a usuarios y administradores acceso a jenkins a un endpoint de script o a una shell. La forma de acceso puede ser por medio de acceso http o SSH.

Podemos obtener información detallada al respecto en la siguiente URL: <https://jenkins.io/doc/book/managing/cli/>

8.1. Lab: Conectando a Jenkins con cliente SSH

Mediante este laboratorio, vamos a ajustar a Jenkins para poder conectarnos a el, mediante SSH y ejecutar algunos comandos

8.1.1. Activando acceso SSH

Desde el dashboard principal de Jenkins, accedemos a la opción **Manage Jenkins > Configure Global Security**

Una vez dentro de la sección, localizamos la sección que indica **SSHD Server**, aquí tenemos varias opciones para habilitar el acceso mediante consola:

- **Fixed**

- Podemos indicar un puerto específico para conectar por SSH a Jenkins

- **Random**

- Indicamos a Jenkins, que nos indique un puerto aleatorio para poder conectar

- **Disable**

- Indicamos a Jenkins que deshabilite la opción de conexión

En nuestro caso, vamos a indicar la opción **Random** y pulsamos **Save**

Una vez hayamos llevado a cabo la acción, debemos de consultar qué puerto ha asignado Jenkins para llevar a cabo conexiones de cliente mediante SSH



Indicamos la URL donde se encuentra alojado el nodo principal

Abrimos una consola y ejecutamos el siguiente comando:

```
$ curl -Lv http://localhost:8081/login 2>&1 | grep -i 'x-ssh-endpoint'  
< X-SSH-Endpoint: localhost:42397
```

En este caso, observamos la cabecera que nos da Jenkins en el response de la petición, y nos informa del puerto que está habilitado en este momento

8.1.2. Creando el par clave pública/privada

Vamos a generar el par de clave pública/privada en nuestro equipo donde queremos actuar como cliente

Dejamos todos los parámetros que nos preguntará por defecto (todos pulsamos INTRO) :D

Ejecutamos la siguiente instrucción en consola:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kubernetes/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kubernetes/.ssh/id_rsa.
Your public key has been saved in /home/kubernetes/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:8vJfvhApLqceIIL4RZ4wgPLiNvBZtXpEvuafvwCBFg kubernetes@kubemaster.local
The key's randomart image is:
+---[RSA 2048]---+
|0.0 . .00      |
|0 = .+.        |
|.. ++ o .     |
|+..o. + E ..   |
|=0+o.o..S o.   |
|=0...++ ...    |
|.. . ++ + ...  |
|       .B . +.  |
|       .o.+o+oo. |
+---[SHA256]---+
```

8.1.3. Añadiendo nuestra clave pública generada en consola a usuario Jenkins

A continuación, vamos a añadir en la sección de configuración de nuestro usuario nuestra clave pública que hemos generado en nuestra consola

La acción la vamos a llevar a cabo con el usuario **admin** que tenemos registrado en Jenkins, en el nodo maestro

Indicamos en el navegador la siguiente URL: <http://localhost:8081/user/admin/configure>

Aparecerá un bloque de configuración llamado **SSH Public Keys**, ahí pegamos el contenido de nuestra clave pública anteriormente generada y pulsamos en **Save**

8.1.4. Consultando la ayuda de la consola

Vamos a ejecutar un comando que nos indique la ayuda de los comandos disponibles:

```
$ ssh -l admin -p 42397 localhost help
```

En este caso, nos mostraría por consola la ayuda de los comandos del cliente Jenkins

La descripción de los parámetros sería:

- **ssh**
 - Invocación de conexión ssh
- **-l**
 - Indicación sobre que vamos a suministrar un nombre de login de usuario
- **admin**
 - Nombre de usuario con el que queremos llevar a cabo la conexión
- **-p**
 - Indicación que vamos a indicar un puerto para la conexión en lugar del de por defecto (22)
- **42397**
 - Puerto mediante el cual queremos conectar
- **localhost**
 - IP donde se encuentra el sistema Jenkins al cual nos queremos conectar por consola
- **help**
 - Comando que queremos ejecutar

8.1.5. Consultando ayuda del comando build

Podemos solicitar a su vez, la ayuda específica de un comando en concreto, en nuestro caso, vamos a solicitar ayuda del comando build

```
$ ssh -l admin -p 42397 localhost help build
```

java -jar jenkins-cli.jar build JOB [-c] [-f] [-p] [-r N] [-s] [-v] [-w]
Starts a build, and optionally waits for a completion.
Aside from general scripting use, this command can be
used to invoke another job from within a build of one job.
With the -s option, this command changes the exit code based on
the outcome of the build (exit code 0 indicates a success)
and interrupting the command will interrupt the job.
With the -f option, this command changes the exit code based on
the outcome of the build (exit code 0 indicates a success)
however, unlike -s, interrupting the command will not interrupt
the job (exit code 125 indicates the command was interrupted).
With the -c option, a build will only run if there has been
an SCM change.
JOB : Name of the job to build
-c : Check for SCM changes before starting the build, and if there's no

```
change, exit without doing a build (default: false)
-f : Follow the build progress. Like -s only interrupts are not passed
      through to the build. (default: false)
-p : Specify the build parameters in the key=value format. (default: {})
-s : Wait until the completion/abortion of the command. Interrupts are passed
      through to the build. (default: false)
-v : Prints out the console output of the build. Use with -s (default: false)
-w : Wait until the start of the command (default: false)
```

8.1.6. Obteniendo información de usuario

Vamos a ejecutar un comando para determinar qué usuario soy al conectar, ejecutamos la siguiente instrucción en consola:

```
$ ssh -l admin -p 42397 localhost who-am-i
```

```
Authenticated as: admin
Authorities:
  authenticated
```

8.1.7. Ejecutar un job

Vamos a crear un job de estilo libre [Tarea1](#) que se va a encargar de mostrar el valor de un parámetro al que le vamos a dar un valor por defecto.

Esta ejecución debe parametrizarse

Parámetro de texto

Name	NOMBRE
Default Value	Charly
Description	
[Plain text] Visualizar	

Añadir un parámetro ▾

Ejecutar

Ejecutar linea de comandos (shell)

Comando `echo "Nombre: $NOMBRE"`

[Visualizar la lista de variables de entorno disponibles](#)

Añadir un nuevo paso ▾

Una vez creado el job, vamos a ejecutarlo usando la opción `build` seguida del nombre de la tarea a ejecutar:

```
$ ssh -l admin -p 42397 localhost build Tarea1
```

8.1.8. Ejecutar un job parametrizado

Podemos ejecutar un job parametrizado, la única diferencia con el ejemplo anterior es que tenemos que mandar los valores de los parámetros usando `-p key=val`.

```
$ ssh -l admin -p 42397 localhost build Tarea1 -p NOMBRE=Charly
```

8.1.9. Mostrar la salida de consola de una ejecución

Al ejecutar los jobs podemos añadir las opciones `-s -v` para que se nos muestre la salida por consola de la ejecución que se ha realizado.

```
$ ssh -l admin -p 42397 localhost build Tarea1 -p NOMBRE=Charly -s -v
```

8.1.10. Listar jobs

También podemos listar todos los jobs que tenemos en Jenkins usando la opción `list-jobs`:

```
$ ssh -l admin -p 42397 localhost list-jobs
```

Chapter 9. Consola de Script Integrada

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

The screenshot shows the Jenkins administration interface. On the left, there's a sidebar with links like 'Nueva Tarea', 'Personas', 'Historial de trabajos', 'Administrar Jenkins', 'Mis vistas', and 'Credentials'. Below that is a section for 'Trabajos en la cola' which says 'No hay trabajos en la cola'. Under 'Estado del ejecutor de construcciones', there are two inactive entries. The main area is titled 'Administrar Jenkins' and lists various management options. One option, 'Consola de scripts' (highlighted with a red box), is described as 'Ejecutar script para la administración, diagnóstico y solución de problemas.' At the bottom of the list are links for 'Administrador Nodos', 'Gestión de credenciales', 'Acerca de Jenkins', 'Datos antiguos', 'Gestión de usuarios', 'In-process Script Approval', and 'Preparar Jenkins para apagar el contenedor'.

The screenshot shows the Jenkins console interface. The left sidebar is identical to the previous one. The main area is titled 'Consola de scripts' and contains a text input field with a Groovy script. The script prints the root directory of Jenkins, iterates through files in that directory, and prints their paths if they exceed a certain size. It also calculates the total space used. A 'Ejecutar' button is at the bottom right of the script editor. Below the editor, the output of the script is shown in a 'Resultado' section. The output shows the script thinking about deleting several files and printing the total space used and number of files.

Chapter 10. API de acceso remoto

Jenkins dispone de una API totalmente integrada para poder obtener información así como llevar a cabo ejecución de tareas de forma remota, sin necesidad de utilizar el panel de control gráfico vía web

Para consultar detalles de la misma, así como las opciones de las que nos provee, visualizando el dashboard principal de Jenkins, abajo a la derecha vamos a observar una opción que dice **REST API**

Si hacemos clic sobre la opción, observaremos el detalle documental de cómo acceder

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

- Lanzar un build de un tarea.
- Deshabilitar/Habilitar una tarea
- Borrar una tarea.
- Etc.

10.1. Lab: Acceso API Remoto

Mediante este laboratorio, vamos a llevar a cabo algunas de las operaciones más comunes con la API de Jenkins

10.1.1. Obtener la API JSON disponible

Para consultar la API que tenemos disponible en formato JSON, indicamos la siguiente URL en el navegador:

```
http://localhost:8081/api/json?pretty=true

{
  "_class" : "hudson.model.Hudson",
  "assignedLabels" : [
    {
      "name" : "master"
    }
  ],
  "mode" : "NORMAL",
  "nodeDescription" : "the master Jenkins node",
  "nodeName" : "",
  "numExecutors" : 2,
  "description" : null,
  "jobs" : [
  ],
  "overallLoad" : {
```

```

},
"primaryView" : {
  "_class" : "hudson.model.AllView",
  "name" : "all",
  "url" : "http://localhost:8081/"
},
"quietingDown" : false,
"slaveAgentPort" : -1,
"unlabeledLoad" : {
  "_class" : "jenkins.model.UnlabeledLoadStatistics"
},
"useCrumbs" : true,
"useSecurity" : true,
"views" : [
  {
    "_class" : "hudson.model.MyView",
    "name" : "My View One",
    "url" : "http://localhost:8081/view/My%20View%20One/"
  },
  {
    "_class" : "hudson.model.MyView",
    "name" : "My View Two",
    "url" : "http://localhost:8081/view/My%20View%20Two/"
  },
  {
    "_class" : "hudson.model.AllView",
    "name" : "all",
    "url" : "http://localhost:8081/"
  }
]
}

```

10.1.2. Obteniendo estadísticas sobre la carga actual de Jenkins

Indicamos en el navegador la siguiente URL:

```
http://localhost:8081/overallLoad/api/json?pretty=true
```

```
{
  "_class" : "hudson.model.OverallLoadStatistics",
  "availableExecutors" : {

  },
  "busyExecutors" : {

  },
  "connectingExecutors" : {

  },
}
```

```
"definedExecutors" : {  
},  
"idleExecutors" : {  
},  
"onlineExecutors" : {  
},  
"queueLength" : {  
},  
"totalExecutors" : {  
},  
"totalQueueLength" : {  
}  
}
```

10.1.3. Obteniendo las vistas que tenga como usuario

Indicamos en el navegador la siguiente URL:

```
http://localhost:8081/me/my-views/api/json?pretty=true  
  
{  
    "_class" : "hudson.model.AllView",  
    "description" : null,  
    "jobs" : [  
  
    ],  
    "name" : "all",  
    "property" : [  
  
    ],  
    "url" : "http://localhost:8081/user/admin/my-views/"  
}
```

10.1.4. Ejecución de tareas

Con la API remota de Jenkins podemos ejecutar las tareas realizando una petición HTTP de tipo POST.

Vamos a empezar creando un job sencillo que después ejecutaremos desde Postman o cualquier otra herramienta que nos permita realizar una petición de este tipo.



Una vez que la tenemos creada, vamos a ver las distintas formas de ejecutar la tarea de forma remota, desde algún script o herramienta que permita realizar una petición POST. En este caso usaremos <https://www.postman.com/>.

Para ejecutar un job de Jenkins, tenemos que hacer una petición **POST** usando uno de los métodos que se van a ver a continuación.

Con crumb

Lo primero que vamos a hacer es realizar una petición POST a la URL del job que queremos ejecutar concatenandole **/build** quedando una url como <http://localhost:8080/job/Tarea-Api-Remota/build>.

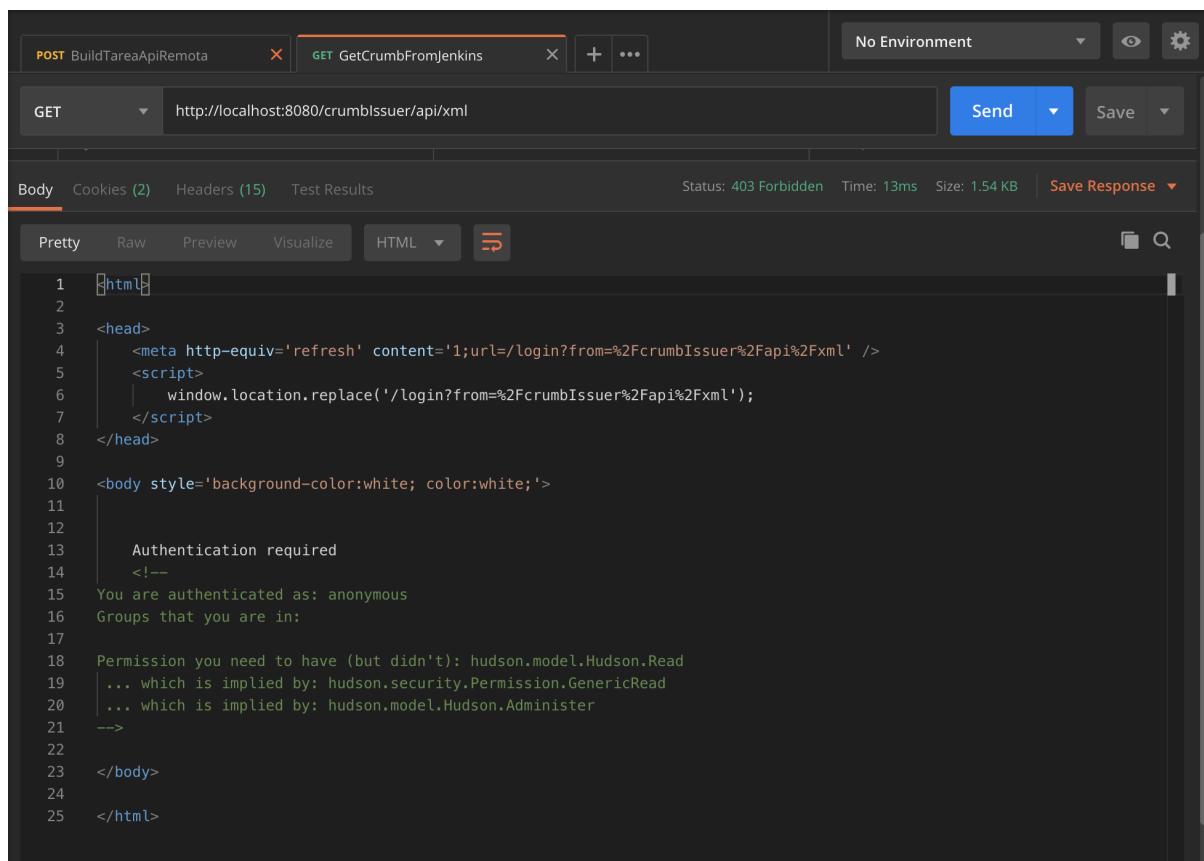
```

1 <html>
2 <head>
3   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4   <title>Error 403 No valid crumb was included in the request</title>
5 </head>
6
7 <body>
8   <h2>HTTP ERROR 403</h2>
9   <p>Problem accessing /job/Tarea-Api-Remota/build. Reason:<br/>
10  |   <pre>    No valid crumb was included in the request</pre>
11  |</pre>
12  </p>
13  <hr><a href="http://eclipse.org/jetty">Powered by Jetty:// 9.4.z-SNAPSHOT</a>
14  <hr />
15
16 </body>
17

```

Esta petición nos ha devuelto un error indicando que no se ha proporcionado **crumb** válido, y esto se debe a que Jenkins nos protege contra ataques CSRF y por lo tanto para evitar que se hagan peticiones haciéndose pasar por nosotros, Jenkins nos tiene que proporcionar un código que se tiene que enviar en la petición que hemos realizado.

Para obtener este código crumb que se nos indica en la respuesta de error, tenemos que realizar primero una petición **GET** a <http://localhost:8080/crumbIssuer/api/xml> que nos devolverá unos datos.

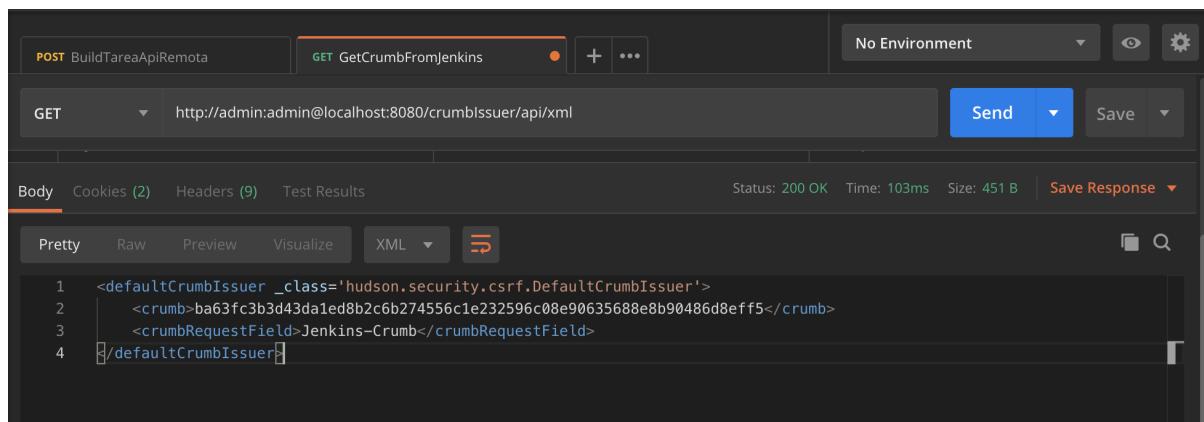


The screenshot shows the Postman interface with a 'GET' request to 'http://localhost:8080/crumbIssuer/api/xml'. The response status is '403 Forbidden'. The body of the response contains an HTML page with an 'Authentication required' message and a 'Groups that you are in:' section, indicating that authentication is needed to access the data.

```
1 <html>
2   <head>
3     <meta http-equiv='refresh' content='1;url=/login?from=%2FcrumbIssuer%2Fapi%2Fxml' />
4     <script>
5       | window.location.replace('/login?from=%2FcrumbIssuer%2Fapi%2Fxml');
6     </script>
7   </head>
8
9   <body style='background-color:white; color:white;'>
10
11
12     Authentication required
13     <!--
14
15 You are authenticated as: anonymous
16 Groups that you are in:
17
18 Permission you need to have (but didn't): hudson.model.Hudson.Read
19 | ... which is implied by: hudson.security.Permission.GenericRead
20 | ... which is implied by: hudson.model.Hudson.Administer
21 -->
22
23 </body>
24
25 </html>
```

En este caso vemos que en lugar de devolvernos los datos que hemos pedido nos responde con otro error. Esta vez nos indica que necesitamos autenticarnos para poder acceder a los datos que estamos pidiendo.

Para enviar los datos de autenticación en la petición y así obtener el crumb, lo que tenemos que hacer es poner **user:password@** justo antes del dominio donde se encuentra nuestro Jenkins, quedando en nuestro caso <http://admin:admin@localhost:8080/crumbIssuer/api/xml>.



The screenshot shows the Postman interface with a 'GET' request to 'http://admin:admin@localhost:8080/crumbIssuer/api/xml'. The response status is '200 OK'. The body of the response contains XML data representing the crumb issuer, specifically the 'defaultCrumbIssuer' element with its 'crumb' value.

```
1 <defaultCrumbIssuer _class='hudson.security.csrf.DefaultCrumbIssuer'>
2   <crumb>ba63fc3b3d43da1ed8b2c6b274556c1e232596c08e90635688e8b90486d8eff5</crumb>
3   <crumbRequestField>Jenkins-Crumb</crumbRequestField>
4 </defaultCrumbIssuer>
```

Ahora ya hemos conseguido obtener la clave-valor de la cabecera que tenemos que enviar con el crumb en la primera petición que habíamos realizado.

Ahora volvemos a realizar la petición POST, añadiendo la cabecera con el crumb como se muestra en la siguiente imagen y además pasandole los datos de autenticación.

The screenshot shows the Postman interface with a successful API call to Jenkins. The 'Headers' tab is selected, showing a Jenkins-Crumb header with a long hex value. The response body is empty.

Una vez que se ha lanzado la petición, ya deberíamos de ver la ejecución del job en el historial de ejecuciones en Jenkins.

Sin crumb, con API Token

En algunos casos las herramientas con las que queremos que interactue Jenkins no nos da la opción de realizar dos peticiones como hemos visto antes para poder ejecutar una tarea. Por ejemplo, este problema nos lo encontramos a la hora de intentar usar un webhook de Github para ejecutar un job cuando se realizan cambios sobre el repositorio.

Para ejecutar el mismo job de antes sin necesidad de tener que pedir el crumb, vamos a necesitar obtener un Token que se enviará junto al usuario como hemos hecho en el ejemplo de antes con el usuario y la contraseña.

Para sacar el token, nos vamos al enlace con el nombre de nuestro usuario.

The screenshot shows the Jenkins user profile page. The 'Configurar' (Configure) link is highlighted with a red box.

Dentro de la página de nuestro usuario, entramos en la sección de **Configurar** donde vamos a añadir un nuevo token.

The screenshot shows the Jenkins 'Configurar' (Configure) page. In the 'Clave del API (Token)' section, a new token 'api-token' has been generated, with a warning message: '⚠️ Copy this token now, because it cannot be recovered in the future.' A 'Copy' button is visible next to the token.

Una vez que tenemos el token generado, en Postman vamos a crear una nueva petición a la url

<http://admin:111c561344bc101ccc727e2bafb80ca9ca@localhost:8080/job/Tarea-Api-Remota/build>

donde hemos sustituido la contraseña que poníamos antes por el token que acabamos de generar. Además, en este caso no hay que mandar ninguna cabecera al igual que hemos hecho antes con la cabecera del crumb.

The screenshot shows the Postman interface with a collection named 'BuildTareaApiRemotaConToken'. A POST request is selected with the URL 'http://admin:111c561344bc101ccc727e2bafb80ca9ca@localhost:8080/job/Tarea-Api-Remota/build'. The 'Params' tab is active, showing a single query parameter 'Key' with value 'Value'. The 'Headers' tab shows the 'Content-Type' header set to 'application/json'. The response status is 201 Created, time 22ms, size 192 B. The response body is a JSON object with a single key '1'.

Basic Authentication con Base64

Al realizar las dos peticiones de antes, hemos tenido que poner los datos de autenticación en la propia URL a la que ibamos a realizar la petición. Esto podemos evitarlo mandando una cabecera **Authorization**. El valor que mandaremos será **usuario:password** o **usuario:token** convertido a **base64** y todo esto precedido de **Basic**.



Podemos usar <https://www.base64encode.net/> para realizar la codificación a base64.

Los dos casos anteriores quedarían de la siguiente forma:

- **Basic YWRtaW46YWRtaW4=**

The screenshot shows the Postman interface with a collection named 'BuildTareaApiRemotaB64'. A POST request is selected with the URL 'http://localhost:8080/job/Tarea-Api-Remota/build'. The 'Headers' tab is active, showing two headers: 'Jenkins-Crumb' with value 'b7830a4997ad9e7dd3429f63bc1647dac6b02b860...', and 'Authorization' with value 'Basic YWRtaW46YWRtaW4='. The 'Body' tab shows the 'Content-Type' header set to 'application/json'. The response status is 201 Created, time 79ms, size 192 B. The response body is a JSON object with a single key '1'.

- Basic YWRtaW46MTEyZU2MTM0NGJjMTAxY2NjNzI3ZTJiYWZiODBjYTljYQ==

The screenshot shows the Postman interface with a successful API call. The request method is POST, the URL is `http://localhost:8080/job/Tarea-Api-Remota/build`, and the response status is 201 Created. The Headers tab shows an Authorization header with the value `Basic YWRtaW46MTEyZU2MTM0NGJjMTAxY2NjNzI3ZTJiYWZiODBjYTljYQ==`. The Body tab shows a JSON response with a single key-value pair: `1`.

Con ambos casos deberíamos de haber podido ejecutar los jobs sin problemas.

10.1.5. Ejecución de tareas con parámetros

Al igual que podemos ejecutar las tareas que no necesitan parámetros, también podemos ejecutar aquellas que si los necesitan.

Para este caso, vamos a crear un job en el que vamos a añadir dos parámetros:

- NOMBRE como parámetro de elección
- APELLIDO como parámetro de cadena

The screenshot shows the Jenkins job configuration page. It displays two parameters: `APELLIDO` (String) and `NOMBRE` (List). The `APELLIDO` parameter has a default value of `Stark`. The `NOMBRE` parameter has options: Robb, Arya, Sansa, Tony, Rickon, and Bran.

Una vez añadidos los parámetros al job, vamos a añadir un comando de shell que muestre los

datos.

The screenshot shows the Jenkins Pipeline Executor interface. At the top, it says "Ejecutar". Below that, there's a section titled "Ejecutar línea de comandos (shell)". Inside this section, there's a text input field containing the command: "echo "Tu \$APELLIDO preferido es \$NOMBRE"". Below the command input, there's a link "Visualizar la lista de variables de entorno disponibles". To the right of the command input, there are "X" and "Avanzado..." buttons. At the bottom left of the main panel, there's a button "Añadir un nuevo paso".

Ahora guardamos y vamos a ver como ejecutar este job parametrizado desde una petición POST.

Para este caso, en lugar de usar `/build` se usará `/buildWithParameters`, y los parámetros se pasarán en el cuerpo de la petición POST.

The screenshot shows the Postman application interface. At the top, it says "POST BuildTareaWithParamsApiRe...". Below that, there's a section titled "BuildTareaWithParamsApiRemotaConTokenB64". On the right side, there are buttons for "Comments (0)", "Examples (0)", "Send", and "Save". The "Headers" tab is selected, showing two headers: "Authorization" (Basic YWRtaW46MTEYzU2MTM0NGjjMTAxY2Nj...) and "Content-Type" (application/x-www-form-urlencoded). Below the headers, there's a "Temporary Headers" section with one entry: "Key" and "Value". At the bottom, there are tabs for "Body", "Cookies (2)", "Headers (5)", and "Test Results". The "Body" tab is selected, showing a "Pretty" view with the number "1".

The screenshot shows the Postman application interface, identical to the previous one but with the "Body" tab selected. The "Body" tab has several options: "none", "form-data" (which is selected), "x-www-form-urlencoded", "raw", "binary", and "GraphQL". Below these options, there are two form fields: "NOMBRE" with value "Tony" and "APELLIDO" with value "Stark". The "Body" tab also includes tabs for "Pretty", "Raw", "Preview", "Visualize", "Text", and "JSON".

A la hora de enviar los valores para los parámetros, tenemos que tener cuidado de enviar datos válidos para que no falle la ejecución. Por ejemplo, si mandamos un valor que no se encuentra en la lista de elecciones como valor del parámetro **NOMBRE**, nos dará un error la petición.

The screenshot shows a Postman interface with the following details:

- Request Method:** POST
- Request URL:** <http://localhost:8080/job/Tarea-Parametrizada-Remote-API/buildWithParameters>
- Body Tab:** Contains form-data parameters:

KEY	VALUE	DESCRIPTION
NOMBRE	Menganito	
APELLIDO	Stark	
Key	Value	Description
- Status:** 500 Server Error
- Stack trace:**

```
java.lang.IllegalArgumentException: Illegal choice for parameter NOMBRE: Menganito
    at hudson.model.ChoiceParameterDefinition.checkValue(ChoiceParameterDefinition.java:138)
    at hudson.model.ChoiceParameterDefinition.createValue(ChoiceParameterDefinition.java:150)
    at hudson.model.ChoiceParameterDefinition.createValue(ChoiceParameterDefinition.java:25)
    at hudson.model.SimpleParameterDefinition.createValue(SimpleParameterDefinition.java:35)
    at hudson.model.ParametersDefinitionProperty.buildWithParameters(ParametersDefinitionProperty.java:182)
```



En el caso de enviar más parámetros de los que se han definido en la tarea, estos no se tendrán en cuenta y se ejecutará el job siempre que el resto de la petición sea correcta.

10.1.6. Desactivar tarea

Algunas veces cuando las tareas empiezan a fallar y son tareas que se ejecutan automáticamente y de forma seguida, podemos desactivarla hasta que se revise el problema para que se deje de ejecutar y en su lugar se vayan ejecutando otras tareas sin necesidad de que se esperen a que esta se termine de ejecutar.

En este caso, al igual que en los anteriores, tenemos que realizar una petición POST a la url del job añadiendo al final **disabled** y alguno de los métodos de autenticación de los vistos anteriormente.

The screenshot shows a Postman interface with the following details:

- Request Method:** POST
- Request URL:** <http://localhost:8080/job/Tarea1/disable>
- Headers Tab:** Contains an Authorization header:

KEY	VALUE	DESCRIPTION
Authorization	Basic YWRtaW46MTEyZU2MTM0NGJjMTAxY2NjN...	
Key	Value	Description

Una vez lanzada la petición, si vamos a la página de la tarea, nos debería de indicar que está desactivada y darnos la opción de volverla a activar.

Proyecto Tarea1



10.1.7. Activar tarea

Y al igual que antes hemos visto como desactivar una tarea, también podemos volver a activarla cambiando de la URL el `disabled` por `enable`.

A screenshot of the Postman application interface. The request URL is "http://localhost:8080/job/Tarea1/enable". The "Headers" tab is selected, showing the following configuration:

KEY	VALUE	DESCRIPTION
Authorization	Basic YWRtaW46MTExYzU2MTM0NGjMTAxY2Nj...	
Key	Value	Description

Ahora si miramos la tarea en Jenkins debería de volverse a poder ejecutar.

Proyecto Tarea1

A screenshot of the Jenkins project page for "Proyecto Tarea1". The status bar at the top indicates "Estado: Activo" (Active). Below the status, there are two links: "Espacio de trabajo" (Workspace) and "Cambios recientes" (Recent changes). On the right side, there are two buttons: "añadir descripción" (Add description) and "Desactivar el Proyecto" (Disable project), which is now grayed out.

Chapter 11. Ejecución Distribuida

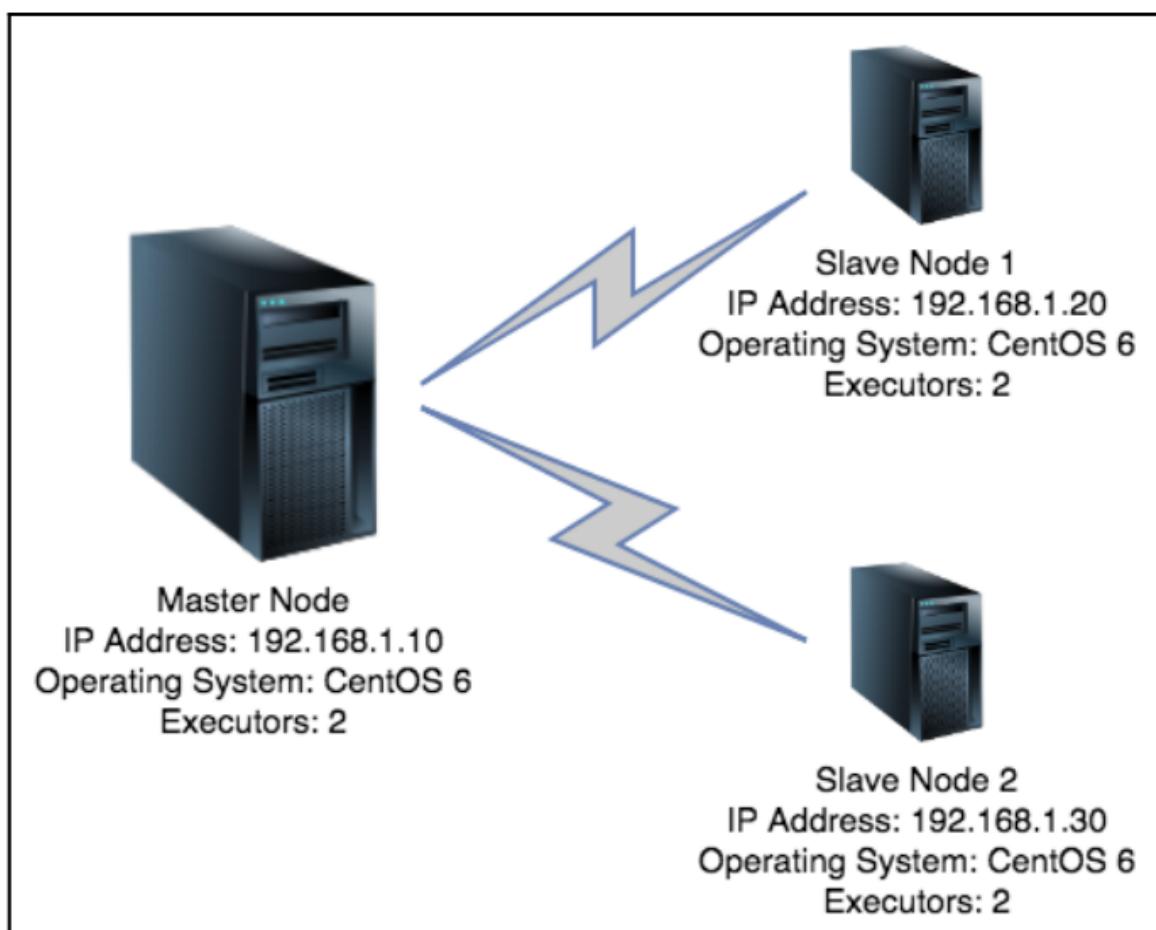
En términos de computación, el término **distribuído** se utiliza para referirse a un sistema compuesto por diferentes componentes que se comunican a través de una red y pasan mensajes e instrucciones a cada uno.

Los diferentes componentes del sistema del sistema se configuran en diferentes servidores dentro de la misma red, esto resulta ideal para conseguir comunicaciones entre sí lo más rápido posible.

En el contexto de Jenkins cuando hablamos de las compilaciones distribuidas, nos estamos refiriendo a la asignación de diferentes nodos (minions) para ejecutar tareas de compilación.

Por otra parte, podemos observar que la creación de más nodos es algo costoso y podría presentar retos en la seguridad, ya que los nodos como hemos comentado, deben comunicarse a través de una red, y estas comunicaciones deben de ser seguras.

11.1. Modelo Master - Minion :)



El modelo de arquitectura **master - minion**, nos permite configurar compilaciones distribuidas en Jenkins.

El nodo máster, es el nodo donde está instalado Jenkins con toda la configuración y ajustes que necesitamos para adaptarlo a nuestras necesidades

Podemos afirmar, que este nodo **master**, es el nodo administrativo que realiza la mayoría de tareas en el sistema de compilación.

Cada nodo en Jenkins tiene una serie de ejecutores por defecto, el nodo maestro tiene 2 ejecutores por defecto

Un ejecutor en Jenkins está definido como "un espacio" para la ejecución de un trabajo, definido por un Pipeline o un proyecto en un nodo determinado



Un nodo puede tener 0 o más ejecutores configurados, este número nos va a indicar cuántos proyectos o secuencias Pipeline concurrentes pueden ejecutarse en ese nodo

Podemos pensar que el concepto de **ejecutor** es un proceso que Jenkins inicia en un nodo para llevar a cabo una tarea

Al configurar nodos **minions**, podremos configurar el número de nodos ejecutores para cada nodo

El número de ejecutores que asignemos a cada nodo, va a estar determinado por las características hardware del nodo en cuestión (RAM, CPU, etc.)

Debemos de tener especial cuidado en la asignación del número de ejecutores disponibles por nodo, ya que podríamos llegar a saturar el nodo si dicho número no es consecuente con los recursos de los que disponemos :D

En ciertas configuraciones, incluso podrían descargar de trabajo al nodo principal, estableciendo el número de ejecutores del **master** a 0, pero este ajuste no es necesariamente recomendado, ya que los tiempos de ejecución de los diferentes trabajos podrían requerir más tiempo si el nodo **master** no se comunica de forma rápida con los **minions** por que la red no esté todo lo fina que debería en cuanto a uso, ancho de banda, etc. :D

11.2. Lab: Ejecución distribuída

Mediante este laboratorio, vamos a ejecutar en la máquina virtual en total 3 instancias de Jenkins, de forma que crearemos proyectos pipelines que podrán ser derivados a que los ejecuten los minions

- 1 Nodo master
- 1 Nodo minion1
- 1 Nodo minion2

11.2.1. Preparando directorios y Ejecutando instancias de Jenkins

Vamos a iniciar 2 nuevas instancias de Jenkins, de forma que operen en puertos diferentes

Nos situaremos dentro de nuestra carpeta `/home/jenkins-ci/software/` y realizaremos 2 copias del .war de jenkins

- Copiamos el archivo y creamos otro con nombre **jenkins-minion1.war**

- Copiamos el archivo y creamos otro con nombre **jenkins-minion2.war**

Primero, vamos a crear las 2 carpetas de expansión del war, para que las dos nuevas instancias de Jenkins no tengan ningún problema al ponerse en marcha de forma independiente:

- Creamos la carpeta **/home/jenkins-ci/software/jenkins_web_root_minion1/**
- Creamos la carpeta **/home/jenkins-ci/software/jenkins_web_root_minion2/**

También vamos a crear 2 carpetas donde se situará el HOME de jenkins para cada instancia

- Creamos la carpeta **/home/jenkins-ci/software/jenkins_home_minion1/**
- Creamos la carpeta **/home/jenkins-ci/software/jenkins_home_minion2/**

Ejecutamos en una terminal la instancia **minion1**:

```
$ export JENKINS_HOME=/home/jenkins-ci/software/jenkins_home_minion1/
$ java -jar jenkins-minion1.war --httpPort=8082 --webroot=/home/jenkins-ci/software/jenkins_web_root_minion1/
```

Ejecutamos en otra terminal la instancia **minion2**:

```
$ export JENKINS_HOME=/home/jenkins-ci/software/jenkins_home_minion2/
$ java -jar jenkins-minion2.war --httpPort=8083 --webroot=/home/jenkins-ci/software/jenkins_web_root_minion2/
```

11.2.2. Inicializando las instancias Jenkins (Minion1 y Minion2)

Una vez las dos instancias hayan arrancado:

- Accedemos a cada una de ellas vía web, poniendo en el navegador <http://localhost:8082> y <http://localhost:8083>
- Obtenemos el token de **initialAdminPassword** de la consola de cada instancia
- Instalamos los plugins sugeridos
- Datos de la cuenta de administración en el nodo minion1
 - Username: minion1
 - Password: minion1
 - Confirm password: minion1
 - Full name: minion1
 - E-mail address: <EMAIL_USER>
- Datos de la cuenta de administración en el nodo minion2
 - Username: minion2
 - Password: minion2
 - Confirm password: minion2

- Full name: minion2
- E-mail address: <EMAIL_USER>

11.2.3. Creando las credenciales de acceso para los nodos minion1 y minion2

Accedemos a la URL del nodo master de Jenkins: <http://localhost:8081>

Desde el dashboard principal, accedemos a la opción **Credentials > System > Global credentials (unrestricted) > Add Credentials**

Indicamos la siguiente configuración para el acceso:

- **Kind**
 - Username with password
- **Scope**
 - Global (Jenkins, nodes, items, all child items, etc)
- **Username**
 - jenkins-ci
- **Password**
 - jenkins-ci
- **Description**
 - Jenkins minions credentials

11.2.4. Añadiendo el nodo minion1 al master

Accedemos a la URL: <http://localhost:8081>

Desde el dashboard principal, accedemos a **Manage Jenkins > Manage Nodes**

Un nodo en Jenkins cuando se pone en marcha por defecto, tiene la etiqueta **master**, hacemos clic en la sección vertical derecha, en la opción de **New Node**

- Indicamos como nombre **minion1**
- Marcamos el check **Permanent Agent**

Pulsamos OK y añadimos la siguiente configuración en el formulario que aparece:

- **Name**
 - minion1
- **Description**
 - Minion node1
- **# of executors**
 - 1

- **Remote root directory**
 - Directorio de trabajos remotos para el nodo en cuestión
 - Este directorio no tiene por qué ser accesible por el nodo master
 - Indicamos: /home/jenkins-ci/software/jenkins_agent_minion1
- **Labels**
 - minion1
- **Usage**
 - Use this node as much as possible
- **Launch Method**
 - Launch agent agents via SSH
- **Launch Method > Host**
 - Introducimos la IP donde se encuentra el nodo minion1
 - 192.168.15.100
- **Launch Method > Credentials**
 - Seleccionamos la que previamente hemos creado
 - jenkins-ci...
- **Launch Method > Host Key Verification Strategy**
 - Non verifying Verification Strategy
- **Availability**
 - Keep this agent online as much as possible

11.2.5. Añadiendo el nodo minion2 al master

Accedemos a la URL: <http://localhost:8081>

Desde el dashboard principal, accedemos a **Manage Jenkins > Manage Nodes**

Un nodo en Jenkins cuando se pone en marcha por defecto, tiene la etiqueta **master**, hacemos clic en la sección vertical derecha, en la opción de **New Node**

- Indicamos como nombre **minion2**
- Marcamos el check **Permanent Agent**

Pulsamos OK y añadimos la siguiente configuración en el formulario que aparece:

- **Name**
 - minion2
- **Description**
 - Minion node2
- **# of executors**

- 1
- **Remote root directory**
 - Directorio de trabajos remotos para el nodo en cuestión
 - Este directorio no tiene por qué ser accesible por el nodo master
 - Indicamos: /home/jenkins-ci/software/jenkins_agent_minion2
- **Labels**
 - minion2
- **Usage**
 - Use this node as much as possible
- **Launch Method**
 - Launch agent agents via SSH
- **Launch Method > Host**
 - Introducimos la IP donde se encuentra el nodo minion1
 - 192.168.15.100
- **Launch Method > Credentials**
 - Seleccionamos la que previamente hemos creado
 - jenkins-ci...
- **Launch Method > Host Key Verification Strategy**
 - Non verifying Verification Strategy
- **Availability**
 - Keep this agent online as much as possible

11.2.6. Comprobando los nodos

Finalmente, accedemos al dashboard principal de Jenkins del nodo master, y **Manage Jenkins > Manage Nodes**

Aquí deberíamos de ver los 3 nodos operativos

11.2.7. Creando proyecto de estilo libre (Ejecución en minion2)

Vamos a crear un proyecto de estilo libre, donde indicaremos un código a ejecutar, pero que deseamos llevar a cabo la ejecución en el nodo **minion2**

Desde el dashboard principal de Jenkins, **New Item > FreeStyle Project**

Indicamos como nombre del proyecto **node-2-project** y pulsamos **Ok**

Una vez dentro del proyecto indicamos los siguientes datos:

- **Description**

- This project will only run on Node2
- **Restrict where this project can be run**
 - Activamos el check
 - Label Expression: **minion2**
 - Mediante la etiqueta, estamos indicando el nombre del nodo donde queremos que se ejecute la tarea

Dentro de la sección de **Build > Add build step > Execute shell**, indicamos el siguiente contenido a ejecutar:

```
#!/bin/bash
sleep 10
hostname
echo "Running on node 2"
```

Pulsamos sobre el botón **Save**

Desde el dashboard principal, accedemos al job **node-2-project** y pulsamos sobre el botón **Build Now**

Observaremos desde el dashboard principal del nodo maestro, que quien está ejecutando la tarea es el nodo minion2

11.2.8. Creando proyecto de pipeline (Ejecución selectiva secuencial en nodos)

Ahora, vamos a crear un proyecto de estilo pipeline, donde crearemos 3 trabajos que deberán de ser ejecutados en nodos distintos

Desde el dashboard principal de Jenkins, **New Item > FreeStyle Project**

Indicamos como nombre del proyecto **node-pipeline** y pulsamos **Ok**

Bajamos a la sección donde indica **Pipeline**

Una vez dentro del proyecto indicamos los siguientes datos:

- **Definition**
 - Pipeline script

Y dentro del bloque de script, indicamos el siguiente pipeline:

```
node {
    stage("This task will execute on master node") {
        printMessage("Running in master node :p")
        sh 'sleep 10'
        sh 'hostname'
```

```

    }

}

node ('minion1') {
    stage("This task will execute on minion1 node") {
        printMessage("Running in minion1 :D")
        sh 'sleep 10'
        sh 'hostname'
    }
}

node ('minion2') {
    stage("This task will execute on minion2 node") {
        printMessage("Running in minion2 ^_^")
        sh 'sleep 10'
        sh 'hostname'
    }
}

def printMessage(message) {
    echo "${message}"
}

```



Si no indicamos en la estructura **node** ninguna referencia a ningún nodo, por defecto, se utilizará el nodo **master**

Ejecutamos el trabajo, y observaremos como se llevará a cabo la ejecución de 3 tareas secuenciales, la primera en el nodo master, cuando finalice esta, la segunda en el nodo minion1, y cuando finalice esta, la tercera en el nodo minion2

11.2.9. Creando proyecto de pipeline (Ejecución selectiva paralela en nodos)

Ahora, vamos a hacer un cambio en el pipeline, de forma que podamos parallelizar de forma simultánea el trabajo en cada uno de los nodos:

```

parallel (
    "Stream WorkFlow1": {
        node {
            stage("This task will execute on master node") {
                printMessage("Running in master node :p")
                sh 'sleep 10'
                sh 'hostname'
            }
        }
    },
    "Stream WorkFlow2": {
        node ('minion1') {
            stage("This task will execute on minion1 node") {
                printMessage("Running in minion1 :D")
            }
        }
    }
)

```

```

        sh 'sleep 10'
        sh 'hostname'
    }
}

},
"Stream WorkFlow3": {
    node ('minion2') {
        stage("This task will execute on minion2 node") {
            printMessage("Running in minion2 ^_^")
            sh 'sleep 10'
            sh 'hostname'
        }
    }
}

def printMessage(message) {
    echo "${message}"
}

```

Ahora, si volvemos a ejecutar el job, observaremos que de forma simultánea, se ejecutan los trabajos en los nodos en lugar de secuencialmente como el caso anterior

Chapter 12. Roles de usuarios

Jenkins es una herramienta que pueden usar bastantes personas de una misma empresa y que tienen distintas responsabilidades dentro un proyecto.

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.

Este enfoque puede resultar interesante en un entorno de pruebas, incluso a nivel interno en una organización con un pequeño equipo donde todos se conocen.

En otro tipo de ámbitos, lo ideal sería poder gestionar una serie de usuarios que dispongan de una serie de permisos, incluso, poder gestionar grupos de usuarios (roles) para hacer más eficiente la gestión de la autenticación.

Es posible que algunas personas solo deban de poder ejecutar unos jobs, y otras personas deban de tener acceso a todos ellos.

Jenkins nos va a permitir asignar roles a estas personas para que según los roles que tengan puedan interactuar con ciertos elementos o no.

Por ejemplo, un desarrollador junior no tendría que poder desplegar una aplicación, pero a lo mejor si que puede ejecutar un job para comprobar que las últimas modificaciones que ha realizado pasan los tests sin necesidad de hacer este proceso de forma manual.

12.1. Tipos de roles

Dentro de la pantalla de roles nos encontramos con 3 secciones:

- **Global roles**
 - Aquí nos encontramos con permisos globales a todos los elementos de Jenkins, es decir que aplican a todas las tareas, la gestión de credenciales, las vistas, los repositorios de código...
- **Item roles**
 - Aquí pondremos permisos a unos elementos de Jenkins muy específicos (como las tareas, los repositorios de código...) y cuyos nombres cumplan un patrón dado al rol.
- **Node roles**
 - Igual que el anterior pero aplica a los elementos que son nodos, credenciales y recursos bloqueables (impresoras, dispositivos móviles, pcs...) de Jenkins.

12.2. Lab: Roles de usuarios

Mediante este laboratorio, vamos a activar una administración de usuarios avanzada, basadas en roles.

12.2.1. Instalar el plugin

Jenkins tiene un plugin que nos va a permitir añadir roles a los usuarios para limitar las acciones que van a poder realizar.

Primero de todo tenemos que instalar el plugin **Role-based Authorization Strategy**.

Desde el menú principal de Jenkins, accedemos a **Manage Jenkins > Manage Plugins > Pestaña Available > Buscamos el plugin: Role-based Authorization Strategy**

Marcamos el check de install y pulsamos sobre **Download now and install after restart**

Empezamos buscandolo en el gestor de plugins, y lo instalamos.

Refrescamos la página una vez que el plugin se haya terminado de instalar.

Filtrar: <input type="text" value="role-based"/>		
	Nombre	Versión
Instalar ↓	Role-based Authorization Strategy <input checked="" type="checkbox"/> Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions.	2.16

12.2.2. Planteando el escenario

Vamos a plantear que tenemos 2 tipos de roles en el equipo:

- Desarrollo
- SQA

Nuestro objetivo sería crear:

- 1 rol para desarrollo
- 1 rol para SQA
- 1 usuario para desarrollo
- 1 usuario para SQA
- 1 vista para desarrollo
 - 1 Job destinado al uso sólo en desarrollo
- 1 vista para SQA
 - 1 Job destinado al uso sólo en SQA
- Establecer reglas de visibilidad de las tareas

12.2.3. Activando autentificación basada en roles

Desde el menú principal de Jenkins, accedemos y seleccionamos **Manage Jenkins > Configure Global Security > Authorization > Role-Based Strategy**

Configuración global de la seguridad

Activar seguridad Disable remember me 

Control de acceso

Seguridad

- Delegar seguridad al contenedor de servlets 
- LDAP 
- Unix user/group database 
- 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' 
- Role-Based Strategy 
- Usuarios autenticados tienen privilegios para todo 

12.2.4. Creando el Job destinado a desarrollo

Desde el menú principal de Jenkins, accedemos a **Jenkins > New Item > Freestyle project**, indicamos como nombre del job **development-app1**

Dentro del Job, nos dirigimos a la sección de **Build > Add build step > Execute shell**

Indicamos como contenido del Job lo siguiente:

```
echo "Executed Development Job!"
```

12.2.5. Creando el Job destinado a sqa

Desde el menú principal de Jenkins, accedemos a **Jenkins > New Item > Freestyle project**, indicamos como nombre del job **sqa-app1**

Dentro del Job, nos dirigimos a la sección de **Build > Add build step > Execute shell**

Indicamos como contenido del Job lo siguiente:

```
echo "Executed SQA Job!"
```

12.2.6. Creando el usuario para desarrollo

Desde el menú principal, accedemos a **Manage Jenkins > Manage Users > Create User**

Indicamos únicamente los siguientes datos:

- Username
 - development
- Password

- development
- Confirm password
 - development
- Full name
 - development
- E-mail address
 - <tua_correo@electronico.com>

12.2.7. Creando el usuario para sqa

Desde el menú principal, accedemos a **Manage Jenkins > Manage Users > Create User**

Indicamos únicamente los siguientes datos:

- Username
 - sqa
- Password
 - sqa
- Confirm password
 - sqa
- Full name
 - sqa
- E-mail address
 - <tua_correo@electronico.com>

12.2.8. Creando el rol global para desarrollo

Desde el menú principal, accedemos a **Jenkins > Manage Jenkins > Manage and Assign Roles > Manage Roles**

Vamos a la sección que tiene por título **Global roles**, desde aquí vamos a añadir un nuevo rol.

Indicamos como **Role to add**: development, pulsamos en el botón **Add**

A continuación, indicamos para el rol **development** el check de lectura global, marcando la columna **Read** de la sección **Overall**

12.2.9. Estableciendo permisos al rol desarrollo

Desde el menú principal, accedemos a **Jenkins > Manage Jenkins > Manage and Assign Roles > Manage Roles**

Vamos a la sección que tiene por título **Item roles**, desde aquí vamos a añadir los diferentes roles que afectarán a los items (jobs).

La idea, es que, podamos indicar un "rol-item" que afecte a una serie de Jobs, con una serie de permisos.

Indicamos los siguientes datos:

- **Role to add**
 - learning-development
- **Pattern**
 - ^development.\$

Mediante la expresión regular, estaríamos indicando que sólo deseamos que los items que comiencen por la palabra development, estén sujetos a las reglas del rol.

Pulsamos sobre el botón **Add**

Marcamos los siguientes checks de columnas:

- **Credentials > View**
- **Job > Build**
- **Job > Cancel**
- **Job > Configure**
- **Job > Discover**
- **Job > Read**
- **Job > Workspace**
- **Run > Replay**
- **SCM > Tag**

Pulsamos sobre el botón **Save**

12.2.10. Creando el rol global para sqa

Desde el menú principal, accedemos a **Jenkins > Manage Jenkins > Manage and Assign Roles > Manage Roles**

Vamos a la sección que tiene por título **Global roles**, desde aquí vamos a añadir un nuevo rol.

Indicamos como **Role to add**: sqa, pulsamos en el botón **Add**

A continuación, indicamos para el rol **sqa** el check de lectura global, marcando la columna **Read** de la sección **Overall**

12.2.11. Estableciendo permisos al rol sqa

Desde el menú principal, accedemos a **Jenkins > Manage Jenkins > Manage and Assign Roles > Manage Roles**

Vamos a la sección que tiene por título **Item roles**, desde aquí vamos a añadir los diferentes roles que afectarán a los items (jobs).

La idea, es que, podamos indicar un "rol-item" que afecte a una serie de Jobs, con una serie de permisos.

Indicamos los siguientes datos:

- **Role to add**
 - learning-sqa
- **Pattern**
 - ^sqa.\$

Mediante la expresión regular, estaríamos indicando que sólo deseamos que los items que comiencen por la palabra development, estén sujetos a las reglas del rol.

Pulsamos sobre el botón **Add**

Marcamos los siguientes checks de columnas:

- **Credentials > View**
- **Job > Build**
- **Job > Cancel**
- **Job > Configure**
- **Job > Discover**
- **Job > Read**
- **Job > Workspace**
- **Run > Replay**
- **SCM > Tag**

Pulsamos sobre el botón **Save**

12.2.12. Asignando roles globales a usuarios (development y sqa)

A continuación, vamos a asignar los roles que hemos creado, a nivel global, para que los usuarios puedan tener acceso de lectura al propio jenkins, en cuanto al menú de opciones inicial.

Desde el menú principal de Jenkins, accedemos a **Jenkins > Manage Jenkins > Manage and Assign Roles > Assign Roles > Global roles**

Indicamos:

- **User/group to add**
 - development

Pulsamos **Add**

Marcamos el check de la columna **development** para el usuario con nombre **development**

- **User/group to add**

- sqa

Pulsamos **Add**

Marcamos el check de la columna **sqa** para el usuario con nombre **sqa**

Pulsamos finalmente el botón de **Save**

12.2.13. Asignando item roles a usuarios (development y sqa)

El siguiente paso, sería asignar los roles que hemos creado, a los usuarios que tenemos registrados en el propio jenkins.

Desde el menú principal de Jenkins, accedemos a **Jenkins > Manage Jenkins > Manage and Assign Roles > Assign Roles > Sección Item Roles**

En primer lugar, indicamos:

- **User/group to add**

- development

Marcamos el check de la columna **learning-development** para el usuario con nombre **development**

Pulsamos **Add**

- **User/group to add**

- sqa

Marcamos el check de la columna **learning-sqa** para el usuario con nombre **sqa**

Pulsamos **Add**

Pulsamos finalmente el botón de **Save**

12.2.14. Comprobando los accesos

Vamos a probar a acceder a jenkins con el usuario **development** y luego, con el usuario **sqa**.

Deberíamos de observar:

- Cuando accedemos con el usuario **development**, únicamente vamos a visualizar jobs, cuyo nombre comience por la palabra development
- Cuando accedemos con el usuario **sqa**, únicamente vamos a visualizar jobs, cuyo nombre comience por la palabra sqa

Los permisos que hemos otorgado a ambos roles, son de sólo lectura en cuanto a la creación de

jobs, únicamente los usuarios podrían ejecutar los jobs, pero no podrían eliminarlos ni crear nuevos, delegando esa operación al administrador de Jenkins.

Chapter 13. Notificaciones

En este tema vamos a ver como notificar a las personas que trabajan en un proyecto que las tareas que se están ejecutando con Jenkins se han ejecutado de forma correcta o han fallado. De esta forma aquellas personas involucradas en un proyecto podrán conocer en todo momento el estado de este, por ejemplo, que se han generado los archivos finales, se han pasado los tests, se ha desplegado, o alguno de estos casos ha fallado.

Para ello veremos como notificar a través de:

- Email con el plugin *Mailer*
- Slack con el plugin *Slack Notification*

13.1. Lab: Notificaciones mediante e-mail

En este laboratorio vamos a ver como hacer para notificar a una persona o varias mediante un email de que un job se ha ejecutado correctamente o de que ha fallado su ejecución.

13.1.1. Plugin Mailer

Para empezar, vamos a necesitar instalar el plugin **Mailer Plugin**.

Activados	Nombre ↓	Versión	Versión previamente instalada.	Desinstalar
Display URL API	Provides the DisplayURLProvider extension point to provide alternate URLs for use in notifications	2.3.2		Desinstalar
Email Extension Plugin	This plugin is a replacement for Jenkins's email publisher. It allows to configure every aspect of email notifications: when an email is sent, who should receive it and what the email says	2.68		Desinstalar
Git plugin	This plugin integrates Git with Jenkins.	4.0.0		Desinstalar
JUnit Plugin	Allows JUnit-format test results to be published.	1.28		Desinstalar
LDAP Plugin	Adds LDAP authentication to Jenkins	1.21		Desinstalar
Lockable Resources plugin	This plugin allows to define external resources (such as printers, phones, computers) that can be locked by builds. If a build requires an external resource which is already locked, it will wait for the resource to be free.	2.7		Desinstalar
Mailer Plugin	This plugin allows you to configure email notifications for build results	1.29		Desinstalar
Matrix Project Plugin	Multi-configuration (matrix) project type.	1.14		Desinstalar
Pipeline: Basic Steps	Commonly used steps for Pipelines.	2.19		Desinstalar
Pipeline: Declarative	An opinionated, declarative Pipeline.	1.5.0		Desinstalar
Script Security Plugin	Allows Jenkins administrators to control what in-process scripts can be run by less-privileged users.	1.68		Desinstalar

Una vez que se ha instalado, tenemos que configurar el plugin para indicarle con que correo tiene que mandar los emails, que servidor de correos se va a usar... Para ello tenemos que ir a la ventana de **Jenkins > Configuración global** y dentro deberíamos de ver una sección **Notificación por correo electrónico** donde tendremos que llenar unos campos.

Si no sabemos como llenar dichos campos, Google nos ayudará.

¿Cómo usar SMTP de Gmail?

^

Vea nuestro tutorial sobre cómo enviar emails desde dentro de WordPress.

1. Servidor (SMTP) de Correo Saliente: smtp.gmail.com.
2. Usar Autenticación: Si
3. Usar Conexión Segura: Sí (puede TLS o SSL dependiendo de su cliente de email)
4. Nombre de usuario: cuenta GMail (email@gmail.com)
5. Contraseña: contraseña GMail.
6. Puerto: 465 o 587.

3 jun. 2019

 kinsta.com › base-de-conocimiento › servidor-gratuito-smtp

Cómo Utilizar el Servidor SMTP Gratuito de Google Para Enviar ...

Buscar: ¿Cómo usar SMTP de Gmail?

Notificación por correo electrónico

Servidor de correo saliente (SMTP)	<input type="text" value="smtp.gmail.com"/>
Sufijo de email por defecto	<input type="text"/>
<input checked="" type="checkbox"/> Use SMTP Authentication	<input type="checkbox"/>
Nombre de usuario	<input type="text" value="████████@gmail.com"/>
Contraseña	<input type="password" value="Concealed"/> Change Password
Usar seguridad SSL	<input checked="" type="checkbox"/>
Puerto de SMTP	<input type="text" value="465"/>
Dirección para la respuesta	<input type="text"/>
Juego de caracteres	<input type="text" value="UTF-8"/>

Probar la configuración enviando un correo de prueba

Configuramos con nuestro servidor SMTP gmail:

- **SMTP server**
 - smtp.gmail.com
- **User SMTP Authentication**
 - Marcamos el check
- **User Name**
 - Tu correo electrónico
- **Password**
 - Tu clave de acceso al correo electrónico
- **Use SSL**

- Marcamos el check

• SMTP Port

- 465

• Charset

- UTF-8

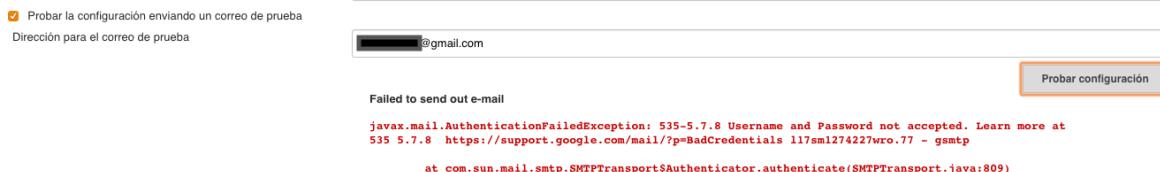


Si tenemos en nuestra máquina anfitriona algún tipo de firewall, antivirus, tipo Avast, Sophos, etc. Podemos experimentar fallos al intentar enviar el correo de prueba, con errores de conexión SSL, pero que en realidad, se trata de cortes que está realizando el propio firewall/antivirus.

Si experimentamos ese tipo de errores, temporalmente desconectamos dicho antivirus/firewall para que no nos de problemas.

Una vez que tenemos rellenados los campos como se muestran en la imagen anterior, podemos probar a enviar un correo de prueba a alguna dirección marcando la opción *Probar la configuración enviando un correo de prueba* y rellenando el campo con la dirección en la que queremos recibir dicho correo.

Cuando le damos al botón de probar la configuración debería de llegarnos un email a la dirección que se ha indicado. Pero en su lugar nos aparece un error debajo del botón indicando que las credenciales introducidas no se han aceptado.



Este error se debe a que no tenemos activada la opción de **Acceso de aplicaciones poco seguras** de Gmail que habilita que cualquier aplicación pueda autenticarse con unas credenciales válidas para usar los servicios de Google.

Podemos activar esta opción en la siguiente página: <https://myaccount.google.com/lesssecureapps>.

Google Cuenta

Buscar en la cuenta de Google

Inicio

Información personal

Datos y personalización

Seguridad (selected)

Contactos y información compartida

Pagos y suscripciones

Gestionar dispositivos

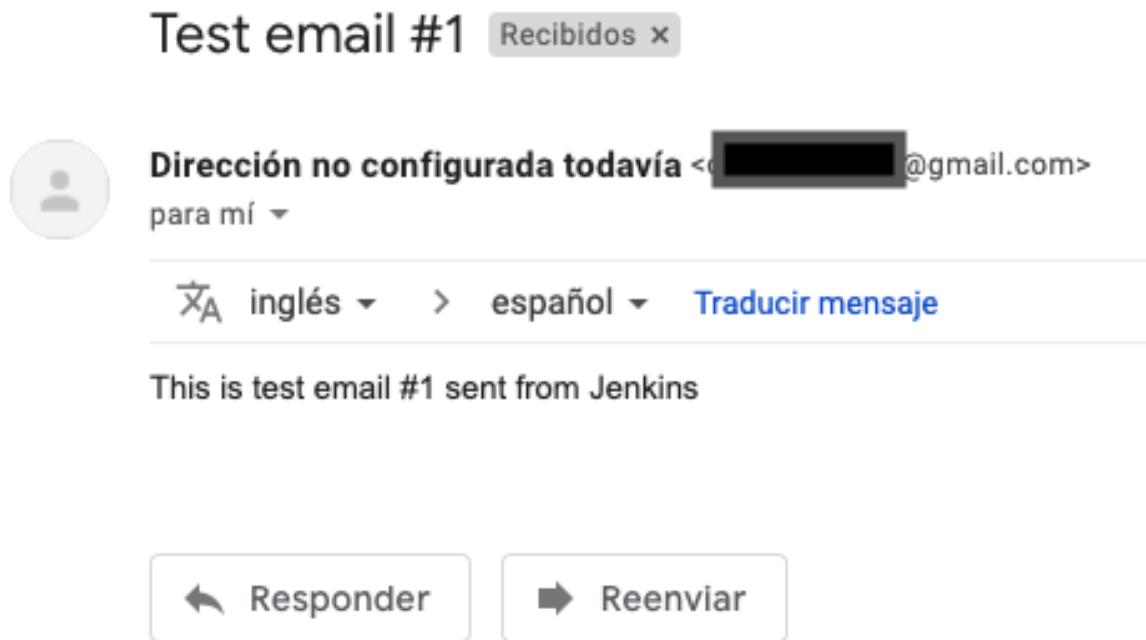
Acceso de aplicaciones poco seguras

Tu cuenta es vulnerable porque permite el acceso de aplicaciones y dispositivos que utilizan una tecnología de inicio de sesión poco segura. Para mantener tu cuenta protegida, Google desactivará automáticamente este ajuste si no se utiliza. [Más información](#)

Activado

[Desactivar acceso \(opción recomendada\)](#)

Una vez activada dicha opción, probamos a enviar otro email, y esta vez tiene que poder enviarlo correctamente.



13.1.2. Usar el plugin con un job de Jenkins

Una vez que ya podemos enviar emails con Jenkins, vamos a crearnos una tarea de Jenkins que se va a encargar de realizar algunas acciones, y si en algún momento falla la tarea, queremos que nos mande un correo para avisarnos de que algo no está funcionando bien.

Para crear el Job, desde el panel principal de Jenkins, seleccionamos la opción **New Item** e indicamos las siguientes opciones:

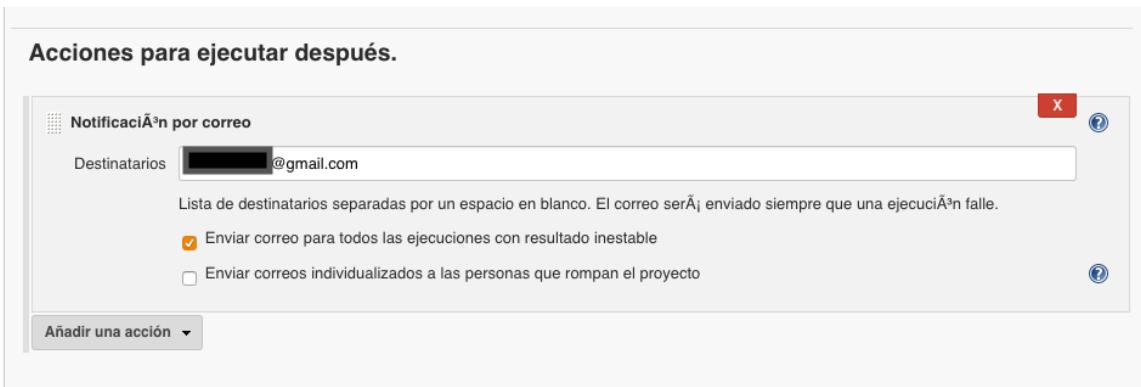
- Enter an item name
 - test-send-email-by-job
- Tipo de proyecto
 - Freestyle project

Añadimos al job los comandos que tenga que ejecutar.



Y ahora vamos a añadir una **Acción para ejecutar después** que va a ser una *Notificación por correo*, donde vamos a indicar los destinatarios a los que queremos mandar el email (separandolos

por espacios en blanco).



Una vez añadido, guardamos el job y si lo ejecutamos debería de salir todo correcto, por lo tanto no se enviará ningún email.

Pero ahora vamos a volver al job para introducir algún error en su ejecución, y vamos a añadir otro comando que no existe para provocar el error de la tarea.



Guardamos la tarea, y al ejecutarla de nuevo y fallar dicha ejecución, recibiremos un email con la información sobre dicha ejecución.

Build failed in Jenkins: notificacion-email #2 Recibidos x



Dirección no configurada todavía <████████@gmail.com>

para mí ▾

✗ inglés ▾ > español ▾ Traducir mensaje

See <<http://localhost:8080/job/notificacion-email/2/display/redirect>>

Changes:

```
-----  
Started by user admin  
Running as SYSTEM  
Building in workspace <http://localhost:8080/job/notificacion-email/ws/>  
[notificacion-email] $ /bin/sh -xe /tmp/jenkins4263595306781477739.sh  
+ echo Funciona bien!  
Funciona bien!  
+ ech funciona mal  
/tmp/jenkins4263595306781477739.sh: 3: /tmp/jenkins4263595306781477739.sh: ech: not found  
Build step 'Execute shell' marked build as failure
```

◀ Responder

➡ Reenviar

Una vez que recibimos el mensaje, investigamos el motivo por el cual ha fallado la ejecución de la tarea y la arreglamos, si volvemos a ejecutar el job, y esta vez si que se ejecuta correctamente, se nos volverá a enviar otro correo indicando que la tarea a vuelto a la normalidad y ya se ejecuta correctamente de nuevo.

Jenkins build is back to normal : notificacion-email #3 Recibidos x



Dirección no configurada todavía <████████@gmail.com>

para mí ▾

✗ inglés ▾ > español ▾ Traducir mensaje

See <<http://localhost:8080/job/notificacion-email/3/display/redirect>>

◀ Responder

➡ Reenviar