



# Certified Tech Developer

The Ultimate Degree

## Infraestructura II

Actividad obligatoria e individual

Dificultad: media

# Continuous Integration con Jenkinsfile - Parte 2

Continuamos la práctica de la clase anterior. En esta clase vamos a configurar un pipeline en base al Jenkinsfile subido a nuestro repositorio y reemplazar el bloque de build con el comando de maven que corresponde.

## Pipelines

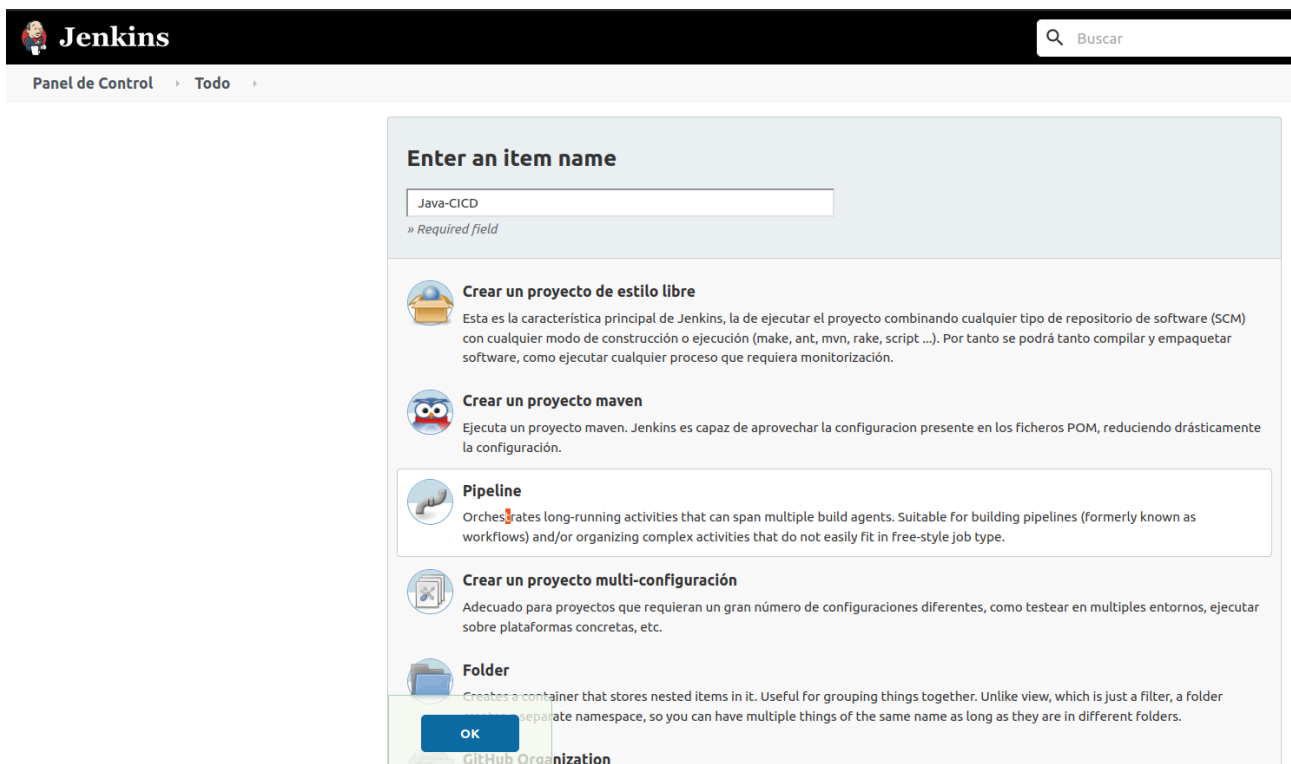
Los pipelines en Jenkins proporcionan un conjunto de herramientas que son flexibles a través de código. Esto nos permite modelar una implementación de la metodología DevOps muy simple o para una organización grande y compleja.

En esta práctica les proponemos crear un pipeline desde el dashboard de Jenkins usando el Jenkinsfile tal como lo tenemos en nuestro repositorio de código. **¡Manos a la obra!**

**En la siguiente página se encuentra la resolución. Continúa únicamente para realizar una autoevaluación.**

## Resolución

Dentro de la página principal de Jenkins vamos a crear un pipeline desde “Nueva Tarea”. En nuestro caso vamos a seleccionar “pipeline” y le colocamos un nombre para identificarlo.



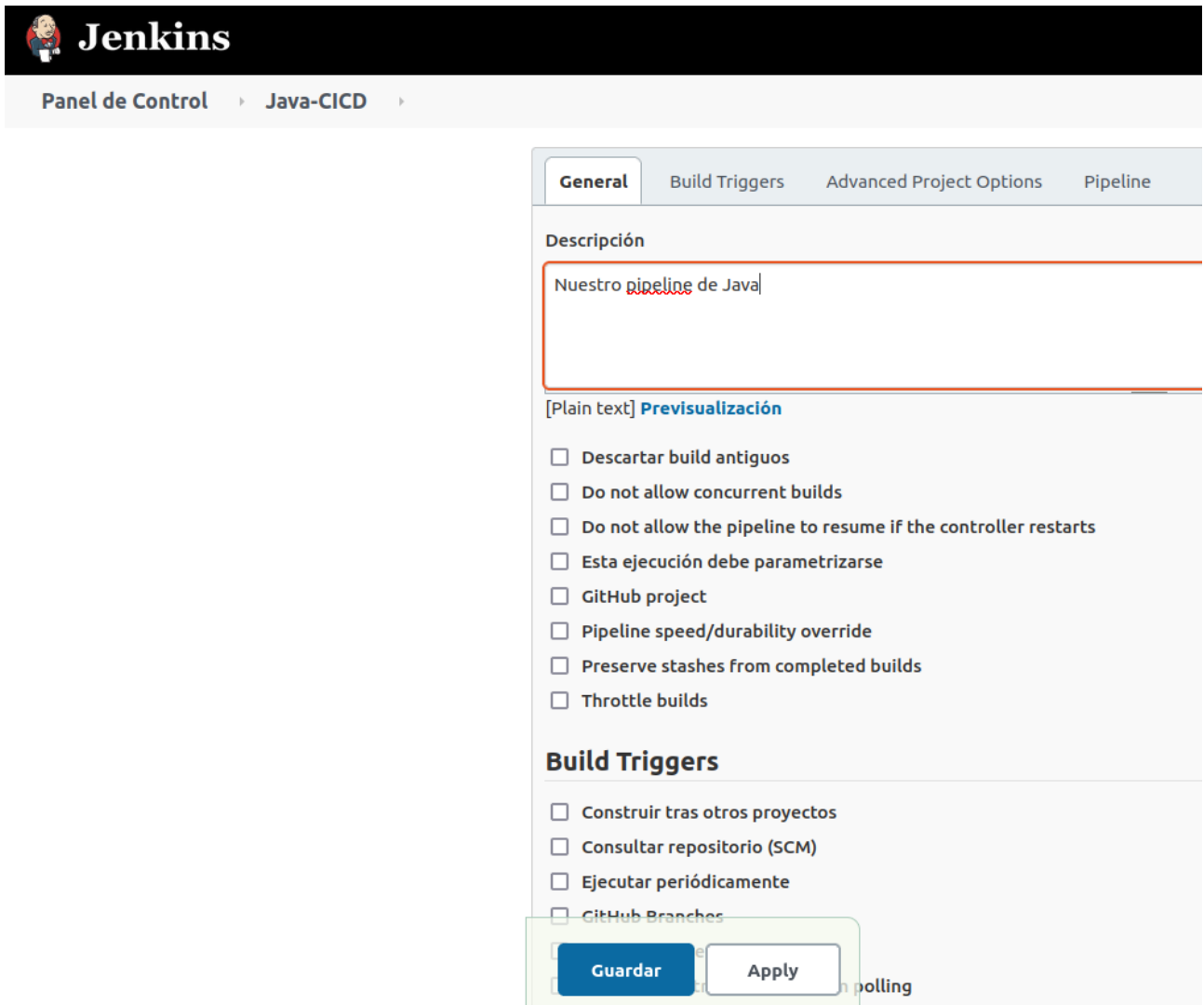
The screenshot shows the Jenkins 'New Item' dialog box. At the top, there's a search bar with the text 'Buscar'. Below it, the 'Enter an item name' section has a text input field containing 'Java-CICD' and a 'Required field' message. Below this, there are five options with icons and descriptions:

- Crear un proyecto de estilo libre**: Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.
- Crear un proyecto maven**: Ejecuta un proyecto maven. Jenkins es capaz de aprovechar la configuración presente en los ficheros POM, reduciendo drásticamente la configuración.
- 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.
- Crear un proyecto multi-configuración**: Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en multiples entornos, ejecutar sobre plataformas concretas, etc.
- 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.

At the bottom, there is a blue 'OK' button and a 'GitHub Organization' link.

Vamos a llamarlo “JavaCICD”. Este pipeline nos acompañará a lo largo de varias clases y lo iremos complejizando cada vez más, hasta completar un ciclo DevOps apto para implementarse en cualquier empresa.

En la descripción vamos a colocar un breve comentario sobre lo que hace nuestro pipeline. Esto nos permite aportar a la documentación de nuestro trabajo.



The screenshot shows the Jenkins web interface. At the top, there's a black header with the Jenkins logo and name. Below it, a breadcrumb trail shows 'Panel de Control' and 'Java-CICD'. The main content area has four tabs: 'General' (selected), 'Build Triggers', 'Advanced Project Options', and 'Pipeline'. Under the 'General' tab, there's a 'Descripción' section with a text area containing 'Nuestro pipeline de Java'. Below this is a 'Previsualización' section with a list of checkboxes for various build options. At the bottom, there are 'Guardar' and 'Apply' buttons. A green box highlights the 'Guardar' button and the 'Apply' button, with a 'polling' label next to it.

**Jenkins**

Panel de Control > Java-CICD >

General Build Triggers Advanced Project Options Pipeline

Descripción

Nuestro pipeline de Java

[Plain text] Previsualización

- ☐ Descartar build antiguos
- ☐ Do not allow concurrent builds
- ☐ Do not allow the pipeline to resume if the controller restarts
- ☐ Esta ejecución debe parametrizarse
- ☐ GitHub project
- ☐ Pipeline speed/durability override
- ☐ Preserve stashes from completed builds
- ☐ Throttle builds

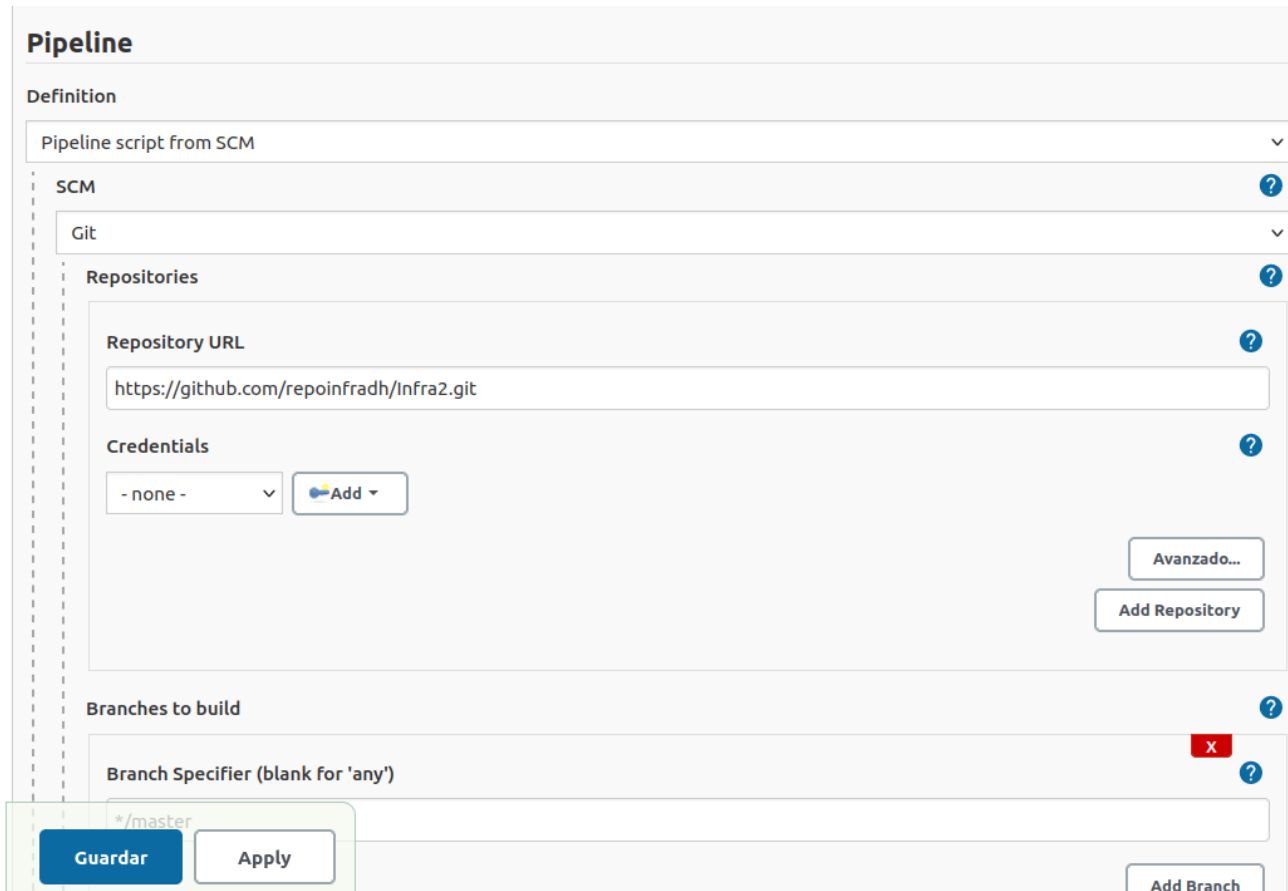
**Build Triggers**

- ☐ Construir tras otros proyectos
- ☐ Consultar repositorio (SCM)
- ☐ Ejecutar periódicamente
- ☐ GitHub Branches

Guardar Apply polling

Salteamos toda la configuración y vamos directo a la configuración del pipeline. Elegimos "Pipeline script from SCM". Esto quiere decir que obtenemos el Jenkinsfile de un versionado de código. En SCM tenemos que elegir GIT (esto es válido para cualquier cliente que use esta tecnología, como GitHub, Gitlab o Bitbucket).

Pegamos la ruta de nuestro repositorio (es la misma ruta que cuando clonamos el repositorio a través del protocolo HTTPS). En caso de que nuestro repositorio sea privado, vamos a seleccionar las credenciales que deben estar ya precargadas (tal como hicimos en la clase anterior). Tengamos en cuenta que en esta etapa no se pueden guardar credenciales. Si no lo hicimos y lo necesitamos, tenemos que guardar e ir a la administración de Jenkins a guardar las credenciales.



The screenshot shows the Jenkins Pipeline configuration page. The 'Definition' section is set to 'Pipeline script from SCM'. The 'SCM' is set to 'Git'. Under 'Repositories', the 'Repository URL' is 'https://github.com/repoinfradh/Infra2.git'. The 'Credentials' dropdown is set to '- none -' with an 'Add' button. There are 'Avanzado...' and 'Add Repository' buttons. The 'Branches to build' section has a 'Branch Specifier (blank for 'any')' field with '\*/master' entered. There are 'Guardar', 'Apply', and 'Add Branch' buttons. A red 'X' icon is visible next to the branch specifier field.

Por último, vamos a especificar la rama en “Branch Specifier” desde la que queremos obtener el código y el Jenkinsfile. Esto es muy útil si queremos crear un pipeline en paralelo solo para realizar pruebas de nuestra automatización sin afectar el pipeline que realmente usan los desarrolladores (el pipeline productivo).

Dentro de “script path” vamos a colocar la ruta donde se encuentra el Jenkinsfile. El ejemplo del repositorio de la materia aplica, ya que está dentro de una sub carpeta. Si nos encontramos en un repositorio donde el Jenkinsfile está en la raíz, únicamente colocamos el nombre del archivo. Recordemos que el hecho de que se llame Jenkinsfile es solo una convención y una buena práctica (es importante, pero no obligatorio, por lo que podemos tener varios Jenkinsfile con distintos nombres en nuestro repositorio o en la misma carpeta).



General

Build Triggers

Advanced Project Options

Pipeline

Branches to build

Branch Specifier (blank for 'any')

\*/main

Add Branch

Navegador del repositorio

(Auto)

Additional Behaviours

Añadir

Script Path

practica-pipelines/Jenkinsfile

☒ Lightweight checkout

Pipeline Syntax

Guardar

Apply

**Ya guardamos nuestro pipeline.** Ahora lo podemos visualizar correctamente configurado pero sin ejecuciones. Hacemos clic en “Construir ahora”. Comenzará a correr. Podemos visualizarlo en el “historial de tareas”.

Actividades | Navegador web Firefox | 29 de ago 20:21

Java-CICD [Jenkins] | localhost:8080/job/Java-CICD/

## Jenkins

Panel de Control > Java-CICD >

- Back to Dashboard
- Status
- Changes
- Construir ahora**
- Configurar
- Eliminar Pipeline
- Full Stage View
- Rename
- Pipeline Syntax

### Pipeline Java-CICD

Nuestro pipeline de Java

Recent Changes

### Stage View

Average stage times:  
(Average full run time: ~15s)

Declarative: Checkout SCM	Build	Test	Deploy
1s	319ms	135ms	143ms

#1 Aug 29 20:20 No Changes

### Accesos directos

- Atom feed para todos
- Atom feed para fallos

Cliqueamos en la ejecución, en nuestro caso es el número "1". Vamos a poder visualizar la salida de nuestro pipeline en el apartado "Console Output".

**¿Qué datos observamos en este log? Leé línea por línea para conocer qué hacen los plugins al ejecutar dentro de Jenkins.**



Panel de Control > Java-CICD > #1

Back to Project

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#1'

Git Build Data

Restart from Stage

Replay

Pipeline Steps

Workspaces

Salida de consola

Lanzada por el usuario **Profe DH**

Obtained practica-pipelines/Jenkinsfile from git <https://github.com/repoinfradh/Infra2.git>

Running in Durability level: MAX\_SURVIVABILITY

[Pipeline] Start of Pipeline

[Pipeline] node

Running on **Jenkins** in /var/lib/jenkins/workspace/Java-CICD

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Declarative: Checkout SCM)

[Pipeline] checkout

Selected Git installation does not exist. Using Default

The recommended git tool is: NONE

No credentials specified

Cloning the remote Git repository

Cloning repository <https://github.com/repoinfradh/Infra2.git>

> git init /var/lib/jenkins/workspace/Java-CICD # timeout=10

Fetching upstream changes from <https://github.com/repoinfradh/Infra2.git>

> git --version # timeout=10

> git --version # 'git version 2.27.0'

> git fetch --tags --force --progress -- <https://github.com/repoinfradh/Infra2.git> +refs/heads/\*:refs/remotes/origin/\* # timeout=10

> git config remote.origin.url <https://github.com/repoinfradh/Infra2.git> # timeout=10

> git config --add remote.origin.fetch +refs/heads/\*:refs/remotes/origin/\* # timeout=10

Avoid second fetch

> git rev-parse refs/remotes/origin/main^{commit} # timeout=10

Checking out Revision 7fe372e6a8436c46453da3b6953d32649a03018d (refs/remotes/origin/main)

> git config core.sparsecheckout # timeout=10

> git checkout -f 7fe372e6a8436c46453da3b6953d32649a03018d # timeout=10

Commit message: "Update README.md"

First time build. Skipping changelog.

[Pipeline] }

[Pipeline] // stage

[Pipeline] withEnv

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Build)

[Pipeline] echo

Building..

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Test)

[Pipeline] echo

Testing..

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Deploy)

[Pipeline] echo

Deploying....

[Pipeline] }

[Pipeline] // stage

[Pipeline] }

[Pipeline] // withEnv

[Pipeline] }

[Pipeline] // node

[Pipeline] End of Pipeline

Finished: SUCCESS

**En la próxima clase vamos a modificar nuestro pipeline dentro del bloque “build” para utilizar maven y compilar una aplicación Java.**