

Pipelines en Jenkins

Neftalí Rodríguez Rodríguez



[Github](#)

Indice

Pipelines en Jenkins.....	1
2. Parte 1.....	2
2. Parte 2.....	3
3. Parte 3.....	6
4. Parte 4.....	7
5. Parte 5.....	8
6. Parte 6.....	9
7. Parte 7.....	10
8. Parte 8.....	11

2. Parte 1

Antes de empezar a crear nuevas tareas en Jenkins nos aseguraremos de tener los plugins necesarios instalados. Para ello nos dirigiremos a “**Administrar Jenkins**”, luego a “**Administrar plugins**” y en dicha pantalla buscaremos “**docker**” en el cuadro de búsqueda. Necesitaremos los plugins “**docker plugin**” y “**docker pipeline**”. Marcamos ambos y pulsamos en instalar.

Instalando/Actualizando plugins

Preparación

- Probando conectividad con Internet
- Probando conectividad con jenkins-ci.org
- Correcto

Authentication Tokens API	⚠ Descarga correcta. Se activará en el próximo arranque.
Docker Commons	⚠ Descarga correcta. Se activará en el próximo arranque.
Docker API	⚠ Descarga correcta. Se activará en el próximo arranque.
Docker	⚠ Descarga correcta. Se activará en el próximo arranque.
Docker Pipeline	⋮ Pendiente
Reiniciando Jenkins	⋮ Pendiente

➡ [Volver al inicio de la página](#)

(puedes empezar a usar los plugins instalados inmediatamente)

➡ ☒ Reiniciar Jenkins cuando termine la instalación y no queden trabajos en ejecución

Marcaremos la opción de que se reinicie Jenkins al acabar la instalación y esperamos a que se instalen los plugins

2. Parte 2


Accedemos al panel de control de Jenkins y pulsamos sobre “**Nueva tarea**”. Nos llevará a la siguiente pantalla en la que le pondremos un nombre a la tarea y luego pulsamos sobre la opción “**Pipeline**”

Panel de Control ▶ **Todo** ▶

Enter an item name


Pipeline_Java_HelloWorld

» *Required field*



Crear un proyecto de estilo libre

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



Pipeline

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

En la siguiente ventana que nos aparecerá a continuación, nos desplazamos al final de esta y pegamos el script que vamos a usar dentro del cuadro de texto “Pipeline script”

Pipeline

Definition

Pipeline script

Script

```
1 pipeline {  
2   agent { docker { image 'maven:3.8.4-openjdk-11-slim' } }  
3   stages {  
4     stage('build') {  
5       steps {  
6         sh 'mvn --version'  
7       }  
8     }  
9   }  
10 }
```

try sample Pipeline...

Una vez se haya creado la tarea pulsamos sobre **“Construir ahora”** y esperamos a que se ejecute la tarea.

Panel de Control ▶ Pipeline_NodeJS ▶

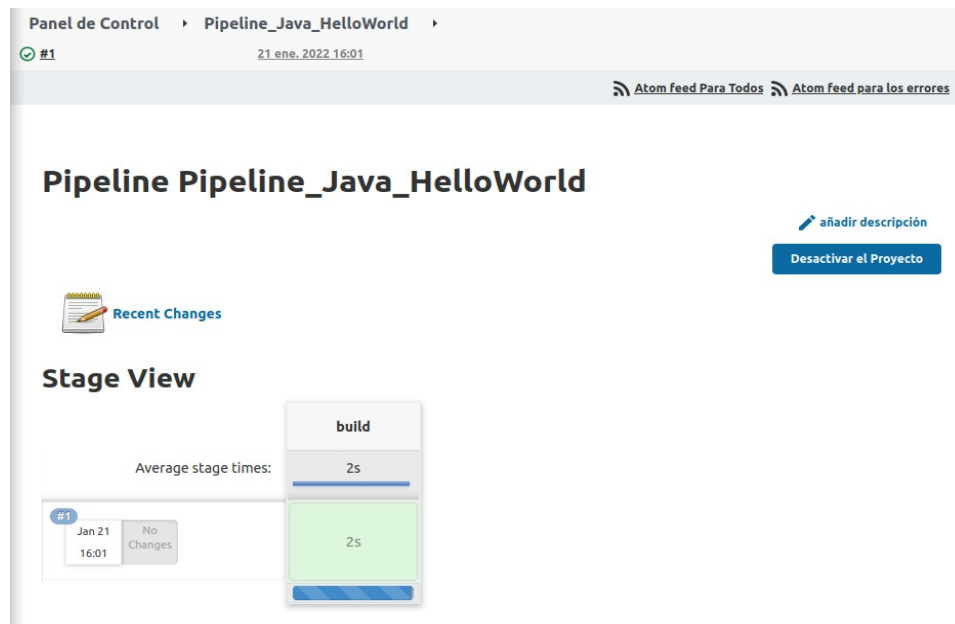
- Status
- Changes
- Construir ahora**
- Configurar
- Borrar Pipeline
- Full Stage View
- Rename
- Pipeline Syntax

Historia de tareas

Filter builds...

#1 21 ene. 2022 16:03

Si la tarea se ha ejecutado con éxito, debemos ver una pantalla como la siguiente.



Si miramos la salida de la tarea, veremos la versión de que se ha ejecutado el comando “**mvn --version**” y se muestra la salida correctamente.

```
Stage Logs (build)
Shell Script -- mvn --version (self time 905ms)

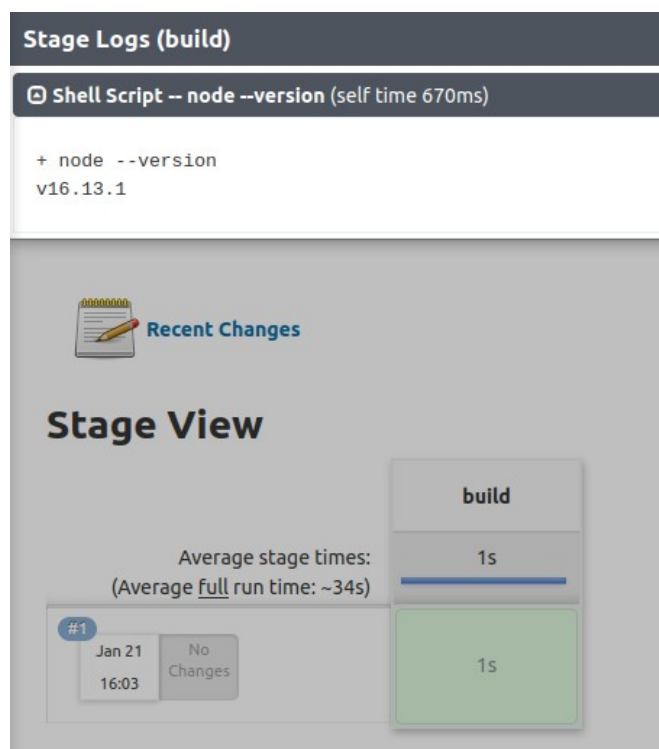
+ mvn --version
Apache Maven 3.8.4 (9b656c72d54e5bacbed989b64718c159fe39b537)
Maven home: /usr/share/maven
Java version: 11.0.13, vendor: Oracle Corporation, runtime: /usr/local/openjdk-11
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-163-generic", arch: "amd64", family: "unix"
```

3. Parte 3

Creamos otra nueva tarea Pipeline, esta vez para **NodeJS**. Tal y como lo hicimos en el paso anterior pero cambiaremos el script por uno que descargue un **contenedor Docker de NodeJS** y ejecute el comando “**node --version**”.



Seguimos completando la tarea y esperamos a que se ejecute, si se ejecuta correctamente deberá mostrar la versión de **NodeJS** del contenedor.

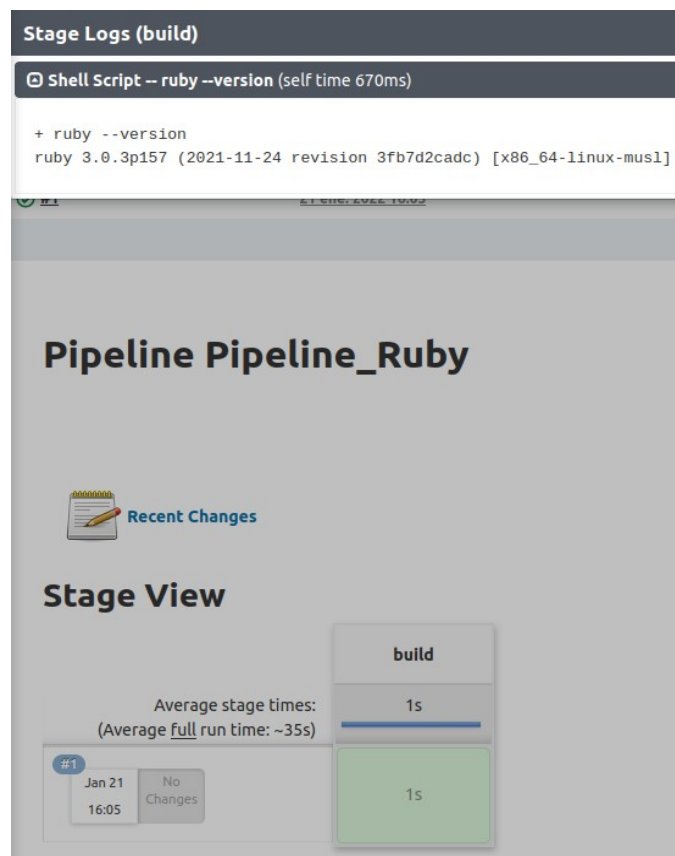


4. Parte 4

Repetimos el proceso, esta vez para un contenedor Docker de **Ruby**



Al ejecutarse la tarea correctamente, nos mostrará la versión de **Ruby** del contenedor.



5. Parte 5

Crearemos otro Pipeline para un contenedor Docker de **Python** mediante el script que se muestra en la imagen.

Pipeline

Definition

Pipeline script

Script

```
1 pipeline {
2   agent { docker { image 'python:3.10.1-alpine' } }
3   stages {
4     stage('build') {
5       steps {
6         sh 'python --version'
7       }
8     }
9   }
10 }
```

try sample Pipeline...


Al finalizar la tarea nos mostrará la versión de **Python** del contenedor.

Stage Logs (build)

Shell Script -- python --version (self time 686ms)

```
+ python --version
Python 3.10.1
```

Pipeline Pipeline_Python

 Recent Changes

Stage View

Average stage times:
(Average full run time: ~33s)

#1	Jan 21 16:08	No Changes
build	1s	1s

6. Parte 6

Esta vez haremos lo mismo para un contenedor de **PHP** usando el siguiente script

Pipeline

Definition

Pipeline script

Script

```
1 pipeline {  
2   agent { docker { image 'php:8.1.0-alpine' } }  
3   stages {  
4     stage('build') {  
5       steps {  
6         sh 'php --version'  
7       }  
8     }  
9   }  
10 }
```

try sample Pipeline...

☒ Use Groovy Sandbox


Al finalizar podremos ver que se nos muestra la versión de **PHP** del contenedor.

Stage Logs (build)

Shell Script -- php --version (self time 812ms)

```
+ php --version  
PHP 8.1.0 (cli) (built: Nov 30 2021 07:15:23) (NTS)  
Copyright (c) The PHP Group  
Zend Engine v4.1.0, Copyright (c) Zend Technologies
```

Pipeline Pipeline_PHP

 Recent Changes

Stage View

Average stage times:
(Average full run time: ~38s)

#	Time	Changes
#1	Jan 21 16:10	No Changes

build

1s

1s

7. Parte 7

Por último, crearemos otro Pipeline para un contenedor **Go** usando el siguiente script

Pipeline

Definition

Pipeline script

Script

```
1 pipeline {
2   agent { docker { image 'golang:1.17.5-alpine' } }
3   stages {
4     stage('build') {
5       steps {
6         sh 'go version'
7       }
8     }
9   }
10 }
```

try sample Pipeline...

Al completarse la tarea, veremos la versión de **Go** del contenedor Docker.


Stage Logs (build)

Shell Script -- go version (self time 644ms)

```
+ go version
go version go1.17.5 linux/amd64
```

#1 21 ene. 2022 16:11

Pipeline Pipeline_Go

 Recent Changes

Stage View


Average stage times:
(Average full run time: ~42s)

















build
1s
1s

#1 Jan 21 16:11 No Changes

8. Parte 8


Si todas las tareas se han ejecutado con éxito, la pagina principal de Jenkins debería mostrarse de una manera similar a esta. Donde vemos con el tick verde que la ultima ejecución de los contenedores fue exitosa, el sol indica que las dos últimas ejecuciones no produjeron errores, el tiempo que ha pasado desde la última ejecución exitosa de la tarea, el último fallo (N/D en nuestro caso ya que ninguno falló), y la duración de la última ejecución.



Todo +						
S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración	
		Pipeline_Go	1 Min 8 Seg - #1	N/D	42 Seg	
		Pipeline_Java_HelloWorld	11 Min - #1	N/D	18 Seg	
		Pipeline_NodeJS	9 Min 14 Seg - #1	N/D	34 Seg	
		Pipeline_PHP	2 Min 44 Seg - #1	N/D	38 Seg	
		Pipeline_Python	4 Min 43 Seg - #1	N/D	33 Seg	
		Pipeline_Ruby	7 Min 9 Seg - #1	N/D	35 Seg	

Icono: S M L

 Guía de iconos

 Atom feed para todos

 Atom feed para fallas

 Atom feed para los más recientes