

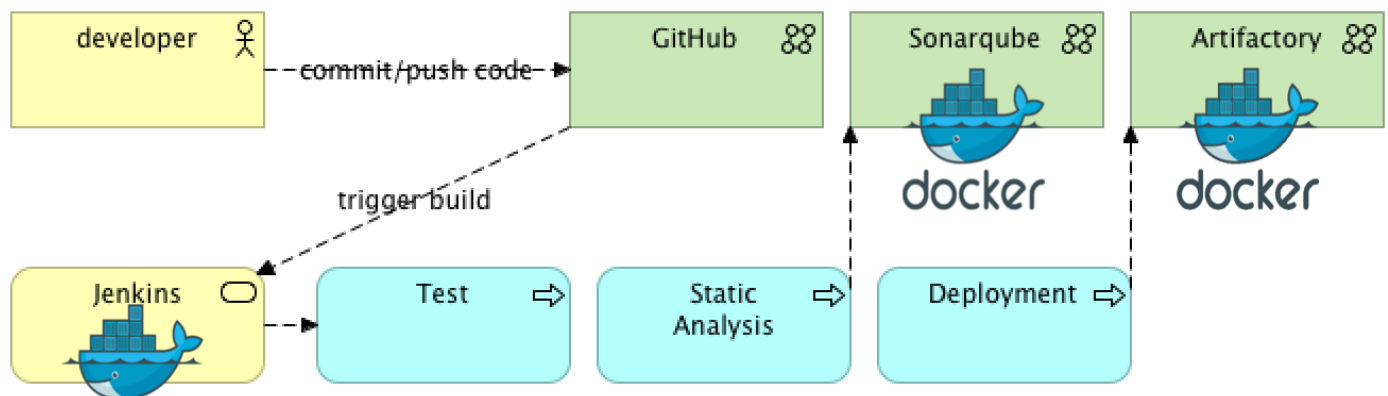


Dockerizing Jenkins, Parte 2: Despliegue con Maven y JFrog Artifactory

por Kayan Azimov · Jul. 31, 17 · DevOps Zone

The Nexus Suite is uniquely architected for a DevOps native world and creates value early in the development pipeline, provides precise contextual controls at every phase, and accelerates DevOps innovation with automation you can trust. Read how in this ebook.

In the first part of this tutorial, we looked at how to dockerize installation of the Jenkins plugins, Java and Maven tool setup in Jenkins 2, and created a declarative build pipeline for a Maven project with test and SonarQube stages. In this part, we will focus on deployment.



Couldn't we simply add another stage for deployment in part 1, you may ask? Well, in fact, deployment requires quite a few steps to be taken, including Maven pom and settings file configuration, artifact repository availability, repository credentials encryption, etc. Let's add them to the list and then implement them step by step like we did in the previous session.

- Running JFrog Artifactory on Docker.
- Configuring the Maven pom file.
- Configuring the Maven settings file.
- Using Config File Provider Plugin for persistence of Maven settings.
- Dockerizing the installation and configuration process.

If you are already familiar with first part of this tutorial, created your project from the scratch and using your own repository, then you can just follow the steps as we go further, otherwise, if you are starting now, you can just clone/fork the work we did in the last example and then add changes as they follow in the tutorial:

```
git clone https://github.com/kenych/jenkins_docker_pipeline_tutorial11 && cd jenkins_docker
```

Please note all steps have been tested on MacOS Sierra and Docker version 17.05.0-ce and you should change them accordingly if you are using MS-DOS, FreeBSD, etc.

The script above is going to take a while as it is downloading Java 7, Java 8, Maven, SonarQube, and Jenkins Docker images, so please be patient. Once done, you should have Jenkins and Sonar up and running as we created in part 1:

Pipeline View

Declarative: Checkout SCM	Declarative: Tool Install	install and sonar parallel
4s	5s	54s

Stage View

Average stage times:

Build #	Time	Status
#1	Jul 16 19:24	No Changes

Permalinks

- [Last build \(#1\), 1 min 7 sec ago](#)

Projects

Project	Quality Gate	Reliability	Security	Maintainability	Coverage	Duplications	Java
berlin-clock	Passed	A	A	A	0.0%	0.0%	301

If you got errors about some port being busy, just use the free ports from your host, as I explain here. Otherwise, you can use dynamic ports.

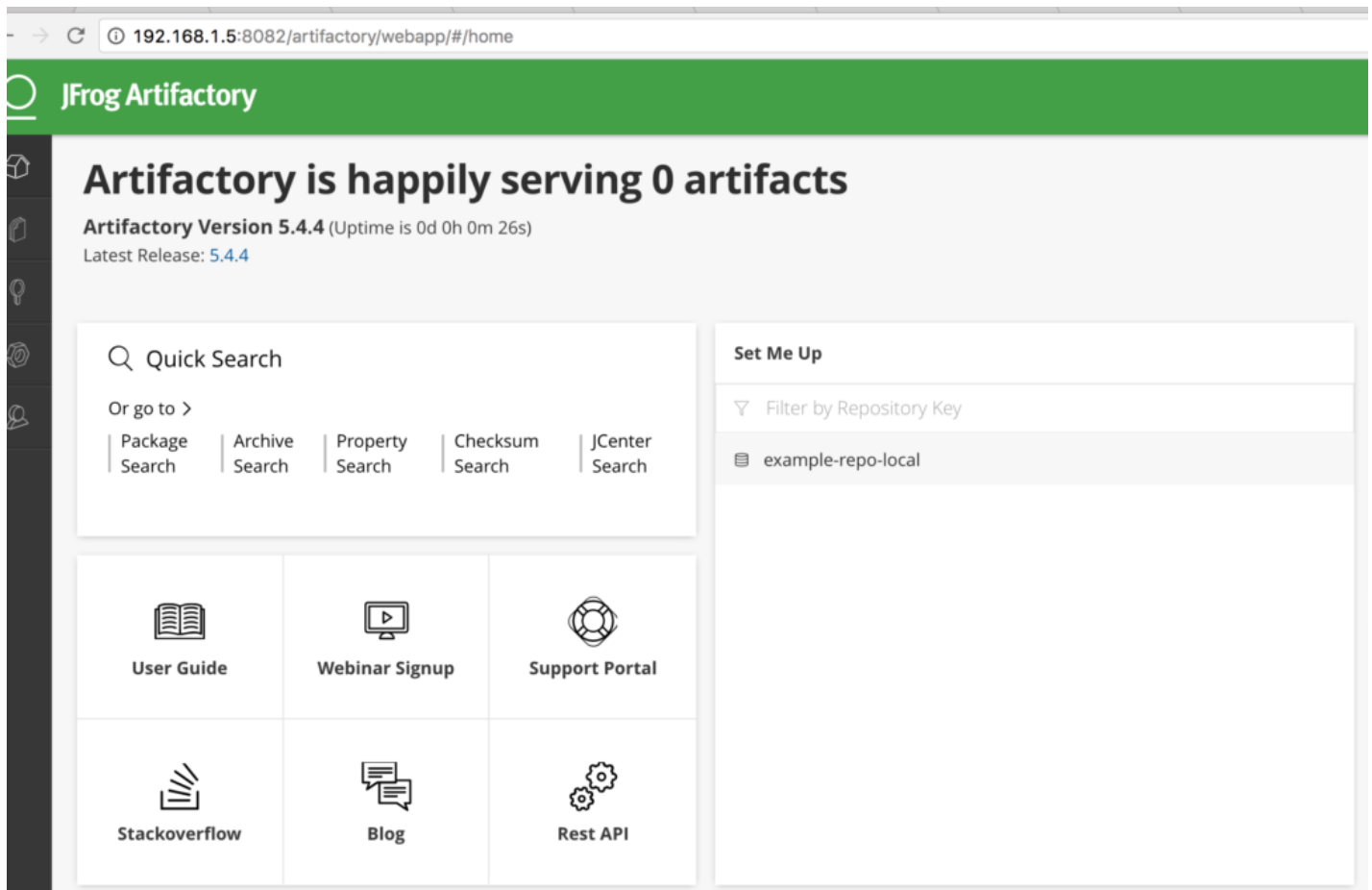
Chapter 1: Running JFrog Artifactory on Docker

Así que veamos el primer paso. Obviamente, si queremos probar la implementación en nuestro ejemplo, necesitamos algún lugar para desplegar nuestros artefactos. Vamos a utilizar una versión de código abierto de JFrog Artifactory llamada "artifactory oss". Vamos a ejecutarlo en Docker para ver

qué fácil es tener su propio repositorio de artefactos. El puerto 8081 en mi máquina estaba ocupado, así que tuve que ejecutarlo en 8082. Debería hacerlo de acuerdo con los puertos libres disponibles en su máquina:

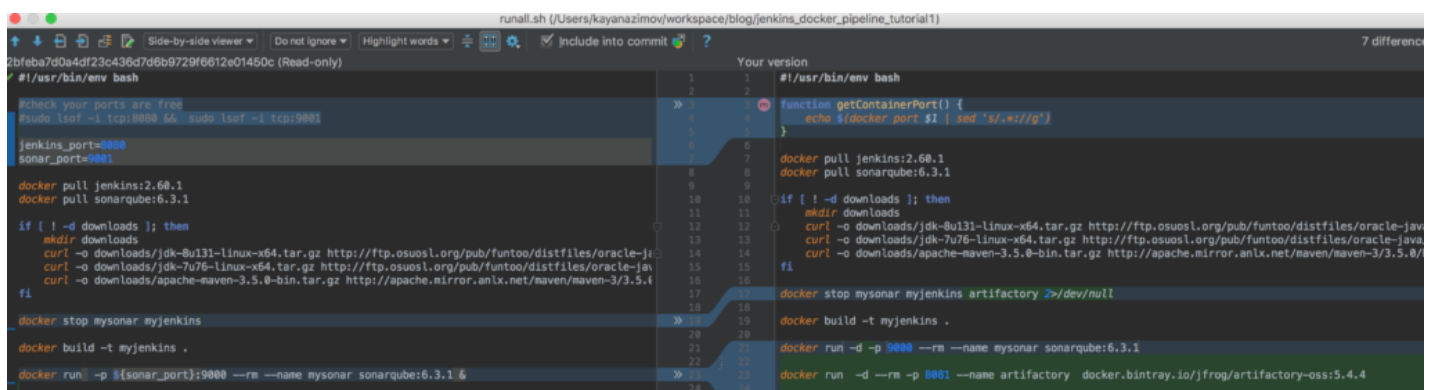
```
estibador ejecutar --rm -p 8082 : 8081 --name Artifactory docker.bintray.io/jfrog/artifactory-oss:5.4.4
```

Normalmente ejecutará Artifactory con volúmenes para preservar su estado (los archivos jar en nuestro caso), pero por el bien de nuestro tutorial que trata sobre Dockerizing Jenkins, simplemente lo ejecutaremos en modo "en memoria". Una vez que esté en funcionamiento, deberíamos estar listos para desplegar nuestros artefactos en él.



Preste atención al nombre de la ruta para el repositorio predeterminado que ha creado; fue un ejemplo-repo-local en mi caso, y nos referiremos a él muy pronto.

Cuando terminé la segunda parte, mejoré el script runall; ahora usa puertos dinámicos y debería ejecutarse sin problemas si algunos de los puertos utilizados por SonarQube, Jenkins o Artifactory estuvieran ocupados (8080, 8081, 9000, etc.):



```

IP=$(ifconfig en0 | awk '/ *inet / {print $2}')
echo "Host ip: $IP"
if [ ! -d m2deps ]; then
    mkdir m2deps
fi

docker run -p $(jenkins_port):8080 -v `pwd`/downloads:/var/jenkins_home/downloads \
-v `pwd`/jobs:/var/jenkins_home/jobs/ \
-v `pwd`/m2deps:/var/jenkins_home/.m2/repository/ --rm --name myjenkins \
-e SONARQUBE_HOST=http://$(IP):$(sonar_port) \
myjenkins:latest

sonar_port=$(getContainerPort mysonar)
artifactory_port=$(getContainerPort artifactory)
IP=$(ifconfig en0 | awk '/ *inet / {print $2}')
if [ ! -d m2deps ]; then
    mkdir m2deps
fi

docker run -d -p 8080 -v `pwd`/downloads:/var/jenkins_home/downloads \
-v `pwd`/jobs:/var/jenkins_home/jobs/ \
-v `pwd`/m2deps:/var/jenkins_home/.m2/repository/ --rm --name myjenkins \
-e SONARQUBE_HOST=http://$(IP):$(sonar_port) \
-e ARTIFACTORY_URL=http://$(IP):$(artifactory_port)/artifactory/example-repo-local \
myjenkins:latest

echo "Sonarqube is running at $(IP):$(sonar_port)"
echo "Artifactory is running at $(IP):$(artifactory_port)"
echo "Jenkins is running at $(IP):$(getContainerPort myjenkins)"

```

Así que cambiemos al script actualizado. Simplemente detendrá todos los contenedores, construirá una imagen de Jenkins y luego volverá a ejecutar con puertos dinámicos:

```

1  #! / usr / bin / env bash
2
3  function getContainerPort () {
4      echo $(docker port $ 1 | sed 's /.*: // g')
5  }
6
7  docker pull jenkins: 2.60.1
8  docker pull sonarqube: 6.3.1
9
10 si [! -d descargas]; entonces
11     mkdir descargas
12     curl -o downloads / jdk-8u131-linux-x64.tar.gz http://ftp.osuosl.org/pub/funtoo/distfi
13     curl -o downloads / jdk-7u76-linux-x64.tar.gz http://ftp.osuosl.org/pub/funtoo/distfi
14     curl -o downloads / apache-maven-3.5.0-bin.tar.gz http://apache.mirror.anlx.net/maver
15 fi
16
17 docker stop mysonar myjenkins artifactory 2 > / dev / null
18
19 estibador construir -t myjenkins.
20
21 estibador ejecutar -d -p 9000 --rm --name mysonar sonarqube: 6.3.1
22
23 estibador ejecutar -d --rm -p 8081 --name Artifactory docker.bintray.io/jfrog/artifa
24
25 sonar_port = $ (getContainerPort mysonar)
26 artifactory_port = $ (getContainerPort artifactory)
27
28 IP = $ (ifconfig en0 | awk '/ * inet / {print $ 2 }')
29
30 si [! -d m2deps]; entonces
31     mkdir m2deps
32 fi
33
34 docker run -d -p 8080 -v `pwd` / downloads: / var / jenkins_home / downloads \

```

```

35 -v `pwd` / jobs: / var / jenkins_home / jobs / \
36 -v `pwd` /m2deps:/var/jenkins_home/.m2/repository/ --rm --name myjenkins \
37 -e SONARQUBE_HOST = http: // $ {IP} : $ {puerto_sonar} \
38 -e ARTIFACTORY_URL = http: // $ {IP} : $ {artifactory_port} / artifactory / example-r
39 myjenkins: último
40
41 echo "Sonarqube se está ejecutando en $ {IP} : $ {sonar_port} "
42 echo "Artifactory se está ejecutando en $ {IP} : $ {artifactory_port} "
43 echo "Jenkins se está ejecutando en $ {IP} : $ (getContainerPort myjenkins) "

```

Ejecútelo en modo de depuración para ver lo que está sucediendo:

```

1 bash + x runall.sh
2 2.60.1: Tirar de la biblioteca / jenkins
3 Resumen: sha256: fa62fcebabeab220e7545d1791e6eea6759b4c3bdba246dd839289f2b28b653e72
4 Estado: la imagen está actualizada para jenkins: 2.60.1
5 6.3.1: Tirar de la biblioteca / sonarqube
6 Resumen: sha256: d5f7bb8aeca46da054bf28d111e5a27f1378188b427db64cc9fb392e1a8d80a
7 Estado: la imagen está actualizada para sonarqube: 6.3.1
8 mysonar
9 myjenkins
10 artefacto
11 Enviar contexto de construcción a Docker daemon 365.1MB
12 Paso 1 /6: a Jenkins: 2.60.1
13 - - y amp; gt; f426a52bafa9
14 Paso 2 /6: MAINTAINER Kayan Azimov
15 - - y amp; gt; Usando caché
16 - - y amp; gt; 760e7bb0f335
17 Paso 3 /6: ENV JAVA_OPTS "-Djenkins.install.runSetupWizard = false"
18 - - y amp; gt; Usando caché
19 - - y amp; gt; e3dbac0834cd
20 Paso 4 /6: COPIAR Plugins.txt /usr/share/jenkins/ref/plugins.txt
21 - - y amp; gt; Usando caché
22 - - y amp; gt; 39966bece010
23 Paso 5 /6: RUN /usr/local/bin/install-plugins.sh & amp; lt; /usr/share/jenkins/ref/p:
24 - - y amp; gt; 987bdeca2517
25 Paso 6 /6: COPIAR maravilloso / * /usr/share/jenkins/ref/init.groovy.d/
26 - - y amp; gt; Usando caché
27 - - y amp; gt; e5ec6b7f49aa
28 Construido con éxito e5ec6b7f49aa
29 Etiquetado con éxito myjenkins: último
30 85d8716d3c7fd6272b6915d977daa37dbe9e4ece0f5c367dd9798fbfca272b2d
31 5fc3f649d3ba5c47df48a54b761d432611ed18872aa727114cdcf1a0f30cac0c
32 21e98466cb4f6e78817d9869d39bda57f475d7174f565031f168eb836118d701
33 Sonarqube corre a 192.168.1.2: 32836
34 Artifactory se ejecuta en 192.168.1.2: 32837

```

```
35 Jenkins corre a 192.168.1.2: 32838
```

Ahora vemos los puertos para todos los contenedores en ejecución en los registros, por lo que podemos acceder a cualquiera de ellos si queremos. Como habrás notado, nuestro script de shell se ha vuelto demasiado complicado; esto es solo para ejecutar tres contenedores, por lo que significa que la próxima vez, probablemente debería cambiar a usar docker compose en su lugar!

Obviamente tenemos que actualizar nuestra cartera de proyectos que creamos en la parte 1, agregarle un paso de implementación, insertar el archivo Jenkins actualizado y crear el trabajo. Si está utilizando su propio repositorio, alternativamente, simplemente modifique el trabajo existente de la siguiente manera utilizando el botón Replicar en la última versión exitosa y luego ejecútelo:

```
1  tubería {
2      agente cualquier
3
4      herramientas {
5          jdk 'jdk8'
6          maven 'maven3'
7      }
8
9      etapas {
10         stage ( 'instalar y sonar paralelo' ) {
11             pasos {
12                 paralelo (
13                     instalar: {
14                         sh "mvn -U cobertura de prueba limpia: cobertura -Dcobertura.
15                     },
16                     sonar: {
17                         sh "mvn sonar: sonar -Dsonar.host.url = $ { env . SONARQUBE_H
18                     }
19                 )
20             }
21             publicar {
22                 siempre {
23                     junit '** / target / * - reports / TEST - *. xml'
24                     step ([ $ class: 'CoberturaPublisher' , coberturaReportFile: 'target
25                 }
26             }
27         }
28         stage ( 'deploy' ) {
29             pasos {
30                 sh "mvn deploy -DskipTests"
31             }
32         }
33     }
```

Esto es lo que va a suceder:

```

1 [INFO] BUILD FAILURE
2 [INFO] - - - - -
3 [INFO] Tiempo total: 9 .265 s
4 [INFO] Finalizado en: 2017 -07 -16T18 : 39: 51Z
5 [INFO] Memoria final: 17M / 95M
6 [INFO] - - - - -
7 [ERROR] Error al ejecutar el objetivo org.apache.maven.plugins: maven-deploy-plugin: 2.7:
8 [ERROR]
9 [ERROR] Para ver el seguimiento completo de la pila de los errores, vuelva a ejecutar Maven
10 [ERROR] Vuelva a ejecutar Maven utilizando el modificador -X para habilitar el registro de
11 [ERROR]
12 [ERROR] Para obtener más información sobre los errores y posibles soluciones, lea los siguientes
13 [ERROR] [Ayuda 1 ] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
14 [Pipeline]}

```

El trabajo falla, pero Maven es lo suficientemente astuto como para descubrir el motivo y nos avisó con un mensaje de que falta la sección de gestión de distribución en el archivo pom. Entonces, el plugin de implementación de Maven básicamente no sabe dónde desplegar el artefacto, ¿lo sabías?

Capítulo 2: Configurando Maven pom File

Vamos a configurar la parte que falta en el archivo pom dentro del proyecto. Si no está utilizando su propio proyecto git para este tutorial, no podrá modificar el archivo POM, ya que está en mi repositorio y no tiene acceso, lamentablemente. Pero puede cambiar a otro proyecto en ese caso, que preparé para usted. Por lo tanto, cree otro trabajo de interconexión para el proyecto laberinto-explorador , que tiene los cambios de pom necesarios. Si está utilizando su propio proyecto Git, simplemente ignore esta nota. Obviamente, podría haber usado solo una rama en el mismo proyecto, ipero tengamos un par de empleos en Jenkins!


Creando una nueva canalización:

:8080/view/all/newJob


Enter an item name

maze-explorer

» Required field


Freestyle project

This is the central feature of Jenkins. Jenkins will build used for something other than software build.


Pipeline

Orchestrates long-running activities that can span multi

and/or organizing complex activities that do not easily f

- External Job**
This type of job allows you to record the execution of a that you can use Jenkins as a dashboard of your existi
- Multi-configuration project**
Suitable for projects that need a large number of differ builds, etc.
- Folder**
Creates a container that stores nested items in it. Usefi separate namespace, so you can have multiple things i
- GitHub Organization**
Scans a GitHub organization (or user account) for all re
- Multibranch Pipeline**
Creates a set of Pipeline projects according to detecte

OK

Configurando el repositorio

ost:8080/job/maze-explorer/configure

maze-explorer >

General Build Triggers **Advanced Project Options** Pipeline

☐ Quiet period

Advanced Project Options

Advanced...

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL github.com/kenych/maze-explorer

Please enter Git repository.

Credentials - none - Add

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any') */master

Add Branch

Repository browser (Auto)

Additional Behaviours Add

Save Apply Script Path Jenkinsfile

ASI ES COMO SE VERAN SUS CAMBIOS EN EL POM, AÑADIENDO `DistributionManagement` a la seccion del proyecto:

```

1  < project xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns = "http://maven
2      < modelVersion > 4.0.0 </ modelVersion >
3
4      < groupId > kayan </ groupId >
5      < artifactId > laberinto </ artifactId >
6      < version > 1.0-SNAPSHOT </ version >
7      < packaging > jar </ packaging >
8
9      < url > http://maven.apache.org </ url >
10
11     < propiedades >
12         < project.build.sourceEncoding > UTF-8 </ project.build.sourceEncoding >
13     </ propiedades >
14
15     < distributionManagement >
16         < snapshotRepository >
17             < id > artifactory </ id >
18             < nombre > artefacto </ name >
19             < url > $ {artifactory_url} </ url >
20         </ snapshotRepository >
21     </ distributionManagement >
22
23
24     (el resto del archivo POM)

```

La URL debe apuntar a la URL de Artifactory, por lo que debemos pasarla a Maven a través de Jenkins como una variable de entorno, tal como lo hicimos con Sonar. Obviamente no puede ser estático ya que la IP del host puede cambiar.

Si ejecuta el trabajo ahora, todavía va a fallar, incluso si tiene configurado `distributionManagement`:

```

1  [INFO] BUILD FAILURE
2  [INFO] - - - - -
3  [INFO] Tiempo total: 4 .459 s
4  [INFO] Finalizado en: 2017 -07 -16T20 : 13: 34Z
5  [INFO] Memoria final: 13M / 137M
6  [INFO] - - - - -
7  [ERROR] Error al ejecutar el objetivo org.apache.maven.plugins: maven-deploy-plugin: 2.7:
8  [ERROR]
9  [ERROR] Para ver el seguimiento completo de la pila de los errores, vuelva a ejecutar Maven
10 [ERROR] Vuelva a ejecutar Maven utilizando el modificador -X para habilitar el registro de

```





Esto se debe a que necesitamos tener credenciales para implementar. Las credenciales predeterminadas de JFrog son "admin: contraseña". Intente iniciar sesión y verifique. Para pasar las credenciales al plugin de implementación, debemos configurarlas en el archivo Maven settings.xml.

Capítulo 3: Configuración del archivo de configuración de Maven

Si tenemos Maven instalado localmente, podemos ejecutarlo primero solo para verificar que la implementación realmente funcione con nuestra configuración, y luego podemos comenzar a buscar cómo configurarlo con Jenkins. Vamos a ver qué tenemos en el archivo de configuración:

```
1 mvn help: efectivos ajustes
```

Verá algo similar si el archivo de configuración está ausente o vacío.

```
1 < settings xmlns = "http://maven.apache.org/POM/4.0.0" xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance" >
2   < localRepository xmlns = "http://maven.apache.org/SETTINGS/1.1.0" > /Users/kayanazimov
3   < pluginGroups xmlns = "http://maven.apache.org/SETTINGS/1.1.0" >
4     < pluginGroup > org.apache.maven.plugins </ pluginGroup >
5     < pluginGroup > org.codehaus.mojo </ pluginGroup >
6   </ pluginGroups >
7 </ configuración >
```

Ahora vayamos a /Users/YOUR_USER_NAME_HERE/.m2/ y cambiemos o creamos settings.xml como a continuación. Para pasar las credenciales del repositorio, necesitamos agregar un servidor en la sección de servidores de la configuración de Maven:

```
1 <? xml version = "1.0" encoding = "UTF-8"?>
2 < settings xsi: schemaLocation = "http://maven.apache.org/SETTINGS/1.1.0 http://maven.apache.org/SETTINGS/1.1.0" >
3   < servidores >
4     < servidor >
5       < id > artifactory </ id >
6       < nombre de usuario > admin </ username >
7       < contraseña > contraseña </ contraseña >
8     </ server >
```

```

9      </ servers >
10 </ configuración >

```

Tenga en cuenta que la identificación debe ser la misma que la que utilizó anteriormente en el archivo pom para SnapshotRepository.

Puedes verificar si Maven lo está recogiendo:

```
1 mvn help: efectivos ajustes
```

Ahora clone el proyecto, o si está utilizando su propio proyecto, simplemente ejecútelo desde la carpeta del proyecto (use el puerto Artifactory, que se mostrará al ejecutar el script runall):

```

1 mvn deploy -DskipTests -Difactory_url = http://localhost:8082/artifactory/example-
2
3 [INFO] Escaneo de proyectos ...
4 [INFO]
5 [INFO] - - - - -
6 [INFO] Building laberinto 1.0-INSTANTÁNEA
7 [INFO] - - - - -
8 [INFO]
9 [INFO] - - maven-resources-plugin: 2.6: resources (default-resources) @ laberinto - -
10 [INFO] Utilizando la codificación 'UTF-8' para copiar los recursos filtrados.
11 [INFO] omite resourceDirectory / Users / kayanazimov / workspace / learn / maze-explorer /
12 [INFO]
13 [INFO] - - maven-compiler-plugin: 2.5.1: compile (default-compile) @ laberinto - -
14 [INFO] Nada para compilar : todas las clases están actualizadas
15 [INFO]
16 [INFO] - - maven-resources-plugin: 2.6: testResources (default-testResources) @ laberinto
17 [INFO] Utilizando la codificación 'UTF-8' para copiar los recursos filtrados.
18 [INFO] Copia de 10 recursos
19 [INFO]
20 [INFO] - - maven-compiler-plugin: 2.5.1: testCompile (default-testCompile) @ laberinto - -
21 [INFO] Nada para compilar : todas las clases están actualizadas
22 [INFO]
23 [INFO] - - maven-surefire-plugin: 2.12.4: test (default-test) @ laberinto - -
24 [INFO] Las pruebas se saltan.
25 [INFO]
26 [INFO] - - maven-jar-plugin: 2.4: jar (default-jar) @ laberinto - -
27 [INFO] Building jar: /Users/kayanazimov/workspace/learn/maze-explorer/target/maze-1.0-SNAPSHOT.jar
28 [INFO]
29 [INFO] - - maven-install-plugin: 2.4: install (default-install) @ laberinto - -
30 [INFO] Instalando /Users/kayanazimov/workspace/learn/maze-explorer/target/maze-1.0-SNAPSHOT.jar
31 [INFO] Instalando /Users/kayanazimov/workspace/learn/maze-explorer/pom.xml en /Users/kayanazimov/workspace/learn/maze-explorer

```

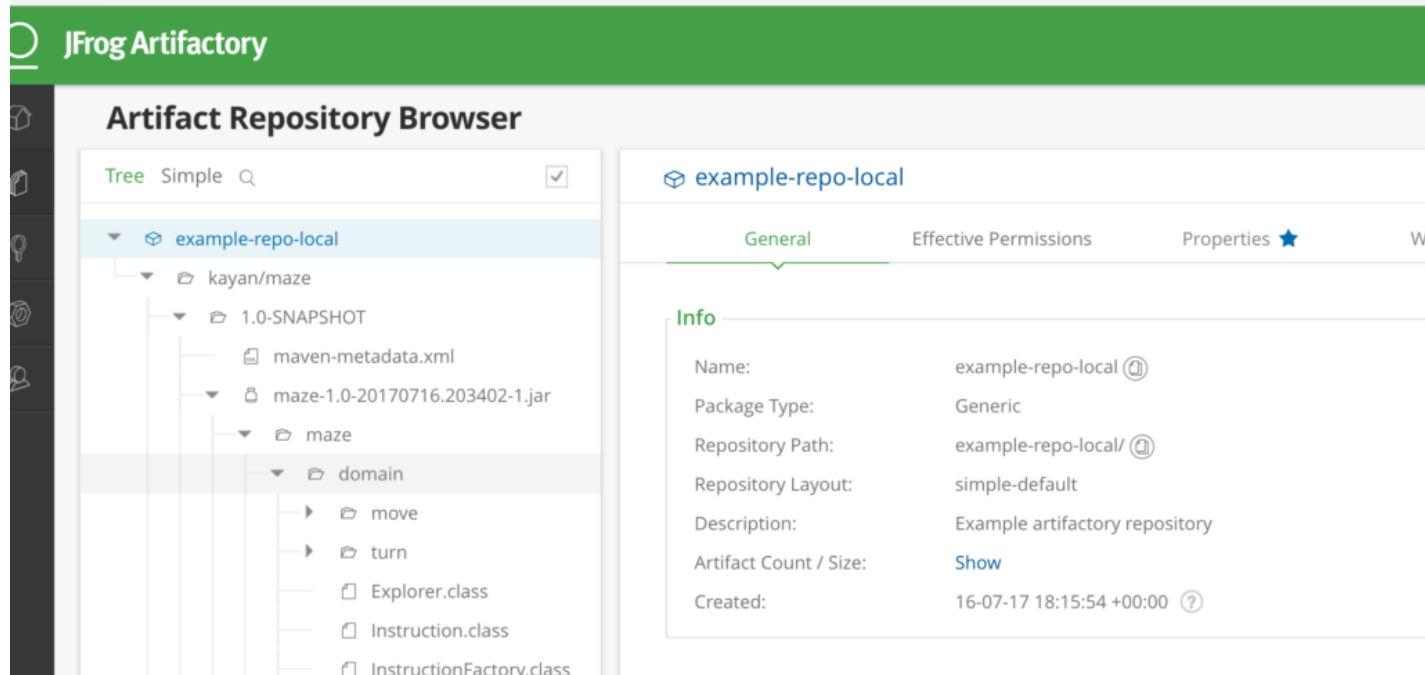
```

31
32 [INFO]
33 [INFO] - - maven-deploy-plugin: 2.7: deploy (default-deploy) @ laberinto - -
34 Descargando: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze /
35 Cargando: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze / 1.0
36 Cargado: http: // localhost: 8082 / artifactory / example-repo-local / kayan / laberinto /
37 Cargando: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze / 1.0
38 Cargado: http: // localhost: 8082 / artifactory / example-repo-local / kayan / laberinto /
39 Descargando: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze /
40 Cargando: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze / 1.0
41 Cargado: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze / 1.0
42 Cargando: http: // localhost: 8082 / artifactory / example-repo-local / kayan / maze / mav
43 Cargado: http: // localhost: 8082 / Artifactory / example-repo local / kayan / laberinto /
44 [INFO] - - - - -
45 [INFO] CONSTRUIR ÉXITO
46 [INFO] - - - - -
47 [INFO] Tiempo total: 1 .828 s
48 [INFO] Finalizado en: 2017 -07 -18T09 : 39: 52 + 01 : 00
49 [INFO] Memoria final: 13M / 207M
50 [INFO] - - - - -
51
52 → maze-explorer git : (maestro)

```

Ahora mira el Artifactory:

localhost:8082/artifactory/webapp/#/artifacts/browse/tree/General/example-repo-local



JFrog Artifactory

Artifact Repository Browser

Tree Simple Q

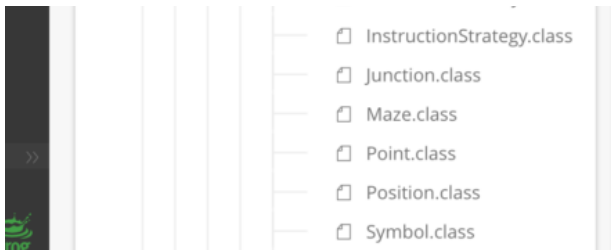
- example-repo-local
 - kayan/maze
 - 1.0-SNAPSHOT
 - maven-metadata.xml
 - maze-1.0-20170716.203402-1.jar
 - maze
 - domain
 - move
 - turn
 - Explorer.class
 - Instruction.class
 - InstructionFactory.class

example-repo-local

General Effective Permissions Properties ★ W

Info

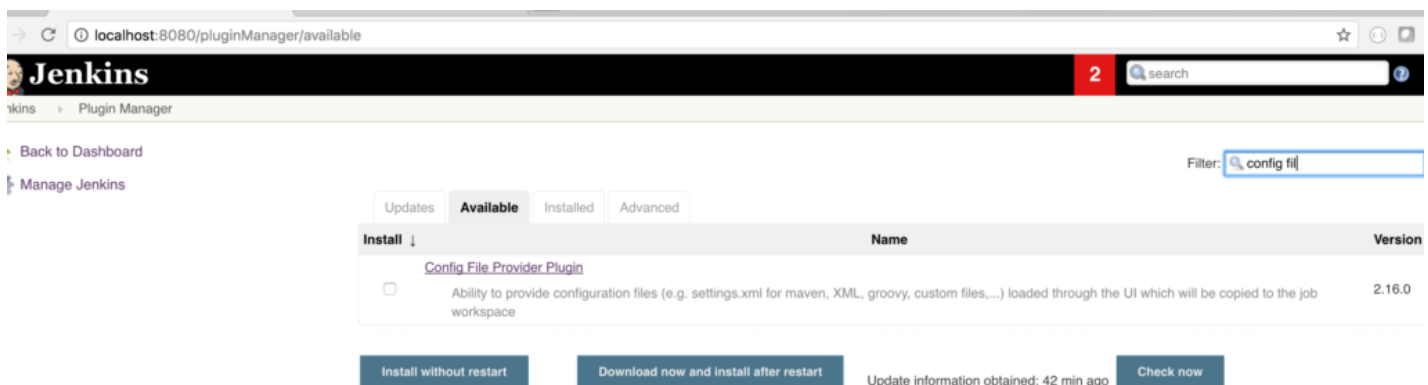
Name:	example-repo-local
Package Type:	Generic
Repository Path:	example-repo-local/
Repository Layout:	simple-default
Description:	Example artifactory repository
Artifact Count / Size:	Show
Created:	16-07-17 18:15:54 +00:00



Como puede ver, lo implementamos con éxito, ¡Yay! Ahora es el momento de preparar este paso en Jenkins.

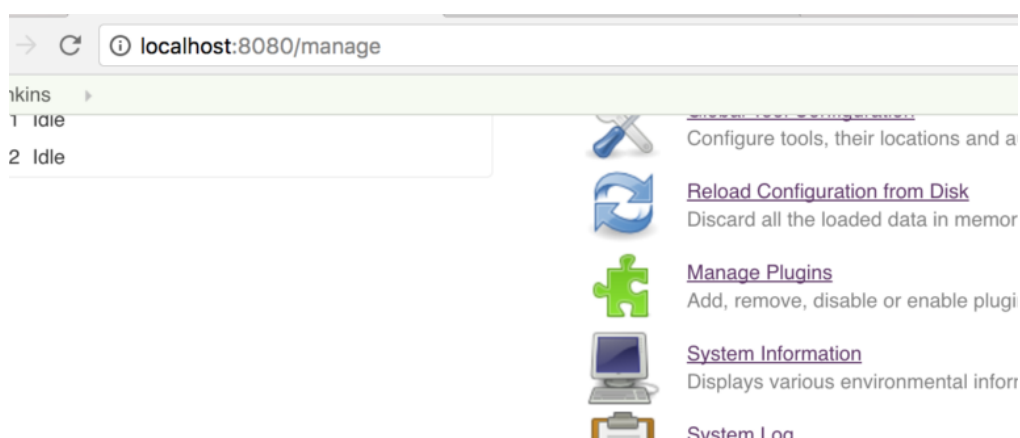
Capítulo 4: Uso del complemento de proveedor de archivos de configuración para la persistencia de las configuraciones de Maven









Para aplicar el archivo de configuración a Maven en Jenkins, necesitamos el complemento de proveedor de archivos de configuración, que nos permite conservar varios archivos de configuración (tenga en cuenta que podríamos necesitar más de una configuración dependiendo de un proyecto ejecutando el trabajo en la vida real). Ahora instalemos el plugin manualmente primero:



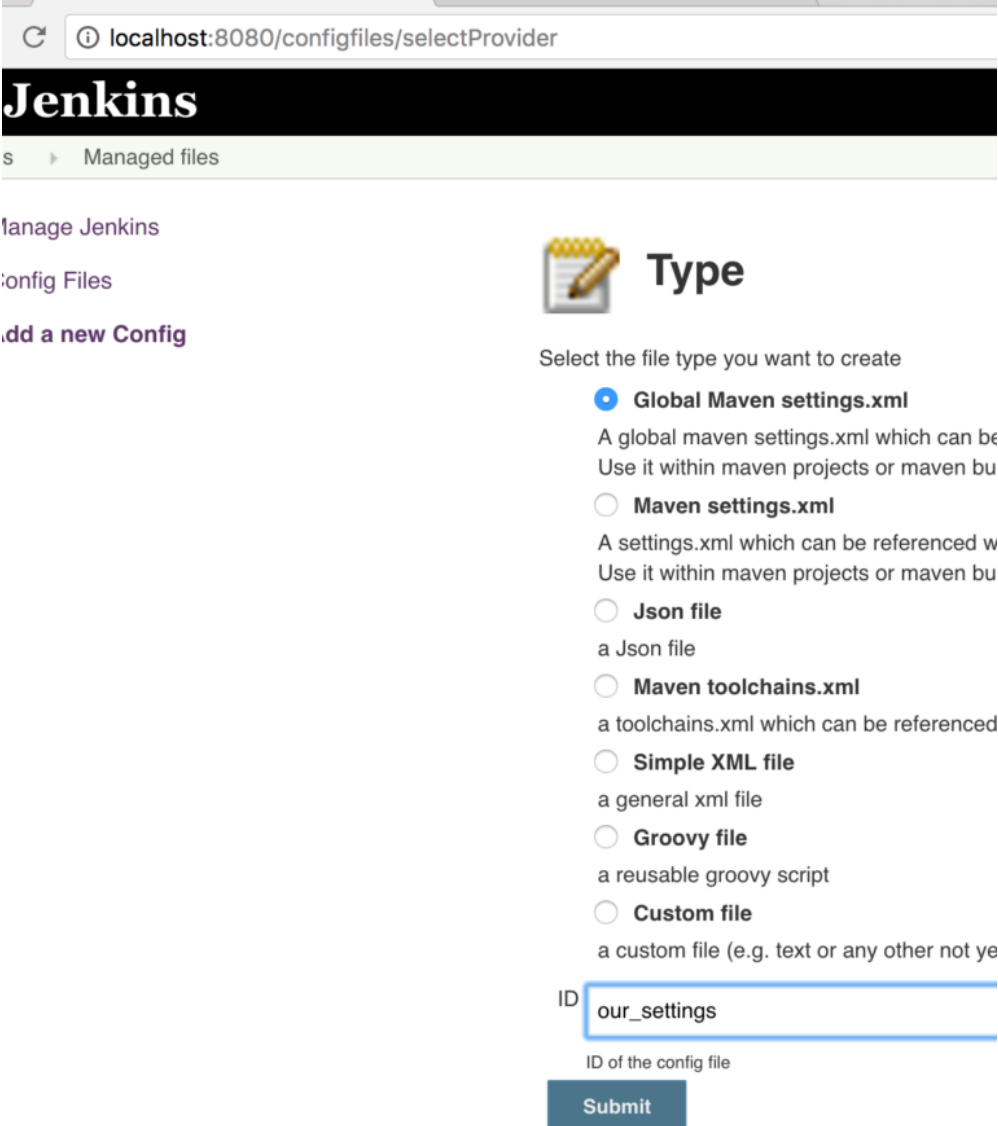
Necesitamos configurar el archivo de configuración de Maven en el complemento. Ve a administrar Jenkins, a los archivos administrados, a agregar nuevas configuraciones, a establecer el ID en "our_settings" y a copiar el contenido de settings.xml que usamos antes:

Haz clic en Archivos administrados:



 [System log](#)
System log captures output from jav
 [Load Statistics](#)
Check your resource utilization and s
 [Jenkins CLI](#)
Access/manage Jenkins from your st
 [Script Console](#)
Executes arbitrary script for administ
 [Manage Nodes](#)
Add, remove, control and monitor the
 [About Jenkins](#)
See the version and license informati
 [Manage Old Data](#)
Scrub configuration files to remove re
 [Managed files](#)
e.g. settings.xml for maven, central n

Selecciona la configuración global de Maven y establece la ID:



The screenshot shows the Jenkins web interface at `localhost:8080/configfiles/selectProvider`. The page title is "Jenkins" and the breadcrumb is "Managed files". On the left sidebar, there are links: "Manage Jenkins", "Config Files", and "Add a new Config". The main content area is titled "Type" with a notepad icon. It says "Select the file type you want to create". There are seven radio button options: "Global Maven settings.xml" (selected), "Maven settings.xml", "Json file", "Maven toolchains.xml", "Simple XML file", "Groovy file", and "Custom file". Each option has a brief description. Below the options is an "ID" input field containing "our_settings". A label "ID of the config file" is below the input. At the bottom is a "Submit" button.

`localhost:8080/configfiles/selectProvider`

Jenkins

Managed files

Manage Jenkins

Config Files

Add a new Config

Type

Select the file type you want to create

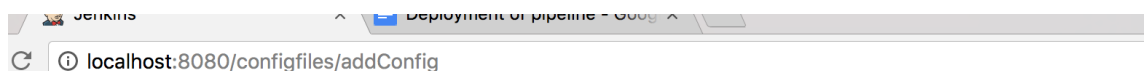
- ☒ **Global Maven settings.xml**
A global maven settings.xml which can be
Use it within maven projects or maven bu
- ☐ **Maven settings.xml**
A settings.xml which can be referenced w
Use it within maven projects or maven bu
- ☐ **Json file**
a Json file
- ☐ **Maven toolchains.xml**
a toolchains.xml which can be referenced
- ☐ **Simple XML file**
a general xml file
- ☐ **Groovy file**
a reusable groovy script
- ☐ **Custom file**
a custom file (e.g. text or any other not ye

ID

ID of the config file

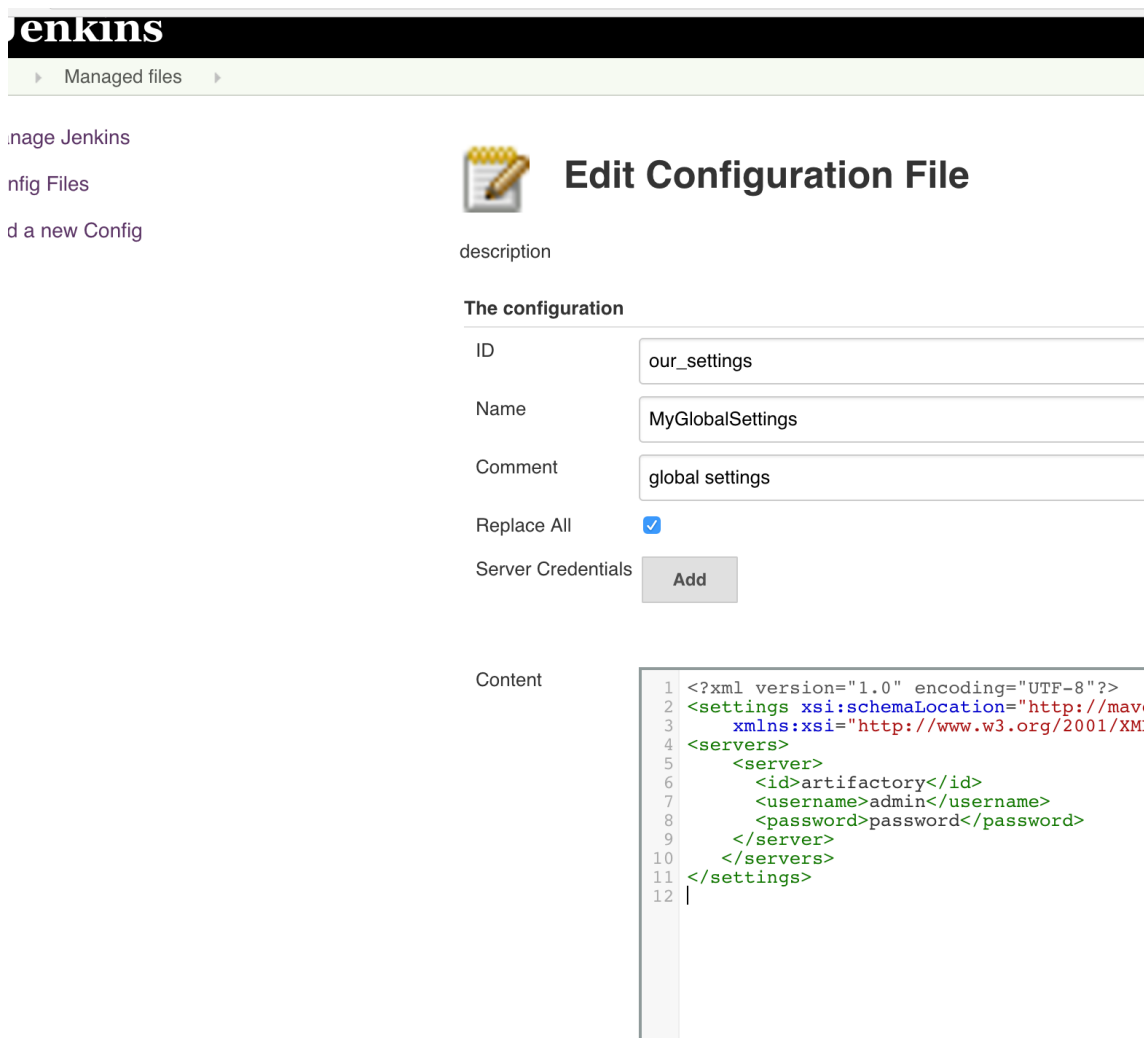
Submit

Y establecer contenido:



The screenshot shows the Jenkins web interface at `localhost:8080/configfiles/addConfig`. The page title is "Jenkins" and the breadcrumb is "Deployment or pipeline - Config".

`localhost:8080/configfiles/addConfig`



enkins

Managed files

Manage Jenkins

Configure Files

Create a new Config

Edit Configuration File

description

The configuration

ID: our_settings

Name: MyGlobalSettings

Comment: global settings

Replace All: ☒

Server Credentials: Add

Content

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <settings xsi:schemaLocation="http://maven.apache.org/setting
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   <servers>
5     <server>
6       <id>artifactory</id>
7       <username>admin</username>
8       <password>password</password>
9     </server>
10  </servers>
11 </settings>
12

```

Ahora podemos utilizar el plugin Proveedor de archivos de configuración en nuestra canalización. Por favor, vuelva a jugar el trabajo y actualice la canalización de la siguiente manera:

```

1 tubería {
2   agente cualquier
3
4   herramientas {
5     jdk 'jdk8'
6     maven 'maven3'
7   }
8
9   etapas {
10    stage ( 'instalar y sonar paralelo' ) {
11      pasos {
12        paralelo (
13          instalar: {
14            sh "mvn -U cobertura de prueba limpia: cobertura -Dcobertura.
15          },
16          sonar: {
17            sh "mvn sonar: sonar -Dsonar.host.url = $ { env . SONARQUE_B
18          }
19        }
20      }
21    }
22  }

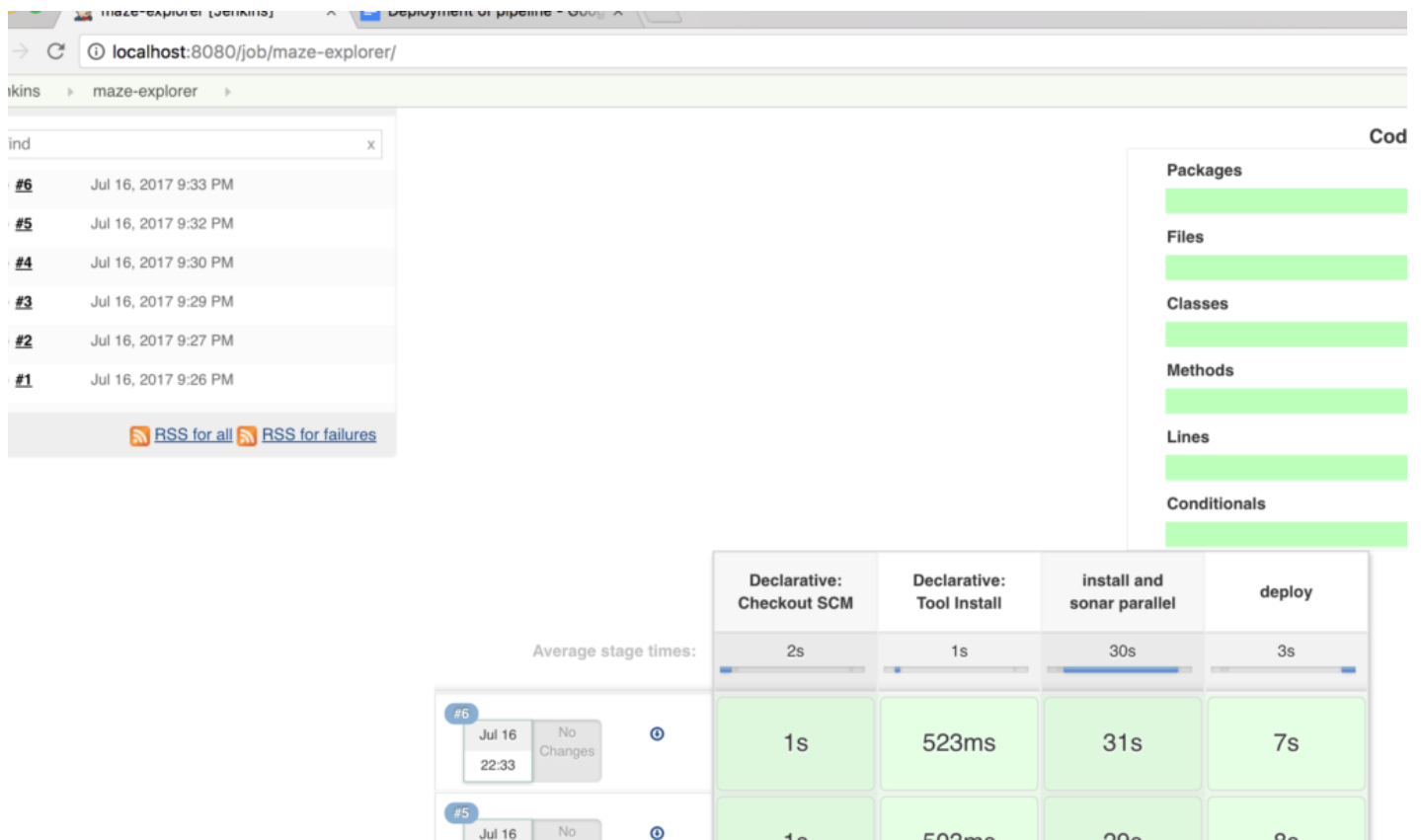
```

```

19      }
20
21      publicar {
22          siempre {
23              junit '** / target / * - reports / TEST - *. xml'
24              step ([ $ class: 'CoberturaPublisher' , coberturaReportFile: 'target'
25          }
26      }
27  }
28  stage ( 'deploy' ) {
29      pasos {
30          configFileProvider ([ configFile ( fileId: 'our_settings' , variable: '!'
31          sh "mvn -s $ SETTINGS deploy -DskipTests -Difactory_url = $ { env . /
32      }
33  }
34  }
35  }
36  }

```

Vamos a ejecutar la construcción:



Si tienes suerte, obtendrás la pantalla de arriba. De lo contrario, lee qué puede salir mal con los contenedores cuando se quede sin memoria .

Capítulo 5: Dockear el proceso de instalación y configuración

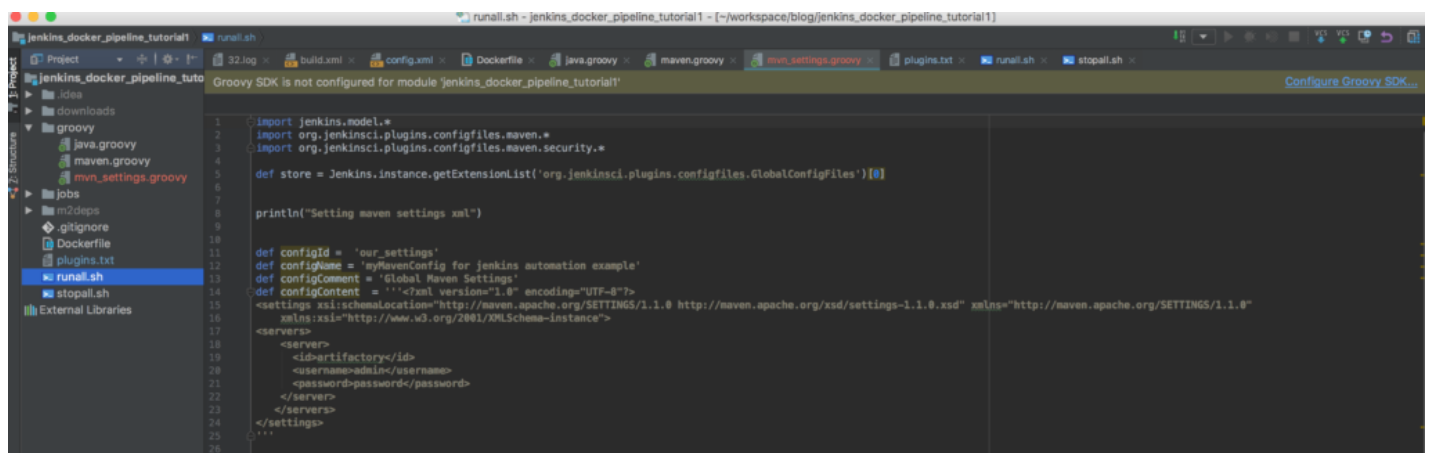
Ahora hagamos la configuración e instalación del plugin. Cree el archivo `mvn_settings.groovy`, cópielo en la carpeta maravillosa que creamos en la primera parte del tutorial y configure el contenido de la siguiente manera:

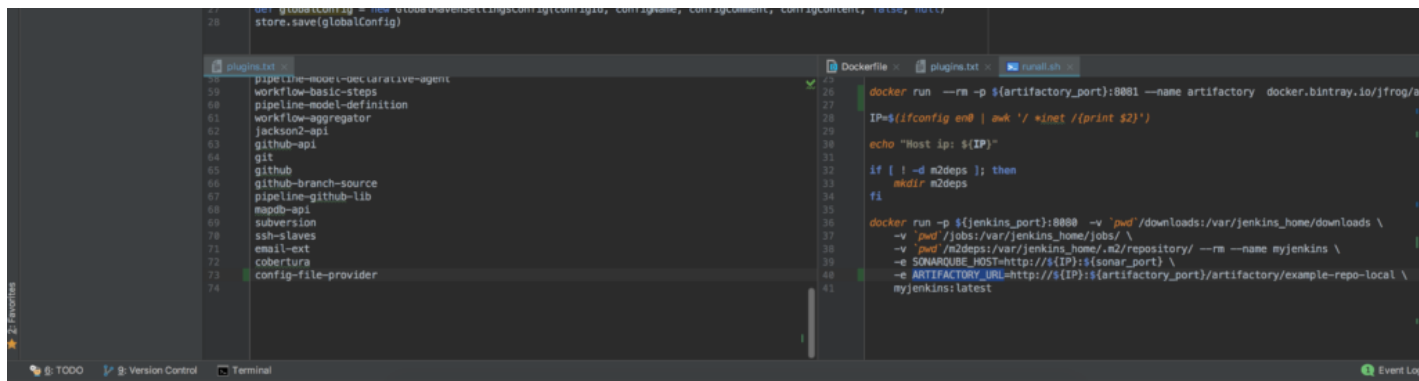
```

1  importar jenkins . modelo . *
2  import org . jenkinsci . plugins . configfiles . experta . *
3  import org . jenkinsci . plugins . configfiles . experta . seguridad . *
4
5  def store = Jenkins . instancia . getExtensionList ( 'org.jenkinsci.plugins.configfile:
6
7
8  println ( "Configuración de configuración maven xml" )
9
10
11 def configId = 'our_settings'
12 def configName = 'myMavenConfig para el ejemplo de automatización de jenkins'
13 def configComment = 'Configuración Global Maven'
14 def configContent = '' '<? xml version = "1.0" encoding = "UTF-8"?>
15 <settings xsi: schemaLocation = "http://maven.apache.org/SETTINGS/1.1.0 http://maven.apac
16 <servidores>
17     <servidor>
18         <id> artifactory </ id>
19         <nombre de usuario> admin </ username>
20         <contraseña> contraseña </ contraseña>
21     </ server>
22 </ servers>
23 </ configuración>
24 '' '
25
26 def globalConfig = new GlobalMavenSettingsConfig ( configId , configName , configComme
27 tienda . guardar ( globalConfig )

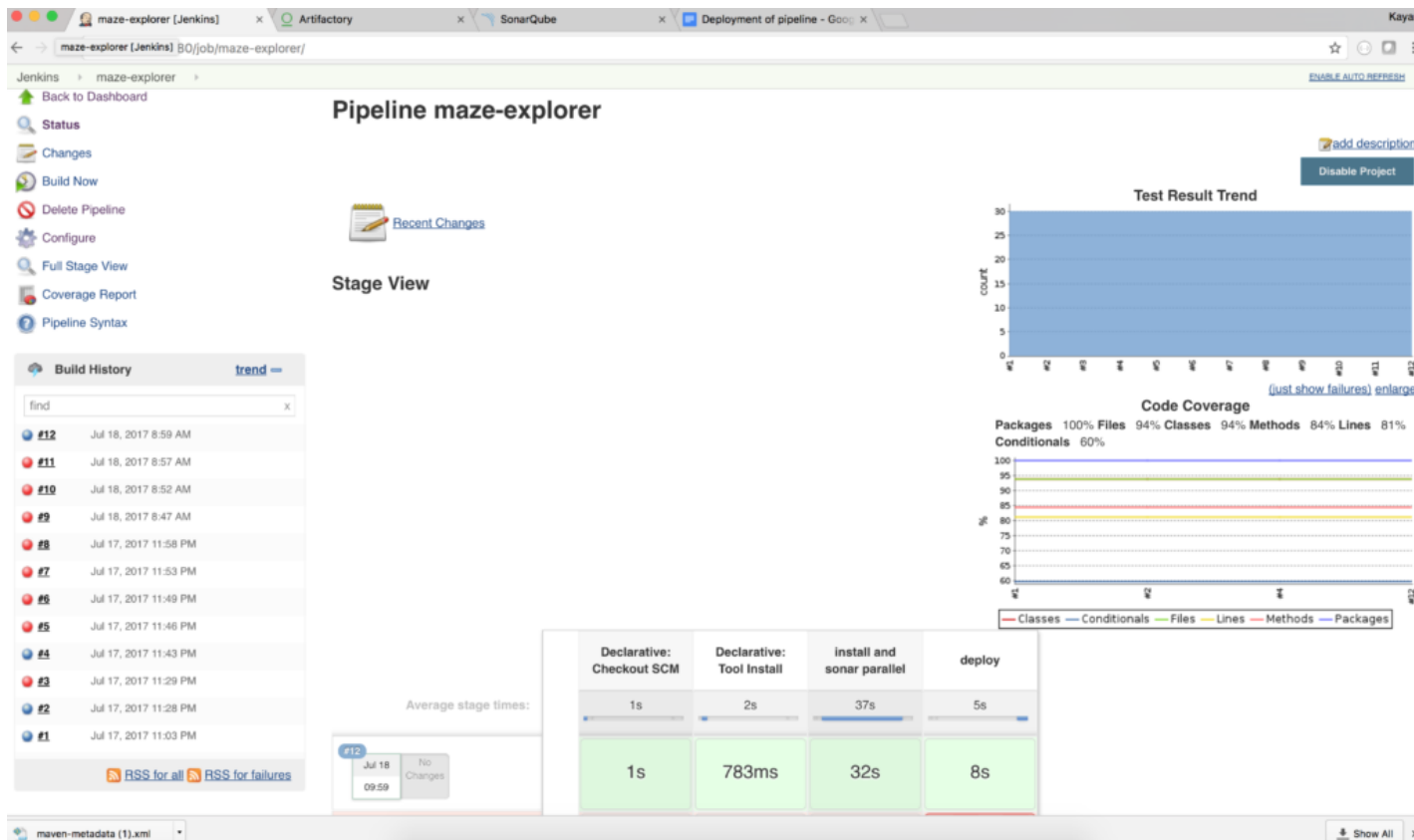
```

Y no olvides instalar el complemento "config-file-provider", solo agrégalo a `plugins.txt` como en esta captura de pantalla:





Es hora de reconstruir y volver a ejecutar tu Jenkins, solo ejecuta el script `runall`. Una vez que Jenkins esté listo, ejecuta la construcción:



Ahora tenemos la cartera de IC para nuestro proyecto y Jenkins casi completamente automatizada. ¿Por qué casi?



Porque necesitamos esconder las contraseñas primero. Pero llego tarde y mi novia ya comenzó a quejarse ...



Pero prometo que veremos cómo usar contraseñas encriptadas en Jenkins muy pronto en la próxima sesión.

De nuevo, si fue flojo y no le gusta el material "práctico", solo clone el código completo para la segunda parte del tutorial, ejecútelo, disfrútelo y compártalo si lo desea:

```
git clone https://github.com/kenych/dockerizing-jenkins-part-2 && cd dockerizing-jenkins-}
```

Espero que hayas logrado ejecutar todo sin problemas y disfrutado este tutorial. Si tienes algún problema, no dudes en comentarlo e intentaré ayudarte.

La zona DevOps se ofrece en colaboración con Sonatype Nexus. Vea cómo la plataforma Nexus infunde inteligencia de componentes de fuente abierta precisa en la tubería de DevOps temprano, en todas partes y a escala. Lea cómo en este ebook .

Me gusta este artículo? Leer más de DZone



**Dockerizing Jenkins 2, Parte 1:
Canal de compilación declarativo
con SonarQube Analysis**



¡Desata los DevOps!



**Configuración de Jenkins para
implementar en Heroku**



**Gratis DZone Refcard
Comenzando con Docker**

Temas: DOCKER, DEPLOY, ARTIFACTORY, JENKINS

Las opiniones expresadas por los contribuidores de DZone son suyas.

Obtenga lo mejor de DevOps en su

Obtenga lo mejor de DevOps en su bandeja de entrada.

Manténgase actualizado con el boletín DevOps quincenal de DZone. VER UN EJEMPLO

SUSCRIBIR

DevOps Partner Resources

Automate Security in Your DevOps Pipeline

Sonatype



Examining the Features of a Good Log File Viewer

Scalyr



Choosing Among Log Management Tools

Scalyr



The DevOps Journey - From Waterfall to Continuous Delivery

Sauce Labs

