

Conectando Github-Jenkins-Docker

ERICK MONTES BEDOLLA

Docker y Jenkins ▾

Docker es una plataforma que facilita la creación y diseño de contenedores, envío de imágenes y creación de versión de la imagen.

Los contenedores incluyen todo lo necesario para que un software se ejecute. Se usan como máquinas virtuales pero suelen utilizarse para levantar máquinas independientes con sistemas operativos muy ligeros.

Los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

Jenkins por su parte es un servidor cuyo uso es compilar y probar proyectos de software de una forma continua, facilitando el integrar cambios de un proyecto e ir entregando al usuario nuevas versiones. Útil en el proceso de Continuous Integration.

Desplegar Jenkins en un contenedor Docker ▾

Después de instalar tanto Dockers como Jenkins en la computadora, se elaboró un programa simple que se subió a Github en un repositorio * <https://github.com/ErickMontesDK/jenkins-docker>.

Crear contenedor ▾

Para conectar Jenkins a docker, instalamos la imagen de jenkins en docker con el cmd

```
docker pull jenkins/jenkins
```

Una vez instalada, podemos ver nombre del contenedor con el comando

```
docker images
```

En mi caso, el contenedor tiene el nombre `jenkins/jenkins tag=latest`

```
erick ~ 19:03 docker images
REPOSITORY          TAG          IMAGE ID        CREATED         SIZE
docker101tutorial   latest       9f15ef66f51d    3 hours ago    28.9MB
jenkins/jenkins      latest       b0f117a7a48f    2 days ago     469MB
jenkins/jenkins      lts-jdk11    25fa92c47840    3 days ago     463MB
alpine/git           latest       42a1cda0ba24    4 weeks ago    43.6MB
```

Para ejecutar el contenedor, se escribe el comando:

```
docker run -p 8080:8080 -p 5000:5000 -d -v jenkins_home:/var/jenkins_home REPOSITORY:TAG
```

Si se ejecutó con éxito, ya **nos podemos conectar al localhost de Jenkins :8080**.

Normalmente pide una contraseña para ingresar por primera vez. La contraseña se obtiene con los siguientes pasos:

- Obtiene el id de nuestros contenedores

```
docker ps
```

- Ingresamos al docker shell. *Solo me funcionó el windows powershell, no trabajaba en git bash*

```
docker exec -i -t <container_id> /bin/bash
```

- Nos da la contraseña para entrar a Jenkins

```
jenkins@<container_id>:/$ cat /var/jenkins_home/secrets/initialAdminPassword
```

```
PS C:\Users\erick> docker ps
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORTS
e9a634d5b4ea	jenkins/jenkins:latest	"/usr/bin/tini -- /u..."	serene_panini	18 minutes ago	Up 18 minutes	0.0.0.0:5000->5000/tcp, 0.0.0.0:8080->8080/tcp, 50000/tcp

```
PS C:\Users\erick> docker exec -i -t e9a634d5b4ea /bin/bash
jenkins@e9a634d5b4ea:/$ cat /var/jenkins_home/secrets/initialAdminPassword
ba16f003b05246628f25826cf7e94b5d
```

📘 Estructura del repositorio ▾

El repositorio consta de dos archivos ubicados en la raíz.

El archivo **helloWorld.sh** que consta de solo enviar un "Hello World!!!" en consola

```
echo Hello World!!!
```

El archivo Jenkinsfile, que especifica el proceso de pipeline que ejecutará Jenkins.

```
pipeline {
    agent any

    stages{
        stage(""){
            steps{
                echo 'Building the application'
            }
        }
        stage("test"){
            steps{
                sh 'chmod +x helloWorld.sh'
                sh './helloWorld.sh'
            }
        }
        stage("deploy"){
            steps{
                echo 'Deploying the application'
            }
        }
    }
}
```

📘 Uso de Ngrok ▾

Dado que Jenkins esta en un servidor local, no podremos conectarlo a Github sin una url que le de acceso, para ello ocuparemos Ngrok.

Una vez descargado, iniciamos el programa y nos abre una consola, primero hay que configurar un token de autenticación. Escribimos el comando

```
ngrok config add-authtoken AQUIVAEHTOKEN
```

Con esto ya podremos usar Ngrok. Escribimos el comando en la consola

```
ngrok http 8080 --host-header="localhost:8080"
```

Con este comando, nos da una url para el servidor en el puerto 8080, en este caso, el servidor de Jenkins.

Existe un segundo comando más sencillo, pero que tiene fallos de conexión en algunas aplicaciones para ciertos usuarios, como fue mi caso en el que no funcionó

```
ngrok http 8080
```

Al hacer la conexión, nos da el url con el que podremos entrar remotamente a nuestro servidor de Jenkins, y que vamos a ocupar para conectar con Github

Visit <http://localhost:4040/> to inspect, replay, and modify your requests

```
Session Status      online
Account             Erick Montes (Plan: Free)
Version             3.1.0
Region              United States (us)
Latency             88ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://ac55-2806-266-485-10e2-18d6-32dc-5d96-da71.ngrok.io -> http://localhost:8080

Connections
t1l    opn    rt1    rt5    p50    p90
9      0      0.00   0.00   30.05   30.46
```

Conectar con el repositorio ▾



Primeramente, hay que conseguir accesos por parte de Jenkins, dando [click en nuestra cuenta y Configuración](#), en el apartado de API Token, se presiona "Add New Token" y se guarda.

Clave del API (Token)

Current token(s) ?



Token created at 2022-11-27T18:08:48+00:00

Created 0 day(s) ago

 Never used 


Token created at 2022-11-27T18:08:48+00:00

Created 1 day(s) ago

 Never used 

Default name

Generate



Add new Token

Se abre el repositorio en Github, en el apartado de [Settings](#), luego en [Webhook](#), y se agrega un nuevo Webhook

Ahi se colocan en payload en url generado por NGROK mas github-webhook/

```
https://urldengrok/github-webhook/
```

Las demás opciones como en la imagen

Webhooks / Manage webhook

Settings

Recent Deliveries

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (`JSON`, `x-www-form-urlencoded`, etc). More information can be found in our [developer documentation](#).

Payload URL *

`https://ac55-2806-266-485-10e2-18d6-32dc-5d96-da71.ngrok.io/g`


Content type

`application/json`

Secret

If you've lost or forgotten this secret, you can change it, but be aware that any integrations using this secret will need to be updated. — [Change Secret](#)

SSL verification

 By default, we verify SSL certificates when delivering payloads.

☒ Enable SSL verification ☐ Disable (not recommended)

Which events would you like to trigger this webhook?

- ☒ Just the push event.
- ☐ Send me everything.
- ☐ Let me select individual events.

☒ Active

We will deliver event details when this hook is triggered.

Update webhook

Delete webhook

Se ocupa otro Token por parte de Github, en la ruta [Perfil de Github-->Settings-->Developer Settings-->Personal access tokens](#), seleccionamos los checkbox de los permisos para [repo y user](#) y generamos un nuevo token y copiamos

GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Beta

Tokens (classic)

Personal access tokens (classic)

Generate new token

Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

alle — <i>admin:enterprise, admin:pgp_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages</i>	Last used within the last week	Delete
Expires on Wed, Dec 21 2022.		
ending — <i>admin:enterprise, admin:pgp_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages</i>	Last used within the last week	Delete
Expires on Wed, Dec 21 2022.		
jenkins-webhook — <i>repo, user</i>	Last used within the last week	Delete
Expires on Wed, Dec 21 2022.		
jenkins — <i>admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, notifications, repo, workflow, write:packages</i>	Last used within the last week	Delete
Expires on Fri, Feb 17 2023.		

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

De regreso en Jenkins, en la ruta de **Administrar Jenkins-->Configurar el Sistema-->Github** agregamos un servidor de Github *Add Github server*, en Credenciales agregamos el token de Github como **Secret Text**, comprobamos la conexión y aceptamos la casilla de **Manage Hooks**

GitHub

GitHub Servers ?

≡ GitHub Server ?

Name ?

API URL ?

https://api.github.com

Credentials ?

git-final

+ Add

☒ Manage hooks


Avanzado...

Add GitHub Server ▾

Ya dentro de la interfaz de Jenkins, creamos nueva tarea, donde asignamos un nombre y que el proyecto sea de **Pipeline**


Enter an item name

test-ended




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

Gestiona actividades de larga duración que pueden abarcar varios agentes de construcción. Apropiado para construir pipelines (conocidas anteriormente como workflows) y/o para la organización de actividades complejas que no se pueden articular fácilmente con tareas de tipo freestyle.



Crear un proyecto multi-configuración

En la configuración general, se selecciona la opción de Github y escribimos el url del repositorio a conectar

Panel de Control > test-docker-github > Configuration

Configure

General

Configurar el origen del código fuente

Disparadores de ejecuciones

Entorno de ejecución

Build Steps

Acciones para ejecutar después.

General

Descripción

[Plain text] [Visualizar](#)

☐ Desechar ejecuciones antiguas ?

☐ Esta ejecución debe parametrizarse ?

☒ GitHub project

Project url ?

https://github.com/ErickMontesDK/jenkins-docker/

Avanzado...

En la sección de Build Triggers, se selecciona **Github hook trigger for GITScm polling**

Build Triggers

- ☐ Construir tras otros proyectos ?
- ☐ Ejecutar periódicamente ?
- ☐ GitHub Branches
- ☐ GitHub Pull Requests ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Consultar repositorio (SCM) ?
- ☐ Periodo de espera ?
- ☐ Lanzar ejecuciones remotas (ejem: desde 'scripts') ?

Y en la sección de Pipeline, se seleccionan las mismas opciones de la imagen. En credenciales pide ingresar el usuario de github y el token que creamos previamente

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/ErickMontesDK/jenkins-docker

Credentials ?

ErickMontesDK/***** (git-ended)

+ Add

Avanzado...

Después, escribimos el nombre de la branch que va usarse. En script path, escribimos la ruta de nuestro archivo Jenkinsfile. En mi caso, solo escribo el nombre porque esta en la raíz del repositorio

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Navegador del repositorio ?

(Auto)

Additional Behaviours

Añadir ▾

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

Ejecución ▾

Ya está todo configurado, solo es necesario hacer la primer construcción manualmente. Una vez que se haya ejecutado, podemos hacer un push a nuestro repositorio y Jenkins iniciará el proceso de construir de nuevo el programa

 Vista de "builds" ▾

Status

Pipeline test-ended

</> Changes

▶ Construir ahora

⚙ Configurar

🗑 Borrar Pipeline

🔍 Full Stage View

🔄 GitHub

✎ Rename

❓ Pipeline Syntax

📄 GitHub Hook Log

Stage View

☀ Historia de tareas **Tendencia** ▼

🔍 Filter builds... /

- ✓ #5 22 nov. 2022 2:04
- ✓ #4 22 nov. 2022 1:39
- ✓ #3 22 nov. 2022 1:22
- ✓ #2 22 nov. 2022 1:18
- ✓ #1 22 nov. 2022 1:13

📡 Atom feed Para Todos
📡 Atom feed para los errores

Average stage times:
(Average full run time: ~8s)

#5
Nov 22 02:04
1 commit

#4
Nov 22 01:39
1 commit

#3
Nov 22 01:22
1 commit

#2
Nov 22 01:18
No Changes

#1
Nov 22 01:13
No Changes

Declarative: Checkout SCM	error	test	deploy
2s	449ms	116ms	156ms
4s	1s	249ms	354ms
1s	111ms	87ms	190ms
1s	155ms	71ms	89ms
1s	142ms	74ms	
2s	124ms	101ms	84ms

Success

📊 Logs

☰ Console Output ▼



Salida de consola

```
Started by user Erick Montes
Obtained Jenkinsfile from git https://github.com/ErickMontesDK/jenkins-docker
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/test-ended
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
using credential git-ended
> git rev-parse --resolve-git-dir /var/jenkins_home/workspace/test-ended/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/ErickMontesDK/jenkins-docker # timeout=10
Fetching upstream changes from https://github.com/ErickMontesDK/jenkins-docker
> git --version # timeout=10
> git --version # 'git version 2.30.2'
using GIT_ASKPASS to set credentials git-ended
> git fetch --tags --force --progress -- https://github.com/ErickMontesDK/jenkins-docker +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision f434cf016d325d043be5d3bf0823221924377974 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f f434cf016d325d043be5d3bf0823221924377974 # timeout=10
Commit message: "Update helloworld.sh"
> git rev-list --no-walk a1f65661edaa0961622d7c6d43865351292eabc9 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (error)
```

```
[Pipeline] { (error)
[Pipeline] echo
Building the application
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (test)
[Pipeline] sh
+ chmod +x helloworld.sh
[Pipeline] sh
+ ./helloworld.sh
Hello World!!!
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (deploy)
[Pipeline] echo
Deploying the application
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

```

Session Status      online
Account             Erick Montes (Plan: Free)
Version             3.1.0
Region              United States (us)
Latency             88ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://ac55-2806-266-485-10e2-18d6-32dc-5d96-da71.ngrok.io -> http://localhost:8080

```

online

```
Erick Montes (Plan: Free)
3.1.0
United States (us)
88ms
http://127.0.0.1:4040
https://ac55-2806-266-485-10e2-18d6-32dc-5d96-da71.ngrok.io -> http://localhost:8080
```

t1	opn	rt1	rt5	p50	p90
9	0	0.00	0.00	30.05	30.46

100

```
302 Found
302 Found
302 Found
302 Found
302 Found
302 Found
302 Found
302 Found
302 Found
```