

# **Evidencia**

## **Laboratorio 4 - DevOps**



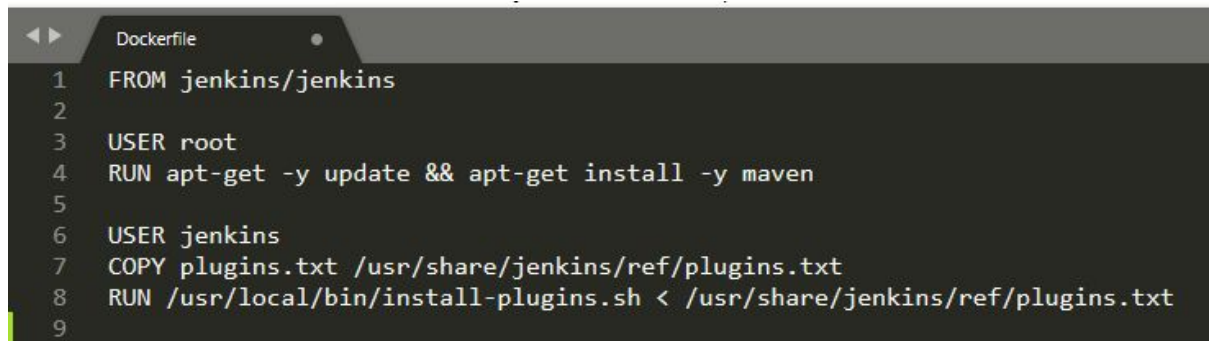
José Fernando Morantes Florez  
Harold Nicolas Saavedra Alvarado

Universidad Nacional de Colombia  
Facultad de Ingeniería  
Departamento de Ingeniería de Sistemas e Industrial  
Docente Yury Yineth Niño Roa  
2019-2

# Construcción de la imagen de Jenkins usando Docker

## Dockerfile

El primer paso a realizar para construir una imagen con Jenkins, a partir de la imagen de jenkins descargada desde DockerHub, es escribir el fichero Dockerfile que contendrá las instrucciones para construir dicha imagen.

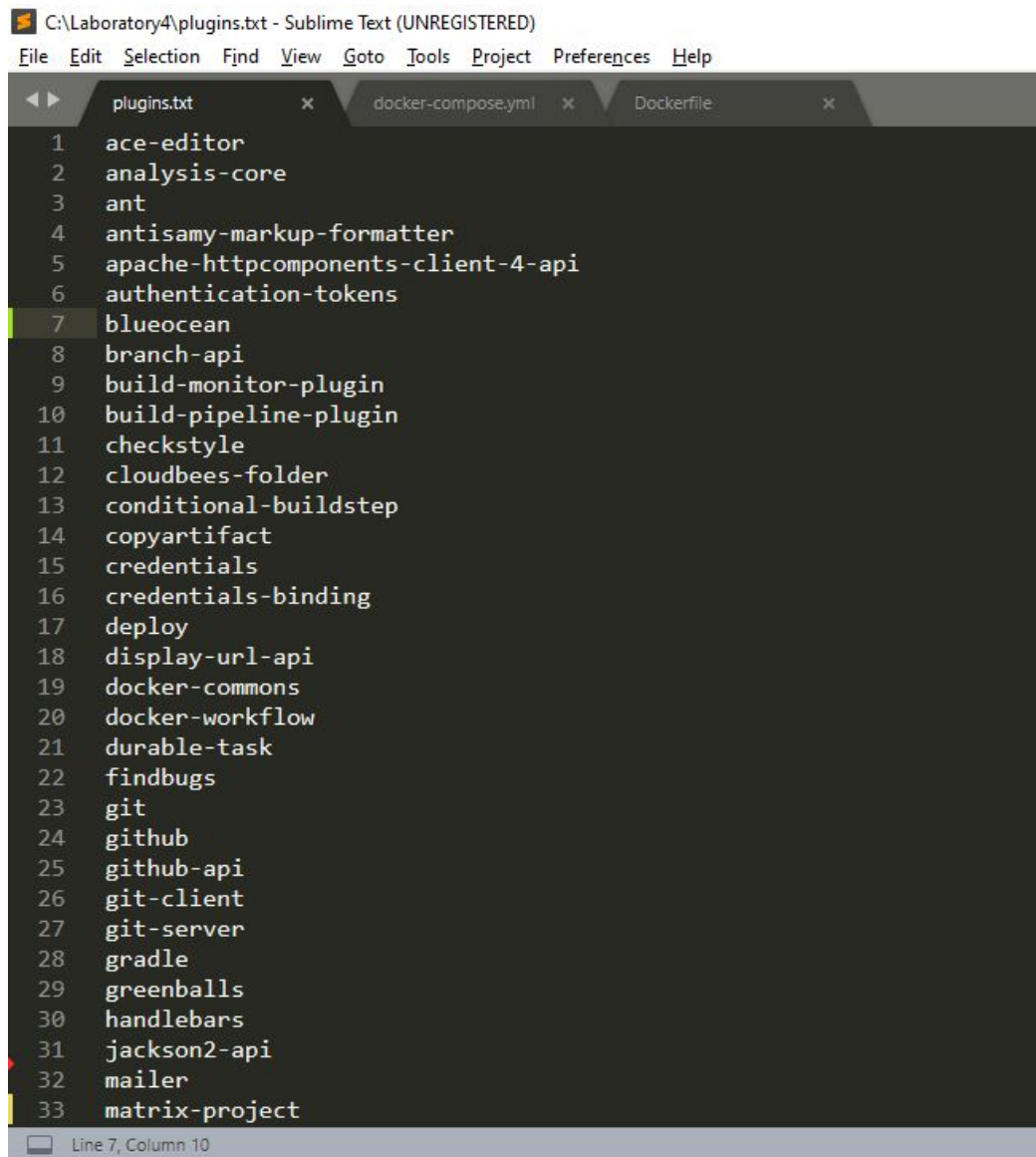


```
1 FROM jenkins/jenkins
2
3 USER root
4 RUN apt-get -y update && apt-get install -y maven
5
6 USER jenkins
7 COPY plugins.txt /usr/share/jenkins/ref/plugins.txt
8 RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/ref/plugins.txt
9
```

- En la línea 1 se indica la imagen base (jenkins/jenkins descargada desde DockerHub) desde la cual se parte para construir la nueva imagen.
- En la línea 3 indicamos que el usuario de ejecución es root.
- Tras ejecutarse los comandos apt-get para la instalación de maven( a pesar de que no usamos maven en este laboratorio debemos instalar las dependencias necesarias para que esto no sea incongruente con los plugins elegidos en plugin.txt )en la línea 4, el resto de comandos pueden ejecutarse con el usuario jenkins.
- En la línea 6 establecemos el nuevo usuario de ejecución.
- En la línea 7 copiamos el fichero plugins.txt en el directorio /usr/share/jenkins/ref/, este fichero contiene la lista de plugins que queremos que se instalen en Jenkins. Más adelante se especificará el contenido de este archivo
- En la última línea se ejecuta el script install-plugins.sh cuya entrada es la lista de plugins incluidos en el fichero plugins.txt y los instala en Jenkins.

## plugins.txt

Como se indicó en la creación del Dockerfile se crea un archivo llamado “plugins.txt” en el cual se incluyen los plugins que se desean instalar. En la imagen a continuación no se muestra la lista completa de plugins, en nuestro caso elegimos los plugins más usados para jenkins en caso de que se desee escalar el uso de este contenedor .



```
C:\Laboratory4\plugins.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
plugins.txt x docker-compose.yml x Dockerfile x
1 ace-editor
2 analysis-core
3 ant
4 antisamy-markup-formatter
5 apache-httpcomponents-client-4-api
6 authentication-tokens
7 blueocean
8 branch-api
9 build-monitor-plugin
10 build-pipeline-plugin
11 checkstyle
12 cloudbees-folder
13 conditional-buildstep
14 copyartifact
15 credentials
16 credentials-binding
17 deploy
18 display-url-api
19 docker-commons
20 docker-workflow
21 durable-task
22 findbugs
23 git
24 github
25 github-api
26 git-client
27 git-server
28 gradle
29 greenballs
30 handlebars
31 jackson2-api
32 mailer
33 matrix-project
Line 7, Column 10
```

En la lista del archivo plugins.txt vale la pena resaltar los siguientes plugins:

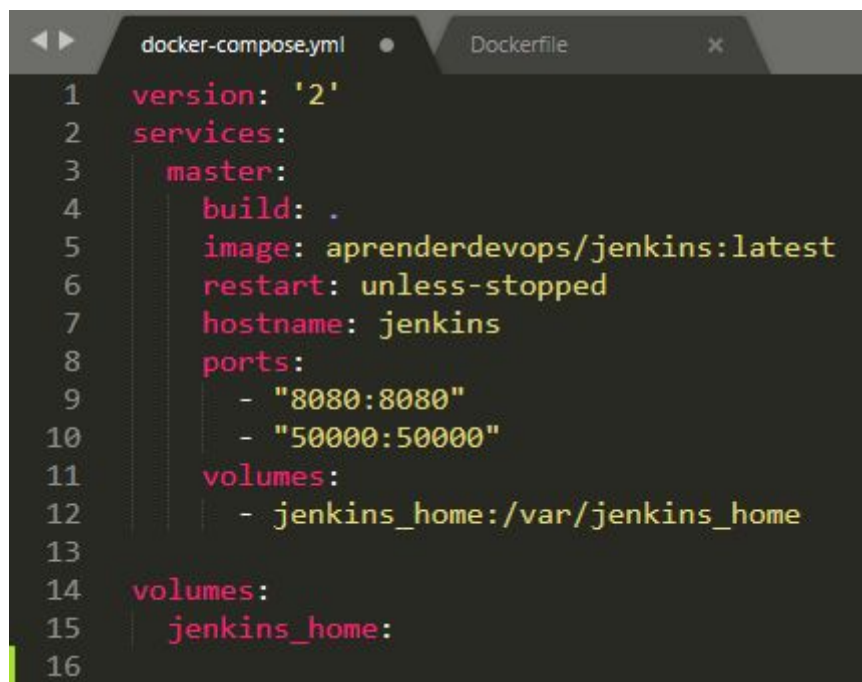
- **blueocean:**  
es un plugin para jenkins el cual modela y representa de manera sencilla y entendible el proceso de creación de pipelines en jenkins

- **pipeline-aws:**

proporciona funcionalidad disponible a través pipeline-compatible steps, los cuales son secuencias de código que se ejecutarán en cada de stage del pipeline, uno de los usos destacados de este plugin es la comunicación con instancias del servicio S3 de AWS, recordemos que un S3 es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, entre otras cosas.

## **docker-compose.yml**

Para una mayor facilidad en la construcción de la imagen y del contenedor de Docker se escribe el siguiente archivo docker-compose:



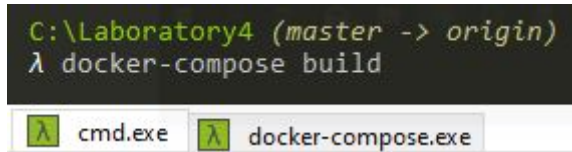
```
1  version: '2'
2  services:
3    master:
4      build: .
5      image: aprenderdevops/jenkins:latest
6      restart: unless-stopped
7      hostname: jenkins
8      ports:
9        - "8080:8080"
10       - "50000:50000"
11      volumes:
12        - jenkins_home:/var/jenkins_home
13
14  volumes:
15    jenkins_home:
16
```

En este archivo se indica:

- **Línea 4:** directorio sobre el que se va a construir la imagen. en nuestro caso, el mismo directorio en el que se encuentra el fichero docker-compose.yml (Laboratory4) que a su vez es la carpeta del repositorio de github.
- **Línea 5:** nombre de la imagen, en nuestro caso aprenderdevops/jenkins:latest
- **Línea 6:** establece que se reinicie el contenedor a menos que se detenga explícitamente o el motor de Docker se detenga o reinicie.
- **Línea 7:** hostname del contenedor.
- **Líneas 9 y 10:** configuraciones de los puertos 8080 y 50000 respectivamente para ejecutar la imagen.
- **Líneas de la 12 a la 15:** definición del volumen jenkins\_home el cual se utiliza para que los cambios que se realicen en la configuración de Jenkins persistan incluso tras la destrucción del contenedor.

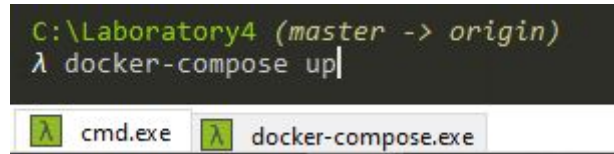
## Construcción de la imagen del contenedor

```
C:\Laboratory4 (master -> origin)
λ docker-compose build
```



## Ejecución del contenedor y su imagen

```
C:\Laboratory4 (master -> origin)
λ docker-compose up
```

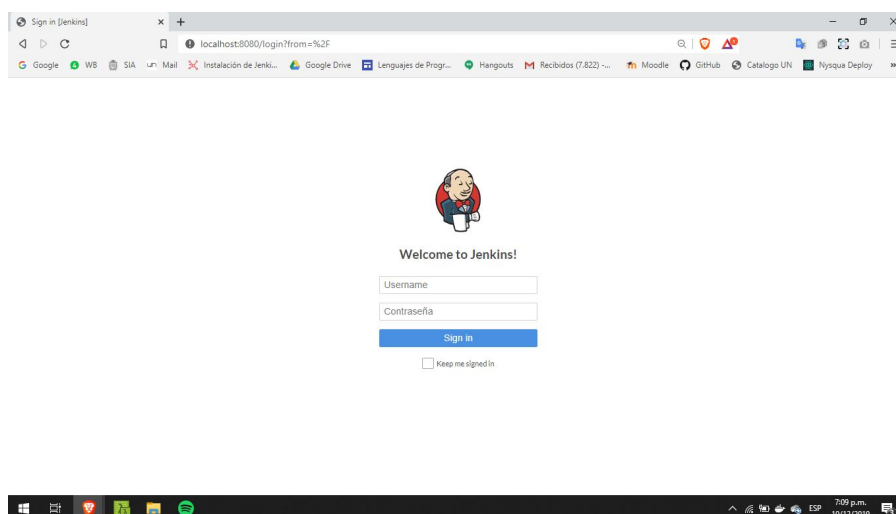


## Entorno de jenkins

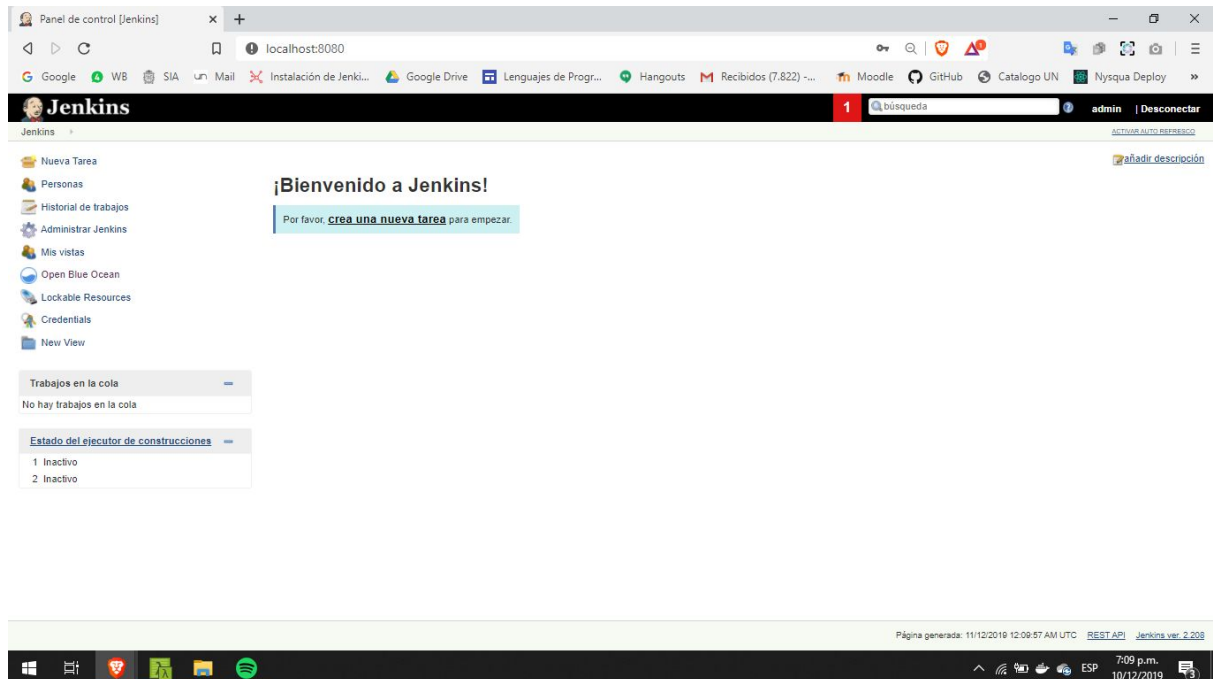
Una vez se creó correctamente el contenedor de Docker con la imagen de jenkins y al ejecutarlo en la dirección localhost:8080 se dara la opcion de instalar manualmente plugins que el sistema recomienda y nos brinda la posibilidad de crear usuarios, en nuestro caso omitimos estos pasos debido a que cargamos los plugins desde la ejecución del contenedor y adoptamos únicamente el manejo de el usuario administrador (usuario otorgado por defecto).

Posteriormente al correr nuevamente el entorno de jenkins nos pedirá usuario y contraseña, por defecto el usuario administrador se expresa con el username “admin” y la contraseña se puede obtener ejecutando el siguiente comando, lo cual (mediante el “exec”) ejecuta lo que se indique en el contenedor que se esté ejecutando)

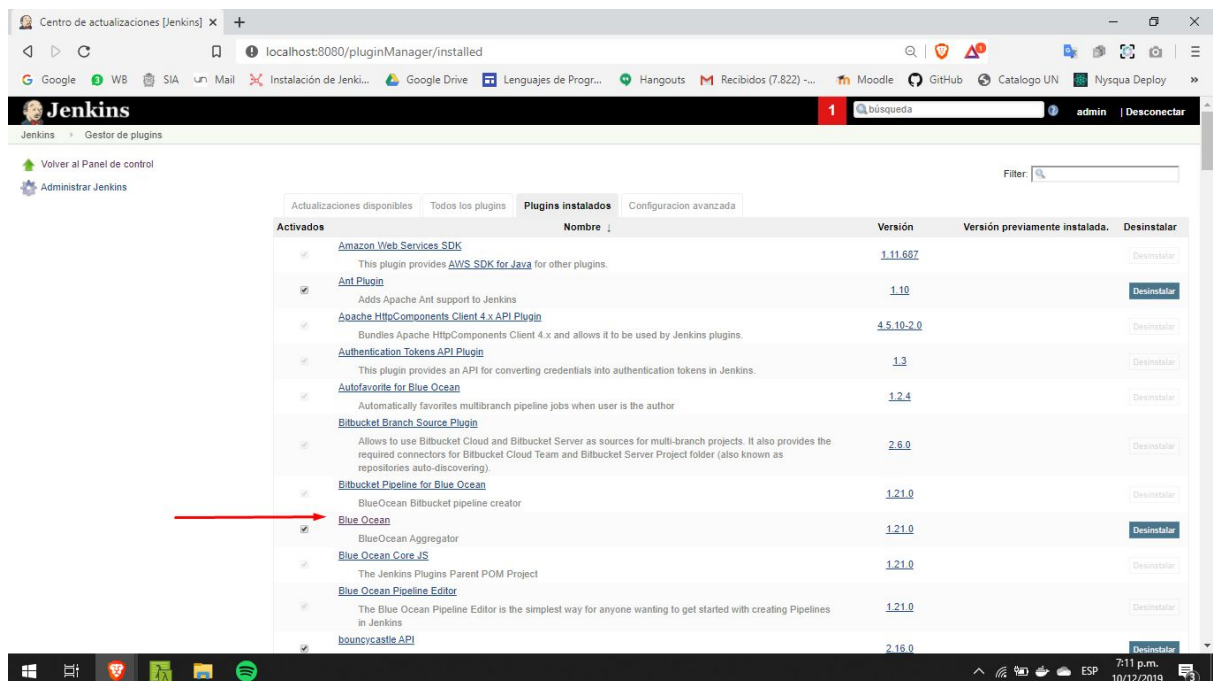
```
C:\Laboratory4 (master -> origin)
λ docker exec -it laboratory4_master_1 cat /var/jenkins_home/secrets/initialAdminPassword
169a19cabcd049e595465b1148951192
```



Una vez iniciada la sesión en Jenkins con los datos del administrador nos encontramos con la pantalla principal, también conocida como el panel de control

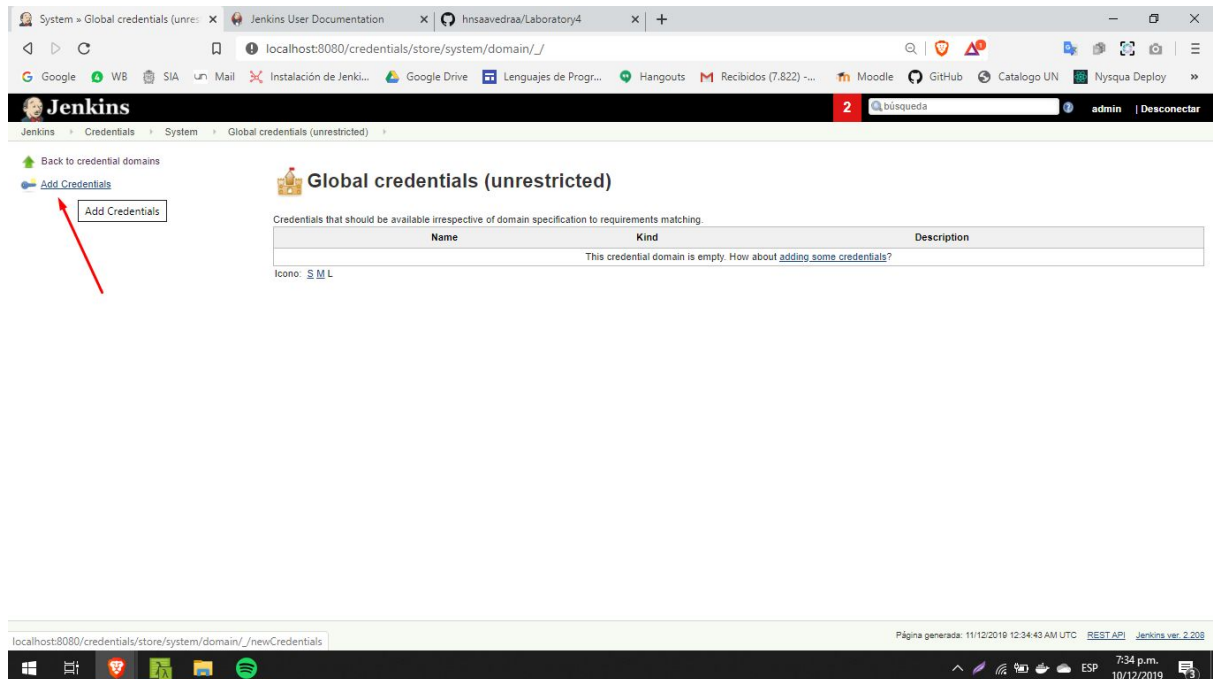


A partir de este punto revisamos que los plugins previamente declarados para su instalación, especialmente BlueOcean y pipeline-aws estén correctamente instalados, esto se puede verificar desde la pestaña de plugins > installed plugins



Ahora hablaremos un poco de la configuración del plugin pipeline- aws, que en primera instancia funciona como la mayoría de plugin orientados a dar conexión con servicios de

AWS, para aplicar esta configuración debemos dirigirnos al apartado de “Credentials -> System -> Add Credentials”

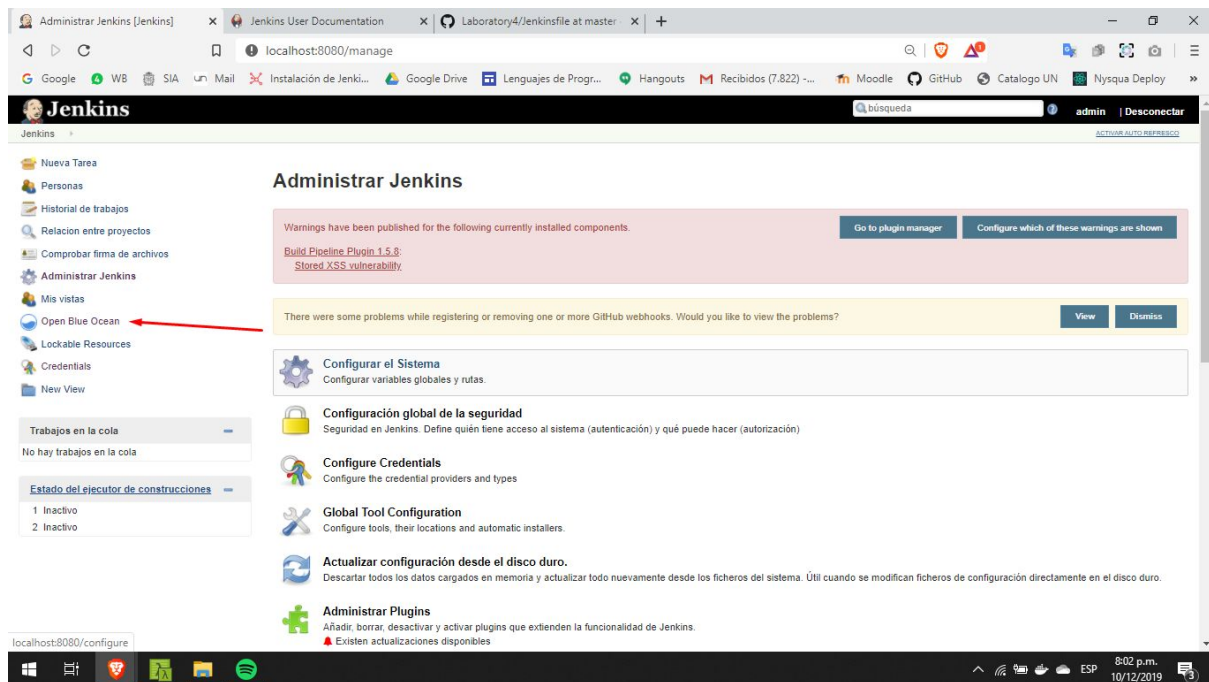


Proporcione las credenciales AWS IAM para permitir que Jenkins Pipeline AWS Plugin acceda a su S3 Bucket. Por tanto es necesario indicar el access key ID creado y el secret access key.

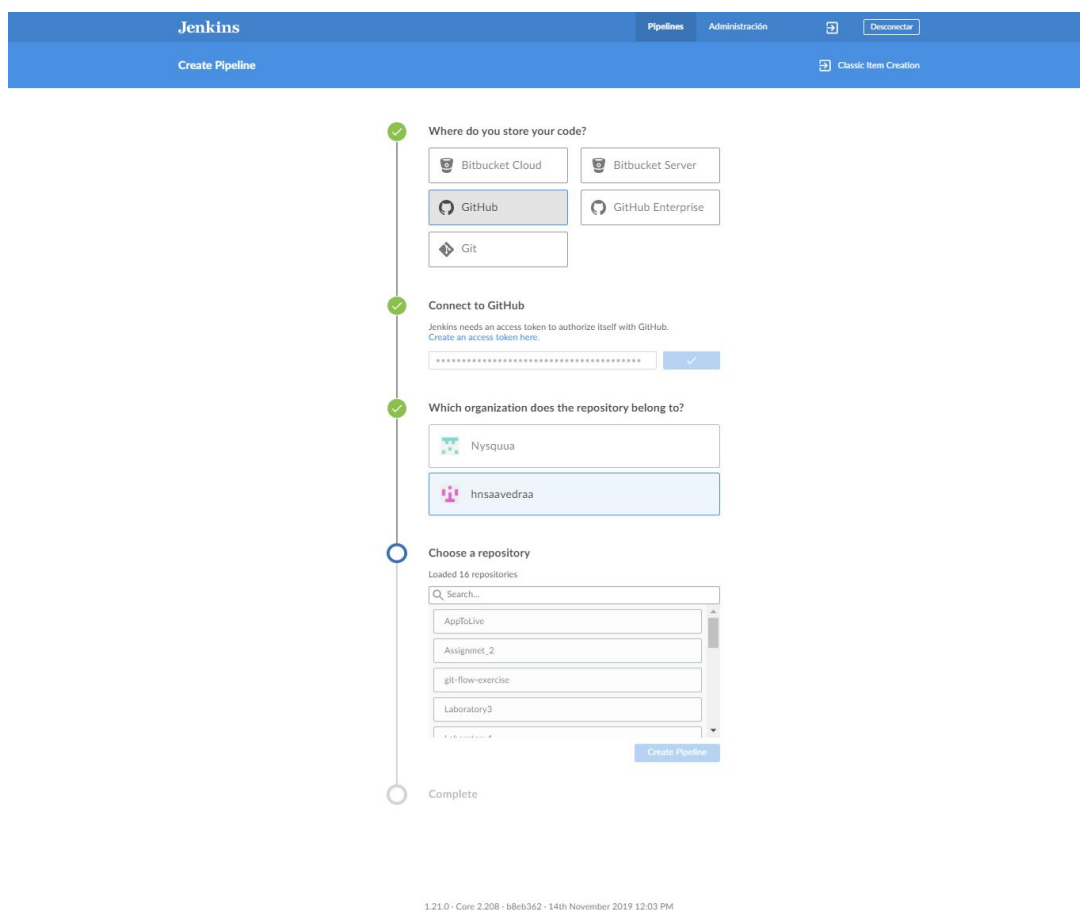
The screenshot shows the 'Add Credentials' form in Jenkins. The 'Kind' dropdown is set to 'AWS Credentials'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'ID', 'Description', 'Access Key ID', and 'Secret Access Key' fields are empty. The 'IAM Role Support' checkbox is unchecked. There is an 'Avanzado...' button on the right and an 'OK' button at the bottom left.

A continuación crearemos el pipeline haciendo uso del plugin BlueOcean, para ello vamos a la pestaña Open Blue Ocean





Una vez allí nos pedirá enlazar el repositorio que manejaremos para el pipeline, en este caso usamos el servicio de alojamiento de GitHub, para esto será necesario crear un token de conexión en nuestra cuenta de Github e indicar el repositorio a trabajar





Ahora tendremos que crear y subir a nuestro repositorio un archivo Jenkinsfile, este es un archivo donde indicaremos la estructura paso a paso o estado a estado del pipeline como lo vemos a continuación, también se define la secuencia de código que se ejecutará en cada uno de los estados, en nuestro caso usamos comandos de prueba que se usarían en un proyecto creado con Node JS

```
72 lines (62 sloc) | 1.64 KB

1 pipeline {
2   agent {
3     docker {
4       image 'node:10-alpine'
5       args '-p 20001-20100:3000'
6     }
7   }
8 }
9 stages {
10  stage('Install Packages') {
11    steps {
12      sh 'npm install'
13    }
14  }
15
16  stage('Test and Build') {
17    parallel {
18      stage('Run Tests') {
19        steps {
20          sh 'npm run test'
21        }
22      }
23
24      stage('Create Build Artifacts') {
25        steps {
26          sh 'npm run build'
27        }
28      }
29    }
30  }
31 }
```

Si bien las primeras etapas son bastante simples (partiendo del hecho de que estos scripts están creados en nuestro proyecto), para las últimas etapas debemos definir comandos propios del plugin pipeline-aws que definirán cómo se realizará el deploy

```

35     stage('Staging') {
36         when {
37             branch 'staging'
38         }
39         steps {
40             withAWS(region: '<your-bucket-region>', credentials: '<AWS-Staging-Jenkins-Credential-ID>') {
41                 s3Delete(bucket: '<bucket-name>', path: '**/*')
42                 s3Upload(bucket: '<bucket-name>', workingDir: 'build', includePathPattern: '**/*');
43             }
44             mail(subject: 'Staging Build', body: 'New Deployment to Staging', to: 'jenkins-mailing-list@mail.com')
45         }
46     }
47     stage('Production') {
48         when {
49             branch 'master'
50         }
51         steps {
52             withAWS(region: '<your-bucket-region>', credentials: '<AWS-Production-Jenkins-Credential-ID>') {
53                 s3Delete(bucket: '<bucket-name>', path: '**/*')
54                 s3Upload(bucket: '<bucket-name>', workingDir: 'build', includePathPattern: '**/*');
55             }
56             mail(subject: 'Production Build', body: 'New Deployment to Production', to: 'jenkins-mailing-list@mail.com')
57         }
58     }
59 }
60 }
61 }
62 }

```

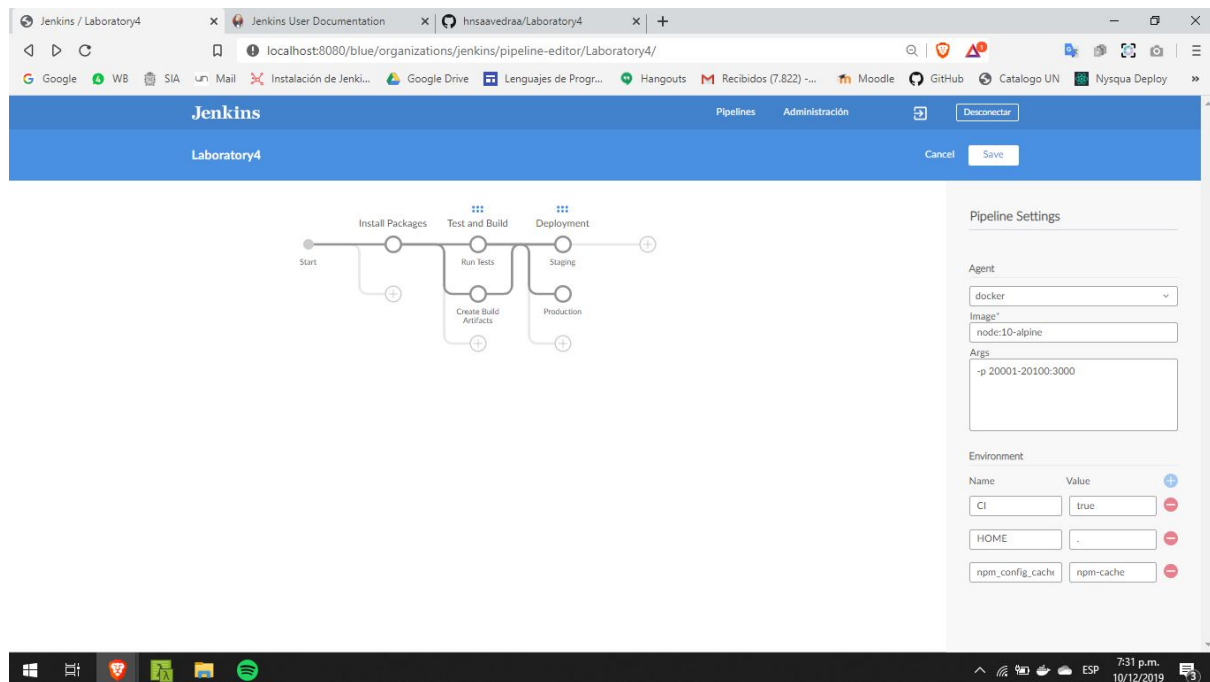
El step `withAWS` proporciona autorización para los steps anidados, a su vez este puede proporcionar información de región y perfil o dejar que Jenkins asuma un rol en la cuenta de AWS.

- El parámetro “region” establece la información de la región
- El parámetro `credentials` establece las credenciales necesarias tanto para las etapas de Staging y Production

**s3Upload** carga un archivo/carpeta desde el espacio de trabajo a un depósito S3. Si el parámetro del archivo denota un directorio, se cargará el directorio completo que incluye todas las subcarpetas. Otra forma de usarlo es con los patrones de inclusión exclusión que se aplican en el subdirectorio especificado (`workingDir`). Las opciones para configurar dichos patrones de inclusión y exclusión están separados por comas.

**s3Delete** elimina un archivo/carpeta de S3. Si la ruta termina en `/`, la ruta se interpretará como una carpeta y se eliminará todo su contenido.

Blue Ocean leerá el Jenkinsfile y genera un gráfico donde podremos ver cada uno de los pasos o etapas que conforman la pipeline.



Por último veremos cómo se ejecuta nuestro pipeline acompañado de un log en caso de errores, en nuestro caso vemos que se presenta un error debido a que en el Jenkinsfile, como ya lo mencionamos anteriormente no tenemos definido los datos de credencial que nos debería suministrar AWS.

