



Tutorial de extremo a extremo para la integración y entrega continuas por Dockerizing un Gasoducto de Jenkins

por Hüseyin Akdoğan · 30 y 17 de noviembre · DevOps Zone

Nexus Suite tiene una arquitectura única para un mundo nativo de DevOps y crea valor al principio de la cartera de desarrollo, proporciona controles contextuales precisos en cada fase y acelera la innovación de DevOps con una automatización en la que puede confiar. Lea cómo en este ebook .

La integración continua y la entrega continua son temas fundamentales en la industria del software, especialmente con las tecnologías de nube y contenedores. Las tecnologías de contenedores como Docker , Kubernetes y OpenShift y los servidores de automatización como Jenkins facilitan la administración de los flujos de trabajo de nuestros proyectos.

El repositorio que he creado responde a la pregunta de cómo administrar automáticamente el proceso de creación, prueba con la mayor cobertura y las fases de implementación.

Nuestro objetivo es garantizar que nuestro oleoducto funcione bien después de cada código. Los procesos que queremos administrar automáticamente:

- Código de pago
- Ejecutar pruebas
- Compila el código
- Ejecute el análisis de Sonarqube en el código
- Crear imagen Docker
- Empuje la imagen al Docker Hub
- Tira y ejecuta la imagen

1) Ejecución de los servicios

Como uno de los objetivos es obtener el informe Sonarqube de nuestro proyecto, deberíamos poder acceder a Sonarqube desde el servicio de Jenkins. Docker compose es la mejor opción para ejecutar servicios trabajando juntos. Configuramos nuestros servicios de aplicaciones en un archivo yaml, como se muestra a continuación.

docker-compose.yml

```

1  versión : '3.2'
2  servicios :
3    sonarqube :
4      construir :
5        contexto : sonarqube /
6      puertos :
7        - 9000: 9000
8        - 9092: 9092
9      container_name : sonarqube
10   Jenkins :
11     construir :
12       contexto : jenkins /
13     privilegiado : verdadero
14     usuario : root
15     puertos :
16       - 8080: 8080
17       - 50000: 50000
18     container_name : jenkins
19     volúmenes :
20       - / tmp / jenkins: / var / jenkins_home #Recuerde que el directorio tmp está diseñad
21       - /var/run/docker.sock:/var/run/docker.sock
22     depends_on :
23       - sonarqube

```

Las rutas de los archivos Docker de los contenedores se especifican en el atributo de contexto en el archivo docker-compose. El contenido de estos archivos es el siguiente:

sonarqube/Dockerfile

```

1  DESDE sonarqube: 6.7-alpine

```

jenkins/Dockerfile

```

1  DE jenkins: 2.60.3

```

Si ejecutamos el siguiente comando en el mismo directorio que el docker-compose.yml archivo, los contenedores Sonarqube y Jenkins girarán y se ejecutarán.

```

1  docker-compose -f docker-compose.yml up --build
2  docker ps
3  MANDO DE IMAGEN ID DE CONTENEDOR CREADO NOMBRES DE PUERTOS DE ESTADO
4  87105432d655 pipeline_jenkins      "/" bin / tini - / usr ...      Hace aproximadamente un r

```

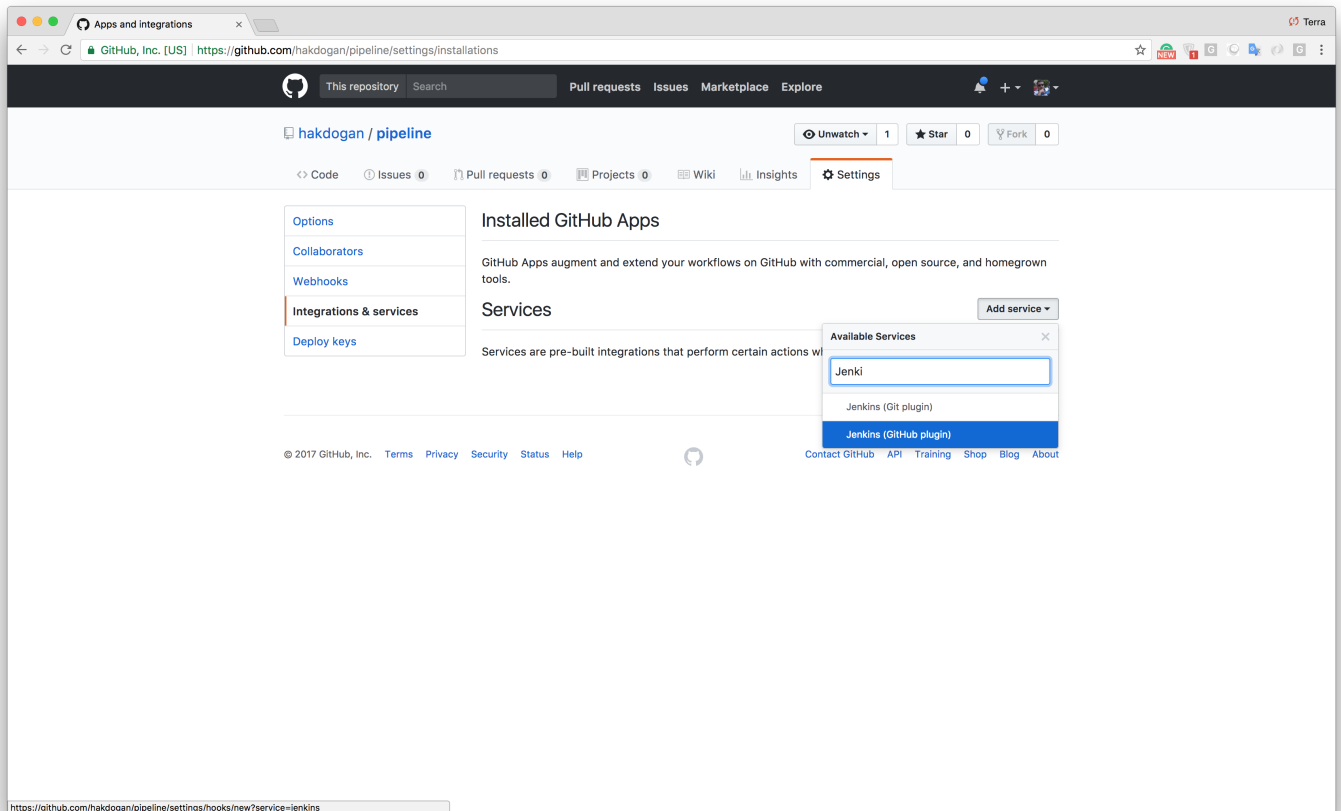
```
f5bed5ba3266 pipeline_sonarqube ".bin/run.sh"
```

Hace aproximadamente un minut

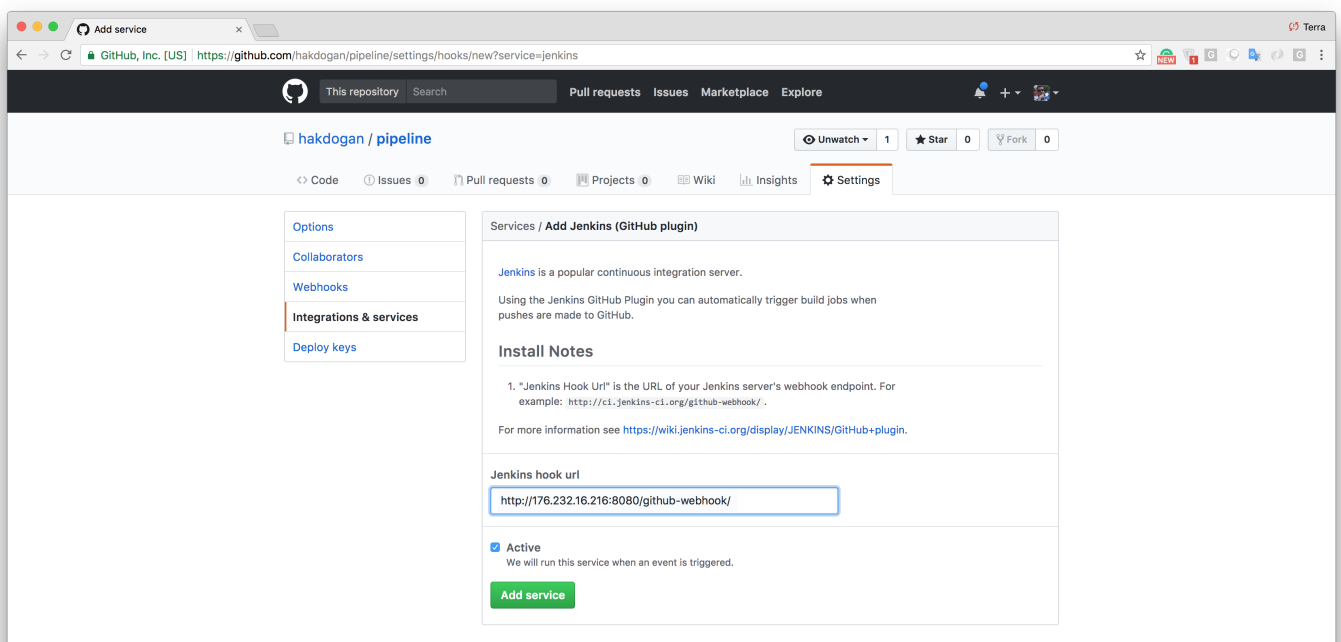
5

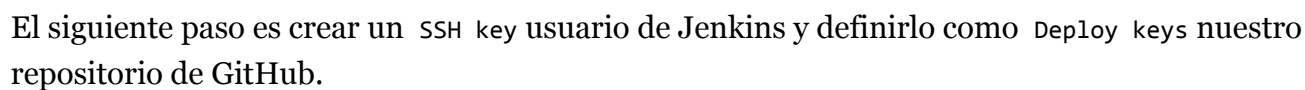
Configuración de GitHub

Definiremos un servicio en GitHub para llamar al Jenkins Github webhook porque queremos activar la canalización. Para hacer esto, vaya a *Configuración -> Integraciones y servicios*. El Jenkins Github plugin que desea mostrar en la lista de servicios disponibles, como a continuación.



Después de esto, deberíamos agregar un nuevo servicio escribiendo la URL del contenedor doblado de Jenkins junto con la `/github-webhook/` ruta.

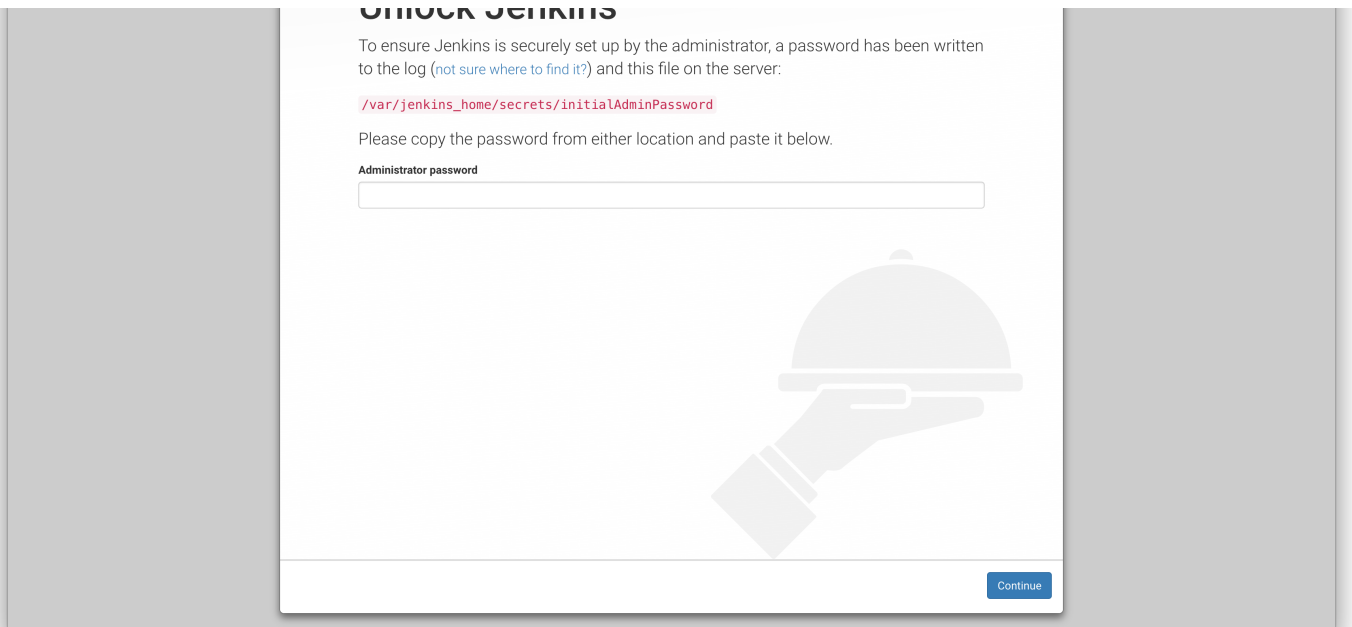




```
1 ssh git @ github.com
2 La solicitud de asignación de PTY falló en el canal 0
  Hola <tu nombre de usuario github> / <nombre del repositorio>! Se ha autenticado con éxito
3
4 La conexión a github.com se cerró.
```

Hemos configurado Jenkins en el archivo de redacción de la ventana acoplable para que se ejecute en el puerto 8080, por lo tanto, si visitamos `http://localhost:8080`, seremos recibidos con una pantalla como esta.





Necesitamos la contraseña de administrador para proceder a la instalación. Se almacena en el `/var/jenkins_home/secrets/initialAdminPassword` directorio y también se escribe como salida en la consola cuando comienza Jenkins.

```

1 Jenkins | *****
2 Jenkins |
3 Jenkins | Se requiere la configuración inicial de Jenkins. Se ha creado un usuario admini:
4 Jenkins | Por favor use la siguiente contraseña para proceder con la instalación:
5 Jenkins |
6 Jenkins | 45638c79cecd4f43962da2933980197e
7 Jenkins |
8 Jenkins | Esto también se puede encontrar en: / var / jenkins_home / secrets / initialAdmi
9 Jenkins |
10 Jenkins | *****

```

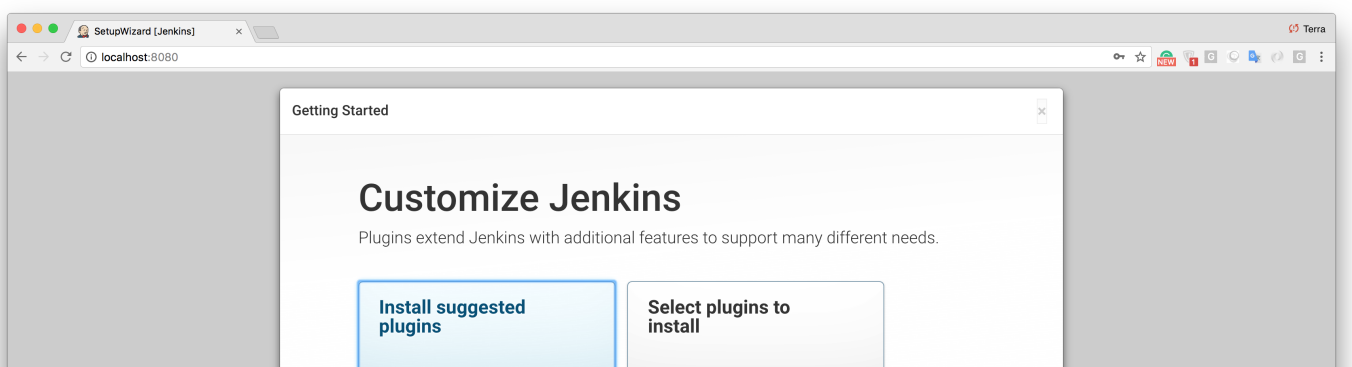
Para acceder a la contraseña desde el contenedor.

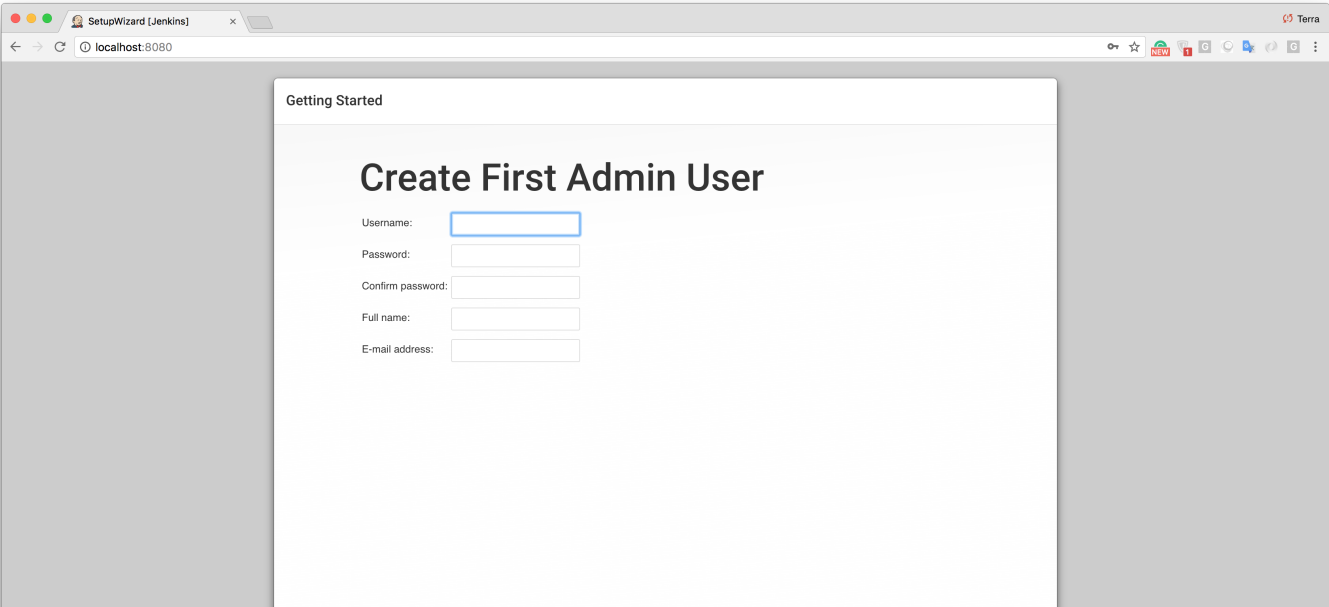
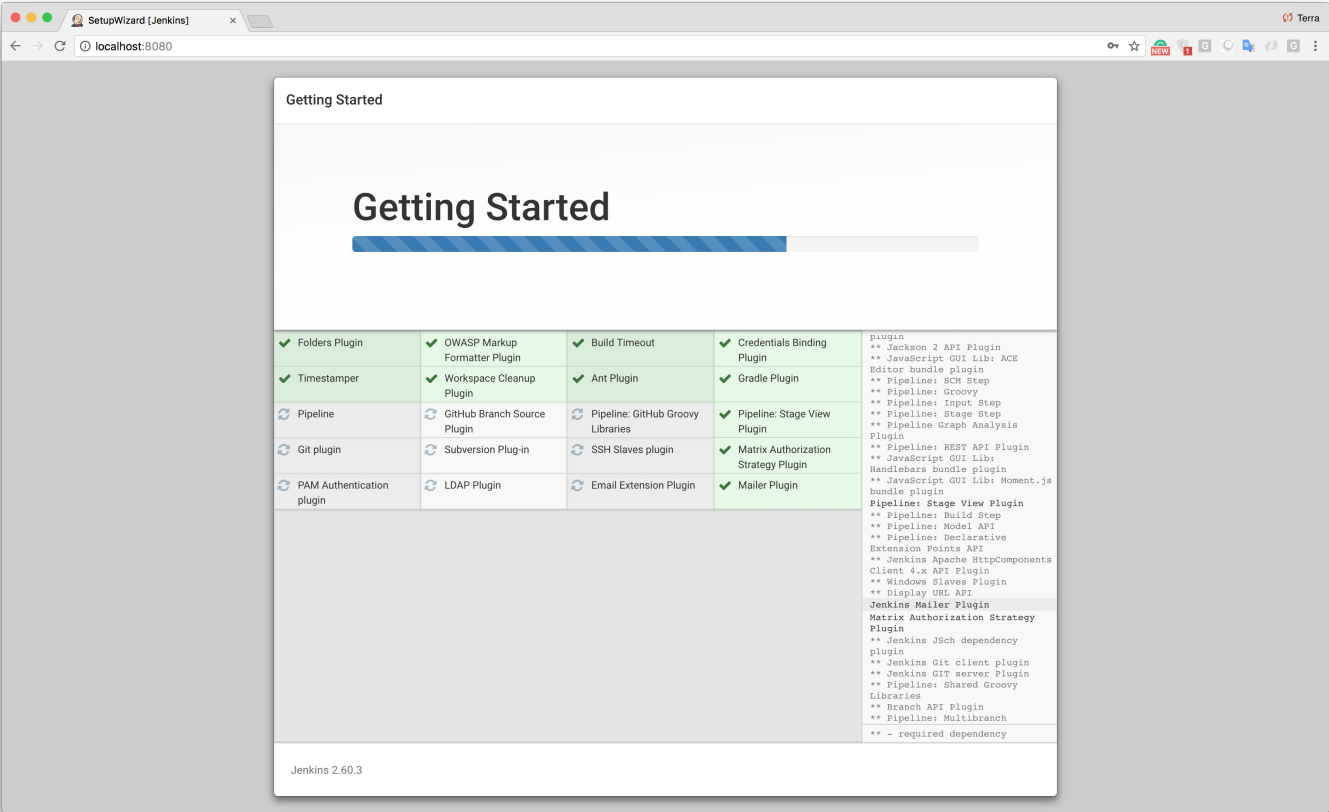
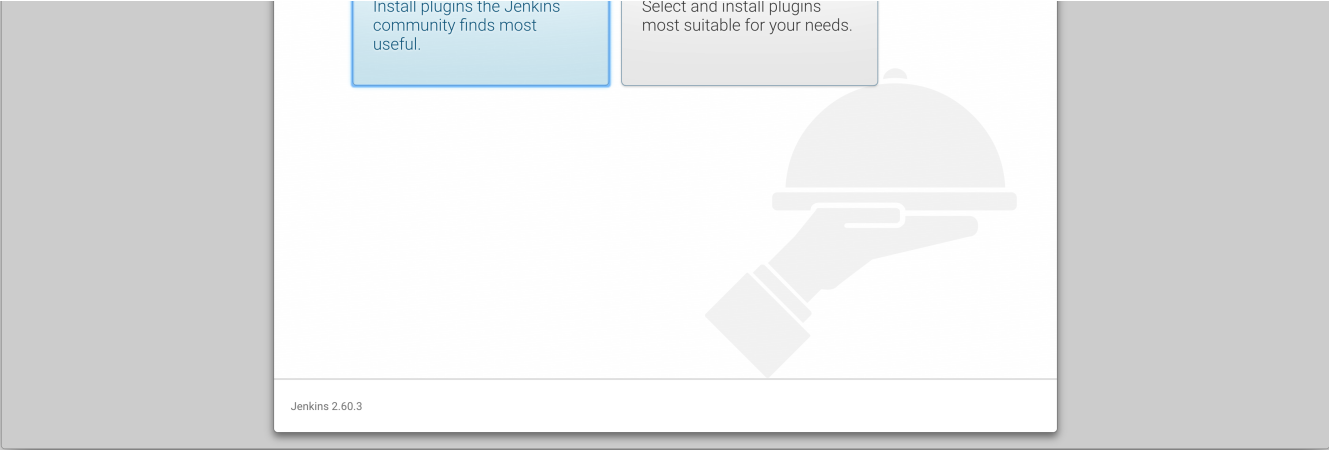
```

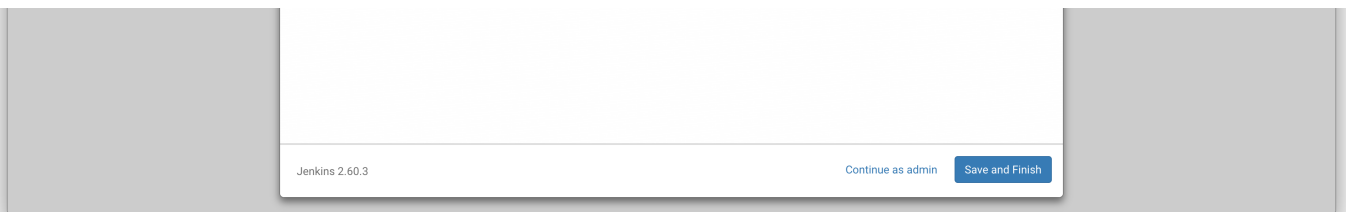
1 docker, ejecutivo , jenkins, sh
2 / $ cat / var / jenkins_home / secrets / initialAdminPassword

```

Después de ingresar la contraseña, descargaremos los complementos recomendados y definiremos un admin user .



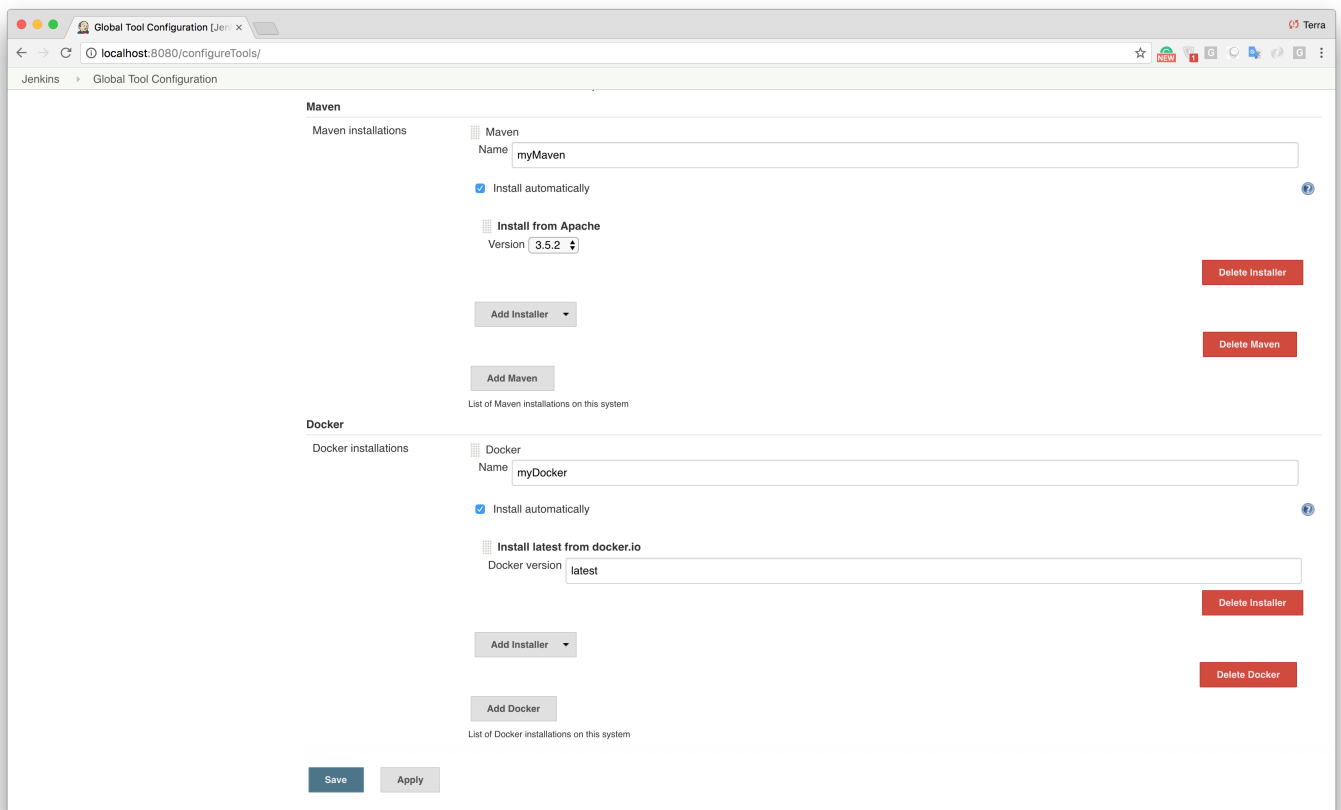




Después de hacer clic en Guardar y Terminar y comenzar a usar los botones de Jenkins, deberíamos ver la página de inicio de Jenkins. Uno de los siete objetivos enumerados anteriormente es que debemos tener la capacidad de construir una imagen en el Jenkins dockerized. Eche un vistazo a las definiciones de volumen del servicio de Jenkins en el archivo de redacción.

```
1 - /var/run/docker.sock:/var/run/docker.sock
```

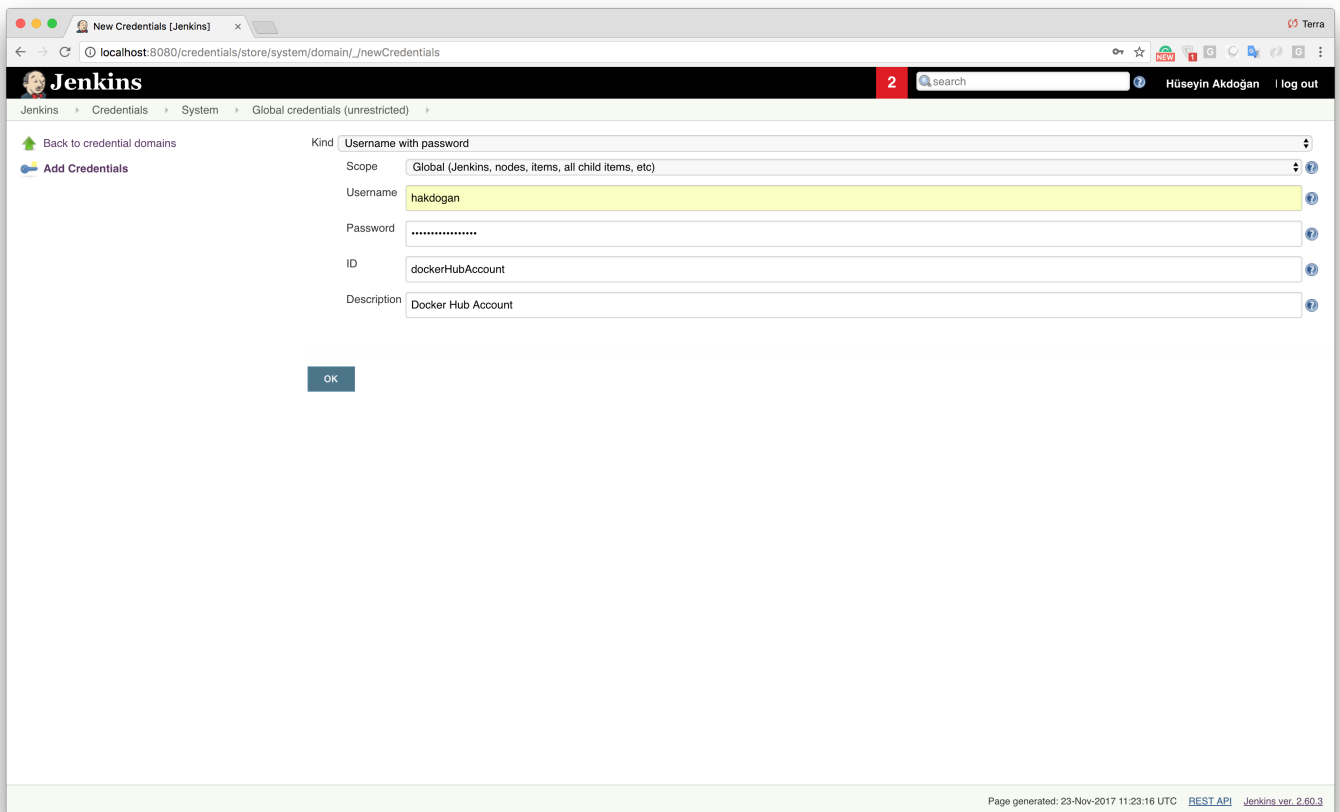
El propósito es comunicarse entre el Docker Daemon y Docker Client (lo instalaremos en Jenkins) sobre el zócalo. Al igual que el cliente de Docker, también necesitamos Maven compilar la aplicación. Para la instalación de estas herramientas, que necesitamos para llevar a cabo la Maven y Docker client configuraciones bajo *Manejo de Jenkins -> Configuración Global de Herramientas del menú*.



Hemos agregado los instaladores de Maven y Docker y hemos marcado la `Install automatically` casilla de verificación. Jenkins instala estas herramientas cuando nuestra secuencia de comandos (*archivo Jenkins*) se ejecuta por primera vez. Damos `myMaven` y `myDocker` nombramos a las herramientas. Accederemos a estas herramientas con estos nombres en el archivo de script.

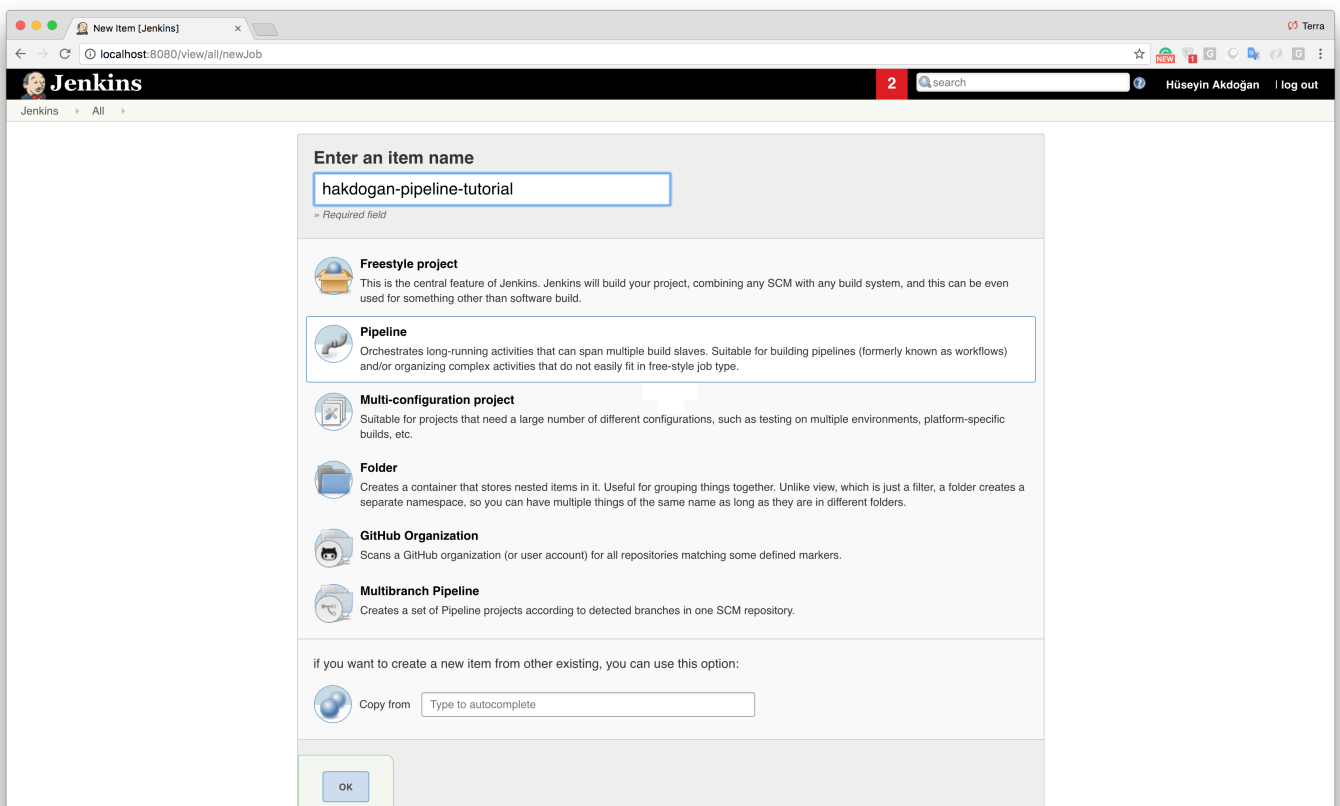
Dado que realizaremos algunas operaciones, como la `checkout` base de código y `pushing an image to Docker Hub`, necesitamos definir el `Docker Hub Credentials`. Tenga en cuenta que si estamos utilizando un repositorio privado, debemos definirlo `Github credentials`. Estas definiciones se realizan en la *Página de inicio de Jenkins -> Credenciales -> Credenciales globales (sin restricciones) -> Agregar*

menú de *credenciales* .



The screenshot shows the 'New Credentials' form in Jenkins. The browser address bar indicates the URL is `localhost:8080/credentials/store/system/domain/_/newCredentials`. The Jenkins header shows the user 'Hüseyin Akdoğan' is logged in. The form is titled 'New Credentials [Jenkins]' and has a breadcrumb trail: 'Jenkins > Credentials > System > Global credentials (unrestricted)'. On the left, there are links for 'Back to credential domains' and 'Add Credentials'. The main form fields are: 'Kind' (set to 'Username with password'), 'Scope' (set to 'Global (Jenkins, nodes, items, all child items, etc)'), 'Username' (set to 'hakdogan'), 'Password' (masked with dots), 'ID' (set to 'dockerHubAccount'), and 'Description' (set to 'Docker Hub Account'). An 'OK' button is at the bottom left. The footer indicates the page was generated on 23-Nov-2017 11:23:16 UTC, with links to the REST API and Jenkins version 2.60.3.

Usamos el valor que ingresamos en el `id` campo para Docker Login en el archivo de script. Ahora, definimos pipeline en *Jenkins Home Page* -> *New Item* menu.



The screenshot shows the 'New Item' form in Jenkins. The browser address bar indicates the URL is `localhost:8080/view/all/newJob`. The Jenkins header shows the user 'Hüseyin Akdoğan' is logged in. The form is titled 'New Item [Jenkins]' and has a breadcrumb trail: 'Jenkins > All'. The main form fields are: 'Enter an item name' (set to 'hakdogan-pipeline-tutorial'), 'Freestyle project' (selected), 'Pipeline' (description: 'Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.'), 'Multi-configuration project' (description: 'Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.'), 'Folder' (description: '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.'), 'GitHub Organization' (description: 'Scans a GitHub organization (or user account) for all repositories matching some defined markers.'), and 'Multibranch Pipeline' (description: 'Creates a set of Pipeline projects according to detected branches in one SCM repository.'). Below these options, there is a section 'If you want to create a new item from other existing, you can use this option:' with a 'Copy from' dropdown menu. An 'OK' button is at the bottom left.

En este paso, seleccionamos `GitHub hook trigger for GITScm` pooling opciones para una ejecución

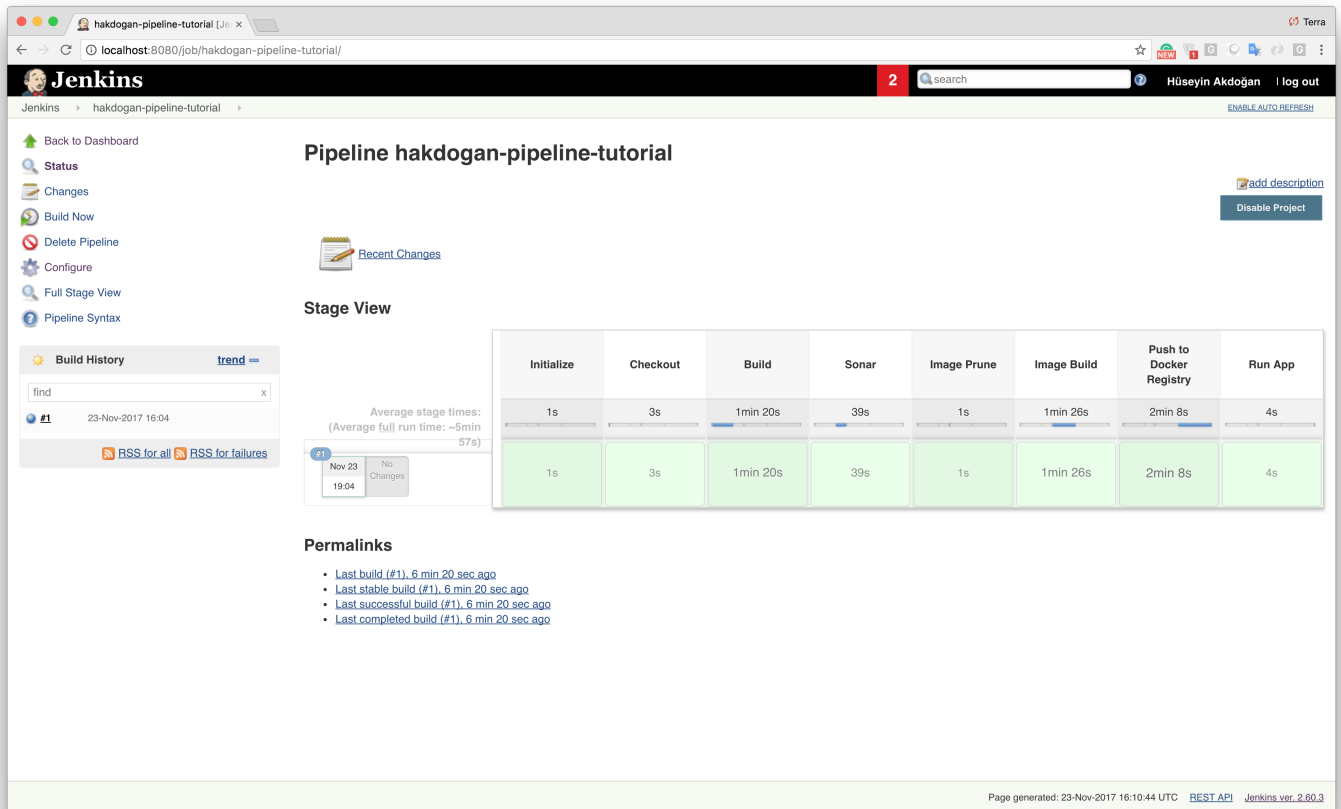
automática de la canalización por Github hook llamada.

The screenshot shows the 'Build Triggers' tab in the Jenkins configuration interface. The 'Build after other projects are built' checkbox is unchecked. The 'Build periodically' checkbox is unchecked. The 'GitHub hook trigger for GITScm polling' checkbox is checked. Below these, there are several other options: 'Poll SCM' (unchecked), 'Disable this project' (unchecked), 'Quiet period' (unchecked), and 'Trigger builds remotely (e.g., from scripts)' (unchecked). Each option has a help icon (question mark) to its right.

También en la sección Pipeline, seleccionamos Pipeline script from SCM como Definición, definimos el repositorio GitHub y el nombre de la rama, y especificamos la ubicación del script (archivo Jenkins).

The screenshot shows the 'Pipeline' tab in the Jenkins configuration interface. The 'Definition' dropdown is set to 'Pipeline script from SCM'. The 'SCM' dropdown is set to 'Git'. The 'Repositories' section shows a single repository with the 'Repository URL' set to 'https://github.com/hakdogan/jenkins-pipeline.g' and 'Credentials' set to '- none -'. The 'Branches to build' section shows a 'Branch Specifier (blank for \'any\')' set to '*/master'. The 'Repository browser' dropdown is set to '(Auto)'. The 'Additional Behaviours' section has an 'Add' button. The 'Script Path' is set to 'Jenkinsfile'. The 'Lightweight checkout' checkbox is checked. At the bottom, there are 'Save' and 'Apply' buttons.

Después de eso, cuando se realiza una inserción en el repositorio remoto o cuando se activa manualmente la tubería por Build Now opción, se ejecutarán los pasos descritos en el archivo Jenkins.



Revisión de puntos importantes del archivo de Jenkins

```

1 stage ( 'Initialize' ) {
2     def dockerHome = herramienta 'myDocker'
3     def mavenHome   = herramienta 'myMaven'
4     env . PATH = "$ { dockerHome } / bin: $ { mavenHome } / bin: $ { env . PATH }"
5 }

```

Las herramientas de cliente de Maven y Docker que hemos definido en Jenkins en el menú Configuración global de herramientas se agregan a la variable de entorno PATH para usar estas herramientas con un comando sh.

```

1 stage ( 'Registro de Push to Docker' ) {
2     withCredentials ( [ usernamePassword ( Id de credenciales: 'dockerHubAccount' , usern:
3         pushToImage ( CONTAINER_NAME , CONTAINER_TAG , USERNAME , PASSWORD )
4     }
5 }

```

withCredentials proporcionado por Jenkins Credentials Binding Plugin y unir credenciales a las variables. Pasamos el valor dockerHubAccount con el credentialsId parámetro. Recuerde que el valor dockerHubAccount es el ID de credenciales de Docker Hub que hemos definido en *Jenkins Home Page -> Credentials -> Global credentials (no restringido) -> Add Credentials* menu. De esta forma, accedemos a la información de nombre de usuario y contraseña de la cuenta para iniciar sesión.

Configuración Sonarqube

Para Sonarqube , hemos hecho las siguientes definiciones en el `pom.xml` archivo del proyecto.

```
1 < sonar.host.url > http: // sonarqube: 9000 </ sonar.host.url >
2 ...
3 < dependencias >
4 ...
5     < dependencia >
6         < groupId > org.codehaus.mojo </ groupId >
7         < artifactId > sonar-maven-plugin </ artifactId >
8         < versión > 2.7.1 </ version >
9         < tipo > maven-plugin </ type >
10    </ dependency >
11 ...
12 </ dependencias >
```

En el archivo de redacción de la ventana acoplable, dimos el nombre del servicio Sonarqube que es sonarqube , por eso en el archivo `pom.xml`, la URL del sonar se definió como `http: // sonarqube: 9000` .

En este artículo, he intentado compartir un tutorial completo, espero que esto pueda ayudarlo.

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



DevOps Intelligence cambia el juego



5 razones por las que Jenkins es tan útil y popular



Explorando DevOps: enfoque en CI / CD



Gratis DZone Refcard Comenzando con Docker

Temas: DOCKER, JENKINS, CI / CD, INTEGRACIÓN CONTINUA, ENTREGA CONTINUA, DEVOPS

Publicado en DZone con permiso de Hüseyin Akdoğan. [Vea el artículo original aquí.](#)

Las opiniones expresadas por los contribuidores de DZone son suyas.

Obtenga lo mejor de DevOps en su bandeja de entrada.

Manténgase actualizado con el boletín DevOps quincenal de DZone. [VER UN EJEMPLO](#)

SUSCRIBIR

Recursos para socios de DevOps

Continúe su transformación digital con un Blueprint para entrega continua.

Atomic



Automatice la seguridad en su oleoducto DevOps

Sonatype



4 Ways to Improve Your DevOps Testing

xMatters



Jenkins, Docker and DevOps: The Innovation Catalysts

CloudBees

