DOCUMENTO ARQUITECTURA DE LA PLATAFORMA DE INTEGRACIÓN CONTINUA - JENKINS

Fábrica de software – ADA S.A.S.

**REVISIÓN Y CONTROL DE CAMBIOS**

**Revisión y Versiones**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Autor | Versión | Descripción | Cargo | Fecha |
| Jhonn Edison Gomez Melo | 1.0 | Documento | Desarrollador | 06/05/2024 |

**Control de Cambios**

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Autor | Versión | Descripción del cambio |
| 06/05/2024 | Jhonn Edison Gomez Melo | 1.0 | Liberación Documento |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

TABLA DE CONTENIDO

Contenido

[**RESUMEN** 4](#_heading=h.4i7ojhp)

[**INTRODUCCIÓN** 5](#_heading=h.2xcytpi)

[**JENKINS** 6](#_heading=h.1ci93xb)

[**¿QUE ES JENKINS?** 6](#_heading=h.3whwml4)

[**CARACTERÍSTICAS DE JENKINS** 6](#_heading=h.2bn6wsx)

[**¿QUÉ ES LA INTEGRACIÓN CONTINUA (CI)?** 7](#_heading=h.qsh70q)

[**¿POR QUÉ USAR JENKINS PARA REALIZAR CI?** 7](#_heading=h.3as4poj)

[**INSTALACIÓN DE LA IMAGEN DOCKERIZADA DE JENKINS** 8](#_heading=h.1pxezwc)

[**¿QUÉ ES DOCKER?** 8](#_heading=h.49x2ik5)

[**¿PARA QUÉ SIRVE DOCKER?** 8](#_heading=h.2p2csry)

[**REQUISITOS DEL SISTEMA:** 9](#_heading=h.147n2zr)

[**INSTRUCCIONES DE INSTALACIÓN DE LA IMAGEN** 10](#_heading=h.3o7alnk)

[**INSTALACIÓN DEL CONTENEDOR DE JENKINS** 11](#_heading=h.23ckvvd)

[**CONFIGURACIÓN INICIAL DE JENKINS** 13](#_heading=h.ihv636)

[**CREACIÓN DE PIPELINE EN JENKINS** 17](#_heading=h.32hioqz)

[**INSTALACIÓN DE PLUGINS ADICIONALES** 31](#_heading=h.1hmsyys)

[**¿QUÉ ES SLACK?** 31](#_heading=h.41mghml)

[**¿PARA QUÉ SIRVE SLACK?** 32](#_heading=h.2grqrue)

[**CONFIGURACIÓN DE SLACK** 32](#_heading=h.vx1227)

[**INSTALACION DEL PLUGIN SLACK EN JENKINS** 33](#_heading=h.3fwokq0)

[**¿QUÉ ES SONARQUBE?** 36](#_heading=h.1v1yuxt)

[**¿PARA QUÉ SIRVE SONARQUBE?** 37](#_heading=h.4f1mdlm)

[**INSTALACIÓN DEL PLUGIN SONARQUBE EN JENKINS** 37](#_heading=h.2u6wntf)

[**CREACIÓN DE UNA IMAGEN DESDE JENKINS HACIA DOCKER HUB** 42](#_heading=h.19c6y18)

[**¿QUÉ ES DOCKER HUB?** 42](#_heading=h.3tbugp1)

[**¿PARA QUÉ SIRVE DOCKER HUB?** 43](#_heading=h.28h4qwu)

[**¿QUE ES EL DESPLIEGUE CONTINUO (CD)?** 50](#_heading=h.nmf14n)

# **RESUMEN**

El presente documento detalla la arquitectura de la plataforma de Integración Continua basada en Jenkins, una herramienta fundamental en el desarrollo de software moderno. Explora cómo Jenkins facilita la automatización de procesos de construcción, pruebas y despliegue, mejorando la eficiencia y la calidad del software. Además, se examina cómo Jenkins se integra con Docker Hub y Kubernetes para permitir el despliegue continuo de aplicaciones en entornos de contenedores. A través de una descripción detallada de sus componentes y su interacción, este documento ofrece una comprensión completa de cómo se implementa y opera una infraestructura de Integración Continua utilizando Jenkins en conjunción con Docker Hub y Kubernetes.

# **INTRODUCCIÓN**

En el contexto del desarrollo de software, la Integración Continua (CI) se ha convertido en una práctica esencial para mejorar la calidad del código, acelerar los ciclos de desarrollo y facilitar la colaboración entre equipos. En respuesta a estas necesidades, Jenkins se erige como una plataforma de código abierto que no solo automatiza los procesos de construcción y pruebas, sino que también habilita el despliegue continuo de aplicaciones en entornos de contenedores.

Esta introducción busca proporcionar una visión general completa de la arquitectura de la plataforma de integración continua basada en Jenkins. Comenzaremos explorando los fundamentos de la Integración Continua y su importancia en el desarrollo de software moderno. Luego, nos adentraremos en la arquitectura específica de Jenkins, examinando sus componentes principales, sus interacciones y las mejores prácticas para su implementación y mantenimiento.

Además, destacaremos cómo Jenkins se integra de manera fluida con herramientas como Docker Hub y Kubernetes para facilitar el despliegue continuo de aplicaciones en entornos de contenedores. Al comprender la arquitectura subyacente de Jenkins y sus conexiones con Docker Hub y Kubernetes, los equipos de desarrollo pueden aprovechar al máximo esta potente herramienta, mejorando la calidad del software, acelerando los procesos de desarrollo y fomentando una cultura de entrega continua (CD).

# **JENKINS**

## **¿QUE ES JENKINS?**

Jenkins es una plataforma de código abierto ampliamente utilizada para la Implementación Continua (CI) y la Entrega Continua (CD) en el desarrollo de software. Como una herramienta fundamental en el proceso de desarrollo moderno, Jenkins automatiza tareas repetitivas y tediosas, permitiendo a los equipos de desarrollo concentrarse en la creación de software de alta calidad.

En su núcleo, Jenkins proporciona un entorno flexible y extensible para la automatización de procesos de construcción, pruebas y despliegue. Su arquitectura modular permite a los usuarios adaptar la plataforma a sus necesidades específicas mediante la instalación de una amplia gama de complementos (plugins) disponibles en su vasto ecosistema.

## **CARACTERÍSTICAS DE JENKINS**

* **Automatización de procesos:** Jenkins permite la automatización de tareas repetitivas y manuales, como la construcción de código, la ejecución de pruebas unitarias y la implementación de aplicaciones.
* **Integración con herramientas y sistemas:** Jenkins se integra con una variedad de herramientas y sistemas utilizados en el desarrollo de software, incluyendo sistemas de control de versiones (Git, SVN), herramientas de gestión de proyectos (Jira, Trello) y plataformas de gestión de contenedores (Docker, Kubernetes).
* **Gestión de flujos de trabajo:** Jenkins facilita la creación y gestión de flujos de trabajo complejos mediante la definición de pipelines, que permiten orquestar y automatizar el proceso de desarrollo desde la integración del código hasta su despliegue en producción.
* **Escalabilidad y distribución:** Jenkins es altamente escalable y admite la distribución de la carga de trabajo a través de una red de agentes distribuidos, lo que permite ejecutar procesos de CI/CD en paralelo para mejorar la eficiencia y el rendimiento.
* **Personalización y extensibilidad:** Jenkins ofrece una amplia gama de plugins que permiten a los usuarios personalizar y extender su funcionalidad según sus necesidades específicas, desde integraciones con herramientas de terceros hasta la creación de nuevas características y funcionalidades.

## **¿QUÉ ES LA INTEGRACIÓN CONTINUA (CI)?**

La integración continua (CI) con Jenkins es un enfoque en el desarrollo de software donde los cambios en el código se integran automáticamente en un repositorio compartido varias veces al día. Jenkins es una herramienta popular de automatización de CI que facilita este proceso.

En términos prácticos, cuando un desarrollador realiza cambios en el código y los sube al repositorio, Jenkins automáticamente toma esos cambios, compila el código y ejecuta pruebas automatizadas. Si todo pasa satisfactoriamente, Jenkins puede incluso implementar automáticamente los cambios en un entorno de prueba o producción.

Esto permite a los equipos de desarrollo detectar problemas de integración y errores de código de manera temprana en el ciclo de desarrollo, lo que ayuda a garantizar la calidad del software y a acelerar el tiempo de entrega. La integración continua con Jenkins ayuda a automatizar tareas repetitivas, reduce el riesgo de errores humanos y fomenta una cultura de desarrollo ágil y colaborativo.

## **¿POR QUÉ USAR JENKINS PARA REALIZAR CI?**

Hay varias razones por las que Jenkins es una opción popular para implementar la integración continua (CI), como por ejemplo:

* **Flexibilidad y Personalización:** Jenkins es altamente configurable y adaptable a una amplia variedad de casos de uso y tecnologías. Ofrece una gran cantidad de complementos (plugins) que permiten extender su funcionalidad para integrarlo con herramientas y sistemas externos.
* **Código Abierto y Comunidad Activa:** Jenkins es un proyecto de código abierto con una comunidad activa de desarrolladores y usuarios que contribuyen a su mejora continua. Esto significa que hay una amplia base de conocimientos disponible y una comunidad que puede ayudar con problemas y preguntas.
* **Amplia Integración con Herramientas y Tecnologías:** Jenkins se integra fácilmente con una variedad de herramientas de desarrollo, control de versiones, sistemas de gestión de proyectos, plataformas de nube, servicios de notificación, entre otros. Esto facilita la construcción de un pipeline de CI/CD completo que se adapte a las necesidades específicas de cada equipo.
* **Escalabilidad:** Jenkins puede adaptarse a proyectos de diferentes tamaños, desde pequeños proyectos de desarrollo de software hasta grandes empresas con múltiples equipos y complejas arquitecturas de software. Su capacidad para distribuir la carga de trabajo en múltiples agentes y servidores lo hace escalable y adecuado para entornos de desarrollo de gran tamaño.
* **Historial y Confianza:** Jenkins ha estado en el mercado durante muchos años y ha sido utilizado por una amplia gama de organizaciones en todo el mundo. Su historial y la confianza que ha ganado en la comunidad de desarrollo lo convierten en una opción confiable para implementar CI.

# **INSTALACIÓN DE LA IMAGEN DOCKERIZADA DE JENKINS**

## **¿QUÉ ES DOCKER?**

Docker es una plataforma de código abierto diseñada para permitir la creación, el despliegue y la ejecución de aplicaciones dentro de contenedores. Los contenedores son unidades de software que encapsulan todas las dependencias necesarias para ejecutar una aplicación, incluidos el código, las bibliotecas y las configuraciones del sistema. Docker proporciona una manera eficiente y portátil de empaquetar, distribuir y ejecutar aplicaciones, independientemente del entorno de ejecución. Utiliza la tecnología de contenedores para permitir la creación de entornos de ejecución aislados y ligeros, conocidos como contenedores. Cada contenedor ejecuta una aplicación o un servicio de manera independiente de otros contenedores y del sistema operativo subyacente.

## **¿PARA QUÉ SIRVE DOCKER?**

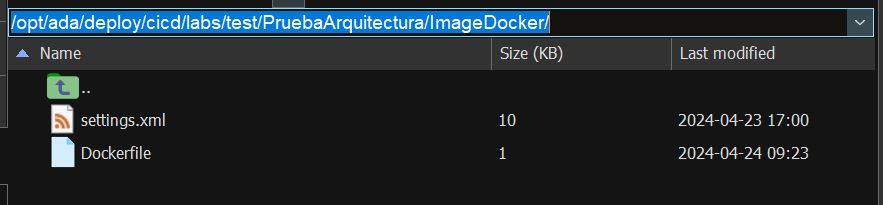
Docker sirve para varios propósitos relacionados con el desarrollo, el despliegue y la gestión de aplicaciones:

* Desarrollo de aplicaciones: Docker facilita la creación de entornos de desarrollo reproducibles y portátiles, permitiendo a los desarrolladores trabajar en aplicaciones con las mismas configuraciones y dependencias, independientemente de su entorno de desarrollo.
* Despliegue de aplicaciones: Docker simplifica el proceso de despliegue de aplicaciones al empaquetarlas junto con todas sus dependencias en contenedores. Esto facilita la implementación de aplicaciones en cualquier entorno, ya sea local, en la nube o en servidores dedicados.
* Gestión de infraestructura: Docker proporciona herramientas para gestionar y orquestar contenedores en entornos de producción. Esto incluye la capacidad de escalar automáticamente los contenedores, administrar el ciclo de vida de las aplicaciones y automatizar tareas de administración.
* Entrega continua y DevOps: Docker se integra fácilmente en prácticas de entrega continua y en metodologías de desarrollo de operaciones (DevOps), permitiendo a los equipos de desarrollo automatizar el proceso de construcción, prueba y despliegue de aplicaciones de manera eficiente y repetible.
* Microservicios: Docker es una tecnología clave en la implementación de arquitecturas de microservicios, donde las aplicaciones se dividen en componentes independientes y desacoplados, cada uno ejecutándose en su propio contenedor.

Se realiza el proceso de instalación de la imagen dockerizada de Jenkins, con el fin de poder crear un contenedor a partir de dicha imagen y de esa manera poder ejecutar Jenkins en el servidor especificado(Para este caso el servidor de desarrollo). Para dicha instalación se definieron los siguientes parámetros:

## **REQUISITOS DEL SISTEMA:**

En el servidor de Desarrollo (10.1.140.21) se realiza la instalación del Dockerfile, el cual estará encargado de generar la imagen de Jenkins adaptándose a las dependencias de software que tiene la compañía en los repositorios Nexus, las cuales se configuran en un archivo settings.xml y se ubican ambos archivos en la siguiente ruta /opt/ada/deploy/cicd/labs/test/PruebaArquitectura/ImageDocker/

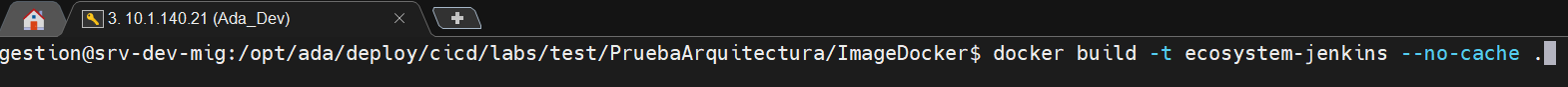


En el archivo DockerFile se pueden evidenciar los requisitos con los cuales se instalará Jenkins, específicamente la instalación de la imagen tiene estas características:

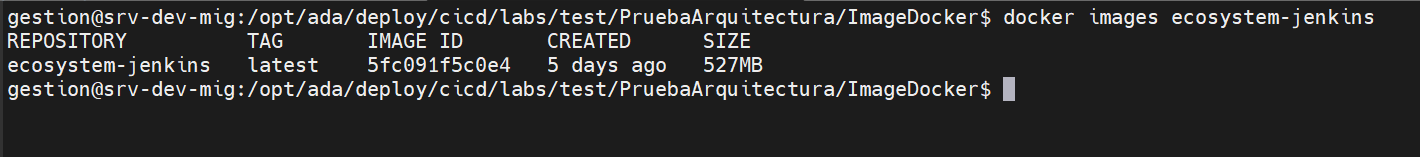
* Imagen oficial de Jenkins, descargada desde docker hub: jenkins/Jenkins
* JDK con el que se ejecutara el contenedor que se cree a partir de esta imagen: jdk21
* Se define un usuario root, el cual controlara el contenedor como usuario admin
* Se especifica la versión de maven con la cual se ejecutará la imagen: MAVEN\_VERSION 3.9.6
* Se define un usuario Jenkins, a el cual se le dan privilegios para que al crear la imagen pueda copiar el archivo settings.xml que se encuentra en la carpeta de instalación y Jenkins lo tome por defecto como el settings.xml del contenedor que se creara.

## **INSTRUCCIONES DE INSTALACIÓN DE LA IMAGEN**

Para realizar la instalación de la imagen dockerizada de Jenkins, debemos ingresar al servidor y acceder por consola a la siguiente ruta, /opt/ada/deploy/cicd/labs/test/PruebaArquitectura/ImageDocker/ en la cual se encuentran los archivos Dockerfile y settings.xml. Donde se encuentra toda la configuración para la instalación, una vez allí se debe ejecutar el siguiente comando para la creación:



Una vez creada la imagen ejecutamos el siguiente comando para realizar la validación de que la imagen jenkins dockerizada fue creada correctamente



Al ejecutar el comando, Docker nos muestra la imagen que tiene desplegada con el nombre que le dimos al crearla para este caso ecosystem-jenkins.

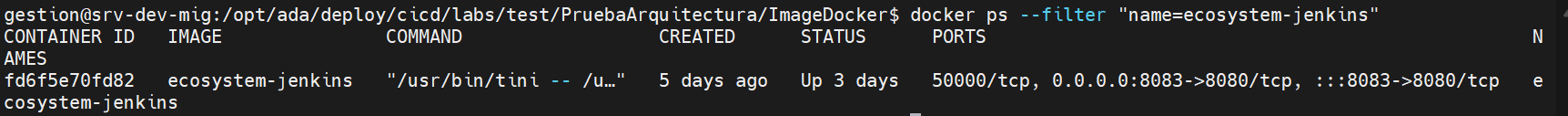
# **INSTALACIÓN DEL CONTENEDOR DE JENKINS**

Una vez creada la imagen dockerizada de Jenkins, procedemos a realizar la creación del contenedor, con el siguiente comando:

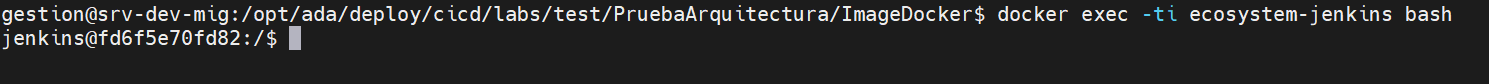


Aquí se especifica el Puerto en el cual Jenkins va a ser desplegado(Para este caso 8083), adicionalmente se le da un nombre al contenedor y se limita la memoria del mismo a 4gb de uso, puesto que para iniciar la migración se tomó la decisión de darle dicha capacidad, la cual puede cambiar a medida que se desarrolle el Proyecto de la migración.

Al ejecutar el comando se genera el contenedor, para estar seguros que se desplegó el contenedor correctamente ejecutamos el siguiente comando por consola:



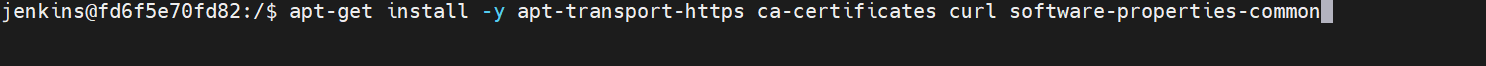
Una vez arriba procedemos a ingresar al contenedor para Validar las versiones de la instalación e instalar docker en el mismo, para ingresar al contenedor utilizamos el siguiente comando:



Dentro del contenedor procedemos a instalar docker dentro del contenedor Jenkins con los siguientes comandos:

* Actualizar los repositorios de paquetes: Una vez dentro del contenedor, es una buena práctica actualizar los repositorios de paquetes con el siguiente comando:



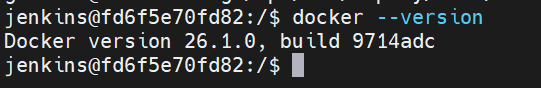
* Instalar Docker: Después de actualizar los repositorios, se procede instalar Docker. Para instalar Docker en un contenedor de Jenkins, primero necesitarás instalar las dependencias necesarias. Se puede hacer con los siguientes comandos:
* Luego, se agrega la clave GPG oficial de Docker:
* Agrega el repositorio de Docker a las fuentes de APT:
* Actualiza nuevamente los repositorios:



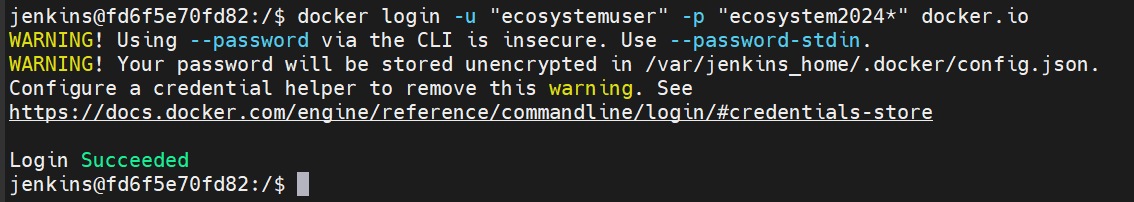
* Finalmente, se instala Docker:



Una vez completada la instalación, puedes verificar que Docker se haya instalado correctamente ejecutando el siguiente comando:

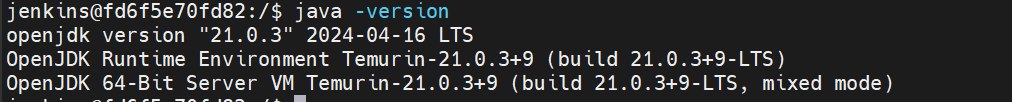


Para sincronizar el contenedor con el repositorio de Docker Hub, se debe ejecutar el siguiente comando:



Aquí se especifican las credenciales de docker hub con el que el contenedor tendrá comunicación para poder publicar las imágenes desde Jenkins hacia el registry de docker.

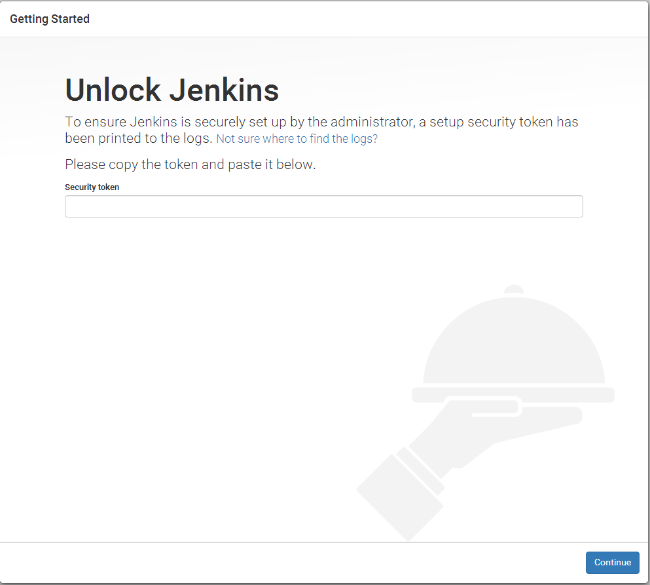
Por último para Validar la instalación ejecutamos el comando java -version para ver que el jdk haya quedado correctamente instalado:



# **CONFIGURACIÓN INICIAL DE JENKINS**

Para acceder a la interfaz de usuario de Jenkins después de la instalación del contenedor hay que acceder a la url del servidor donde se está ejecutando el contenedor y referenciar el Puerto en el cual hicimos la instalación, para este caso el Puerto es el 8083, así que nos dirigimos a un navegador y accedemos a la siguiente url: <http://10.1.140.21:8083/>

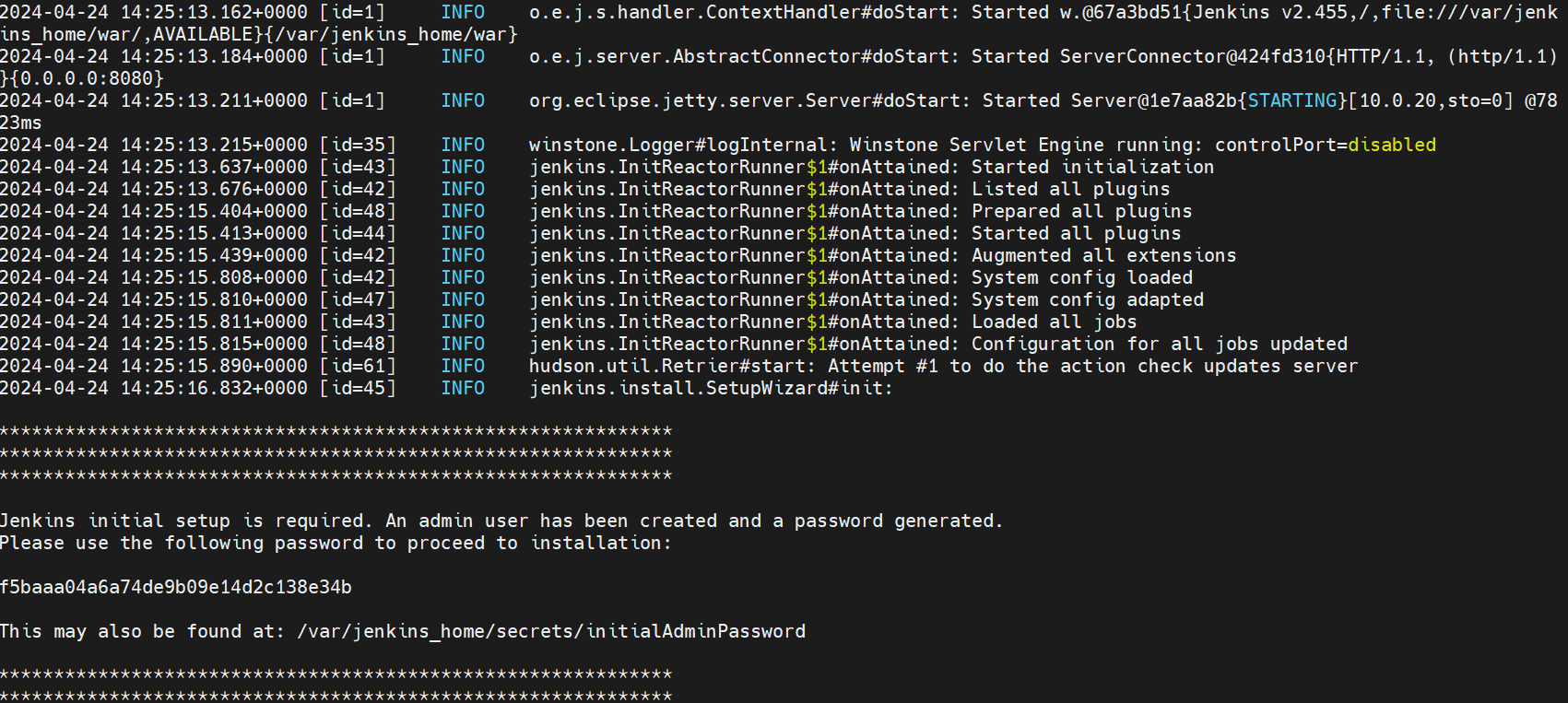
Al abrir la url Podemos ver que Jenkins se encuentra bloqueado y necesita un código de desbloqueo



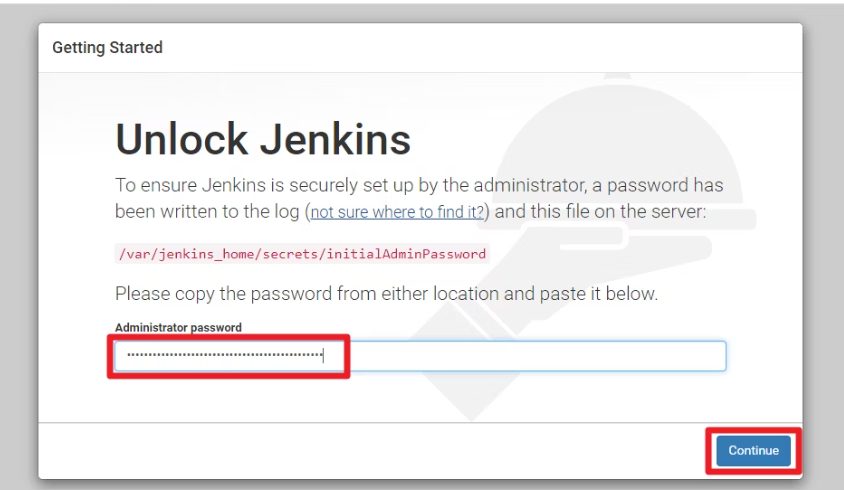
Volvemos a la consola del servidor y ejecutamos el siguiente comando:



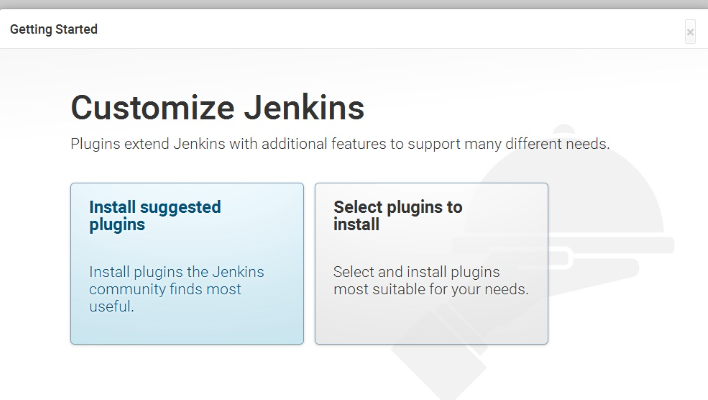
Una vez ejecutado vemos la clave que se debe ingresar para poder desbloquear Jenkins, se copia y pega en la interfaz gráfica.



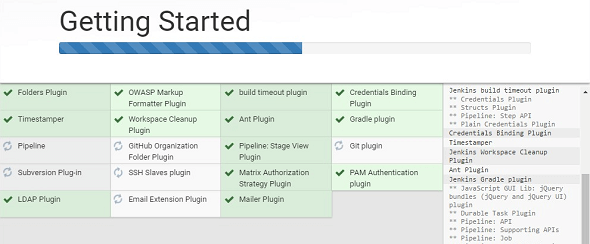
Se digita la clave que se encuentra en la consola y se da click en el botón continuar



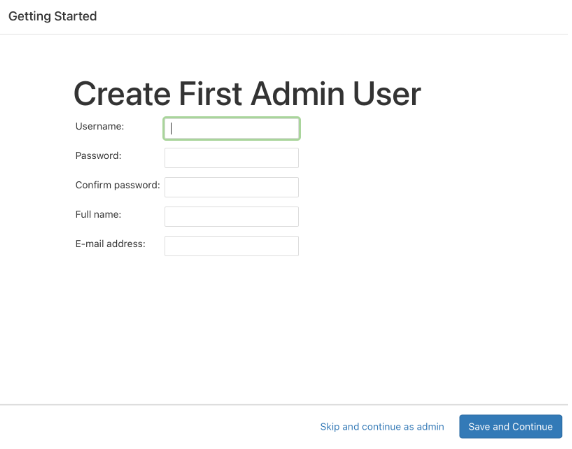
Al dar click en continuar, Jenkins nos muestra dos tipos de instalación, una manual y la otra con los plugins sugeridos, para este caso tomaremos la instalación de plugins sugeridos



Automáticamente, Jenkins empieza a instalar los plugins sugeridos en nuestro contenedor.



Al realizar la instalación de los plugins sugeridos, Jenkins nos solicita la creación de un usuario admin para la gestión de su interfaz gráfica:



Para este caso en específico se dejan las siguientes credenciales, con las cuales se gestionará todo el entorno de jenkins:

usuario: ecosystem-deploy

contraseña: ecosystem-deploy

En la siguiente Ventana nos muestra la url donde quedará la aplicación de Jenkins, damos click en continuar y visualizamos la siguiente pantalla

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Esta pantalla nos indica que Jenkins fue instalado correctamente con los plugins sugeridos.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

# **CREACIÓN DE PIPELINE EN JENKINS**

***NOTA: “Esta creacion de Pipeline es especifica para la rama Master del Proyecto PruebaArquitectura, ubicado en la ruta*** [***http://10.1.140.120/ada-microservices-ecosystem/lab/test/pruebaarquitectura***](http://10.1.140.120/ada-microservices-ecosystem/lab/test/pruebaarquitectura)***, en caso de necesitar mas pipelines para aplicar CI/CD en conjunto con Gitflow, otro Proyecto o otra rama se aconseja la creacion de un Pipeline siguiendo los siguientes pasos”***

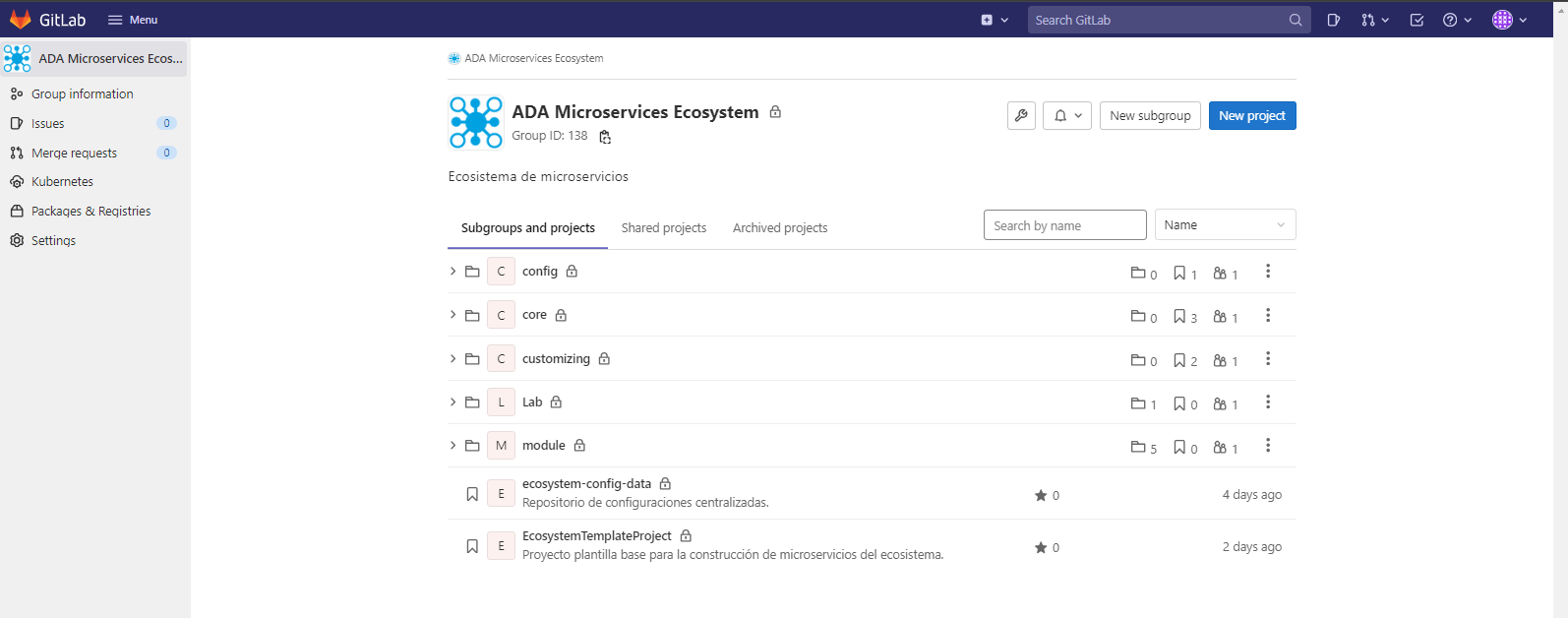
Para realizar la creación de un pipeline en Jenkins, como primera medida debemos realizar la conexión entre Jenkins y el repositorio donde se encuentra el código a automatizar, para este caso puntual se utilizara el repositorio de la migracion SICOF el cual se encuentra en Gitlab(<http://10.1.140.120/ada-microservices-ecosystem> ). La conexión entre ambas herramientas se realiza de la siguiente forma:

* Como primera medida debemos preparar a Jenkins para la conexión con GitLab, lo hacemos instalando los siguientes plugins los cuales se pueden buscar en la sección Panel de Control / Administrar Jenkins / Plugins

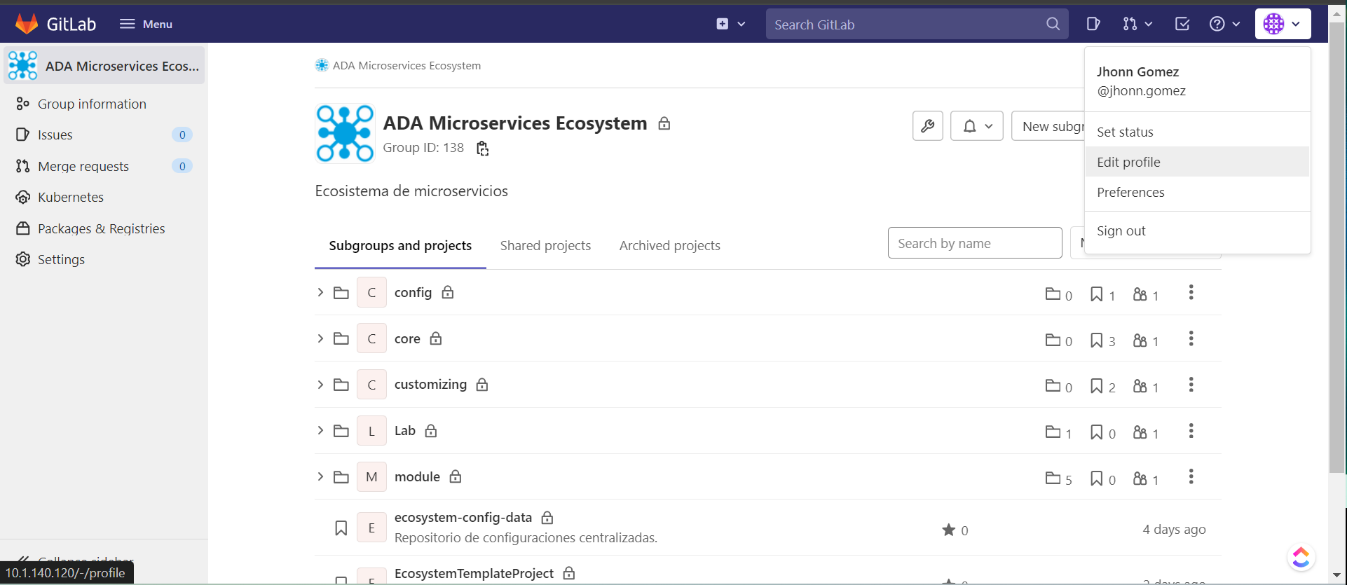
Interfaz de usuario gráfica, Texto, Aplicación

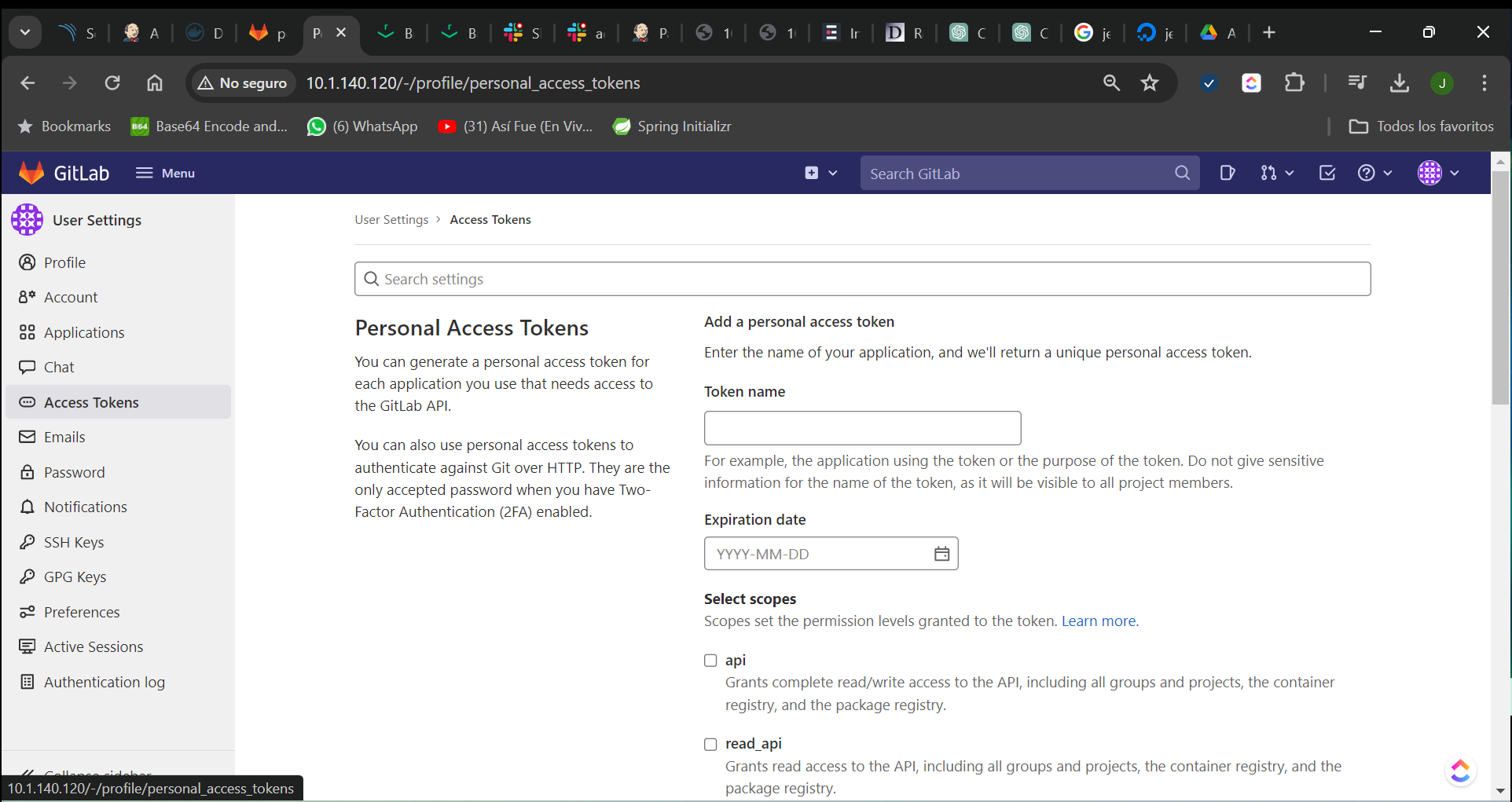
Descripción generada automáticamente

* Nos logueamos en el repositorio de GitLab al cual queremos realizar la conexión con Jenkins.

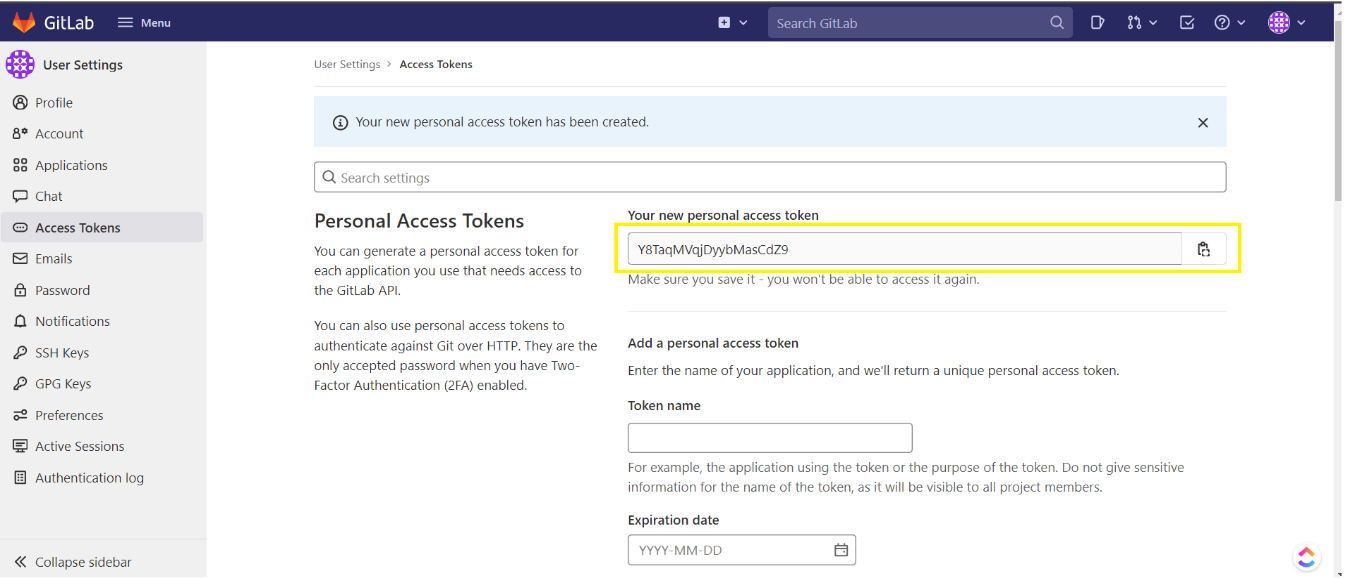


* En GitLab, damos click en la opción edit profile y luego vamos a la sección de access token, con el fin de crear un token de comunicación entre GitLab y Jenkins

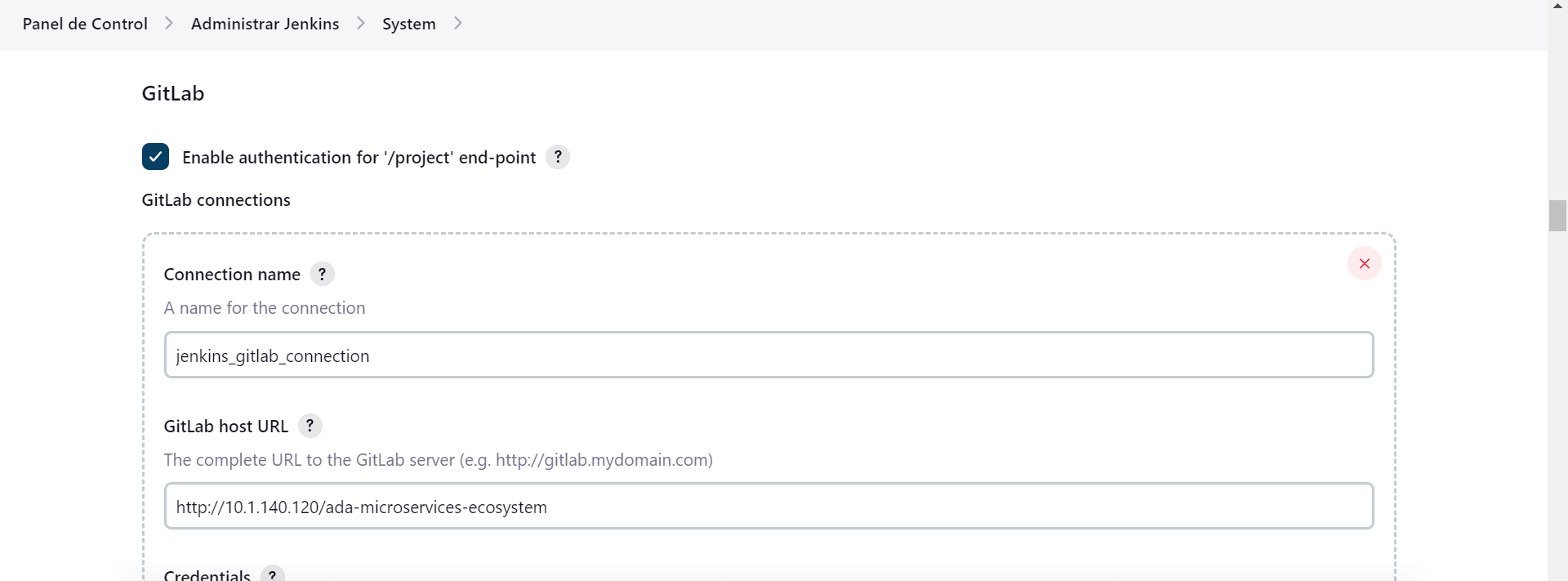




* Una vez creado el token, se debe copiar ya que si no se copia en este momento se debe Volver a crear ya que el token no vuelve a ser visible.



* Ya con el Token de GitLab copiado, nos dirigimos a Jenkins en el menú Panel de Control / Administrar Jenkins / System en la sección GitLab le damos un nombre a la conexión y referfenciamos la url del repositorio de GitLab



* En la parte de credenciales, damos click en el boton add para adicionar la nueva conexion a GitLab con el token

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* En la Ventana que nos muestra Jenkins seleccionamos en la seccion Kind la opcion GitLab API Token

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* En la sección de API token se pone el token que generamos en gitlab y se le da un ID y una descripción, damos click en el botón Add para adicionar las credenciales

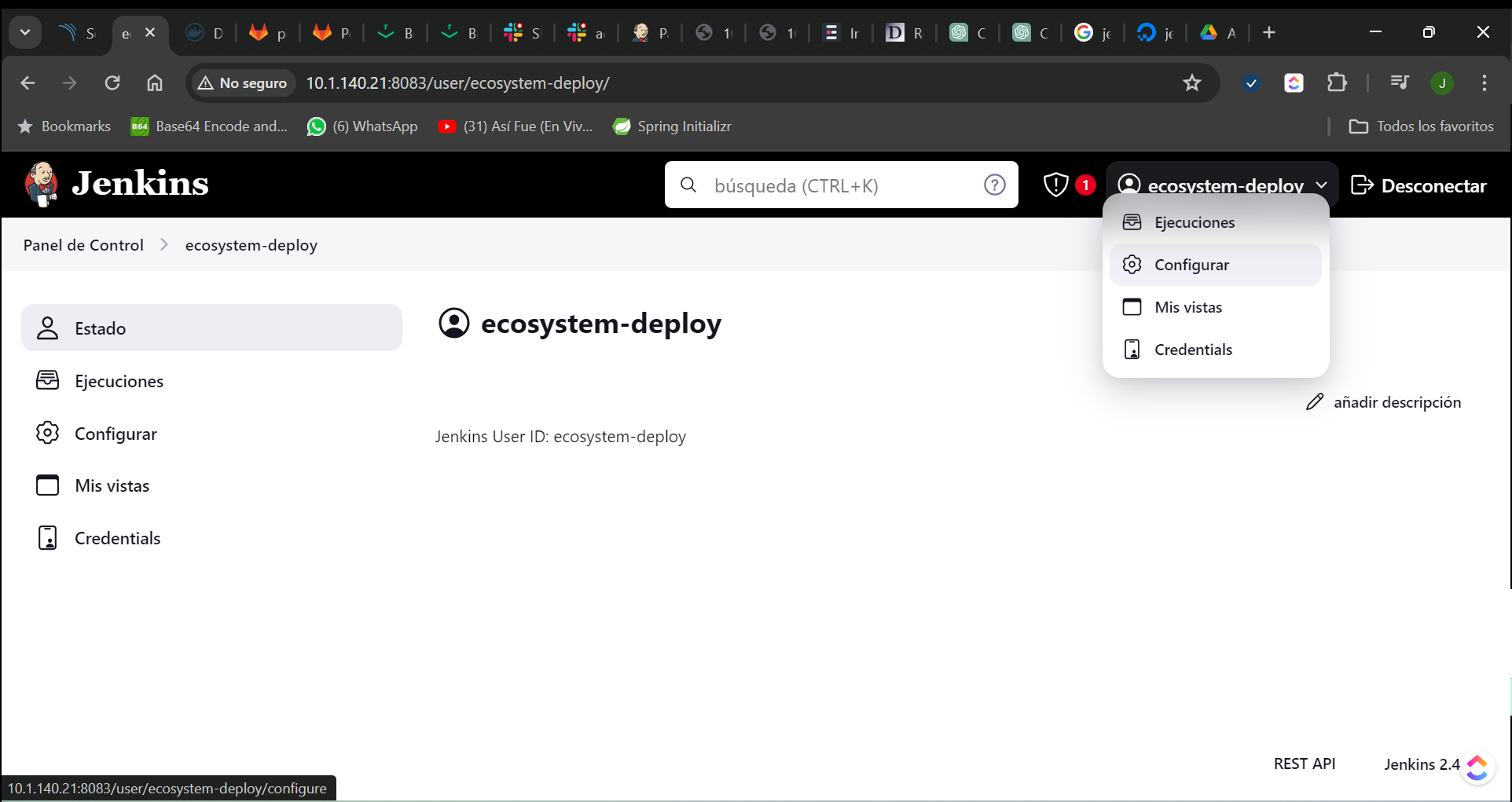


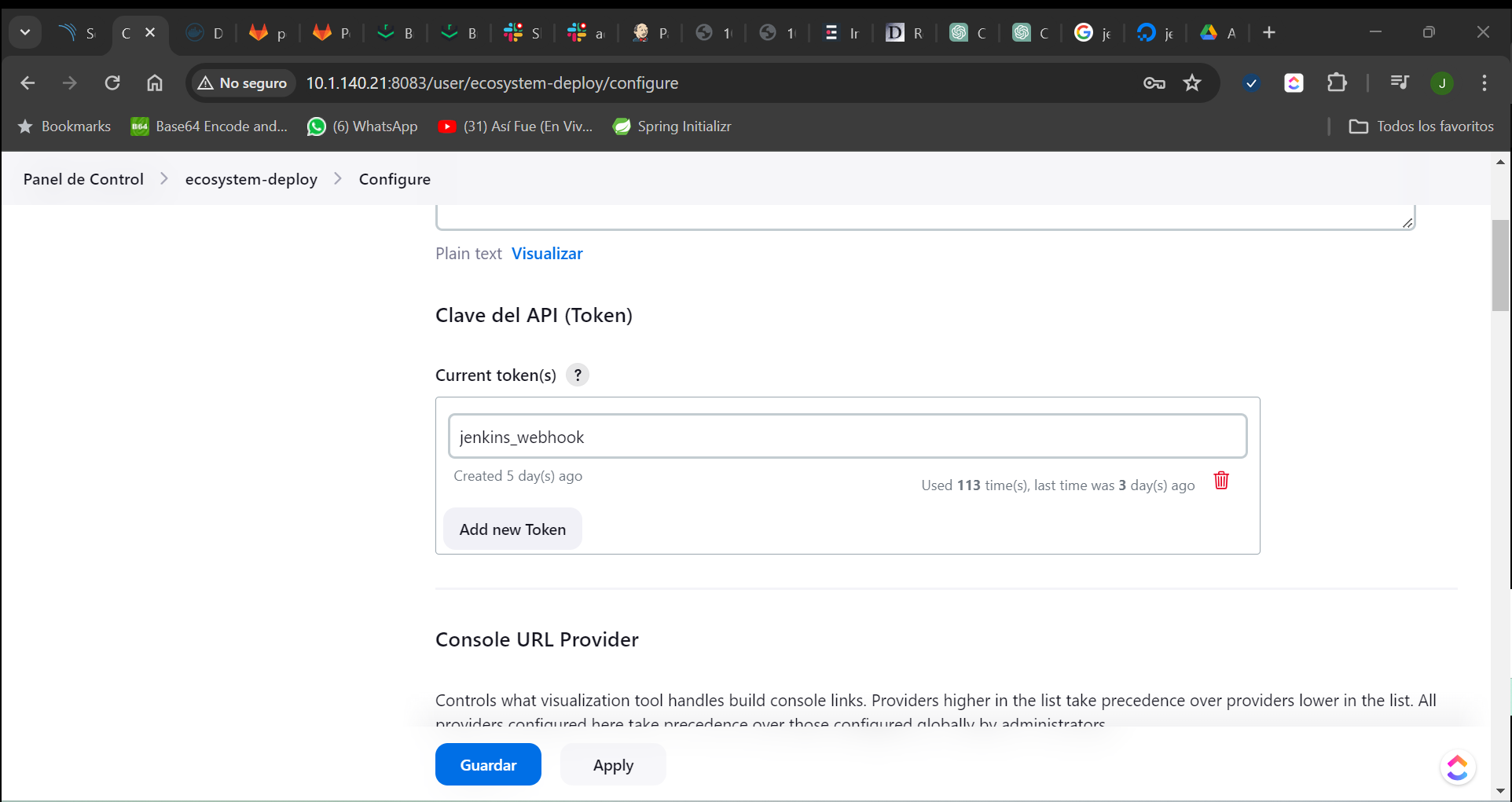
* En la sección Credenciales adicionamos la credencial del token de GitLab que acabamos de crear, damos click en el botón Apply y luego en el botón guardar. De esta manera ya tenemos conectado a GitLab con Jenkins.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

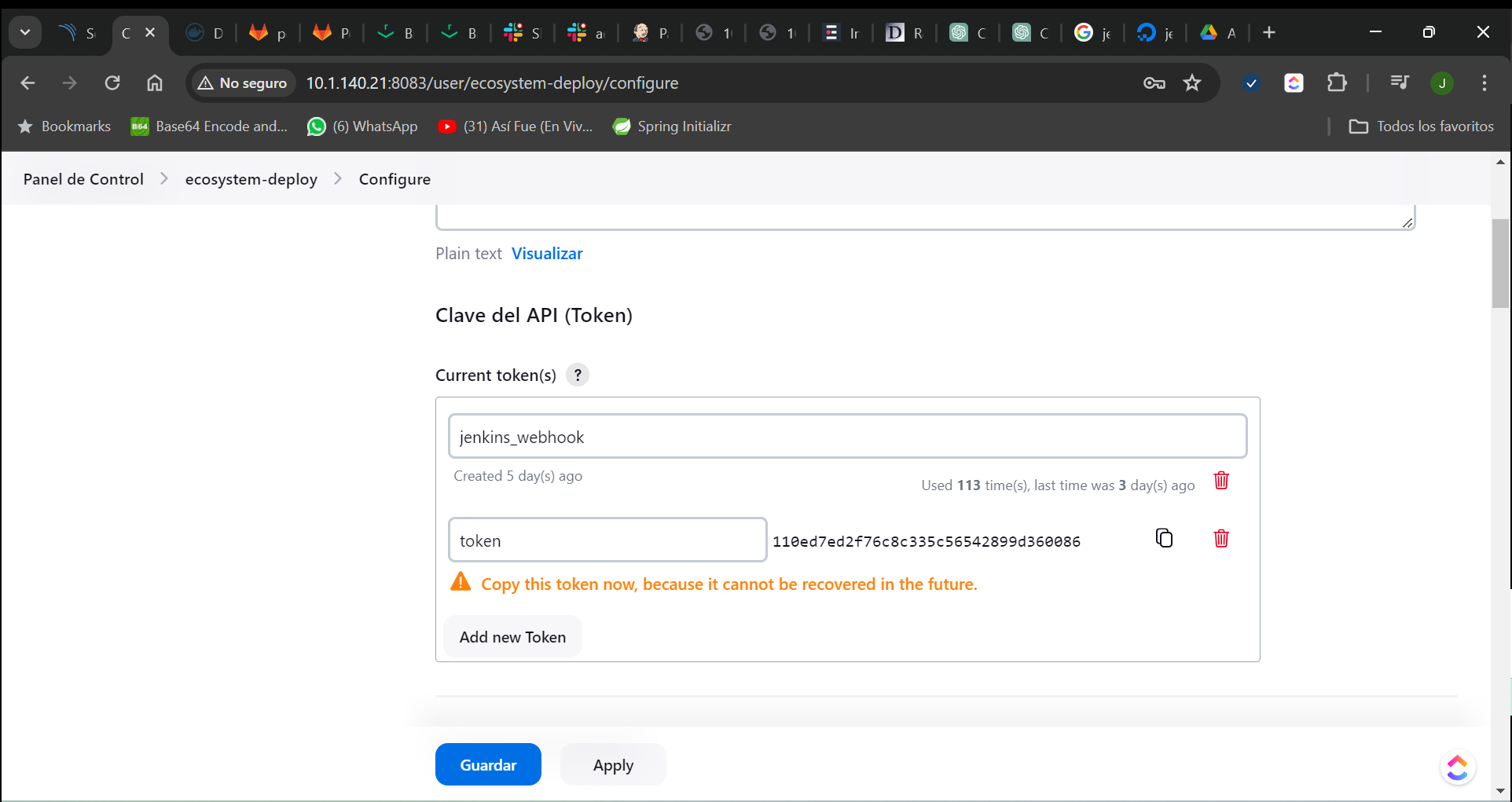
Descripción generada automáticamente

* Procedemos a crear un token de Jenkins , en la sección configurar de la pestaña del usuario damos click vamos a la sección de API Token y damos click en el botón add new token





* Una vez creado el token se debe copiar, ya que va a ser ese el único momento donde el token de Jenkins se pueda visualizar, de no copiarlo hay que regenerarlo

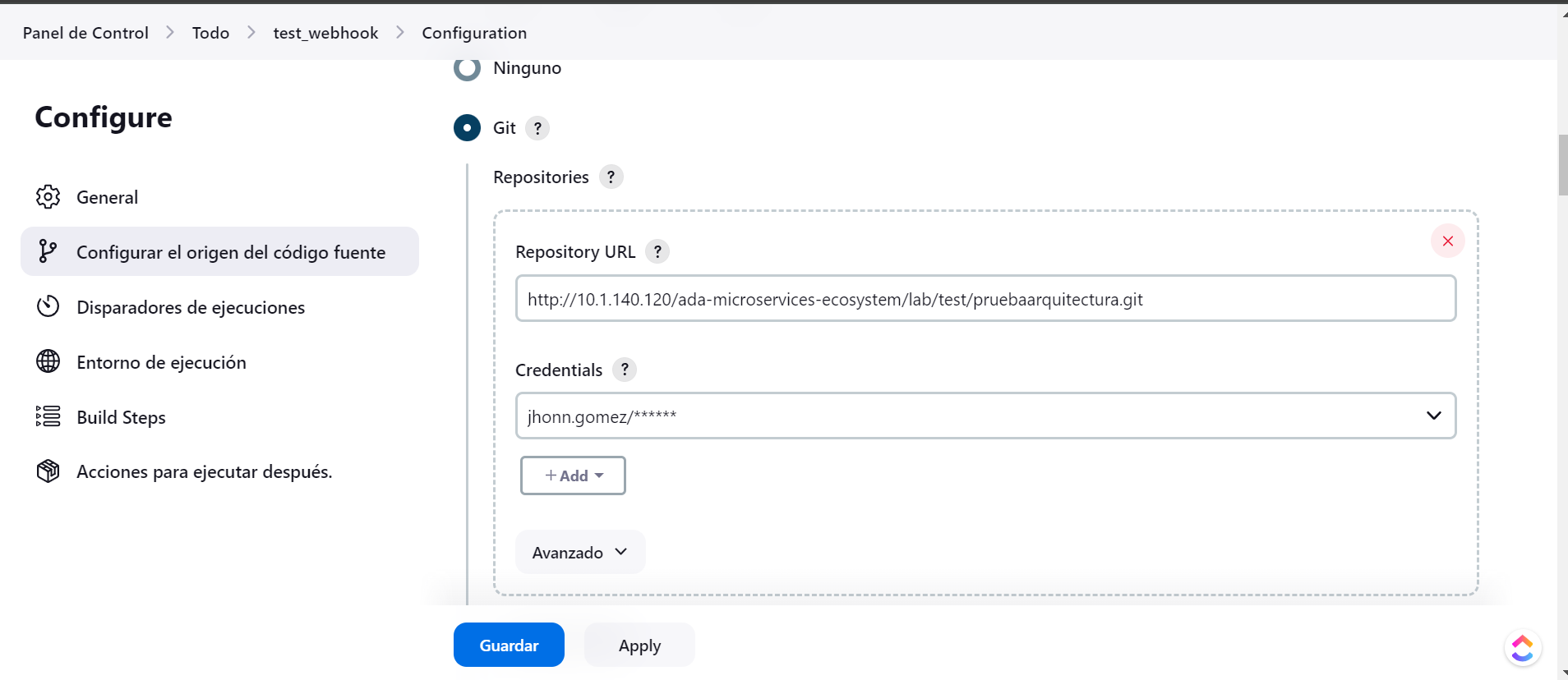


* Al tener la configuración y los tokens referenciados, procedemos a crear el nuevo pipeline en la siguiente opción Panel de Control / Todo / Nueva Tarea, seleccionamos la opción Crear un Proyecto de estilo libre, le asignamos un nombre y damos click en Ok

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

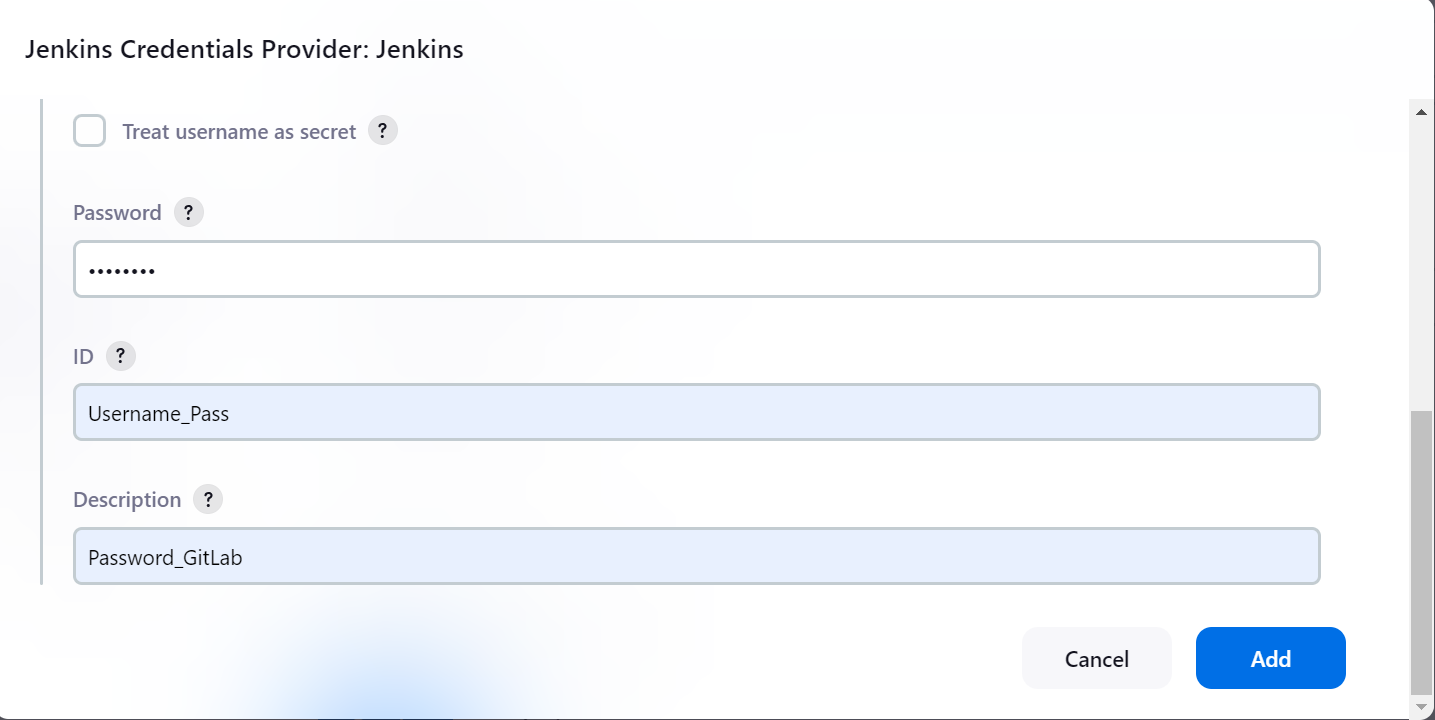
* Una vez creadfa la tarea del Pipeline, procedemos a configurarla, iniciamos configurando la seccion de Git donde se especifica el repositorio al cual va a apuntar el pipeline en la opcion repository URL que para este caso punctual sera este repositorio <http://10.1.140.120/ada-microservices-ecosystem/lab/test/pruebaarquitectura.git>



En la opción de credenciales damos click en add / Jenkins y en la sección Kind especificamos que el método de certificación sera usuario y contraseña de gitlab



Diligenciamos los campos de usuario, contraseña, se le asigna un id a las credenciales y una descripción, damos click en add



Una vez adicionada la contraseña procedemos a utilizarla a la conexión con GitLab, damos click en Apply y en Save

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* En la sección disparadores de ejecuciones seleccionamos los siguientes disparadores, damos click en Apply y en Save:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

* En la sección Branches to Build especificamos que este pipeline se va a ejecutar únicamente para la rama Master

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

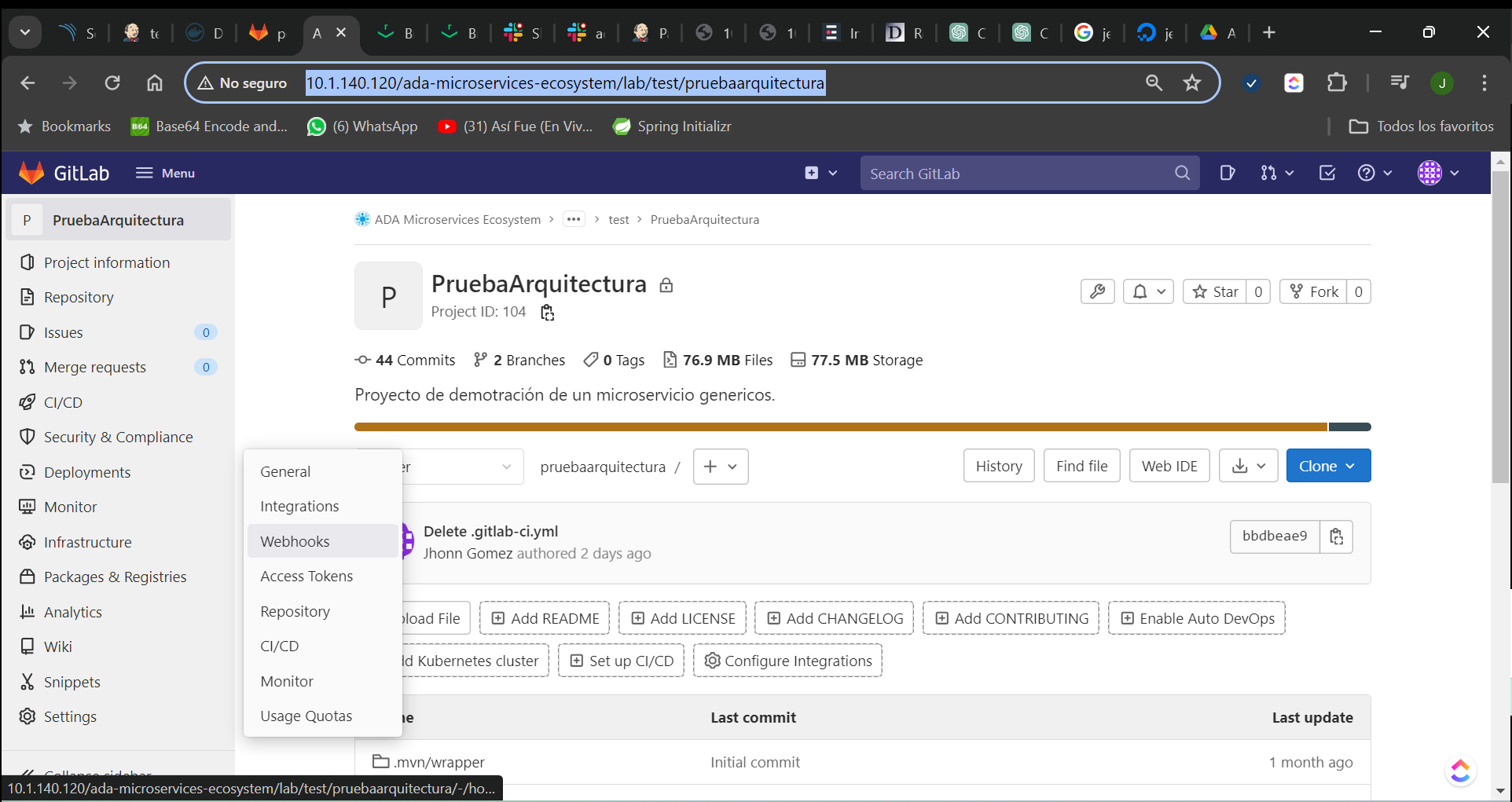
Descripción generada automáticamente

* En la sección Build Steps se crean pasos para que el pipeline ejecute durante el lanzamiento del mismo, para este primer paso se configura el comando clean install de maven para poder compilar el Proyecto y que Jenkins genere el archivo .jar

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* Para terminar de crear la comunicacion entre GitLab y Jenkins, procedemos a crear un WebHooks desde GitLab, el cual disparara un evento cada vez que el componente sufra algun cambio, nos dirigimos al repositorio que vamos a automatizar, <http://10.1.140.120/ada-microservices-ecosystem/lab/test/pruebaarquitectura>, nos dirigimos a la opcion Settings / Webhooks

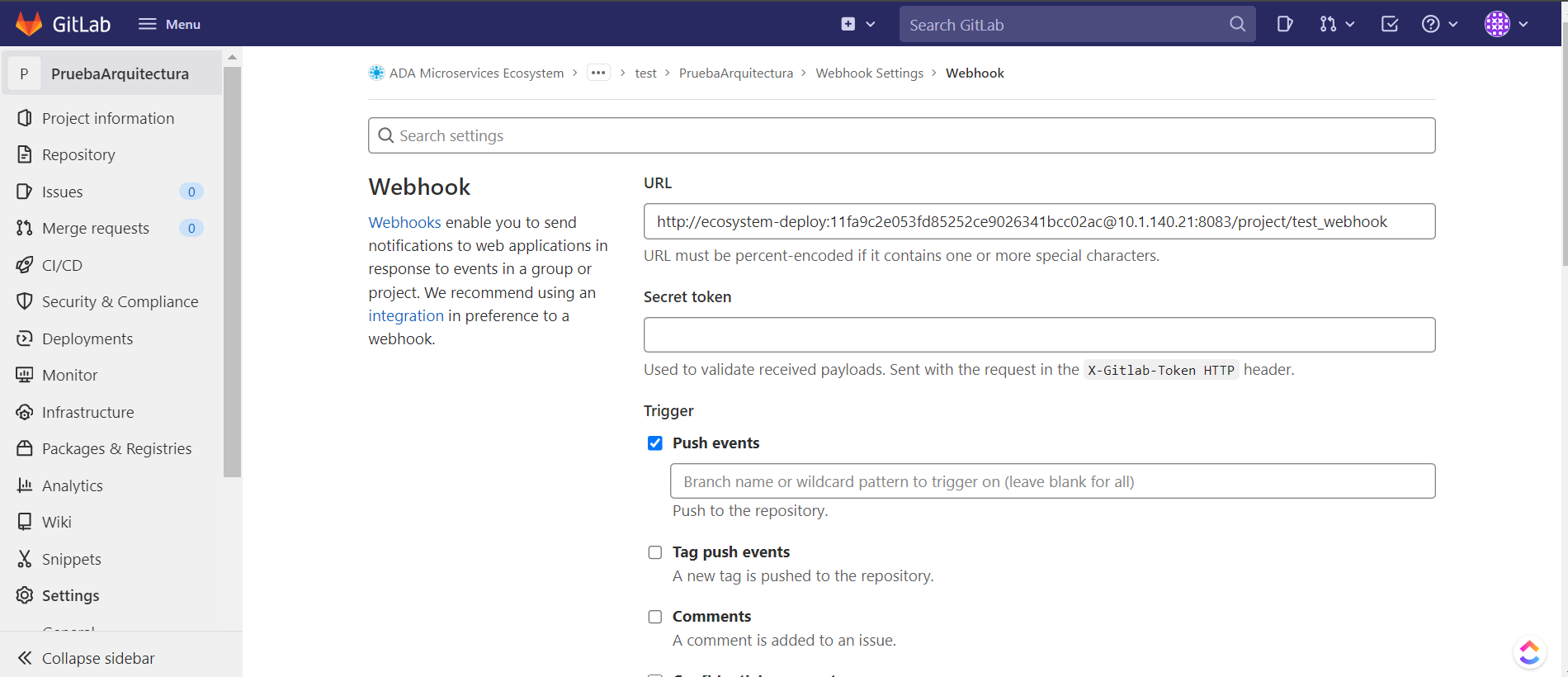


* Una vez abierto el menú de los webhooks, se debe armar la url con base a la url de Jenkins de la siguiente forma

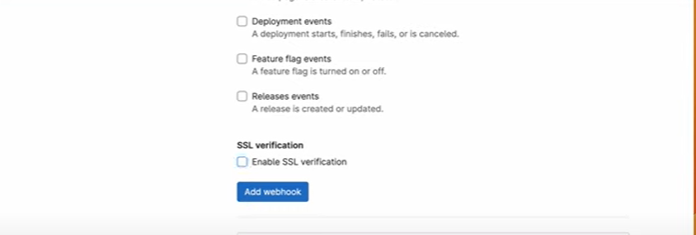
Url Jenkins= <http://10.1.140.21:8083/job/test_webhook/>

Url del Webhook= <http://ecosystem-deploy:11fa9c2e053fd85252ce9026341bcc02ac@10.1.140.21:8083/project/test_webhook>

Donde el primer parámetro es el usuario de Jenkins seguido de dos puntos y el token que generamos desde Jenkins para conectarlo con GitLab, seguido de un @ y la url de Jenkins del webhook con la diferencia que en el path ya no va job, si no project, en la opción de trigger marcamos push events, Merge request events y Job events.



Desmarcamos la opción de SSL Verification ya que estamos usando nuestro token para autenticarnos y le damos click en Add Webhook



Una vez adicionado el WebHook realizamos un test para comprobar que la conexión es exitosa

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Validamos la correcta ejecucion del webhook desde Gitlab

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Se evidencia en Jenkins que el Pipeline También fue ejecutado con la prueba que se lanzo desde GitLab



## **INSTALACIÓN DE PLUGINS ADICIONALES**

Se crea una nueva cuenta en slack con el fin de crear una comunidad y se crea un canal específico para las notificaciones de los pipelines(ada-ecosystem-deploy)

### **¿QUÉ ES SLACK?**

Slack es una plataforma de comunicación empresarial que permite a los equipos colaborar de manera eficiente y efectiva en línea. Aunque se puede comparar con otras aplicaciones de mensajería instantánea, como Microsoft Teams o Discord, Slack está especialmente diseñado para entornos laborales y empresariales, es una herramienta de mensajería en tiempo real que organiza las conversaciones en canales temáticos, lo que permite una comunicación estructurada y orientada a proyectos. Además de la mensajería instantánea, Slack ofrece una variedad de funciones adicionales que lo convierten en una plataforma de colaboración integral, incluyendo la integración con otras herramientas y aplicaciones empresariales.

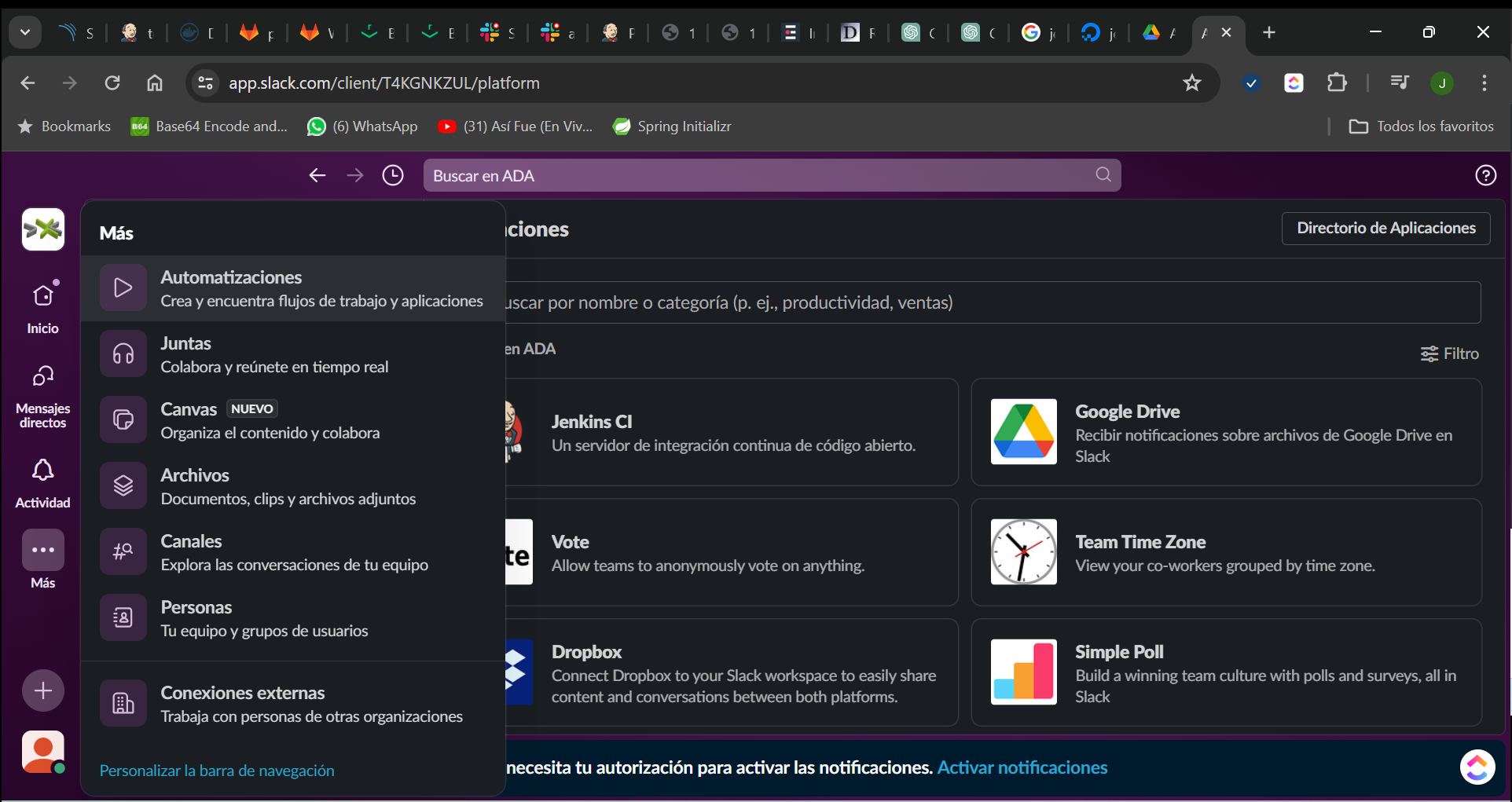
### **¿PARA QUÉ SIRVE SLACK?**

Slack sirve para facilitar la comunicación y la colaboración entre equipos de trabajo en entornos laborales. Algunos de los usos más comunes de Slack incluyen:

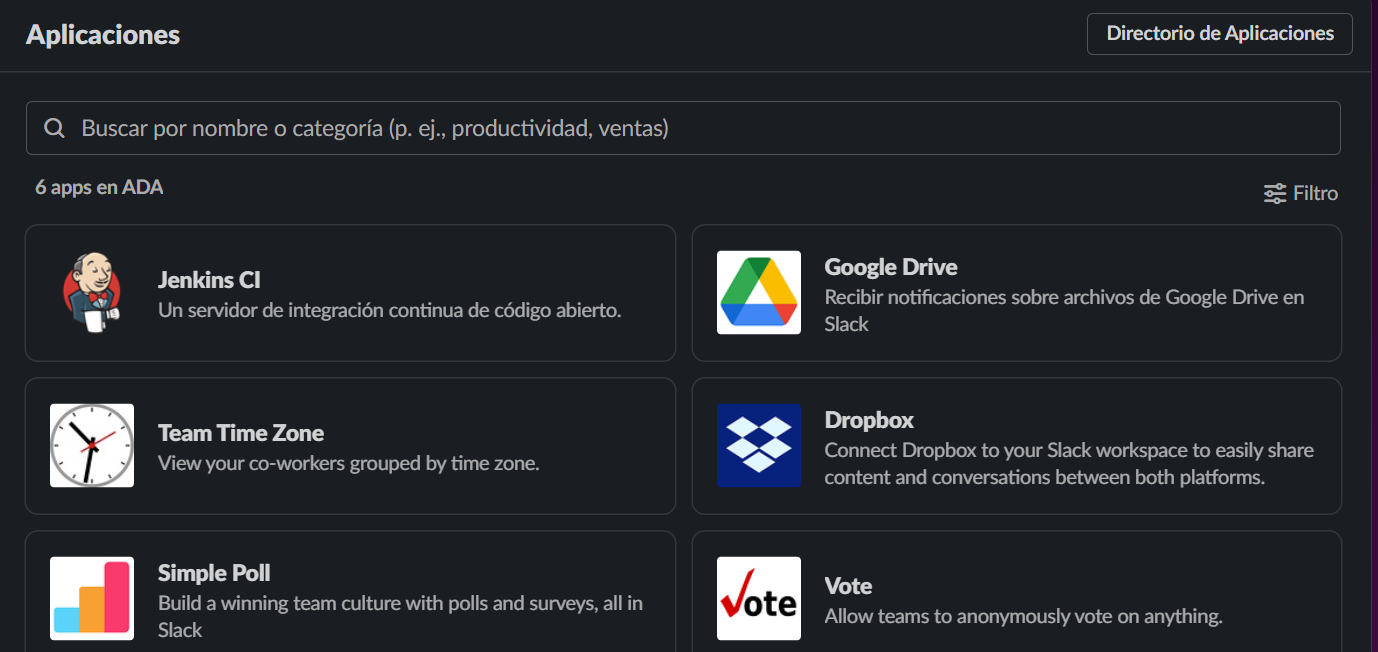
* **Comunicación en tiempo real:** Los equipos pueden comunicarse de manera instantánea a través de mensajes directos o en canales temáticos, lo que facilita la discusión de proyectos y la resolución rápida de problemas.
* **Organización de conversaciones:** Slack organiza las conversaciones en canales, lo que permite a los equipos mantenerse enfocados en temas específicos y evitar la saturación de los mensajes en un único espacio.
* **Integración con herramientas de productividad:** Slack se integra con una amplia gama de herramientas y aplicaciones empresariales, como Google Drive, Trello, GitHub, entre otros, permitiendo a los equipos acceder y compartir información de manera más eficiente.
* **Búsqueda y archivo de información:** Slack proporciona funciones de búsqueda avanzada que permiten a los usuarios encontrar fácilmente mensajes, archivos y otros recursos compartidos en la plataforma.
* **Gestión de proyectos**: Muchos equipos utilizan Slack como una herramienta de gestión de proyectos, aprovechando sus funciones de organización, notificaciones y seguimiento de tareas.

### **CONFIGURACIÓN DE SLACK**

Una vez logueado en Slack vamos a la opción de automatizaciones, buscamos y adicionamos el plugin de Jenkins CI

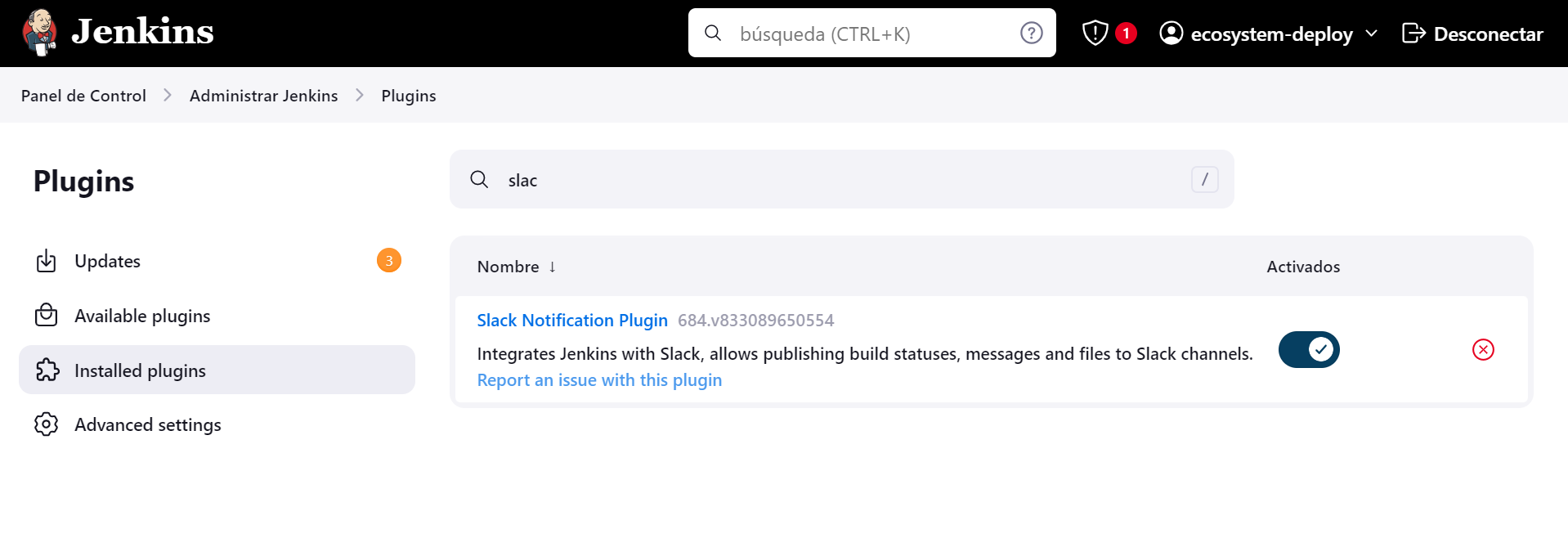


Damos click en el plugin de Jenkins y lo adicionamos al Slack, elegimos el canal al que va a estar asociado los eventos de Jenkins. Se adicionan a las personas que van a estar asociadas al canal

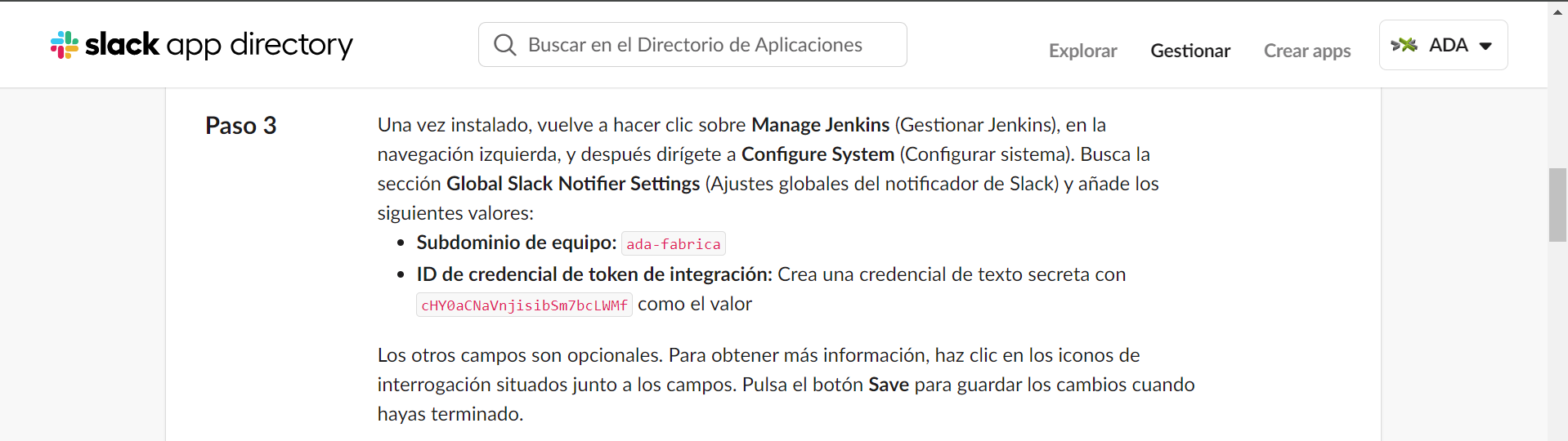


### **INSTALACION DEL PLUGIN SLACK EN JENKINS**

En la herramienta de Jenkins procedemos a instalar el plugin de Slack en la sección Panel de Control / Administrar Jenkins / Plugins, el plugin a instalar es el siguiente:



Una vez instalado el plugin de slack volvemos a la siguiente sección Panel de Control / Administrar Jenkins / System. Donde vamos a configurar el slack con Jenkins de la siguiente manera (Como lo indica la documentación de slack)



En Jenkins la configuracion seria la siguiente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

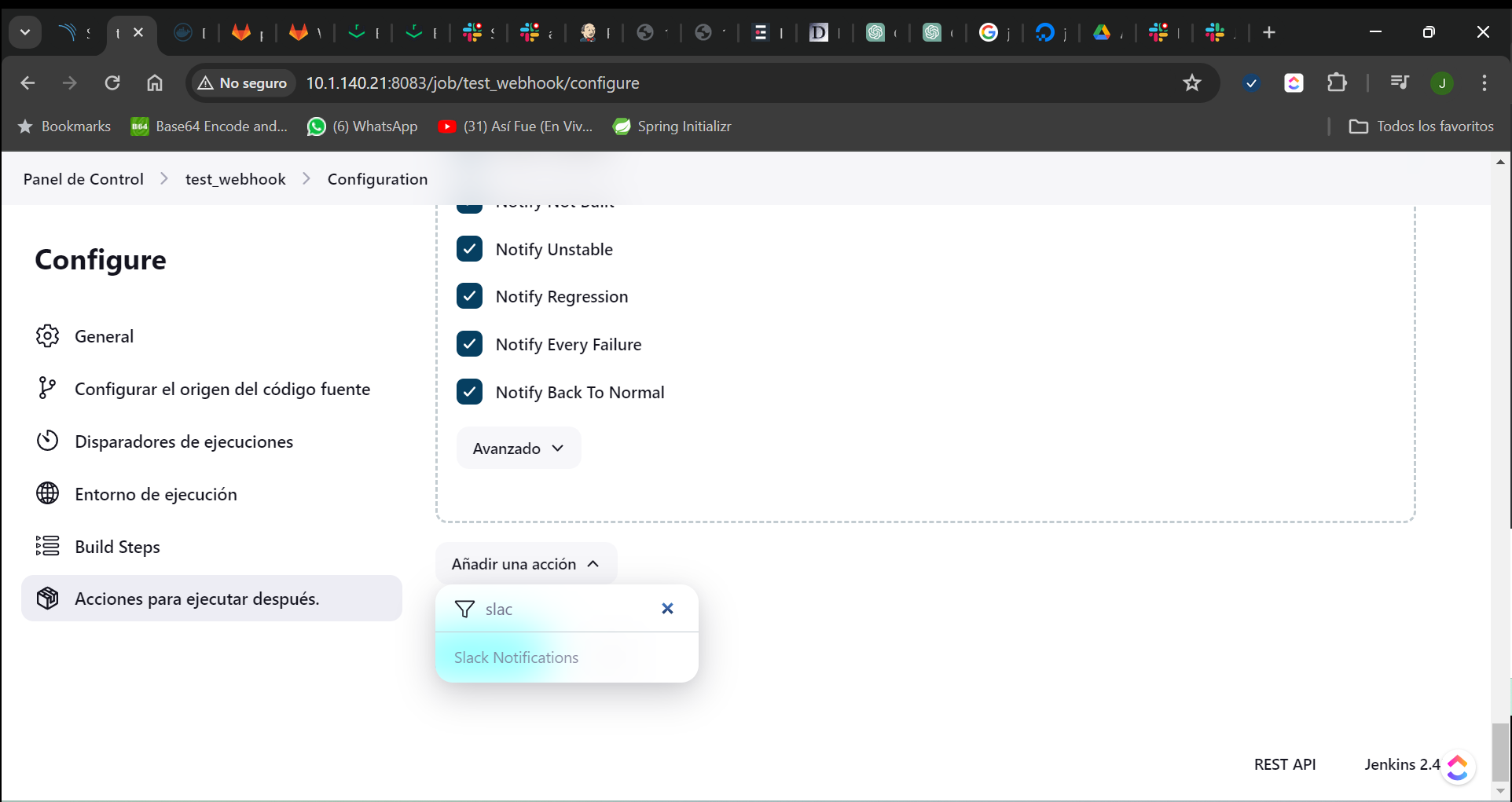
Se crea la credencial con el token que nos brinda slack a la hora de integrar el plugin de Jenkins, se da click en el botón test connection y si la respuesta es success, se da click en apply y save. Ya que la comunicación entre Jenkins y Slack fue creada correctamente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Luego de tener configurado y conectado Jenkins con Slack, vamos a configurar el pipeline que hemos creado para que envíe notificaciones por el canal de slack. La configuración la hacemos en la sección Panel de Control / test\_webhook / Configuration / Acciones para ejecutar después.

En dicha sección se agrega un nuevo paso a la automatización con el plugin de slack, añadiendo la opción slack notifications

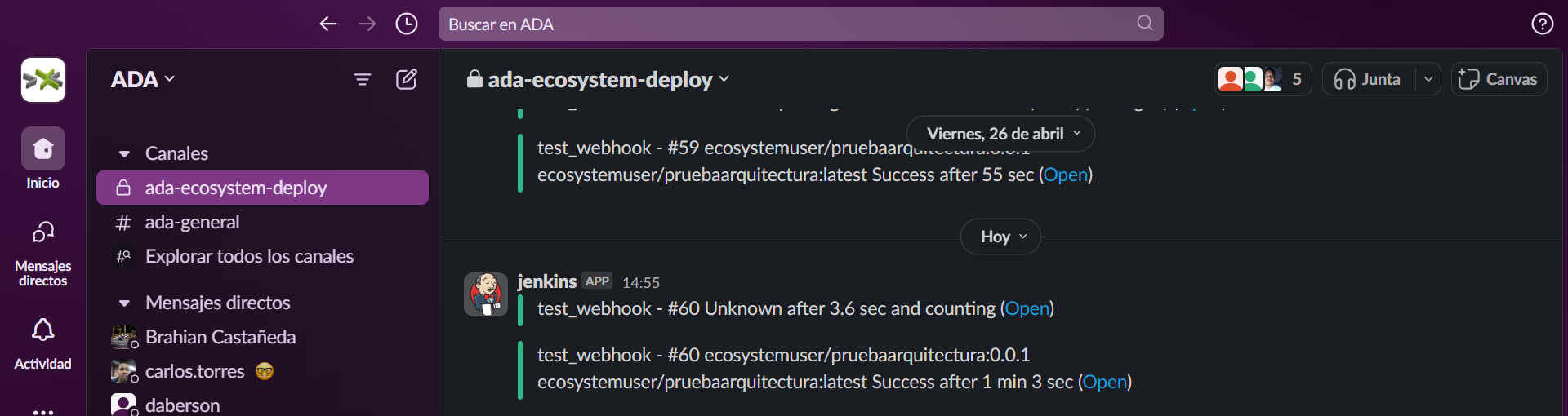


Una vez añadida, marcamos las notificaciones que queremos recibir en este caso se marcan todas para que siempre slack nos notifique por el canal cualquier evento que ocurra con el branch automatizado y se da click en apply y save

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Se ejecuta el pipeline y se evidencia que el canal está funcionando de forma correcta



### **¿QUÉ ES SONARQUBE?**

SonarQube es una plataforma de código abierto para la inspección continua de la calidad del código fuente. Es una herramienta utilizada principalmente en el desarrollo de software para evaluar y mejorar la calidad del código, identificando posibles problemas, vulnerabilidades, errores y áreas de mejora en el código base, proporciona una plataforma integral para evaluar la calidad del código en varios lenguajes de programación, incluyendo Java, C#, JavaScript, Python, entre otros. Utiliza un conjunto de reglas predefinidas y personalizables para analizar el código fuente y proporciona métricas y estadísticas detalladas sobre su calidad y mantenibilidad.

### **¿PARA QUÉ SIRVE SONARQUBE?**

SonarQube sirve para varios propósitos relacionados con la mejora de la calidad del código en proyectos de desarrollo de software:

* Análisis estático del código: SonarQube realiza un análisis estático del código fuente para identificar posibles problemas y vulnerabilidades, como código duplicado, falta de comentarios, complejidad ciclomática, entre otros.
* Detección de errores y vulnerabilidades: Identifica errores de programación comunes y posibles vulnerabilidades de seguridad en el código fuente, lo que ayuda a mejorar la fiabilidad y la seguridad del software.
* Medición de la calidad del código: Proporciona métricas y estadísticas sobre la calidad del código, incluyendo índices de mantenibilidad, confiabilidad y seguridad, que permiten a los equipos de desarrollo evaluar y monitorear la evolución de la calidad del código a lo largo del tiempo.
* Integración con herramientas de desarrollo: SonarQube se integra con diversas herramientas de desarrollo, como sistemas de control de versiones (Git, SVN), sistemas de integración continua (Jenkins, Azure DevOps), IDEs (Eclipse, IntelliJ IDEA) y sistemas de gestión de proyectos (Jira, Trello), lo que facilita su incorporación en el flujo de trabajo de desarrollo.
* Generación de informes: SonarQube genera informes detallados sobre la calidad del código, que pueden ser utilizados por los equipos de desarrollo para identificar áreas de mejora y priorizar actividades de refactorización y mantenimiento.

### **INSTALACIÓN DEL PLUGIN SONARQUBE EN JENKINS**

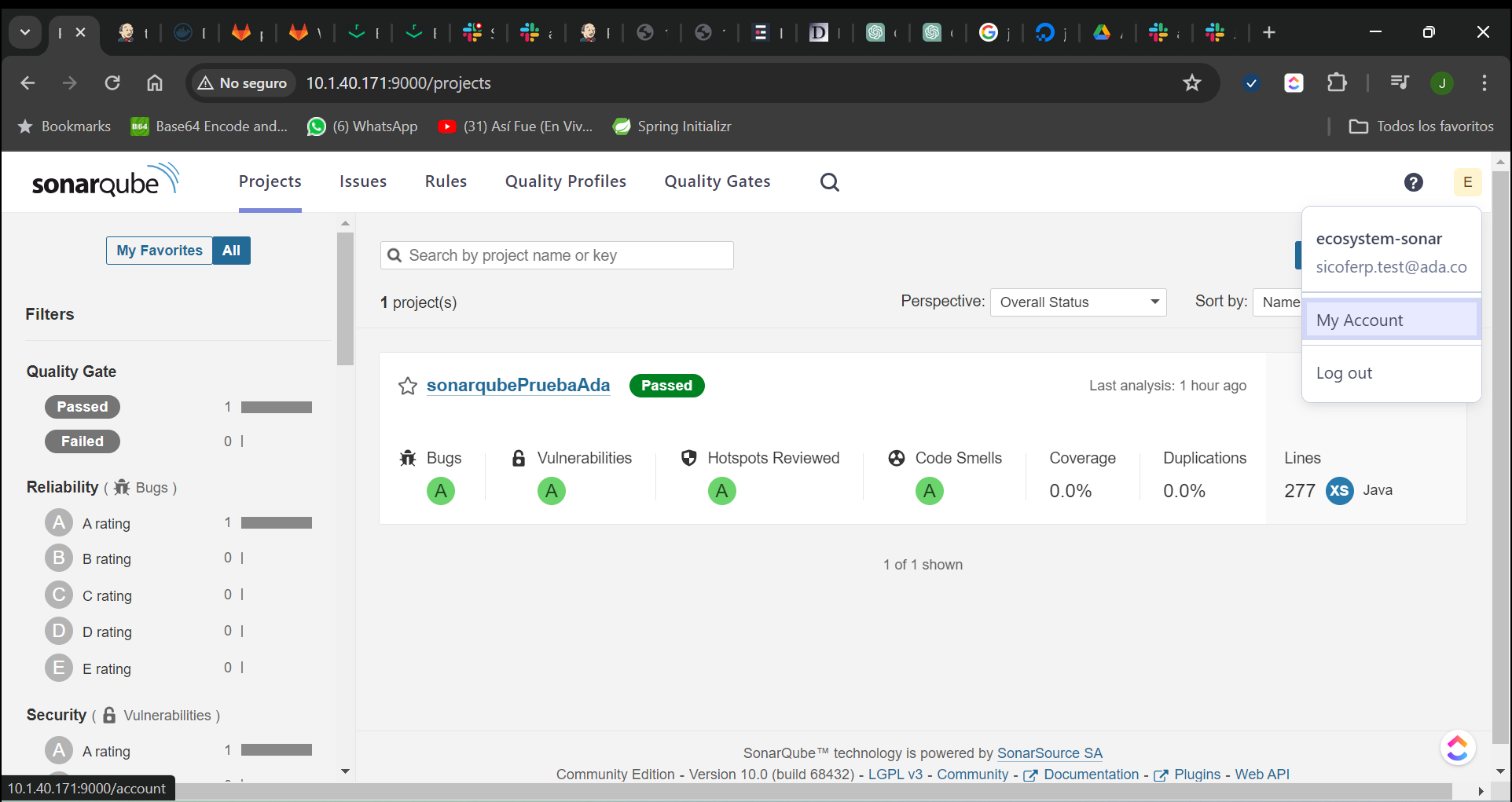
Se crea la conexión entre SonarQube(<http://10.1.40.171:9000> ) y Jenkins con el fin de que ambas aplicaciones interactúen entre sí para el proceso de CI/CD

Las credenciales del Sonar son:

Usuario: ecosystem-sonar

Contraseña: ecosystem-sonar

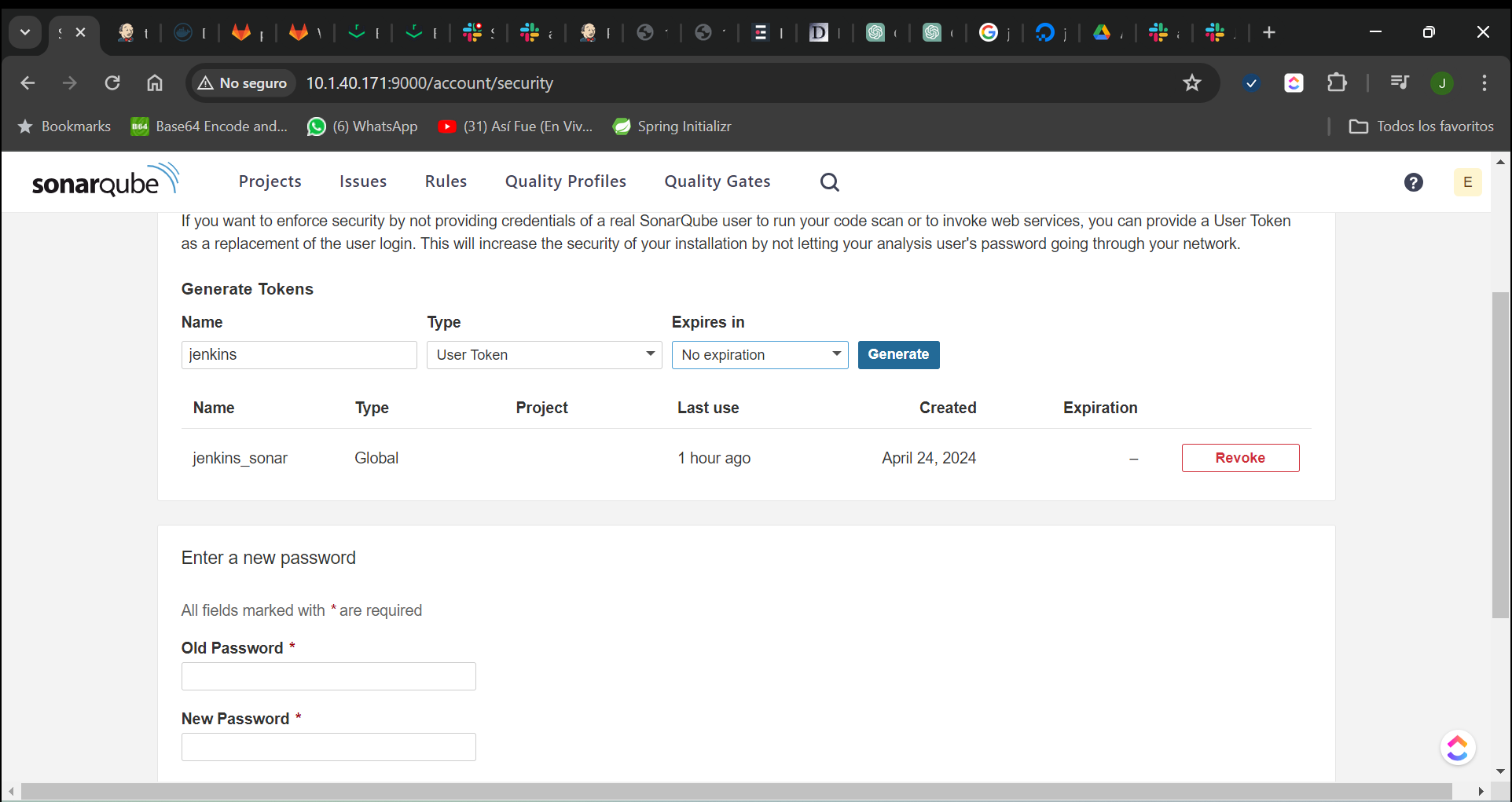
En Sonar vamos a la parte de mi cuenta / seguridad



Captura de pantalla de computadora

Descripción generada automáticamente

En esta sección vamos a generar un token con el cual nos comunicaremos desde Jenkins, al igual que los demás token se debe copiar y almacenar al crearlo ya que es la única vez que éste será visible

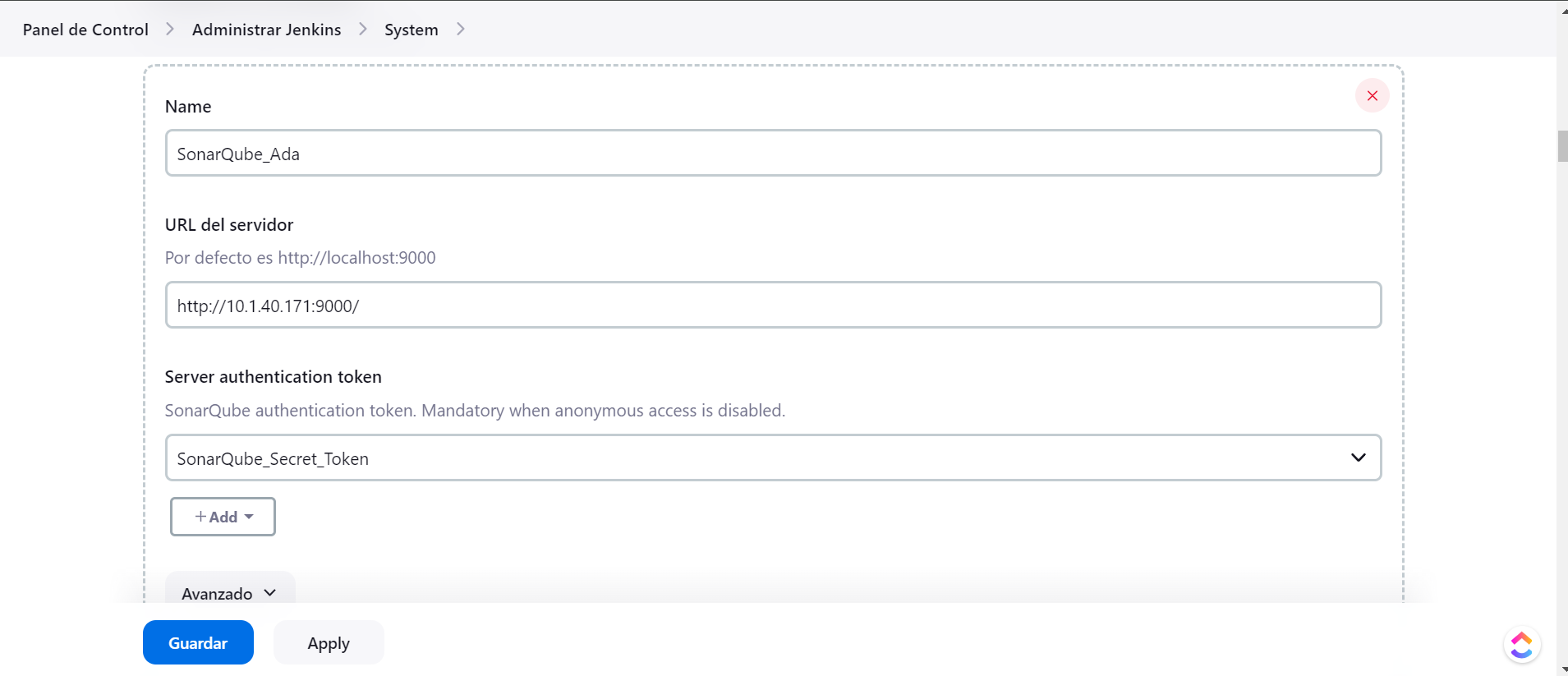


* En Jenkins vamos a la sección Panel de Control / Administrar Jenkins / Plugins, e instalamos el siguiente plugin, el cual nos permitirá realizar la conexión con sonarqube

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Una vez instalado dicho Plugin, nos dirigimos a la siguiente sección Panel de Control / Administrar Jenkins / System para poder configurarlo, nos dirigimos a la opción SonarQube servers.



Donde se indica el nombre de la conexión, la url de sonar y el token que generamos en sonar para crear la comunicación entre ambas herramientas. Se da click en Apply y guardar.

Adicionalmente nos dirigimos a esta sección en Jenkins, Panel de Control / Administrar Jenkins / Tools y vamos hasta la opcion instalaciones de SonarQube Scanner, damos click en Añadir SonarQube Scanner, seleccionamos la versión de Sonar y damos click en Apply y en Save.

Tabla

Descripción generada automáticamente

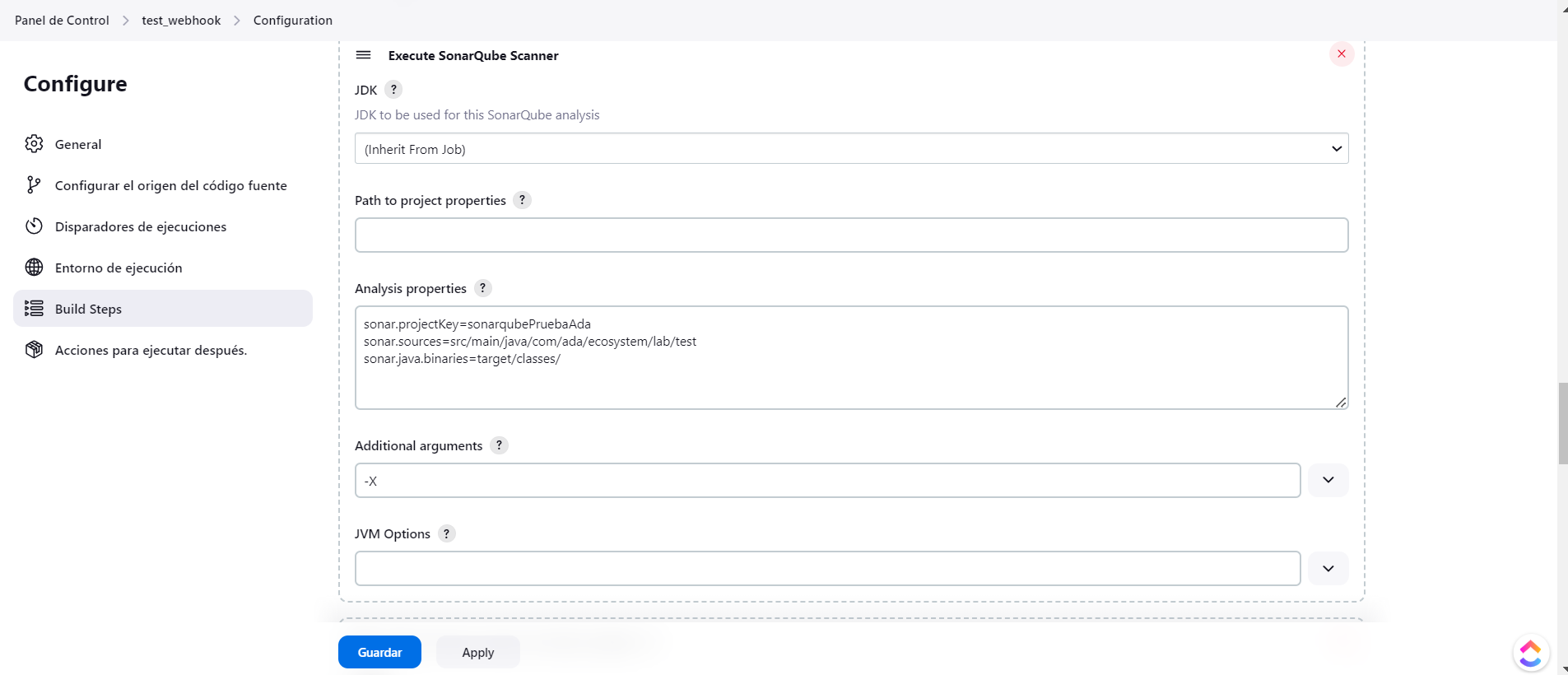
De esta manera queda configurado Jenkins con sonarqube y se comunican entre sí.

* Ahora procedemos a configurar un paso del pipeline para que interactúe con SonarQube. Nos dirigimos a esta sección del pipeline en Jenkins, Panel de Control / test\_webhook / Configuration y adicionamos un paso el cual se llama Execute SonarQube Scanner

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Una vez seleccionado lo configuramos de la siguiente manera, damos click en Apply y Guardar:



El ejecutar el Pipeline se evidencia las estadísticas del Proyecto en SonarQube

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

***NOTA: “Se hace la aclaración que en la fase I de la migración para la aplicación SICOF, las reglas de SonarQube se encuentran apagadas, se aconseja definirlas y activarlas para poder garantizar la Calidad de código por medio de los análisis que realiza sonar”***

# **CREACIÓN DE UNA IMAGEN DESDE JENKINS HACIA DOCKER HUB**

Para las creaciones de las imágenes Dockerizadas desde Jenkins, se abre una cuenta en Docker Hub puesto que allí es donde el Pipeline realizará el despliegue de las mismas. La cuenta creada en Docker Hub (<https://hub.docker.com/>) tiene las siguientes credenciales:

Correo: [ecosystem.user@ada.co](mailto:ecosystem.user@ada.co)

Usuario: ecosystemuser

Contraseña: ecosystem2024\*

## **¿QUÉ ES DOCKER HUB?**

Docker Hub es un servicio en la nube que permite a los usuarios almacenar, distribuir y gestionar imágenes de contenedores Docker. Es el registro de imágenes oficial de Docker, donde los desarrolladores pueden compartir sus imágenes de contenedores públicamente o de forma privada. Es una plataforma centralizada donde los usuarios pueden acceder a una amplia variedad de imágenes de contenedores Docker, que contienen aplicaciones y servicios listos para ser ejecutados en entornos de contenedores. Además de almacenar imágenes de contenedores, Docker Hub proporciona herramientas para colaborar en proyectos, automatizar la construcción y el despliegue de imágenes, y gestionar la seguridad de las imágenes.

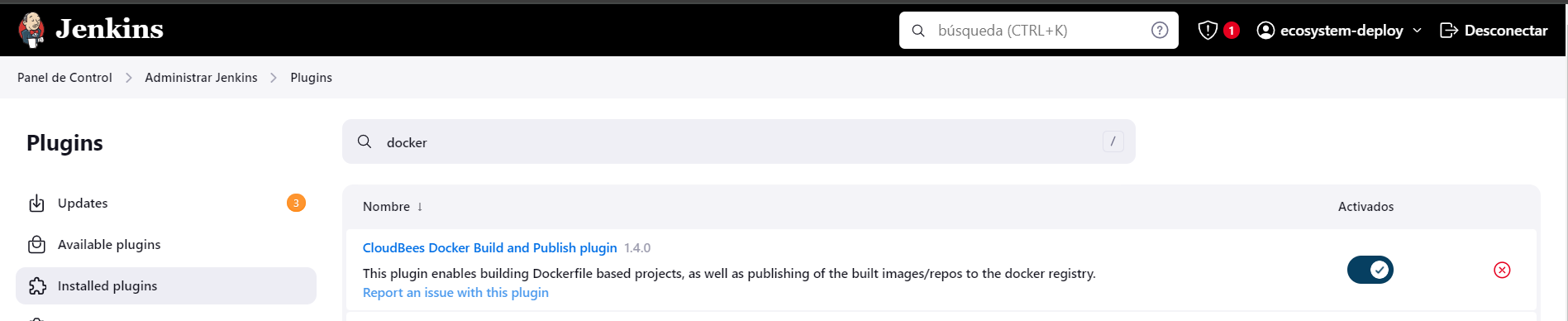
## **¿PARA QUÉ SIRVE DOCKER HUB?**

Docker Hub sirve para varios propósitos relacionados con la gestión y distribución de imágenes de contenedores Docker:

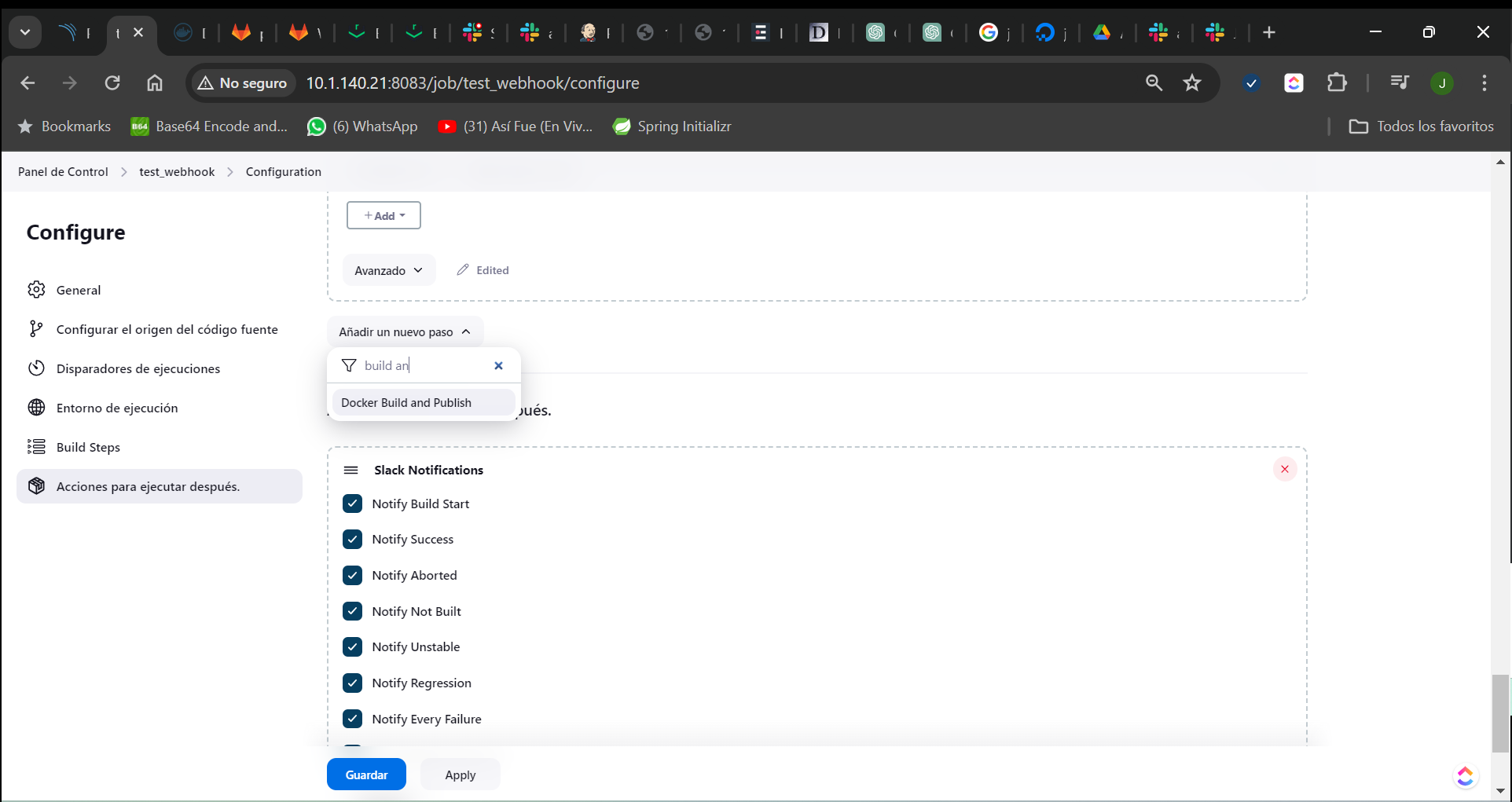
* Almacenamiento de imágenes: Los usuarios pueden almacenar sus imágenes de contenedores en Docker Hub, lo que les permite acceder a ellas desde cualquier lugar y compartirlas fácilmente con otros usuarios.
* Distribución de imágenes: Docker Hub actúa como un repositorio centralizado para la distribución de imágenes de contenedores Docker. Los usuarios pueden buscar y descargar imágenes de contenedores públicos de una amplia variedad de fuentes.
* Colaboración en proyectos: Docker Hub permite a los equipos colaborar en proyectos de desarrollo de software, proporcionando herramientas para compartir imágenes de contenedores, colaborar en la construcción de imágenes y gestionar el acceso de los miembros del equipo.
* Automatización del ciclo de vida de las imágenes: Docker Hub proporciona herramientas para automatizar el proceso de construcción, prueba y despliegue de imágenes de contenedores, lo que facilita la implementación continua y la entrega de software.
* Gestión de la seguridad: Docker Hub ofrece funciones de seguridad integradas, como el escaneo de vulnerabilidades en imágenes de contenedores, que permiten a los usuarios identificar y remediar posibles problemas de seguridad en sus aplicaciones y servicios.

Para realizar la configuración de Jenkins con Docker Hub se deben realizar los siguientes pasos:

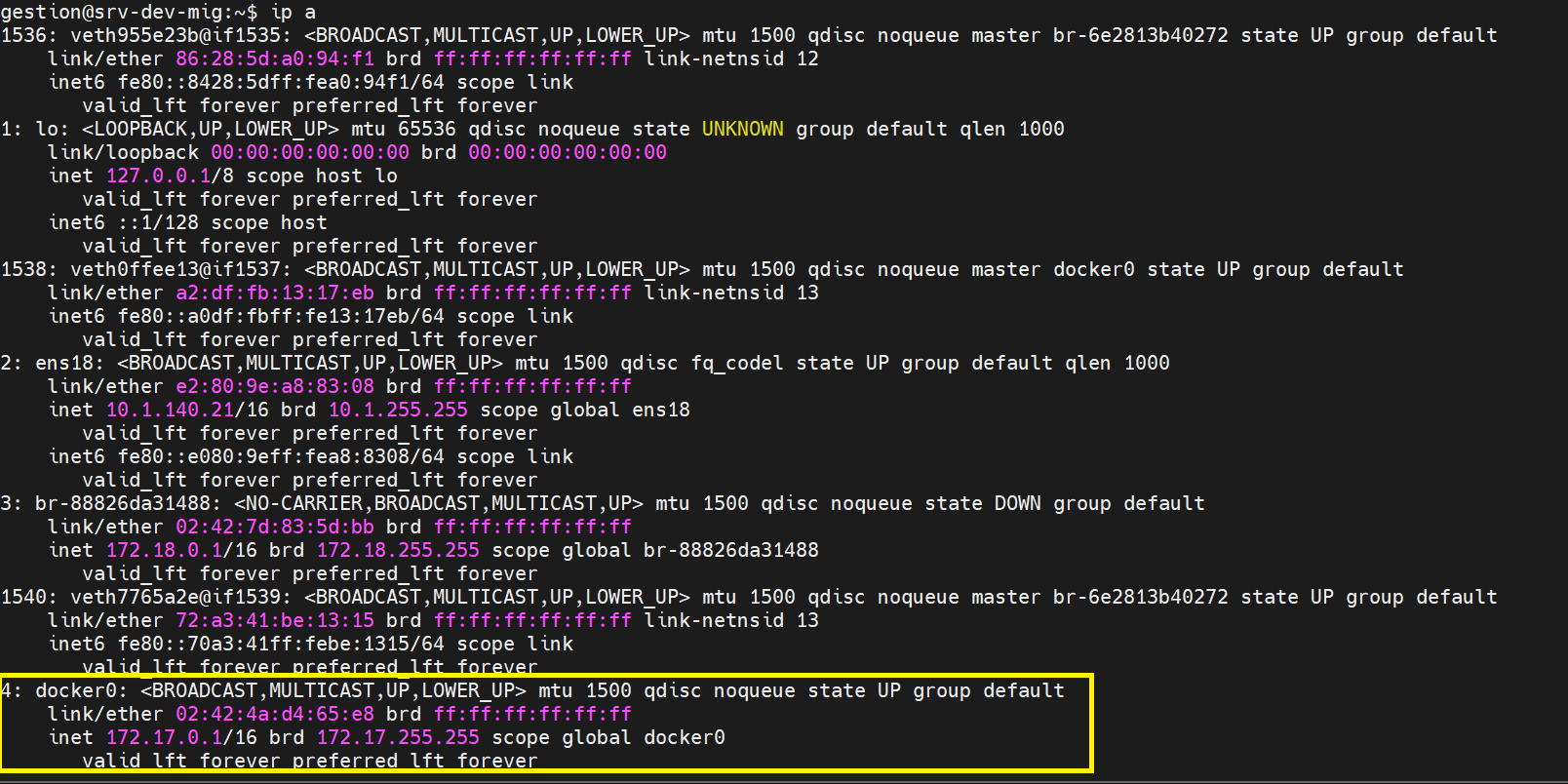
* En Jenkins se va a la sección Panel de Control / Administrar Jenkins / Plugins y se instala el siguiente plugin para que Jenkins pueda publicar imágenes en el registry de Docker Hub



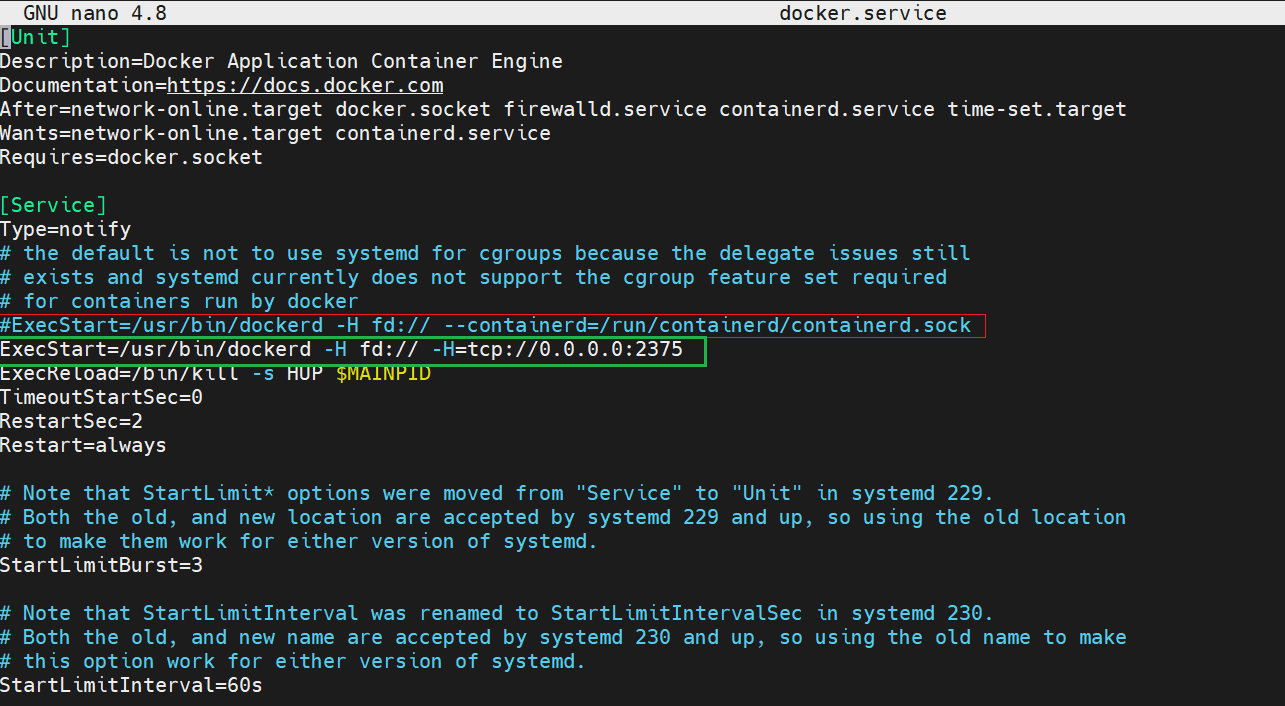
* Una vez instalado el plugin, nos dirigimos al pipeline que hemos automatizado y vamos a la sección Panel de Control / test\_webhook / Configuration, vamos a la opción acciones para ejecutar después y agregamos el paso Docker Public and Push



Para indicar la instalación de Docker Engine ejecutamos el siguiente comando ip a en la consola del servidor, buscamos la ip de docker para poder configurarla en jenkins en el campo Docker Host URI, donde le indicaremos que se conectara por el protocol TCP (Transmission Control Protocol).



Para poder habilitar dicha comunicación se debe realizar un cambio en el archivo docker.service ubicado en la ruta /usr/lib/systemd/system, se comenta la línea marcada en rojo y se crea la línea marcada en verde, con el fin de activar el protocolo tcp, se cierra el archivo, se guarda



A continuación recargamos y releemos los servicios de daemon con el siguiente comando

* sudo systemctl daemon-reload

Y para que docker tome los cambios realizados, reiniciamos el docker engine con el siguiente comando:

* sudo service docker restart

***NOTA: “Una vez reiniciado el docker engine se aconseja Validar los contenedores ya que al reiniciar algunos pueden no quedar arriba”***

Ejecutamos el siguiente comando para identificar si la configuración quedó correctamente:



Como evidenciamos el comando nos lista las imagenes que estan en el docker registry con el fin de poder publicarlas en docker hub

Una vez garantizada esta comunicación entre Jenkins y Docker Hub continuamos con la configuración del plugin en Jenkins de la siguiente manera

Tabla

Descripción generada automáticamente con confianza baja

En la opción de Registry credentials se crea una nueva credencial con el usuario y la contraseña de Docker Hub donde se va a desplegar la imagen desde el pipeline

Aclaración, a la hora de generar la imagen y el despliegue se deben agregar los archivos Dockerfile y settings.xml al Proyecto automatizado, puesto que el pipeline va a buscar dichos archivos para poder generar la imagen

En la configuración que estábamos viendo anteriormente debemos dar click en la opción advance donde se especificará la ruta del .jar y del dockerfile

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

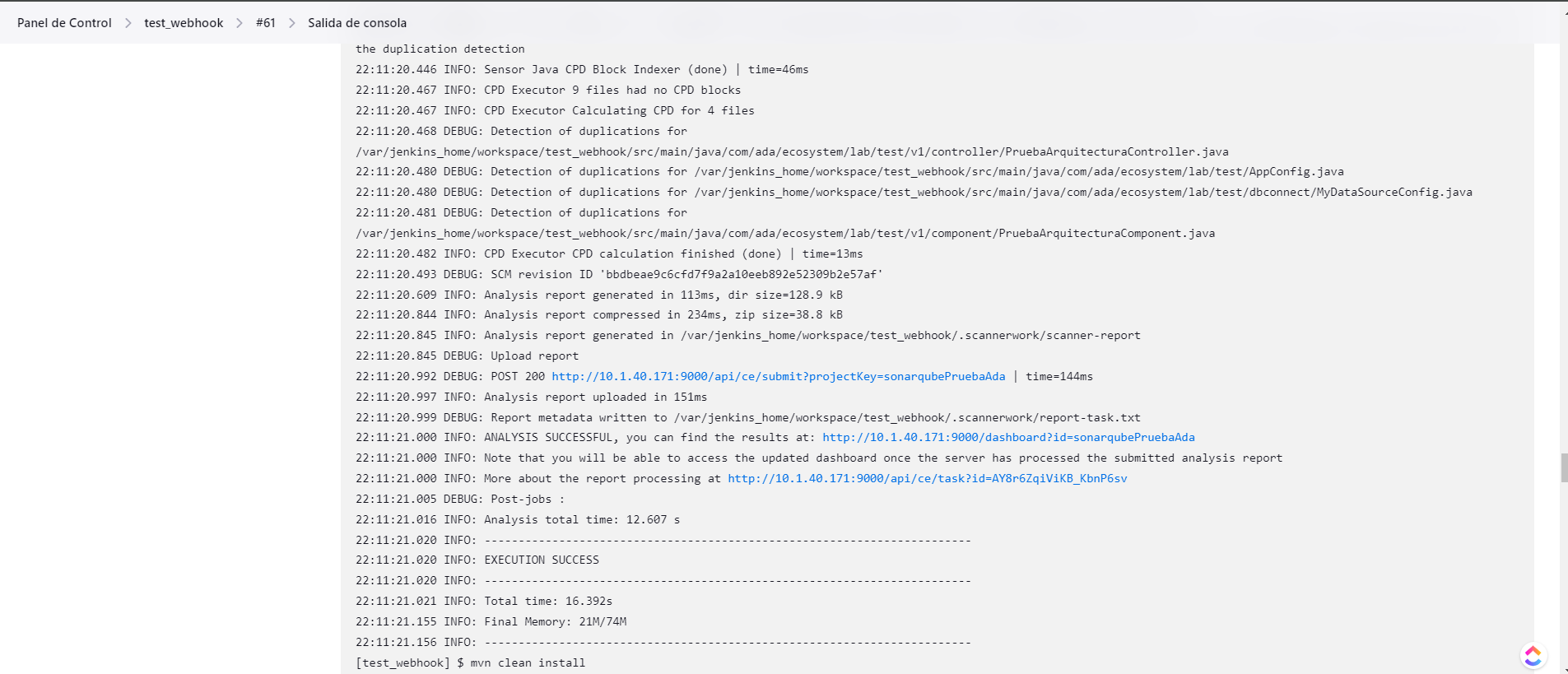
En la opción Build Context, se especifica dónde se encuentra el archivo Dockerfile con el cual se generará la imagen y en la opción Additional Build Arguments se debe especificar dónde queda creado el archivo .jar con el fin que el pipeline lo despliegue como imagen en Docker Hub

Interfaz de usuario gráfica, Texto, Aplicación

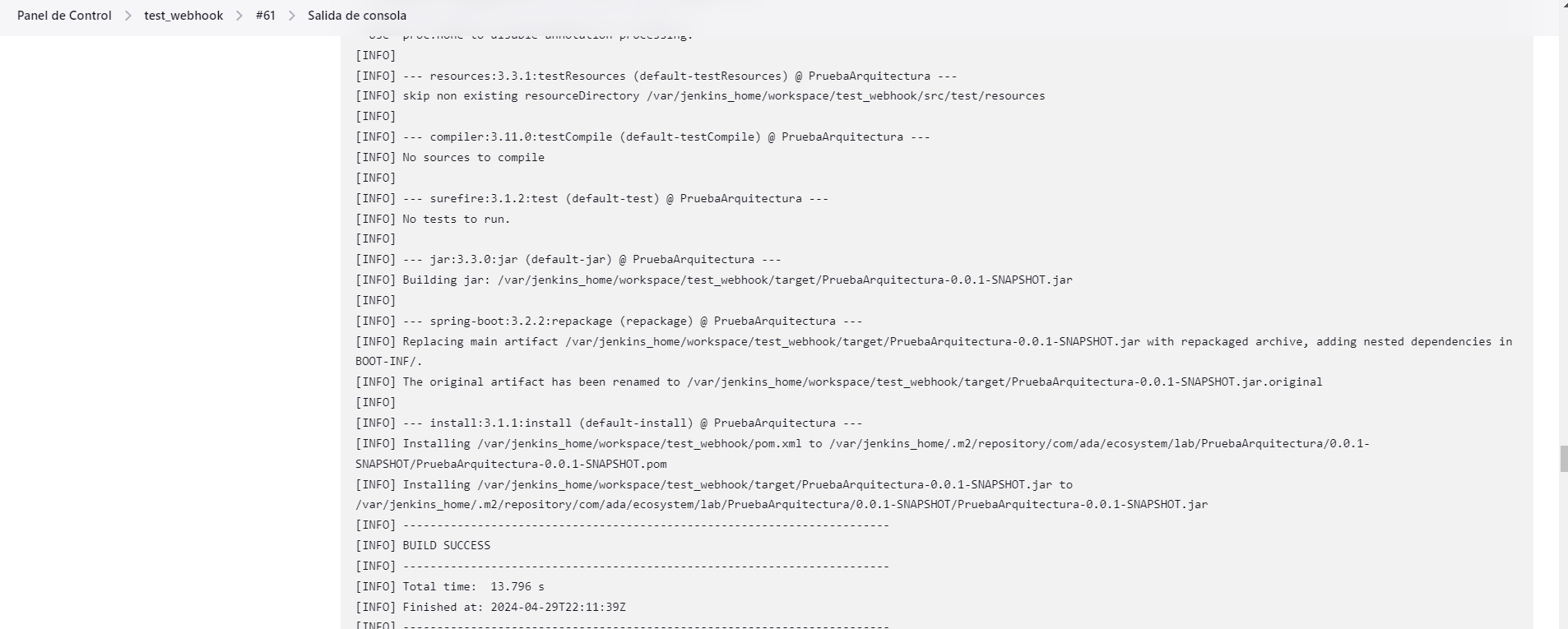
Descripción generada automáticamente

Por último se ejecuta el pipeline y se evidencia que Jenkins ejecuta todos los pasos configurados y despliega la imagen en el repositorio de Docker Hub como se configuró

Se evidencia que el pipeline conecta, ejecuta y pasa sonar correctamente



El pipeline compila e instala correctamente el componente con maven, generando el archivo .jar



El pipeline genera y publica la imagen correctamente

Texto

Descripción generada automáticamente

Por último el pipeline envía las notificaciones al slack cómo se configure

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Se identifica que el despliegue de la imagen en Docker Hub queda realizado correctamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# **¿QUE ES EL DESPLIEGUE CONTINUO (CD)?**

El despliegue continuo es una práctica en el desarrollo de software que consiste en automatizar el proceso de entrega de aplicaciones a entornos de producción de manera frecuente y sin interrupciones. En otras palabras, implica la automatización de la implementación de cambios en el código de una aplicación, de manera que estos cambios se desplieguen automáticamente en un entorno de producción tan pronto como se realicen y se superen las pruebas necesarias.

En el contexto de la Integración Continua (CI) y el Despliegue Continuo (CD), el despliegue continuo es una extensión natural de la CI. Mientras que la CI se enfoca en la integración frecuente de cambios en el código y la ejecución de pruebas automáticas, el despliegue continuo va un paso más allá al automatizar la entrega de esos cambios a los usuarios finales.

El despliegue continuo tiene varios beneficios, incluyendo:

* **Rápida retroalimentación:** Permite a los equipos obtener rápidamente comentarios sobre los cambios realizados en el código, lo que facilita la detección temprana de errores y la iteración rápida.
* **Reducción del tiempo de entrega:** Al automatizar el proceso de implementación, se reduce significativamente el tiempo necesario para entregar nuevas funcionalidades y correcciones a los usuarios finales.
* **Mayor confiabilidad:** Al automatizar el despliegue, se minimiza el riesgo de errores humanos y se garantiza que las implementaciones se realicen de manera consistente y confiable.
* **Mayor productividad:** Al liberar a los equipos de tareas manuales repetitivas, el despliegue continuo permite que se centren en la creación de valor añadido y en la mejora de la calidad del software.