**1- Poniendo en funcionamiento Jenkins**

* Crear una imagen de Docker que se base en la imagen oficial de Jenkins y que tenga instalado .NET Core. Crear un archivo llamado Dockerfile.jenkins con el siguiente contenido:

FROM jenkins/jenkins:lts

USER root

# Instala dependencias necesarias

RUN apt-get update && apt-get install -y \

apt-transport-https \

software-properties-common \

wget

# Agrega el repositorio de Microsoft y actualiza

RUN wget -q https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb && \

dpkg -i packages-microsoft-prod.deb && \

apt-get update

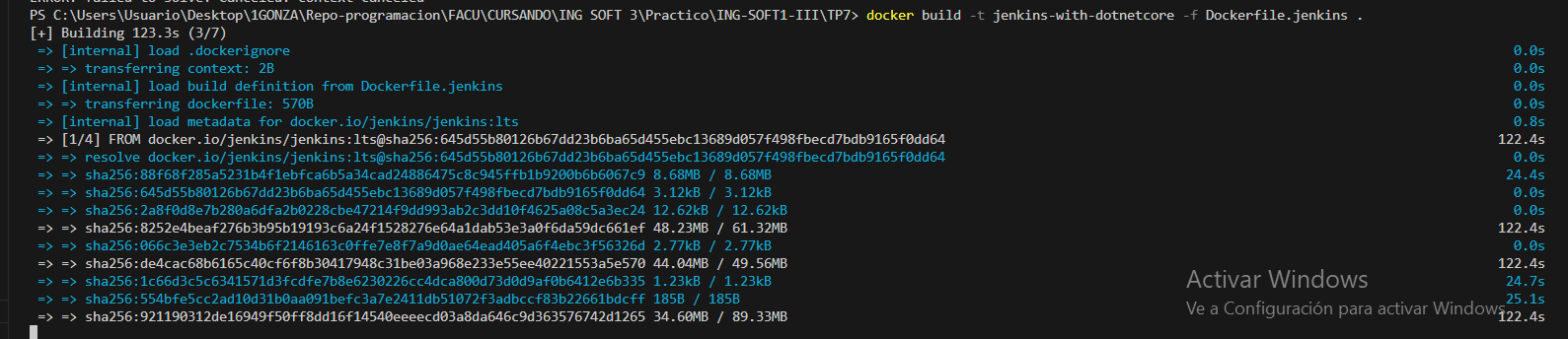
# Instala el SDK de .NET Core

RUN apt-get install -y dotnet-sdk-7.0

USER jenkins

* Desde la misma ubicación donde tengas el archivo Dockerfile personalizado, ejecuta el siguiente comando para construir una nueva imagen de Docker con .NET Core y Jenkins. Esto creará una nueva imagen de Docker llamada jenkins-with-dotnetcore:

docker build -t jenkins-with-dotnetcore -f Dockerfile.jenkins .



* Ejecuta un Contenedor con la Nueva Imagen. Ahora, puedes ejecutar un contenedor utilizando la imagen que acabas de crear:

# Linux / Mac OS

mkdir -p ~/jenkins

docker run -d -p 8080:8080 -p 50000:50000 --name jenkins \

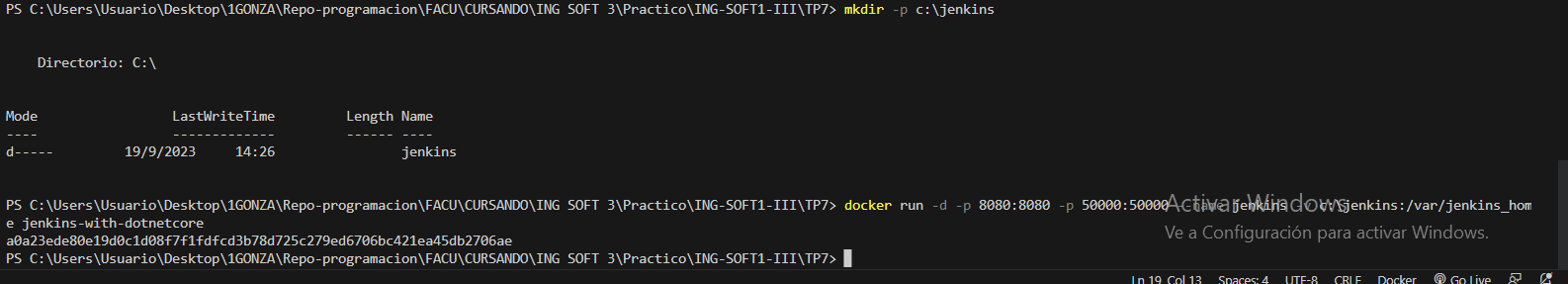
-v jenkins\_home:/var/jenkins\_home \

jenkins-with-dotnetcore

# windows

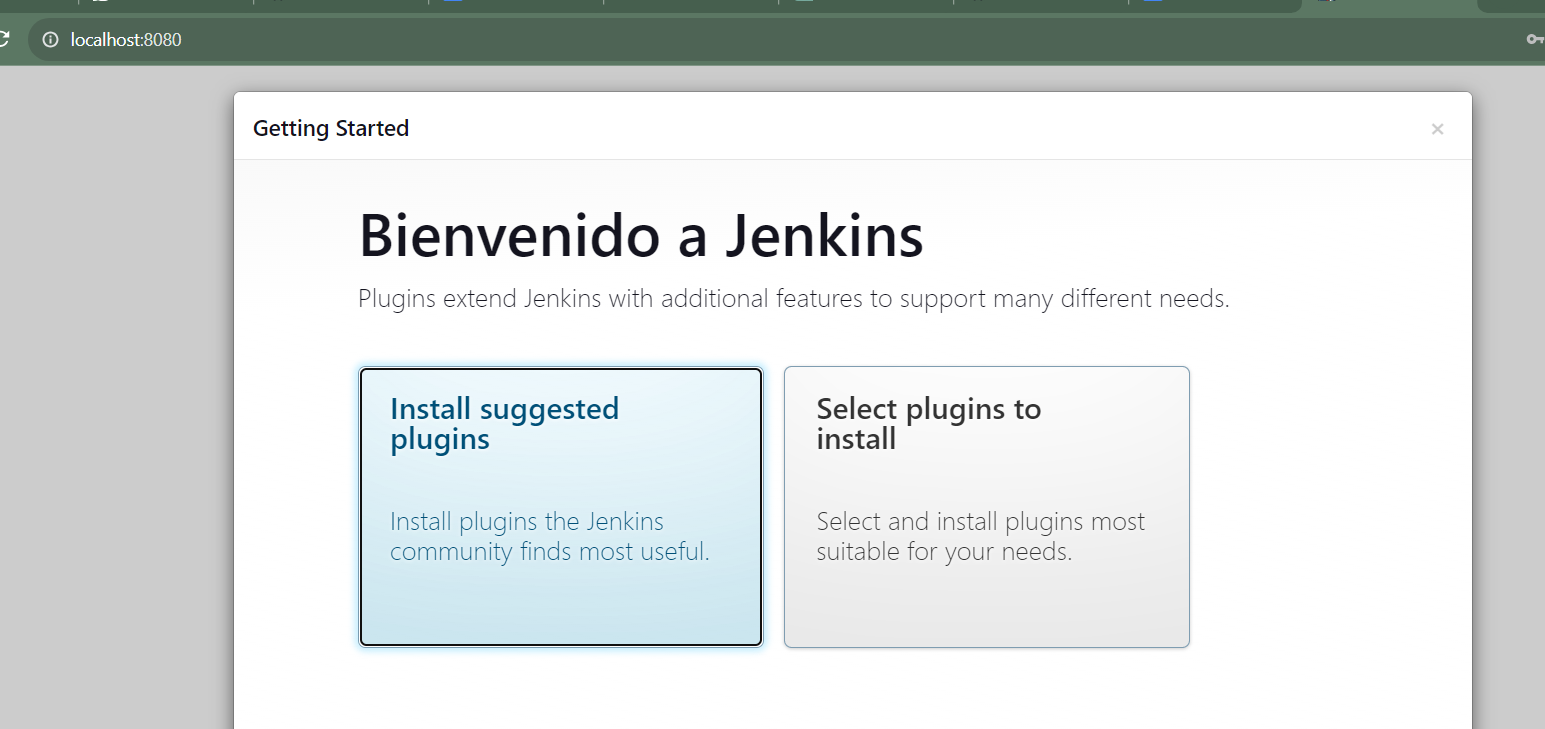
mkdir -p c:\jenkins

docker run -d -p 8080:8080 -p 50000:50000 --name jenkins -v c:\jenkins:/var/jenkins\_home jenkins-with-dotnetcore

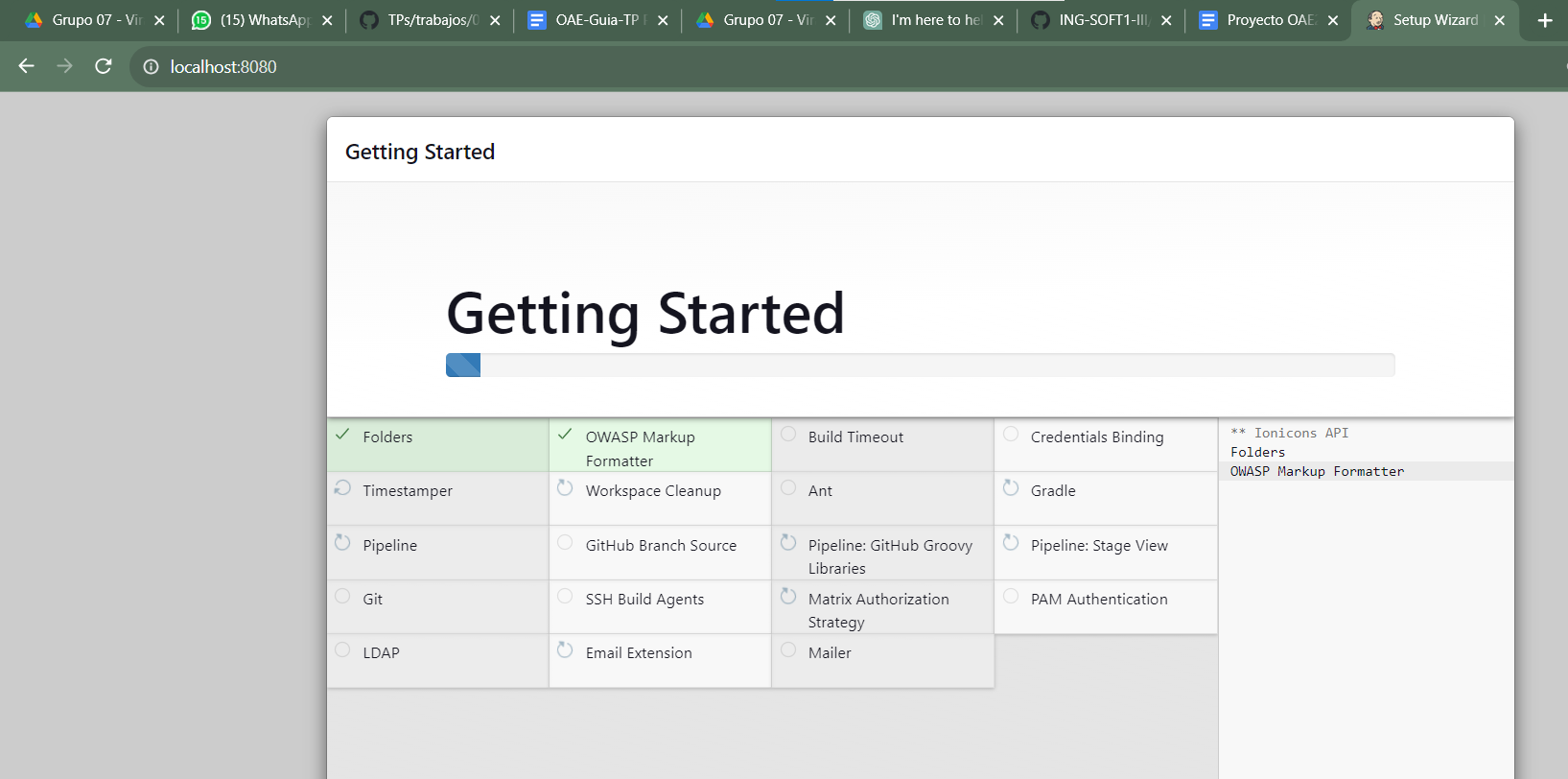


* Una vez en ejecución, abrir [http://localhost:8080](http://localhost:8080/)
* Inicialmente deberá especificar el texto que se encuentra dentro del archivo ~/jenkins/secrets/initialAdminPassword **en el contenedor**

cat ~/jenkins/secrets/initialAdminPassword



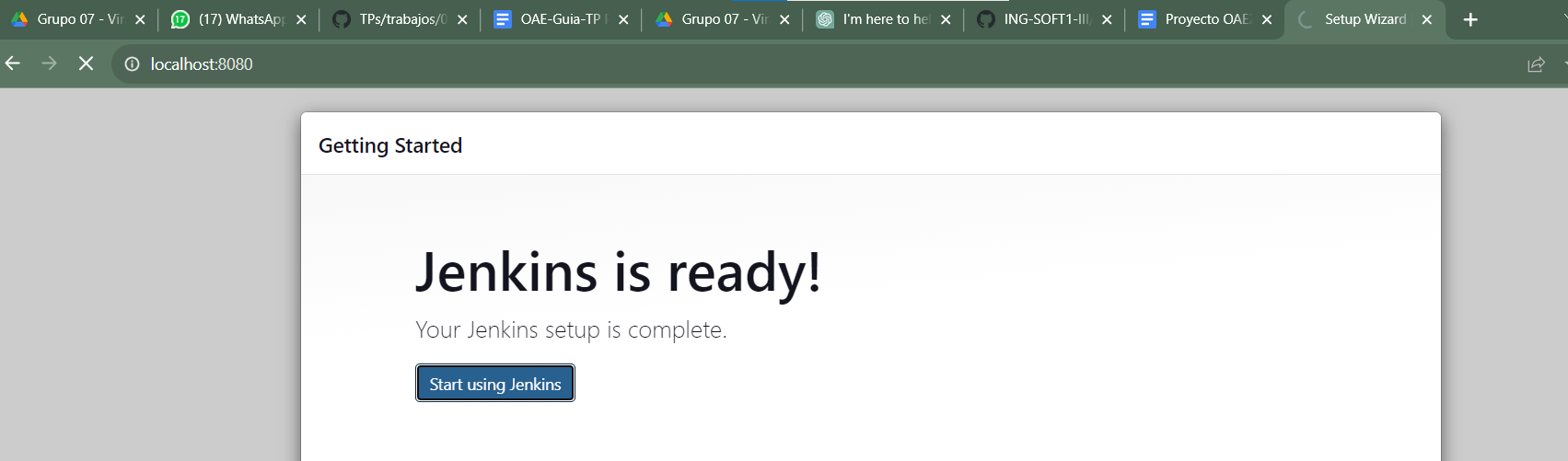
* Instalar los plugins por defecto

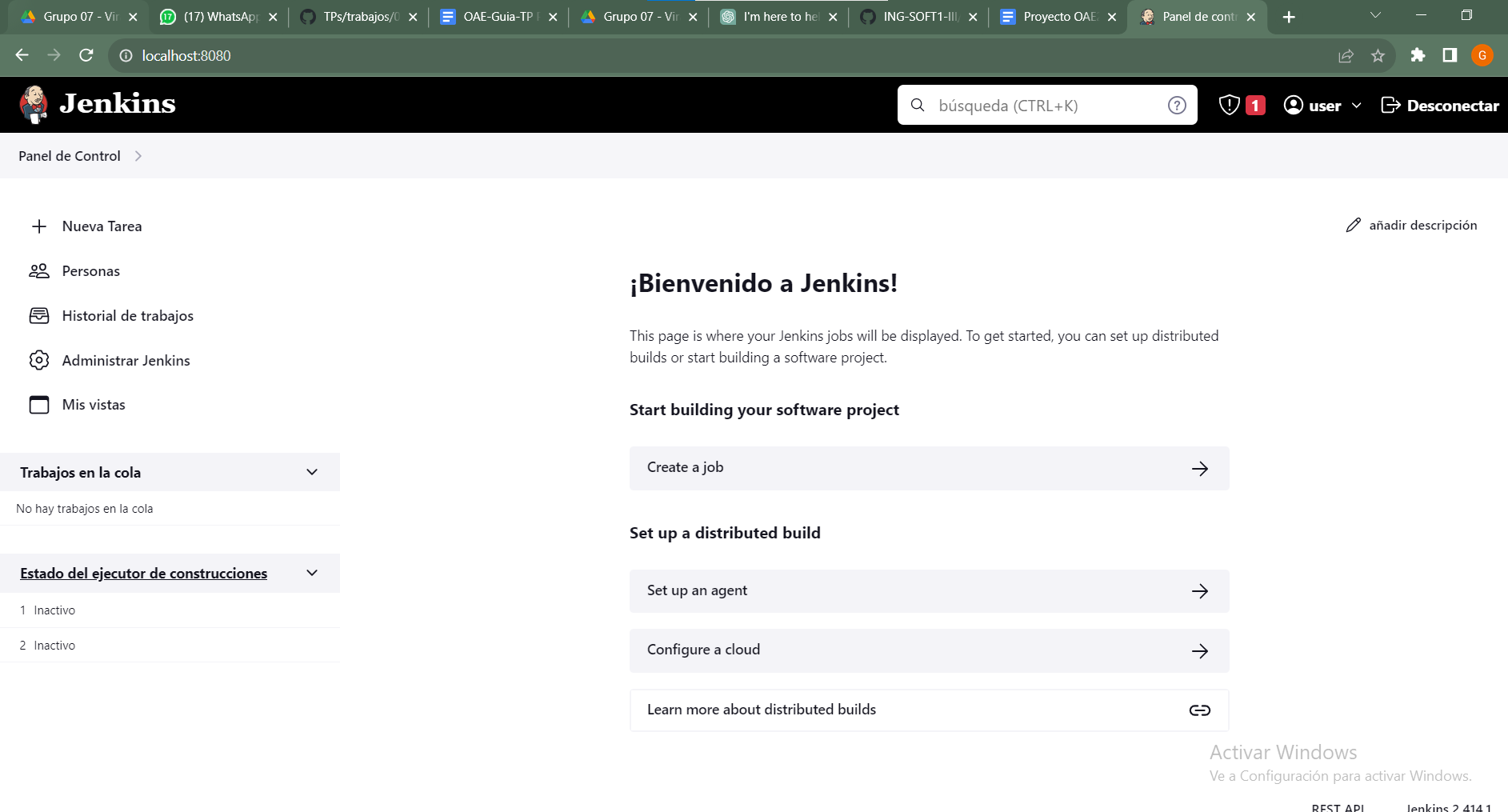


* Crear el usuario admin inicial. Colocar cualquier valor que considere adecuado.

User : user

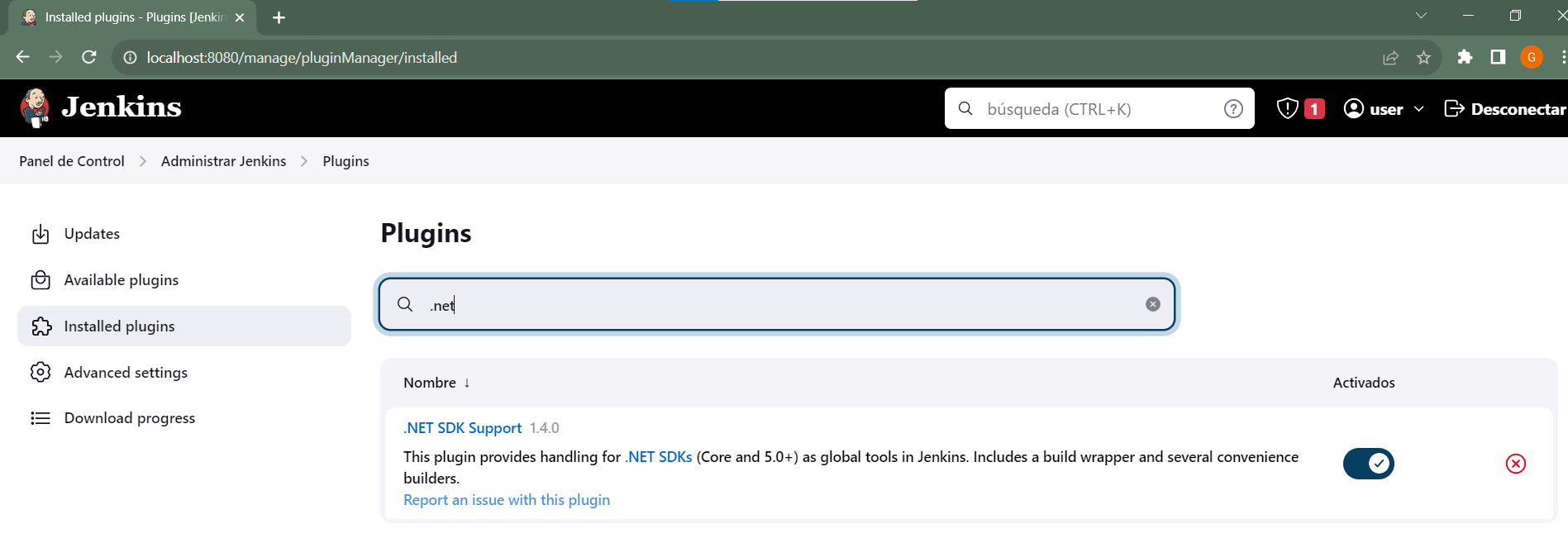
Password: user



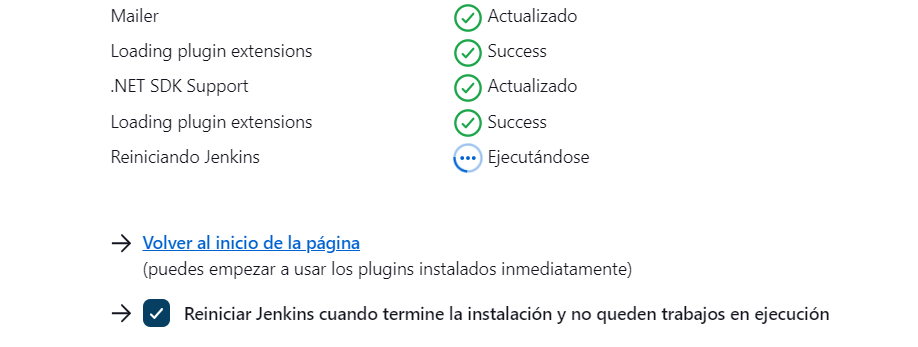


**2- Instalando Plugins y configurando herramientas**

* En Administrar Jenkins vamos a la sección de Administrar Plugins
* De la lista de plugins disponibles instalamos .NET SDK Support



* Reiniciamos el servidor



* Abrir nuevamente página de Plugins y explorar la lista, para familiarizarse qué tipo de plugins hay disponibles.

**Tipos de Jobs**

En Jenkins, los proyectos de estilo libre (también conocidos como proyectos de construcción de estilo libre o freestyle projects) y los pipelines son dos enfoques diferentes para crear y configurar trabajos de automatización. **Proyecto de Estilo Libre (Freestyle Project):**

-Interfaz Gráfica Amigable: Los proyectos de estilo libre utilizan una interfaz gráfica basada en formularios y menús desplegables que facilita la configuración del trabajo sin necesidad de escribir código o scripts.

-Configuración Basada en Pasos: En un proyecto de estilo libre, configuras el trabajo mediante una serie de pasos predefinidos, como "Construir", "Publicar", "Notificar", etc. Cada paso se configura utilizando opciones y campos específicos.

-Facilidad para Usuarios No Técnicos: Los proyectos de estilo libre son ideales para usuarios que no están familiarizados con la creación de scripts o que desean una configuración rápida y simple.

-Limitaciones en la Lógica de Flujo: Pueden resultar limitados para implementar flujos de trabajo más complejos y condicionales, ya que la lógica de ejecución se basa principalmente en la configuración de pasos secuenciales.

**Pipeline:**

-Definición como Código: Los pipelines se definen como código en un DSL (lenguaje específico de dominio) llamado "Pipeline DSL". Esto significa que defines tu flujo de trabajo como un script de Jenkinsfile, generalmente escrito en Groovy.

-Mayor Flexibilidad: Los pipelines ofrecen una mayor flexibilidad y capacidad para crear flujos de trabajo altamente personalizados y condicionales. Puedes utilizar lógica de programación y estructuras de control en tu Jenkinsfile.

-Gestión de Versiones: Los Jenkinsfiles se pueden gestionar en sistemas de control de versiones (como Git), lo que permite un control de versiones completo y seguimiento de cambios en tus flujos de trabajo.

-Reutilización y Modularidad: Puedes reutilizar secciones de código en múltiples trabajos de Jenkins al utilizar funciones y definiciones compartidas en tus Jenkinsfiles.

-Mayor Escalabilidad: Los pipelines son más adecuados para proyectos grandes y complejos, donde se requiere una gestión más avanzada de flujos de trabajo, paralelismo y manejo de errores.

-En resumen, la principal diferencia radica en la forma en que se configuran y gestionan los trabajos de Jenkins. Los proyectos de estilo libre son más simples y se basan en una configuración gráfica, mientras que los pipelines son más flexibles, versátiles y se definen como código, lo que los hace ideales para flujos de trabajo más complejos y para equipos de desarrollo que prefieren la gestión de código fuente para sus flujos de trabajo de Jenkins. La elección entre uno u otro dependerá de tus necesidades y preferencias específicas.

**3- Creando el primer Job de estilo libre**

* Crear un nuevo item, del tipo estilo libre con nombre **first-job**
* Una vez creado el job, en la sección Build Steps seleccionamos **Ejecutar linea de comandos (shell)** y escribimos:

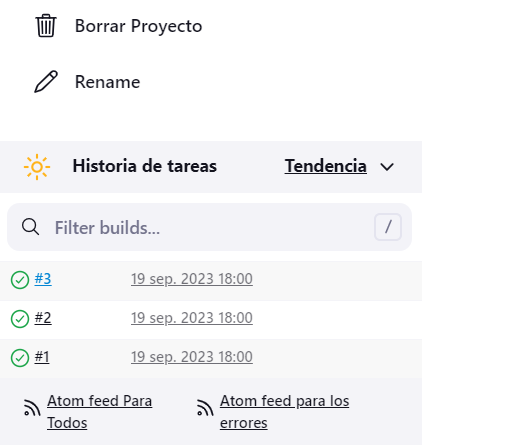
[](https://user-images.githubusercontent.com/140459109/267107552-b086b436-c41f-4d9b-be65-e8649b4a85e1.png)

current\_datetime=$(date +"%Y-%m-%d %H:%M:%S")

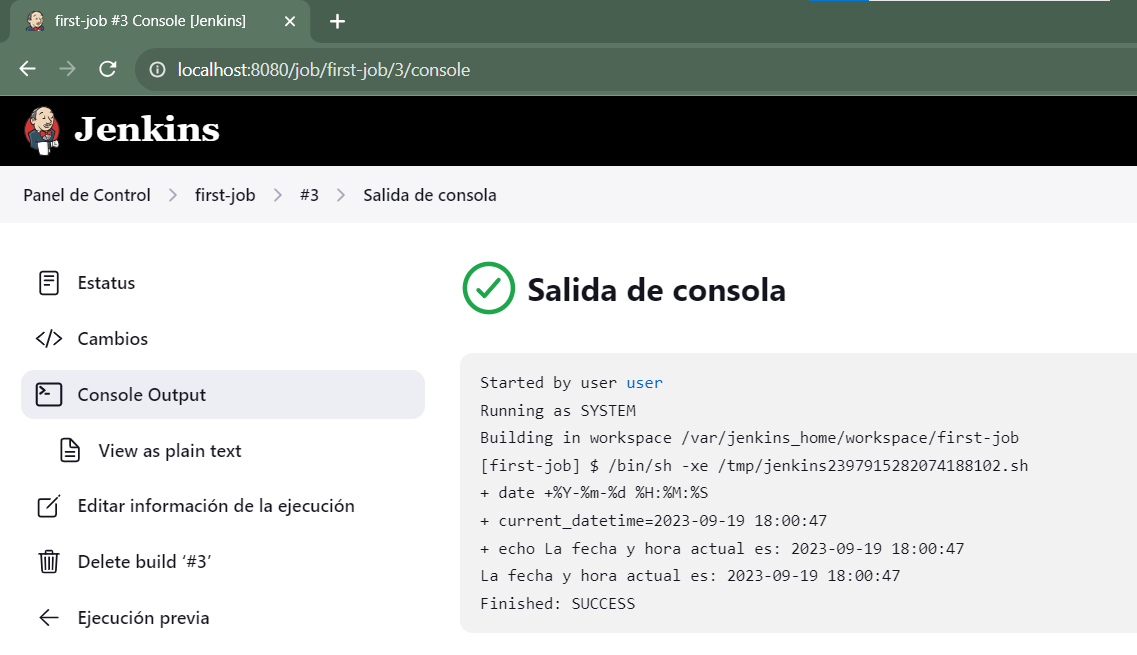
Imprime la fecha y hora actual utilizando el comando echo

echo "La fecha y hora actual es: $current\_datetime"

* Guardamos y ejecutamos el Job

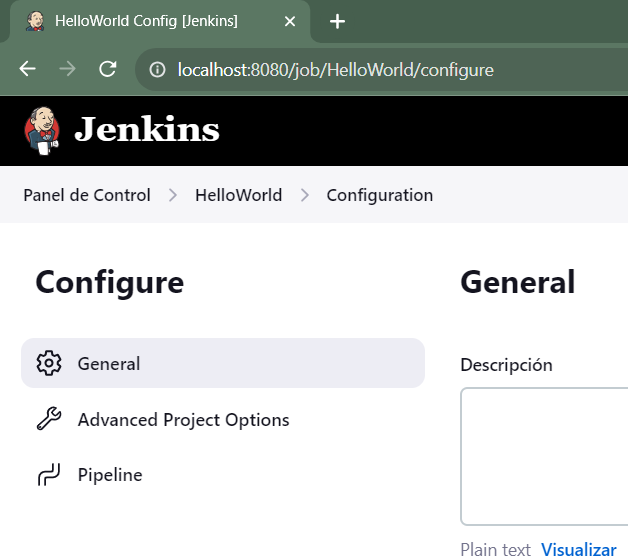


* Analizar la salida del mismo

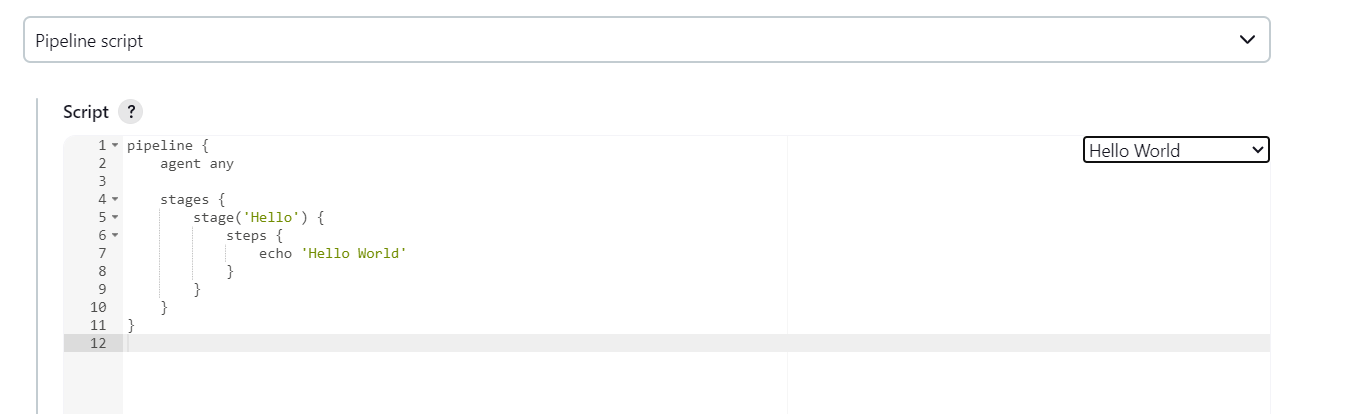


**4- Creando el primer Pipeline Job**

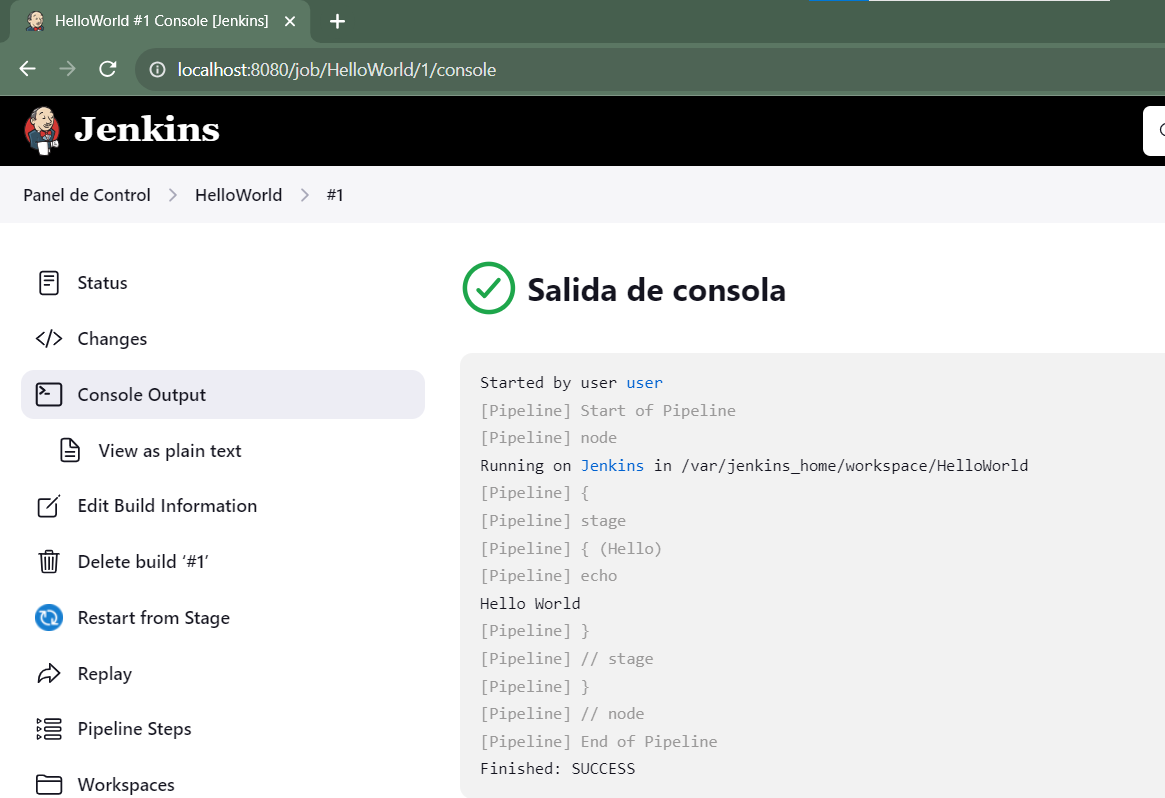
* Crear un nuevo item, del tipo Pipeline con nombre **hello-world**



* Una vez creado el job, en la sección Pipeline seleccionamos **try sample Pipeline** y luego **Hello World**



* Guardamos y ejecutamos el Job
* Analizar la salida del mismo



**5- Creando un Pipeline Job con Git**

* Similar al paso anterior creamos un ítem con el nombre **github-job**
* En script escibir:

pipeline {

agent any

stages {

stage('Build') {

steps {0

// Get some code from a GitHub repository

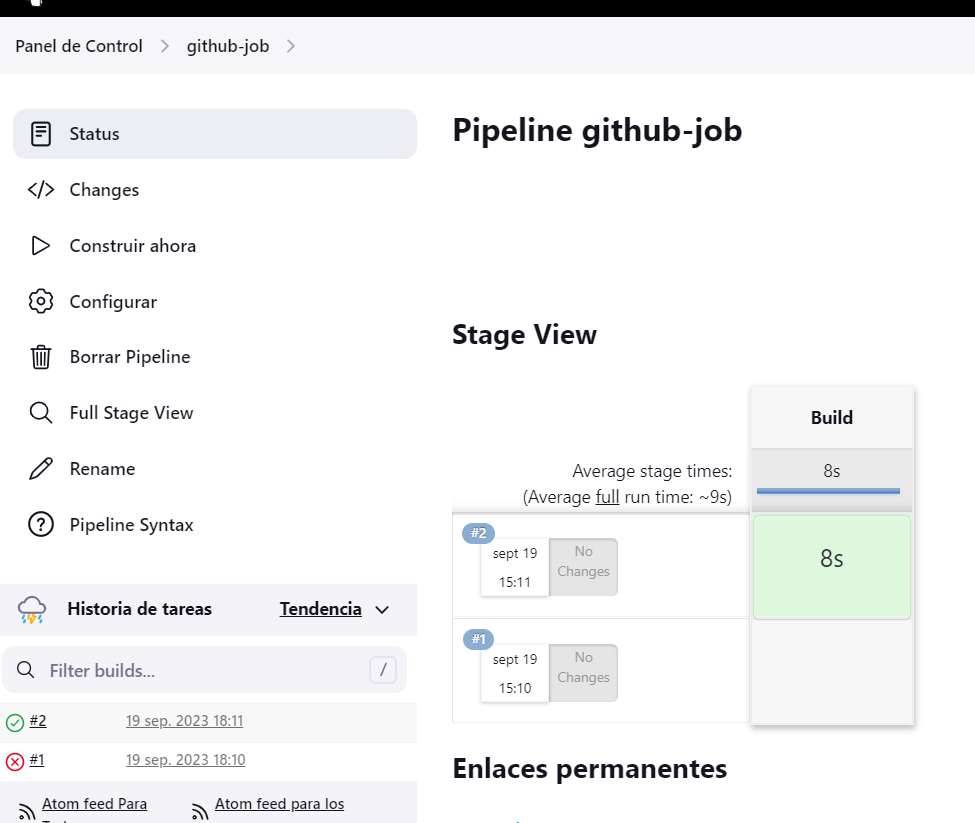
git branch: 'main', url: 'https://github.com/ingsoft3ucc/SimpleWebAPI'

}

}

}

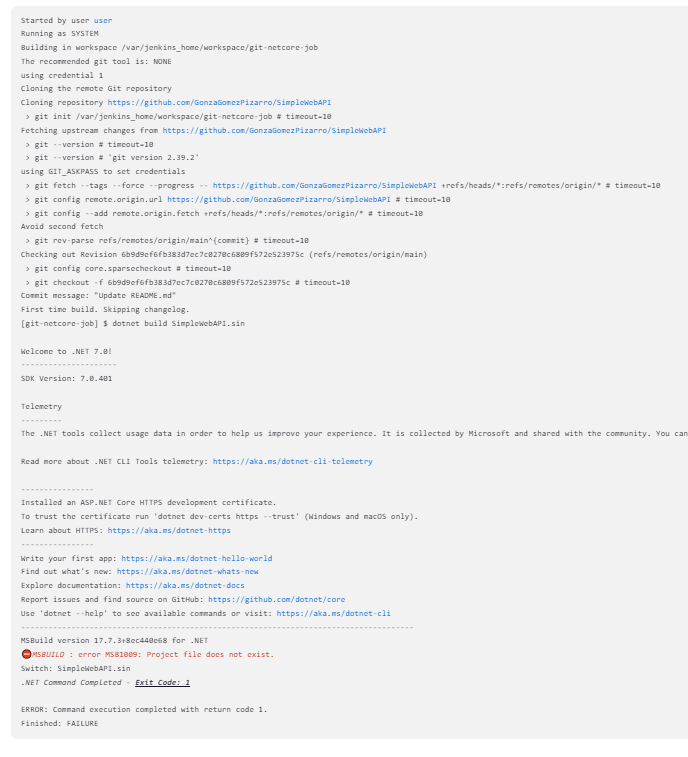
* Guardar y ejecutar el Job



**- Utilizando nuestros proyectos.**

**6.1- .NET Core**

* Crear un Job de estilo libre llamado "**git-netcore-job**" que construya el proyecto en .NET Core del ejercicio 1 del TP 05.
* Configurar el Job para obtener el código desde el repositorio de cada alumno:
* Agregar las credenciales correspondientes:
* Indicar la rama desde la cual se construira
* Configurar las etapas de Build y Publish del proyecto
* Corremos el Job y revisamos la salida.



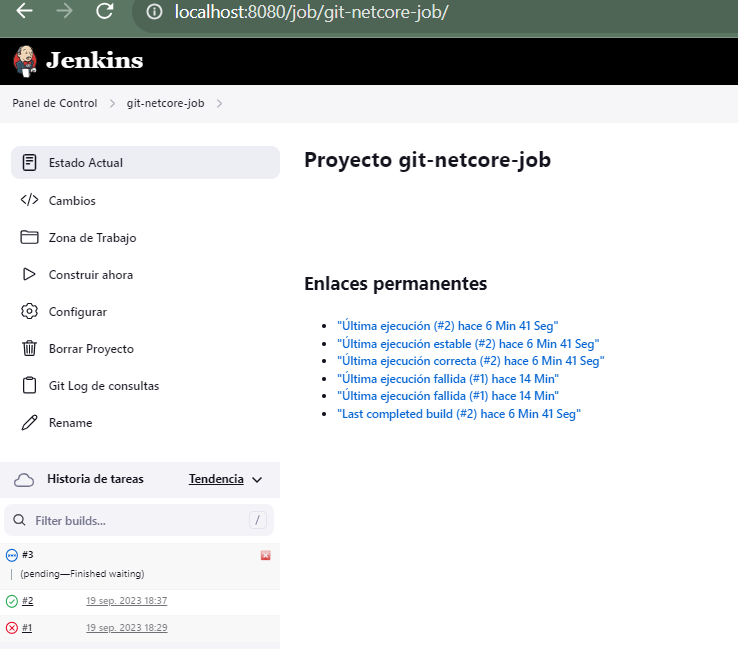


**6.2- Monitorear Cambios en el repo de GitHub**

[](https://user-images.githubusercontent.com/140459109/268796446-ecbd6b73-7009-4da1-b54b-905a603a0ae9.png)

NOTA -> CRONOS SYNTAX

* Configurar Poll cada 1 minuto
* Verificar su funcionamiento al hacer un commit y cuando no se hizo commit



* Explicar por que no es posible tener un WebHook en nuestro TP

Por que seria mas un ejercicio de redes que un ejercicio para explicar Jenkins. El servicio de Jenkins debería tener una ip publica para que git le pueda avisar que ha habido un cambio

* Explicar Diferencia con ejecutar periódicamente:

Que ejecutar periodicamente hace eso, ejecuta el build periodicamente, en cambio el poll solo ejecuta si github le responde que ha habido un cambio, mientras tanto solo consulta periodicamente

**6.3- Node.js**

* Repetir el ejercicio 6.1 para construir el proyecto del ejercicio 2 del TP 05.