UD3 - Despliegue y Automatización con Jenkins

Cristóbal Suárez Abad

2º ASIR

Instrucciones Generales

Entregar:

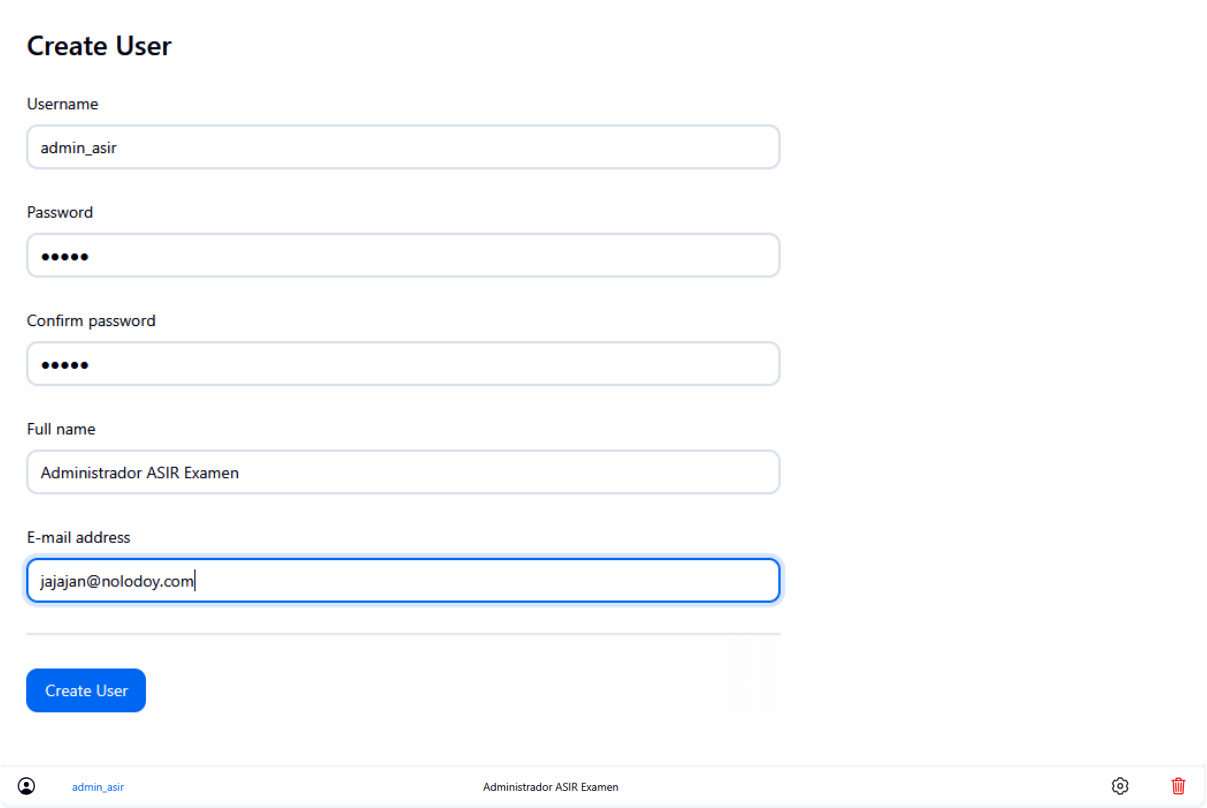
- Un documento PDF con las capturas de pantalla que evidencien el funcionamiento de cada apartado
- Los ficheros de configuración utilizados (Jenkinsfile, scripts, etc.).
- **Muestra capturas y explicaciones de la configuración realizada durante todo el proceso, además demuestra que todo funciona como esperamo**s.

Parte 1: Gestión de Usuarios y Seguridad (1,5 puntos)

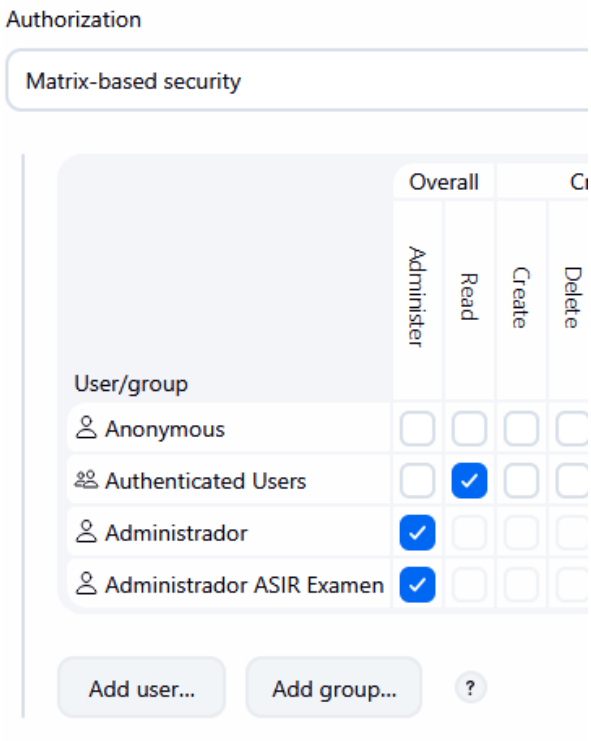
Objetivo: Aplicar el modelo de seguridad basado en roles.

1. **Cree un primer usuario administrador con el nombre `admin_asir` y una contraseña segura.**

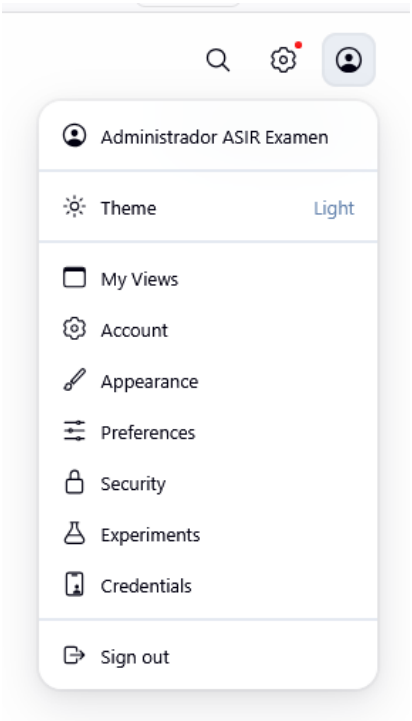
Creamos el usuario: Administrar Jenkins → Users → Crear usuario.



Le damos privilegios de Administrador: **Administrar Jenkins → Security → Authentication:**

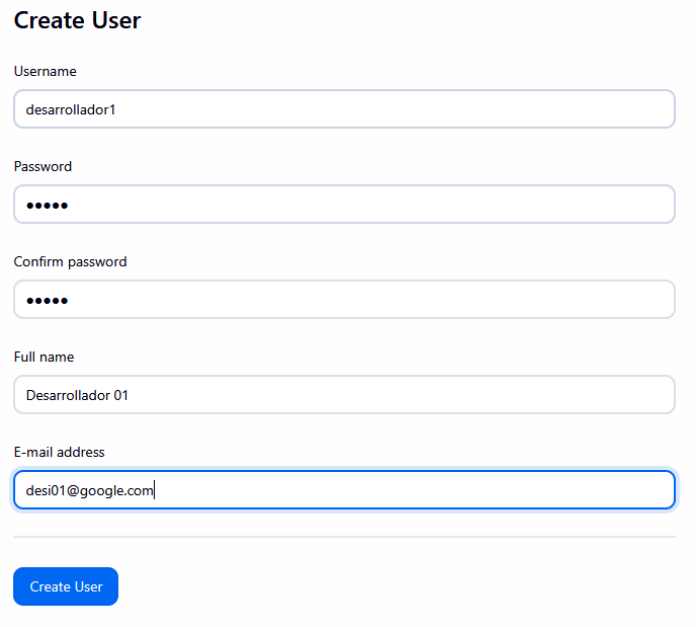


Cerramos la sesión actual y entramos con el nuevo Administrador:

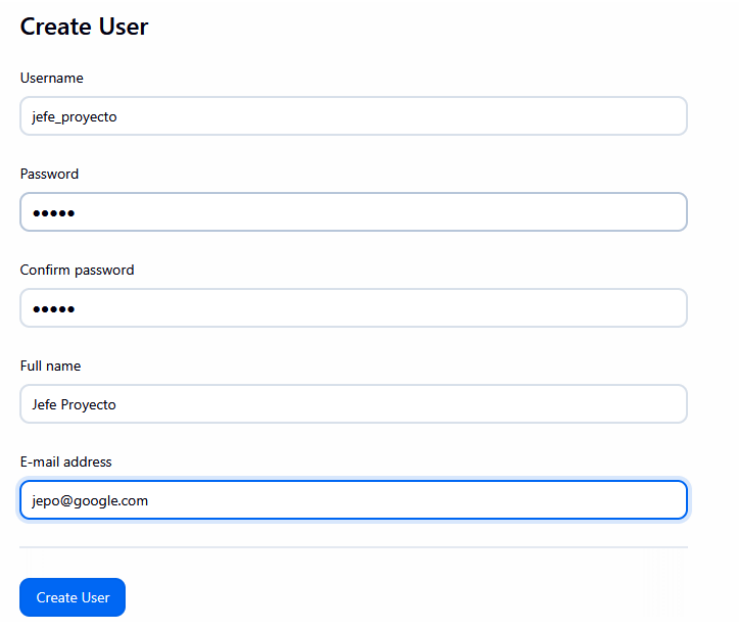


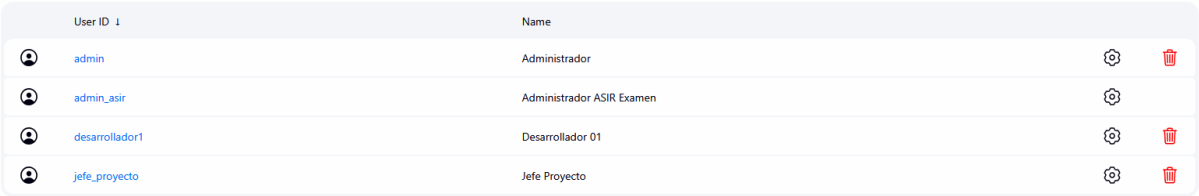
2. **Creación de Usuarios:**
    - Cree dos nuevos usuarios: `desarrollador1` y `jefe_proyecto`.

- **`desarrollador1`**

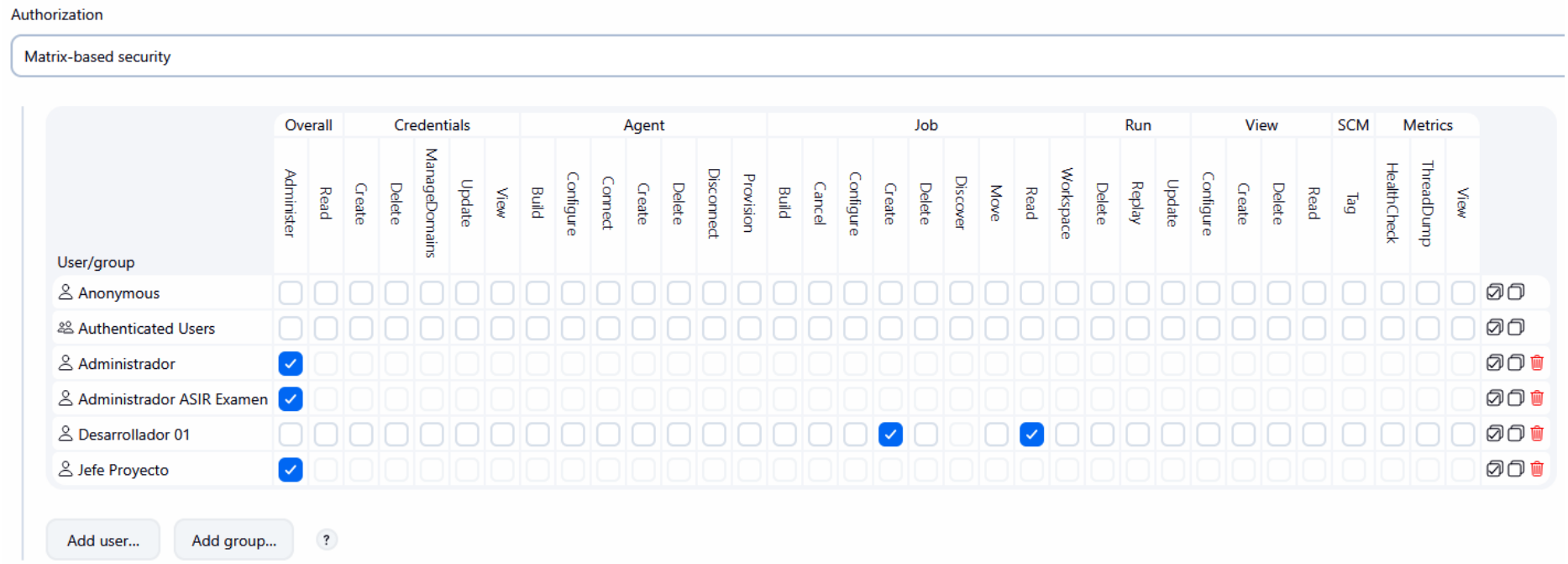


- **`jefe_proyecto`**





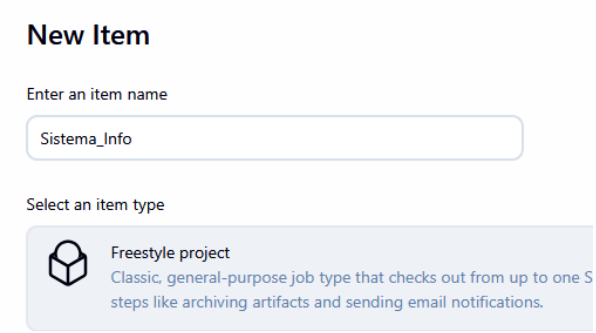
3. **Matriz de Autorización:**
    - Configure la seguridad global para que:
        - **Usuarios anónimos:** No tengan acceso de lectura ni ejecución.
        - **jefe_proyecto:** Tenga control total (Admin) sobre Jenkins.
        - **desarrollador1:** Solo tenga permisos para **Ver** (Read) y **Construir** (Build) tareas, pero NO para configurar Jenkins ni gestionar plugins.



Parte 2: Freestyle Jobs y Shell Scripting (1,5 puntos)

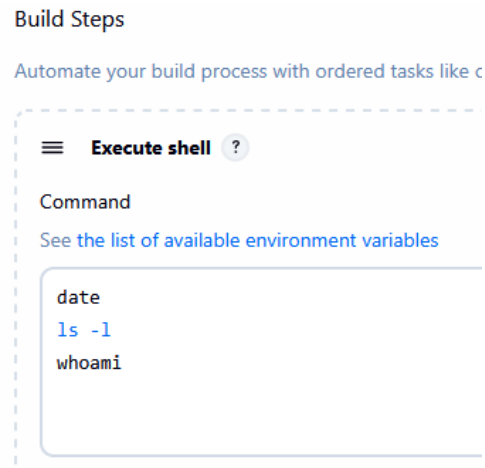
Objetivo: Crear tareas básicas de automatización.

1. **Job "Sistema_Info":**



   - Añade un script de shell que realice lo siguiente:
     - Muestre la fecha y hora actual.
     - Liste los archivos del directorio de trabajo (workspace).
     - Muestre el usuario del sistema que está ejecutando el job (`whoami`).

Usamos comandos de Linux:
"date" nos da la hora.
"ls" lista objetos en un directorio.
"`whoami`" indica el usuario que tiene la sesión.



Console Output

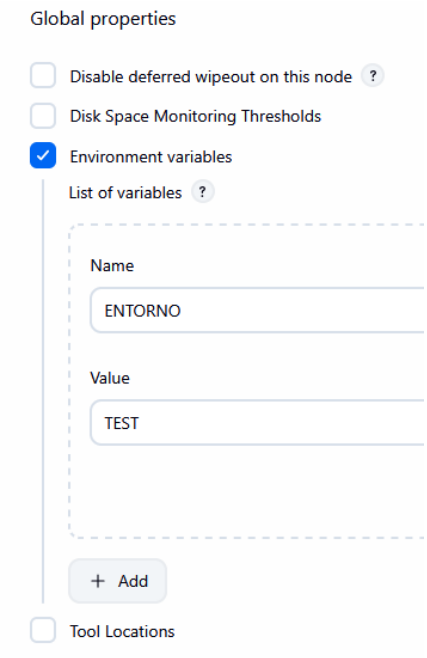
```
Started by user Administrador ASIR Examen
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Sistema_Info
[Sistema_Info] $ /bin/sh -xe /tmp/jenkins15838757071345453215.sh
+ date
Fri Dec 12 08:49:32 UTC 2025
+ ls -l
total 0
+ whoami
jenkins
Finished: SUCCESS
```

2. **Parámetros:**
    - Modifique el job para que acepte un **parámetro de texto** llamado `ENTORNO` (con valor por defecto "TEST").
    - Haga que el script imprima: *"Desplegando en el entorno: $ENTORNO"*.

Tenemos que crearlo:

Podemos crear variables de entorno personalizadas:

Administrar Jenkins → System → Propiedades Globales: Marca la casilla de “Variables de entorno” → Añadir.



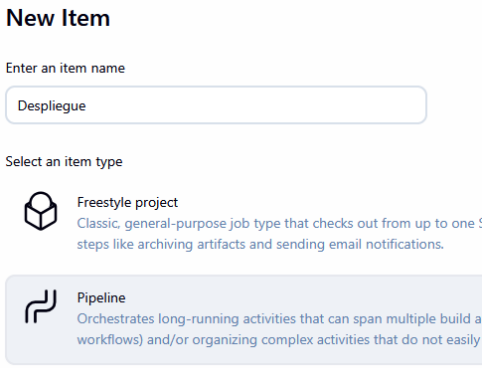
Ahora incluimos un “echo” con la sentencia que se especifica en el enunciado.

Console Output

```
Started by user Administrador ASIR Examen
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Sistema_Info
[Sistema_Info] $ /bin/sh -xe /tmp/jenkins10331023766246928213.sh
+ date
Fri Dec 12 08:54:59 UTC 2025
+ ls -l
total 0
+ whoami
jenkins
+ echo Desplegando en el entorno: TEST
Desplegando en el entorno: TEST
Finished: SUCCESS
```

Parte 3: Pipelines y Git (3 puntos)

Objetivo: Implementar un pipeline llamado “Despliegue”



1. Usa el siguiente repositorio https://github.com/octocat/Hello-World.git. No es necesario credenciales
2. Diseñe un pipeline con las siguientes etapas (*Stages*):
    - **Stage 'Build':** Imprima un mensaje "Compilando aplicación..." y descarga el repositorio indicado en la carpeta de trabajo.

Los pipelines dividen sus acciones en “stage” y dentro de estas están los pasos o “steps”. En este caso se hace un “echo” y un comando simple de git para descargarnos el repositorio, al no necesitar credenciales.

```
stage('Build') {
    steps {
        echo "Compilando aplicación..."
        // Clona el repositorio en el workspace
        git url: 'https://github.com/octocat/Hello-World.git'
    }
}
```

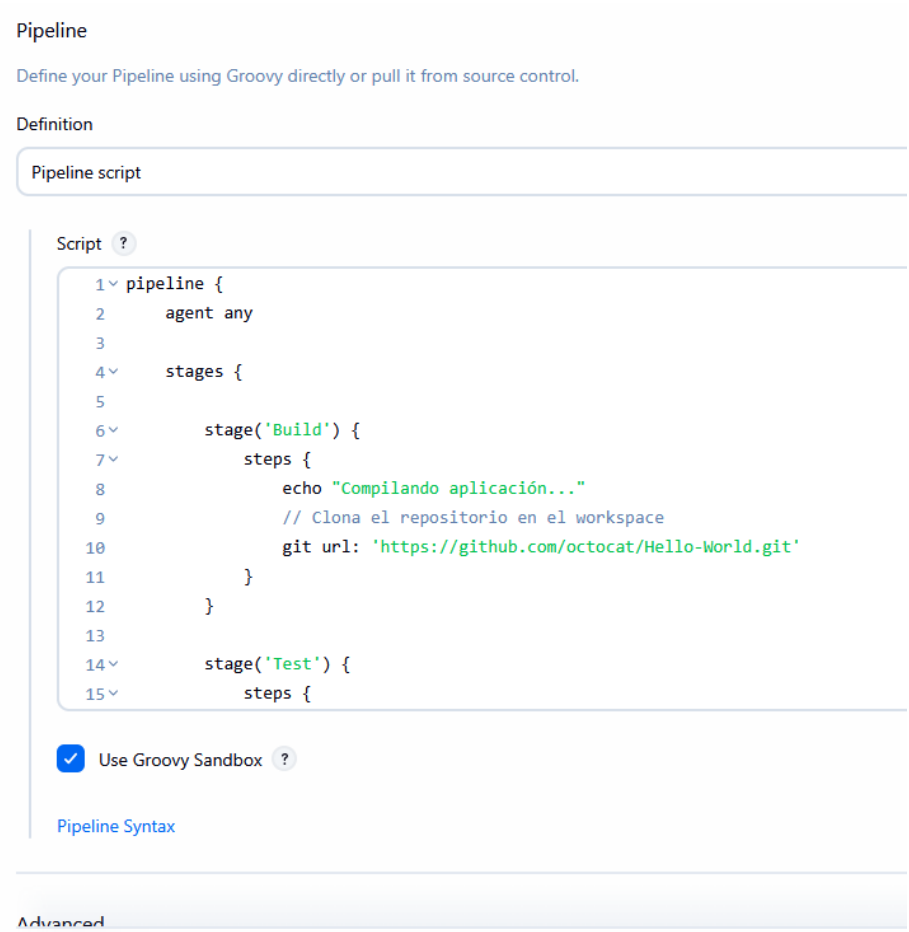
    - **Stage 'Test':** Compruebe si el archivo `README` existe. Si existe, el paso es exitoso; si no, debe fallar.

```
stage('Test') {
    steps {
        script {
            echo "Verificando existencia de README..."
            if (fileExists('README')) {
                echo "README encontrado. Test OK."
            } else {
                error("ERROR: No se encontró el archivo README. Falla la etapa Test.")
            }
        }
    }
```

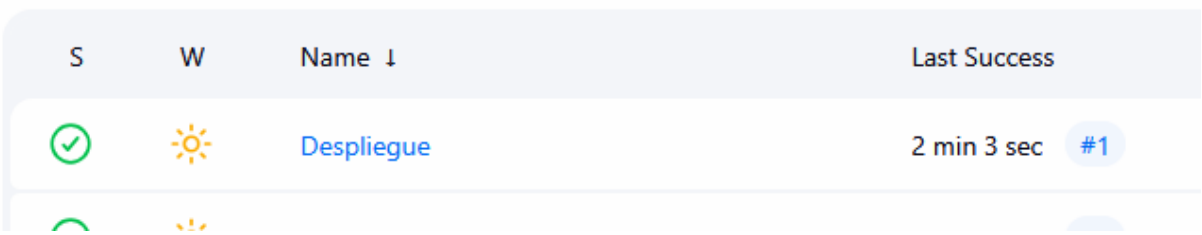
- **Stage 'Deploy':** Imprima "Desplegando a producción...".

```
    stage('Deploy') {
        steps {
            echo "Desplegando a producción..."
        }
    }
}
```

El script lo he puesto en: **Script Pipeline Punto 3.txt**



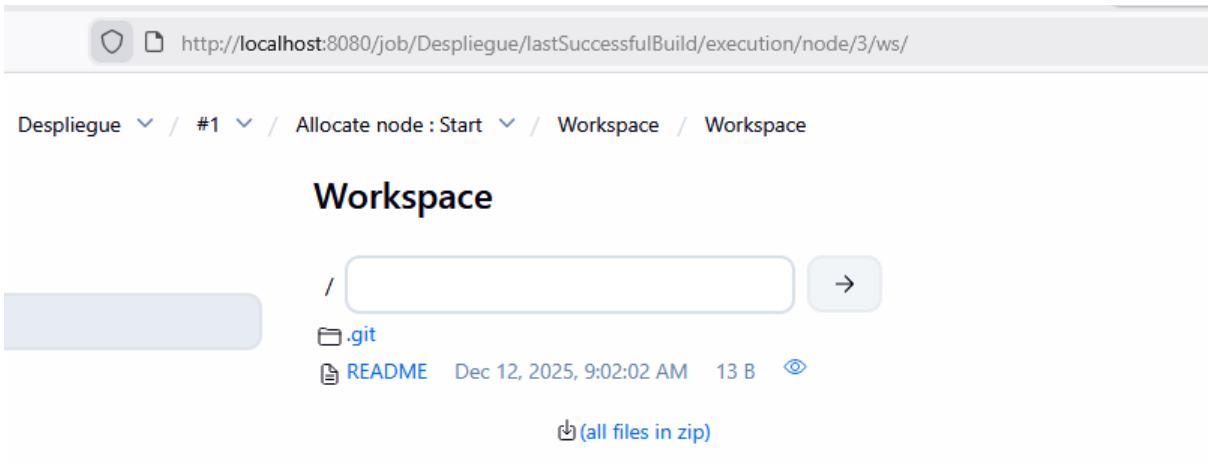
Ha sido exitoso:



```
[Pipeline] echo
Verificando existencia de README...
[Pipeline] fileExists
[Pipeline] echo
README encontrado. Test OK.
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Desplegando a producción...
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

El output completo lo he puesto en: **Output - Script Pipeline Punto 3.txt**

Si vamos al Workspace de **Despliegue**, podemos ver como se ha descargado el repositorio (directorio ".git") con todos sus archivos (archivo "README").

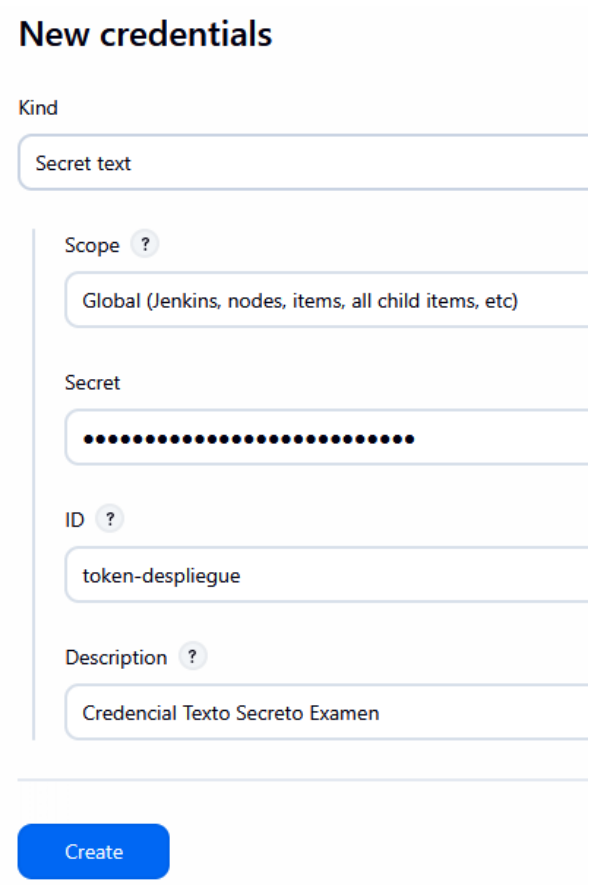


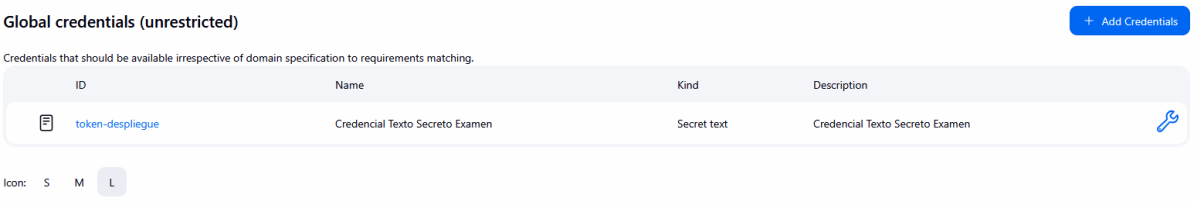
Parte 4: Pipelines y Git (4 puntos)

1. Creación de Credenciales

1. Usa el almacén de credenciales de Jenkins para crear un texto secreto `SUPER_CLAVE_PRODUCCION_2025` y el id `token-despliegue`

Vamos a: **Administrar Jenkins → Credentials → System → Global Credentials**



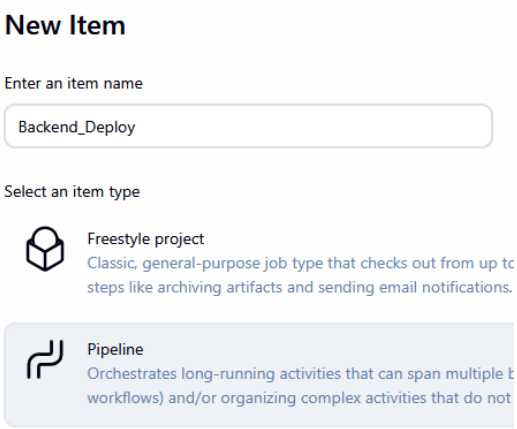


2. Crear el Pipeline "Hijo" (Backend_Deploy)

Este será el pipeline que es *llamado* por el principal. Simulará el despliegue final.

1. Cree una **Nueva Tarea** tipo Pipeline llamada `Backend_Deploy`.

El script de este Pipeline está en: **Backend_Deploy.txt**



2. En la configuración del Pipeline, inserte estos dos parámetros que será VERSION y AUTOR, ambos strings

```
parameters {
    string(name: 'VERSION', description: 'Versión de la aplicación')
    string(name: 'AUTOR', description: 'Persona que solicita el despliegue')
}

stages {
```

3. Tendrá los siguientes stages:
    a. **Stage Verificación del entorno**: mostrará los mensajes siguientes y esperará 2 segundos:
        - Iniciando despliegue de la versión [VERSIÓN]
        - Solicitado por [AUTOR]

```
stages {

    stage('Verificación del entorno') {
        steps {
            echo "Iniciando despliegue de la versión ${params.VERSION}"
            echo "Solicitado por ${params.AUTOR}"

            // Esperar 2 segundos
            sleep(time: 2, unit: 'SECONDS')
        }
    }
```

b. **Stage Despliegue Exitoso:** mostrará el mensaje:
   - La aplicación ha sido desplegada correctamente en el puerto 8080.

```
stage('Despliegue Exitoso') {
    steps {
        echo "La aplicación ha sido desplegada correctamente en el puerto 8080."
    }
}
```

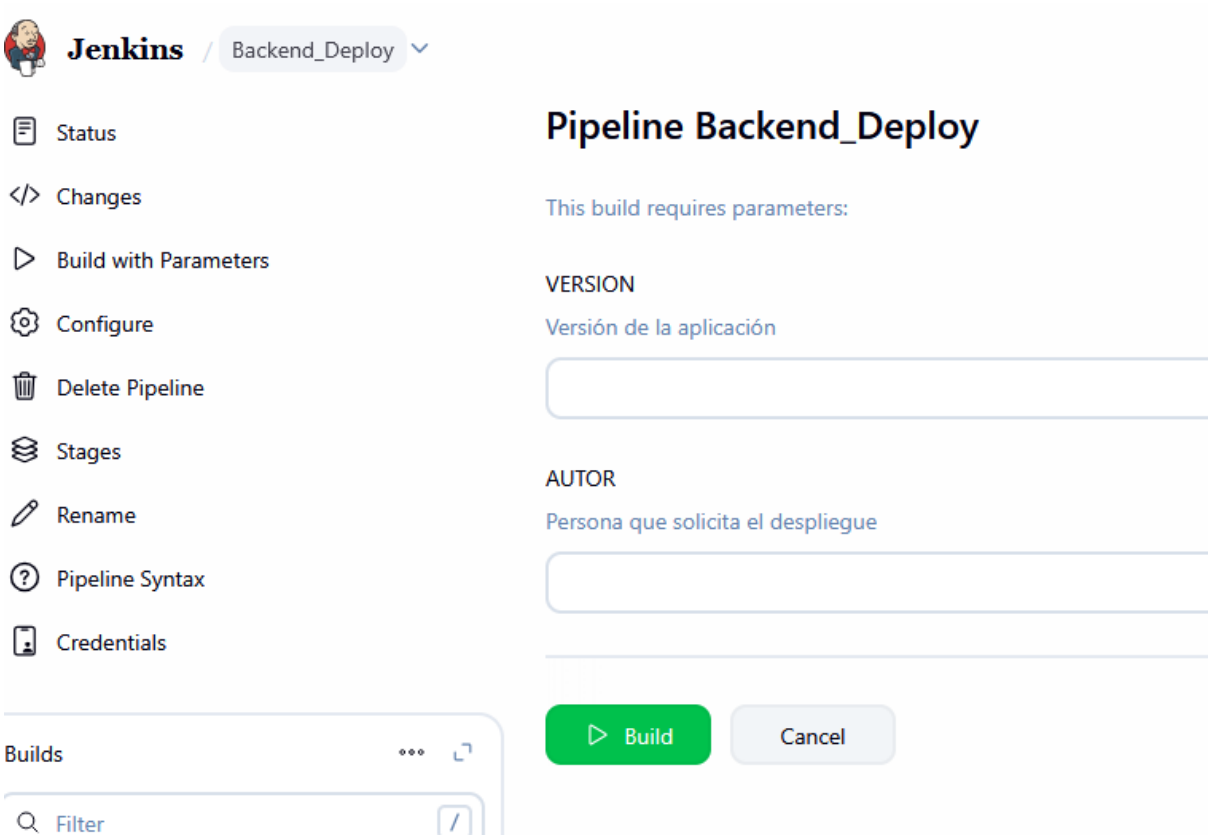
Comprobación:

Console Output

```
Started by user Administrador ASIR Examen
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Backend_Deploy
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Verificación del entorno)
[Pipeline] echo
Iniciando despliegue de la versión
[Pipeline] echo
Solicitado por
[Pipeline] sleep
Sleeping for 2 sec
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Despliegue Exitoso)
[Pipeline] echo
La aplicación ha sido desplegada correctamente en el puerto 8080.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

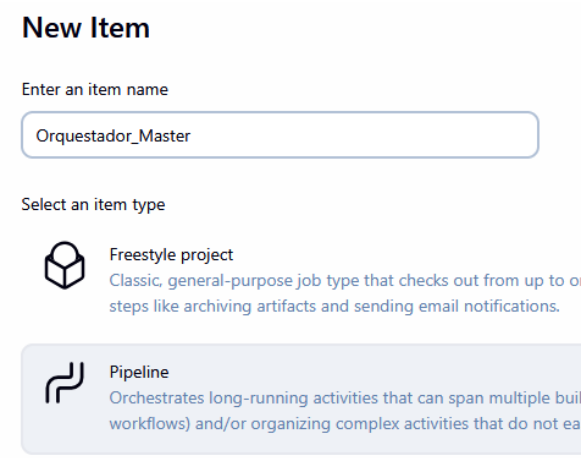
El output está en: **Output - Backend_Deploy.txt**

Ahora, cuando vayamos construir de nuevo el Pipeline, nos saldrá que se construye con parámetros; y veremos la opción de usar los parámetros que hemos puesto antes. Ahora mismo cuando vayamos a ejecutarlo, podemos dejarlos vacíos, porque en el script no hay nada que haga uso de ellos y por lo tanto no dará error.



3. Crear el Pipeline "Padre" (Orquestador_Master)

1. Cree una **Nueva Tarea** tipo Pipeline llamada `Orquestador_Master`.



2. Tendrá los siguientes stages:
    a. Stage 'Build & Test': Mostrará dos mensajes "Compilando código fuente..." y "Tests pasados correctamente."

```
stages {

    stage('Build & Test') {
        steps {
            echo "Compilando código fuente..."
            echo "Tests pasados correctamente."
        }
    }
```

    b. Stage 'Aprobación de Pase a Producción': Muestra un mensaje '¿Deseas desplegar a Producción?" que debe ser aceptado para continuar el proceso. Si no contesta en 1 minuto el proceso debe ser abortado

```
stage('Aprobación de Pase a Producción') {
    steps {
        timeout(time: 1, unit: 'MINUTES') {
            input message: "¿Deseas desplegar a Producción?"
        }
    }
}
```

c. Stage ‘Llamada a Despliegue’:
   - Muestra mensaje "Autorizando con token seguro:[API_TOKEN]"
   - Llamamos al job hijo pasándole los parámetros que deseemos

“**build job**”: llama al otro pipeline para que se ejecute. En nuestro caso “Backend_Deploy”.
“**wait: true**”: hace que se espere a la ejecución del otro script y que este termine antes de continuar.

```
stage('Llamada a Despliegue') {
    steps {
        echo "Autorizando con token seguro: ${API_TOKEN}"

        // Llamada al JOB HIJO Backend_Deploy
        build job: 'Backend_Deploy',
              wait: true,
              parameters: [
                  string(name: 'VERSION', value: '1.0.0'),
                  string(name: 'AUTOR', value: 'Administrador')
              ]
    }
```

3. Si el pipeline finaliza correctamente mostrará "El orquestador finalizó con éxito. El job hijo debería haberse ejecutado."
4. Si el pipeline no finaliza correctamente mostrará "El orquestador finalizó con éxito. El job hijo debería haberse ejecutado."

```
    post {
        success {
            echo "El orquestador finalizó con éxito. El job hijo debería haberse ejecutado."
        }
        failure {
            echo "El orquestador falló. El job hijo no se ejecutó o falló en su ejecución."
        }
        aborted {
            echo "El orquestador fue abortado. No se completó el proceso."
        }
    }
```

El script del pipeline se encuentra en: **Orquestador_Master.txt**

Ha finalizado con éxito.

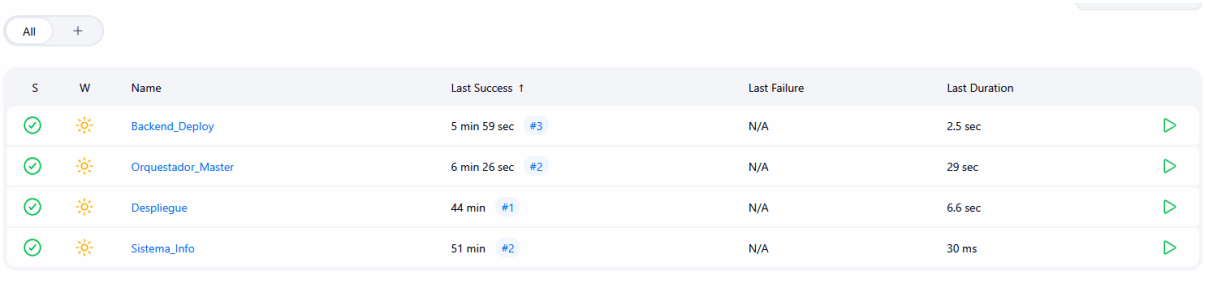
```
¿Deseas desplegar a Producción?
Proceed or Abort
Approved by Administrador ASIR Examen
[Pipeline] }
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Llamada a Despliegue)
[Pipeline] echo
Autorizando con token seguro: 12345-SECURE-TOKEN
[Pipeline] build (Building Backend_Deploy)
Scheduling project: Backend_Deploy
Starting building: Backend_Deploy #3
Build Backend_Deploy #3 completed: SUCCESS
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
El orquestador finalizó con éxito. El job hijo debería haberse ejecutado.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

¿Qué ocurre si falla el Pipeline Hijo? (no lo encuentra o cualquier otro motivo).
"El orquestador falló. El job hijo no se ejecutó o falló en su ejecución."

```
¿Deseas desplegar a Producción?
Proceed or Abort
Approved by Administrador ASIR Examen
[Pipeline] }
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage (hide)
[Pipeline] { (Llamada a Despliegue)
[Pipeline] echo
Autorizando con token seguro: 12345-SECURE-TOKEN
[Pipeline] build
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
El orquestador falló. El job hijo no se ejecutó o falló en su ejecución.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: No item named Backend_Deployjasjasjjas found
Finished: FAILURE
```

El output completo está en: **Output Proceed - Orquestador_Master.txt**
También: **Output Fail - Orquestador_Master.txt**
Para una versión en la que no se contesta a la pregunta, usamos:
Output Abort - Orquestador_Master.txt

En el Dashboard principal podemos ver como tras la ejecución de **Orquestador_Master**, también se ejecuta **Backend_Deploy**.



S	W	Name	Last Success ↑	Last Failure	Last Duration
✓	☼	Backend_Deploy	5 min 59 sec #3	N/A	2.5 sec
✓	☼	Orquestador_Master	6 min 26 sec #2	N/A	29 sec
✓	☼	Despliegue	44 min #1	N/A	6.6 sec
✓	☼	Sistema_Info	51 min #2	N/A	30 ms

Al principio del "**Console Output**" de "**Backend_Deploy**", podemos ver como se inicia su ejecución desde "**Orquestador_Master**".

Console Output

```
Started by upstream project "Orquestador_Master" build number 2
originally caused by:
 Started by user Administrador ASIR Examen
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Backend_Deploy
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Verificación del entorno)
[Pipeline] echo
Iniciando despliegue de la versión 1.0.0
[Pipeline] echo
Solicitado por Administrador
[Pipeline] sleep
Sleeping for 2 sec
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Despliegue Exitoso)
[Pipeline] echo
La aplicación ha sido desplegada correctamente en el puerto 8080.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```