



**DevSecOps**

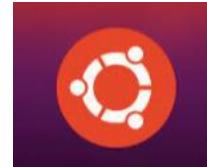
**Antes de iniciar vamos a romper el hielo...**

¿Qué conocen de Jenkins, CI/CD, DevOps? ¿Han trabajado con alguna herramienta?



## Introducción

- Docker es un proyecto de código abierto que permite automatizar el despliegue de aplicaciones dentro de contenedores.
- Fue iniciado por Salomón Hykes dentro de una empresa llamada dotCloud, junto con sus colaboradores
- Fue liberado como código abierto en marzo de 2013
- No es el único entorno, pero si es el mas utilizado

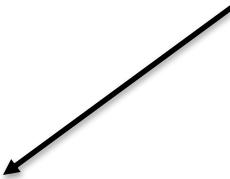


## Introducción

- Son un símil de la problemática en la gestión de mercancías en años pasados, cada integrante de la cadena de transporte manejaba sus propios contenedores o embalajes, lo que resultaba en un severo problema de logística.
- Se decide adoptar un formato estándar para evitar estos problemas, todos los participantes en la cadena, camiones barcos, puertos adoptan el estándar.
- Obteniendo beneficios de reducción de costos y energía, entre otros.



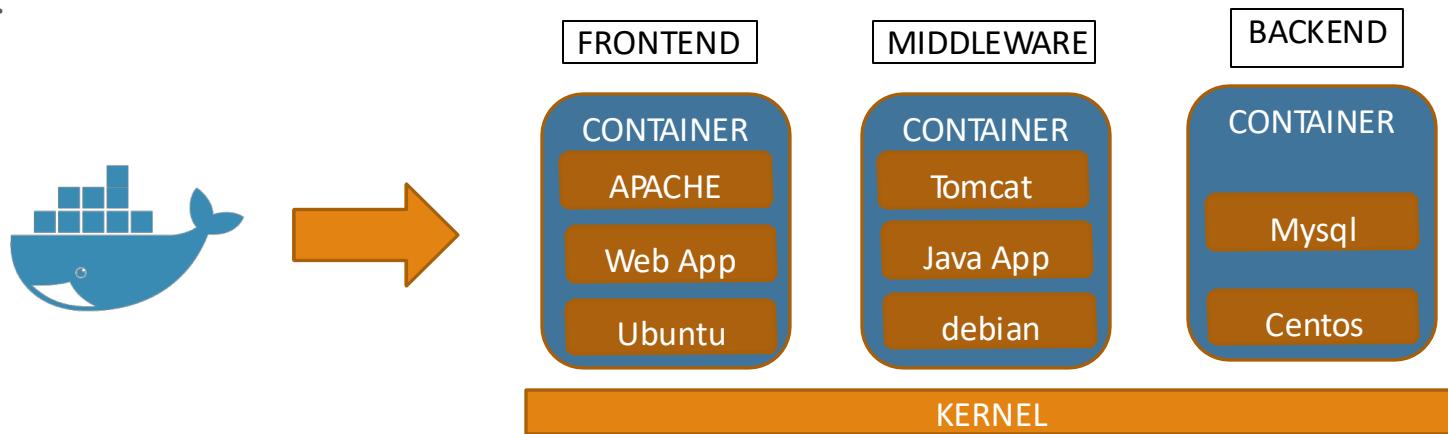
# Introducción



## ¿Qué es un contenedor?

Un contenedor empaqueta de forma ligera todo lo necesario para que uno o mas procesos funcionen: código, herramientas y bibliotecas del sistema, dependencias, etc.

Esto garantiza que siempre se podrá ejecutar, independientemente del entorno en el que se despliegue.



## ¿Qué es un contenedor?

Por lo tanto, Docker esta orientado a solucionar un problema similar, pero en el mundo de las tecnologías de información y comunicación, poder usar un contenedor que pueda ser usado en cualquier plataforma sin tener que cambiar nada.



## ¿Qué es un contenedor?

**Google:** Son paquetes de software que incluyen todos los elementos necesarios para que ejecutes tus productos en cualquier entorno.

Son paquetes ligeros que incluyen el código de las aplicaciones junto con sus dependencias, como versiones concretas de entornos de ejecución de ciertos lenguajes de programación



## ¿Qué es un contenedor?

**Microsoft:** Un paquete de software estándar agrupa el código de una aplicación con las bibliotecas y los archivos de configuración asociados, junto con las dependencias necesarias para que la aplicación se ejecute. Los contenedores proporcionan una infraestructura ligera e immutable para el empaquetado y la implementación de aplicaciones, dado que, sus dependencias y su configuración se empaquetan como una imagen de contenedor.



## ¿Qué es un contenedor?

**IBM:** Son unidades ejecutables de software que empaquetan el código de la aplicación junto con sus bibliotecas y dependencias. Permiten que el código se ejecute en cualquier entorno informático.



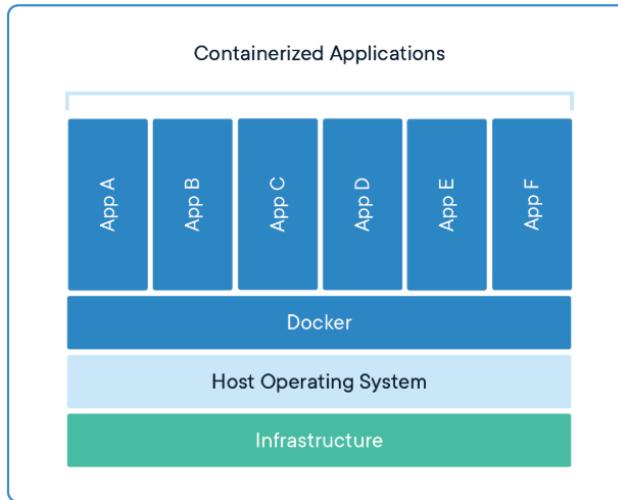
## ¿Qué es un contenedor?

**Docker:** Es una unidad estándar de software que empaqueta el código y todas sus dependencias, de forma que la aplicación se ejecute rápida y confiablemente de un ambiente computacional a otro.

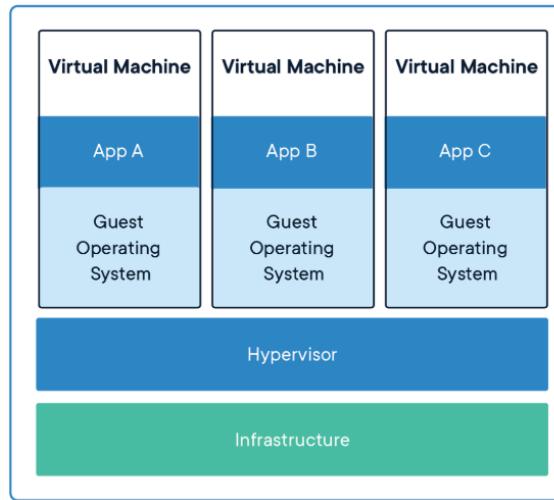
Una imagen de contenedor es paquete de software ejecutable ligero e independiente que incluye todo lo necesario para ejecutar una aplicación: Código, runtime (entorno de ejecución), herramientas del sistema, librerías del sistema y configuraciones.



## Contenedor vs máquina virtual



Un contenedor virtualiza el sistema operativo subyacente.



Las máquinas virtuales virtualizan el hardware subyacente.

---

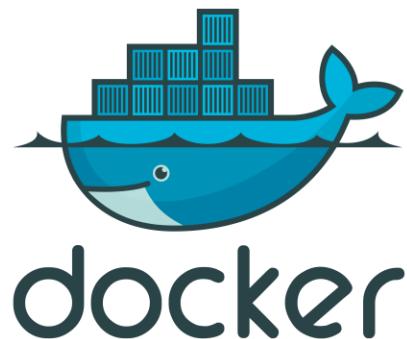
## Ventajas de los contenedores

### Ventajas:

- Agilidad
- Portabilidad
- Escalabilidad rápida
- Utilización eficiente
- Aislamiento de apps
- Separación de responsabilidades

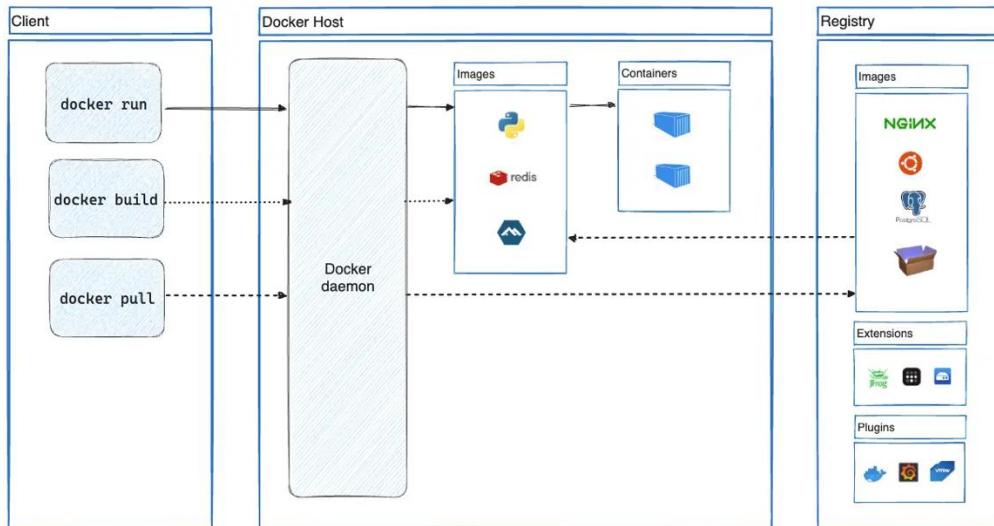
## Docker

- Es una plataforma abierta para desarrollar, desplegar y ejecutar aplicaciones.
- Permite separar las aplicaciones de la infraestructura para poder entregar software de manera rápida.
- Asimismo, brinda la posibilidad de empaquetar y ejecutar aplicaciones en un entorno aislado, llamado contenedor, dado el aislamiento y la seguridad, permite ejecutar varios contenedores simultáneamente en un host determinado.
- Son livianos y contienen todo lo necesario para ejecutar la aplicación, por lo tanto, no depende de lo que está instalado en el host.



# Docker

Usa una arquitectura cliente-servidor donde, el cliente de docker habla con docker daemon, quien se encarga de tareas mas "pesadas" como construir, ejecutar y distribuir contenedores. Estos se conectan a través de REST API.



**Daemon:** recibe las peticiones API de docker y gestiona objetos como imágenes, contenedores, redes y volúmenes.

**Cliente:** es la forma primaria que muchos usuarios interactúan con docker. Cuando ejecutas un comando, el cliente lo envía al daemon.

**Registry:** Almacena imágenes de docker.

---

## Objetos de Docker

**Imagen:** Es una plantilla de solo lectura con instrucciones para crear un contenedor de docker. Se puede crear una imagen propia con un dockerfile con una sintaxis simple para definir los pasos necesarios para crear la imagen y ejecutarla.

**Contenedor:** Es una instancia ejecutable de la imagen. Es definido por su imagen así como cualquier opción de configuración dada cuando se crea. Cuando es eliminado, cualquier cambio en su estado que no fue guardado en almacenamiento persistente, desaparece.

# Instalacion de Docker

Para instalar docker debemos hacer lo siguiente:

- Instalar docker engine, para ello vamos a la siguiente página  
<https://docs.docker.com/engine/install/>
  - Seleccionamos el sistema operativo y seguimos las instrucciones.
- ✓ Configurar el repositorio apt de docker  
✓ Instalar los paquetes de docker  
✓ Verificar la instalación ha sido exitosa.

```
parallels@ubuntu-linux-2404: ~ # Add Docker's official GPG key:  
sudo apt-get update  
sudo apt-get install ca-certificates curl  
sudo install -n 0755 -d /etc/apt/keyrings  
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyring  
s/docker.asc  
sudo chmod a+r /etc/apt/keyrings/docker.asc  
  
# Add the repository to Apt sources:  
echo '  
deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]  
https://download.docker.com/linux/ubuntu /  
\$ . /etc/os-release && echo "SVERSION_CODENAME" stable" | \\\n sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
sudo apt-get update  
[sudo] password for parallels:
```

```
parallels@ubuntu-linux-2404: ~ $ sudo apt-get install docker-ce docker-ce-cli cont  
ainerd.io docker-buildx-plugin docker-compose-plugin  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done
```

```
parallels@ubuntu-linux-2404: ~ sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
478afc919002: Pull complete  
Digest: sha256:305243c734571da2d100c8c8b3c3167a098cab6049c9a5b066b6021a60fc966  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
(arm64v8)  
3. The Docker daemon created a new container from that image which runs the  
executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/
```

---

## Objetos de Docker

**Imagen:** Es una plantilla de solo lectura con instrucciones para crear un contenedor de docker. Se puede crear una imagen propia con un dockerfile con una sintaxis simple para definir los pasos necesarios para crear la imagen y ejecutarla.

**Contenedor:** Es una instancia ejecutable de la imagen. Es definido por su imagen así como cualquier opción de configuración dada cuando se crea. Cuando es eliminado, cualquier cambio en su estado que no fue guardado en almacenamiento persistente, desaparece.

---

## Comandos esenciales de Docker

- **Docker run:** Creará y ejecutará un nuevo contenedor de una imagen
- **Docker ps:** Enlistará los contenedores...
- **Docker pull:** descargará una imagen de un registro
- **Docker push:** cargará una imagen a un registro
- **Docker images:** enlistará las imágenes
- **Docker restart:** Reinicia uno o mas contenedores
- **Docker rm:** elimina uno o mas contenedores.
- **Docker start:** Inicia uno o mas contenedores detenidos
- **Docker stop:** Detiene uno o mas contenedores que se ejecutan.
- Ejecutemos Docker únicamente...

---

## Comandos esenciales de Docker

- Es momento de que interactuemos con algunos de los comandos.

**Docker images**

**Docker run -name nginx -d -p 8080:80**

**Docker ps**

**Docker ps -a**

**Docker stop id contenedor**

**Docker start id contenedor**

**Docker rename id contenedor nombre nuevo**

---

## Comandos esenciales de Docker

- Es momento de que interactuemos con algunos de los comandos.

**Docker images**

**Docker run -name nginx -d -p 8080:80**

**Docker ps**

**Docker ps -a**

**Docker stop id contenedor**

**Docker start id contenedor**

**Docker rename id contenedor nombre nuevo**

---

## Actividad 1

Crear un contenedor en Docker a partir de una imagen pre existente en Docker Hub, que ejecute una página web sencilla, a través de esta actividad, aprenderás como usar imágenes disponibles de Docker para crear y ejecutar de manera rápida contenedores, sin necesidad de crear un Docker file desde cero.

Una vez creado, detengan el contenedor, vuelvan a iniciarlos, renómbrenlos, eliminen el contenedor y la imagen de la que se creó.



¡manos a la obra!

## Actividad 1

- Buscar una imagen como apache o nginx (servicio web)
- Obtener la imagen con Docker pull nombreImagen (Docker pull nginx)
- Una vez descargada la imagen, ejecutar el contenedor con Docker run (docker run -d -p 8080:80 nginx) 80= puerto del contenedor 8080: maquina local
- Comprobamos el estado de los contenedores con Docker ps
- Ejecutamos localhost:xxxx para ver si el servicio está ejecutándose
- Ejecutamos Docker stop id
- Ejecutamos Docker start id
- Docker rename id nuevonombre
- Docker rm id
- Docker rmi nombreImagen

---

## Creación y administración de imágenes.

- Para la crear una imagen se utiliza un archivo llamado Docker file, que contiene un conjunto de instrucciones que le indican a Docker como construir la imagen, paso a paso.
- Especifican como configurar el entorno del contenedor, que sw debe instalarse, como se debe configurar el contenedor, el software necesario y archivos que debe tener el contenedor.
- Para poder crear una imagen se debe usar el comando Docker build sobre dockerfile.

## Creación y administración de imágenes.

A continuación, se enlistan las instrucciones mas comunes:

- **FROM:** define la imagen base a partir de la cual se construirá la nueva imagen. Es la primera línea de un Dockerfile. **Ejemplo:** FROM Ubuntu:20.04
- **RUN:** Permite ejecutar comandos dentro del contenedor durante la construcción de la imagen. **Ejemplo:** RUN apt-get update && apt-get install -y python3 (actualiza el sistema de archivos de UBUNTU e instala Python 3)
- **COPY o ADD:** Se usan para copiar archivos o directorios desde el sistema de archivos local al contenedor. **Ejemplo:** COPY ./app /usr/src/app (copia el contenido de ./app que lo toma de la maquina local y lo copia a la ruta dentro del contenedor)
- **WORKDIR:** Cambia el directorio de trabajo en el contenedor. **Ejemplo:** WORKDIR /usr/src/app (establece este directorio como el del trabajo en el contenedor).

---

## Creación y administración de imágenes.

A continuación, se enlistan las instrucciones mas comunes:

- **EXPOSE:** Sirve para indicar a Docker que los puertos que usara el contenedor. **Ejemplo:** EXPOSE 8080. (indica que el contenedor expondrá el puerto 8080).
- **CMD y ENTRYPOINT:** Definen que comando o programa se debe ejecutar cuando el contenedor de inicie. **Ejemplo:** CMD ["python3", "app.py"] (ejecutara pyhton3 app.py cuando se inicie el contenedor)

---

## Creación y administración de imágenes.

Ejemplo de una aplicación simple en python

```
# Usa una imagen base de Python 3
FROM python:3.9-slim

# Define el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copia el archivo requirements.txt al contenedor
COPY requirements.txt .

# Instala las dependencias de la aplicación
RUN pip install -r requirements.txt

# Copia el resto de los archivos de la aplicación al contenedor
COPY ..

# Expone el puerto que usará la aplicación
EXPOSE 5000

# Comando para ejecutar la aplicación cuando el contenedor se inicie
CMD ["python", "app.py"]
```

## Creación y administración de imágenes.

- Vamos a construir una nueva imagen basada en una existente, una imagen de Ubuntu y NGINIX.
- Para ello en un IDE como VSC creamos una carpeta con el nombre del proyecto.
- Dentro de ella generamos un documento que se llama dockerfile y en el agregamos lo siguiente:

```
# Usamos una imagen base de Ubuntu
FROM ubuntu:20.04

# Actualizamos el sistema y instalamos nginx
RUN apt-get update && apt-get install -y nginx

# Exponemos el puerto 80, que es el puerto por defecto de nginx
EXPOSE 80

# Comando para iniciar nginx cuando el contenedor se ejecute
CMD ["nginx", "-g", "daemon off;"]
```

## Creación y administración de imágenes.

- Desde la línea de comandos, nos posicionamos en la ruta donde se encuentra Dockerfile y ejecutamos lo siguiente: **Docker build -t mi-imagen-nginx**.

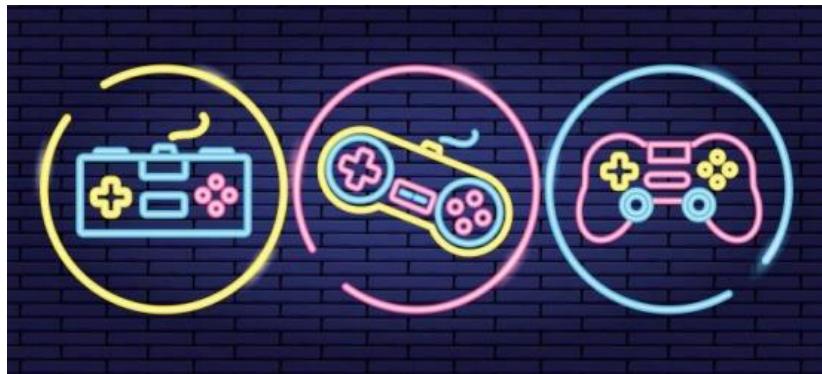
```
(base) luiscarmona@MacBook-Pro-de-Luis-2 Imagen_ubuntu % docker build -t mi-imagen-nginx
[+] Building 7.2s (5/6)
      docker:desktop-linux
=> [internal] load build definition from dockerfile
```

- Donde:
  - ❑ -t mi-imagen-nginx le da un nombre a la imagen (en este caso, mi-imagen-nginx).
  - ❑ El punto (.) indica que Docker debe buscar el Dockerfile en el directorio actual.

---

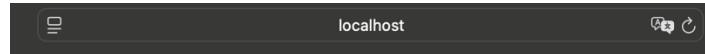
**Antes de iniciar vamos a romper el hielo...**

¿cuál es su libro y/o pelicula favorita y por qué?



## Creación y administración de imágenes.

- Podemos verificar que la imagen esta creada con el comando, **Docker images**.
- Ahora, una vez creada la imagen es necesario crear el contenedor a partir de esta imagen, para ello ejecutamos en el CLI lo siguiente: **docker run -d -p 8080:80 mi-imagen-nginx**
- d: Ejecuta el contenedor en segundo plano.
- p 8080:80: Mapea el **puerto 80** del contenedor al puerto 8080 de tu máquina local.
- mi-imagen-nginx: Especifica la imagen de Docker que quieres usar.



### Welcome to nginx!

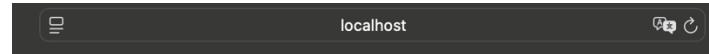
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

## Creación y administración de imágenes.

- Podemos verificar que la imagen esta creada con el comando, **Docker images**.
- Ahora, una vez creada la imagen es necesario crear el contenedor a partir de esta imagen, para ello ejecutamos en el CLI lo siguiente: **docker run -d -p 8080:80 mi-imagen-nginx**
- d: Ejecuta el contenedor en segundo plano.
- p 8080:80: Mapea el **puerto 80** del contenedor al puerto 8080 de tu máquina local.
- mi-imagen-nginx: Especifica la imagen de Docker que quieres usar.



### Welcome to nginx!

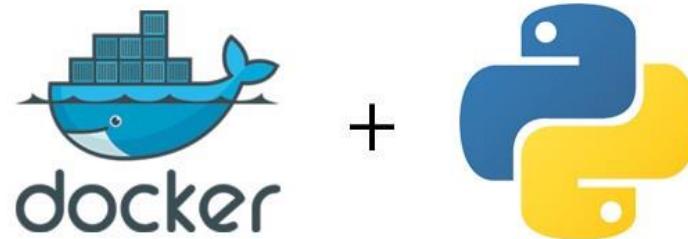
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

## Ejercicio 2

- Crear un archivo dockerfile para contenerizar una aplicación en Python que despliegue hola mundo desde Docker, después crear una imagen y de ésta genera un contenedor. A través de esta actividad, empaquetarás en un contenedor una aplicación, generado desde una imagen creada a partir de un Dockerfile.



## Ejercicio 2

- Generamos una carpeta para el proyecto. (mkdir hola-mundo-Python cd hola-mundo-Python)
- Creamos un archivo app.py para guardar el programa. print("hola mundo desde Docker!")
- Creamos un Dockerfile en VSC, por ejemplo.

```
# Usar una imagen base de Python
FROM python:3.9-slim
# Copiar el archivo Python a la imagen
COPY holapython.py .
# Comando para ejecutar el programa
CMD ["python3", "holapython.py"]
```

- Construimos la imagen, con el comando Docker build -t holapython .
- Creamos el contenedor Docker run holapython

---

## Contenedores interactivos

- Los contenedores interactivos son aquellos que están configurados para permitir la interacción directa con el usuario, a través de la terminal o Shell, lo que permite ejecutar comandos en tiempo real desde el contenedor:
- Tienen acceso a la línea de comandos dado que tienen acceso a un Shell como /bin/bash o /bin/sh
- Entrada estándar abierta (STDIN), lo que permite enviar comandos y por ende, recibir respuestas de forma interactiva
- Se ejecuta con una “terminal” asignada, esto a la opción -t, lo que permite una experiencia similar a trabajar en una consola local.

## Contenedores interactivos

- EL comando para ejecutar un contenedor en modo interactivo es: **docker run -it** donde:  
-i (interactive): Mantiene la entrada estándar (STDIN) abierta.  
-t (terminal): asigna una psedo-terminal a la salida.

**Ejemplo:** docker run -it ubuntu /bin/bash      Donde:

docker run → inicia el contenedor

-it → activa el modo interactivo

/bin/bash abre el Shell bash dentro del contenedor

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker run -it --name contenedor2 ubuntu /bin/bash
root@6958ae0778cb:/# █
```

## Contenedores interactivos

Si ejecutamos ls, ¿Qué pasa?

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker run -it --name contenedor2 ubuntu /bin/bash
root@6958ae0778cb:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

O podríamos obtener información adicional sobre el SO que estamos ejecutando en el contenedor.

```
root@6958ae0778cb:/# cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
root@6958ae0778cb:/# □
```

## Contenedores interactivos

Para salir del contenedor escribimos **exit**.

Es importante mirar algunos detalles, si enlistamos los contenedores vemos lo siguiente:

```
root@6958ae0778cb:/# exit
exit
parallels@ubuntu-linux-22-04-02-desktop:~$ docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED    STATUS     PORTS      NAMES
parallels@ubuntu-linux-22-04-02-desktop:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND
6958ae0778cb   ubuntu      "/bin/bash"

```

→ ¿Porque no se ve el contenedor?

Pero, podemos iniciarla con Docker start xxxx

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker start 6958
6958
parallels@ubuntu-linux-22-04-02-desktop:~$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED        STATUS      PORTS      NAMES
6958ae0778cb   ubuntu      "/bin/bash"   16 minutes ago   Up 5 minutes

```

## Contenedores interactivos

¿Que debo hacer si quiero ingresar de nuevo al contenedor de forma interactiva?

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker run -it --name contenedor2 ubuntu /bin/bash
docker: Error response from daemon: Conflict. The container name "/contenedor2" is already in use by container "6958ae0778cb6e79e129d7ccac057ca55
37511062a47938b67b3a332e02a6e11". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
```

Corremos el contenedor, los enlistamos y con **docker exec -it id o <nombre del c> <acción>**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS        PORTS     NAMES
6958ae0778cb   ubuntu      "/bin/bash"   35 minutes ago   Up 25 minutes
parallels@ubuntu-linux-22-04-02-desktop:~$ docker exec -it 6958ae0778cb bash
root@6958ae0778cb:/#
```

## Contenedores en segundo plano

- Un contenedor en segundo plano se ejecuta sin bloquear la terminal o sesión del usuario, y así permitir que el SO siga funcionando mientras el contenedor está en ejecución.
- Cuando se ejecuta de esta forma, se hace en el background sin necesidad de que interactúes directamente con él a través de la terminal.
- Si se requiere ejecutar un contenedor en segundo plano se debe usar la opción **-d** (detached mode). Con el siguiente comando básico: **docker run -d -name <nombre contenedor> <nombre\_imagen>**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker run -d --name Cdetached3 nginx
a6264c4561af1fe42f3b2dcbb7224f34ee6758517b94fa19de5d0704a2adbb8c
parallels@ubuntu-linux-22-04-02-desktop:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS        PORTS     NAMES
a6264c4561af   nginx      "/docker-entrypoint...."   3 seconds ago   Up 3 seconds   80/tcp    Cdetached3
parallels@ubuntu-linux-22-04-02-desktop:~$ █
```

Como podemos ver, la terminal queda libre, pero el contenedor se sigue ejecutando en segundo plano, por ende cualquier proceso que se ejecute dentro o como parte del contenedor

## Contenedores en primer plano

- Para ejecutar un contenedor en primer plano, basta con remover el argumento **-d**. Veamos un ejemplo.

**Docker run --name <nombre contenedor> <imagen>**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker run --name cprimerplano nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/12/03 06:59:15 [notice] 1#1: using the "epoll" event method
2024/12/03 06:59:15 [notice] 1#1: nginx/1.27.3
2024/12/03 06:59:15 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/12/03 06:59:15 [notice] 1#1: OS: Linux 5.15.0-126-generic
2024/12/03 06:59:15 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/12/03 06:59:15 [notice] 1#1: start worker processes
2024/12/03 06:59:15 [notice] 1#1: start worker process 29
2024/12/03 06:59:15 [notice] 1#1: start worker process 30
exit
[2024/12/03 06:59:40] [notice] 1#1: signal 2 (SIGINT) received, exiting
```

## Seguridad en los contenedores.

- Dado que los contenedores pueden desplegarse en cualquier “lugar”, esto que crea nuevas superficies de ataque que rodean el entorno. Por ejemplo:
  - Las imágenes
  - Registros
  - Tiempos de ejecución
  - Plataformas de orquestación
  - Sistemas operativos del host

Por lo tanto, las empresas deberían incluir estas tecnologías en sus políticas y estrategias de seguridad, e implementar marcos o metodologías como “zero trust” o “DevSecOps”.

## Mejores prácticas.

- Hay algunas prácticas que las empresas pueden seguir como las siguientes:
  - Principio del menor privilegio:** Contenedores y usuarios solo pueden realizar el conjunto mínimo de funciones que necesitan para hacer su trabajo. **Ejemplos:** Políticas granulares de IAM, Ejecutar en modo no root, bloquear la capa de red.
  - Usar solo contenedores de confianza:** Reduciendo el uso a solo imágenes de confianza y firmadas, por ejemplo, con Docker Content Trust.
  - Aplicar cuotas de recursos:** Limitando la cantidad de recursos que puede consumir un contenedor.
  - Supervisión proactiva:** Permite detectar rápidamente amenaza, identificar vulnerabilidades y solucionar problemas rápidamente.

## Mejores prácticas.

- Usar imágenes oficiales y verificadas: siempre que sea posible usa imágenes oficiales de Docker hub o repositorios de confianza. Asimismo, evita usar imágenes sin mantenimiento, dado que pueden tener vulnerabilidades.
- Mantener las imágenes actualizadas para evitar vulnerabilidades conocidas.
- Usa contenedores con permisos mínimos, evitando ejecutarlos como root
- Aislamiento de red y puertos, con configuración de redes y puertos.
- Limitar los recursos que puede usar el contenedor.
- Escanear los contenedores en búsqueda de vulnerabilidades
- Auditoria y registros de actividad
- Actualizar el SO del host

---

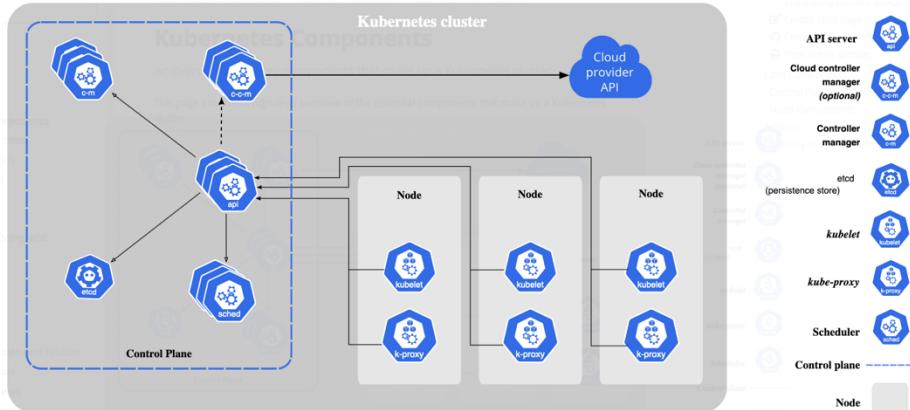
## Introducción a Kubernetes.

- Es una plataforma de orquestación de contenedores de código abierto, que automatiza la implementación, escalado y gestión de aplicaciones en contenedores, fue creado originalmente por Google y ahora es mantenido por Cloud Native Computing Foundation.
- Es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Facilita la automatización y configuración declarativa.
- Ofrece un entorno de administración centrado en contenedores, orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo.

## Introducción a Kubernetes.

- **Automatización del despliegue y gestión:** Kubernetes facilita la automatización de la implementación y gestión de aplicaciones en contenedores, eliminando gran parte del trabajo manual.
- **Escalabilidad:** Kubernetes puede escalar las aplicaciones automáticamente, añadiendo o eliminando instancias de contenedores según la demanda.
- **Alta disponibilidad:** Kubernetes puede distribuir las aplicaciones entre múltiples nodos, lo que proporciona tolerancia a fallos y alta disponibilidad.
- **Gestión del ciclo de vida de las aplicaciones:** Kubernetes permite gestionar el ciclo de vida completo de una aplicación, desde la implementación hasta las actualizaciones y el despliegue continuo.
- **Portabilidad:** Kubernetes es independiente del proveedor de infraestructura (puede ejecutarse en entornos de nube pública, privada o híbrida, o incluso en servidores locales), lo que permite mover aplicaciones entre diferentes entornos con facilidad.

# Arquitectura de Kubernetes.



**Kube-apiserver:** Expone la API HTTP de K8S

**etcd:** almacena de valores clave, consistente y altamente disponible para los datos del servidor API

**Kube-scheduler:** Busca Pods no unidos a un nodo y lo asigna a uno.

**Kube-controller-manager (opcional):** Ejecuta controladores para implementar el comportamiento de la API de K8S

**Kubelet:** Asegura que los Pods están ejecutándose, incluyendo sus contenedores.

**Kube-proxy (opcional):** Mantiene las reglas de red en los nodos para implementar servicios

**Container runtime:** Software responsable para ejecutar contenedores.

Hay componentes adicionales opcionales como:

- **DNS**
- **Web UI**
- **Container resource monitoring**
- **Cluster-level monitoring**

---

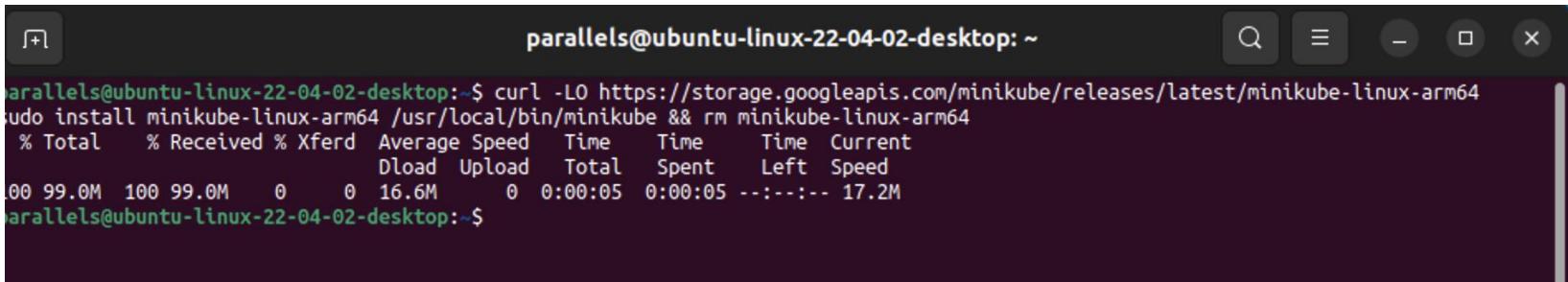
## Introducción a Kubernetes.

- **Pod:** Es la unidad mas pequeña y básica de ejecución que puede desplegar y gestionar el sistema, los contenedores dentro de un mismo Pod Es un grupo de uno o mas contenedores con almacenamiento y red compartidos. Los contenidos de un Pod son siempre coubicados, coprogramados y ejecutados en un contexto compartido.
- **Servicios:** Es el objeto de la API de K8S que describe como se accede a las aplicaciones, tal como un conjunto de Pods, y que puede descubrir puertos y balanceadores de carga. Es un objeto REST, similar a un Pod. Es un objeto que proporciona una interfaz estable y accesible para acceder a un conjunto de Pods que realizan la misma función o servicio.
- **Despliegues:** Es un objeto de alto nivel, que gestiona el ciclo de vida de los contenedores dentro de un clúster, uno de sus objetivos es asegurar que un numero determinado de replicas de un pod estén ejecutándose de manera constante y estable.

## Configuración inicial de un cluster de K8S (Minikube)

- Para poder ejecutar un cluster de K8S de manera local, tenemos una herramienta llamada minikube, que ejecuta un cluster de un nodo, ya sea en una máquina virtual o un contenedor.
- Para instalarlo, podemos buscar las instrucciones en el explorador, en este caso, para Linux, y ejecutamos los siguientes comandos:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-arm64
sudo install minikube-linux-arm64 /usr/local/bin/minikube && rm minikube-linux-arm64
```



A screenshot of a terminal window titled "parallels@ubuntu-linux-22-04-02-desktop: ~". The terminal shows the command being run and its output. The output includes the curl command, the sudo command to install the binary, and a progress bar for the download. The progress bar shows 100% received, 0% total, and a speed of 17.2M. The terminal window has a dark theme with light-colored text and icons.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-arm64
sudo install minikube-linux-arm64 /usr/local/bin/minikube && rm minikube-linux-arm64
  % Total    % Received % Xferd  Average Speed   Time   Time     Current
          Dload  Upload   Total Spent    Left  Speed
  0.00  99.0M  100  99.0M    0     0  16.6M    0  0:00:05  0:00:05  --:--:-- 17.2M
parallels@ubuntu-linux-22-04-02-desktop:~$
```

## Configuración inicial de un clúster de K8S (Minikube)

- Ahora debemos iniciar el clúster, con el comando **minikube start**, como se ve en la siguiente imagen.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ minikube start
└─ minikube v1.34.0 on Ubuntu 22.04 (arm64)
  └─ Automatically selected the docker driver. Other choices: none, ssh

  ┌─ The requested memory allocation of 1966MiB does not leave room for system overhead (total system memory: 1966MiB). You may face stability issues.
  └─ Suggestion: Start minikube with less memory allocated: 'minikube start --memory=1966mb'

  ┌─ Using Docker driver with root privileges
  └─ Starting "minikube" primary control-plane node in "minikube" cluster
    └─ Pulling base image v0.0.45 ...
      └─ Downloading Kubernetes v1.31.0 preload ...
          > preloaded-images-k8s-v18-v1...: 307.61 MiB / 307.61 MiB 100.00% 14.88 M
          > gcr.io/k8s-minikube/kicbase...: 441.45 MiB / 441.45 MiB 100.00% 13.26 M
        └─ Creating docker container (CPUs=2, Memory=1966MB) ...
          └─ Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
              ┌─ Generating certificates and keys ...
              ┌─ Booting up control plane ...
              ┌─ Configuring RBAC rules ...
              └─ Configuring bridge CNI (Container Networking Interface) ...
            └─ Verifying Kubernetes components...
                ┌─ Using image gcr.io/k8s-minikube/storage-provisioner:v5
                └─ Enabled addons: storage-provisioner, default-storageclass
                  kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
                  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
parallels@ubuntu-linux-22-04-02-desktop:~$ 
```

# Configuración de kubectl

- Una de las formas de interactuar con **kubernetes** es a través de **kubectl**, para eso, debemos instalarla, para ello podemos buscar en el explorador como se ve en la siguiente imagen y seguir las instrucciones.

Kubernetes Documentation / Tasks / Install Tools / Install and Set Up kubectl on Linux

## Install and Set Up kubectl on Linux

### Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.31 client can communicate with v1.30, v1.31, and v1.32 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.

### Install kubectl on Linux

The following methods exist for installing kubectl on Linux:

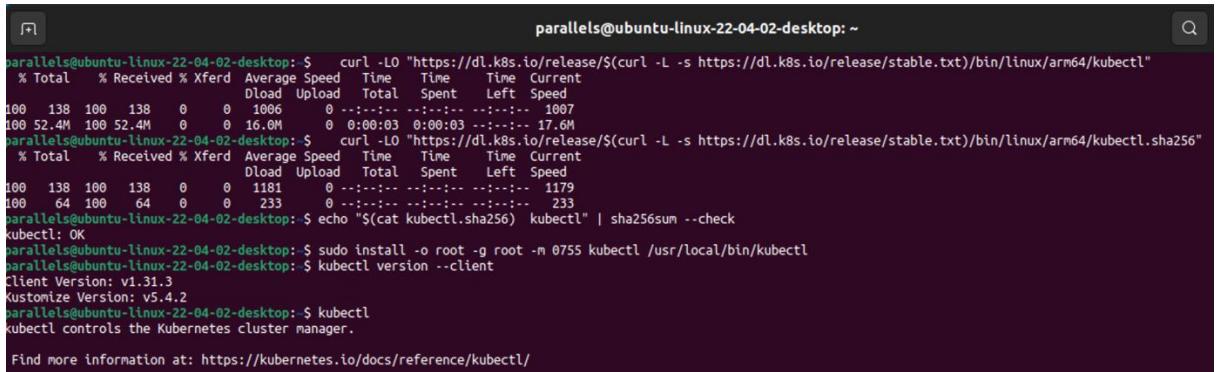
- Install kubectl binary with curl on Linux
- Install using native package management
- Install using other package management

### Install kubectl binary with curl on Linux

- Download the latest release with the command:

```
x86-64 ARM64
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl"
```

- Descargar la ultima liberacion
- Validar el binario (opcional)
- Instalar kubectl
- Probar para asegurar que la version está actualizada



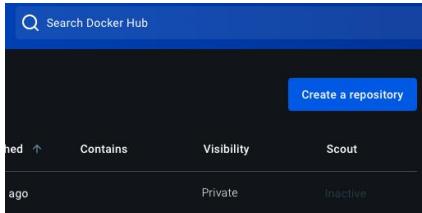
A screenshot of a terminal window titled 'parallels@ubuntu-linux-22-04-02-desktop: ~'. The window shows the process of downloading the kubectl binary from the Kubernetes releases page. It includes the command 'curl -LO "https://dl.k8s.io/release/\$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl"', the resulting file download progress (100% received), and the verification command 'echo "\$(cat kubectl.sha256)" | sha256sum --check' which outputs 'kubectl: OK'. The terminal also shows the path '/usr/local/bin/kubectl' and the output of 'kubectl version --client' which shows 'Client Version: v1.31.3' and 'Kustomize Version: v5.4.2'.

```
parallels@ubuntu-linux-22-04-02-desktop: ~
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl"
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total Spent   Left Speed
100 138 100 138    0     0 1806  0:--:--:--:--:--:--:--: 1007
100 52.4M 100 52.4M   0     0 16.0M  0:00:03 0:00:03  .:--:--: 17.6M
parallels@ubuntu-linux-22-04-02-desktop: ~ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl.sha256"
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total Spent   Left Speed
100 138 100 138    0     0 1181  0:--:--:--:--:--:--:--: 1179
100 64 100 64    0     0 233  0:--:--:--:--:--:--:--: 233
parallels@ubuntu-linux-22-04-02-desktop: ~ echo "$(cat kubectl.sha256)" | sha256sum --check
kubectl: OK
parallels@ubuntu-linux-22-04-02-desktop: ~ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
parallels@ubuntu-linux-22-04-02-desktop: ~ kubectl version --client
Client Version: v1.31.3
Kustomize Version: v5.4.2
parallels@ubuntu-linux-22-04-02-desktop: ~ kubectl
kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/
```

# Publicación de imágenes Docker en repositorios

- Para la publicación de imágenes en Docker hub, es necesario crear una cuenta en Docker hub, o si ya se tiene, iniciar sesión.
- Podemos acceder a través del siguiente link <https://hub.docker.com>
- Dentro de, creamos un repositorio.



Repositories / Create

Create repository

Namespace: luis1021      Repository Name: devsecops

Short description: repositorio para el curso de DevSecOps

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 1 of 1 private repositories. [Get more](#)

Public  Appears in Docker Hub search results

Private  Only visible to you

[Cancel](#) [Create](#)

- Llenamos los campos con la información necesaria

## Publicación de imágenes Docker en repositorios

- Una vez creada la cuenta vamos a validar que esté la imagen, con el comando **Docker image**, en este caso usaremos la primera, que generamos al inicio.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
imagen-ubuntu	latest	4e4048f864c6	About a minute ago	160MB
sotoboter/o/billingapp	latest	d8d8dd605618	8 days ago	290MB
gcr.io/k8s-minikube/kicbase	v0.0.45	8411aacd61cb	2 months ago	1.19GB
ubuntu	latest	07db2ffa22da	7 months ago	98.8MB
datadog/agent	latest	7ac56599cc5f	7 months ago	1.06GB
jenkins/jenkins	latest	3ee72417d7e1	11 months ago	502MB
sotoboter/o/frontend-angular-billingapp	latest	05ab1871ab91	12 months ago	48.6MB
hello-world	latest	ee301c921b8a	19 months ago	9.14kB
centos	latest	e6a0117ec169	3 years ago	272MB
sotoboter/o/udemy-devops	0.0.1	37af101e3eb7	3 years ago	189MB

# Publicación de imágenes Docker en repositorios

- Podemos construir una imagen o usar una nueva, usemos la que creamos previamente.
- Cualquiera que sea el caso, debemos agregar una etiqueta a la imagen, lo hacemos con el comando **docker tag my-app:latest miusuario/my-app:latest**

**docker tag imagen-ubuntu:latest luis1021/ubuntu\_nginx**

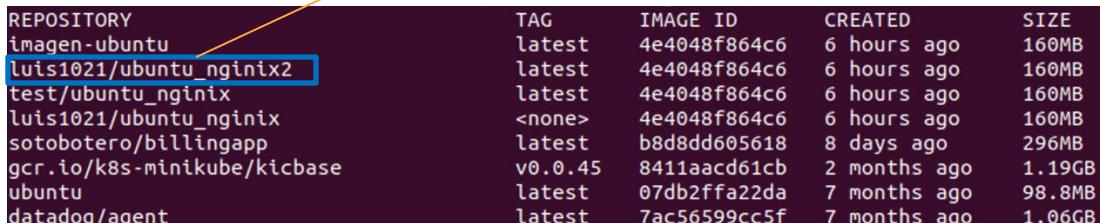
```
parallels@ubuntu-linux-22-04-02-desktop: $ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
imagen-ubuntu      latest   4e4048f864c6  5 hours ago  160MB
luis1021/ubuntu_nginx  latest   4e4048f864c6  5 hours ago  160MB
sotobotero/billingapp    latest   b8d8dd605618  8 days ago   296MB
gcr.io/k8s-minikube/kicbase v0.0.45  8411aacd61cb  2 months ago  1.19GB
ubuntu              latest   07db2ffa22da  7 months ago  98.8MB
datadog/agent        latest   7ac56599cc5f  7 months ago  1.06GB
jenkins/jenkins      latest   3ee72417d7e1  11 months ago 502MB
sotobotero/frontend-angular-billingapp latest   05ab1871ab91  12 months ago 48.6MB
hello-world          latest   ee301c921b8a  19 months ago 9.14kB
centos              latest   e6a0117ec169  3 years ago   272MB
sotobotero/udemy-devops 0.0.1    37af101e3eb7  3 years ago   189MB
parallels@ubuntu-linux-22-04-02-desktop: $
```

# Publicación de imágenes Docker en repositorios

- Ahora nos autenticamos con **docker login**
- Despues, con **docker push nombreusuario/nombreImagen** subimos la imagen añ hub

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/parallels/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

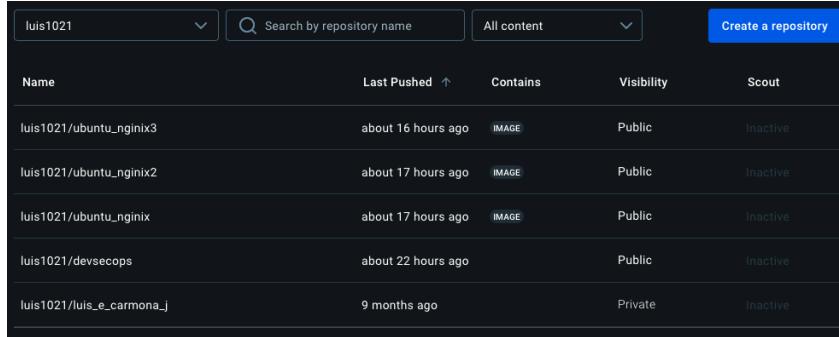
Login Succeeded
parallels@ubuntu-linux-22-04-02-desktop:~$ docker push luis1021/ubuntu_nginxx2
Using default tag: latest
The push refers to repository [docker.io/luis1021/ubuntu_nginxx2]
aed7b178a244: Mounted from luis1021/ubuntu_nginxx
171652ecd561: Mounted from luis1021/ubuntu_nginxx
latest: digest: sha256:a092003419a59eea13ead2730b9196885729c744c9fdf64617be451dc04ed928 size: 741
```



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
imagen-ubuntu	latest	4e4048f864c6	6 hours ago	160MB
luis1021/ubuntu_nginxx2	latest	4e4048f864c6	6 hours ago	160MB
test/ubuntu_nginxx	latest	4e4048f864c6	6 hours ago	160MB
luis1021/ubuntu_nginxx	<none>	4e4048f864c6	6 hours ago	160MB
sotobotero/billingapp	latest	b8d8dd605618	8 days ago	296MB
gcr.io/k8s-minikube/kicbase	v0.0.45	8411aacd61cb	2 months ago	1.19GB
ubuntu	latest	07db2ffa22da	7 months ago	98.8MB
datadog/agent	latest	7ac56599cc5f	7 months ago	1.06GB

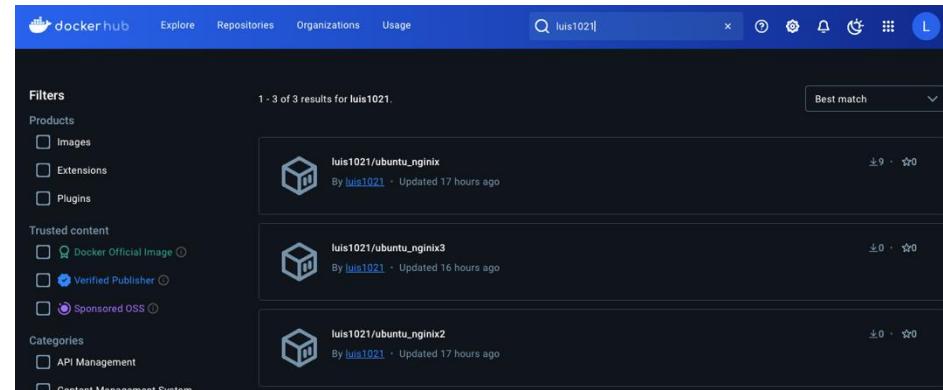
# Publicación de imágenes Docker en repositorios

Si vamos a nuestra cuenta de docker Hub podemos ver que ya está la imagen.



A screenshot of a Docker registry interface. At the top, there is a search bar with the placeholder "Search by repository name" and a dropdown menu set to "All content". A blue button labeled "Create a repository" is visible. Below the header, a table lists five repositories under the heading "Name". The columns include "Name", "Last Pushed", "Contains", "Visibility", and "Scout".

Name	Last Pushed	Contains	Visibility	Scout
luis1021/ubuntu_nginx3	about 16 hours ago	IMAGE	Public	Inactive
luis1021/ubuntu_nginx2	about 17 hours ago	IMAGE	Public	Inactive
luis1021/ubuntu_nginxx	about 17 hours ago	IMAGE	Public	Inactive
luis1021/devsecops	about 22 hours ago		Public	Inactive
luis1021/luis_e_carmona_j	9 months ago		Private	Inactive



A screenshot of the Docker Hub search results for the user "luis1021". The search bar at the top contains "luis1021". On the left, there are filters for "Products" (Images, Extensions, Plugins), "Trusted content" (Docker Official Image, Verified Publisher, Sponsored OSS), and "Categories" (API Management, Content Management System). The main area shows three repository results: "luis1021/ubuntu\_nginxx" (updated 17 hours ago), "luis1021/ubuntu\_nginx3" (updated 16 hours ago), and "luis1021/ubuntu\_nginx2" (updated 17 hours ago). Each result includes a thumbnail, the repository name, the author, and the last update time.

# Vamos Conociendo y teniéndonos confianza

## Pregunta de Descongelamiento.

**¿Cuál tu desayuno favorito?**



## Despliegue de contenedores en K8S

Dado que un Pod es la unidad mínima de K8S, necesitamos crear un Pod, lo hacemos con el siguiente comando:

```
kubectl run nombreDelPod --image=imagenDockerQueElContenedorUsará --port
```

```
kubectl run miprimerpod --image=nginx:alpine
```

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl run 3pod --image=nginx:alpine
pod/3pod created
```

Podemos revisar los pods que se tienen con el comando **kubectl get pods**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl get pods
NAME      READY   STATUS        RESTARTS   AGE
1pod      0/1     ImagePullBackOff  0          2m27s
2pod      0/1     ImagePullBackOff  0          57s
3pod      0/1     ContainerCreating  0          6s
mipod3    1/1     Running       3 (5h16m ago)  5d9h
mipoddos  1/1     Running       3 (5h16m ago)  5d10h
miprimerpod 0/1     ImagePullBackOff  0          5d10h
nginx-pod  1/1     Running       2 (5h16m ago)  14h
```

## Despliegue de contenedores en K8s

- Tambien puedes entrar en modo interactivo con el contenedor del Pod, en este caso con el comando, **kubectl exec -ti mipod3 - sh**, donde entramos en modo interactivo (-ti) a mipod3 (nombre del pod)

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl exec -ti mipod3 -- sh
# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
#
```

Podemos ver que estamos dentro de Ubuntu...

# Despliegue de contenedores en K8S

- Se puede obtener información de los Pods, una es través del comando **Kubectl get pods**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
mipod3      1/1     Running   0          10h
mipoddos    1/1     Running   0          11h
miprimerpod 0/1     ImagePullBackOff 0          11h
parallels@ubuntu-linux-22-04-02-desktop:~$
```

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl describe pod mipod3
Name:         mipod3
Namespace:    default
Priority:    0
Service Account: default
Node:        minikube/192.168.49.2
Start Time:  Thu, 28 Nov 2024 13:48:38 -0600
Labels:      run=mipod3
Annotations: <none>
Status:      Running
IP:          10.244.0.13
IPs:
  IP: 10.244.0.13
Containers:
  mipod3:
    Container ID: docker://b3a3a65c27efa9aa0bdd68b0334bc9a4cbd68b6c35123ff1efc1badc2ca5690
    Image:          luis1021/ubuntu_nginx:2
    Image ID:       docker-pullable://luis1021/ubuntu_nginx@sha256:a092003419a59eea13ead2730b9196885729c744c9fdf64617be451dc04ed928
    Port:          <none>
    Host Port:    <none>
    State:        Running
      Started:   Thu, 28 Nov 2024 13:48:49 -0600
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7zgkf (ro)
Conditions:
  Type        Status
  PodReadyToStartContainers  True
  Initialized  True
  Ready        True
  ContainersReady  True
  PodScheduled  True
Volumes:
  kube-api-access-7zgkf:
    Type:          Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:   kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:    true
    QoS Class:      BestEffort
    Node Selectors: <none>
    Tolerations:   node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                   node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
    Events:        <none>
```

- O, se puede obtener mayor detalle de cada uno de los Pods con el comando **kubectl describe pod nombrePod**

## Despliegue de contenedores en K8S

- Se puede obtener información del contenedor primario con **kubectl logs -f <nombrePod>**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl logs -f 3pod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/12/04 05:06:17 [notice] 1#1: using the "epoll" event method
2024/12/04 05:06:17 [notice] 1#1: nginx/1.27.3
2024/12/04 05:06:17 [notice] 1#1: built by gcc 13.2.1 20240309 (Alpine 13.2.1_git20240309)
2024/12/04 05:06:17 [notice] 1#1: OS: Linux 5.15.0-126-generic
2024/12/04 05:06:17 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/12/04 05:06:17 [notice] 1#1: start worker processes
2024/12/04 05:06:17 [notice] 1#1: start worker process 30
2024/12/04 05:06:17 [notice] 1#1: start worker process 31
```

## Despliegue de contenedores en K8S

- Se pueden borrar 1 o mas pods con **kubectl delete pod <nombrepod> <nombrepod2>**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl delete pod 1pod 2pod miprimerpod
pod "1pod" deleted
pod "2pod" deleted
pod "miprimerpod" deleted
parallels@ubuntu-linux-22-04-02-desktop:~$ █
```

## Despliegue de contenedores en K8S

- Intentemos crear un pod con un contenedor de una imagen que no existe, como se ve en el ejemplo.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl run 4pod --image=nginx:alpinezzz
pod/4pod created
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
3pod      1/1     Running   0          34m
4pod      0/1     ErrImagePull 0          6s
mipod3    1/1     Running   3 (5h51m ago) 5d9h
mipoddos  1/1     Running   3 (5h51m ago) 5d11h
nginx-pod 1/1     Running   2 (5h51m ago) 15h
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
3pod      1/1     Running   0          35m
4pod      0/1     ErrImagePull 0          15s
mipod3    1/1     Running   3 (5h51m ago) 5d9h
mipoddos  1/1     Running   3 (5h51m ago) 5d11h
nginx-pod 1/1     Running   2 (5h51m ago) 15h
parallels@ubuntu-linux-22-04-02-desktop:~$
```

- Podemos ver que el pod si se crea (4pod) pero en la columna status, marca un error.

# Despliegue de contenedores en K8S

- Tenemos el comando

**kubectl describe pod <nombrePod>**

**Kubectl descrbe pod 4pod**

```
parallels@ubuntu-linux-22-04-02-desktop: $ kubectl describe pod 4pod
Name:           4pod
Namespace:      default
Priority:       0
Service Account: default
Node:          minikube/192.168.49.2
Start Time:    Tue, 03 Dec 2024 23:40:58 -0600
Labels:         run=4pod
Annotations:   <none>
Status:        Pending
IP:            10.244.0.38
IPs:
  IP: 10.244.0.38
Containers:
  4pod:
    Container ID: nginx:alpinezzz
    Image ID:      nginx:alpinezzz
    Port:          <none>
    Host Port:    <none>
    State:        Waiting
      Reason:     ImagePullBackOff
    Ready:        False
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-hwxmx (ro)
Conditions:
  Type        Status
  PodReadyToStartContainers  True
  Initialized  True
  Ready        False
  ContainersReady  False
  PodsScheduled  True
Volumes:
  kube-api-access-hwxmx:
    Type:          Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:   kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:    true
    QoS Class:     BestEffort
    Node-Selectors: <none>
    Tolerations:   node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                    node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type  Reason  Age   From            Message
  ----  -----  --   --              -----
  Normal  Scheduled  7m44s  default-scheduler  Successfully assigned default/4pod to minikube
  Normal  Pulling   6m6s  (x4 over 7m44s)  kubelet  Pulling image "nginx:alpinezzz"
  Warning Failed    6m4s  (x4 over 7m41s)  kubelet  Failed to pull image "nginx:alpinezzz": Error response from daemon: manifest for nginx:alpinezzz not found: manifest unknown: manifest unknown
  Warning Failed    6m4s  (x4 over 7m41s)  kubelet  Error: ErrImagePull
  Warning Failed    5m51s (x6 over 7m41s)  kubelet  Error: ImagePullBackOff
  Normal  BackOff   2m35s  (x20 over 7m41s)  kubelet  Back-off pulling image "nginx:alpinezzz"
```

## Despliegue de contenedores en K8S

- Podemos obtener archivos .yaml de un pod existente con el comando **kubectl get pod <nombrePod> -o yaml**

```
parallels@ubuntu-linux-22-04-02-desktop:~$ kubectl get pod 4pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2024-12-04T05:40:58Z"
  labels:
    run: 4pod
  name: 4pod
  namespace: default
  resourceVersion: "64339"
  uid: 6ad8fee4-ac55-4e90-b39d-40eb2f1d13ec
spec:
  containers:
  - image: nginx:alpinezzz
    imagePullPolicy: IfNotPresent
    name: 4pod
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-hwmx
      readOnly: true
```

## Actividad adicional

# Vamos fortaleciendo habilidades

Para ello formaremos equipos mismos que tendrá un tiempo de 40 minutos para resolver el crimen sucedido mediante una serie de consultas.

Para ellos es necesario de forma colaborativa un integrante sea quien proyecte y entre todos resuelvan este misterio

Accederemos de favor al siguiente link.

<https://mystery.knightlab.com/>

---

## Introducción a los archivos YAML para configuración

- En K8s los archivos YAML se utilizan para definir y describir los recursos del cluster de manera declarativa (se enfoca en el qué, no en el como). Se utilizan para especificar objetos de Kubernetes como Pods, deployments, services, ConfigMaps, Secrets, etc.
- La estructura sigue una convención y sintaxis que debe ser respetada para que Kubernetes interprete adecuadamente los recursos.

---

# Introducción a los archivos YAML para configuración

Estructura básica de un archivo YAML en Kubernetes

- Un archivo YAML en Kubernetes tiene una estructura jerárquica basada en clave-valor, con una indentación espaciada (usualmente de 2 espacios). El archivo define el tipo de recurso y sus configuraciones.
- Los elementos clave que usualmente encontrarás en un archivo YAML de Kubernetes son:
- **apiVersion:** Especifica la versión de la API de Kubernetes que se utilizará para crear el recurso.
- **kind:** Define el tipo de recurso que se está creando (por ejemplo, Pod, Deployment, Service).
- **metadata:** Contiene información como el nombre, las etiquetas y las anotaciones del recurso.
- **spec:** Define la configuración y las características del recurso. Dependiendo del tipo de recurso, el contenido de spec será diferente.

# YAML para configuración de un Pod

```
YAMLS > ! pod1.YAML
1   apiVersion: v1          # Versión de la API de Kubernetes
2   kind: Pod              # Tipo de recurso (en este caso, un Pod)
3   metadata:               # Información de metadatos
4     name: my-app-pod      # Nombre del Pod
5     labels:                # Etiquetas del Pod
6       app: my-app
7   spec:                   # Especificación del Pod
8     containers:            # Lista de contenedores que se ejecutarán en el Pod
9       - name: my-app-container  # Nombre del contenedor
10      image: nginx:latest    # Imagen del contenedor (nginx en este caso)
11      ports:                  # Puertos expuestos por el contenedor
12        - containerPort: 80    # Puerto expuesto dentro del contenedor
13
```

**apiVersion:** Indica la versión de la API de Kubernetes que se utilizará para el recurso. En este caso, v1 es la versión estable para la mayoría de los recursos básicos, como los Pods.

**kind:** Define el tipo de recurso, en este caso, un Pod.

**metadata:** Contiene metadatos que identifican el recurso. Aquí, se define el nombre del Pod (my-app-pod) y una etiqueta (app: my-app).

**spec:** Especifica la configuración de los contenedores que se ejecutarán en el Pod. En este caso, se define un contenedor llamado my-app-container que ejecuta la imagen nginx:latest y expone el puerto 80.

## YAML para configuración de un Pod

Podemos crear un pod, un despliegue o un servicio desde un archivo .yml con el comando **kubectl create –f <nombrearchivo.yml>**

Pero antes, debemos crear nuestro archivo .yml como ejemplo tenemos el siguiente

```
1 apiVersion: v1                      # Versión de la API de Kubernetes
2 kind: Pod                            # El tipo de recurso es Pod
3 metadata:
4   name: nginx-pod                   # El nombre del Pod
5   labels:                           # Etiquetas opcionales
6     app: nginx
7 spec:
8   containers:
9     - name: nginx-container          # Nombre del contenedor
10    image: nginx:latest             # Imagen del contenedor
11    ports:
12      - containerPort: 80           # Puerto que expone el contenedor (HTTP)
```

# YAML para configuración de un Deployment

```
YAMLS > ! deployment1.YAML
1  apiVersion: apps/v1          # Versión de la API para el tipo "Deployment"
2  kind: Deployment            # Tipo de recurso (Deployment)
3  metadata:
4    name: my-app-deployment   # Nombre del Deployment
5  spec:
6    replicas: 3                # Número de réplicas de Pods
7    selector:
8      matchLabels:
9        app: my-app            # Las etiquetas para coincidir con los Pods
10   template:
11     metadata:
12       labels:
13         app: my-app          # Plantilla para los Pods creados por el Deployment
14   spec:
15     containers:
16       - name: my-app-container # Nombre del contenedor
17         image: nginx:latest   # Imagen del contenedor
18         ports:
19           - containerPort: 80   # Puerto expuesto dentro del contenedor
```

**replicas:** Especifica el número de réplicas de Pods que deben estar corriendo para este Deployment. Kubernetes asegura que siempre haya 3 Pods en ejecución.

**selector:** Especifica cómo Kubernetes selecciona los Pods que serán gestionados por este Deployment. Aquí, se seleccionan los Pods con la etiqueta app: my-app.

**template:** Define la plantilla de los Pods que se crearán. Contiene la misma estructura que el recurso Pod, pero en este caso está dentro del contexto del Deployment.

**containers:** Define los contenedores que se ejecutarán dentro de los Pods. En este ejemplo, el contenedor se llama my-app-container y usa la imagen nginx:latest.

# YAML para configuración de un deployment

Podemos crear un pod, un despliegue o un servicio desde un archivo .yml con el comando **kubectl create –f <nombrearchivo.yml>** o con **kubectl apply -f nombrearchivo>.yaml**

Pero antes, debemos crear nuestro archivo .yml como ejemplo tenemos el siguiente

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3 # Número de réplicas que se desean
  selector:
    matchLabels:
      app: nginx # Las réplicas se gestionarán según estas etiquetas
  template:
    metadata:
      labels:
        app: nginx # Las etiquetas del pod, deben coincidir con el selector
    spec:
      containers:
        - name: nginx-container
          image: nginx:latest # Imagen de Docker del contenedor
          ports:
            - containerPort: 80 # Puerto expuesto por el contenedor
```

# YAML para configuración de un deployment

Esto es lo que podemos observar en la linea de comandos

```
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl apply -f deployment1.yml
deployment.apps/nginx-deployment created
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           18s
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
mypod                1/1     Running   0          6h39m
mypod2               1/1     Running   0          6h39m
nginx-deployment-6575f54b8f-bjgcm  1/1     Running   0          44s
nginx-deployment-6575f54b8f-c6rq4  1/1     Running   0          44s
nginx-deployment-6575f54b8f-dqhgj  1/1     Running   0          44s
nginx-pod             1/1     Running   0          27m
```

---

**Antes de iniciar vamos a romper el hielo...**

¿Qué superpoder te gustaría tener? ¿Por qué?



# YAML para configuración de un deployment

Podemos escalar o modificar el numero de replicas, por ejemplo, con el comando **kubectl scale deployment <nombredeployment> --replicas=#replicas**

```
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl scale deployment nginx-deployment --replicas=2
deployment.apps/nginx-deployment scaled
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl scale deployment nginx-deployment --replicas=4
```

Si obtenemos la informacion de los deployments, deberia coincidir con la última modificación que son 4 replicas

```
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   4/4      4           4           10m
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
mypod                1/1     Running   0          6h51m
mypod2               1/1     Running   0          6h50m
nginx-deployment-6575f54b8f-bjgcm 1/1     Running   0          12m
nginx-deployment-6575f54b8f-dqh gj 1/1     Running   0          12m
nginx-deployment-6575f54b8f-xgp96 1/1     Running   0          2m58s
nginx-deployment-6575f54b8f-xrxzk 1/1     Running   0          2m58s
nginx-pod             1/1     Running   0          39m
```

## YAML para configuración de un deployment

Tambien podemos borrar el deployment, con el comando **kubectl delete -f <nombrearchivo.yml>**

```
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl delete -f deployment1.yml
deployment.apps "nginx-deployment" deleted
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ kubectl get deployments
No resources found in default namespace.
parallels@ubuntu-linux-22-04-02-desktop:~/K8s$ █
```

# YAML para configuración de un Service

```
! service.yml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: mi-servicio
5    namespace: default
6  spec:
7    selector:
8      app: mi-aplicacion
9    ports:
10      - protocol: TCP
11        port: 80
12        targetPort: 8080
13        type: ClusterIP
14
```

- **apiVersion:** La versión de la API que se usa para definir el recurso, en este caso, v1 para un Service.
- **kind:** El tipo de recurso que se está creando, en este caso, un Service.
- **metadata:name:** El nombre del Service. En este caso, se llama mi-servicio.
- **namespace:** El namespace en el que se crea el Service, que en este ejemplo es default.
- **spec:selector:** Este es el conjunto de etiquetas que Kubernetes usará para seleccionar los pods que serán parte de este Service. En este caso, selecciona los pods con la etiqueta app=mi-aplicacion.
- **ports:** Especifica los puertos que el Service expondrá.
  - **port:** El puerto en el que el Service estará disponible dentro del clúster, en este caso 80.
  - **targetPort:** El puerto en los pods seleccionados donde el Service enviará el tráfico, en este caso 8080.
- **type:** El tipo de Service. En este ejemplo es ClusterIP, lo que significa que el Service es accesible solo dentro del clúster. Otros tipos comunes son NodePort y LoadBalancer.

## Recursos de k8s definidos en un yml

- En Kubernetes, un archivo YAML (YAML Ain't Markup Language) se utiliza para definir recursos que Kubernetes gestionará dentro de un clúster. Estos recursos pueden ser de diferentes tipos y tienen una estructura específica.
- **Pods:** Ejecutan contenedores y pueden ser gestionados por otros recursos.
- **Deployments:** Gestionan la creación y el ciclo de vida de los Pods, incluyendo escalado y actualizaciones.
- **Services:** Exponen los Pods y gestionan el acceso a ellos dentro y fuera del clúster.
- **ConfigMaps:** Almacenan configuraciones que los Pods pueden utilizar.
- **Secrets:** Almacenan información sensible, como contraseñas y claves.

## Recursos de k8s definidos en un yml

- En Kubernetes, un archivo YAML (YAML Ain't Markup Language) se utiliza para definir recursos que Kubernetes gestionará dentro de un clúster. Estos recursos pueden ser de diferentes tipos y tienen una estructura específica.
- **Ingress:** Gestiona el acceso externo a los servicios mediante HTTP/HTTPS.
- **PersistentVolume (PV) y PersistentVolumeClaim (PVC):** Estos recursos se usan para gestionar almacenamiento persistente en Kubernetes. Un **PersistentVolume** es un recurso de almacenamiento en el clúster, y un **PersistentVolumeClaim** es una solicitud de almacenamiento por parte de un Pod.

## Archivo yaml para un Pod

- Un **Pod** es la unidad básica de ejecución en Kubernetes. Define uno o más contenedores, volúmenes, redes, etcv1

```
apiVersion: v1
kind: Pod
metadata:
  name: mi-pod
spec:
  containers:
    - name: nginx
      image: nginx:latest
      ports:
        - containerPort: 80
```

## Archivo yaml para un Deployment

- Un **Deployment** administra un conjunto de Pods replicados, asegurando que haya un número determinado de Pods en ejecución y gestionando actualizaciones y rollbacks de forma controlada.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mi-aplicacion
  template:
    metadata:
      labels:
        app: mi-aplicacion
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

## Archivo yaml para un Service

Un **Service** expone un conjunto de Pods a través de una dirección IP estable y un nombre DNS. Puede ser de diferentes tipos (ClusterIP, NodePort, LoadBalancer).

```
apiVersion: v1
kind: Service
metadata:
  name: mi-servicio
spec:
  selector:
    app: mi-aplicacion
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

## Archivo yaml para un Configmap

Un **ConfigMap** permite almacenar datos de configuración que pueden ser utilizados por los Pods y otros recursos en el clúster. Es útil para separar los datos de configuración de la lógica de la aplicación sin necesidad de modificar los contenedores.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mi-configmap
  namespace: default
  labels:
    app: mi-aplicacion
data:
  database_host: "db.example.com"
  database_port: "5432"
  api_key: "1234567890abcdef"
  app_mode: "production"
  log_level: "debug"
```

**apiVersion:**

- Define la versión de la API de Kubernetes que estamos utilizando para este recurso. En este caso, v1 es la versión para recursos básicos como ConfigMap.

**kind:**

- Especifica el tipo de recurso que estamos creando. En este caso, es un ConfigMap.

**metadata:**

- **name:** Define el nombre del ConfigMap. En este caso, se llama mi-configmap.

- **namespace:** Especifica el espacio de nombres (namespace) en el que se debe crear el ConfigMap. El valor por defecto es default, pero podrías cambiarlo a cualquier otro namespace según lo necesites.

## Archivo yaml para un Configmap

Un **ConfigMap** permite almacenar datos de configuración que pueden ser utilizados por los Pods y otros recursos en el clúster. Es útil para separar los datos de configuración de la lógica de la aplicación sin necesidad de modificar los contenedores.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mi-configmap
  namespace: default
  labels:
    app: mi-aplicacion
data:
  database_host: "db.example.com"
  database_port: "5432"
  api_key: "1234567890abcdef"
  app_mode: "production"
  log_level: "debug"
```

•**labels:** Las etiquetas son clave-valor que se pueden usar para identificar y organizar el recurso dentro del clúster. En este caso, tenemos la etiqueta app: mi-aplicacion.

**data:**

- En esta sección se define la configuración que el ConfigMap va a almacenar. Los datos se representan como pares clave-valor.
- Cada clave representa el nombre de la variable de configuración, y cada valor es el valor de esa variable.

•**Ejemplos:**

- database\_host: La dirección del servidor de base de datos.
- database\_port: El puerto en el que la base de datos está escuchando.
- api\_key: Una clave de API para interactuar con un servicio externo.
- app\_mode: El modo de operación de la aplicación, como "producción" o "desarrollo".
- log\_level: El nivel de logs de la aplicación (por ejemplo, "debug", "info", "error").

## Archivo yaml para un Secret

Un **Secret** se utiliza para almacenar información sensible como contraseñas, tokens, o claves SSH. Los Secrets se gestionan de forma más segura que los ConfigMaps. K8s los codifica los datos almacenados en base64 y los Pods pueden acceder de forma segura como variables de entorno

```
apiVersion: v1
kind: Secret
metadata:
  name: mi-secreto
  namespace: default
  labels:
    app: mi-aplicacion
type: Opaque
data:
  username: bXI1c2Vy # (base64)
  password: cGFzc3dvcmQ= # (b
```

**apiVersion:** Define la versión de la API de Kubernetes que estamos utilizando para este recurso. En este caso, v1 es la versión para recursos básicos como Secret.

**kind:** Especifica el tipo de recurso. En este caso, es un Secret.

**metadata:**

- **name:** Especifica el nombre del Secret (mi-secreto).

- **namespace:** Define el espacio de nombres (namespace) en el que se creará el Secret. En este caso, se está creando en el namespace default.

- **labels:** Añade etiquetas para facilitar la identificación y gestión del recurso. En este caso, tenemos la etiqueta app: mi-aplicacion.

**type:** El tipo de Secret, en este caso es Opaque, lo que significa que es un Secret genérico. Otros tipos incluyen kubernetes.io/dockerconfigjson (para almacenar credenciales de Docker), kubernetes.io/tls (para certificados TLS), entre otros.

**data:**

- En esta sección se definen los pares clave-valor que almacenan la información sensible. Las claves (como username y password) son las que se utilizarán en el contenedor para acceder a los valores.

- Los valores de los datos deben estar **codificados en base64**. Por ejemplo:

- myuser codificado en base64 es bXI1c2Vy.
- password codificado en base64 es cGFzc3dvcmQ.

## Archivo yaml para un ingress

Un **Ingress** se utiliza para gestionar el acceso externo a los servicios dentro de un clúster, proporcionando una capa de abstracción para manejar el tráfico HTTP/HTTPS.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mi-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: mi-aplicacion.mydomain.com
    http:
      paths:
      - path: /mi-aplicacion
        pathType: Prefix
        backend:
          service:
            name: mi-servicio
            port:
              number: 80
    # Opcionalmente, también podrías definir reglas HTTPS aquí.
    tls:
    - hosts:
      - mi-aplicacion.mydomain.com
      secretName: mi-certificado-tls
```

**apiVersion:** networking.k8s.io/v1 es la versión de la API que se utiliza para crear un recurso de tipo **Ingress**.

**kind:** Define el tipo de recurso, que en este caso es un **Ingress**.

**metadata:**

- name:** El nombre del recurso **Ingress**, en este caso es mi-ingress.

- namespace:** El espacio de nombres en el que se creará el Ingress. En este ejemplo, es default.

- annotations:** Anotaciones que se utilizan para configurar opciones adicionales en el controlador de Ingress. En este caso, la anotación nginx.ingress.kubernetes.io/rewrite-target establece la regla de reescritura de la URL, lo que significa que cualquier solicitud que llegue a /mi-aplicacion será redirigida a la raíz del servicio backend (es decir, la ruta /).

## Archivo yaml para un ingress

Un **Ingress** se utiliza para gestionar el acceso externo a los servicios dentro de un clúster, proporcionando una capa de abstracción para manejar el tráfico HTTP/HTTPS.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mi-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: mi-aplicacion.mydomain.com
    http:
      paths:
      - path: /mi-aplicacion
        pathType: Prefix
        backend:
          service:
            name: mi-servicio
            port:
              number: 80
      # Opcionalmente, también podrías definir reglas HTTPS aquí.
      tls:
      - hosts:
        - mi-aplicacion.mydomain.com
        secretName: mi-certificado-tls
```

### spec:

- **rules:** Define las reglas para enrutar el tráfico a los servicios backend.
  - **host:** Especifica el nombre del host al que se deben enrutar las solicitudes. En este caso, el tráfico que llega a mi-aplicacion.mydomain.com será gestionado por este Ingress.
  - **http:** Define las reglas de rutas HTTP.
    - **paths:** Define los caminos dentro del host que se deben enrutar a un servicio específico.
    - **path:** La ruta que se debe emparejar. En este caso, cualquier solicitud que llegue a /mi-aplicacion será redirigida.
  - **pathType:** Prefix significa que cualquier ruta que comience con /mi-aplicacion será dirigida al servicio de backend. Otras opciones de pathType son Exact (coincidencia exacta de la ruta) y ImplementationSpecific (dependiente del controlador de Ingress).
  - **backend:** Define a qué servicio debe enrutar el tráfico.
    - **service:** El nombre del servicio que gestionará las solicitudes entrantes.
      - **name:** mi-servicio es el nombre del servicio de Kubernetes al que se enrutarán las solicitudes.
      - **port:** El puerto del servicio al que se enrutarán las solicitudes. En este caso, el puerto es 80.

## Archivo yaml para un Persistent Volume y PVC

Estos recursos se usan para gestionar almacenamiento persistente en Kubernetes. Un **PersistentVolume** es un recurso de almacenamiento en el clúster, y un **PersistentVolumeClaim** es una solicitud de almacenamiento por parte de un Pod.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mi-pv
spec:
  capacity:
    storage: 5Gi # Espacio de almacenamiento solicitado
  volumeMode: Filesystem # Define el tipo de volumen, generalmente 'Filesystem'
  accessModes:
    - ReadWriteOnce # Define el modo de acceso. En este caso, solo un Pod puede acceder en modo lectura/escritura
  persistentVolumeReclaimPolicy: Retain # Define qué sucede con el volumen cuando el PVC se elimina (otras opciones: Recycle, Delete)
  storageClassName: standard # La clase de almacenamiento a la que pertenece el volumen
  hostPath:
    path: /mnt/data/mi-pv # Usando un volumen hostPath (para entornos locales o de desarrollo)
```

## Archivo yaml para un Persistent Volume y PVC

Estos recursos se usan para gestionar almacenamiento persistente en Kubernetes. Un

**PersistentVolume** es un recurso de almacenamiento en el clúster, y un **PersistentVolumeClaim** es una solicitud de almacenamiento por parte de un Pod.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mi-pv
spec:
  capacity:
    storage: 5Gi # Espacio de almacenamiento solicitado
  volumeMode: Filesystem # Define el tipo de volumen, generalmente 'Filesystem'
  accessModes:
    - ReadWriteOnce # Define el modo de acceso. En este caso, solo un Pod puede acceder en modo lectura/escritura
  persistentVolumeReclaimPolicy: Retain # Define qué sucede con el volumen cuando el PVC se elimina (otras opciones: Recycle, Delete)
  storageClassName: standard # La clase de almacenamiento a la que pertenece el volumen
  hostPath:
    path: /mnt/data/mi-pv # Usando un volumen hostPath (para entornos locales o de desarrollo)
```

# Archivo yaml para un Persistent Volume y PVC

- **capacity:** Define la cantidad de almacenamiento disponible para el PV. En este caso, es de 5 GiB.
- **volumeMode:** Define si el volumen será un sistema de archivos o un bloque. Normalmente se usa Filesystem.
- **accessModes:** Especifica los modos de acceso permitidos. En este caso, ReadWriteOnce significa que el volumen puede ser montado en solo un Pod a la vez con acceso de lectura/escritura. Otros modos comunes son ReadOnlyMany y ReadWriteMany.
- **persistentVolumeReclaimPolicy:** Determina qué sucede con el volumen cuando el PVC asociado se elimina. Puede ser Delete (eliminar el volumen), Retain (mantener el volumen) o Recycle (volver a formatear el volumen).
- **storageClassName:** Especifica la clase de almacenamiento que se usará para este volumen (por ejemplo, standard).
- **hostPath:** Utiliza un directorio local del nodo como almacenamiento. En este ejemplo, el volumen está respaldado por el directorio /mnt/data/mi-pv en el nodo.  
**Nota:** hostPath se usa generalmente para propósitos de desarrollo o pruebas. En producción, es más común usar volúmenes como awsElasticBlockStore, gcePersistentDisk, nfs, etc., según el entorno de Kubernetes.

# Archivo yaml para un Persistent Volume y PVC

Estos recursos se usan para gestionar almacenamiento persistente en Kubernetes. Un **PersistentVolume** es un recurso de almacenamiento en el clúster, y un **PersistentVolumeClaim** es una solicitud de almacenamiento por parte de un Pod.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mi-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce # Modo de acceso requerido
  resources:
    requests:
      storage: 5Gi # Cantidad de almacenamiento solicitado
  storageClassName: standard # La clase de almacenamiento para la reclamación
```

• **accessModes:** Define los modos de acceso que el PVC requiere. En este caso, ReadWriteOnce, lo que significa que solo un Pod puede montar el volumen en modo lectura/escritura.

• **resources:**

- **requests:** Especifica los recursos solicitados. En este caso, el PVC solicita 5Gi de almacenamiento.

• **storageClassName:** Define la clase de almacenamiento que se usará para este PVC. En este caso, se está utilizando la clase standard.

**Nota:** El PVC se asociará con el PV que tenga una clase de almacenamiento y tamaño compatible con lo que se solicita. Si no se especifica un storageClassName, se usará la clase predeterminada.

## Archivo yaml para un HPA

Un **HorizontalPodAutoscaler** ajusta automáticamente el número de réplicas de un Deployment o ReplicaSet basado en la carga de trabajo (por ejemplo, uso de CPU o memoria).

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: mi-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mi-deployment # El nombre del Deployment que será escalado
  minReplicas: 2 # Número mínimo de réplicas
  maxReplicas: 10 # Número máximo de réplicas
  metrics:
    - type: Resource
      resource:
        name: cpu
      target:
        type: Utilization
        averageUtilization: 50 # La CPU promedio a la que se debe escalar (en porcentaje)
```

**apiVersion:** autoscaling/v2: Define la versión de la API de autoscaling utilizada. En Kubernetes 1.18 y versiones posteriores, autoscaling/v2 es la versión más común, ya que permite usar métricas más avanzadas como memory, custom metrics, y la definición de objetivos de tipo Utilization o Value.

**kind:** HorizontalPodAutoscaler: Define el tipo de recurso como un autoscalador de Pods.  
**metadata:**

- **name:** El nombre del HPA. En este caso, se llama mi-hpa.

- **namespace:** El namespace en el que se va a crear el HPA. En este ejemplo, es default.

**spec:**

- **scaleTargetRef:**

- **apiVersion:** La versión de la API para el recurso que se va a escalar. En este caso, es apps/v1 para un Deployment.

- **kind:** El tipo de recurso a escalar. En este caso, es un Deployment.

- **name:** El nombre del recurso que se va a escalar. En este caso, es mi-deployment.

## Archivo yaml para un HPA

Un **HorizontalPodAutoscaler** ajusta automáticamente el número de réplicas de un Deployment o ReplicaSet basado en la carga de trabajo (por ejemplo, uso de CPU o memoria).

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: mi-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mi-deployment # El nombre del Deployment que será escalado
  minReplicas: 2 # Número mínimo de réplicas
  maxReplicas: 10 # Número máximo de réplicas
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50 # La CPU promedio a la que se debe escalar (en porcentaje)
```

- **minReplicas:** El número mínimo de réplicas del Deployment que debe tener el HPA. En este caso, el mínimo es 2 réplicas.
- **maxReplicas:** El número máximo de réplicas del Deployment que puede alcanzar el HPA. En este caso, el máximo es 10 réplicas.
- **metrics:** La lista de métricas que el HPA usará para decidir cómo escalar el número de réplicas.
  - **type:** Resource indica que la métrica es un recurso del sistema, como cpu o memory.
  - **resource:**
    - **name:** El tipo de recurso que se mide, en este caso cpu.
    - **target:**
      - **type:** Utilization indica que la métrica se basa en la utilización del recurso (porcentaje de uso de CPU o memoria).
      - **averageUtilization:** El porcentaje objetivo de utilización de la CPU. Si la utilización promedio de la CPU de los Pods supera este valor, el HPA escalará hacia arriba. En este caso, si la CPU promedio supera el 50%, Kubernetes aumentará el número de réplicas.

## Ejemplo de un Pod con un configmap

```
apiVersion: v1
kind: Pod
metadata:
  name: mi-pod
  namespace: default
  labels:
    app: mi-aplicacion
spec:
  containers:
    - name: mi-container
      image: nginx:latest
      ports:
        - containerPort: 80
      env:
        - name: ENV_MODE
          value: "production"
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: mi-configmap
```

**apiVersion:** Define la versión de la API que se utilizará para el recurso. En este caso, estamos usando v1 para crear un Pod.

**kind:** Define el tipo de recurso, que en este caso es un Pod.

**metadata:**

- **name:** El nombre del Pod, que en este caso es mi-pod.

- **namespace:** El namespace donde se creará el Pod. En este caso es default.

- **labels:** Un conjunto de etiquetas que se pueden usar para organizar y seleccionar recursos. En este caso, el Pod tiene la etiqueta app: mi-aplicacion.

**spec:**

- **containers:** Define los contenedores que deben ejecutarse dentro del Pod.

- **name:** El nombre del contenedor, en este caso mi-container.
- **image:** La imagen de Docker que se utilizará para crear el contenedor. En este caso, estamos usando la imagen oficial de nginx en su versión más reciente (nginx:latest).
- **ports:** Expone el puerto 80 dentro del contenedor para que el tráfico entrante pueda acceder a la aplicación.
- **env:** Define variables de entorno para el contenedor. En este caso, se está configurando la variable de entorno ENV\_MODE con el valor production.
- **volumeMounts:** Especifica los volúmenes que se deben montar dentro del contenedor. En este caso, el volumen config-volume se montará en la ruta /etc/config dentro del contenedor.

- **volumes:** Define los volúmenes que el Pod utilizará.

- **name:** El nombre del volumen, en este caso config-volume.
- **configMap:** Este volumen está basado en un ConfigMap llamado mi-configmap. El contenido del ConfigMap será montado como archivos dentro del contenedor en la ruta /etc/config.

## Ejemplo de un Pod con un configmap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mi-configmap
  namespace: default
data:
  config1.txt: "Configuración A"
  config2.txt: "Configuración B"
```

Este ConfigMap contiene dos archivos de configuración (config1.txt y config2.txt), y los valores serán montados dentro del contenedor en la ruta /etc/config del Pod.

## Actividad 3

El objetivo de esta actividad es crear un Pod en Kubernetes utilizando un archivo YAML y desplegar una aplicación sencilla, por ejemplo nginx en el clúster o un httpd, una aplicación de python



## Ejemplo usando configmap, secret, deployment, service, HPA

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: default
data:
  app.properties: |
    key1=value1
    key2=value2
```

```
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: default
type: Opaque
data:
  username: YWRtaW4= # "admin" codificado en base64
  password: cGFzc3dvcmQ= # "password" codificado en base64
```

# Ejemplo usando configmap, secret, deployment, service, HPA

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
          image: nginx:latest
          ports:
            - containerPort: 80
          env:
            - name: APP_CONFIG
              valueFrom:
                configMapKeyRef:
                  name: app-config
                  key: app.properties
            - name: DB_USERNAME
              valueFrom:
                secretKeyRef:
                  name: db-secret
                  key: username
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: db-secret
                  key: password
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
  namespace: default
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP # Exposición interna del servicio dentro del clúst
```

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app-deployment
  minReplicas: 2
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

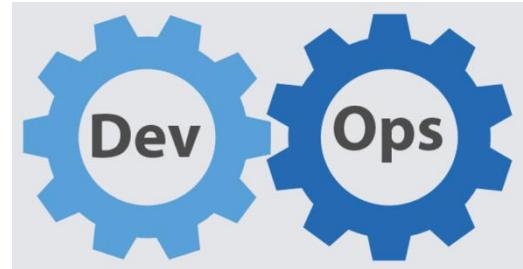
## Ejemplo usando configmap, secret, deployment, service, HPA

Este archivo YAML demuestra cómo se pueden gestionar diferentes recursos en Kubernetes, todo dentro de un solo archivo. Los recursos utilizados son:

- **ConfigMap**: Para gestionar configuraciones.
  - **Secret**: Para almacenar datos sensibles.
  - **Deployment**: Para administrar Pods de manera declarativa.
  - **Service**: Para exponer los Pods dentro del clúster.
  - **HorizontalPodAutoscaler (HPA)**: Para escalar automáticamente los Pods según el uso de recursos.
- 
- Este tipo de estructura es común en entornos donde se necesita manejar configuraciones, secretos, escalabilidad y exposición de servicios dentro de Kubernetes de manera eficiente y automatizada.

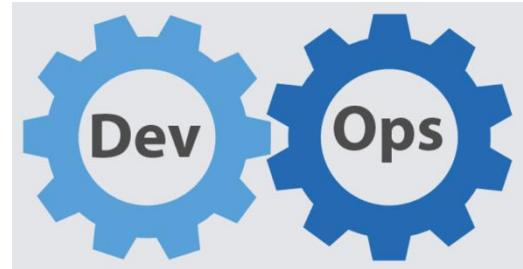
## Introducción

- De la misma forma en que Agile, Test Driven Development y CI han cambiado la forma en que se escribe código y se gestionan los proyectos...
- DevOps y tecnologías relacionadas como IaC (Infrastructure as Code) and CD (Continuos Deliver/deployment) están cambiando las operaciones la forma en que se entregan los servicios de IT.
- Asimismo, como Scrum y XP (Extreme programming) reemplazaron CMMi y waterfall, DevOps “reemplaza” ITIL como la forma preferida de gestionar TI.



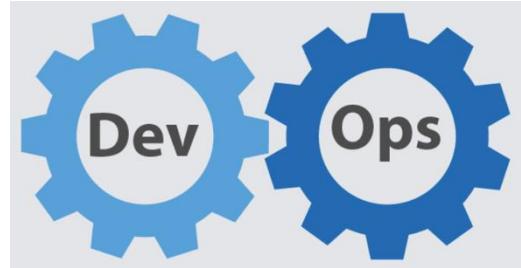
## Introducción

- Antes del año 2000 , un gran porcentaje del software era desarrollado y actualizado usando la metodología en cascada (waterfall)...
- Los **equipos de desarrollo** pasaban meses desarrollando grandes cantidades de código nuevo que impactaba la mayoría o totalidad de la aplicación, pero se **demoraban meses** en integrarlo al código base, y no sabian si el resultado, despues de tanto tiempo sería el esperado.
- Posterior, los equipos de QA, seguridad y operaciones hacian sus pruebas y **demoraban aún más**. Sumado eran **varios meses hasta llegar a veces a años**, entre **versiones de software**, o "parches", asimismo, **planes complejos** que requerian tambien un equipo complejo.



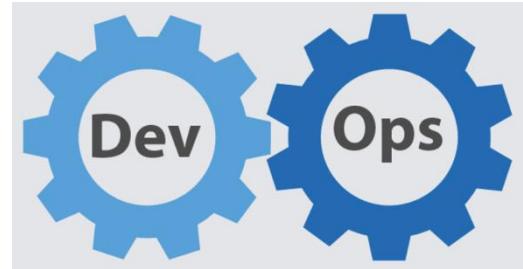
## Introducción

- Para acelerar el proceso, se empezaron a implementar metodologías de desarrollo ágiles (LEAN y Agile manifesto), que eran **iterativas** en lugar de **lineales... como la integración y entrega continua**. Fragmentos más pequeños de código nuevo se integraban al código base con mayor frecuencia (semanas).
- Pero esto generó nuevos cuellos de botella...principalmente en el equipo de operaciones que empezaron a tener contratiempos de responder a las necesidades de los desarrolladores.
- Por lo tanto, surgió la necesidad de nuevos mecanismos de aprovisionamiento, automatización, nuevas formas de gestión, monitoreo y respuesta.



## Introducción

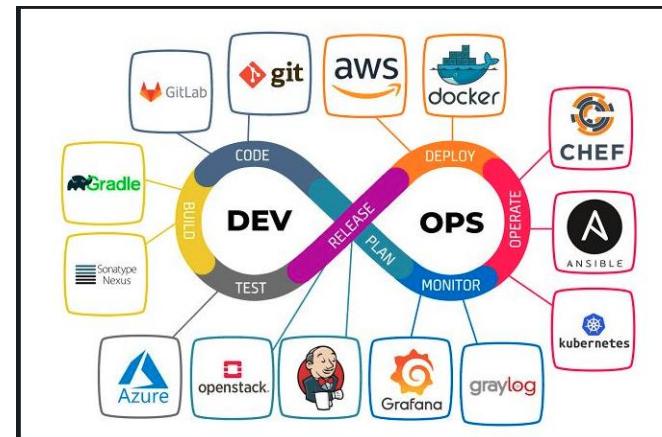
- Así nace **DevOps**... un marco de trabajo/metodología, que busca complementar las metodologías **ágiles** con nuevos procesos y herramientas que aumentan la iteración y automatización de CI/CD del ciclo de vida de desarrollo de software.
- Aumentando y estrechando la colaboración de los equipos de **desarrollo** y **operaciones**.
- **DevOps** nos ayuda a acelerar la entrega de software de alta calidad combinando y automatizando el trabajo de ambos equipos.
- Es un cambio “**cultural**” de trabajo y colaboración de dos equipos que solían hacerlo por separado.
- Busca satisfacer la demanda, que cada vez es mayor de **software**, asimismo, características **nuevas, frecuentes e innovadoras**.



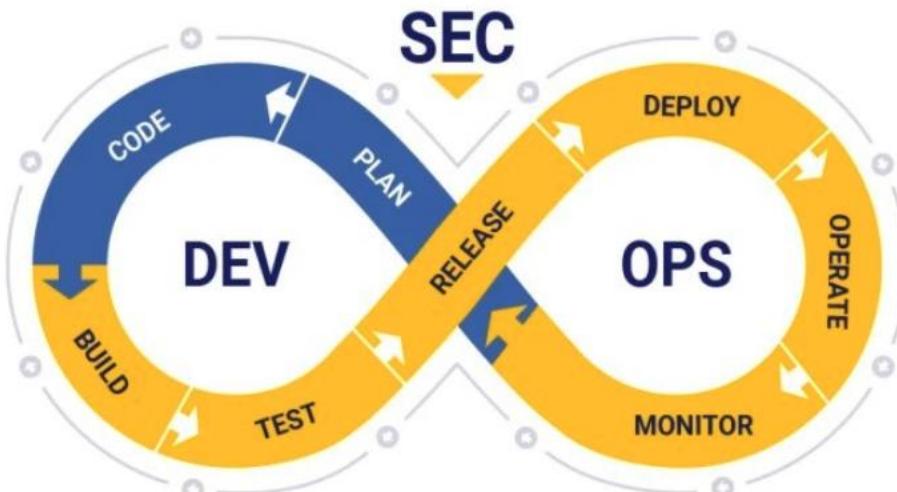
30 veces mas

## Introducción

- Ha permitido desplegar cambios 30 veces más que otras organizaciones, con tiempos de entrega hasta 200 veces más cortos.
- La tasa de éxito asociada a los cambios generados es 60 veces mayor
- Cuando algo no sucede de la mejor forma, puede recuperarse de fallas hasta 168 veces más rápido



## ¿Qué es DevSecOps?



- Desarrollo seguro y eficiente
- Unión de desarrollo, seguridad y operaciones
- Permite proactivamente identificar y mitigar:
- Riesgos
- Fomenta un desarrollo más ágil y resiliente
- Integracion de seguridad en SDLC (planeacion, código, pruebas y despliegue) detección proactiva de vulnerabilidades
- Automatizacion de controles de seguridad: Detección y remediaciion de problemas de seguridad
- Cambio cultural hacia una responsabilidad compartida. Entre los equipos de desarrollo, operaciones y seguridad. Lo que permite o facilita la colaboracion y comunicaciion y responsabilidad en la gestion de riesgos y poder obtener un mejor resultado

---

## Preparación del ambiente.

### Instalación de Jenkins

- Descargamos la imagen con **docker pull jenkins/jenkins**
- Creamos el contenedor con el comando **docker run --name jenkins -d -p 8080:8080 -p 50000:50000 -v jenkins\_home:/var/jenkins\_home jenkins/jenkins:lts**

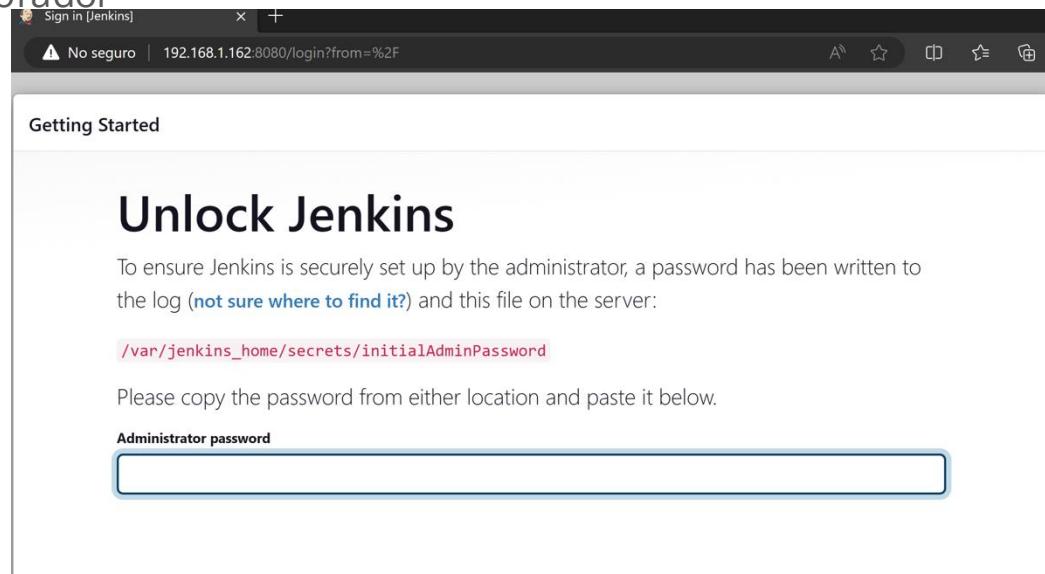
Crea y ejecuta un contenedor de Jenkins, exponiendo el puerto 8080 para la interfaz web y el puerto 50000 para la comunicación con agentes de Jenkins. Además, utiliza un volumen (jenkins\_home) para almacenar los datos de Jenkins de forma persistente.

- En el explorador abrimos <http://localhost:8080>
- En la terminal ejecutamos **docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword**
- Para obtener el primer password que nos pide para iniciar, mismo que pegamos en la pagina de jenkins

# Preparación del ambiente.

## Instalación de Jenkins

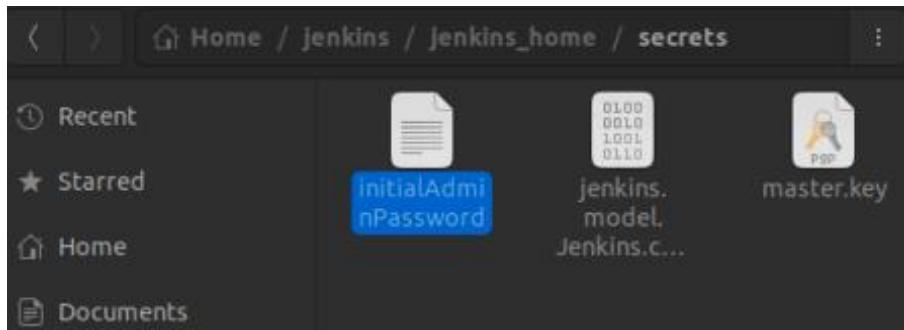
Una vez instalado todo, es necesario inicializar los servicios, y lo hacemos ejecutando lo siguiente: **localhost:8080** en un explorador



# Preparación del ambiente.

## Instalación de Jenkins

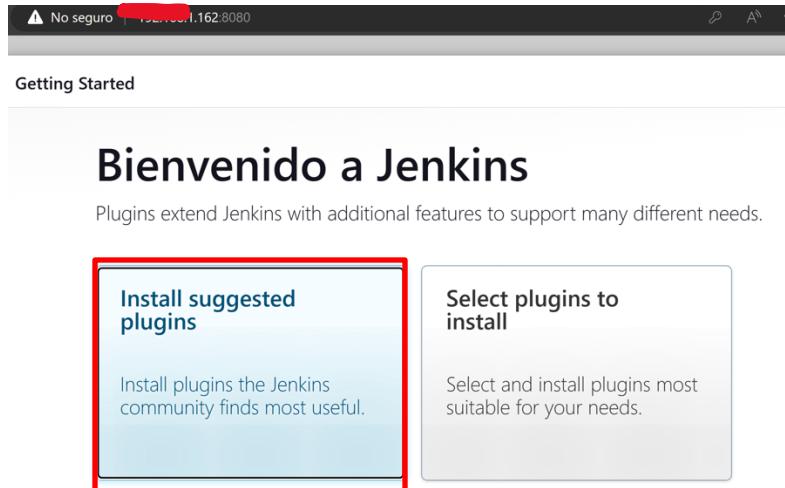
Para obtener la contraseña hay varias formas, la más simple y rápida es acceder una carpeta llamada **secrets** que está dentro de **Jenkins / Jenkins/Jenkins/secrets** donde hay un fichero con el nombre **initialAdminPassword** en él se encontrará la contraseña para ingresar por primera vez.



# Preparación del ambiente.

## Instalación de Jenkins

Una vez que tenemos las credenciales, y accedemos, la siguiente pantalla es lo primero que nos pregunta Jenkins, ¿qué plugins deseamos instalar?.

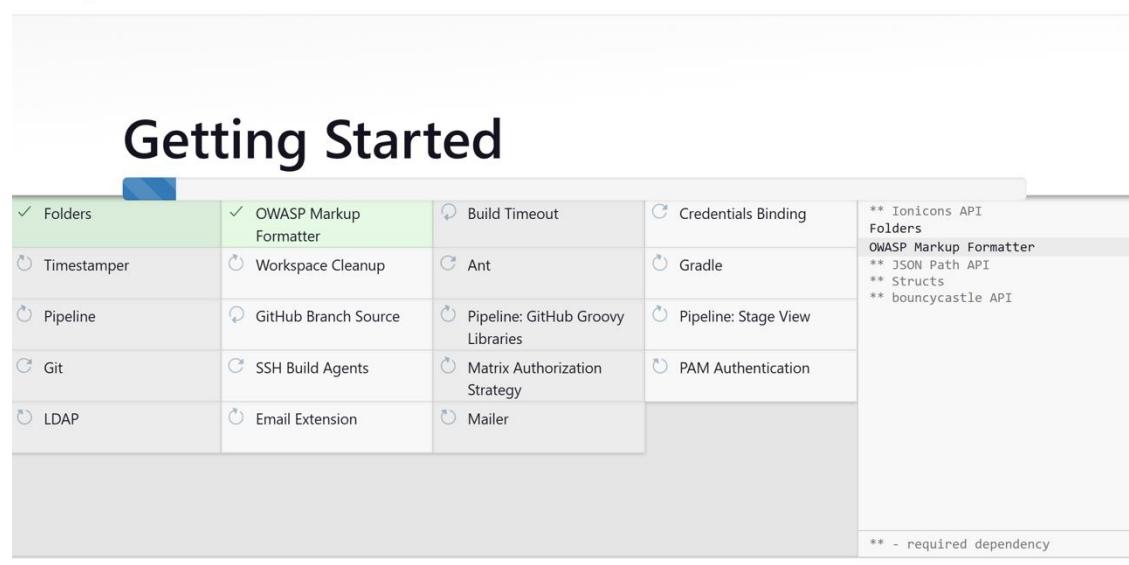


# Preparación del ambiente.

## Instalación de Jenkins

Instalamos los plugins Recomendados, que están Marcados por defecto, Después durante el curso, Instalaremos algunos Adicionales.

### Getting Started



The screenshot shows the Jenkins 'Getting Started' page. At the top, it says 'Getting Started'. Below that is a large heading 'Getting Started'. A table lists recommended Jenkins plugins:

✓ Folders	✓ OWASP Markup Formatter	⌚ Build Timeout	⌚ Credentials Binding
⌚ Timestamper	⌚ Workspace Cleanup	⌚ Ant	⌚ Gradle
⌚ Pipeline	⌚ GitHub Branch Source	⌚ Pipeline: GitHub Groovy Libraries	⌚ Pipeline: Stage View
⌚ Git	⌚ SSH Build Agents	⌚ Matrix Authorization Strategy	⌚ PAM Authentication
⌚ LDAP	⌚ Email Extension	⌚ Mailer	

To the right of the table, there is a sidebar with the following text:

- \*\* Ionic API
- Folders
- OWASP Markup Formatter
  - \*\* JSON Path API
  - \*\* Structs
  - \*\* bouncycastle API

At the bottom of the table, it says '\*\* - required dependency'.

# Preparación del ambiente.

## Instalación de Jenkins

Cuando termine, debemos establecer un usuario administrador, que usaremos gran parte del curso.

**¡Es importante que recordemos estas credenciales!**

Getting Started

### Create First Admin User

Usuario

Contraseña

Confirma la contraseña

Nombre completo

# Preparación del ambiente.

## Instalación de Jenkins

Ponemos la IP de nuestro ambiente, o dejamos si es correcta con el puerto 8080  
Damos click en "save and finish". Despues en "start using Jenkins"

### Getting Started

## Instance Configuration

Jenkins URL:

http://[REDACTED]:8080/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD\_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Not now

Save and Finish

### Getting Started

# Jenkins is ready!

Your Jenkins setup is complete.

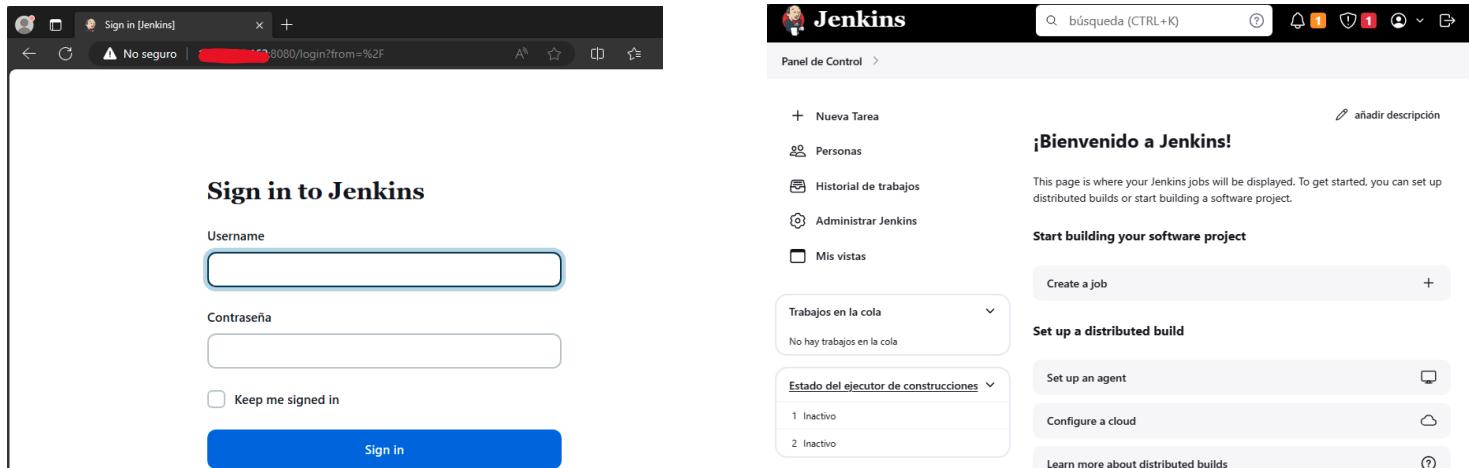
Start using Jenkins

# Preparación del ambiente.

## Instalación de Jenkins

Ahora, abrimos un explorador y dado que no es local, si no a través de una conexión SSH, debemos de cambiar el **localhost:puerto**, por **ip:8080** (puerto).

Ingresamos las credenciales y listo, tenemos acceso a Jenkins, como se ve en las imágenes.



The image contains two screenshots of the Jenkins web interface. The left screenshot shows the 'Sign in to Jenkins' page with fields for 'Username' and 'Contraseña', and a 'Keep me signed in' checkbox. The right screenshot shows the Jenkins dashboard with sections for 'Nueva Tarea', 'Personas', 'Historial de trabajos', 'Administrador Jenkins', 'Mis vistas', 'Trabajos en la cola' (empty), 'Estado del ejecutor de construcciones' (1 Inactivo, 2 Inactivo), and various setup options like 'Create a job', 'Set up a distributed build', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'.

---

# Jenkins

## ¿Qué es Jenkins?

Basicamente, podemos decir que Jenkins es un motor de automatización que soporta varios patrones de automatización.

Es una herramienta de CI/CD de código abierto escrita en java.

Es de las más utilizadas en el mundo, y tiene alrededor de 400K instalaciones.

Automatiza algunos procesos dentro del ciclo de vida del software, como el build y el deliver, en otras palabras, ayuda a construir(build) el producto y desplegar/entregarlo a:

- Producción
- Q&A
- Pruebas de rendimiento

Una de las funciones que lo volvió tan popular es el seguimiento a tareas repetidas que surgen durante el desarrollo de un proyecto.

---

# Jenkins

## ¿Qué es Jenkins?

Todos los roles gozan de sus beneficios, por ejemplo:

- Si eres desarrollador escribir tu pipeline as code será mas fácil y “natural”
- Si eres un experto en DevOps mantener el pipeline sera mas fácil, dado que será tratado como cualquier otro conjunto de código.
- Para un “tester” puedes ser mas ágil con características como el paralelismo para que los esfuerzos se materialicen mejor

---

# Jenkins

## ¿Qué es Jenkins?

Podemos diferenciar 2 versiones de Jenkins, Jenkins 1 y 2, nosotros trabajaremos con la segunda, dado que es la más recien y robusta.

- DSL incluida en Jenkins 2
- JenkinsFile tambien desde Jenkins 2
- Blue Ocean para visualizar graficamente nuestros pipelines

Estas y más características ahora están disponibles de forma nativa en Jenkins 2, cuando en la version previa muchas de estas debian ser utilizadas a traves de plugins y muchas veces en interfaces adicionales.

---

# Jenkins

## Jenkins Job (antes project)

- Son tareas ejecutables que son supervisadas y controladas por Jenkins.
- Es la descripción configurada por el usuario de una tarea o trabajo(job) que jenkins debe realizar ejemplo: la creacion de una “pieza” de software.
  
- Maestro y esclavo (master and slave).

## Jenkins

**Maestro:** tambien conocido como controller, tiene acceso completo a todas las opciones, configuraciones y Jobs, es el “lugar” por defecto para ejecutar los jobs si otro sistema no es especificado, y tiene varias funciones:

- Encargado de programar el build job (no ejecutar).
- Envia la compilacion(build) al esclavo para que se ejecute el job.
- Supervisar el trabajo del esclavo y registrar los resultados de la compilacion (build).
- Elección del esclavo para ejecutar los jobs.
- Despliegue o muestra de los resultados.

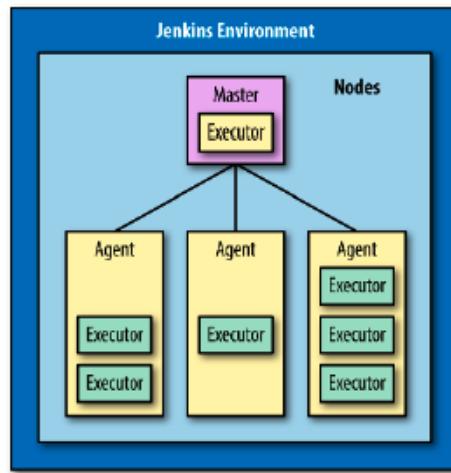
## Jenkins

**Agente/Esclavo:** cualquier sistema que no sea maestro, son máquinas virtuales programadas para construir los proyectos que el maestro le requiere, mandando los resultados al maestro, para ser mostrados.

**Executor:** es una secuencia separada de compilaciones que se ejecutarán en un nodo en paralelo. Basicamente es un slot en donde podemos ejecutar un job, en un nodo/agente. Un nodo puede tener 0 o "n" ejecutores, este define el número de Jobs que puede ejecutar concurrentemente.

**Nodo:** Es un nombre genérico usado desde Jenkins 2 para nombrar cualquier sistema que puede ejecutar un Job de Jenkins, cubre ambos, Agentes/esclavos y maestros.

# Jenkins



La forma en que jenkins trabaja es traves de los “jobs” y de estos se derivan los masters, slaves, nodes, etc. Entonces antes de crear nuestro primer Job, los tipos de Jobs que tenemos en Jenkins.

---

# Jenkins

Tipos de Jobs en Jenkins

**Proyecto de estilo libre:** son los proyectos base para la mayoría de los "jobs" de jenkins, el nombre indica que pueden ser construidos para hacer diferentes tareas.

**Proyecto maven:** busca simplificar algunas tareas comunes que utilizan Maven.

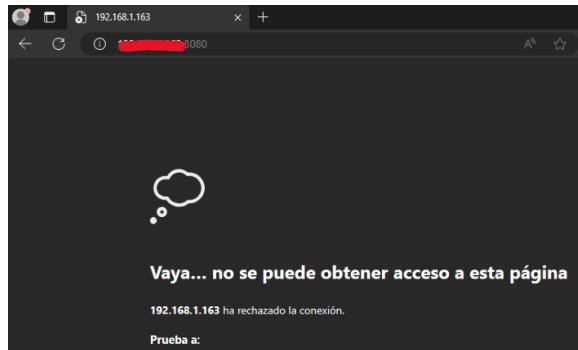
**Pipeline:** Es un tipo de proyecto donde los pasos o tareas y lógica de éste se especifican en un script de groovy en lugar de la herramienta web.

**Proyecto multiconfiguración:** Diseñado para simplificar la ejecución de un conjunto de compilaciones de proyectos que solo cambian en los parametros.

**Folder:** Es más una forma de organización que un tipo de job, originalmente las vistas han sido la forma de filtrar y mostrar información en panel de control.

**Multibranch:** la caracteristica principal es que puede automaticamente gestionar y compilar branches de proyectos manejados en SCM.

# Jenkins



Si queremos iniciar jenkins como lo hemos hecho en láminas anteriores, vemos que no es posible, esto es porque no hemos iniciado el servicio de jenkins, para hacerlo, ejecutamos uno de los comandos que mencionabamos previamente.

## docker-compose start

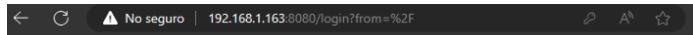
Pero antes, debemos posicionarnos en la carpeta/directorio donde se encuentra, con el comando cd Nombre de la carpeta (cd jenkins)

```
parallels@ubuntu-linux-22-04 ~ % docker-compose start
ERROR: Can't find a suitable configuration file in this directory
or any
parent. Are you in the right directory?

Supported filenames: docker-compose.yml, docker-compose.ya
ml, compose.yml, compose.yaml

parallels@ubuntu-linux-22-04 ~ % cd jenkins
parallels@ubuntu-linux-22-04 ~ % docker-compose
start
Starting jenkins ... done
parallels@ubuntu-linux-22-04 ~ %
```

# Jenkins



Ahora, como pueden ver, con la misma ip con la que no podíamos avanzar, ahora que ya está inicializado el servicio, lo podemos hacer.

Ingresamos las credenciales que definimos al inicio y listo.

**Sign in to Jenkins**

Username

Contraseña

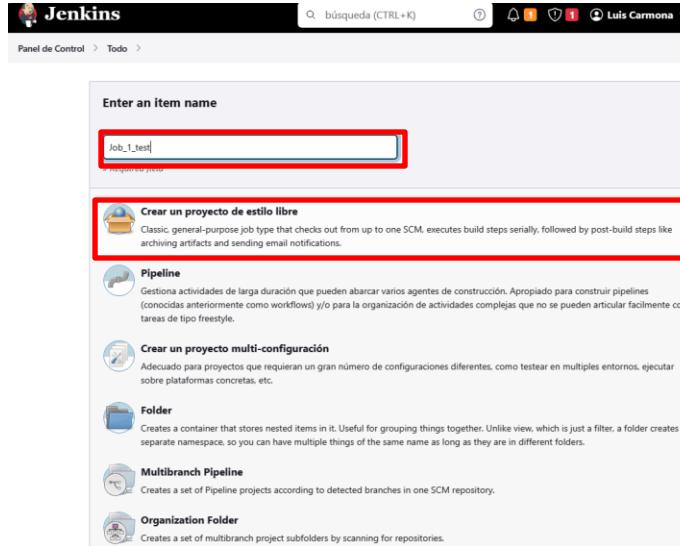
Keep me signed in

**Sign in**

# Jenkins

Vamos a crear nuestro primer Job. (estilo libre)

En nuestro explorador, con el servicio de Jenkins inicializado vamos a crearlo. Seleccionamos **Nueva tarea > asignamos nombre > Proyecto estilo libre**



Panel de Control >

+ Nueva Tarea

Personas

Historial de trabajos

Administrar Jenkins

Mis vistas

Enter an item name

Job\_1\_test

Crear un proyecto de estilo libre

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline

Gestiona actividades de larga duración que pueden abarcar varios agentes de construcción. Apto para construir pipelines (conocidas anteriormente como workflows) y/o para la organización de actividades complejas que no se pueden articular fácilmente con tareas de tipo freestyle.

Crear un proyecto multi-configuration

Adecuado para proyectos que requieren un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.

Folder

Crea un contenedor que almacena ítems anidados en él. Útil para agrupar cosas juntas. Diferente a view, que es solo un filtro, un folder crea un espacio de nombre separado, así que puedes tener múltiples cosas del mismo nombre tan largo como estén en diferentes carpetas.

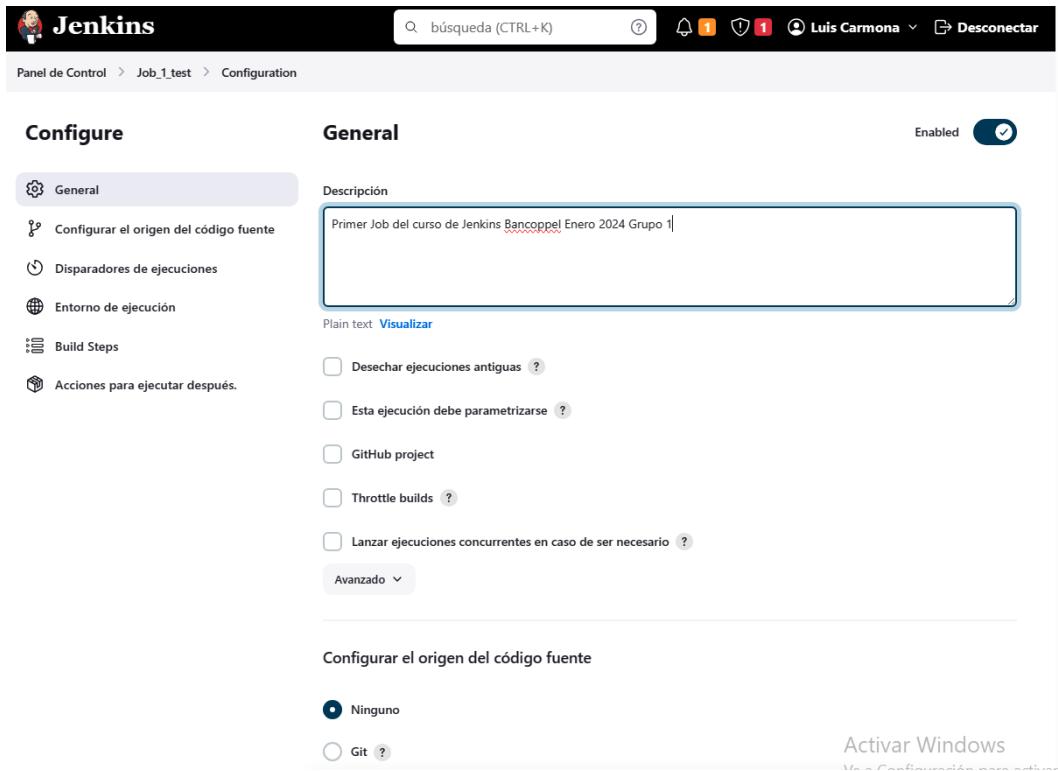
Multibranch Pipeline

Crea un conjunto de Pipeline projects según las ramas detectadas en uno SCM repository.

Organization Folder

Crea un conjunto de subcarpetas de proyecto multibranch escaneando por repositorios.

# Jenkins



The screenshot shows the Jenkins configuration interface for a job named "Job\_1\_test". The left sidebar lists several configuration sections: General, Configurar el origen del código fuente, Disparadores de ejecuciones, Entorno de ejecución, Build Steps, and Acciones para ejecutar después. The "General" section is currently selected.

**General** tab settings:

- Enabled:** A toggle switch is set to "Enabled" (indicated by a green checkmark).
- Description:** The description field contains the text: "Primer Job del curso de Jenkins Bancoppel Enero 2024 Grupo 1".
- Plain text:** A link labeled "Visualizar" is present.
- Build options:** Several checkboxes are available:
  - Deshacer ejecuciones antiguas
  - Esta ejecución debe parametrizarse
  - GitHub project
  - Throttle builds
  - Lanzar ejecuciones concurrentes en caso de ser necesario
- Advanced:** A "Avanzado" button with a dropdown arrow is located at the bottom of the build options section.

**Configurar el origen del código fuente** section:

- Ninguno:** This option is selected.
- Git:** An unselected radio button.

**Activar Windows** (Footer):

Más Configuración para este job

Descartar los "builds" previos, en función de los parámetros definidos.

Si el proyecto se debe parametrizar, por ejemplo, tomando como referencia un .JAR

Puedes vincular tu cuenta o proyecto de GitHub a Jenkins, con la URL.

Número de veces concurrentes que se ejecutara este Job en el servidor.

# Jenkins

Panel de Control &gt; Job\_1\_test &gt; Configuration

## Configure

### Configurar el origen del código fuente

#### General

#### Ninguno

#### Git

#### Disparadores de ejecuciones

#### Entorno de ejecución

#### Build Steps

#### Acciones para ejecutar después.

Sirve para elegir cuando se activa o ejecuta el "builid".

Esta opción es para cuando se necesita o quiere ejecutar remotamente.

Ejecutar el proceso cuando se cumpla la condición, por ejemplo, que uno previo haya terminado.

Ejecución con cierta frecuencia.

#### Disparadores de ejecuciones

- Lanzar ejecuciones remotas (ejem: desde 'scripts') ?
- Construir tras otros proyectos ?
- Ejecutar periódicamente ?
- GitHub hook trigger for GITScm polling ?
- Consultar repositorio (SCM) ?

#### Entorno de ejecución

- Delete workspace before build starts
- Use secret text(s) or file(s) ?
- Abortar la ejecución si se atasca
- Add timestamps to the Console Output
- Inspect build log for published build scans
- With Ant ?

# Jenkins

Ahora, en esta sección, seleccionamos ejecutar linea de comandos (shell) , vemos que tenemos una linea de comandos.

## Build Steps

Añadir un nuevo paso ^

Filter

- Ejecutar Ant
- Ejecutar linea de comandos (shell)**
- Ejecutar tareas 'maven' de nivel superior
- Ejecutar un comando de Windows
- Invoke Gradle script
- Run with timeout
- Set build status to "pending" on GitHub commit

En la linea de comandos, validamos que el servicio esté funcionando con el comando **docker ps**. Despues ingresamos al contenedor con el comando **docker exec -ti jenkins bash**, una vez dentro escribimos el siguiente comando echo "Hola jenkins".

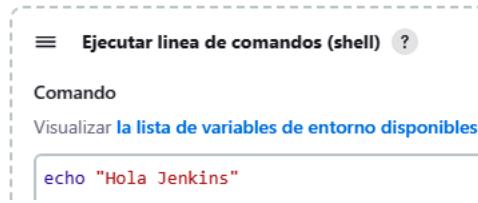
Vemos que imprime, **hola Jenkins**

```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS          NAMES
12632ad11607   jenkins/jenkins   "/usr/bin/tini -- /u..."   6 days ago    Up 5 hours   0.0.0.0:8080->8080/tcp,
                ::::8080->8080/tcp, 0.0.0.0:50000->50000/tcp, ::::50000->50000/tcp   jenkins
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker exec -ti jenkins bash
jenkins@12632ad11607:/$ echo "hola Jenkins"
hola Jenkins
jenkins@12632ad11607:/$ |
```

# Jenkins

Desde la interfaz de jenkins hagamos lo mismo, ejecutar **echo "hola jenkins"** y guardamos.

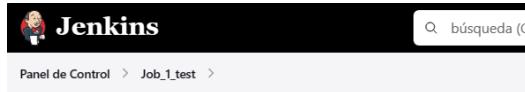
## Build Steps



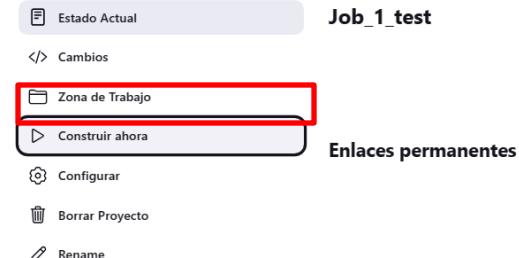
The screenshot shows the Jenkins build steps configuration for a job named 'Job\_1\_test'. It includes a section for 'Ejecutar linea de comandos (shell)' containing the command 'echo "Hola Jenkins"'.

```
echo "Hola Jenkins"
```

En la opción construir ahora, le damos clic. Aparece en la parte inferior el número de ejecucion del job.



The screenshot shows the Jenkins job configuration page for 'Job\_1\_test'. The 'Zona de Trabajo' and 'Construir ahora' buttons are highlighted with red boxes.



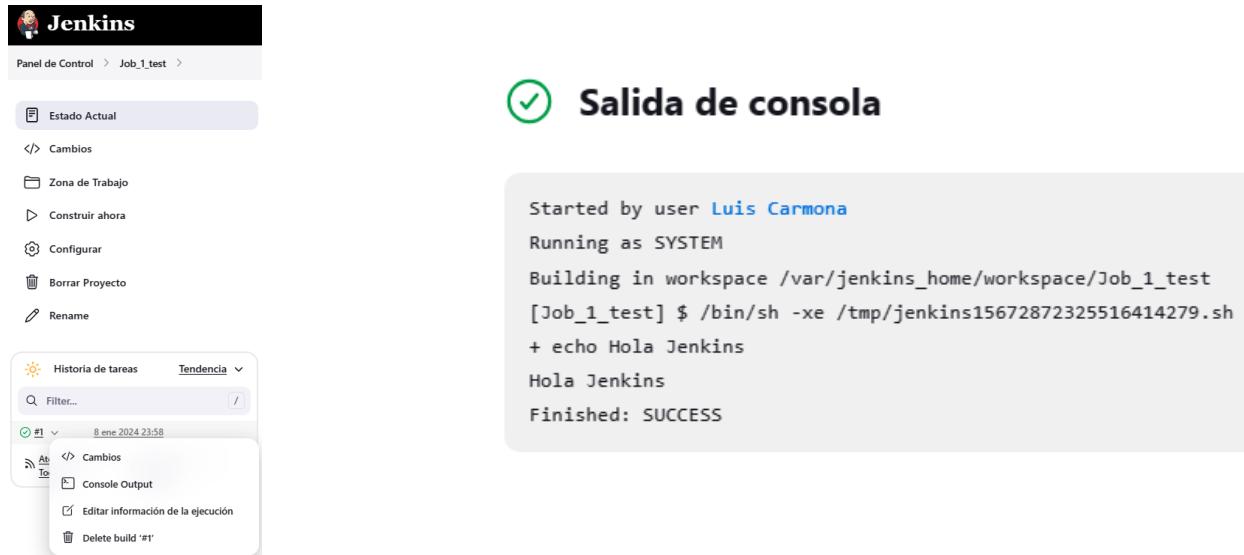
The screenshot shows the Jenkins job configuration page for 'Job\_1\_test'. The 'Enlaces permanentes' section is visible, and the 'Historia de tareas' section at the bottom shows a single build entry with the number '#1'.



The screenshot shows the Jenkins build history for job '#1'. It includes a 'Filter...' search bar and links for 'Atom feed Para Todos' and 'Atom feed para los errores'.

# Jenkins

Ahora, seleccionamos **"console output"** y vemos los detalles del job, de esta manera tenemos nuestro primer Job.



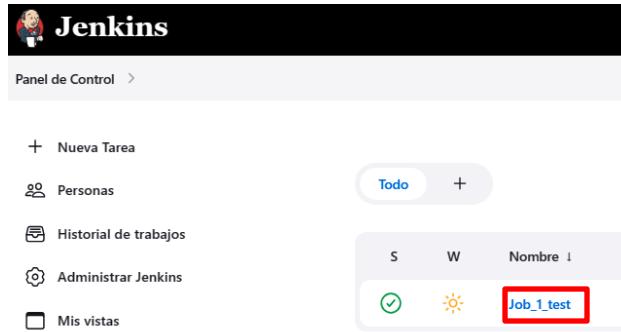
The screenshot shows the Jenkins interface for the 'Job\_1\_test' job. On the left, there's a sidebar with options like 'Estado Actual', 'Cambios', 'Zona de Trabajo', 'Construir ahora', 'Configurar', 'Borrar Proyecto', and 'Rename'. Below that is a 'Historia de tareas' section showing a single build (#1) from '8 ene 2024 23:58'. A context menu is open over this build, with options 'Cambiros', 'Console Output' (which is checked), 'Editar información de la ejecución', and 'Delete build #1'. The main content area is titled 'Salida de consola' and displays the following log output:

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/sh -xe /tmp/jenkins15672872325516414279.sh
+ echo Hola Jenkins
Hola Jenkins
Finished: SUCCESS
```

# Jenkins

En el ejercicio anterior, creamos un Job sencillo, que imprime Hola jenkins, es momento de hacerlo más “robusto”, parametrizandolo.

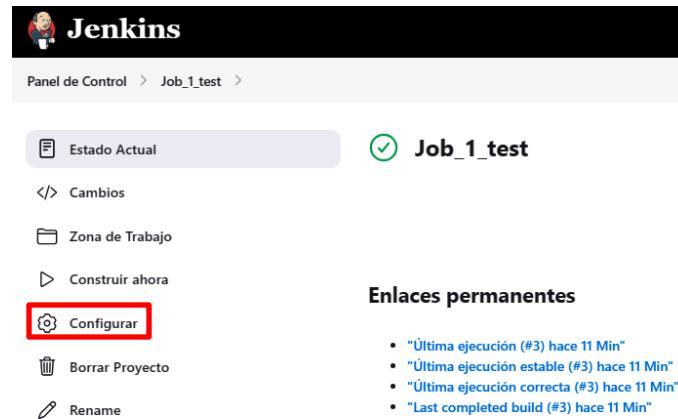
Ingresamos a nuestro Job, desde el **panel de control > configurar**



The screenshot shows the Jenkins dashboard with the following interface elements:

- Header: Jenkins
- Breadcrumbs: Panel de Control >
- Left sidebar:
  - + Nueva Tarea
  - Personas
  - Historial de trabajos
  - Administrar Jenkins
  - Mis vistas
- Top navigation buttons: Todo, +
- Job list table:

S	W	Nombre
✓	☀	Job_1_test

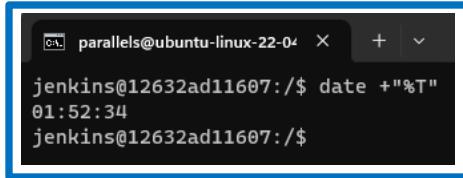


The screenshot shows the Jenkins configuration page for the 'Job\_1\_test' job. The page includes:

- Header: Jenkins
- Breadcrumbs: Panel de Control > Job\_1\_test >
- Job status: Estado Actual (green checkmark)
- Job name: Job\_1\_test
- Job actions:
  - Cambios
  - Zona de Trabajo
  - Construir ahora
  - Configurar (highlighted with a red box)
  - Borrar Proyecto
  - Rename
- Permanent links:
  - Última ejecución (#3) hace 11 Min
  - Última ejecución estable (#3) hace 11 Min
  - Última ejecución correcta (#3) hace 11 Min
  - Last completed build (#3) hace 11 Min

# Jenkins

Ahora, desde la consola, ingresamos (si no estamos en él) al contenedor, una vez dentro ejecutamos el siguiente código **date +"%T"** mismo que nos da la hora actual.



```
parallels@ubuntu-linux-22-04 ~ % jenkins@12632ad11607:/$ date +"%T" 01:52:34 jenkins@12632ad11607:/$
```



Panel de Control > Job\_1\_test >

Estado Actual

Cambios

Zona de Trabajo

Construir ahora

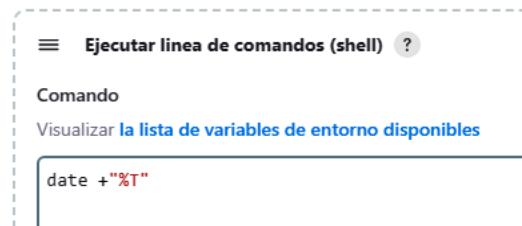
Configurar

Borrar Proyecto

Rename

Ahora, hacemos lo mismo en **shell de jenkins > Construir ahora**

## Build Steps



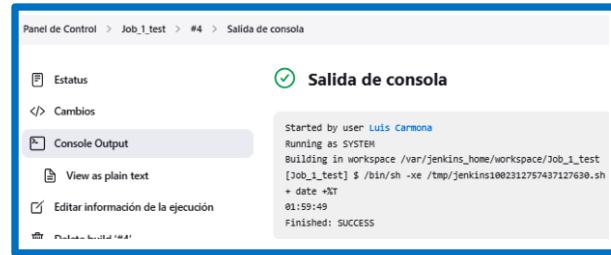
Ejecutar linea de comandos (shell)

Comando

Visualizar la lista de variables de entorno disponibles

```
date +"%T"
```

%T= time as %H:%M:%S



Panel de Control > Job\_1\_test > #4 > Salida de consola

Estatus

Cambios

Console Output

View as plain text

Editar información de la ejecución

Salida de consola

Started by user Luis Carmona  
Running as SYSTEM  
Building in workspace /var/jenkins\_home/workspace/Job\_1\_test  
[Job\_1\_test] \$ /bin/sh -xe /tmp/jenkins1002312757437127630.sh  
+ date +%T  
01:59:49  
Finished: SUCCESS



Historia de tareas Tendencia

Filter...

#	Fecha
4	9 ene 2024 1:59
3	9 ene 2024 0:48
2	9 ene 2024 0:29
1	8 ene 2024 23:58

Atom feed Para Todos Atom feed para los errores

# Jenkins

Ejecutaremos algo similar, pero con variables de entorno. Así que nos vamos a la interfaz de jenkins y en la opcion del shell realizamos lo siguiente:

```
AHORA=$(date + "%r")
```

```
echo "la hora actual es: $AHORA" > /tmp/ahora
```

Guardamos y construimos nuevamente.

Build Steps

Ejecutar linea de comandos (shell)

```
Visualizar la lista de variables de entorno disponibles
AHORA=$(date + "%r")
echo "la hora actual es: $AHORA" > /tmp/ahora
```

Avanzado ▾

Añadir un nuevo paso ▾

Acciones para ejecutar después.

Añadir una acción ▾

Guardar Apply

Panel de Control > Job\_1\_test >

Estado Actual

Cambios

Zona de Trabajo

Construir ahora

Configurar

Borrar Proyecto

Rename

Historia de tareas Tendencia ▾

Panel de Control > Job\_1\_test > #7 > Salida de consola

Estatus

Cambios

Console Output

View as plain text

Editar información de la ejecución

Delete build '#7'

← Ejecución previa

Salida de consola

Started by user Luis Carmona  
Running as SYSTEM  
Building in workspace /var/jenkins\_home/workspace/Job\_1\_test  
[Job\_1\_test] \$ /bin/sh -xe /tmp/jenkins4852178565020601819.sh  
+ date +%r  
+ AHORA=11:17:34 PM  
+ echo la hora actual es: 11:17:34 PM  
Finished: SUCCESS

# Jenkins

Desde el panel de control, abrimos nuevamente nuestro job, lo empezamos a configurar, cambiamos la descripción y en la sección de shell ponemos el siguiente código, guardamos y construimos...

```
#!/bin/bash #linea shebang, sirve para especificar el interprete, en este caso bash
nombre= "Luis" #declaramos la variable nombre y su valor

for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 #inicializamos un bucle, for en este caso
do #ejecutar todo lo que este dentro de do hasta done
    if[$a ==8] #si a es igual a 8
        then #
            sleep 10 #una pausa de 10 seg para ejecutar la siguiente linea
            echo " Es hora de descansar de la clase de $nombre"
        fi
        echo "clase numero $a"
done
```

# Jenkins

Panel de Control > Job\_1\_test > #27 > Salida de consola

- Estatus
- Cambios
- Console Output
- View as plain text
- Editar información de la ejecución
- Delete build '#27'
- Ejecución previa

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/bash /tmp/jenkins8261815402499503040.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
Finished: SUCCESS
```

```
#!/bin/bash
nombre="Luis"
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do

    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
```

# Jenkins

Podemos agregar tantas variables y elementos que queramos o necesitemos, vamos a configurar nuevamente nuestro “job” agregando tres líneas de código. (**subrayadas**)

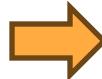
```
#!/bin/bash
nombre="Luis"
curso="jenkins"
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
sleep 10
echo "muy bien compañero $nombre, las clases de $curso terminaron."
```

# Jenkins

Construimos el job con el nuevo, código y vemos la salida por consola.

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/bash /tmp/jenkins14375496436975303905.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
```



## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/bash /tmp/jenkins14375496436975303905.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de jenkins terminaron
Finished: SUCCESS
```



# Jenkins

Ahora, vamos a ejecutar el mismo script del ejemplo anterior, pero ahora lo haremos desde el contenedor.

Entramos a nuestro job en jenkins y copiamos el codigo que dejamos en la última construccion o build.

## Build Steps

```
≡ Ejecutar linea de comandos (shell) ?  
  
Comando  
Visualizar la lista de variables de entorno disponibles  
  
#!/bin/bash  
nombre="Luis"  
curso="jenkins"  
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
do  
    if [ $a == 8 ]  
    then  
        sleep 10  
        echo " a descansar de clase $nombre"  
    fi  
    echo "clase numero $a"  
done  
sleep 10  
echo "muy bien compañero $nombre, las clases de $curso terminaron."
```

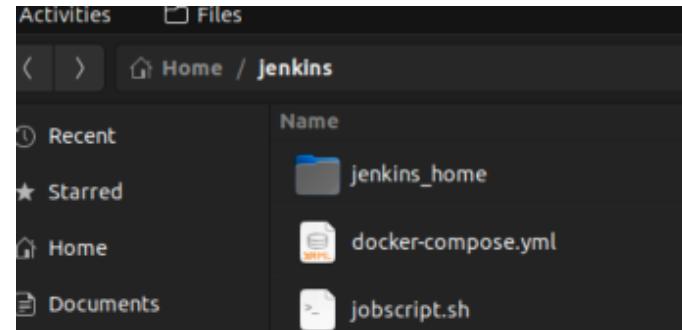
# Jenkins

Ahora, desde la terminal donde hemos estado trabajando, ejecutamos el siguiente comando **vi jobsript.sh** que abrirá el editor de texto vim de ubuntu creando el documento jobsript.sh. **Recuerden apretar "i" para iniciar la edición del documento.**

## Importante recordar que debemos estar en la carpeta jenkins

```
parallels@ubuntu-linux-22-04 ~ + v
#!/bin/bash
nombre="Luis"
curso="jenkins"
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
sleep 10
echo "muy bien compañero $nombre, las clases de $curso terminaron."|
```

Apretamos esc para salir y finalmente :wq! Para guardar los cambios en el archivo... podemos ver en ubuntu que el archivo se generó.



# Jenkins

Ejecutamos el script con el comando `./jobscript.sh...`

Vemos que el resultado es el mismo que desde el shell de jenkins, como se ve en las imágenes.

```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ ./jobscript.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de jenkins terminaron.
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ |
```

✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/bash /tmp/jenkins12092045924565824174.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de jenkins terminaron.
Finished: SUCCESS
```

# Jenkins

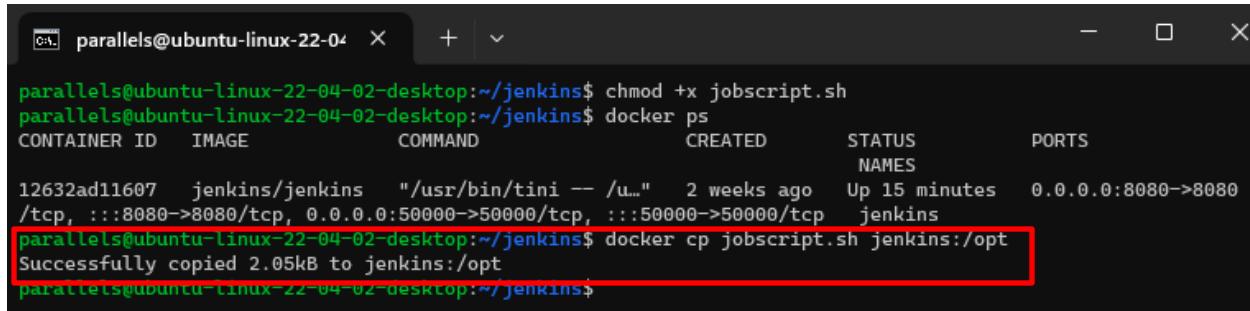
Observando lo que hemos hecho, vemos que estamos solo en la carpeta de jenkins, pero queremos hacerlo dentro del contenedor, para ello necesitamos posicionarnos dentro del él y paralelamente brindarle los permisos necesarios de ejecución al script. Para los permisos, ejecutamos **chmod +x jobscript.sh**

Después, ejecutamos Docker ps, para ver los contenedores ejecutándose, vemos que está el de jenkins (**12632ad11607**), que es donde está el File System del servidor de jenkins.

```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ chmod +x jobscript.sh
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS      PORTS          NAMES
12632ad11607   jenkins/jenkins   "/usr/bin/tini -- /u..."   2 weeks ago   Up 15 minutes   0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000/tcp, 0.0.0.0:50000->50000/tcp   jenkins
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$
```

# Jenkins

Ahora con el comando **docker cp jobsctipt.sh jenkins:/opt** donde estamos ejecutando que copie el archivo jobsctipt.sh en la ruta jenkins/opt... elejimos opt, dado que son archivos opcionales...



```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ chmod +x jobsctipt.sh
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED      STATUS      PORTS
12632ad11607   jenkins/jenkins  "/usr/bin/tini -- /u..."  2 weeks ago  Up 15 minutes  0.0.0.0:8080->8080
/tcp, :::8080->8080/tcp, 0.0.0.0:50000->50000/tcp, :::50000->50000/tcp   jenkins
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker cp jobsctipt.sh jenkins:/opt
Successfully copied 2.05kB to jenkins:/opt
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$
```

## Jenkins

Ingresamos al contenedor....

**¿Recuerdan el comando para hacerlo?**

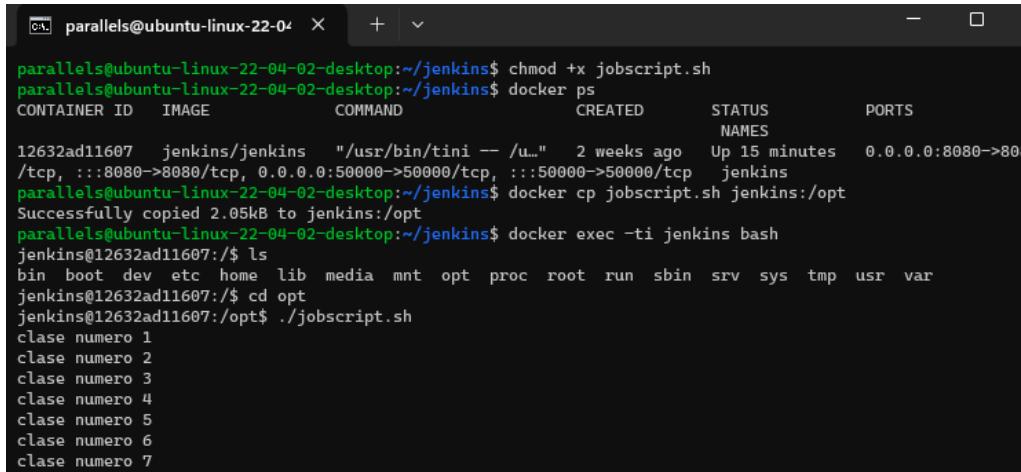
# Jenkins

Ingresamos al contenedor con el comando **docker exec -ti jenkins bash**

Y después **ls** para listar los directorios/carpetas en la ruta y confirmar que se creó o movió a **opt** el archivo **jobscrip.sh**, para ello nos posicionamos en la carpeta opt con el comando **cd opt**

Ahora, ejecutamos el mismo comando que en las anteriores láminas, para el archivo **jobscrip.sh**

**./jobscrip.sh**



```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ chmod +x jobscrip.sh
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED     STATUS      PORTS
NAMES
12632ad11607   jenkins/jenkins   "/usr/bin/tini -- /u..."   2 weeks ago   Up 15 minutes   0.0.0.0:8080->8080
/tcp, :::8080->8080/tcp, 0.0.0.0:50000->50000/tcp, :::50000->50000/tcp   jenkins
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker cp jobscrip.sh jenkins:/opt
Successfully copied 2.05kB to jenkins:/opt
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker exec -ti jenkins bash
jenkins@12632ad11607:~$ ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
jenkins@12632ad11607:~$ cd opt
jenkins@12632ad11607:/opt$ ./jobscrip.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
```

# Jenkins

Ahora hagamos lo mismo, pero desde el shell de jenkins para ver las diferencias, abrimos el navegador con la GUI de Jenkins, y entramos en la opción de configuración para ir al shell y poner la ruta donde guardamos o movimos el archivo **jobsript.sh**. **(opt)** guardamos y construimos de nuevo el proyecto.

## Build Steps

Ejecutar linea de comandos (shell) ?

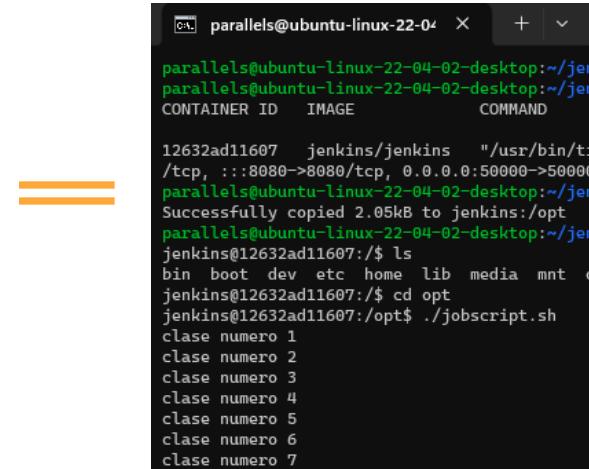
Comando

Visualizar la lista de variables de entorno disponibles

```
/opt/jobsript.sh
```

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/sh -xe /tmp/jenkins12227349028274116612.sh
+ /opt/jobsript.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de jenkins terminaron.
Finished: SUCCESS
```



```
parallels@ubuntu-linux-22-04:~$ ./jobsript.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
```

# Jenkins

Volvemos a la terminal para revisar y ver gráficamente lo siguiente. Realizaremos algunas modificaciones a script (jobscrip.sh) para ello ejecutamos el comando **vi jobscrip.sh** (ojo, debemos estar "fuera del contenedor")

Eliminamos las variables para ponerlas directamente en jenkins. (recuerden poner :wq! al final para que los cambios se guarden).

```
parallels@ubuntu-linux-22-04 ~ + -  
#!/bin/bash  
  
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
do  
    if [ $a == 8 ]  
    then  
        sleep 10  
        echo " a descansar de clase $nombre"  
    fi  
    echo "clase numero $a"  
done  
sleep 10  
echo "muy bien compañero $nombre, las clases de $curso terminaron.  
~  
~  
~  
~  
~  
~  
~
```

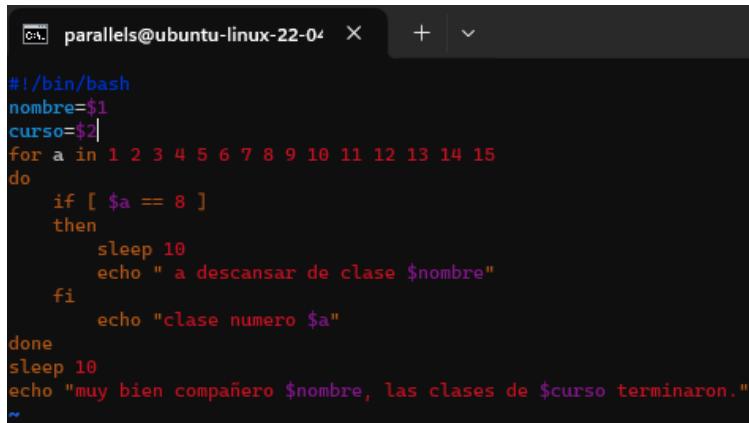
# Jenkins

Una vez modificado, al igual que en los pasos anteriores, debemos “moverlo” al contenedor con el comando **docker cp jobsript.sh jenkins:/opt** ingresamos al cotenedor , nos posicionamos en la ruta adecuada y ejecutamos, para ver qué pasa.

```
parallels@ubuntu-linux-22-04 ~]$ docker cp jobsript.sh jenkins:/opt
Successfully copied 2.05kB to jenkins:/opt
parallels@ubuntu-linux-22-04 ~]$ docker exec -ti jenkins bash
jenkins@12632ad11607:~$ cd opt
jenkins@12632ad11607:/opt$ ./jobsript.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien companero , las clases de terminaron.
jenkins@12632ad11607:/opt$ |
```

# Jenkins

Pero entonces que ¿pasaría o tendría que hacer si yo quisiera, por ejemplo, desde jenkins modificar o cambiar las variables y que estos cambios apliquen en el script?, para ello tenemos que ejecutar algunos cambios en el script (jobscrip.sh), por lo tanto, ejecutamos **vi jobscrip.sh y agregamos nombre=\$1 curso=\$2**



```
#!/bin/bash
nombre=$1
curso=$2
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
sleep 10
echo "muy bien compañero $nombre, las clases de $curso terminaron."
~
```

Con esto, le indicamos que nombre será el primer argumento y curso el segundo.

# Jenkins

Como en los anteriores pasos, muevo el archivo hacia la carpeta opt, con el siguiente comando  
**docker cp jobscript.sh jenkins:/opt**

```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ docker cp jobscript.sh jenkins:/opt
Successfully copied 2.05kB to jenkins:/opt
```

En la shell de jenkins realizamos los siguientes cambios.

Ejecutar linea de comandos (shell) ?

Comando

Visualizar [la lista de variables de entorno disponibles](#)

```
nombre="Luis"
curso="jenkins"
/opt/jobscript.sh $nombre $curso
```

Construimos...

✓ Salida de consola

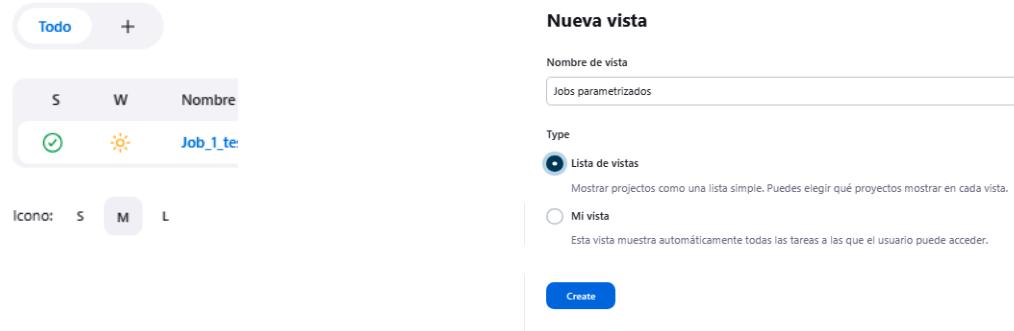
```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job_1_test
[Job_1_test] $ /bin/sh -xe /tmp/jenkins13058987322833748599.sh
+ nombre=Luis
+ curso=jenkins
+ /opt/jobscript.sh Luis jenkins
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de jenkins terminaron.
Finished: SUCCESS
```

# Jenkins

## Vamos a parametrizar Jenkins

Para ello, vamos a crear una nueva vista en la GUI de jenkins, esto con el fin de tener un control más organizado, para ello generamos una nueva vista en el panel de control.

+ > Nombre de la vista > todo lo demás lo dejamos en el valor por defecto > ok



The screenshot shows the Jenkins 'New View' configuration page. On the left, there's a sidebar with tabs 'Todo' (selected) and '+'. Below it are icons for 'S' (Sticky), 'W' (Windowed), and 'Nombre' (Name). Under 'Nombre', 'Job\_1\_te' is selected with a checked checkbox. At the bottom of the sidebar, there are buttons for 'Icono:' followed by 'S' (selected), 'M', and 'L'. The main right-hand panel has a title 'Nueva vista'. It contains fields for 'Nombre de vista' (set to 'Jobs parametrizados') and 'Type' (set to 'Lista de vistas'). A note below says 'Mostrar proyectos como una lista simple. Puedes elegir qué proyectos mostrar en cada vista.' There are two radio button options: 'Lista de vistas' (selected) and 'Mi vista'. A note for 'Mi vista' says 'Esta vista muestra automáticamente todas las tareas a las que el usuario puede acceder.' At the bottom right is a blue 'Create' button.

# Jenkins

## Vamos a parametrizar los jobs de Jenkins con cadenas

Creamos nuestro segundo job (se sugiere el nombre de job2\_parametrizacion), igual en un proyecto libre y vamos a seleccionar algunas de las casillas que no revisamos en el primer job.

Volvemos a usar el shell para su construcción.

Desechar ejecuciones antiguas  ?

Strategy

Log Rotation

Número de días para mantener ejecuciones de proyectos  
si no está vacío, sólo se mantendrán las ejecuciones con una edad inferior a este número de días  
2

Número máximo de ejecuciones para guardar  
si no está vacío, sólo se guardarán un número de ejecuciones inferior a este valor  
10

Build Steps

Ejecutar línea de comandos (shell)  ?

Comando

Visualizar [la lista de variables de entorno disponibles](#)

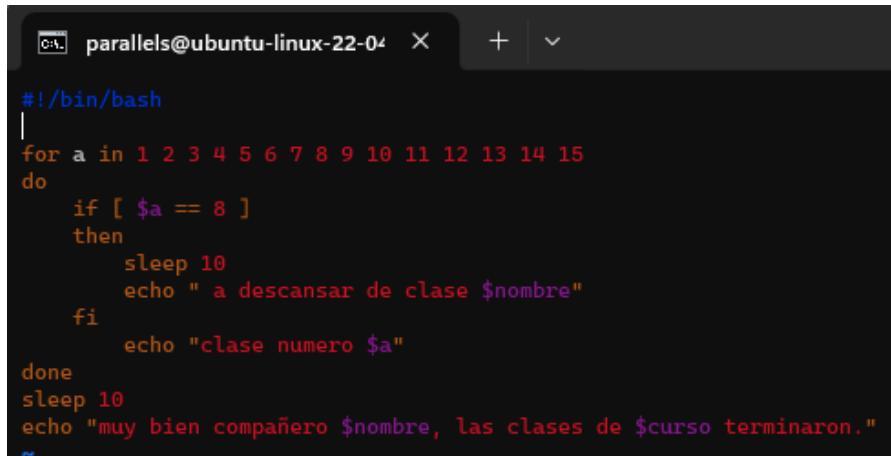
Guardar  Apply

Avanzado

Activar Wir

# Jenkins

Antes de continuar, debemos revisar el script (jobscrip.sh) para confirmar que todos tenemos el mismo. De no ser asi, deben modificarlo para que quede de la siguiente manera (recuerden que si lo modifican desde "fuera" del contenedor es necesario "moverlo al contenedor" con **docker cp jobscrip.sh jenkins:/opt**)

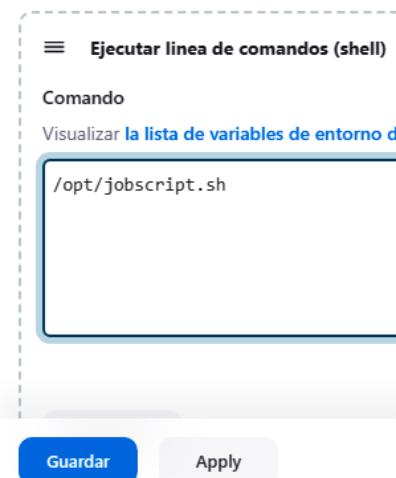


```
parallels@ubuntu-linux-22-04 ~ % ./jobscrip.sh
#!/bin/bash
|
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
sleep 10
echo "muy bien companero $nombre, las clases de $curso terminaron."
~
```

# Jenkins

Ahora, en el shell de jenkins ponemos la ruta donde se encuentra el script

## Build Steps



The screenshot shows the Jenkins configuration interface for a build step. It is a 'Execute shell' step. The 'Command' field contains the path to a script: '/opt/jobsript.sh'. There are two buttons at the bottom: 'Guardar' (Save) and 'Apply'.

```
/opt/jobsript.sh
```

Guardar Apply

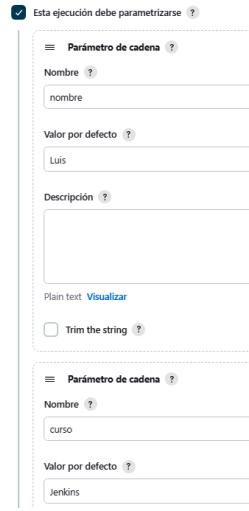
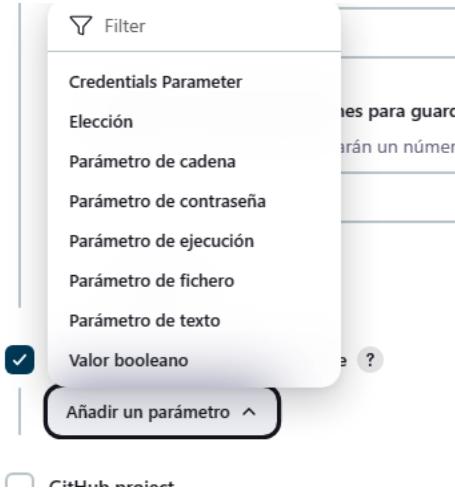
Pero como se dieron cuenta en la lámina anterior, no tenemos las variables definidas (nombre y curso) que vamos a definirlas en este job de una forma diferente...

# Jenkins

Dentro de jenkins nos vamos a la sección “esta ejecución debe parametrizarse”

## Añadir parametro > parámetro de cadena

Agregamos los dos parametros que requiere el job (nombre y curso) ambos de cadena.



The screenshot shows the Jenkins configuration interface for a job. It displays two 'String' parameters. The first parameter, 'nombre', has a 'Nombre' field containing 'nombre', a 'Valor por defecto' field containing 'Luis', and a 'Descripción' field. The second parameter, 'curso', has a 'Nombre' field containing 'curso', a 'Valor por defecto' field containing 'Jenkins', and a 'Descripción' field. There are also 'Plain text' and 'Visualizar' buttons, and a 'Trim the string' checkbox.

# Jenkins

Ahora, haremos el mismo proceso que la lámina anterior agregando el script, directamente en el shell de jenkins.

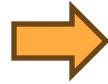
Para ello copiamos el código del script y lo pegamos en el shell de jenkins, guardamos y construimos.

## Ejecutar linea de comandos (shell) ?

### Comando

Visualizar [la lista de variables de entorno disponibles](#)

```
#!/bin/bash
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
sleep 10
echo "muy bien compañero $nombre, las clases de $curso terminaron."
```



## Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/job2_parametrizacion
[job2_parametrizacion] $ /bin/bash /tmp/jenkins9578461125856530517.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de Jenkins terminaron.
Finished: SUCCESS
```

# Jenkins

Vamos a parametrizar los Jenkins jobs con parametros de elección.

Nos dirigimos a nuestro segundo Job y configuramos. Donde vamos a cambiar el parámetro de cadena curso, a parámetro de elección y agregaremos la variable idioma.

Para el segundo caso, vamos a modificar la última linea echo por la siguiente:

**echo "muy bien compañero \$nombre, las clases de \$curso en idioma \$idioma terminaron."**

```
#!/bin/bash
for a in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
    if [ $a == 8 ]
    then
        sleep 10
        echo " a descansar de clase $nombre"
    fi
    echo "clase numero $a"
done
sleep 10
echo "muy bien compañero $nombre, las clases de $curso en idioma $idioma terminaron."
```

# Jenkins

Nos desplazamos hacia la sección donde están definidos los parámetros de cadena, y borramos el parámetro de curso, dado que, como comentamos al inicio éste lo modificaremos por uno de elección.



The screenshot shows the Jenkins Parameter Definition screen. A modal dialog box titled 'Dutton' is open, containing a red 'X' button in the top right corner. The main area displays a single parameter entry: 'Nombre' with the value 'curso'. This indicates that the 'curso' parameter is being deleted or modified.

Después, agregamos curso, pero como parámetro de elección y no de cadena, como estaba anteriormente. Y en la sección de opciones, agregamos las opciones que tendrá el parámetro.



The screenshot shows the Jenkins Parameter Definition screen again. It includes three sections: 'Parámetro de cadena' (String Parameter) which has been deleted; 'Elección' (Choice Parameter) where 'Nombre' is set to 'curso'; and 'Opciones' (Options) where 'Python' and 'Jenkins' are listed as choices. A red callout at the bottom left states 'Requires Choices.'.

# Jenkins

Ahora, definimos otra variable de elección, para la nueva que agregamos en el shell (idioma). De la misma forma que acabamos de agregar la de curso.

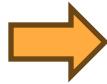
Guardamos y construimos con parámetros

**Eleción ?**

Nombre ?  
idioma

Opciones ?  
Español  
Ingles  
Aleman  
Frances

! Requires Choices.



## Proyecto job2\_parametrizacion

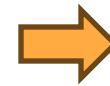
Esta ejecución requiere parámetros adicionales:

nombre  
Luis

curso  
Python

idioma  
Python  
Jenkins

Español



## Proyecto job2\_parametrizacion

Esta ejecución requiere parámetros adicionales:

nombre  
Luis

curso  
Python

idioma  
Español

Español  
Ingles  
Aleman  
Frances

# Jenkins

Construimos varias veces, seleccionando las diferentes opciones para ver su salida por consola.

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/job2_parametrizacion
[job2_parametrizacion] $ /bin/bash /tmp/jenkins1057538646176898312.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de Python en idioma Español terminaron.
Finished: SUCCESS
```

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/job2_parametrizacion
[job2_parametrizacion] $ /bin/bash /tmp/jenkins1012169124123103659.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de Python en idioma Aleman terminaron.
Finished: SUCCESS
```

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/job2_parametrizacion
[job2_parametrizacion] $ /bin/bash /tmp/jenkins559399339040183645.sh
clase numero 1
clase numero 2
clase numero 3
clase numero 4
clase numero 5
clase numero 6
clase numero 7
    a descansar de clase Luis
clase numero 8
clase numero 9
clase numero 10
clase numero 11
clase numero 12
clase numero 13
clase numero 14
clase numero 15
muy bien compañero Luis, las clases de Jenkins en idioma Frances terminaron.
Finished: SUCCESS
```

# Jenkins

Ahora, veremos un parametro más... el booleano. Para ello, creamos un nuevo Job y en la sección del shell agregamos el siguiente código.

## Build Steps

```
≡ Ejecutar linea de comandos (shell) ?  
  
Comando  
Visualizar la lista de variables de entorno disponibles  
  
#!/bin/bash  
echo "Bienvenidos al planeta $planeta"  
if [ "$agente" = "true" ]  
then  
    echo "Prepárese para la misión agente"  
else  
    echo "Disfruta el planeta $nombre."  
fi  
echo "..."  
sleep 5  
echo "Suerte!"
```

Guardamos y listo.

Ahora tenemos que definir los 3 parámetros que tenemos.

1. Planeta
2. agente
3. Nombre

¿Cuál consideran será o serán los booleanos?

# Jenkins

✓ Esta ejecución debe parametrizarse ?

☰ Parámetro de cadena ?

Nombre ?

Valor por defecto ?

☰ Elección ?

Nombre ?

Opciones ?

- Mercurio
- Venus
- Marte
- Jupiter
- Saturno

☰ Valor booleano ?

Nombre ?

Valor por defecto ?



Si no se selecciona valor por defecto, queda en valor falso, si se selecciona, el valor es verdadero.

# Jenkins

Construimos y veamos los posibles escenarios...

## Proyecto Job3\_Parametro\_booleano

Esta ejecución requiere parámetros adicionales:

nombre

planeta

agente

 Ejecución Cancel

¿se seleccionó la casilla agente?

### Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job3_Parametro_booleano
[job3_Parametro_booleano] $ /bin/bash /tmp/jenkins16279736977818433475.sh
Bienvenidos al planeta Marte
Disfruta el planeta Luis.
...
Suerte!
Finished: SUCCESS
```

¿se seleccionó la casilla agente?

### Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job3_Parametro_booleano
[job3_Parametro_booleano] $ /bin/bash /tmp/jenkins14177579478007991793.sh
Bienvenidos al planeta Jupiter
Prepárese para la misión agente
...
Suerte!
Finished: SUCCESS
```

# Jenkins

## Ejercicio 1:

Desde Jenkins GUI, crear un job que una vez que lo construyas o compiles, desde la salida de consola nos muestre como resultado, lo siguiente:

“ La hora actual es: 9:10 am”

“ La fecha de es: DD/MM/AAAA”

Después ejecutar lo mismo, pero desde un script.

---

# Jenkins

## Ejercicio 2:

Genere un script (job2script.sh) con el código necesario para que imprima ademas de los números del 2 al 20 de 2 en 2 (2,4,6... 20) y la siguiente frase:

**“hola soy nombre, me gusta el beisbol y le voy a equipo”**

Ejemplo: hola soy Luis, me gusta el beisbol y le voy a los tomateros.

Considere que las variables no estén en el script...

En una nueva construccion que ejecute lo mismo, pero ahora desde la GUI de Jenkins.

---

## Preparación del ambiente.

### Instalación de Docker compose

Es necesario, instalar tambien Docker Compose con el comando **sudo apt install docker-compose**

# Preparación del ambiente.

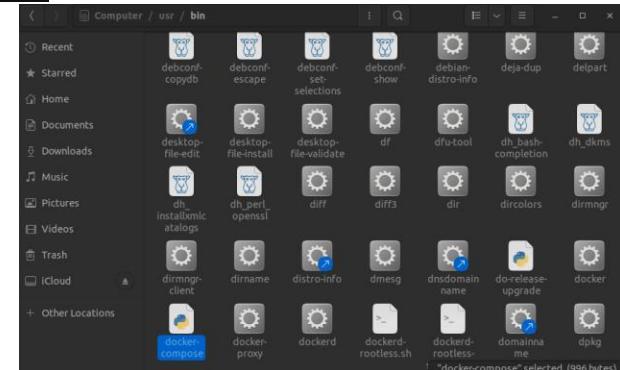
## Instalación de Docker compose

Finalmente instalaremos los permisos ejecutables al binario con el comando **sudo chmod +x /usr/bin/docker-compose**

```
parallels@ubuntu-linux-22-04-02-desktop: ~
parallels@ubuntu-linux-22-04-02-desktop:~$ sudo chmod +x /usr/local/bin/docker-compose
chmod: cannot access '/usr/local/bin/docker-compose': No such file or directory
parallels@ubuntu-linux-22-04-02-desktop:~$ sudo chmod +x /usr/bin/docker-compose
parallels@ubuntu-linux-22-04-02-desktop:~$ -
```

Si marca un error, es necesario encontrar la ruta adecuada de docker-compose para ello pueden ir al GUI y buscarlo.

**Other locations > usr > search (docker-compose) > properties**  
Para saber la ruta.



# Preparación del ambiente.

## Instalación de Docker compose

Finalmente, con el comando **docker-compose --version** confirmamos la version de docker-compose instalada... si aparece una version menor a la 2, como en la imagen.

```
parallels@ubuntu-linux-22-04-02-desktop: ~
parallels@ubuntu-linux-22-04-02-desktop:~$ docker-compose --version
docker-compose version 1.29.2, build unknown
parallels@ubuntu-linux-22-04-02-desktop:~$
```

Debemos ejecutar el siguiente código. **sudo apt-get update**, despues **sudo apt-get install docker-compose-plugin** y finalmente confirmamos con **docker compose version**

```
Seleccionar parallels@ubuntu-linux-22-04-02-desktop: ~
parallels@ubuntu-linux-22-04-02-desktop:~$ docker-compose --version
docker-compose version 1.29.2, build unknown
parallels@ubuntu-linux-22-04-02-desktop:~$ sudo apt-get update
[sudo] password for parallels:
Hit:1 https://download.docker.com/linux/ubuntu jammy InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports jammy InRelease
Hit:3 http://ports.ubuntu.com/ubuntu-ports jammy-updates InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports jammy-backports InRelease
Hit:5 http://ports.ubuntu.com/ubuntu-ports jammy-security InRelease
Reading package lists... Done
parallels@ubuntu-linux-22-04-02-desktop:~$ sudo apt-get install docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-compose-plugin is already the newest version (2.21.0-1~ubuntu.22.04~jammy).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
parallels@ubuntu-linux-22-04-02-desktop:~$ docker compose version
Docker Compose version v2.21.0
```

# Preparación del ambiente.

## Instalación de Docker compose

Si en algun momento se genera el mensaje de un kernel más reciente o que algunos servicios deben de ser actualizados, se puede resolver con el siguiente comando.

```
Newer kernel available

The currently running kernel version is 5.15.0-76-generic which is not the expected kernel version 5.15.0-91-generic.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

M

Restarting services...
Daemons using outdated libraries
-----
1. accounts-daemon.service 14. multipathd.service      27. systemd-manager
2. avahi-daemon.service    15. networkd-dispatcher.service 28. systemd-oomd.service
3. colord.service          16. NetworkManager.service   29. systemd-resolved.service
4. containererd.service    17. packagekit.service       30. systemd-timesyncd.service
5. cron.service            18. polkit.service        31. systemd-udevd.service
6. cups-browsed.service    19. power-profiles-daemon.service 32. udisks2.service
7. cups.service             20. prltoolsd.service     33. unattended-upgrades.service
8. dbus.service             21. rtkit-daemon.service   34. upower.service
9. docker.service           22. snapd.service        35. user@1000.service
10. fwupd.service          23. ssh.service          36. wpa_supplicant.service
11. gdm.service             24. switcheroo-control.service 37. none of the above
12. kerneloops.service     25. systemd-journald.service
13. ModemManager.service    26. systemd-logind.service

(Enter the items or ranges you want to select, separated by spaces.)

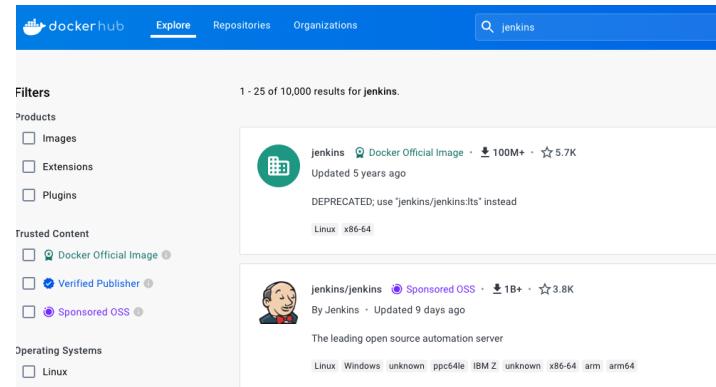
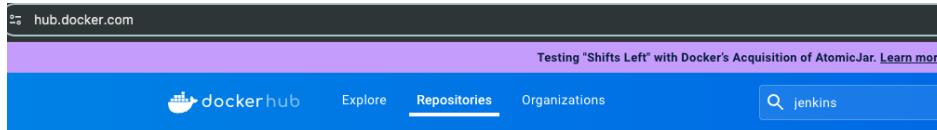
which services should be restarted? _
```

**sudo apt-get update && sudo apt-get full-upgrade -y**

# Preparación del ambiente.

## Instalación de Jenkins

Vamos a al hub de Jenkins, a través del siguiente link <https://hub.docker.com/> y buscamos Jenkins, como se muestra en la imagen.



1 - 25 of 10,000 results for jenkins.

**jenkins/Docker Official Image** · Docker Official Image · 100M+ · 5.7K · Updated 5 years ago · DEPRECATED; use "jenkins/jenkins:its" instead · Linux · x86-64

**jenkins/jenkins** · Sponsored OSS · 1B+ · 3.8K · By Jenkins · Updated 9 days ago · The leading open source automation server · Linux · Windows · unknown · ppc64le · IBM Z · unknown · x86-64 · arm · arm64

Seleccionamos **jenkins/jenkins**

# Preparación del ambiente.

[Overview](#) [Tags](#)

## Jenkins Continuous Integration and Delivery server.

This is a fully functional Jenkins server, based on the weekly and LTS releases .



# Jenkins

- To use the latest LTS: `docker pull jenkins/jenkins:lts-jdk17`
- To use the latest weekly: `docker pull jenkins/jenkins:jdk17`
- Lighter alpine, Windows and other JDKs based image also available
- (Recommended) Specific versions (to be pinned) are also available: `docker pull jenkins/jenkins:<version>-<jdk>` (Example: `jenkins/jenkins:2.414.3-jdk17` or `jenkins/jenkins:2.430-jdk21`)

Read [documentation](#) for usage

### Docker Pull Command

```
docker pull jenkins/jenkins
```

De aquí, tomamos el comando para ejecutar en la terminal. Debemos estar conectados por ssh

# Preparación del ambiente.

## Instalación de Jenkins

Ejecutamos el comando **docker pull jenkins/jenkins**, previamente tomado del hub de docker, damos un tiempo para que se pueda instalar.

```
Last login: Tue Dec 26 23:37:20 2023 from 192.168.1.152
parallels@ubuntu-linux-22-04-02-desktop:~$ docker pull jenkins/jenkins
Using default tag: latest
latest: Pulling from jenkins/jenkins
b66b4ecd3ecf: Pull complete
654720c366d6: Pull complete
7cb1be6d6fbf: Pull complete
3a30b851a06: Pull complete
b6b1f0e95547: Pull complete
73e1121b372b: Pull complete
491a8670a1a6: Pull complete
4e798a050b4a: Pull complete
46ddbcd01c47: Downloading [====>                               ] 7.009MB/73.74MB
69b717b42f32: Download complete
5e5f7863e852: Download complete
00e5d5f67073: Download complete
```

Para comprobar, ejecutamos el comando **docker images**, y vemos que está instalado.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
jenkins/jenkins    latest   3ee72417d7e1  2 days ago   502MB
hello-world         latest   ee301c921b8a  8 months ago  9.14kB
```

# Preparación del ambiente.

## Instalación de Jenkins

Siguiendo con la preparación del ambiente, creamos una carpeta para el curso, con el siguiente comando **mkdir -p jenkins/jenkins\_home** y con **ls** revisamos las carpetas que tenemos.

```
parallels@ubuntu-linux-22-04-02-desktop: ~
parallels@ubuntu-linux-22-04-02-desktop:~$ mkdir -p jenkins/jenkins_home
parallels@ubuntu-linux-22-04-02-desktop:~$ ls
'.' '\004' Desktop Documents Downloads Music Pictures Public Templates Videos jenkins
snap
parallels@ubuntu-linux-22-04-02-desktop:~$
```

Ahora, debemos otorgar los permisos necesarios a la carpeta de jenkins y la capeta para las actividades del curso, con el comando **chown 1000 jenkins** (Jenkins utiliza el id 1000).

```
parallels@ubuntu-linux-22-04-02-desktop: ~
parallels@ubuntu-linux-22-04-02-desktop:~$ chown 1000 jenkins
parallels@ubuntu-linux-22-04-02-desktop:~$
```

# Preparación del ambiente.

## Instalación de Jenkins

Con algún editor de texto (Visual Studio Code o Vim) escribimos el siguiente código, ésta es la parametrización del ambiente que se ejecutará en Docker.

```
version: '3' # es la version de jenkins a instalar

services:
  jenkins:
    image: jenkins/jenkins #establecemos la imagen que instalamos en los pasos anteriores
    ports:
      - 8080:8080 #puerto 8080 de la maquina al puerto 8080 del contenedor
      - 50000:50000 #puerto 50000 de la maquina al puerto 50000 del contenedor
    container_name: jenkins
    volumes:
      - $PWD/jenkins_home:/var/jenkins_home # para establecer los volumenes de jenkins en la carpeta creada
(jenkins_home)
    networks:
      - net
networks:
  net:
```

# Preparación del ambiente.

## Instalación de Jenkins

- Ingresamos a Ubuntu a través de SSH desde la terminal y por el mismo medio ingresamos al directorio con el comando **cd Jenkins**
- Dentro de esta carpeta vamos a generar el archivo con el editor de texto que instalamos al principio (Vim) con el siguiente comando, **vi docker-compose.yml**, esto nos permitirá ingresar al editor creando el documento con este nombre, una vez en él, apretamos **i** para que nos permita editar.

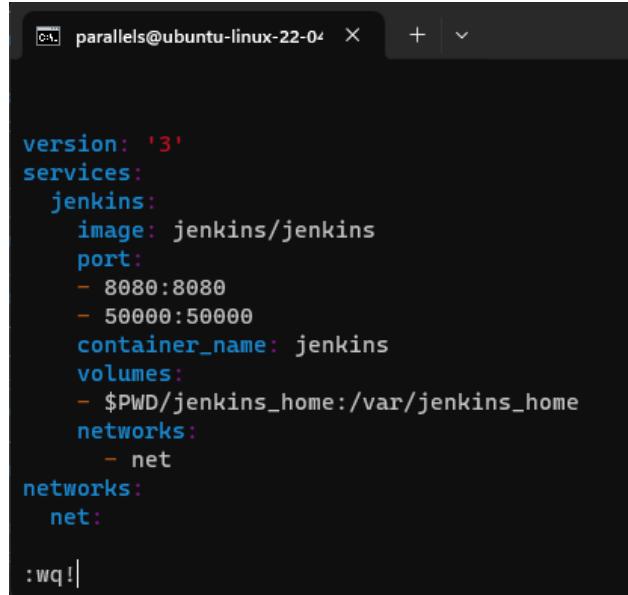


A screenshot of a terminal window titled "parallels@ubuntu-linux-22-04 ~". The window is dark-themed. At the bottom, there is a status bar with the text "-- INSERT --" on the left, "0,1" in the center, and "All" on the right. The main area of the terminal is completely blank, showing only a few vertical scroll marks on the left edge.

# Preparación del ambiente.

## Instalación de Jenkins

Pegamos el código en el editor, después apretamos **esc** para salir de la edición, y finalmente **wq!** para guardar y salir.



```
version: '3'
services:
  jenkins:
    image: jenkins/jenkins
    port:
      - 8080:8080
      - 50000:50000
    container_name: jenkins
    volumes:
      - $PWD/jenkins_home:/var/jenkins_home
    networks:
      - net
networks:
  net:
```

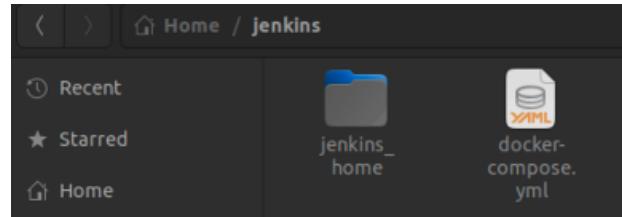
:wq !

# Preparación del ambiente.

## Instalación de Jenkins

Con el comando **ls** listamos los directorios para poder revisar que el archivo se creó y con **cat docker-compose.yml** ingresamos al archivo revisamos o confirmamos que el contenido es el adecuado.

```
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ vi docker-compose.yml
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ ls
docker-compose.yml jenkins_home
parallels@ubuntu-linux-22-04-02-desktop:~/jenkins$ cat docker-compose.yml
version: '3'
services:
  jenkins:
    image: jenkins/jenkins
    port:
      - 8080:8080
      - 50000:50000
    container_name: jenkins
    volumes:
      - $PWD/jenkins_home:/var/jenkins_home
    networks:
      - net
networks:
  net
```



Esto, es un archivo de docker,  
donde se establecen los  
parámetros para iniciar jenkins...

## **Preparación del ambiente.**

# Instalación de Jenkins

- Es momento de inicializar Docker compose con el comando **docker-compose up -d**
  - Y después con **docker ps** para desplegar los contenedores ejecutándose.
  - **docker run -u 0 -d -p8080:8080 -p50000:50000 -v data/Jenkins:var/Jenkins\_home Jenkins/Jenkins:its**

# Jenkins

Como se podran imaginar, herramientas adicionales como GitHub se usan mucho en estas plataformas, de forma complementaria... entonces vamos a implementar el plug in de GitHub.

Para ello nos vamos a **admin de jenkins > plugins > plugins instalados > GitHub Branch Source**

Panel de Control > Administrar Jenkins

+ Nueva Tarea  
Personas  
Historial de trabajos  
**Administrador Jenkins**  
Mis vistas

Trabajos en la cola  
No hay trabajos en la cola

Estado del ejecutor de construcciones  
1 Inactivo  
2 Inactivo

System Configuration

System  
Configurar variables globales y rutas.

Plugins  
Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.



Plugins

Updates 18  
Available plugins  
**Installed plugins**  
Advanced settings

Nombre: GitHub API Plugin 1.318-461.v7a\_c09c9fa\_d63 Activados Report an issue with this plugin

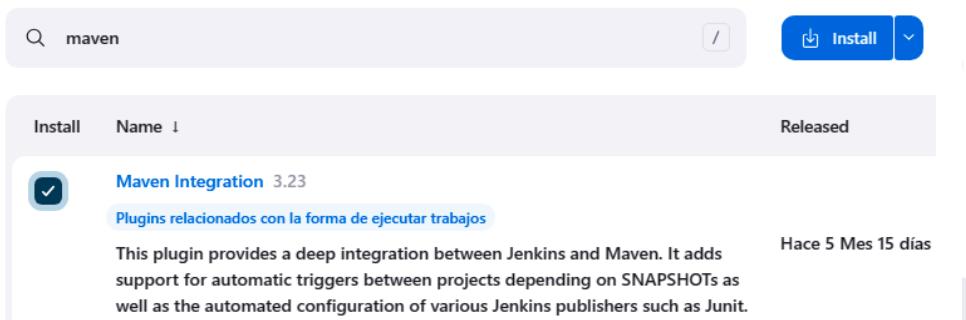
GitHub Branch Source 1767.va\_7d01ea\_c7256 Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc. Report an issue with this plugin

GitHub plugin 1.37.3.1 This plugin integrates GitHub to Jenkins. Report an issue with this plugin

Pipeline: GitHub Groovy Libraries 42.v0739460cda\_c4 Allows Pipeline Groovy libraries to be loaded on the fly from GitHub. Report an issue with this plugin

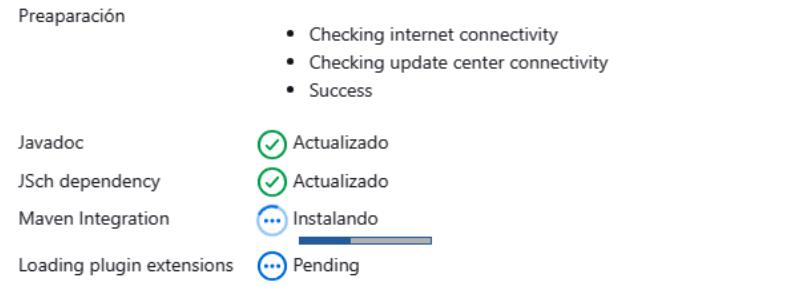
# Jenkins

Vamos a instalar el plugin “maven integration plugin”. En la sección de **plugins disponibles** lo buscamos > install



A screenshot of the Jenkins plugin search interface. A search bar at the top contains the text "maven". Below it, a table lists the "Maven Integration" plugin. The table has columns for "Install", "Name", and "Released". The "Maven Integration" row shows a checked checkbox under "Install", the name "Maven Integration 3.23", and the status "Released Hace 5 Mes 15 días". A tooltip "Plugins relacionados con la forma de ejecutar trabajos" is visible. A detailed description below the table states: "This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTs as well as the automated configuration of various Jenkins publishers such as Junit."

## Download progress



A screenshot of the Jenkins download progress page. It shows a "Preparación" section with three items: "Checking internet connectivity", "Checking update center connectivity", and "Success", each marked with a green checkmark. Below this is a "Download progress" section with four items: "Javadoc" (Actualizado), "JSch dependency" (Actualizado), "Maven Integration" (Instalando, accompanied by a progress bar), and "Loading plugin extensions" (Pending). A note at the bottom right says "Puedes cerrar esta ventana una vez que los plugins estén instalados".

→ [Volver al inicio de la página](#)

(puedes empezar a usar los plugins instalados inmediatamente)

→  Reiniciar Jenkins cuando termine la instalación y no queden trabajos en ejecución

# Jenkins

Vamos a instalar el plugin de maven integration plugin. En la sección de **plugins disponibles lo buscamos > install** esperamos a que termine.



**Por favor espera hasta que Jenkins acabe de reiniciarse. ...**

Su navegador recargará esta página cuando Jenkins esté listo.

**Safe Restart**

Builds on agents can usually continue.

# Jenkins

Una vez reiniciado, es necesario configurar algunas variables del plugin de maven, esta vez desde la sección de **herramientas (tools) > maven > añadir maven > guardamos**

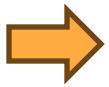


## Tools

Configure tools, their locations and automatic installers.

### instalaciones de Maven

Añadir Maven



Añadir Maven

Maven

Nombre

Maven\_Curso\_Jenkins\_Equipo\_1

Instalar automáticamente ?

Instalar desde Apache

Versión

3.9.6

Añadir un instalador ▾

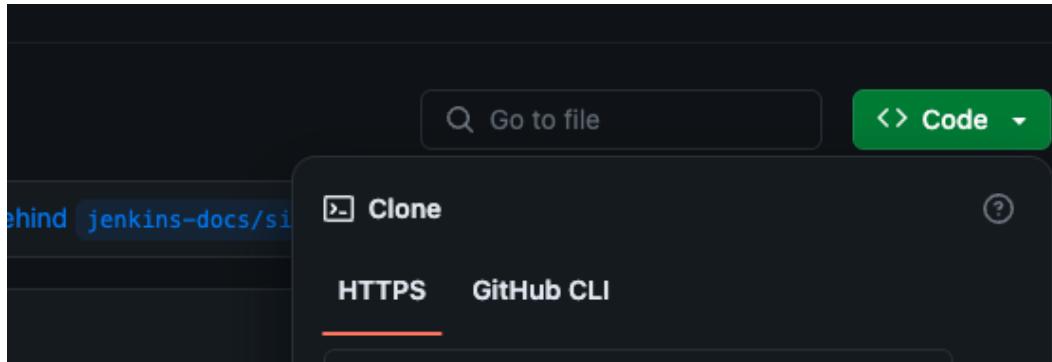
Añadir Maven

Save Apply

# Jenkins

Desde el repositorio de GitHub, copiamos el enlace https del botón code. Es decir, la dirección del repositorio.

[https://github.com/LuisECJ21/Curso\\_Jenkins\\_24.git](https://github.com/LuisECJ21/Curso_Jenkins_24.git)



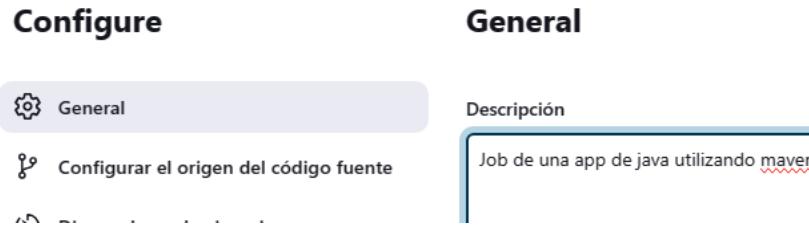
# Jenkins

Ahora vamos a crear un nuevo proyecto/job de estilo libre desde jenkins, el nombre sugerido es **Java App con maven**.

Se sugiere como siempre poner una descripción, en los jobs, como se ve en la imagen.

**Configure**

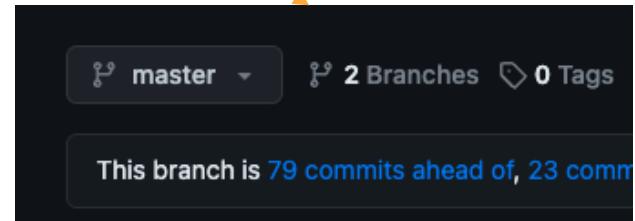
**General**



The screenshot shows the Jenkins 'General' configuration page. On the left, there's a sidebar with a gear icon labeled 'General'. Below it are other tabs: 'Configurar el origen del código fuente' (Configure code source origin), 'Advanced...' (Advanced...), and 'Build' (Build). The main area has a title 'Descripción' (Description) with a text input field containing the text 'Job de una app de java utilizando maven'.

## Jenkins

- En la sección de configurar el origen del código fuente, pegamos el link https copiado de GitHub.
- Sin credenciales por ahora, dado que, es un repositorio público.
- El branch es el master, que tambien lo podemos ver en GitHub.
- Guardamos y construimos.



# Jenkins

Configurar el origen del código fuente

Ninguno

Git 

**Repositories** 

Repository URL 

Credentials   
- none -

Avanzado 

Add Repository

**Branches to build** 

Branch Specifier (blank for 'any') 

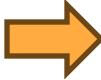
Add Branch

Navegador del repositorio 

Additional Behaviours  
 

## ✓ Salida de consola

Started by user Luis Carmona  
Running as SYSTEM  
Building in workspace /var/jenkins\_home/workspace/Java\_App\_Maven  
The recommended git tool is: NONE  
No credentials specified  
Cloning the remote Git repository  
Cloning repository <https://github.com/maclojulian/simple-java-maven-app.git>  
> git init /var/jenkins\_home/workspace/Java\_App\_Maven # timeout=10  
Fetching upstream changes from <https://github.com/maclojulian/simple-java-maven-app.git>  
> git --version # timeout=10  
> git --version # 'git version 2.39.2'  
> git fetch --tags --force --progress -- <https://github.com/maclojulian/simple-java-maven-app.git>  
+refs/heads/\*:refs/remotes/origin/\* # timeout=10  
> git config remote.origin.url <https://github.com/maclojulian/simple-java-maven-app.git> # timeout=10  
> git config --add remote.origin.fetch +refs/heads/\*:refs/remotes/origin/\* # timeout=10  
Avoid second fetch  
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10  
Checking out Revision 9396b7c42f467673997e3d5f60b9f460a734e178 (refs/remotes/origin/master)  
> git config core.sparsecheckout # timeout=10  
> git checkout -f 9396b7c42f467673997e3d5f60b9f460a734e178 # timeout=10  
Commit message: "javamavenDSL3.groovy"  
First time build. Skipping changelog.  
Finished: SUCCESS



# Jenkins

Si nos dirigimos a la ruta resaltada en la lámina anterior, vemos los archivos que se generaron.

## ✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Java_App_Maven
The recommended git tool is: NONE
```

Name	Size	Modified
DSL	3 items	22:10
jenkins	2 items	22:10
src	2 items	22:10
Dockerfile	111 bytes	22:10
Dockerfile2	180 bytes	22:10
Jenkinsfile1	578 bytes	22:10
Jenkinsfile3	1.1 kB	22:10
Jenkinsfile4	1.9 kB	22:10
README.md	744 bytes	22:10
pom.xml	1.4 kB	22:10

# Jenkins

Ahora haremos la construcción o build de la app, vamos al job que creamos en las láminas pasadas, en la sección de configuración.

A diferencia de los jobs pasados, donde seleccionábamos la shell para la construcción, esta vez seleccionaremos **ejecutar tareas maven de nivel superior > maven\_curso\_jenkins\_equipo1> goles** (pegamos una parte del comando de la sección build del archivo jenkinsfile en GitHub) > guardamos y construimos

## Build Steps

Añadir un nuevo paso ^

Filter

- Ejecutar Ant
- Ejecutar línea de comandos (shell)
- Ejecutar tareas 'maven' de nivel superior
- Ejecutar un comando de Windows
- Invoke Gradle script
- Run with timeout
- Set build status to "pending" on GitHub commit

Build Steps

Ejecutar tareas 'maven' de nivel superior

Version de Maven

- (Por defecto)
- Maven\_Curso\_Jenkins\_Equipo\_1

Build Steps

Ejecutar tareas 'maven' de nivel superior

Version de Maven

(Por defecto)

Goles

-B -DskipTests clean package

Avanzado ▾

Guardar Apply

# Jenkins

```
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/iq80/snappy/snappy/0.4/snappy-0.4.jar (58 kB at 205 kB/s)
```

```
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.5/xz-1.5.jar (100 kB at 331 kB/s)
```

```
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.24/plexus-utils-3.0.24.jar (247 kB at 768 kB/s)
```

```
[INFO] Building jar: /var/jenkins_home/workspace/Java_App_Maven/target/my-app-1.0-SNAPSHOT.jar
```

```
[INFO]
```

```
[INFO] BUILD SUCCESS
```

Construcción satisfactoria

```
[INFO]
```

```
[INFO] Total time: 10.577 s
```

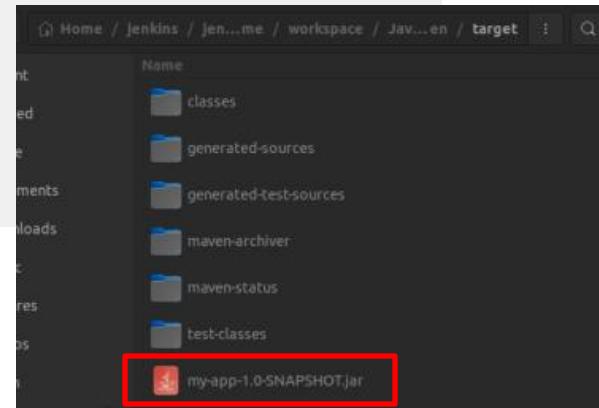
```
[INFO] Finished at: 2024-01-18T05:44:33Z
```

```
[INFO]
```

```
Finished: SUCCESS
```

Finalización exitosa

Ubicación del .JAR



# Jenkins

- Ahora vamos a hacer a ejecutar las pruebas o test del build o construcción.
- En un proceso similar al anterior, ingresando al archivo jenkins file en GitHub, pero ahora en la sección de Test.
- Vamos a jenkins a configurar el **job > build steps > agregamos un nuevo paso >ejecutar tareas maven de nivel superior > goles (copiamos el codigo que tenemos en GitHub)**
- Guardamos y construimos.



Acciones para ejecutar después.

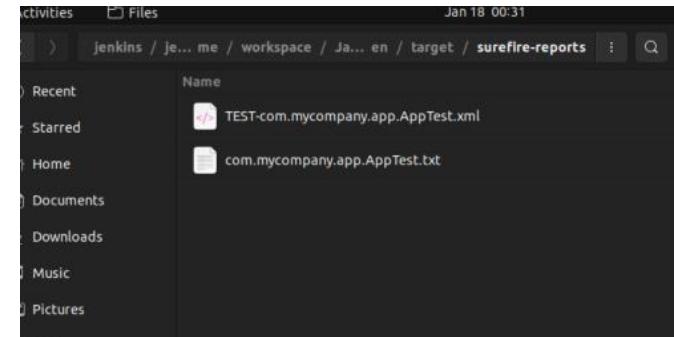
Añadir una acción ▾

# Jenkins

Resultados de la construcción de las pruebas.

```
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.mycompany.app.AppTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.033 s -- in com.mycompany.app.AppTest
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  1.782 s
[INFO] Finished at: 2024-01-18T06:23:33Z
[INFO] -----
Finished: SUCCESS
```

Archivos generados de la construcción.



## Jenkins

Ahora es momento de ejecutar o desplegar la aplicación , ¿Por qué? Ya pasó por todas las etapas del ciclo de vida, entonces es una aplicación “válida”.

Desde la terminal, tenemos que ubicar donde se encuentra el .JAR o graficamente en Ubuntu obtenerla.

Para la primera opción ingresamos al contenedor con los siguientes comandos.

**docker exec -ti jenkins bash** → con este comando ingresamos al contenedor.

**cd /var/jenkins\_home** → para entrar a la ruta/carpeta jenkins\_home.

**cd workspace** → para entrar en la ruta/carpeta workspace.

**ls** → para listar todos los archivos/carpetas en la ruta.

**cd Java\_App\_Maven (nombre de la carpeta o del job)** → ruta que tiene el job.

**cd target** → para posicionarnos en la carpeta donde está el .JAR

**ls** → para listar todos los archivos/carpetas en la ruta.

**Pwd** → para tener la ruta completa, copiarla y pegarla en su momento.

# Jenkins

```
parallels@ubuntu-linux-22-04 ~ % + | ~  
  
parallels@ubuntu-linux-22-04-02-desktop:~$ docker exec -ti jenkins bash  
-bash: dcoker: command not found  
parallels@ubuntu-linux-22-04-02-desktop:~$ docker exec -ti jenkins bash  
jenkins@12632ad11607:/$ cd /var/jenkins_home  
jenkins@12632ad11607:~$ cd workspace  
jenkins@12632ad11607:~/workspace$ ls  
Java_App_Maven Job3_Parametro_booleano Job_1_test job2_parametrizacion  
jenkins@12632ad11607:~/workspace$ cd Java_App_Maven  
jenkins@12632ad11607:~/workspace/Java_App_Maven$ ls  
DSL Dockerfile2 Jenkinsfile3 README.md pom.xml target  
Dockerfile Jenkinsfile1 Jenkinsfile4 jenkins src  
jenkins@12632ad11607:~/workspace/Java_App_Maven$ cd target  
jenkins@12632ad11607:~/workspace/Java_App_Maven/target$ ls  
classes generated-test-sources maven-status surefire-reports  
generated-sources maven-archiver my-app-1.0-SNAPSHOT.jar test-classes  
jenkins@12632ad11607:~/workspace/Java_App_Maven/target$ pwd  
/var/jenkins_home/workspace/Java_App_Maven/target  
jenkins@12632ad11607:~/workspace/Java_App_Maven/target$
```

Todos los comandos de la lámina anterior, desde la terminal.

# Jenkins

Una vez con la ruta del .JAR completa **/var/jenkins\_home/workspace/Java\_App\_Maven/target**

Vamos a Jenkins y configuramos el job, en la sección de build steps, agregamos un Shell como lo hemos hecho antes. (Sin borrar o eliminar los previos), donde agregamos los siguientes comandos:

**java -jar (ruta obtenida en la lámina anterior)**

**java -jar/var/jenkins\_home/workspace/Java\_App\_Maven/target/*my-app-1.0-SNAPSHOT.jar***

Guardamos y construimos.

```
≡ Ejecutar línea de comandos (shell) ?  
Comando  
Visualizar la lista de variables de entorno disponibles  
  
java -jar/var/jenkins_home/workspace/Java_App_Maven/target/my-app-1.0-SNAPSHOT.jar
```



Esto lo tenemos que agregar, recuerden nos da la ruta del archivo, pero sin incluir el nombre del archivo

# Jenkins

```
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  0.613 s
[INFO] Finished at: 2024-01-18T07:10:24Z
[INFO] -----
[Java_App_Maven] $ /bin/sh -xe /tmp/jenkins6889006844096319692.sh
+ java -jar /var/jenkins_home/workspace/Java_App_Maven/target/my-app-1.0-SNAPSHOT.jar
Hello World!
Finished: SUCCESS
```

Aquí tenemos un pequeño, simple, pero gráfico de CI/CD

## Jenkins

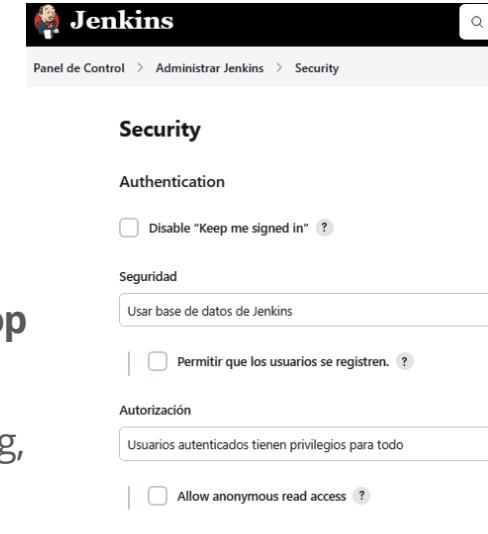
Como en la mayoría de las tecnologías de la información, la seguridad es uno de los factores más importantes que se deben de considerar. Así que a continuación algunos puntos y/o sugerencias:

- Conservar la herramienta “fuera” o aislado de internet.
- Agregar las IP de GitHub/BitBucker y configurar para que los complementos puedan ejecutarse (Maven).
- Establecer la autenticación y autorizaciones adecuadas, para ello complementarlo con herramientas más “avanzadas” como LDAP o SAML.
- Tener a Jenkins siempre actualizado, también los plugins.
- Usar la versión LTS (Long Term Support).
- Leer el registro de cambios (Change log).

# Jenkins

## Autenticación en jenkins.

En la sección de administración de Jenkins > seguridad. Podemos observar que ya no tenemos la opción de habilitar o deshabilitar la Autenticación, esto desde las versiones **2.214 y LTS 2.222.1** que se eliminó, aunque aún lo podemos hacer desde la terminal.



The screenshot shows the Jenkins Administration interface under the 'Security' tab. In the 'Authentication' section, there is a checkbox labeled 'Disable "Keep me signed in"' which is currently unchecked. Below this, there is a section titled 'Seguridad' with a checkbox for 'Usar base de datos de Jenkins'. At the bottom, there are sections for 'Autorización' and 'Allow anonymous read access'.

Para ello debemos detener Jenkins con el comando **docker-compose stop**  
O **docker stop #de contenedor**

Entramos a la carpeta Jenkins\_home, donde se encuentra el archivo config, que editaremos en el editor de texto de ubuntu (Vim) con el comando  
**vi config.xml**

# Jenkins

```
parallels@ubuntu-linux-22-04 ~ + v
<?xml version='1.1' encoding='UTF-8'?>
<hudson>
  <disabledAdministrativeMonitors/>
  <version>2.438</version>
  <numExecutors>2</numExecutors>
  <mode>NORMAL</mode>
  <useSecurity>true</useSecurity>
    <authorizationStrategy class="hudson.security.FullControlOnceLoggedInAuthorizationStrategy">
      <denyAnonymousReadAccess>true</denyAnonymousReadAccess>
    </authorizationStrategy>
    <securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
      <disableSignup>true</disableSignup>
      <enableCaptcha>false</enableCaptcha>
    </securityRealm>
  <disableRememberMe>false</disableRememberMe>
  <projectNamingStrategy class="jenkins.model.ProjectNamingStrategy$DefaultProjectNamingStrategy"/>
  <workspaceDir>${JENKINS_HOME}/workspace/${ITEM_FULL_NAME}</workspaceDir>
  <buildsDir>${ITEM_ROOTDIR}/builds</buildsDir>
  <jdks/>
  <viewsTabBar class="hudson.views.DefaultViewsTabBar"/>
  <myViewsTabBar class="hudson.views.DefaultMyViewsTabBar"/>
  <clouds/>
  <scmCheckoutRetryCount>0</scmCheckoutRetryCount>
  <views>
    <hudson.model.AllView>
      <owner class="hudson" reference=". . . . ."/>
      <name>all</name>
      <filterExecutors>false</filterExecutors>
      <filterQueue>false</filterQueue>
    </hudson.model.AllView>
  </views>
-- INSERT --
```

Este valor lo cambiamos a false

Borramos todo lo que está en las etiquetas de Authorization Strategy

Apretamos :wq! Para que guarde los cambios y volvemos a iniciar el contenedor y jenkins... ¿Cuál fue la diferencia?

# Jenkins

Es momento de restaurar el cambio que hicimos.

**Administrar Jenkins > Seguridad > usar BD de Jenkins > usuarios autenticados tienen privilegio para todo > guardamos > actualizamos el explorador**

¿Qué pasa?

## Security

### Authentication

Disable "Keep me signed in" ?

### Seguridad

None

Delegar seguridad al contenedor de servlets

LDAP

Unix user/group database

Usar base de datos de Jenkins

None



### Autorización

Cualquiera puede hacer cualquier acción

Configuración de seguridad

Cualquiera puede hacer cualquier acción

Estrategia de seguridad para el proyecto

Modo 'legacy'

Usuarios autenticados tienen privilegios para todo

Markup Formatter



## Sign in to Jenkins

Username

Contraseña

Keep me signed in

Sign in

# Jenkins

Vamos a establecer y homologar los conceptos de los que hemos estado hablando.

**Autenticación:** es la acción de verificar la identidad de un usuario.

**Autorización:** es la función de especificar los derechos de acceso a los recursos relacionados con la seguridad de la información en general y al control de acceso en particular.

Dentro de jenkins tenemos principalmente 4 opciones de autorización.

- Cualquiera puede hacer cualquier acción.
- Usuarios autenticados tienen privilegios para todo.
- Estrategia de seguridad para el proyecto.
- Role-based strategy plan.

# Jenkins

Ahora, si tenemos la posibilidad de tener distintos usuarios, ¿cómo los defino?

Dentro de jenkins nos posicionamos en **admin. De jenkins > permitir que los usuarios se registren > Guardamos**

## Authentication

Disable "Keep me signed in" ?

## Seguridad

Usar base de datos de Jenkins

Permitir que los usuarios se registren. ?

 With signup enabled, anyone on your network can sign up to any authenticated user.

Desde otro navegador, ingresamos a la GUI de Jenkins ¿Qué vemos diferente?...

## Sign in to Jenkins

Username

Contraseña

Keep me signed in

Sign in

or register

# Jenkins

¡Exacto! Ahora tenemos la opción de crear o registrar una cuenta nueva...

Hagámoslo para ver el funcionamiento.

## Register

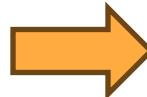
Username

Full name

Email

Password  Show

A strong password is a long password that's unique for every site. Try using a phrase with 5-6 words for the best security.



## Register

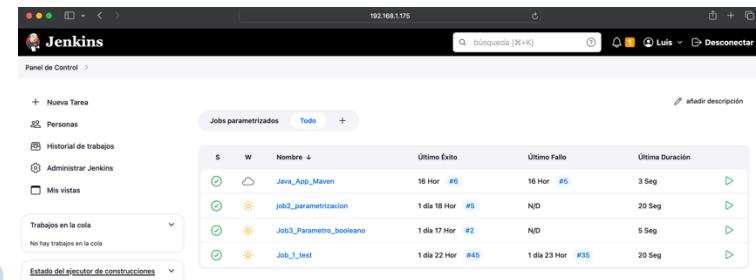
Username

Full name

Email

Password  Show

Strength: Poor



The Jenkins dashboard displays the following information:

- Panel de Control >**
- Nueva Tarea**: +
- Personas**: Personas
- Historial de trabajos**: Historial de trabajos
- Administrador Jenkins**: Administrador Jenkins
- Mis vistas**: Mis vistas
- Trabajos en la cola**: Trabajos en la cola
- Jobs parametrizados**: Jobs parametrizados
- Último Éxito**: Último Éxito
- Último Fallo**: Último Fallo
- Última Duración**: Última Duración
- Java\_App\_Maven**: 16 Hor #6, 16 Hor #5, 3 Seg
- job2\_parametrizacion**: 1 dia 18 Hor #5, N/D, 20 Seg
- Job3\_Parametro\_booleano**: 1 dia 17 Hor #2, N/D, 5 Seg
- Job1\_test**: 1 dia 22 Hor #45, 1 dia 23 Hor #35, 20 Seg

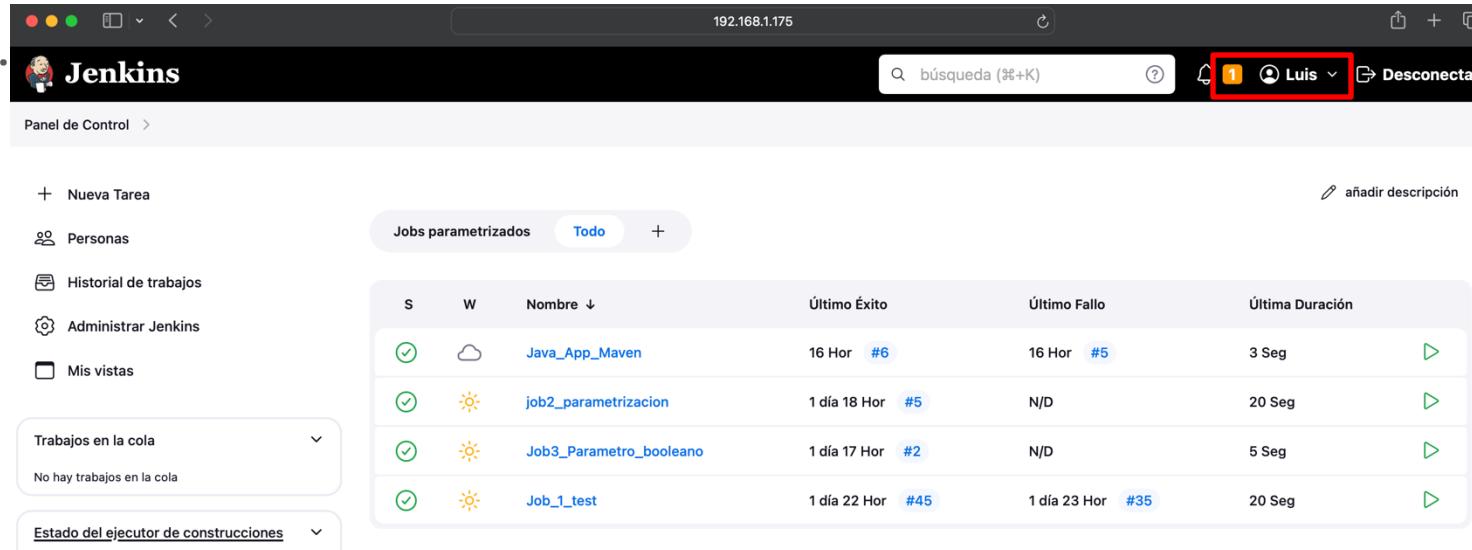
Create account

or sign in

Create account

or sign in

# Jenkins



The screenshot shows the Jenkins dashboard at the URL 192.168.1.175. The top navigation bar includes a search bar, a user icon with the number '1' (highlighted with a red box), the name 'Luis', and a 'Desconectar' button. The main content area displays a list of jobs: Java\_App\_Maven, job2\_parametrizacion, Job3\_Parametro\_booleano, and Job\_1\_test. Each job entry shows its status (green checkmark or sun icon), name, last success time, last failure time, and duration.

S	W	Nombre	Último Éxito	Último Fallo	Última Duración
✓	cloud	Java_App_Maven	16 Hor #6	16 Hor #5	3 Seg
✓	sun	job2_parametrizacion	1 día 18 Hor #5	N/D	20 Seg
✓	sun	Job3_Parametro_booleano	1 día 17 Hor #2	N/D	5 Seg
✓	sun	Job_1_test	1 día 22 Hor #45	1 día 23 Hor #35	20 Seg

Tenemos las mismas opciones, dado que así lo seleccionamos, que todos los perfiles autenticados tengan los mismos privilegios.

# Jenkins

En la sección de personas o people (depende el idioma configurado) a la que podemos acceder desde **admin. Jenkins > seguridad > usuarios**

**Security**

- Security**  
Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización)
- Credential Providers**  
Configure the credential providers and types

- Credentials**  
Configure credentials
- Users**  
Crear/borrar/editar usuarios que puedan utilizar Jenkins Activar Windows Jenkins Ve a Configuración para activar \

Y desde aquí, tambien podemos generar cuentas o usuarios nuevos.

### Crear un usuario

Usuario

Contraseña

Confirma la contraseña

Nombre completo

Dirección de email

**Crear un usuario**

## Jenkins

Recordando lo que hemos hecho, en láminas anteriores seleccionamos en la autorización que todos los usuarios autenticados tienen privilegios para todo, y como hemos visto, de esta forma, todos los usuarios tienen acceso a ejecutar todo dentro de jenkins... eso no es una mejor práctica.

### Autorización

Usuarios autenticados tienen privilegios para todo

Entonces, debemos cambiar por una opción que nos permita seleccionar en función del usuario, las actividades que éste puede ejecutar en Jenkins. Esta puede ser Estrategia de seguridad del proyecto.

### Autorización

Usuarios autenticados tienen privilegios para todo

Configuración de seguridad

Cualquiera puede hacer cualquier acción

Estrategia de seguridad para el proyecto

Modo 'legacy'

Usuarios autenticados tienen privilegios para todo

# Jenkins

## Autorización

		Estrategia de seguridad para el proyecto										?			
		Global		Credentials		Nuevo		Tarea		Ejecutar		Vistas		Repositorio de software (SCM)	
Usuario/Grupo		Read	Delete	Create	Configure	Update	Replay	Delete	Workspace	Read	Move	Discover	Configure	Update	Tag
Anonymous		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Luis Carmona	✓	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

[Add user...](#) [Add group...](#) [?](#)

# Jenkins

## Autorización

Estrategia de seguridad para el proyecto



Usuario/Grupo	Global	Credentials	Nodo	Tarea	Ejecutar	Vistas	Repositorio de software (SCM)	Tag	
								Read	Delete
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				
Luis Carmona	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
! salomon	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Luis	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				

Add user...

Add group...



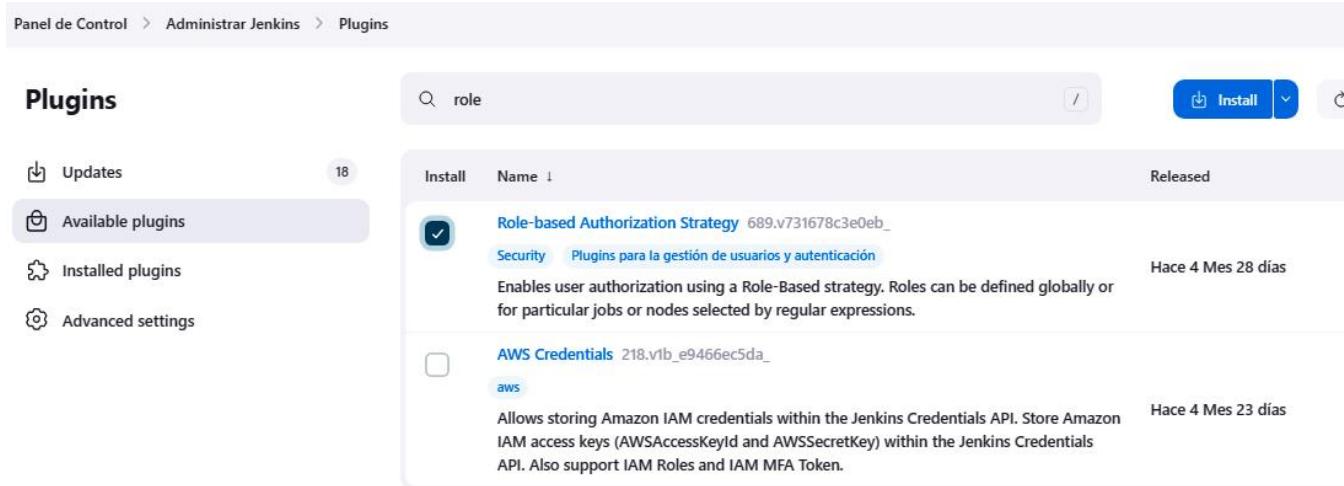
Aquí, podemos agregar todos los usuarios previamente generados, como Luis, pero marca error, con los que aún no hemos creado, como Salomon... para evitar este tipo de contratiempos, lo mejor es centralizar su gestión.

# Jenkins

Para centralizar, y evitar ciertas situaciones, por ejemplo, que todos se registren y tener un “sin fin” de usuarios, lo haremos a traves de un plugin **“role-based authorization strategy”**

En la seccion de **admin. Jenkins > Plugins > role-based authorization strategy > install > verificamos instalacion**

Panel de Control > Administrar Jenkins > Plugins



Install	Name	Released
<input checked="" type="checkbox"/>	<a href="#">Role-based Authorization Strategy</a> 689.v731678c3e0eb_	Hace 4 Mes 28 días
<input type="checkbox"/>	<a href="#">AWS Credentials</a> 218.v1b_e9466ec5da_	Hace 4 Mes 23 días

The 'Available plugins' section is highlighted with a grey background. The 'Role-based Authorization Strategy' plugin has a checked checkbox next to its name. The 'AWS Credentials' plugin has an unchecked checkbox.

# Jenkins

Una vez instalado el plugin, vamos de nuevo a la sección de **seguridad > autorizacion**  
Vemos que se agregó la opción "**Role-based Strategy**"

Antes

Autorización

- Usuarios autenticados tienen privilegios para todo
- Configuración de seguridad
- Cualquiera puede hacer cualquier acción
- Estrategia de seguridad para el proyecto
- Modo 'legacy'
- Usuarios autenticados tienen privilegios para todo

Después

Autorización

- Estrategia de seguridad para el proyecto
- Cualquiera puede hacer cualquier acción
- Modo 'legacy'
- Usuarios autenticados tienen privilegios para todo
- Configuración de seguridad
- Estrategia de seguridad para el proyecto
- Role-Based Strategy

Seleccionamos esta nueva opcion y guardamos.

# Jenkins

En la misma de sección de seguridad, vemos que despues de cambiar a “**Role-based Strategy**” hay una nueva opción “**manage and assign roles**”.

Antes

## Security

 **Security**  
Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización)

 **Credentials**  
Configure credentials

 **Credential Providers**  
Configure the credential providers and types

 **Users**  
Crear/borrar/editar usuarios que puedan utilizar Jenkins  Activar Windows  
Ve a Configuración para activar V

Después

## Security

 **Security**  
Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización)

 **Credential Providers**  
Configure the credential providers and types

 **Credentials**  
Configure credentials

 **Manage and Assign Roles**  
Handle permissions by creating roles and assigning them to users/groups

 **Users**  
Crear/borrar/editar usuarios que puedan utilizar Jenkins

# Jenkins

Uno de los perfiles o roles más usados es el de lectura, es decir, sólo ver, pero no poder modificar. Así que nos dirigimos a esta nueva sección, "**manage and assign roles**".

Role	Global	Credentials	Nodo	Tarea	Ejecutar	Vistas	Read	Re	Re
admin	✓						Delete	Create	Configure
							Read	Update	Update
							Delete	Read	Read
							Create	Delete	Delete
							Configure	Configure	Configure
							Update	Update	Update
							Read	Read	Read
							Delete	Delete	Delete
							Read	Read	Read
							Delete	Create	Create
							Move	Move	Move
							Discover	Discover	Discover
							Delete	Create	Create
							Configure	Configure	Configure
							Cancel	Cancel	Cancel
							Build	Build	Build
							Provision	Provision	Provision
							Disconnect	Disconnect	Disconnect
							Delete	Delete	Delete
							Create	Create	Create
							Connect	Connect	Connect
							Configure	Configure	Configure
							Build	Build	Build
							View	View	View
							Update	Update	Update
							ManageDomains	ManageDomains	ManageDomains
							Delete	Delete	Delete
							Create	Create	Create
							Read	Read	Read
							Administrator	Administrator	Administrator

Similar a la tabla o matriz de la estrategia de seguridad del proyecto, solo que aquí en lugar de asignarse por usuario es por rol.

# Jenkins

Empezaremos a crear nuestro rol de lectura.

**Role to add > nombre/descripcion > add**

## Manage Roles

### Global roles

Role	Global	Credentials	Manage Domains
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Role to add**

**Read-only**

Add

## Manage Roles

### Global roles

Role	Global	Credentials	Manage Domains
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Read-only	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Role to add**

**Read-only**

Add

De esta forma, ya tenemos agregado nuestro nuevo rol de “sólo lectura”.

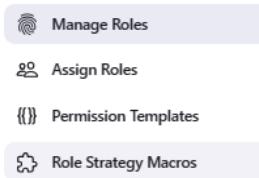
# Jenkins

Seleccionamos los permisos o accesos que queremos tenga. Merece la pena aclarar que la opción **read** de la sección **global** debe estar habilitado para que todo lo demás funcione.

Global roles

Role	Global		
	Administer	Read	Create
Read-only	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Es importante recordar que, en esta sección, estamos asignando roles, pero estos no están aún asociados a ningún usuario, por ende, debemos asignarlo a alguno. En la sección de **asignar roles** o **assign roles**.



# Jenkins

## Assign Roles

### Global roles

User/Group	Read-only	admin
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>
Luis Carmona	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Add User   Add Group

### Item roles

User/Group
Anonymous
Authenticated Users

Add User   Add Group

### Agent roles

User/Group
Anonymous
Authenticated Users

Add User   Add Group

Save   Apply

Aquí podemos ver que el rol de admin está asignado a Luis Carmona, que generé al inicio del curso, pero el perfil “read-only” recientemente generado, aún no lo está.

## Assign Roles

### Global roles

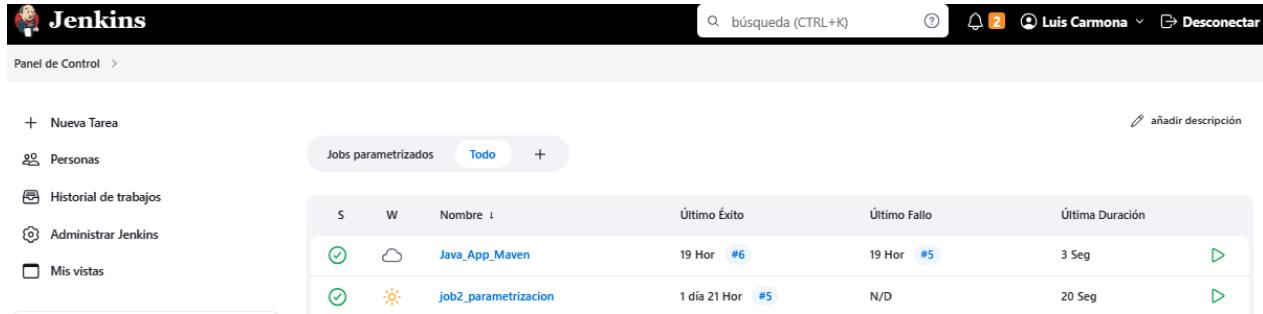
User/Group	Read-only	admin
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>
Luis Carmona	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Luis	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Add User   Add Group

Agregamos al usuario recientemente creado, desde el botón “add user”, y a éste le asignamos el rol “**read-only**” y guardamos.

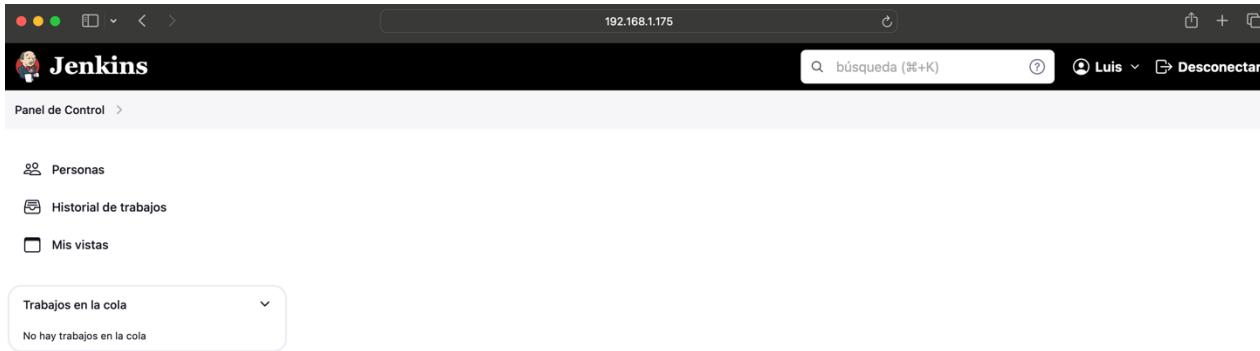
¿Recuerdan qué al momento de crear el usuario nuevo, este podía ejecutar todas las acciones que el primer usuario?

# Jenkins



The screenshot shows the Jenkins dashboard for a user named Luis Carmona. The top navigation bar includes a search bar, a help icon, a notification bell with 2 notifications, the user's name, and a 'Desconectar' (Disconnect) button. On the left, there is a sidebar with links for 'Nueva Tarea', 'Personas', 'Historial de trabajos', 'Administrar Jenkins', and 'Mis vistas'. The main content area displays a table of jobs, with 'Jobs parametrizados' selected. The table has columns for S (Status), W (Workdir), Nombre (Name), Último Éxito (Last Success), Último Fallo (Last Failure), and Última Duración (Last Duration). Two jobs are listed: 'Java\_App\_Maven' and 'job2\_parametrizacion', both marked as successful (green checkmark).

Esta es la vista del rol de admin asignado al primer usuario (puede ejecutar todo).



The screenshot shows the Jenkins dashboard for a user named Luis. The top navigation bar is identical to the previous one. The sidebar on the left shows 'Personas', 'Historial de trabajos', and 'Mis vistas'. The main content area displays a table titled 'Trabajos en la cola' (Jobs in the queue). A message at the bottom states 'No hay trabajos en la cola' (There are no jobs in the queue).

Esta es la vista del rol “only-read” asignada al usuario LuisECJ1021.

# Jenkins

Vamos a agregar/crear un rol de ejecucion y lo agregaremos a un usuario. Repetimos el proceso que para el rol de solo lectura.

**Admin. Jenkins > manage and asign roles > manage roles > role to add > add > save.**

Le asignamos las casillas dentro de **tarea > build, cancel y read**.

Guardamos y construimos.

Panel de Control > Administrar Jenkins > Manage and Assign Roles > Manage Roles

**Manage Roles**

**Global roles**

Role	Global	Credentials	Nodo	Tarea	Ejecutar	Vistas	Repositorio de software (SCM)
Administrador	<input type="checkbox"/> Read	<input checked="" type="checkbox"/> Create	<input type="checkbox"/> Delete	<input type="checkbox"/> Provision	<input checked="" type="checkbox"/> Build	<input type="checkbox"/> Configure	<input type="checkbox"/> Workspace
Read-only	<input type="checkbox"/> Read	<input type="checkbox"/> View	<input type="checkbox"/> Update	<input type="checkbox"/> Disconnect	<input checked="" type="checkbox"/> Cancel	<input type="checkbox"/> Delete	<input type="checkbox"/> Read
admin	<input checked="" type="checkbox"/> Read	<input checked="" type="checkbox"/> Create	<input type="checkbox"/> Delete	<input type="checkbox"/> Connect	<input checked="" type="checkbox"/> Build	<input type="checkbox"/> Configure	<input type="checkbox"/> Workspace

**Role to add**

Ejecucion

Add

**Manage Roles**

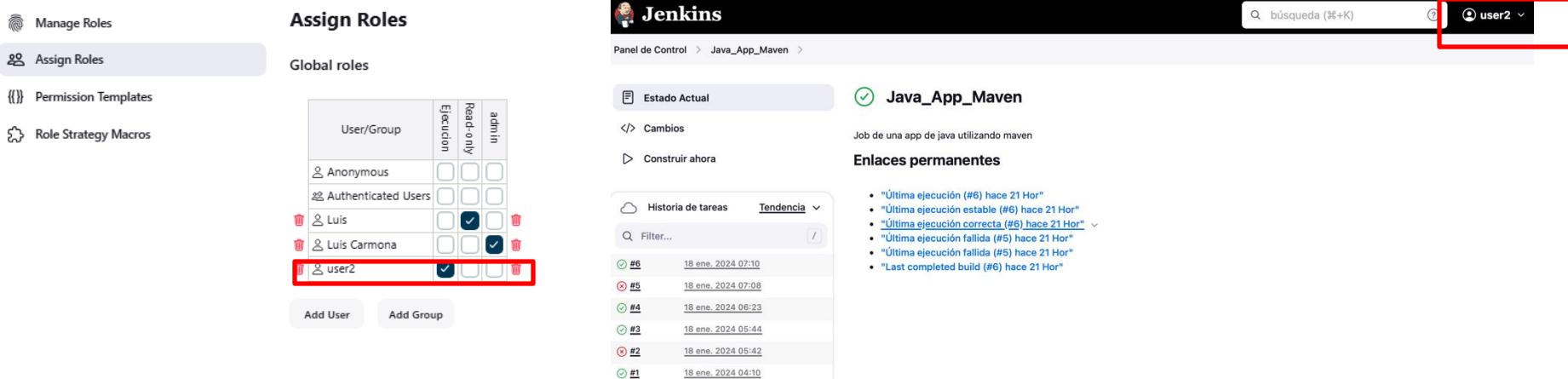
**Global roles**

Role	Global	Credentials	Nodo	Tarea	Ejecutar	Vistas	Repositorio de software (SCM)
Ejecucion	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Read-only	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



# Jenkins

En la sección de asignación de rol, asignamos el rol de ejecución a otro usuario, como se ve en la imagen. Y vemos como tenemos habilitadas las opciones que le asignamos al rol y usuario.



The image shows two screenshots of the Jenkins interface. On the left, the 'Assign Roles' page under 'Manage Roles' shows a grid where the 'Execution' role is assigned to the user 'user2'. On the right, the 'Java\_App\_Maven' job page shows the history of builds, with the last completed build (#6) being successful.

**Assign Roles (Left Screenshot)**

**Global roles**

User/Group	Execution	Read-only	admin
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Luis	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Luis Carmona	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>user2</b>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Add User   Add Group

**Java\_App\_Maven (Right Screenshot)**

Estado Actual

Cambios

Construir ahora

Historia de tareas

#	Fecha
#6	18 ene. 2024 07:10
#5	18 ene. 2024 07:08
#4	18 ene. 2024 06:23
#3	18 ene. 2024 05:44
#2	18 ene. 2024 05:42
#1	18 ene. 2024 04:10

Enlaces permanentes

- "Última ejecución (#6) hace 21 Hor"
- "Última ejecución estable (#6) hace 21 Hor"
- "Última ejecución correcta (#6) hace 21 Hor"
- "Última ejecución fallida (#5) hace 21 Hor"
- "Última ejecución fallida (#5) hace 21 Hor"
- "Last completed build (#6) hace 21 Hor"

# Jenkins

Si quisieramos darle acceso a solo ciertos jobs, por ejemplo, de un area específica (compras, RH, etc.) es posible hacerlo. Desde la pestaña de “**manage and assign roles**” > **item roles** > **role to add** > **pattern** > **add** > **save**.

Item roles

Role	Pattern	Template	Credentials		Tarea		Ejecutar		Vistas		Repositorio de software (SCM)		Tag
			Read	Delete	Read	Delete	Create	Configure	Update	Reply	Update	Read	Delete
Job	“Job.*”		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						

Role to add

Pattern ?

Permission Template

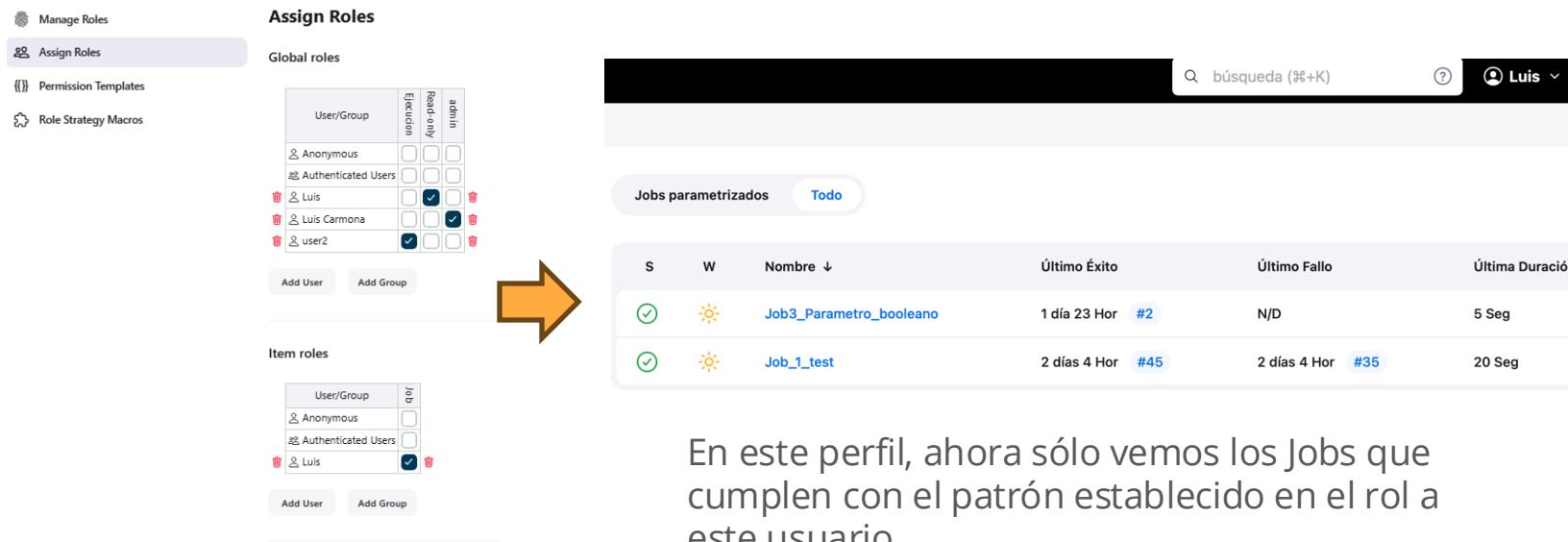
Add

Jobs parametrizados Todo

S	W	Nombre ↓
		Java_App_Maven
		job2_parametrizacion
		Job3_Parametro_booleano
		Job_1_test

# Jenkins

Ahora, para que el usuario tenga este rol, tenemos que asignarlo, pero en la sección “**assign roles**” > “**item roles**” en este caso el usuario de sólo lectura es **LuisECJ1021** > guardamos y revisamos



The screenshot shows the Jenkins Role Strategy interface. On the left, there are two sections: "Global roles" and "Item roles". In both sections, a user named "Luis" has the "Read Only" checkbox checked. An orange arrow points from the "Item roles" section to the right side of the screen, where a list of jobs is displayed.

**Assign Roles**

**Global roles**

User/Group	Execute	Read only	Admin
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Luis	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Luis Carmona	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
user2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Add User Add Group

**Item roles**

User/Group	Job
Anonymous	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>
Luis	<input checked="" type="checkbox"/>

Add User Add Group

**Jobs parametrizados** **Todo**

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
		Job3_Parametro_booleano	1 día 23 Hor #2	N/D	5 Seg
		Job_1_test	2 días 4 Hor #45	2 días 4 Hor #35	20 Seg

En este perfil, ahora sólo vemos los Jobs que cumplen con el patrón establecido en el rol a este usuario.

---

# Jenkins

## Ejercicio 7. Duración. 50

Desde la GUI de Jenkins, ejecuta lo siguiente:

Crea los siguientes roles dentro de Jenkins, recuerda hacerlo desde el perfil o usuario admin y que debe estar seleccionada en la autorización la opción de role-based strategy

Brindale la autorizacion o rol para poder realizar lo siguiente:

Lectura

Con otro rol le vamos a permitir que pueda

Crear

Creamos un tercer rol para :

Construir

Crea un nuevo usuario. (ponle el nombre que deseas, sólo recuerdalo) y asigne estos 3 roles.

Creamos un usuario nuevo y le asignamos que solo tenga acceso de lectura a los jobs con el patron X.

---

# Jenkins

## Job DSL. (Domain Specific Languaje)

Es un tipo de lenguaje de programación para un contexto particular, el contexto en este caso es jenkins y es para crear jobs, pipelines y todo lo que nos brinda Jenkins

Es una manera de programar/automatizar con código el Job.

Es el lenguaje de programación para Jenkins. Está basado en groovy.

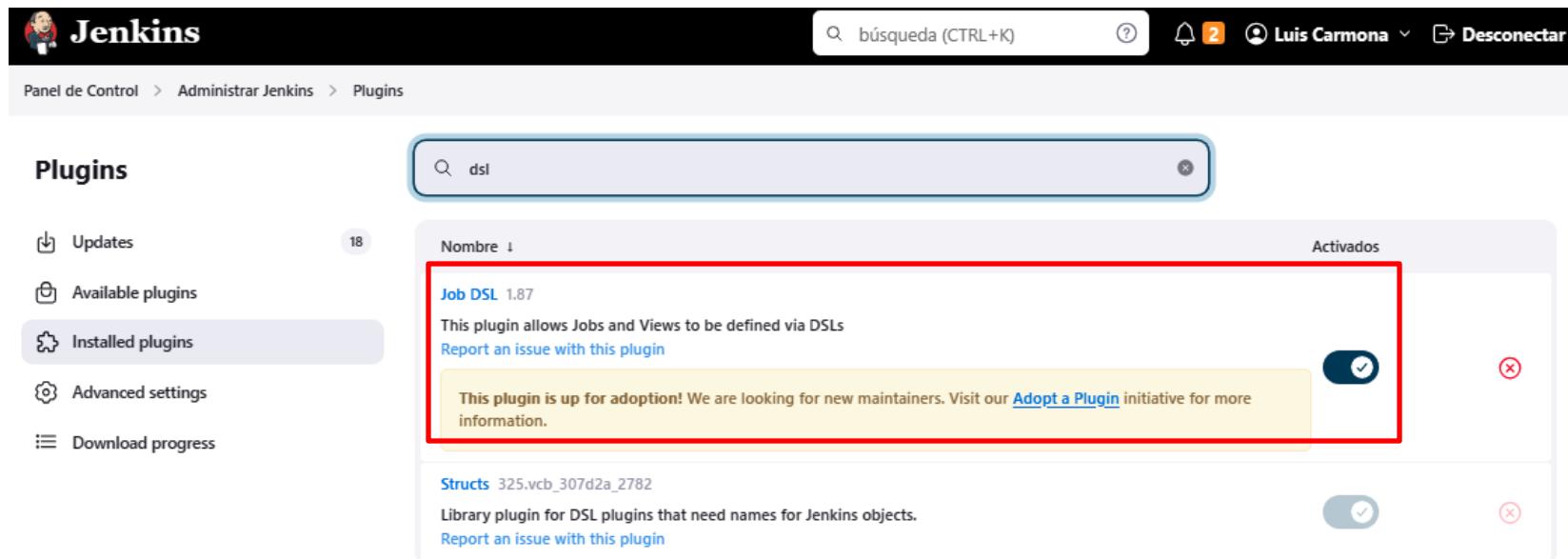
Para ello vamos a utilizar el plugin Job DSL que nos ayudará a definir y construir jobs en una forma programable con los archivos de código.

Se hace con un “lenguaje” llamado, Groovy Base Script basado en Java, pero mucho más simple.

Permite escribir scripts en un lenguaje declarativo DSL para definir, configurar y gestionar trabajos (jobs) y otras configuraciones relacionadas, en lugar de configurar los trabajos manualmente, se puede escribir scripts.

# Jenkins

## Instalación del plugin “job DSL”.



The screenshot shows the Jenkins Plugins management interface. A search bar at the top contains the text "dsl". On the left, a sidebar lists navigation options: Updates (18), Available plugins, **Installed plugins**, Advanced settings, and Download progress. The main area displays a table of available plugins. The "Job DSL" plugin is highlighted with a red box. It has a version of 1.87 and a brief description: "This plugin allows Jobs and Views to be defined via DSLs". Below the description is a link "Report an issue with this plugin". A yellow callout box with a checkmark icon states: "This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information." To the right of the plugin's name is a "Activated" column with a switch button that is checked. At the bottom of the table, another plugin entry for "Structs" is partially visible.

Nombre	Activados
Job DSL 1.87	<input checked="" type="checkbox"/>
Structs 325.vcb_307d2a_2782	<input checked="" type="checkbox"/>

## Jenkins

Para poder usar el **plugin DSL** es necesario crear un **seed Job**, que es un job de jenkins que ejecuta **código DSL** y por medio de este código se genera un nuevo Job con los requerimientos establecidos.

En otras palabras es un trabajo especial utilizado para generar dinámicamente otros trabajos de jenkins a través de un script.

El plugin DSL no permite crear el archivo configurable con los parámetros del nuevo Job.

---

## Jenkins

El uso de un Seed Job tiene varias ventajas:

**Automatización:** Permite crear trabajos de Jenkins automáticamente a partir de un script (generalmente un script de Job DSL o un script de tipo Pipeline).

**Repetibilidad:** Facilita la creación de trabajos de Jenkins de manera consistente en diferentes entornos, ya sea en entornos de desarrollo, pruebas o producción.

**Mantenimiento simplificado:** Si necesitas realizar cambios en la configuración de múltiples trabajos, puedes hacerlo de una sola vez editando el script del Seed Job y regenerando todos los trabajos afectados.

**Versionado de la configuración:** El Seed Job y sus scripts pueden ser versionados (por ejemplo, en un repositorio Git), lo que te permite tener un historial de cambios en la configuración de los trabajos.

**Escalabilidad:** Es útil cuando necesitas crear o modificar muchos trabajos similares en Jenkins. En lugar de hacerlo manualmente, puedes escribir un script que describa todos esos trabajos y regenerarlos automáticamente.

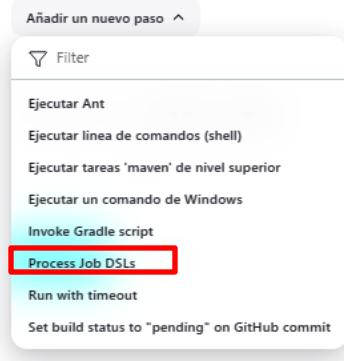
# Jenkins

Vamos a crear un nuevo Job.

Recuerden poner un nombre que les permita reconocer que hace el Job, y complementarlo con una descripción adecuada.

Una vez creado el Job, en la sección de ejecutar seleccionamos “**process Job DSL**”

## Build Steps



Añadir un nuevo paso ^

Filter

- Ejecutar Ant
- Ejecutar línea de comandos (shell)
- Ejecutar tareas 'maven' de nivel superior
- Ejecutar un comando de Windows
- Invoke Gradle script
- Process Job DSLs** (highlighted with a red box)
- Run with timeout
- Set build status to "pending" on GitHub commit

# Jenkins

## Build Steps

☰ Process Job DSLs

See [Job DSL API](#) for syntax reference.

Use the provided DSL script

Look on Filesystem

[DSL Scripts](#) ?

Ignore missing files: ?

Use Groovy Sandbox: ?

Action for existing jobs and views: ?

Ignore changes

Action for existing jobs and views managed by another seed job: ?

Fail if seed collision

Action for removed jobs: ?

Ignore

Action for removed views: ?

Ignore

Action for removed config files: ?

Ignore

Avanzado ▼

Añadir un nuevo paso ▼

Nos pregunta si en la interfaz misma hacemos el script o si está en algún repositorio como GitHub.

En este caso elejimos la primera y escribimos el siguiente código.

# Jenkins

## Build Steps

### Process Job DSLs

See [Job DSL API](#) for syntax reference.

- Use the provided DSL script
- Look on Filesystem
- [DSL Scripts](#) ?

Ignore missing files: ?

Use Groovy Sandbox: ?

Action for existing jobs and views: ?

Ignore changes

Action for existing jobs and views managed by another seed job: ?

Fail if seed collision

Action for removed jobs: ?

Ignore

Action for removed views: ?

Ignore

Action for removed config files: ?

Ignore

Avanzado ▾

Añadir un nuevo paso ▾

1. Nos pregunta si en la interfaz misma hacemos el script o si está en algún repositorio como GitHub.

2. En este caso elejimos la primera y escribimos **el código de la imagen > guardamos > Construimos**

## Build Steps

### Process Job DSLs

See [Job DSL API](#) for syntax reference.

- Use the provided DSL script

#### DSL Script ?

```
1 job('ejemplo_job_DSL'){\n2\n3 }
```

#### Look on Filesystem

Use Groovy Sandbox: ?

## Salida de consola

```
Started by user Luis Carmona\nRunning as SYSTEM\nBuilding in workspace /var/jenkins_home/workspace/Job4_DSL\nProcessing provided DSL script\nAdded items:\n    GeneratedJob{name='ejemplo-job-DSL'}\nFinished: SUCCESS
```

# Jenkins

Desde el panel de control, podemos ver los Jobs generados.

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración	
...	☀️	ejemplo-job-DSL	N/D	N/D	N/D	▶
✓	☁️	Java_App_Maven	1 día 0 Hor #6	1 día 0 Hor #5	3 Seg	▶
✓	☀️	job2_parametrizacion	2 días 2 Hor #5	N/D	20 Seg	▶
✓	☀️	Job3_Parametro _booleano	2 días 1 Hor #2	N/D	5 Seg	▶
✓	☁️	Job4_DSL	4 Min 58 Seg #13	14 Min #12	0.17 Seg	▶
✓	☀️	Job_1_test	2 días 5 Hor #45	2 días 6 Hor #35	20 Seg	▶

Este es el job generado, del seed job (Job4\_DSL).

# Jenkins

Ahora, una vez que creamos nuestro primer Seed Job y un Job a partir de éste, le empezaremos a agregar cosas.

En el Seed Job, en el código DSL agregamos lo siguiente **description ('Job DSL ejemplo para jenkins')**. > **Construimos la el job generado del Job DSL, para ver los cambios**

## Build Steps

≡ Process Job DSLs

See [Job DSL API](#) for syntax reference.

Use the provided DSL script

DSL Script ?

```
1 job('ejemplo-job-DSL') {  
2   description('Job DSL ejemplo para Jenkins')  
3 }
```



**General**

Descripción

Job DSL ejemplo para Jenkins

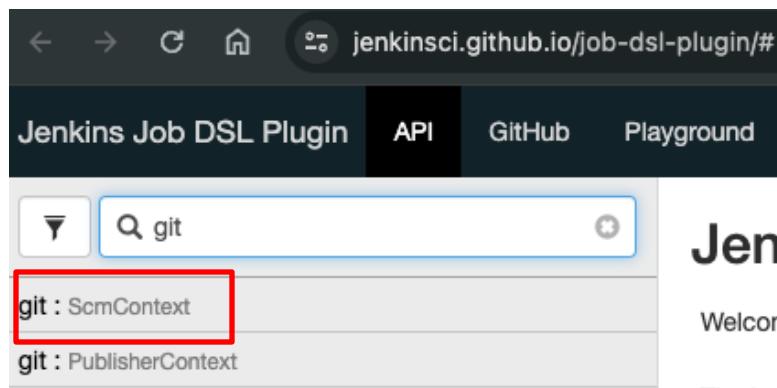
Plain text [Visualizar](#)

En el job, vemos como en la sección de descripción se agregó la descripción que pusimos mediante código

## Jenkins

Ahora, veremos como integrar GitHub desde DSL, específicamente buscamos para el SCM (Source Control Management).

Para ello vamos a la página <https://jenkinsci.github.io/job-dsl-plugin/#> donde se encuentra toda la información asociada a DSL.



The screenshot shows a web browser with the URL [jenkinsci.github.io/job-dsl-plugin/#](https://jenkinsci.github.io/job-dsl-plugin/#). The page title is "Jenkins Job DSL Plugin". There are tabs for "API", "GitHub", and "Playground". A search bar contains the query "git". Below the search bar, there is a list of results. The first result is highlighted with a red border and labeled "git : ScmContext". Other results include "git : PublisherContext".



```
// add user name and email options
job('example-3') {
    scm {
        git('git@git') { node -> // is hudson.plugins.git.GitSCM
            node / gitConfigName('DSL User')
            node / gitConfigEmail('me@me.com')
        }
    }
}
```

De la página, usamos el tercer ejemplo.

# Jenkins

Nos dirigimos al seed Job (**Job4\_DSL**) >configurar > Build Step > modificamos el código DSL > guardamos > construimos

## Build Steps

☰ Process Job DSLs

See [Job DSL API](#) for syntax reference.

Use the provided DSL script

DSL Script ?

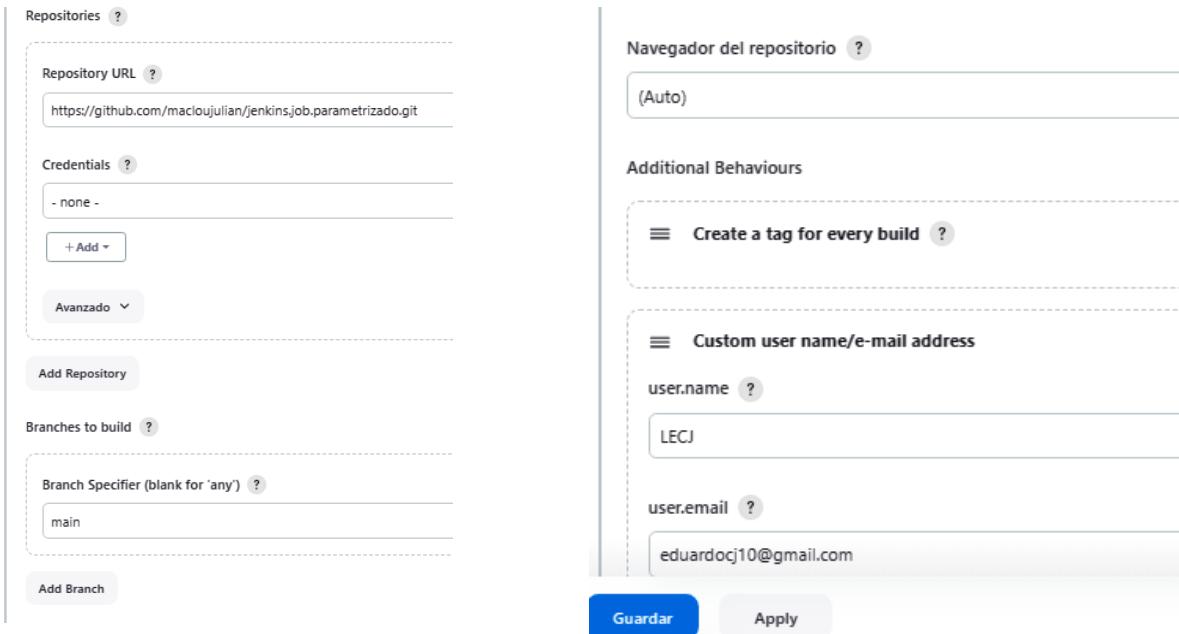
```
1 job('ejemplo-job-DSL') {
2     description('Job DSL ejemplo para Jenkins')
3
4     job('example-3') {
5         scm {
6             git('https://github.com/macloujulian/jenkins.job.parametrizado.git', 'main') { node -
7                 node / gitConfigName('LECJ')
8                 node / gitConfigEmail('eduardocj10@gmail.com')
9             }
10        }
11    }
}
```

✓ Salida de consola

```
Started by user Luis Carmona
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job4_DSL
Processing provided DSL script
Added items:
    GeneratedJob{name='Job_booleano_con_DSL'}
Existing items:
    GeneratedJob{name='ejemplo-job-DSL'}
Unreferenced items:
    GeneratedJob{name='example-3'}
Finished: SUCCESS
```

# Jenkins

Ingresamos al job generado desde el seed Job, en la sección > **configurar**



The screenshot shows the configuration page for a Jenkins job. On the left, under 'Repositories', there is a 'Repository URL' field containing 'https://github.com/mac loujulian/jenkins.job.parametrizado.git'. Below it are sections for 'Credentials' (set to '- none -') and 'Advanced' options. A 'Add Repository' button is also present. On the right, under 'Additional Behaviours', there are two sections: 'Create a tag for every build' (which is collapsed) and 'Custom user name/e-mail address' (which is expanded). In the 'user.name' field, 'LECJ' is entered. In the 'user.email' field, 'eduardocj10@gmail.com' is entered. At the bottom of the configuration page are 'Guardar' and 'Apply' buttons.

Podemos apreciar que se agregó toda la información que fue adicionada mediante el configo DSL. De donde descargar el código fuente... etc.

# Jenkins

Vamos a parametrizar un job, pero no desde la GUI de Jenkins, si no con DSL.

En la página donde encontramos el ejemplo anterior para usar DSL, ahora buscamos **parameters** > **freeStyleJob**



A screenshot of a web browser displaying the Jenkins Job DSL Plugin documentation. The URL in the address bar is `jenkinsci.github.io/job-dsl`. The page shows a search bar with 'parameters' typed into it. Below the search bar is a list of Java class names, with 'parameters' and 'FreeStyleJob' highlighted in blue. To the right of the list is a detailed description of the 'parameters' block. It starts with the syntax `parameters { ... }`, followed by a description: 'Allows to parameterize the job.' Below this is a section titled 'Examples' containing a Groovy script snippet:

```
job('example') {  
    parameters {  
        booleanParam('FLAG', true)  
        choiceParam('OPTION', ['option 1 (default)', 'option 2', 'option 3'])  
    }  
}
```

An orange arrow points from the left side of the slide towards the 'parameters' section of the documentation. The code example in the documentation is highlighted with a red box.

¿Se recuerdan que estos parametros los agregamos desde la GUI de Jenkins?...

# Jenkins

Vamos a implementar el DSL para parametrizar un Job, para ello vamos a Jenkins y entramos a la configuración del Job booleano para ver su configuración.

Esta ejecución debe parametrizarse ?

Parámetro de cadena ?

Nombre ?

nombre

Valor por defecto ?

Luis

Descripción ?

Plain text [Visualizar](#)

Trim the string ?

Elección ?

Nombre ?

planeta

Opciones ?

Mercurio  
Venus  
Marte  
Jupiter  
Saturno

Descripción ?

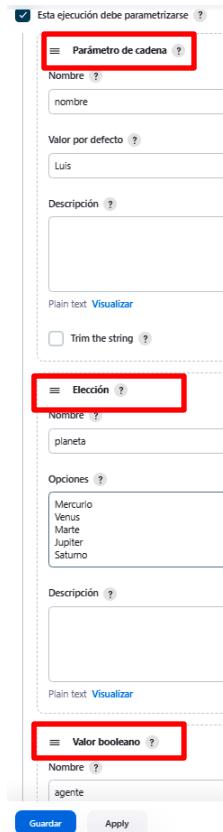
Plain text [Visualizar](#)

Valor booleano ?

Nombre ?

agente

Guardar Apply



# Jenkins

Abrimos el Seed Job, en mi caso, **Job4\_DSL > configurar > build steps**. Agregamos el código del recuadro rojo.

```
job('ejemplo-job-DSL') {
    description('Job DSL ejemplo para Jenkins')

    job('Job_booleano_con_DSL') {
        scm {
            git('https://github.com/macloujulian/jenkins.job.parametrizado.git', 'main') { node ->
                node / gitConfigName('LECJ')
                node / gitConfigEmail('eduardocj10@gmail.com')
            }
        parameters{
            StringParam('nombre', defaultValue = 'Luis', description = 'parametro de cadena para el Job booleano')
            choiceParam('planeta', ['mercurio', 'venus', 'marte', 'jupiter'])
            booleanParam('agente', false)
        }
    }
}
```

# Jenkins

Para que quede de la siguiente manera > guardamos y construimos.

## Build Steps

### ≡ Process Job DSLs

See [Job DSL API](#) for syntax reference.

- Use the provided DSL script

#### DSL Script [?](#)

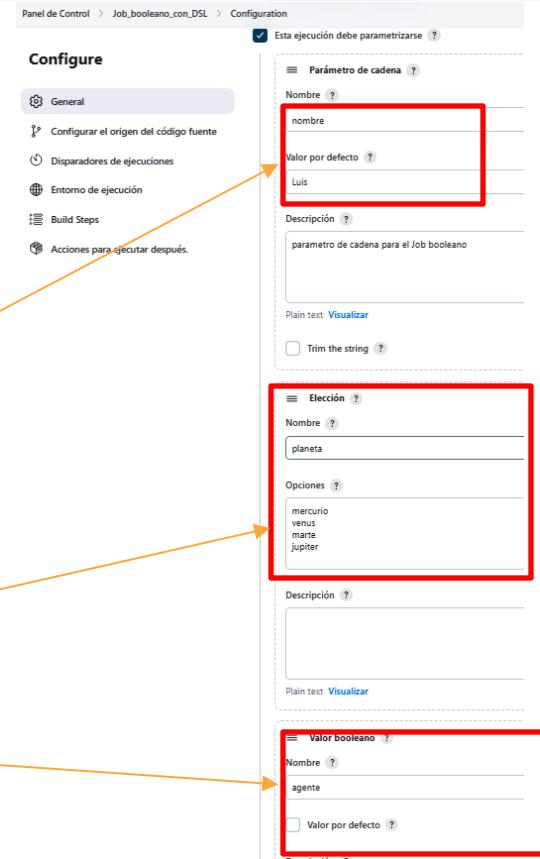
```
1 job('ejemplo-job-DSL') {
2     description('Job DSL ejemplo para Jenkins')
3
4     job('Job_booleano_con_DSL') {
5         scm {
6             git('https://github.com/maclojulian/jenkins.job.parametrizado.git', 'main') { node ->
7                 node / gitConfigName('LECJ')
8                 node / gitConfigEmail('eduardocj10@gmail.com')
9             }
10            parameters{
11                StringParam('nombre', defaultValue = 'Luis', description = 'parametro de cadena para el Job booleano')
12                choiceParam('planeta', ['mercurio', 'venus', 'marte', 'jupiter'])
13                booleanParam('agente', false)
14            }
15        }
16    }
17 }
```

# Jenkins

Ahora entramos al job que comentamos y podemos observar que todo lo que agregamos en las líneas de código, ahora se ven reflejadas graficamente en el job.

```
job('ejemplo-job-DSL') {
    description('Job DLS ejemplo para Jenkins')

    job('Job_booleano_con_DSL') {
        scm {
            git('https://github.com/mac loujulian/jenkins.job.parametrizado.git', 'main') { node ->
                node / gitConfigName('LECJ')
                node / gitConfigEmail('eduardocj10@gmail.com')
            }
        }
        parameters{
            StringParam('nombre', defaultValue = 'Luis', description = 'parametro de cadena para el Job booleano')
            choiceParam('planeta', ['mercurio', 'venus', 'marte', 'jupiter'])
            booleanParam('agente', false)
        }
    }
}
```

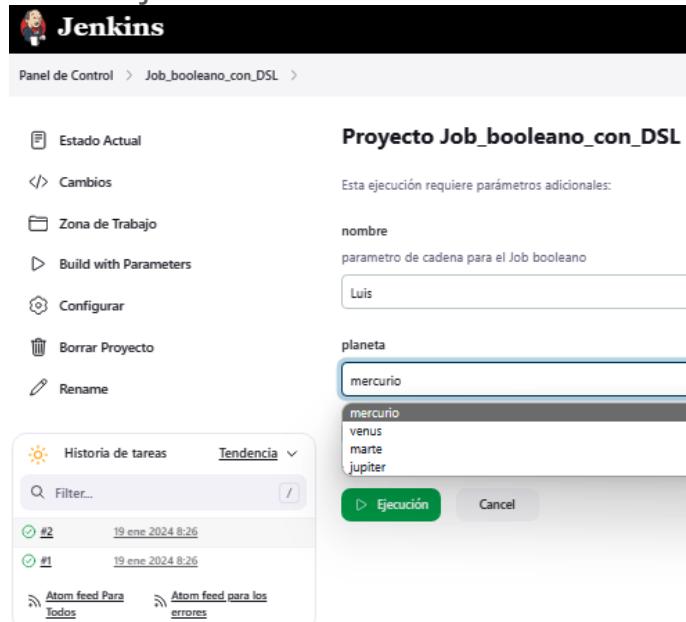


The screenshot shows the Jenkins job configuration for 'Job\_booleano\_con\_DSL'. It includes three parameter definitions:

- Parámetro de cadena**: Nombre (Luis) - Default value: Luis. Description: parametro de cadena para el Job booleano.
- Eleción**: planeta (mercurio, venus, marte, jupiter)
- Valor booleano**: agente (false)

# Jenkins

Si construimos el job, debemos obtener el mismo resultado que cuando lo hicimos directamente desde jenkins.

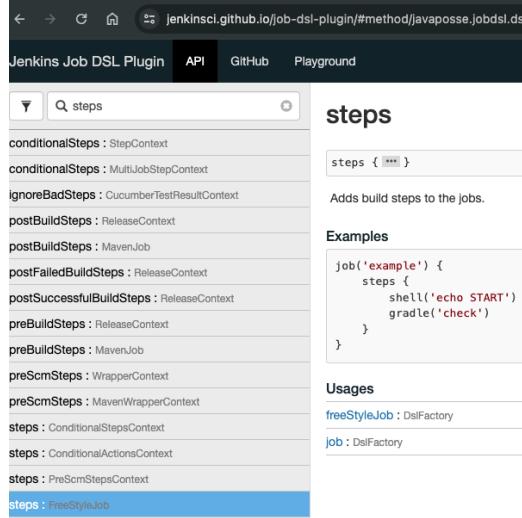


The screenshot shows the Jenkins interface for a job named "Job\_booleano\_con\_DSL". On the left, there's a sidebar with options like "Estado Actual", "Cambios", "Zona de Trabajo", "Build with Parameters", "Configurar", "Borrar Proyecto", and "Rename". Below that is a "Historia de tareas" section showing two recent builds (#2 and #1) both completed on "19 ene 2024 8:26". At the bottom are links for "Atom feed Para Todos" and "Atom feed para los errores". The main content area is titled "Proyecto Job\_booleano\_con\_DSL". It says "Esta ejecución requiere parámetros adicionales:" followed by input fields for "nombre" (with "Luis" typed) and "planeta" (with "mercurio" typed). A dropdown menu also lists "mercurio", "venus", "marte", and "jupiter". At the bottom right of the main area are buttons for "Ejecución" and "Cancel".

# Jenkins

Ahora, como también lo hicimos desde jenkins, al definir las etapas por las que debe pasar un Job, lo podemos hacer desde DSL.

En la página de DSL buscamos “steps” y buscamos el que sea para “freeStyleJob” que es el tipo de Job que hemos usado.



The screenshot shows a web browser displaying the Jenkins Job DSL Plugin documentation at [jenkinsci.github.io/job-dsl-plugin/#method/javaposse.jobdsl.dsl](https://jenkinsci.github.io/job-dsl-plugin/#method/javaposse.jobdsl.dsl). The page title is "Jenkins Job DSL Plugin". The search bar contains "steps". The main content area is titled "steps" and contains the following information:

- Definition:** `steps { ... }`
- Description:** Adds build steps to the jobs.
- Examples:**

```
job('example') {
    steps {
        shell('echo START')
        gradle('check')
    }
}
```
- Usages:**
  - `freeStyleJob : DslFactory`
  - `job : DslFactory`

# Jenkins

En el seed Job, empezamos a configurarlo agregando el código del recuadro, guardamos y construimos.

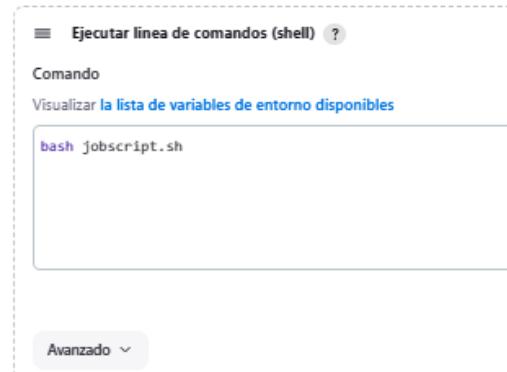
DSL Script [?](#)

```
1 job('ejemplo-job-DSL') {
2     description('Job DSL ejemplo para Jenkins')
3
4     job('Job_booleano_con_DSL') {
5         scm {
6             git('https://github.com/mac loujulian/jenkins.job.parametrizado.git', 'main') { node -
7                 node / gitConfigName('LECJ')
8                 node / gitConfigEmail('eduardocj10@gmail.com')
9             }
10            parameters{
11                stringParam('nombre', defaultValue = 'luis', description = 'parametro de cadena para el Job booleano')
12                choiceParam('planeta', ['mercurio', 'venus', 'marte', 'jupiter'])
13                booleanParam('agente', false)
14            }
15            steps{
16                shell ("bash jobsript.sh")
17            }
18        }
19    }
20 }
```

# Jenkins

Vamos a la sección del shell y vemos que se ve reflejado el comando que pusimos a través de código en la lámina pasada.

## Build Steps



The screenshot shows the Jenkins build steps configuration for a job. It displays a single step: "Ejecutar línea de comandos (shell)". The command entered is "bash jobsript.sh". There is also an "Avanzado" button and a link to "Añadir un nuevo paso".

Ejecutar línea de comandos (shell) ?

Comando

Visualizar la lista de variables de entorno disponibles

```
bash jobsript.sh
```

Avanzado ▾

Añadir un nuevo paso ▾

# Jenkins

Podemos configurar el poder recibir notificaciones acerca de lo que sucede en Jenkins. En este caso la más común y práctica, por correo electrónico, esto se hace con los “**publishers**” así que vamos de nuevo a la página de DSL y buscamos publishers.

Jenkins Job DSL Plugin API GitHub Playground

publisher

postBuildPublishers : ReleaseContext  
postFailedBuildPublishers : ReleaseContext  
postSuccessfulBuildPublishers : ReleaseContext  
preBuildPublishers : ReleaseContext  
**publishers** : ConditionalActionsContext  
publishers : JoinTriggerContext  
**publishers** : FreeStyleJob    
publishers : IvyJob  
publishers : MatrixJob  
publishers : MavenJob  
publishers : MultiJob  
testDataPublishers : ArchiveJUnitContext

## publishers

**publishers { ... }**  

Adds post-build actions to the job.

### Examples

```
job('example') {  
    publishers {  
        archiveArtifacts('build/test-output/**/*.html')  
        archiveJunit('**/target/surefire-reports/*.xml')  
    }  
}
```

### Usages

freeStyleJob : DslFactory  
job : DslFactory

Dado que hemos creado proyectos de estilo libre, seleccionamos el mismo comando, damos clic en los ... de publishers Seleccionamos “mailer”.

```
// Sends email notifications.  
mailer(String recipients, Boolean dontNo
```

# Jenkins

En la sección de configuracion del “seed Job” vamos nuevamente a agregar el código del recuadro rojo para configurar por este medio las notificaciones de correo electrónico.

```
job('ejemplo-job-DSL') {  
    description('Job DLS ejemplo para Jenkins')  
  
    job('Job_booleano_con_DSL') {  
        scm {  
            git('https://github.com/mac loujulian/jenkins.job.parametrizado.git', 'main') { node ->  
                node / gitConfigName('LECJ')  
                node / gitConfigEmail('eduardocj10@gmail.com')  
            }  
            parameters{  
                stringParam('nombre', defaultValue = 'Luis', description = 'parametro de cadena para el Job booleano')  
                choiceParam('planeta', ['mercurio', 'venus', 'marte', 'jupiter'])  
                booleanParam('agente', false)  
            }  
            steps{  
                shell ("bash jobscrip.t.sh")  
            }  
            publishers{  
                mailer('eduardocj10@gmail.com', true, true)  
            }  
        }  
    }  
}
```

# Jenkins

Quedando de la siguiente manera. **Guardamos > construimos**

Use the provided DSL script

DSL Script [?](#)

```
1 job('ejemplo-job-DSL') {
2   description('Job DLS ejemplo para Jenkins')
3
4   job('Job_booleano_con_DSL') {
5     scm {
6       git('https://github.com/macloujulian/jenkins.job.parametrizado.git', 'main') { node ->
7         node / gitConfigName('LECJ')
8         node / gitConfigEmail('eduardocj10@gmail.com')
9       }
10    parameters{
11      stringParam('nombre', defaultValue = 'Luis', description = 'parametro de cadena para el Job booleano')
12      choiceParam('planeta', ['mercurio', 'venus', 'marte', 'jupiter'])
13      booleanParam('agente', false)
14    }
15    steps{
16      shell ("bash jobscrip.sh")
17    }
18    publishers{
19      mailer('eduardocj10@gmail.com', true, true)
20    }
21  }
22}
23}
```

# Jenkins

Nos vamos al Job que hemos estado usando, y vemos los cambios.

Acciones para ejecutar después.

≡ Notificación por correo ? ×

**Destinatarios**  
Lista de destinatarios separadas por un espacio en blanco. El correo será enviado siempre que una ejecución falle.

eduardocj10@gmail.com

Enviar correo para todos las ejecuciones con resultado inestable

?  Enviar correos individualizados a las personas que rompan el proyecto

Añadir una acción ▾

---

# Jenkins

## Ejercicio.

Para usar DSL, crea el seed Job, asignale el nombre que deseas siguiendo la nomenclatura de los previos.

Recuerda que el código DSL lo puedes encontrar en la siguiente página: <https://jenkinsci.github.io/job-dsl-plugin/#path/job>

Al job generado (no seed job) agregale una descripción con código DSL

Agrega al job el código de github que usamos en el ejercicio 6

Agrega los tres parámetros vistos:

Uno de cadena con tu nombre y apellido

Uno de selección con 5 elementos

Uno booleano donde el valor por defecto sea falso

Con DSL configura que se mande un correo electrónico cuando acabe la construcción deñ Job

# Jenkins

Jenkinsfile.

Es una archivo que contiene el flujo completo del proceso y puede ser manejado con control de version (SCM). Se utiliza para la construcción del pipeline

```
pipeline {  
    agent any  
  
    stages[  
        stage("Build"){  
            steps[  
                echo "Construyendo la aplicación"  
            ]  
        }  
        stage("Test"){  
            steps[  
                echo "Ejecutando pruebas a la aplicación"  
            ]  
        }  
        stage("Deploy"){  
            steps[  
                echo "Desplegando aplicación al área X"  
            ]  
        }  
    ]  
}
```

Pipeline: Es el bloque que define todo el contenido, con el proceso completo de las etapas.

Agent: indica a jenkins que asigne un ejecutor a las compilaciones y un WS para el pipeline

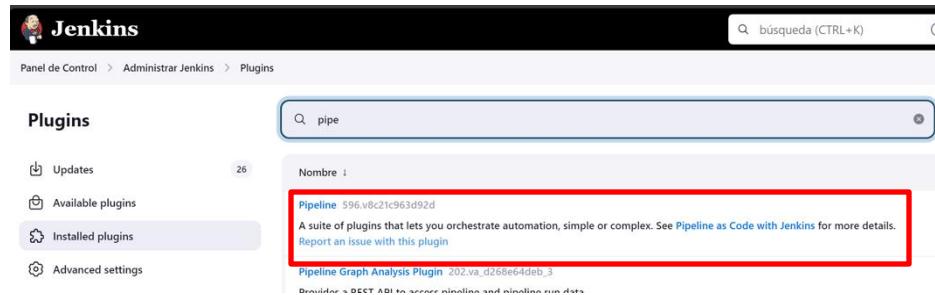
Steps: son las tareas o comandos que ejecutados de forma secuencial conforman la “lógica del flujo de trabajo”. Se deben especificar en el orden que queremos se ejecuten.

Stage: Son las etapas o pasos lógicos en que se dividen los flujos de trabajo

Podemos ver que la ultima etapa es deploy, pero no es necesario que sea asi, es flexible y las pueden modificar adaptar a sus necesidades

# Jenkins

Antes de continuar es necesario revisar si tenemos instalado el plugin pipeline, de no tenerlo, instalalo como lo hemos hecho previamente.



The screenshot shows the Jenkins Plugins management interface. The top navigation bar includes the Jenkins logo, a search bar with placeholder "búsqueda (CTRL+K)", and a help icon. Below the navigation is a breadcrumb trail: "Panel de Control > Administrar Jenkins > Plugins". The main area has a title "Plugins" and a sidebar with links: "Updates" (26), "Available plugins" (selected), "Installed plugins" (disabled), and "Advanced settings". A search bar at the top of the main content area contains the text "pipe". The results list shows the "Pipeline" plugin, which is highlighted with a red box. The plugin details are as follows:

- Nombre:** Pipeline 596.v8c21c963d92d
- A suite of plugins that lets you orchestrate automation, simple or complex. See [Pipeline as Code with Jenkins](#) for more details.
- [Report an issue with this plugin](#)

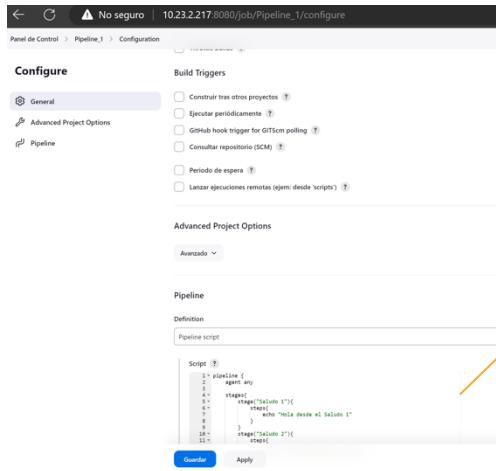
Below the Pipeline plugin, another plugin is listed:

- Nombre:** Pipeline Graph Analysis Plugin 202.va\_d268e64deb\_3
- Provides a REST API to access pipeline and pipeline run data.

# Jenkins

Pipeline.

Para crear un pipeline creamos un nuevo proyecto, pero no de estilo libre, sino de pipeline, y podemos ver que tienen opciones similares, pero al final, no hay build steps, ni otras opciones de un proyecto de estilo libre, en su lugar tenemos la opcion de pipeline como se ve en la imagen



Y justo aquí, en pipeline es donde vamos a escribir el código para poder automatizar las tareas, donde tenemos un par de opciones, escribir directamente el código o brindarle un archivo con el código (SCM)

# Jenkins

Pipeline.

Para crear un pipeline creamos un nuevo proyecto, pero no de estilo libre, sino de pipeline

Pipeline

Definition

Pipeline script

Script ?

```
1 * pipeline {  
2     agent any  
3  
4     stages{  
5         stage("Saludo 1"){  
6             steps{  
7                 echo "Hola desde el Saludo 1"  
8             }  
9         }  
10    }  
11 }  
12 }
```



Use Groovy Sandbox ?

```
pipeline {      #palabra reservada Pipeline  
    agent any   #donde se ejecutara  
  
    stages{      #que escenarios o etapas tiene el pipe  
        stage("Saludo 1"){ #este es el escenario 1  
            steps{      #estas son las tareas de la etapa  
                echo "Hola desde el Saludo 1"  
            }  
        }  
    }  
}
```

# Jenkins

## Pipeline

Vemos que despues de construir el pipeline en la sección de estatus tenemos una sección grafica nueva y diferente.

### Pipeline\_1

#### Stage View



Saludo 1 es el stage



Si posicionamos el cursor sobre, vemos que no aparece el recuadro logs y sobre el el mensaje de la parte superior.

# Jenkins

## Pipeline

La vista o salida por consola es la siguiente

### ✓ Salida de consola

```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Pipeline_1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Saludo 1)
[Pipeline] echo
Hola desde el Saludo 1
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Vemos que las secciones cambian, dado que es un proyecto tambien diferente, pero, ¿en que sentido es diferente?...

Los pipelines se usan para "dividir" un proceso largo, complejo y/o completo en tareas o actividades (más pequeñas) de forma ordenada

Ejemplo, si se require hacer un despliegue, puede que este tenga que realizar:

Descargar el código  
Realizar pruebas unitarias  
Limpiar caché  
Push al server  
Etc.

Despliegue

# Jenkins

## Pipeline

Volvemos a editar el pipeline que tenemos, para agregar otro “stage” con dos acciones, quedando el código de la siguiente manera

### Pipeline

#### Definition

##### Pipeline script

###### Script ?

```
1 * pipeline {
2     agent any
3
4     stages{
5         stage("Saludo 1"){
6             steps{
7                 echo "Hola desde el Saludo 1"
8             }
9         }
10        stage("Saludo 2"){
11            steps{
12                echo "Hola desde Saludo 2, paso 1"
13                echo "Hola desde Saludo 2, paso 2"
14            }
15        }
16    }
17}
18 }
```

```
pipeline {
    agent any

    stages{
        stage("Saludo 1"){
            steps{
                echo "Hola desde el Saludo 1"
            }
        }
        stage("Saludo 2"){
            steps{
                echo "Hola desde Saludo 2, paso 1"
                echo "Hola desde Saludo 2, paso 2"
            }
        }
    }
}
```

# Jenkins

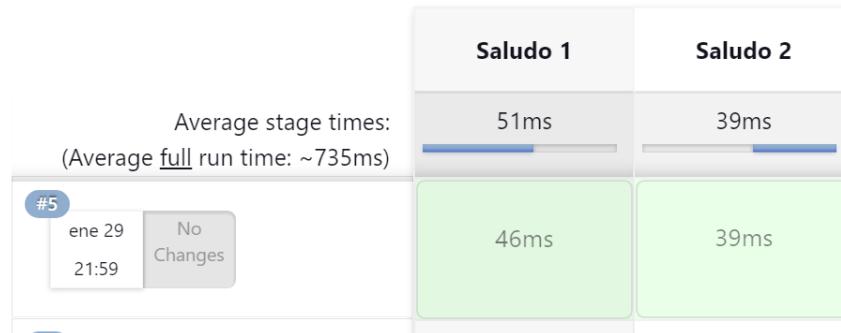
## Pipeline

Veamos de nuevo la salida por consola y la sección gráfica para ver cómo se comporta.

### Salida de consola

```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Pipeline_1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Saludo 1)
[Pipeline] echo
Hola desde el Saludo 1
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Saludo 2)
[Pipeline] echo
Hola desde Saludo 2, paso 1
[Pipeline] echo
Hola desde Saludo 2, paso 2
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

### Stage View



# Jenkins

## Pipeline

Veamos de nuevo la salida por consola y la sección gráfica para ver cómo se comporta.

**Stage Logs (Saludo 1)**

Print Message -- Hola desde el Saludo 1 (self time 2ms)

```
Hola desde el Saludo 1
```

**Stage View**

Average stage times:  
(Average full run time: ~735ms)

#5	ene 29 21:59	No Changes
<b>Success</b> Saludo 1	Logs	Saludo 2
46ms	39ms	39ms

**Pipeline**

**Definition**

Pipeline script

**Script**

```
1 * pipeline {
2     agent any
3
4     stages{
5         stage("Saludo 1"){
6             steps{
7                 echo "Hola desde el Saludo 1"
8             }
9         }
10    }
11 }
12 }
```

Use Groovy Sandbox

# Jenkins

## Pipeline

Veamos de nuevo la salida por consola y la sección gráfica para ver cómo se comporta.



Pipeline

Definition

Pipeline script

Script ?

```
1 > pipeline {  
2   agent any  
3  
4   stages{  
5     stage("Saludo 1"){  
6       steps{  
7         echo "Hola desde el Saludo 1"  
8       }  
9     }  
10    stage("Saludo 2"){  
11      steps{  
12        echo "Hola desde Saludo 2, paso 1"  
13        echo "Hola desde Saludo 2, paso 2"  
14      }  
15    }  
16  }  
17}  
18 }
```

Use Groovy Sandbox ?

Es importante ver como divide el job no solamente en “stages”, sino también en “steps”.

# Jenkins

Veamos un ejemplo más, con stages o etapas más “reales”

Pipeline

Definition

Pipeline script

```
1 ~ pipeline {  
2 ~   agent any  
3 ~   stages {  
4 ~     stage('Build') {  
5 ~       steps {  
6 ~         echo 'Construyendo la Aplicación'  
7 ~       }  
8 ~     }  
9 ~     stage('Test') {  
10 ~       steps {  
11 ~         echo 'Arranca el proceso de pruebas unitarias'  
12 ~       }  
13 ~     }  
14 ~     stage('Deploy') {  
15 ~       steps {  
16 ~         echo 'Desplegando al área de desarrollo'  
17 ~       }  
18 ~     }  
19 ~   }  
20 ~ }  
21 ~
```

Pipe\_1

Stage View



Enlaces permanentes

Use Groovy Sandbox 

# Jenkins

Si observamos, en los pipelines hemos visto que solo tenemos una accion en la sección de steps

Pipeline

Definition

Pipeline script

Script ?

```
1 pipeline {
2   agent any
3   stages {
4     stage('Build') {
5       steps {
6         echo 'Construyendo la Aplicación'
7       }
8     }
9     stage('Test') {
10    steps {
11      echo 'Arranca el proceso de pruebas unitarias'
12    }
13  }
14  stage('Deploy') {
15    steps {
16      echo 'Desplegando al área de desarrollo'
17    }
18  }
19 }
20 }
```



Use Groovy Sandbox ?

¿Qué pasa si queremos o necesitamos agregar más de 1 accion en steps?

# Jenkins

Es posible hacerlo, con la siguiente sintaxis

```
pipeline {  
    agent any  
  
    stages{  
        stage("Build"){  
            steps{  
                sh 'echo "Construyendo la aplicación"'  
                sh '''  
                    echo "Segundo paso en la etapa de build"  
                    pwd  
                    '''  
            }  
        }  
    }  
}
```

# Jenkins

Tambien tenemos alternativas, como retry, que como el nombre indica, nos permite reintentar n veces la ejecucion de un step dentro de un stage, a continuacion un ejemplo:

```
pipeline {  
    agent any  
  
    stages{  
        stage("Retry"){  
            steps{  
                retry(5){  
                    sh 'no funcionará'  
                }  
            }  
        }  
    }  
}
```

Aquí estamos definiendo que el step (sh 'no funciona') se reintente 5 veces dentro del mismo build o ejecucion, antes de pasar a la siguiente linea/acción/etapa

# Jenkins

Podemos ver de manera gráfica y la salida por consola los resultados

**Stage Logs (Retry)**

- Shell Script -- no funcionará (self time 289ms)
- Shell Script -- no funcionará (self time 275ms)
- Shell Script -- no funcionará (self time 272ms)
- Shell Script -- no funcionará (self time 269ms)
- Shell Script -- no funcionará (self time 266ms)

▷ Construir ahora

⚙ Configurar

trash Borrar Pipeline

🔍 Full Stage View

✍ Rename

ⓘ Pipeline Syntax

cloud Historia de tareas

Tendencia

Average stage times:

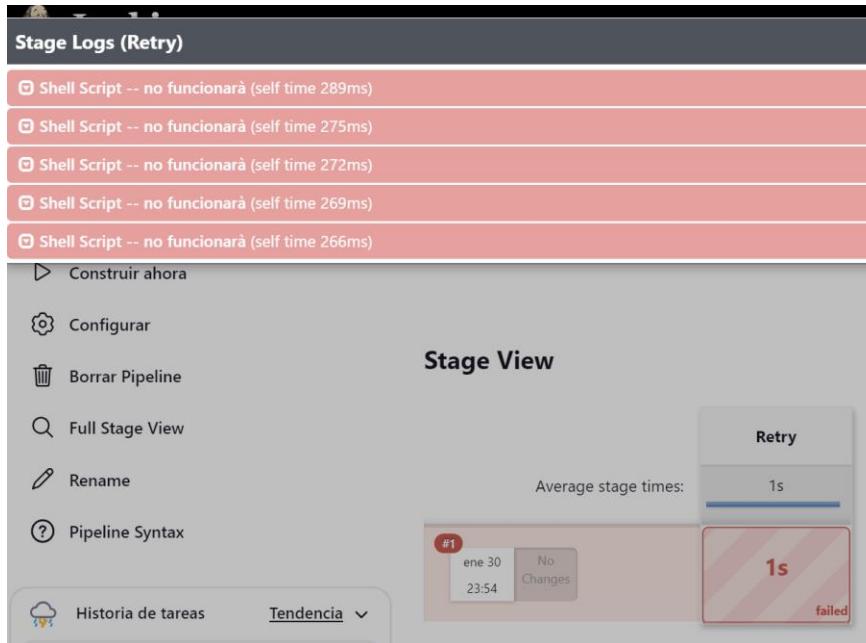
Stage View

Retry

1s

#1 ene 30 23:54 No Changes

1s failed



```
+ no funcionará [REDACTED]
/var/jenkins_home/workspace/pipeline_retry@tmp/durable-de40dc3f/script.sh: 1: no: not found
[Pipeline] }
ERROR: script returned exit code 127
Retrying
[Pipeline] {
[Pipeline] sh
+ no funcionará [REDACTED]
/var/jenkins_home/workspace/pipeline_retry@tmp/durable-35bcd66/script.sh: 1: no: not found
[Pipeline] }
ERROR: script returned exit code 127
Retrying
[Pipeline] {
[Pipeline] sh
+ no funcionará [REDACTED]
/var/jenkins_home/workspace/pipeline_retry@tmp/durable-815ee11e/script.sh: 1: no: not found
[Pipeline] }
ERROR: script returned exit code 127
Retrying
[Pipeline] {
[Pipeline] sh
+ no funcionará [REDACTED]
/var/jenkins_home/workspace/pipeline_retry@tmp/durable-be3b3401/script.sh: 1: no: not found
[Pipeline] }
ERROR: script returned exit code 127
Retrying
[Pipeline] {
[Pipeline] sh
+ no funcionará [REDACTED]
/var/jenkins_home/workspace/pipeline_retry@tmp/durable-4ebf657e/script.sh: 1: no: not found
[Pipeline] }
[Pipeline] // retry
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 127
Finished: FAILURE
```

# Jenkins

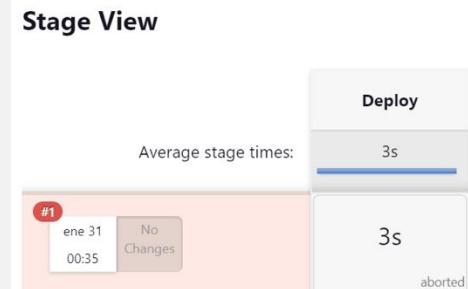
Por otro lado, cuando un proceso sea sensible al tiempo, es decir, que se ejecute en un lapso menor a x, y este no sucede, que jenkins marque error porque el “tiempo” se agotó.

Veamos un ejemplo:

```
pipeline {
    agent any

    stages{
        stage("Deploy"){
            steps{
                retry(3){
                    sh 'echo Hola, no hará retry'
                }
                timeout(time: 3, unit: 'SECONDS'){
                    sleep 5
                }
            }
        }
    }
}
```

```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_timeout
[Pipeline] {
[Pipeline] stage
[Pipeline] {
  [Pipeline] Deploy
  [Pipeline] retry
  [Pipeline] {
  [Pipeline] sh
  + echo Hola, no hará retry
  Hola, no hará retry
  [Pipeline] }
  [Pipeline] // retry
  [Pipeline] timeout
  Timeout set to expire in 3 sec
  [Pipeline] {
  [Pipeline] sleep
  Sleeping for 5 sec
  Cancelling nested steps due to timeout
  [Pipeline] }
  [Pipeline] // timeout
  [Pipeline] }
  [Pipeline] // stage
  [Pipeline] {
  [Pipeline] // node
  [Pipeline] End of Pipeline
Timeout has been exceeded
org.jenkinsci.plugins.workflow.actions.ErrorAction$ErrorId: d8636d2c-5a06-4a46-8de7-894ab12e888c
Finished: ABORTED
```



# Jenkins

Como podrán imaginar, estos comandos o acciones se pueden mezclar o pueden complementarse, como se puede ver en el siguiente código.

```
pipeline {  
    agent any  
  
    stages{  
        stage("Deploy"){  
            steps{  
                timeout(time: 3, unit:'SECONDS '){  
                    retry(5) {  
                        sh 'sleep 5'  
                    }  
                }  
            }  
        }  
    }  
}
```

En el recuadro rojo, estamos diciendo que el step sleep 5 tiene un timeout de 3, si lo excede que lo reintente (retry) 5 veces, antes de marcar error

# Jenkins

Por otro lado, tambien tenemos acciones posteriores a la ejecución, y que normalmente estan sujetas a alguna condicion. Para ello tenemos un ejemplo como el siguiente código:

```

66 pipeline {
67   agent any
68
69   stages{
70     stage("Test"){
71       steps{
72         sh 'echo fail; exit 1'
73       }
74     }
75   }
76 }
77
78 post{
79   always{
80     echo 'siempre me ejecutare'
81   }
82   success{
83     echo 'solo me ejecuto si el build no falla'
84   }
85   failure{
86     echo 'solo me ejecutare si el build falla'
87   }
88   unstable{
89     echo 'solo me ejecutare si se marca como inestable'
90   }
91   changed{
92     echo 'el pipe estaba fallando, pero ya no, o se modifico'
93   }
94 }
95 }
```

## ✖ Salida de consola



```

Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_post
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] sh
+ echo fail
fail
+ exit 1
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
siempre me ejecutare
[Pipeline] echo
solo me ejecutare si el build falla
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 1
Finished: FAILURE

```

# Jenkins

En el caso anterior el código tenia intencionalmente un error, pero en este segundo caso, ya no, entonces las opciones que se ejecutan tambien cambian, veamos cuales:

```

102 pipeline {
103     agent any
104
105     stages{
106         stage("Test"){
107             steps{
108                 sh 'echo Hola'
109             }
110         }
111     }
112 }
113
114 post{
115     always{
116         echo 'siempre me ejecutare'
117     }
118     success{
119         echo 'solo me ejecuto si el build no falla'
120     }
121     failure{
122         echo 'solo me ejecutare si el build falla'
123     }
124     unstable{
125         echo 'solo me ejecutare si se marca como inestable'
126     }
127     changed{
128         echo 'el pipe estaba fallando, pero ya no, o se modifico'
129     }
130 }

```

## ✓ Salida de consola

```

Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_post
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] sh
+ echo Hola
Hola
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
siempre me ejecutare
[Pipeline] echo
el pipe estaba fallando, pero ya no, o se modifico
[Pipeline] echo
solo me ejecuto si el build no falla
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

# Jenkins

Todas las acciones post, como podrán imaginar se pueden mezclar, cambiar de posición, usar mas de 1 vez, y eso nos brinda una amplia gama de posibilidades, alternativas y opciones al momento de ejecutar...

Status  pipe\_post

</> Changes

▷ Construir ahora

⚙️ Configurar

trash Borrar Pipeline

🔍 Full Stage View

✍️ Rename

❓ Pipeline Syntax

Cloud Historia de tareas Tendencia 

Filter... 

#3 31 ene 2024 11:35

#2 31 ene 2024 11:25

#1 31 ene 2024 11:24

Atom feed para Todos Atom feed para los errores

### Stage View

Test	Declarative: Post Actions
Average stage times: (Average full run time: ~581ms)	294ms
#3 ene 31 04:35 No Changes	291ms
#2 ene 31 04:25 No Changes	23ms
#1 ene 31 04:24 No Changes	299ms

# Jenkins

Tenemos tambien las variables de entorno en pipeline y son importantes, y de gran utilidad.  
Veamos un ejemplo en el siguiente bloque de código.

```
pipeline {  
    agent any  
  
    environment{  
        NOMBRE= 'Luis'  
        APELLIDO= 'Carmona'  
    }  
    stages{  
        stage("Build"){  
            steps{  
                sh 'echo $NOMBRE $APELLIDO'  
            }  
        }  
    }  
}
```

Estamos definiendo dos variables de entorno: NOMBRE y APELLIDO, que utilizaremos en la etapa de build para imprimir su valor en un step dentro de este stage (build)

Guardamos y construimos para ver que sucede.

# Jenkins

Tenemos tambien las variables de entorno en pipeline y son importantes, y de gran utilidad. Veamos la salida gráfica y por consola del código anterior.

**Stage Logs (Build)**

```
Shell Script -- echo $NOMBRE $APELLIDO (self time 268ms)
```

```
+ echo Luis Carmona  
Luis Carmona
```

</> Changes

▷ Construir ahora

⚙️ Configurar

trash icon Borrar Pipeline

🔍 Full Stage View

edit icon Rename

info icon Pipeline Syntax

Stage View

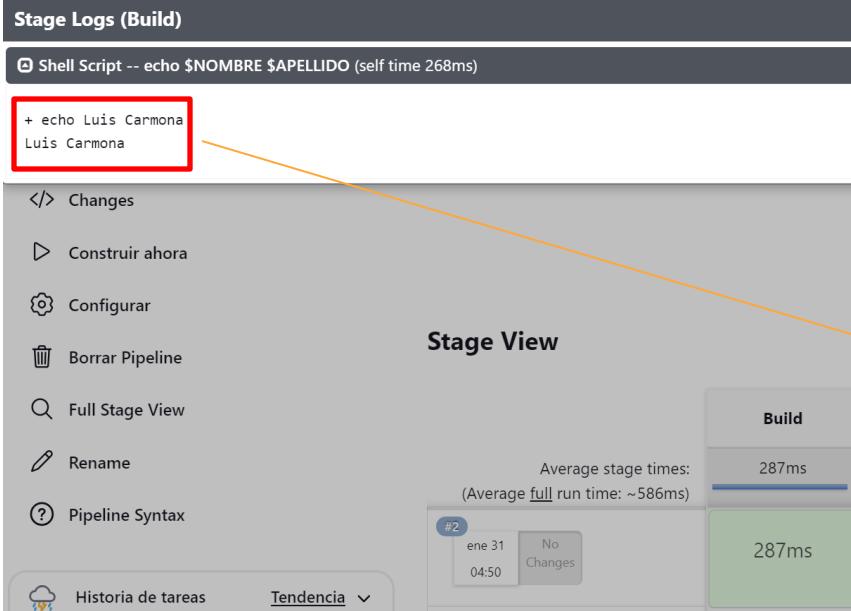
Average stage times:  
(Average full run time: ~586ms)

#2    ene 31    04:50    No Changes

Build    287ms

287ms

Historia de tareas    Tendencia



✓ **Salida de consola**

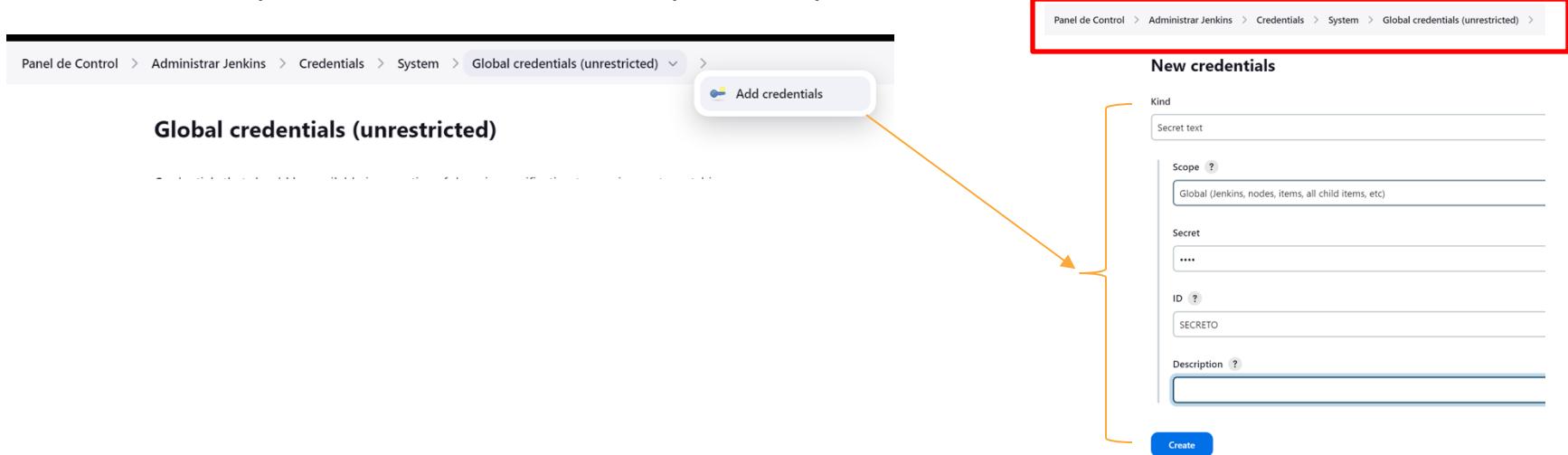
```
Started by user Luis Carmona
[Pipeline] Start of Pipeline (hide)
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_variables_entorno
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] sh
+ echo Luis Carmona
Luis Carmona
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```



# Jenkins

Podemos manejar información sensible o confidencial que quizá necesite un trato “especial” como que no se pueda ver o aparezca con un formato diferente. Podemos hacerlo mediante el uso de la función credentials.

Para ello nos posiciomans en la ruta u opciones que se ve en la siguiente imagen:



The screenshot shows the Jenkins interface for managing global credentials. On the left, the 'Global credentials (unrestricted)' page is displayed with a button labeled 'Add credentials'. An orange arrow points from this button to the 'Add credentials' button on the right. The right side shows the 'New credentials' dialog box, which is a detailed form for creating a new credential. The form fields include:

- Kind:** Secret text
- Scope:** Global (Jenkins, nodes, items, all child items, etc)
- Secret:** (redacted)
- ID:** SECRETO
- Description:** (empty field)

A large red box highlights the top navigation bar and the 'System' link in the breadcrumb trail. A blue 'Create' button is at the bottom right of the dialog box.

# Jenkins

Seleccionamos las opciones que vemos en la imagen:

Panel de Control > Administrar Jenkins > Credentials > System > Global credentials (unrestricted) >

### New credentials

Kind  
Secret text

Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

Secret  
....

ID ?  
SECRETO

Description ?

Create

Secret text  
Global  
Asignamos el secreto (cualquier texto)  
Asignamos un ID (importante recordarlo)

Panel de Control > Administrar Jenkins > Credentials > System > Global credentials (unrestricted) >

### Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind
cfef776c-02f9-4df5-a5dc-6c9fcf33b570	/*****	Username with password
8733a7c0-6ffb-4206-b101-122d4ac0cb93	/*****	Username with password
2e32d166-aa24-4a63-9162-4facd29c86f3	/*****	Username with password
admin	admin/*****	Username with password
parallels	parallels/*****	Username with password
SECRETO	SECRETO	Secret text

# Jenkins

Ahora, veamos el siguiente código:

```
155 pipeline {  
156     agent any  
157  
158     environment{  
159         confidential = credentials('SECRETO')  
160     }  
161     stages{  
162         stage("Build"){  
163             steps{  
164                 sh 'echo $confidencial'  
165             }  
166         }  
167     }  
168 }  
169 }
```

Definimos la variable de entorno "confidencial" que tiene como valor a la función credentials, misma que como argumento tiene el nombre que definimos previamente.

En el stage, estamos imprimiendo el valor de confidential, para confirmar o corroborar que se guardó adecuadamente.

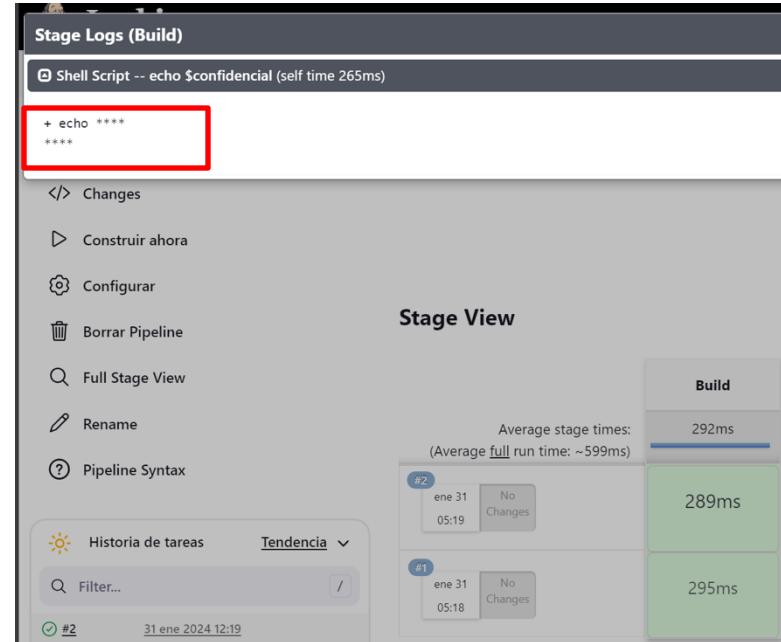
# Jenkins

Despues, guardamos y construimos, y listo. Veamos la salida por consola y gráficamente.

## ✓ Salida de consola

```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_credentials
[Pipeline] {
[Pipeline] withCredentials
Masking supported pattern matches of $confidencial
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline]
+ echo ****
****

[Pipeline]
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```



The screenshot shows the Jenkins interface. On the left, the 'Stage Logs (Build)' panel displays the console output of a pipeline stage. Two lines of output, '+ echo \*\*\*\*' and '\*\*\*\*', are highlighted with a red box. On the right, the 'Stage View' panel provides a summary of the build stages. It shows two stages: '#2' and '#1'. Stage #2 has a duration of 292ms and is shown in blue. Stage #1 has a duration of 289ms and is shown in green. Both stages are labeled 'No Changes'.

Stage	Duration	Status
#2	292ms	Blue (Success)
#1	289ms	Green (Success)

# Jenkins

## Ejercicio

Cree un nuevo proyecto, pero esta de vez de pipeline

Defina 3 etapas o stage para este:

- Build
  - Imprima el mensaje building
  - Imprima la ruta actual en la que se esta generando o ejecutando
  -
- Test
  - Imprima el mensaje testing
  - Imprima "aqui se esta probando la aplicación" ||
- Deploy
  - Imprima el mensaje deploying<
  - Impirma "desplegando la aplicación a QA"

Modifiquemos el ejercicio anterior para poder ejecutar lo siguiente:

- Build
  - Modifique el código que imprime el mensaje building para que el pipeline intente ejecutarlo 5 veces, no le sea posible y por lo tanto el build no sea exitoso
- Test
  - Al código que imprime el mensaje Testing, agreguele una pausa de 5 segundos, y un timeout de 3 segundos, con esto no se podrá ejecutar y debería marcar error, pero como en el anterior paso, agregue el codigo para lo que lo intente al menos 3 veces.
- Deploy
  - Agregar las 5 acciones posteriores o post, pero que solo se ejecuten las siguientes:
    - La que siempre se ejecuta, sea correcto o no el build.
    - La que se ejecuta si el build fue modificado
    - La exitosa, que se ejecuta cuando la contrucción, no marca errores

Agregue dos variables de entorno y uselas en un nuevo stage en el código previamente usado , en este stage, imprima ambas una en un renglon cada 1, y despues imprima una frase, la que usted quiera, usandolas.

Quedando el pipeline con 4 stage

- Build
- Test
- Deploy
- Nueva stage

Finalmente a cada una de las etapas, agregue e imprima 1 texto secreto (el que usted decida)

# Jenkins

Tambien tenemos operadores como los condicionales o los bucles, veremos este primer caso

```
pipeline {  
    agent any  
  
    environment{  
        tool = "jenkins"  
    }  
    stages{  
        stage("condicional"){  
            steps{  
                script{  
                    if(tool=="jenkins") {  
                        echo "CI/CD"  
                    }  
                    else{  
                        echo "No CI/CD"  
                    }  
                }  
            }  
        }  
    }  
}
```

```
155 pipeline {  
156     agent any  
157  
158     environment{  
159         tool = "jenkins"  
160     }  
161     stages{  
162         stage("condicional"){  
163             steps{  
164                 script{  
165                     if(tool=="jenkins") {  
166                         echo "CI/CD"  
167                     }  
168                     else{  
169                         echo "No CI/CD"  
170                     }  
171                 }  
172             }  
173         }  
174     }  
175 }
```

## Console Output

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/if_condition  
[Pipeline] {  
[Pipeline] withEnv  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (condicional)  
[Pipeline] script  
[Pipeline] {  
[Pipeline] echo  
CI/CD  
[Pipeline] }  
[Pipeline] // script  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // withEnv  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Tambien tenemos operadores como los condicionales o los bucles, veremos este primer caso

```
155 pipeline {  
156     agent any  
157  
158     stages{  
159         stage("condicional"){  
160             steps{  
161                 script{  
162                     for(int i=0; i<=5; i++)  
163                         println(i)  
164                 }  
165             }  
166         }  
167     }  
168 }
```

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_loop  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (condicional)  
[Pipeline] script  
[Pipeline] {  
[Pipeline] echo  
0  
[Pipeline] echo  
1  
[Pipeline] echo  
2  
[Pipeline] echo  
3  
[Pipeline] echo  
4  
[Pipeline] echo  
5  
[Pipeline] }  
[Pipeline] // script  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Variables en Jenkins.

Tenemos dos tipos de variables en Jenkins:

- Predefinidas: estan en Jenkins por defecto.
- Definidas por el usuario: el usuario las puede crear

Para revisar las primeras, podemos acceder desde la barra de busqueda del explorador con la siguiente sentencia:

**Host:8080/env-vars.html**

**Ip:8080/env-vars.html**

# Jenkins



The screenshot shows a Jenkins dashboard with the URL `localhost:8080/env-vars.html/`. The page lists various environment variables available for shell and batch build steps, each with a detailed description. The variables include `BRANCH_NAME`, `BRANCH_IS_PRIMARY`, `CHANGE_ID`, `CHANGE_URL`, `CHANGE_TITLE`, `CHANGE_AUTHOR`, `CHANGE_AUTHOR_DISPLAY_NAME`, `CHANGE_AUTHOR_EMAIL`, `CHANGE_TARGET`, `CHANGE_BRANCH`, `CHANGE_FORCE`, `TAG_NAME`, `TAG_TIMESTAMP`, `TAG_UNIXTIME`, and `TAG_DATE`.

Aquí estan todas las variables predefinidas en jenkins

# Jenkins

← → ⌂ ⌂ localhost:8080/env-vars.html/

Dashboard >

URL that will redirect to a Build in a preferred user interface

**RUN\_ARTIFACTS\_DISPLAY\_URL**

URL that will redirect to Artifacts of a Build in a preferred user interface

**RUN\_CHANGES\_DISPLAY\_URL**

URL that will redirect to Changelog of a Build in a preferred user interface

**RUN\_TESTS\_DISPLAY\_URL**

URL that will redirect to Test Results of a Build in a preferred user interface

**CI**

Statically set to the string "true" to indicate a "continuous integration" execution environment.

**BUILD\_NUMBER**

The current build number, such as "153".

**BUILD\_ID**

The current build ID, identical to BUILD\_NUMBER for builds created in 1.597+, but a YYYY-MM-DD\_hh-mm-ss timestamp for older builds.

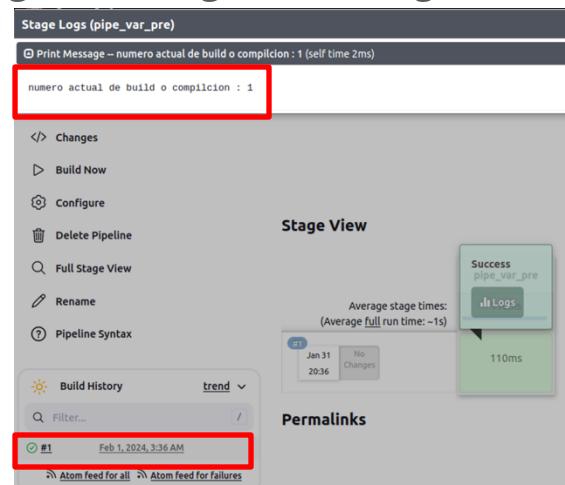
Usemos un ejemplo con BUILD\_ID

# Jenkins

Creamos un nuevo proyecto de pipeline, lo nombramos y agregamos el siguiente código:

```
pipeline{  
    agent any  
    stages{  
        stage("pipe_var_pre"){  
            steps{  
                echo "numero actual de build o compilacion : $BUILD_ID"  
            }  
        }  
    }  
}
```

Número actual de build o compilacion y lo imprimimos como una variable.



Stage Logs (pipe\_var\_pre)  
Print Message – numero actual de build o compilacion : 1 (self time 2ms)  
numero actual de build o compilacion : 1

Changes Build Now Configure Delete Pipeline Full Stage View Rename Pipeline Syntax Build History trend Filter... #1 Feb 1, 2024, 3:36 AM Atom feed for all Atom feed for failures

Average stage times: (Average full run time: -1s)

Success pipe\_var\_pre Logs 110ms

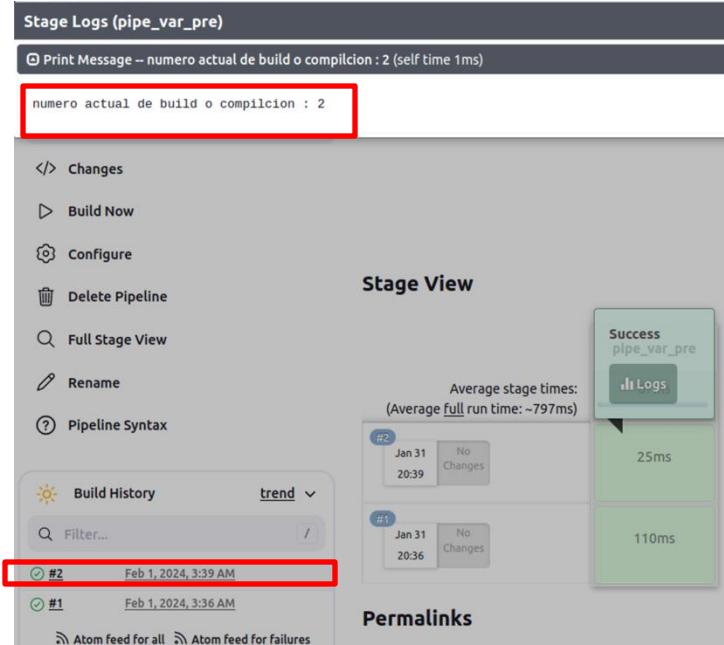
Stage View

Permalinks

Vemos que nos da el valor actual del # de compilación, como decia la descripción.

# Jenkins

Si volvemos a construir, el valor que imprime, tomando como antecedente que el anterior build era 1, en este deberia ser 2 ¿cierto?



The screenshot shows the Jenkins Stage View for a pipeline named "pipe\_var\_pre". The stage has completed successfully with an average stage time of 25ms. The build history shows two builds: #2 (Feb 1, 2024, 3:39 AM) and #1 (Feb 1, 2024, 3:36 AM). The log output for build #2 is displayed, showing the message "numero actual de build o compilcion : 2". A red box highlights this message. The pipeline configuration options (Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, Pipeline Syntax) are visible on the left.

Stage Logs (pipe\_var\_pre)

Print Message -- numero actual de build o compilcion : 2 (self time 1ms)

numero actual de build o compilcion : 2

</> Changes

▷ Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

Build History

#2 Feb 1, 2024, 3:39 AM

#1 Feb 1, 2024, 3:36 AM

Average stage times:  
(Average full run time: ~797ms)

Success pipe\_var\_pre

Logs

25ms

110ms

Permalinks

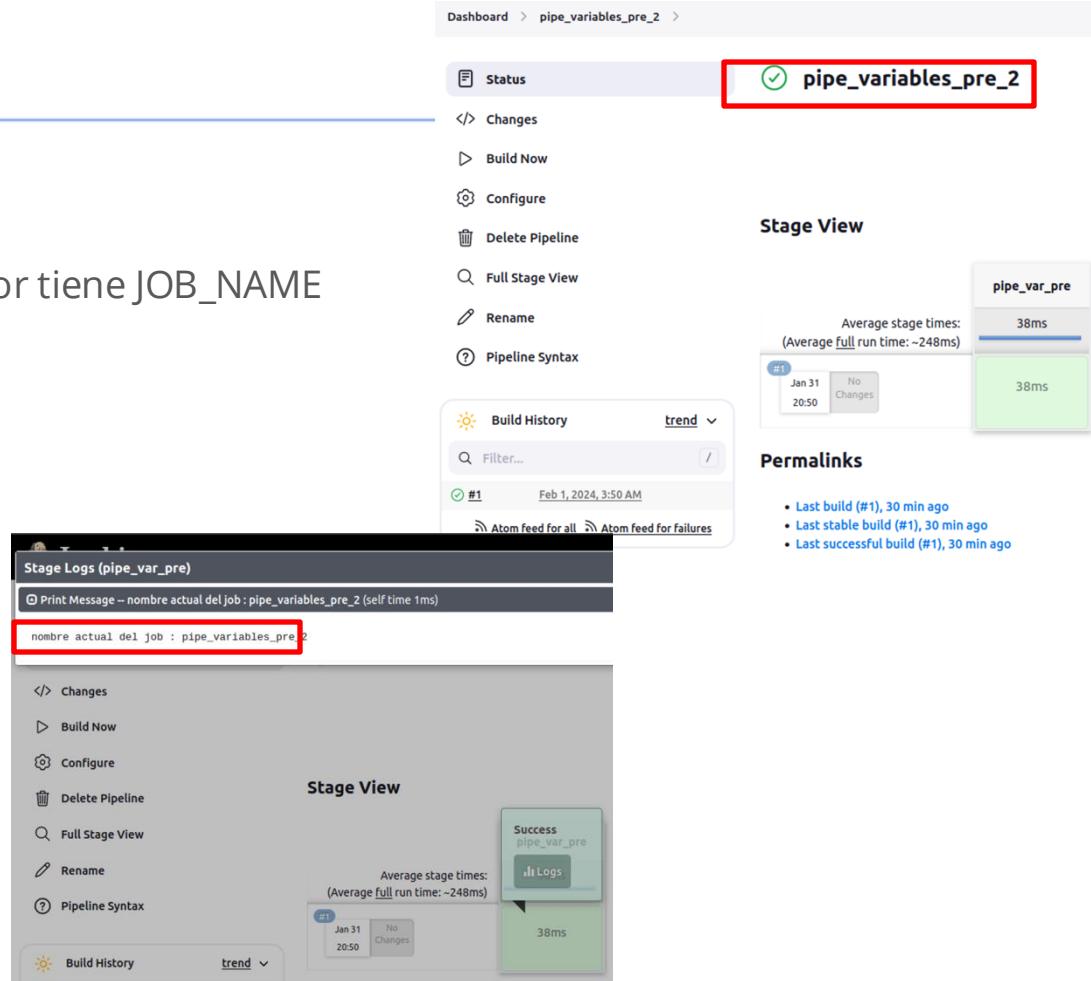
Atom feed for all Atom feed for failures

Efectivamente vemos que esta variable tiene dicho valor.

# Jenkins

Un ejemplo mas, veamos que valor tiene JOB\_NAME

```
pipeline{
    agent any
    stages{
        stage("pipe_var_pre"){
            steps{
                echo "nombre actual del job : $JOB_NAME"
            }
        }
    }
}
```



The screenshot shows the Jenkins Pipeline interface for the job 'pipe\_variables\_pre\_2'. The top navigation bar shows 'Dashboard > pipe\_variables\_pre\_2 > pipe\_variables\_pre\_2'. The pipeline configuration on the left includes stages for 'agent any' and a single stage named 'pipe\_var\_pre'.

**Stage View:** The 'pipe\_var\_pre' stage is currently running, indicated by a green progress bar at 38ms. The stage history shows one build (#1) completed on Feb 1, 2024, at 3:50 AM, with no changes.

**Build History:** The build history panel shows the log entry: 'nombre actual del job : pipe\_variables\_pre\_2'.

**Permalinks:** The page lists the last three builds as successful.

- Last build (#1), 30 min ago
- Last stable build (#1), 30 min ago
- Last successful build (#1), 30 min ago

# Jenkins

Retomando las variables definidas por el usuario, tenemos que éstas pueden estar definidas en 3 niveles, con 3 alcances o lugares diferentes.

- **Nivel global (a nivel de pipeline)**
- Nivel de Stage
- Nivel de Script (step)

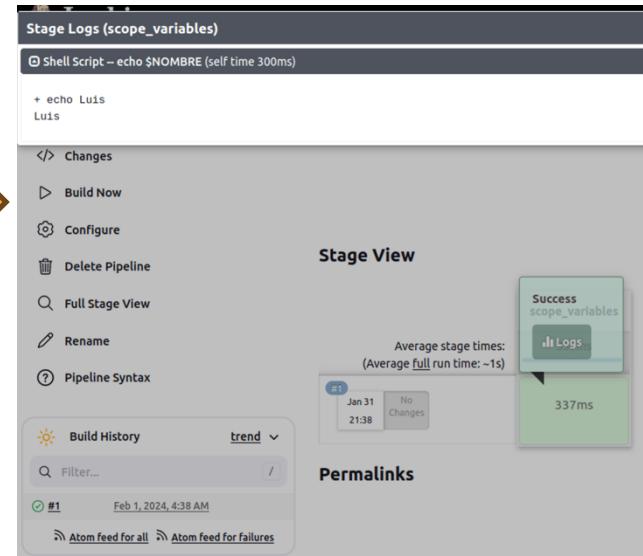
Retomando el ejemplo de las variables de Entorno y modificandolo al siguiente código  
Tenemos:

```

pipeline {
    agent any

    environment{
        NOMBRE= 'Luis'
    }
    stages{
        stage("scope_variables"){
            steps{
                sh 'echo $NOMBRE'
            }
        }
    }
}

```



# Jenkins

Retomando las variables definidas por el usuario, tenemos que éstas pueden estar definidas en 3 niveles, con 3 alcances o lugares diferentes.

```
pipeline{  
    agent any  
  
    environment{  
        NOMBRE= 'Luis'  
    }  
    stages{  
        stage("scope_variables"){  
            steps{  
                echo "$NOMBRE"  
            }  
        }  
        stage("scope_variables_2"){  
            steps{  
                echo "usamos la misma variable $NOMBRE"  
            }  
        }  
    }  
}
```

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_Scope_Variables  
[Pipeline] {  
[Pipeline] withEnv  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (scope_variables)  
[Pipeline] echo  
Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (scope_variables_2)  
[Pipeline] echo  
usamos la misma variable Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // withEnv  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Retomando las variables definidas por el usuario, tenemos que éstas pueden estar definidas en 3 niveles, con 3 alcances o lugares diferentes.

```
pipeline{  
    agent any  
  
    stages{  
        stage("scope_variables"){  
            environment{  
                NOMBRE= 'Luis'  
            }  
            steps{  
                echo "$NOMBRE"  
            }  
        }  
        stage("scope_variables_2"){  
            steps{  
                echo "usamos la misma variable $NOMBRE"  
            }  
        }  
    }  
}
```

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_Scope_Variables  
[Pipeline] {  
[Pipeline] withEnv  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (scope_variables)  
[Pipeline] echo  
Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (scope_variables_2)  
[Pipeline] echo  
usamos la misma variable Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // withEnv  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Retomando las variables definidas por el usuario, tenemos que éstas pueden estar definidas en 3 niveles, con 3 alcances o lugares diferentes.

```
pipeline {  
    agent any  
  
    stages{  
        stage("scope variables"){  
            steps{  
                script{  
                    NOMBRE= 'Luis'  
                }  
                echo "$NOMBRE"  
            }  
        }  
        stage("scope_variables_2"){  
            steps{  
                echo "usamos la misma variable $NOMBRE"  
            }  
        }  
    }  
}
```

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_Scope_Variables  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (scope_variables)  
[Pipeline] script  
[Pipeline] {  
[Pipeline] }  
[Pipeline] // script  
[Pipeline] echo  
Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (scope_variables_2)  
[Pipeline] echo  
usamos la misma variable Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Retomando las variables definidas por el usuario, tenemos que éstas pueden estar definidas en 3 niveles, con 3 alcances o lugares diferentes.

```

pipeline {
    agent any
    environment{
        NOMBRE= 'Luis'
    }
    stages{
        stage("scope_variables"){
            environment{
                NOMBRE= 'Eduardo'
            }
            steps{
                script{
                    NOMBRE= 'Hector'
                }
                echo "$NOMBRE"
            }
        }
        stage("scope_variables_2"){
            steps{
                echo "usamos la misma variable $NOMBRE"
            }
        }
    }
}

```

Console Output

```

Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_Scope_variables
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (scope_variables)
[Pipeline] withEnv
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] }
[Pipeline] // script
[Pipeline] echo
Hector
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (scope_variables_2)
[Pipeline] echo
usamos la misma variable Hector
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```



Vemos que la variable se llama igual (NOMBRE) pero el valor cambia en cada etapa... ¿Qué observan?

El orden jerarquico en realidad es al revés ¿no?

- Step (script) level
- Stage level
- Pipeline level

# Jenkins

Ahora, con el siguiente código, ¿Qué salida por consola esperamos obtener?

```
pipeline {  
    agent any  
    environment{  
        NOMBRE= 'Luis'  
    }  
    stages{  
        stage("scope_variables"){  
            environment{  
                NOMBRE= 'Eduardo'  
            }  
            steps{  
                script{  
                    NOMBRE= 'Hector'  
                }  
                echo "$NOMBRE"  
            }  
        }  
        stage("scope_variables_2"){  
            steps{  
                echo "usamos la misma variable $NOMBRE y ${env.NOMBRE}"  
            }  
        }  
    }  
}
```



# Jenkins

Ahora, con el siguiente código, ¿Qué salida por consola esperamos obtener?

```
pipeline {  
    agent any  
    environment{  
        NOMBRE= 'Luis'  
    }  
    stages{  
        stage("scope_variables"){  
            environment{  
                NOMBRE= 'Eduardo'  
            }  
  
            steps{  
                script{  
                    NOMBRE= 'Hector'  
                }  
                echo "Usamos la misma variable $NOMBRE y ${env.NOMBRE} y ${env.NOMBRE}"  
            }  
        }  
    }  
}
```

Vemos que de alguna forma, respeta la “jerarquía”

## Console Output

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_Scope_variables  
[Pipeline] {  
    [Pipeline] withEnv  
    [Pipeline] {  
        [Pipeline] stage  
        [Pipeline] { (scope_variables)  
            [Pipeline] withEnv  
            [Pipeline] {  
                [Pipeline] script  
                [Pipeline] {  
                    [Pipeline] {  
                        [Pipeline] // script  
                        [Pipeline] echo  
                        Usamos la misma variable Hector y Eduardo y Eduardo  
                }  
            }  
        }  
    }  
} // stage  
[Pipeline] {  
    [Pipeline] withEnv  
    [Pipeline] {  
        [Pipeline] // node  
    }  
}[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Ahora, con el siguiente código, ¿Qué salida por consola esperamos obtener?

```
pipeline {  
    agent any  
    environment{  
        NOMBRE= 'Luis'  
    }  
    stages{  
        stage("scope_variables"){  
            steps{  
                script{  
                    NOMBRE= 'Hector'  
                }  
                echo "Usamos la misma variable $NOMBRE y ${env.NOMBRE} y ${env.NOMBRE}"  
            }  
        }  
    }  
}
```

## Console Output

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_Scope_variables  
[Pipeline] {  
[Pipeline] withEnv  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (scope_variables)  
[Pipeline] script  
[Pipeline] {  
[Pipeline] }  
[Pipeline] // script  
[Pipeline] echo  
Usamos la misma variable Hector y Luis y Luis  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // withEnv  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

Podemos tambien concatenar cadenas, como se puede ver en el siguiente código y salida por consola, esto con el carácter +.

```
pipeline{  
    agent any  
    environment{  
        NOMBRE= "Luis"  
        NOMBRE2= "Carmona"  
    }  
    stages{  
        stage("concatenacion"){  
            steps{  
                script{  
                    nombre = NOMBRE + " " + NOMBRE2  
                }  
                echo "Bienvenido $nombre"  
            }  
        }  
    }  
}
```

## Console Output

```
Started by user Luis Carmona  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /var/jenkins_home/workspace/pipe_concatenacion  
[Pipeline] {  
[Pipeline] withEnv  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (concatenacion)  
[Pipeline] script  
[Pipeline] {  
[Pipeline] }  
[Pipeline] // script  
[Pipeline] echo  
Bienvenido Luis Carmona  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // withEnv  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

# Jenkins

En jenkins tenemos la posibilidad de “disparar” jobs desde el pipeline con una instrucción llamada **build(jobname)**

```
pipeline{
    agent any
    stages{
        stage("Trigger_builds"){
            steps{
                build('Ejercicio_1')
                build('Ejercicio_2')
            }
        }
    }
}
```

**Console Output**

```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_trigger
[Pipeline] {
    [Pipeline] stage
    [Pipeline] { (Trigger_builds)
        [Pipeline] build (Building Ejercicio_1)
        Scheduling project: Ejercicio_1
        Starting building: Ejercicio_1 #4
        Build Ejercicio_1 #4 completed: SUCCESS
        [Pipeline] build (Building Ejercicio_2)
        Scheduling project: Ejercicio_2
        Starting building: Ejercicio_2 #7
        Build Ejercicio_2 #7 completed: SUCCESS
    [Pipeline] }
    [Pipeline] // stage
    [Pipeline] }
    [Pipeline] // node
    [Pipeline] End of Pipeline
Finished: SUCCESS
```

The image shows three Jenkins dashboard screenshots. The first screenshot shows the 'Status' page for 'Ejercicio\_1' with a recent build (#4) highlighted. The second screenshot shows the 'Build History' for 'Ejercicio\_1' with the first two builds (#1 and #2) highlighted. The third screenshot shows the 'Status' page for 'Ejercicio\_2' with a recent build (#7) highlighted. The fourth screenshot shows the 'Build History' for 'Ejercicio\_2' with the first three builds (#1, #2, and #3) highlighted.

Podemos ver la fecha y hora de la construcción mas reciente...

Project	Build #	Date
Ejercicio_1	#1	Jan 23, 2024, 3:36 AM
Ejercicio_1	#2	Jan 23, 2024, 3:37 AM
Ejercicio_1	#4	Feb 1, 2024, 9:26 AM
Ejercicio_2	#1	Jan 23, 2024, 3:27 AM
Ejercicio_2	#2	Jan 23, 2024, 3:28 AM
Ejercicio_2	#3	Jan 23, 2024, 3:29 AM
Ejercicio_2	#7	Feb 1, 2024, 9:26 AM

# Jenkins

En jenkins tenemos la posibilidad de “disparar” jobs desde el pipeline con una instrucción llamada **build(jobname)**.

EL codigo anterior funciona porque ambos jobs se pudieron ejecutar, no tienen errores, pero ¿qué pasaria si uno de los dos tuviera errores?

Cuando falla, en la salida por consola, marca un codigo con una palabra reservada que es **propagate** como se ve en la imagen.

## Console Output

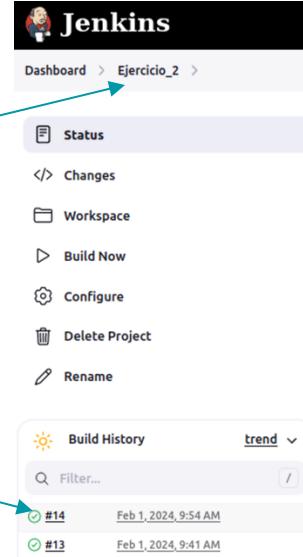
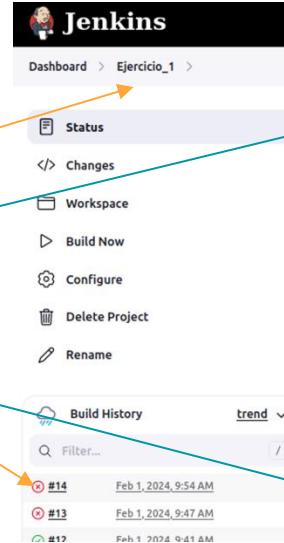
```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_trigger
[Pipeline] {
[Pipeline] stage
[Pipeline] {
[Trigger.builds]
[Pipeline] build (Building Ejercicio_1)
Scheduling project: Ejercicio_1
Starting building: Ejercicio_1 #11
Build Ejercicio_1 #11 completed: FAILURE
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Ejercicio_1 #11 completed with status FAILURE (propagate: false to ignore)
jenkins@jenkins: ~ [14:55:2015]
```

# Jenkins

Entonces, para que Jenkins pueda “saltarse” la falla, en este caso del Job 1 y que el job 2 si se ejecute a pesar de esta falla, es necesario modificar el código del pipeline, con la palabra que vimos en la lámina anterior. El código quedaría de la siguiente forma:

```
pipeline{  
    agent any  
    stages{  
        stage("Trigger_builds"){  
            steps{  
                build(job: 'Ejercicio_1', propagate:false)  
                build('Ejercicio_2'  
            }  
        }  
        }  
    }  
}
```

Donde le decimos, que a pesar de que falle el job (ejercicio\_1) el segundo si se ejecute.



# Jenkins

Podemos almacenar el valor o resultado de un job en una variable, asistido o ayudado por la función **result** y una condicional como **if**. Veamos un ejemplo:

```
pipeline{
    agent any
    stages{
        stage("Trigger_builds"){
            steps{
                script{
                    jobresult=build(job: 'Ejercicio_1', propagate:false).result
                    build'Ejercicio_2'
                    if(jobresult=="FAILURE"){
                        currentBuild.result="UNSTABLE"
                    }
                }
            }
        }
    }
}
```

¿Qué podemos observar?

**Console Output**

```
Started by user Luis Carmona
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipe_trigger
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Trigger_builds)
[Pipeline] script
[Pipeline] {
[Pipeline] build (Building Ejercicio_1)
Scheduling project: Ejercicio_1
Starting building: Ejercicio_1 #16
Build Ejercicio_1 #16 completed: FAILURE
[Pipeline] build (Building Ejercicio_2)
Scheduling project: Ejercicio_2
Starting building: Ejercicio_2 #16
Build Ejercicio_2 #16 completed: SUCCESS
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: UNSTABLE
```

**Stage Logs (Trigger\_builds)**

- Building Ejercicio\_1 (self time 9s)
  - Scheduling project: Ejercicio\_1
  - Starting building: Ejercicio\_1 #16
  - Build Ejercicio\_1 #16 completed: FAILURE
- Building Ejercicio\_2 – Ejercicio\_2 (self time 9s)
  - Build NOW
  - Expand
  - Delete Pipeline
  - Full Stage View
  - Rename

**Stage View**

Trigger_builds	Average stage times: (Average full run time: ~16s)
Feb 01 03:16	14s
Feb 01 03:01	19s
Feb 01 03:01	15s

**Build History**

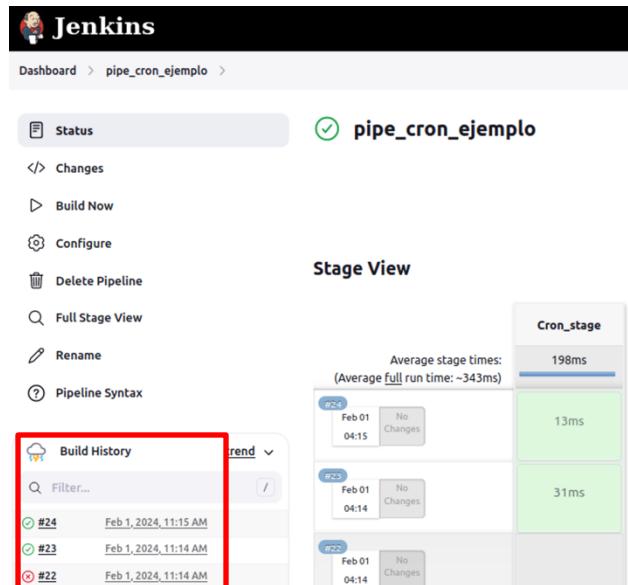
#12	Feb 01 03:16	No Changes
#13	Feb 01, 2024, 10:16 AM	No Changes

**Pipeline Syntax**

# Jenkins

Tambien podemos agendar o programar los jobs, veamos algunos ejemplos.  
Creamos un pipeline, le asignamos nombre y agregamos el siguiente código:

```
pipeline{  
    agent any  
    triggers{  
        cron('* * * * *')      Min,hr, DOM,mes,DOW  
                                0-59 0-23 1-31  1-12   0-7  
    }  
    stages{  
        stage("Cron_stage"){  
            steps{  
                echo "se repite cada minuto"  
            }  
        }  
    }  
}
```



The screenshot shows the Jenkins Pipeline interface for the job 'pipe\_cron\_ejemplo'. The pipeline has one stage named 'Cron\_stage'. The stage view shows three builds: #24 (success), #23 (success), and #22 (failure). The build history table is highlighted with a red box. The table includes columns for build number, date, status, and duration.

Build	Date	Status	Duration
#24	Feb 1, 2024, 11:15 AM	Success	13ms
#23	Feb 1, 2024, 11:14 AM	Success	31ms
#22	Feb 1, 2024, 11:14 AM	Failure	

# Jenkins

Tambien podemos agendar o programar los jobs, veamos algunos ejemplos.  
Creamos otro pipeline, le asignamos nombre y agregamos el siguiente código:

```
pipeline{
    agent any
    triggers{
        pollSCM('* * * * *')
    }
    stages{
        stage("Cron_stage"){
            steps{
                echo "se repite cada minuto"
                git url:'https://github.com/LuisECj21/Curso_Jenkins_24.git'
            }
        }
    }
}
```

```
Started by timer
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipeline_pollSCM
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Cron_stage)
[Pipeline] echo
se repite cada minuto
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/jenkins_home/workspace/pipeline_pollSCM/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/LuisECj21/Curso_Jenkins_24.git # timeout=10
Fetching upstream changes from https://github.com/LuisECj21/Curso_Jenkins_24.git
> git --version # timeout=10
> git --version # 'git version 2.39.2'
> git fetch --tags --force --progress .. https://github.com/LuisECj21/Curso_Jenkins_24.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 9396b7c42f467673997e3d5f60b9f460a734e178 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 9396b7c42f467673997e3d5f60b9f460a734e178 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git branch -D master # timeout=10
> git checkout -b master 9396b7c42f467673997e3d5f60b9f460a734e178 # timeout=10
Commit message: "javamaven05L3.groovy"
> git rev-list --no-walk 9396b7c42f467673997e3d5f60b9f460a734e178 # timeout=10
[Pipeline]
[Pipeline] //
stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

## Jenkins

- Job de estilo libre
- Ejecucion desde GUI y DSL
- Parametrizacion
- Maven
- Despliegue de un .JAR
- Notificaciones
- Perfiles, roles, usuarios
- Autorizacion y autenticacion
- Seed Job
- Pipeline
- Loops/bucles
- Variables y su alcance
- Build, result, cron

Formen equipos de 4 personas y piensen en un ejemplo de sus actividades laborales/profesionales, donde puedan aplicar los temas enlistados a su izquierda.  
Terminando, cada equipo en equipo, presentará su ejercicio.

## Actividad Final de Sección

Actividad: utilizando la herramienta (Kubernetes) vamos a crear por equipo 2 sitios web y el servicio de Jenkins, recuerda que podemos utilizar plantillas para la creación de los sitios web. Para jenkins debemos crear un pipeline para desplegar un .jar como lo hicimos en el curso.

Ya creados validemos, si son accesibles mediante su ip :puerto

Éxito!!





Tel: 7022 6027 / 2625 3107  
[informes@sulens.com](mailto:informes@sulens.com)  
[www.sulens.com](http://www.sulens.com)

