

Informe de Proyecto DataOps – Diplomado de Ingeniería de Datos

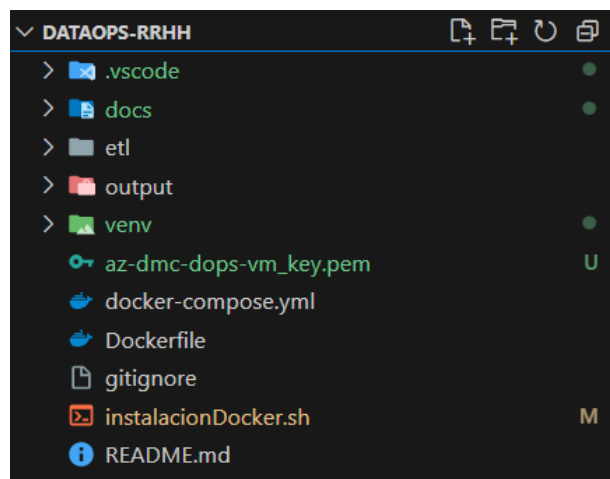
1. Introducción

Este proyecto consiste en el desarrollo de un pipeline ETL automatizado utilizando principios de DataOps. Se integraron datos desde una base de datos PostgreSQL, se aplicaron transformaciones con Python, y el proceso fue empaquetado usando Docker. Finalmente, se implementa una etapa de CI/CD mediante Jenkins para permitir despliegues controlados.

2. Estructura del Proyecto

La siguiente imagen muestra la estructura del repositorio local:

- etl: contiene el script de transformación y dependencias.
- output: almacena los archivos CSV generados.
- docs: carpeta destinada a la documentación del proyecto.
- Dockerfile y docker-compose.yml: archivos de contenerización.
- .gitignore y README.md: archivos de configuración y guía del repositorio.
- instalacionDocker.sh: script para configurar el entorno base en la VM de Jenkins.



3. Entorno de Desarrollo

El desarrollo del proyecto se realizó en el sistema operativo Windows utilizando Visual Studio Code como entorno principal. Se creó un entorno virtual (venv) con Python 3.13.2 para gestionar las dependencias, las cuales fueron registradas en el archivo requirements.txt.

Aunque la mayor parte del trabajo se ejecutó desde VS Code, se utilizaron comandos básicos en la terminal (Git Bash) principalmente para gestionar el control de versiones con Git y para ejecutar el script ETL dentro de un contenedor mediante Docker Desktop.

Se construyó una imagen Docker a partir de un Dockerfile personalizado, y se ejecutó el contenedor localmente como parte del flujo de pruebas. Esto permitió validar que el proceso podía correr de manera aislada y replicable.

El repositorio del proyecto fue alojado en GitHub, sirviendo de base para su integración posterior con Jenkins en la fase de automatización CI/CD.

```
(venv) C:\Users\joaqu\Documents\DMC Diplomados\Data Engineer\GIT\dataops-rrhh>python --version
Python 3.13.2

(venv) C:\Users\joaqu\Documents\DMC Diplomados\Data Engineer\GIT\dataops-rrhh>
```

```
joaqu@Laptop-JR MINGW64 ~/Documents/DMC Diplomados/Data Engineer/GIT/dataops-rrh
h (main)
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
joaqu@Laptop-JR MINGW64 ~/Documents/DMC Diplomados/Data Engineer/GIT/dataops-rrh
h (main)
$ docker --version
Docker version 28.1.1, build 4eba377
```

Además del script principal main.py, se trabajó inicialmente en un archivo main.ipynb (Jupyter Notebook), el cual fue utilizado para pruebas exploratorias, conexión a la base de datos y validación de transformaciones antes de consolidar el pipeline final. Por esta razón, el archivo requirements.txt incluye la librería notebook.

4. Desarrollo del ETL

El script main.py realiza la conexión a PostgreSQL, lee los datos de la tabla rrhh.empleado, aplica transformaciones como conversión de texto, normalización, creación de campos derivados y exportación a un archivo CSV con timestamp.

Conexión a la base de datos PostgreSQL

```

7  # === Configuración de conexión ===
8  db_user = "usr_ro_dmc_rrhh_estudiantes"
9  raw_pass = "fZp!jHt0j6%89^B4I*L*29bz4b^"
10 db_pass = urllib.parse.quote_plus(raw_pass)
11 db_host = "mgg.vps.webdock.cloud"
12 db_port = "5432"
13 db_name = "dmc"
14 schema = "rrhh"
15 table = "empleado"

```

```

17 # === Crear engine SQLAlchemy ===
18 engine = sqlalchemy.create_engine(
19     f"postgresql+psycopg2://{db_user}:{db_pass}@{db_host}:{db_port}/{db_name}"
20 )
21
22 # === Ejecutar consulta ===
23 query = f"SELECT * FROM {schema}.{table}"
24
25 try:
26     df = pd.read_sql_query(query, engine)
27 except Exception as e:
28     print("❌ Error al conectar o consultar:", e)
29     exit()

```

Ejecución de transformaciones:

```

31 # === Transformaciones ===
32 df["tip_documento"] = df["tip_documento"].str.upper()
33 df["nom_empleado"] = df["nom_empleado"].str.title()
34 df["ape_empleado"] = df["ape_empleado"].str.title()
35 df["nom_empleado_completo"] = df["nom_empleado"] + " " + df["ape_empleado"]
36 df["mnt_tope_comision"] = df["mnt_tope_comision"].fillna(0)
37 df["salario_anual"] = round(df["mnt_salario"] * 12, 2)
38

```

Disposición de archivo CSV:

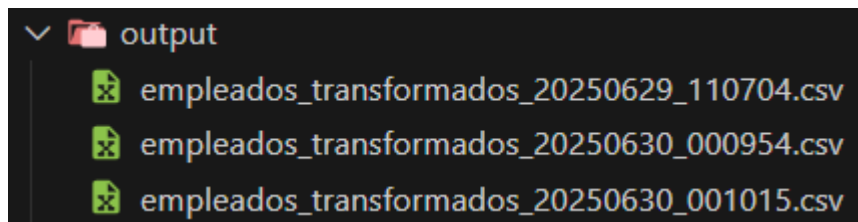
```

51 # === Exportar resultado ===
52 try:
53     df.to_csv(output_file, index=False, encoding="utf-8-sig")
54     print(f"✅ Archivo exportado correctamente en: {output_file}")
55 except Exception as e:
56     print(f"❌ Error al exportar el archivo: {e}")

```

5. Carga de Datos Transformados

El archivo resultante se exporta en formato CSV a la carpeta output/. El nombre del archivo incluye la fecha y hora de ejecución como medida de control y trazabilidad.



```

1 empleado_id,tipo_documento,num_documento,nom_empleado,ape_empleado,cod_cargo,cod_departamento,mnt_salario,mnt_tope_comision,nom_empleado_completo,salario_anual
2 1,DNI,78981234,Ana,Pérez,CAR004,DEP004,1800.0,0.0,Ana Pérez,21600.0
3 2,CE,AB123456,Carlos,López,CAR002,DEP002,2500.5,300.0,Carlos López,30006.0
4 3,PASAPORTE,XYZ789,Sofía,Martínez,CAR003,DEP003,2200.75,0.0,Sofía Martínez,26409.0
5 4,DNI,12345678,Daniel,Gómez,CAR001,DEP001,4800.0,500.0,Daniel Gómez,48000.0
6 5,CE,CD987654,Elena,Rodríguez,CAR005,DEP005,2000.2,0.0,Elena Rodríguez,24002.4
7

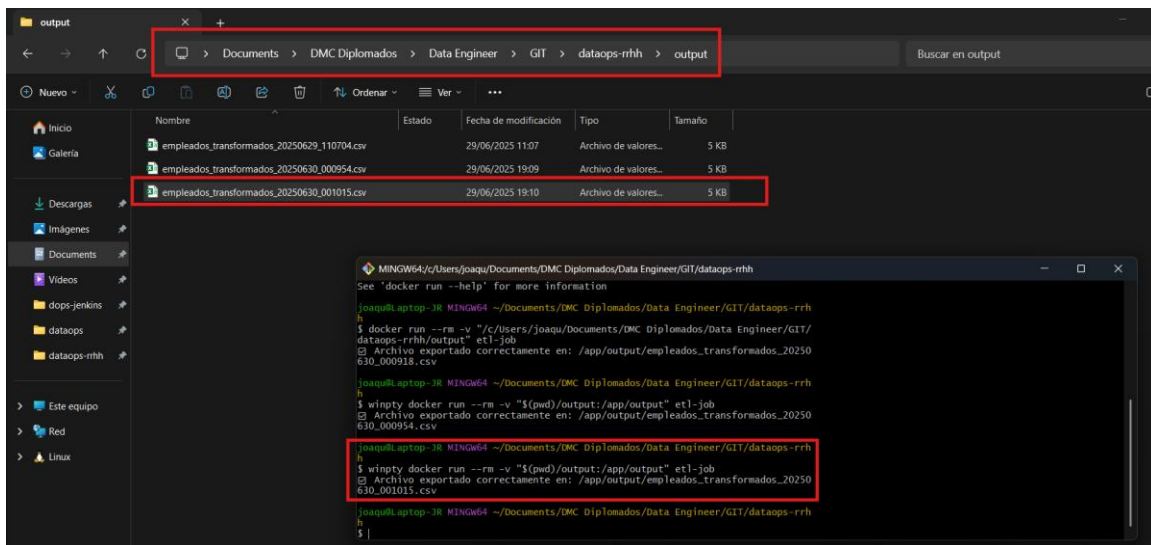
```

6. Empaquetado con Docker

Se creó un Dockerfile que define una imagen ligera basada en python:3.13-slim. La imagen contiene el código del ETL, las dependencias y un comando CMD para ejecutar el script automáticamente al correr el contenedor.

```
1 # Imagen base con Python
2 FROM python:3.13-slim
3
4 # Establecer el directorio de trabajo dentro del contenedor
5 WORKDIR /app
6
7 # Copiar los archivos del proyecto al contenedor
8 COPY . .
9
10 # Instalar las dependencias del ETL
11 RUN pip install --no-cache-dir -r etl/requirements.txt
12
13 # Crear la carpeta de salida si no existe
14 RUN mkdir -p output
15
16 # Comando por defecto al ejecutar el contenedor
17 CMD ["python", "etl/main.py"]
18
```

Nota: durante el desarrollo se presentaron dificultades al montar volúmenes usando Docker desde Git Bash en Windows debido a rutas con espacios. Se solucionó exitosamente utilizando winpty para ejecutar el contenedor, permitiendo que las rutas fueran interpretadas correctamente por Docker Desktop.

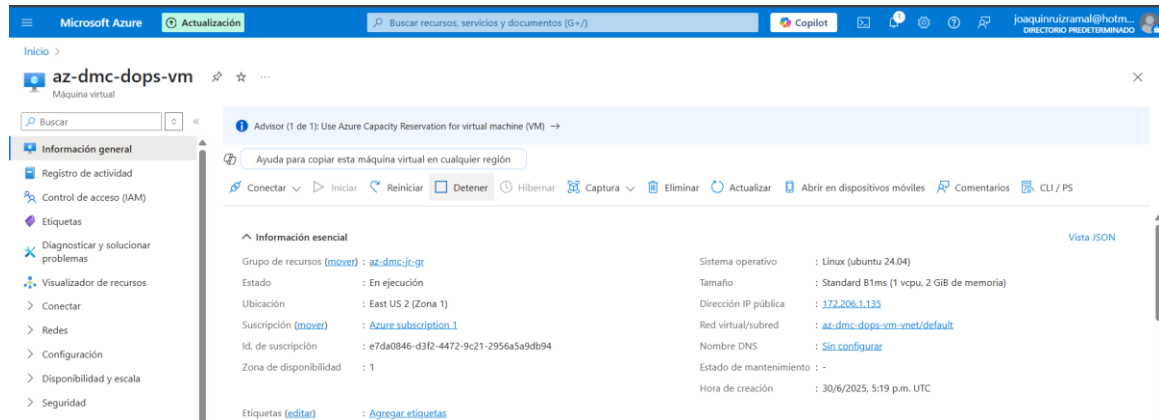


7. Automatización con Jenkins (CI/CD)

Automatización del entorno Jenkins

Para este proyecto se utilizó una máquina virtual con sistema operativo Linux (Ubuntu) en la que se instaló Docker, Java y Git utilizando un script automatizado llamado instalacionDocker.sh. Este archivo contiene todos los comandos necesarios

para configurar el entorno base de trabajo, permitiendo dejar la VM lista sin necesidad de instalaciones manuales.



```
azureuser@az-dmc-dops-vm:~$ sh instalacionDocker.sh
-----
> Instalando Java y Herramientas
-----
git is already the newest version (1:2.43.0-1ubuntu7.2).
git set to manually installed.
The following additional packages will be installed:
adwaita-icon-theme alsa-topology-conf alsa-ucm-conf at-spi2-common
at-spi2-core ca-certificates-java dconf-gsettings-backend dconf-service
fontconfig fontconfig-config fonts-dejavu-core fonts-dejavu-extra
fonts-dejavu-mono gsettings-desktop-schemas gtk-update-icon-cache
hicolor-icon-theme humanity-icon-theme java-common libasound2-data
libasound2t64 libatk-bridge2.0-0t64 libatk-wrapper-java
libatk-wrapper-java-jni libatk1.0-0t64 libatspi2.0-0t64 libavahi-client3
libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2
libcups2t64 libdatrie1 libdconf1 libdeflate0 libdrm-amdgpu1 libdrm-intel1
libdrm-nouveau2 libdrm-radeon1 libfontconfig1 libgail-common libgail18t64
libgbm1 libgdk-pixbuf-2.0-0 libgdk-pixbuf2.0-bin libgdk-pixbuf2.0-common
libgif7 libgl1 libgl1-amd-glx libgl1-mesa-dri libglapi-mesa libglvnd0
libglx-mesa0 libglx0 libgraphite2-3 libgtk2.0-0t64 libgtk2.0-bin
libgtk2.0-common libharfbuzz0b libice-dev libice6 libjpeg-turbo8
libjpeg8 liblcms2-2 liblerc4 libllvm19 libpango-1.0-0 libpangocairo-1.0-0
libpangoft2-1.0-0 libpciaccess0 libpcsc-lite1 libpixmap-1-0
libpthread-stubs0-dev librsvg2-2 librsvg2-common libsharpyuv0 libsm-dev
libsm6 libthai-data libthai0 libtiff6 libvulkan1 libwayland-client0
libwayland-server0 libwebp7 libx11-dev libx11-xcb1 libxau-dev libxaw7
libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-randr0
libxcb-render0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0
libxcb1-dev libxcomposite1 libxcursor1 libxdamage1 libxdmcp-dev libxfixes3
libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxmuu1 libxrandr2
```

Comprobacion del funcionamiento al utilizar *Docker run hello-world* y *java --version*

```
172.206.1.135 (azureuser)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
/home/azureuser/
Name
.cache
.ssh
.bash_logout
.bashrc
.profile
.Xauthority
docker-compose.yml
instalacionDocker.sh
Remote monitoring
Follow terminal folder
0 upgraded, 0 newly installed, 0 to remove and 51 not upgraded.
azureuser@az-dmc-dops-vm:~$ java --version
openjdk 17.0.15 2025-04-15
OpenJDK Runtime Environment (build 17.0.15+6-Ubuntu-0ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.15+6-Ubuntu-0ubuntu124.04, mixed mode, sharing)
azureuser@az-dmc-dops-vm:~$ docker run hello-world

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.
   (amd64)
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

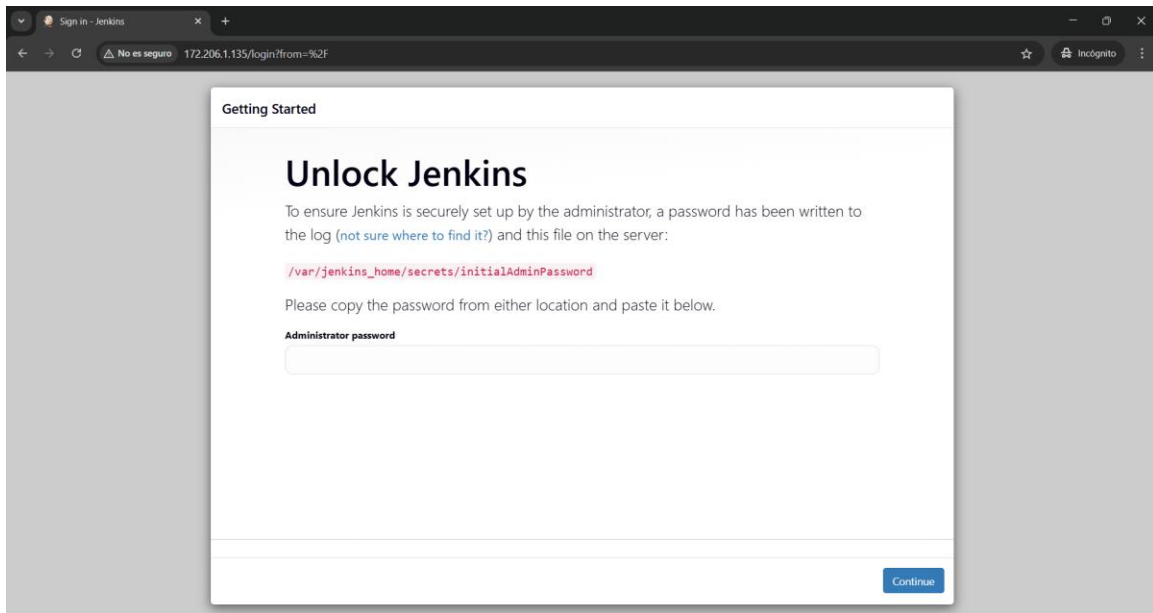
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

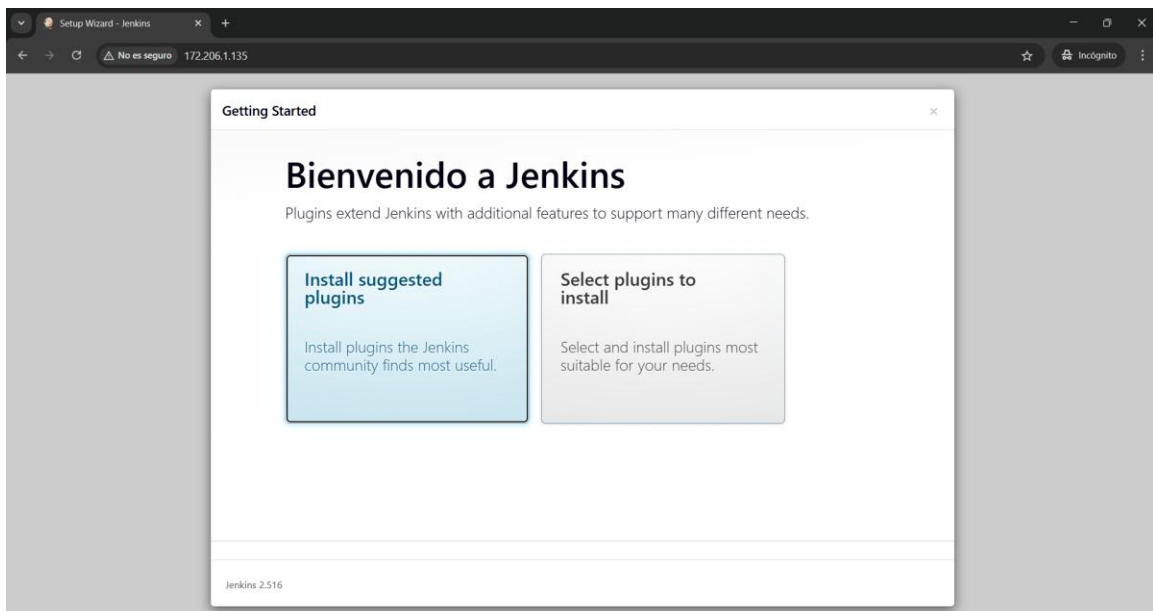
azureuser@az-dmc-dops-vm:~$ java --version
openjdk 17.0.15 2025-04-15
OpenJDK Runtime Environment (build 17.0.15+6-Ubuntu-0ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.15+6-Ubuntu-0ubuntu124.04, mixed mode, sharing)
azureuser@az-dmc-dops-vm:~$
```

```
azureuser@az-dmc-dops-vm:~$ docker compose up -d
[+] Running 13/13
 ✓ jenkins Pulled 20.5s
 ✓ 0c01110621e0 Pull complete 6.5s
 ✓ bbb7e19972478 Pull complete 13.8s
 ✓ f775cc3d3265 Pull complete 14.1s
 ✓ daf323b9a67c Pull complete 14.2s
 ✓ c9f37f79f693 Pull complete 14.3s
 ✓ 80e1e7a1bee3 Pull complete 16.4s
 ✓ ea45d56f7d63 Pull complete 16.5s
 ✓ a803b656f371 Pull complete 16.7s
 ✓ 5c3a6f2bcd86 Pull complete 19.8s
 ✓ 2f0cb4d4615d Pull complete 19.9s
 ✓ 859818620762 Pull complete 20.0s
 ✓ ae9f59287d32 Pull complete 20.1s
[+] Running 3/3
 ✓ Network azureuser_default Created 0.1s
 ✓ Volume "azureuser_jenkins_home" Created 0.0s
 ✓ Container jenkins Started 0.4s
azureuser@az-dmc-dops-vm:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
PORTS         NAMES
5917bbb7eb5b   jenkins/jenkins:latest             "/usr/bin/tini -- /u..." 18 seconds ago Up 18 secon
ds            50000/tcp, 0.0.0.0:80->8080/tcp, [::]:80->8080/tcp   jenkins
```

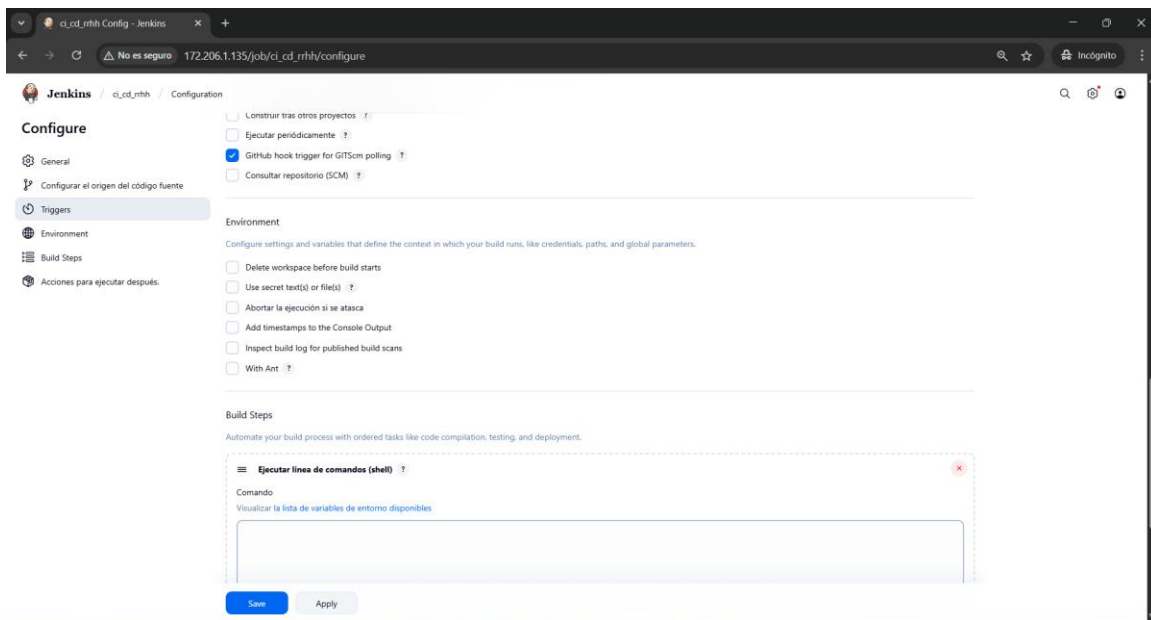
Una vez completada la instalación, Jenkins se desplegó como contenedor utilizando el archivo docker-compose.yml. Este archivo levanta el servicio en el puerto 80, accediendo a Jenkins desde el navegador.



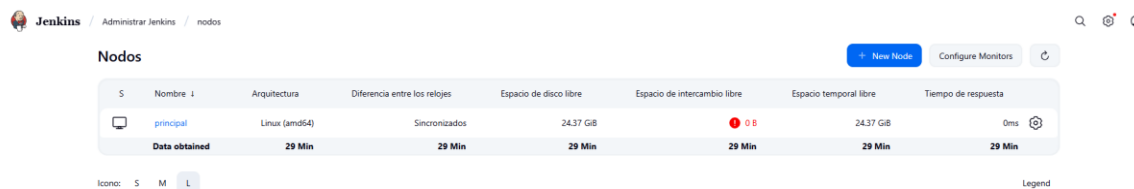
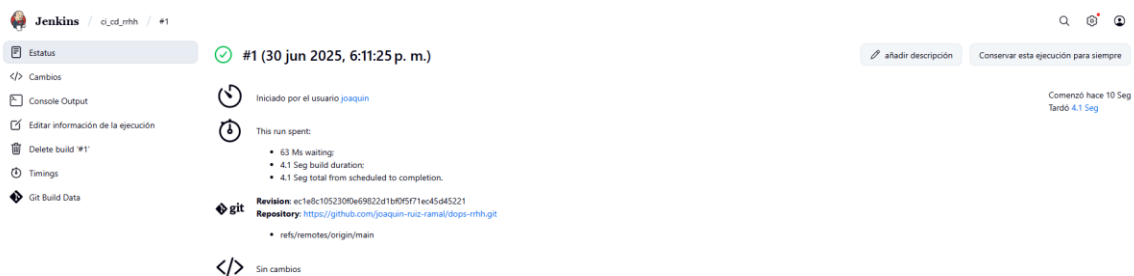
Se obtuvo la clave ejecutando la línea *docker logs jenkins*

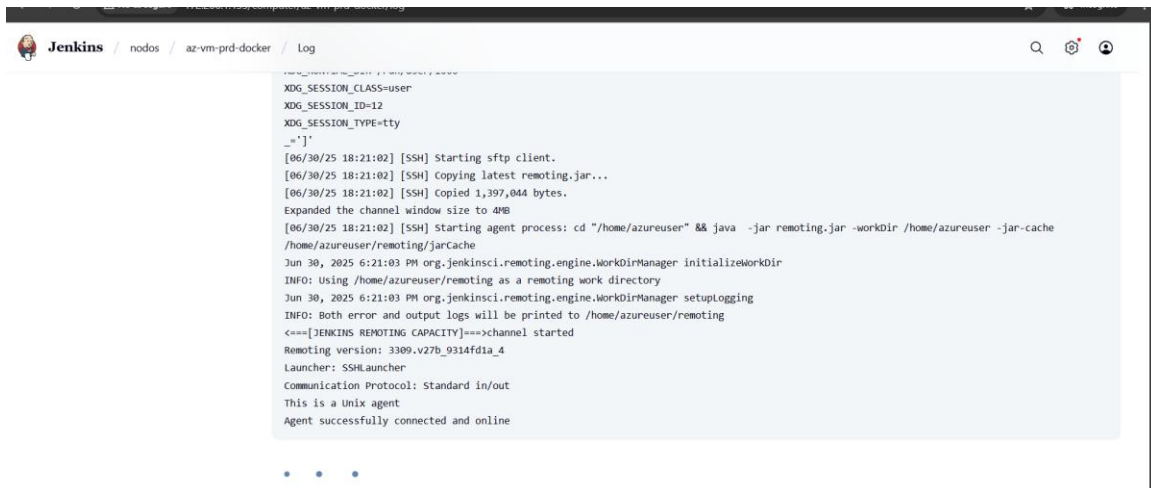


Desde la interfaz web de Jenkins se creó manualmente un proyecto, replicando el procedimiento visto en clase. Asegurándonos de marcar la casilla Github hook trigger for GITScm polling



Se verificó el funcionamiento del servidor Jenkins tras su despliegue. Posteriormente, se configuró un nodo adicional (az-vm-prd-docker) como agente de ejecución. De esta forma, el nodo principal solo tendrá carga administrativa (configuración adicional realizada).





Se creó un segundo nodo az-vm-prd-docker, y se configuro el nodo principal para que solo ejecutara tareas asociadas al mismo.

Nodos							
S	Nombre ↓	Arquitectura	Diferencia entre los relojes	Espacio de disco libre	Espacio de intercambio libre	Espacio temporal libre	Tiempo de respuesta
	az-vm-prd-docker	Linux (amd64)	Sincronizados	24.18 GiB	0 B	24.18 GiB	137ms
	principal	Linux (amd64)	Sincronizados	24.18 GiB	0 B	24.18 GiB	0ms
Data obtained		21 Seg	21 Seg	21 Seg	21 Seg	21 Seg	21 Seg

Iconos: S M L Legend

Se comprobó el funcionamiento de la tarea:



En la siguiente imagen se puede observar como terminó la construcción (build) a las 7:27 pm. En las siguientes imágenes veremos como el archivo csv se exporto a las 7:28 pm tanto en la fecha de modificación, así como en el nombre del archivo (para mayor trazabilidad)

Jenkins / ci_cd_rrhh

Estado Actual

Cambios

Zona de Trabajo

Construir ahora

Configurar

Borrar Proyecto

GitHub Hook Log

Rename

ci_cd_rrhh

Despliegue continuo de ETL rrhh Docker

Enlaces permanentes

- Última ejecución (#5) hace 1 Min 17 Seg
- Última ejecución estable (#5) hace 1 Min 17 Seg
- Última ejecución correcta (#5) hace 1 Min 17 Seg
- Última ejecución fallida (#2) hace 1 Hor 5 Min
- Última ejecución fallida (#2) hace 1 Hor 5 Min
- Last completed build (#5) hace 1 Min 17 Seg

Builds

Filter

Today

5 7:27 p. m.

Jenkins / ci_cd_rrhh / #5 / Salida de consola

```
prompt_toolkit-3.0.51 psutil-7.0.0 pycopg2-binary-2.9.10 ptyprocess-0.7.0 pure-eval-0.2.3 pycparser-2.22 pygments-2.19.2 python-dateutil-2.9.0.post0 python-json-logger-3.2.0 pytz-2025.2 pyyaml-6.0.2 pyzmq-27.0.0 referencing-0.36.2 requests-2.32.4 rfc3339-validator-0.1.4 rfc3986-validator-0.1.1 rpds-py-0.25.1 send2trash-1.8.3 setuptools-80.9.0 six-1.17.0 sniffio-1.3.1 soupsieve-2.7 sqlalchemy-2.0.41 stack_data-0.6.3 terminado-0.18.1 tinycss2-1.4.0 tornado-6.5.1 traitlets-5.14.3 types-python-dateutil-2.9.0.20250516 typing-extensions-4.14.0 tzdata-2025.2 url-template-1.3.0 urllib3-2.5.0 wcwidth-0.2.13 webcolors-24.11.1 webencodings-0.5.1 websocket-client-1.8.0
#8 42.75 WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
#8 DONE 44.6s

#9 [5/5] RUN mkdir -p output
#9 DONE 0.5s

#10 exporting to image
#10 exporting layers
#10 exporting layers 18.1s done
#10 writing image sha256:6c01e8c7611be142cbbc962abb5e34b383832b5ef0f24dfc28104eb54fc7b7cb
#10 writing image sha256:6c01e8c7611be142cbbc962abb5e34b383832b5ef0f24dfc28104eb54fc7b7cb done
#10 naming to docker.io/library/etl-job 0.0s done
#10 DONE 18.2s
+ docker run --rm -v /home/azureuser/workspace/ci_cd_rrhh/output:/app/output etl-job
[+] Archivo exportado correctamente en: /app/output/empleados_transformados_20250630_192812.csv
Finished: SUCCESS
```

Jenkins / ci_cd_rrhh / Workspace of ci_cd_rrhh on az-vm-prd-docker

Estado Actual

Cambios

Zona de Trabajo

Construir ahora

Configurar

Borrar Proyecto

GitHub Hook Log

Rename

Workspace of ci_cd_rrhh on az-vm-prd-docker

ci_cd_rrhh / output /

empleados_transformados_20250629_110704.csv	30 jun 2025, 6:23:10 p. m.	4.12 KiB		
empleados_transformados_20250630_000954.csv	30 jun 2025, 6:23:10 p. m.	4.12 KiB		
empleados_transformados_20250630_001015.csv	30 jun 2025, 6:23:10 p. m.	4.12 KiB		
empleados_transformados_20250630_183926.csv	30 jun 2025, 6:39:26 p. m.	4.12 KiB		
empleados_transformados_20250630_192625.csv	30 jun 2025, 7:26:25 p. m.	4.12 KiB		
empleados_transformados_20250630_192812.csv	30 jun 2025, 7:28:12 p. m.	4.12 KiB		

[\(Descargar archivos en formato zip\)](#)

El pipeline fue configurado para ejecutar los siguientes pasos:

- Clonar el repositorio desde GitHub.
- Construir una imagen Docker llamada etl-job.
- Ejecutar el contenedor, el cual corre el script ETL desarrollado en Python y genera como salida un archivo .csv en la carpeta “output”.

Esta configuración demuestra una implementación funcional de un flujo CI/CD sin necesidad de Jenkinsfile, aprovechando la configuración manual desde la consola web de Jenkins mediante un proyecto tipo “libre”.

8. Conclusión

Este proyecto permitió aplicar principios reales de DataOps, desde la conexión a una fuente de datos relacional hasta la ejecución automática vía contenedores. Se reforzó el uso de versionamiento, entornos aislados y automatización.