Informe de Proyecto DataOps – Diplomado de Ingeniería de Datos

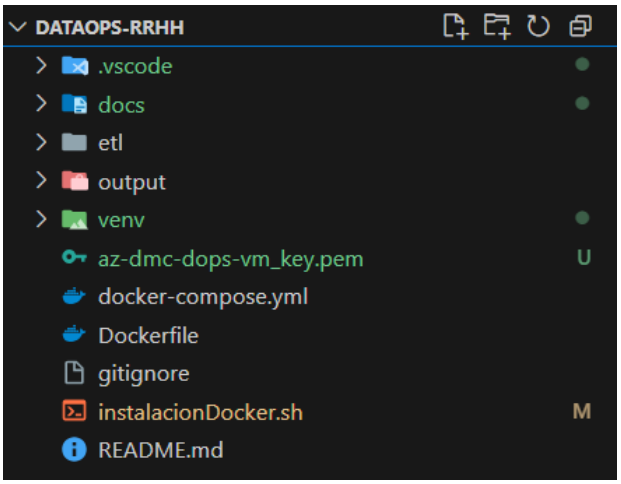
1. Introducción

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.



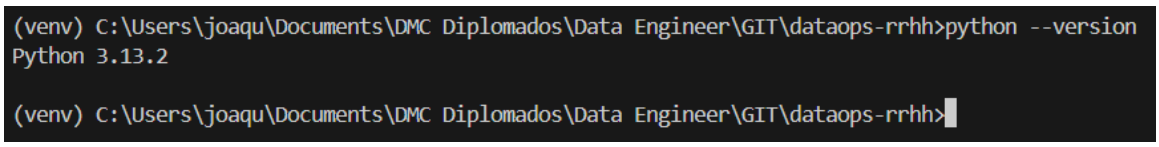
3. Entorno de Desarrollo

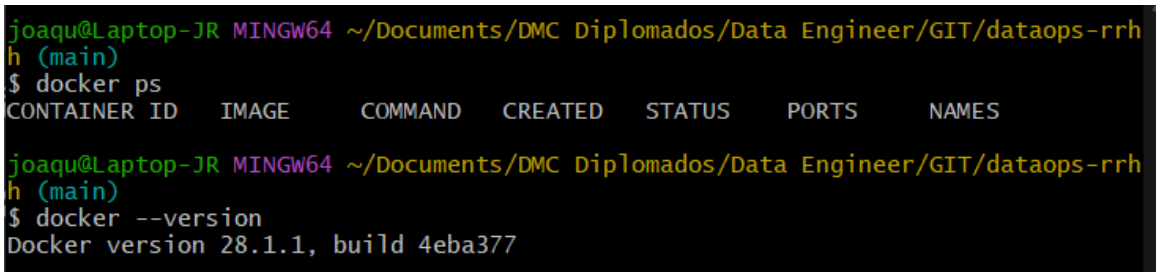
El desarrollo del proyecto se realizó en el sistema operativo Windows utilizando Visual Studio Code como entorno principal. Se creó un entorno virtual (venv) con Python 3.13.2 para gestionar las dependencias, las cuales fueron registradas en el archivo requirements.txt.

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.





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

```python
# === Configuración de conexión ===
db_user = "usr_ro_dmc_rrhh_estudiantes"
raw_pass = "fZp!jHt0j6%89^B4I*L*29bz4b^"
db_pass = urllib.parse.quote_plus(raw_pass)
db_host = "mgg.vps.webdock.cloud"
db_port = "5432"
db_name = "dmc"
schema = "rrhh"
table = "empleado"
```

```python
# === Crear engine SQLAlchemy ===
engine = sqlalchemy.create_engine(
    f"postgresql+psycopg2://{db_user}:{db_pass}@{db_host}:{db_port}/{db_name}"
)

# === Ejecutar consulta ===
query = f"SELECT * FROM {schema}.{table}"

try:
    df = pd.read_sql_query(query, engine)
except Exception as e:
    print("❌ Error al conectar o consultar:", e)
    exit()
```

Ejecución de transformaciones:

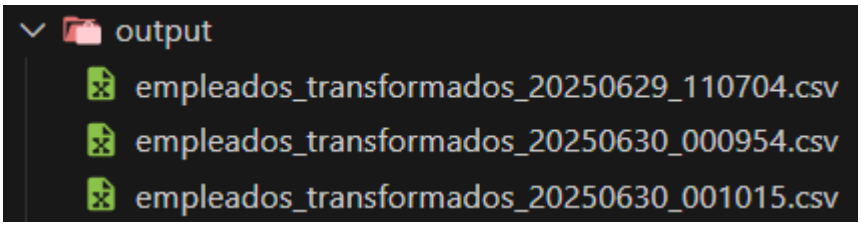
```python
# === Transformaciones ===
df["tip_documento"] = df["tip_documento"].str.upper()
df["nom_empleado"] = df["nom_empleado"].str.title()
df["ape_empleado"] = df["ape_empleado"].str.title()
df["nom_empleado_completo"] = df["nom_empleado"] + " " + df["ape_empleado"]
df["mnt_tope_comision"] = df["mnt_tope_comision"].fillna(0)
df["salario_anual"] = round(df["mnt_salario"] * 12, 2)

```

Disposición de archivo CSV:

```python
# === Exportar resultado ===
try:
    df.to_csv(output_file, index=False, encoding="utf-8-sig")
    print(f"✅ Archivo exportado correctamente en: {output_file}")
except Exception as e:
    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.



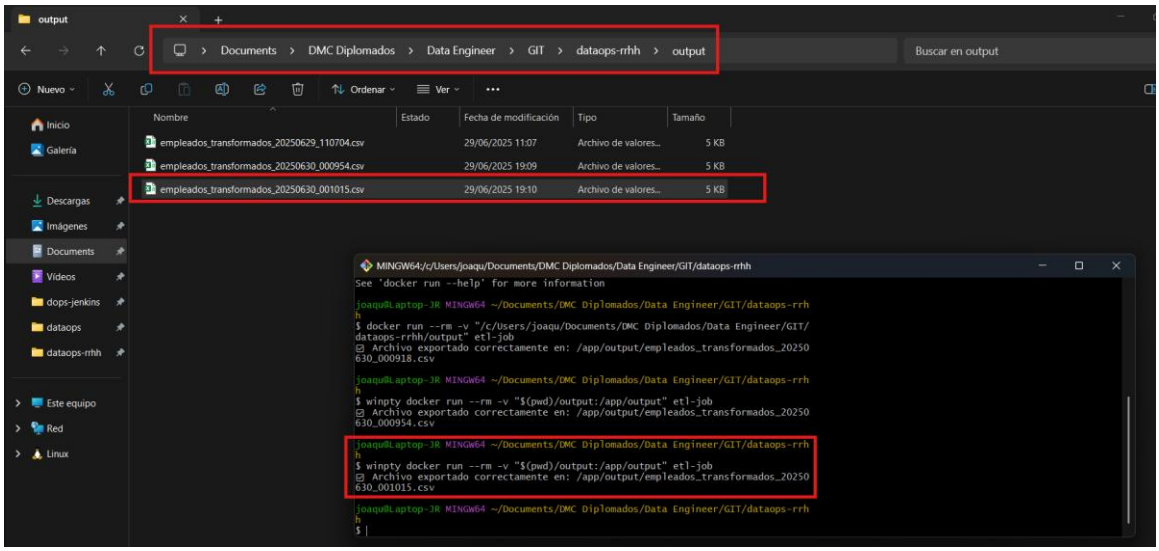

```csv
empleado_id,tip_documento,num_documento,nom_empleado,ape_empleado,cod_cargo,cod_departamento,mnt_salario,mnt_tope_comision,nom_empleado_completo,salario_anual
1,DNI,78901234,Ana,Pérez,CAR004,DEP004,1800.0,0.0,Ana Pérez,21600.0
2,CE,AB123456,Carlos,López,CAR002,DEP002,2500.5,300.0,Carlos López,30006.0
3,PASAPORTE,XYZ789,Sofía,Martínez,CAR003,DEP003,2200.75,0.0,Sofía Martínez,26409.0
4,DNI,12345678,Daniel,Gómez,CAR001,DEP001,4000.0,500.0,Daniel Gómez,48000.0
5,CE,CD987654,Elena,Rodríguez,CAR005,DEP005,2000.2,0.0,Elena Rodríguez,24002.4

```

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.

```dockerfile
# Imagen base con Python
FROM python:3.13-slim

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

# Copiar los archivos del proyecto al contenedor
COPY . .

# Instalar las dependencias del ETL
RUN pip install --no-cache-dir -r etl/requirements.txt

# Crear la carpeta de salida si no existe
RUN mkdir -p output

# Comando por defecto al ejecutar el contenedor
CMD ["python", "etl/main.py"]
```

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.

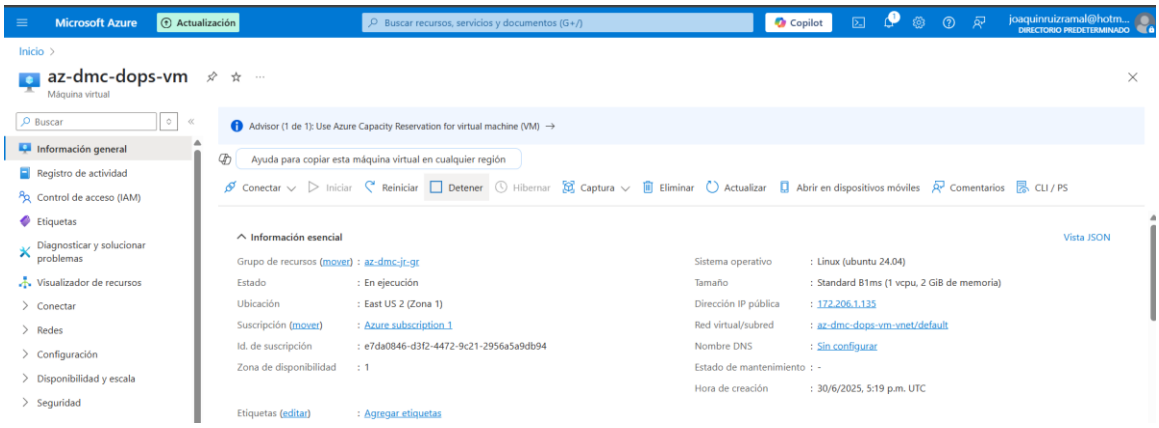


7. Automatización con Jenkins (CI/CD)

Automatización del entorno Jenkins

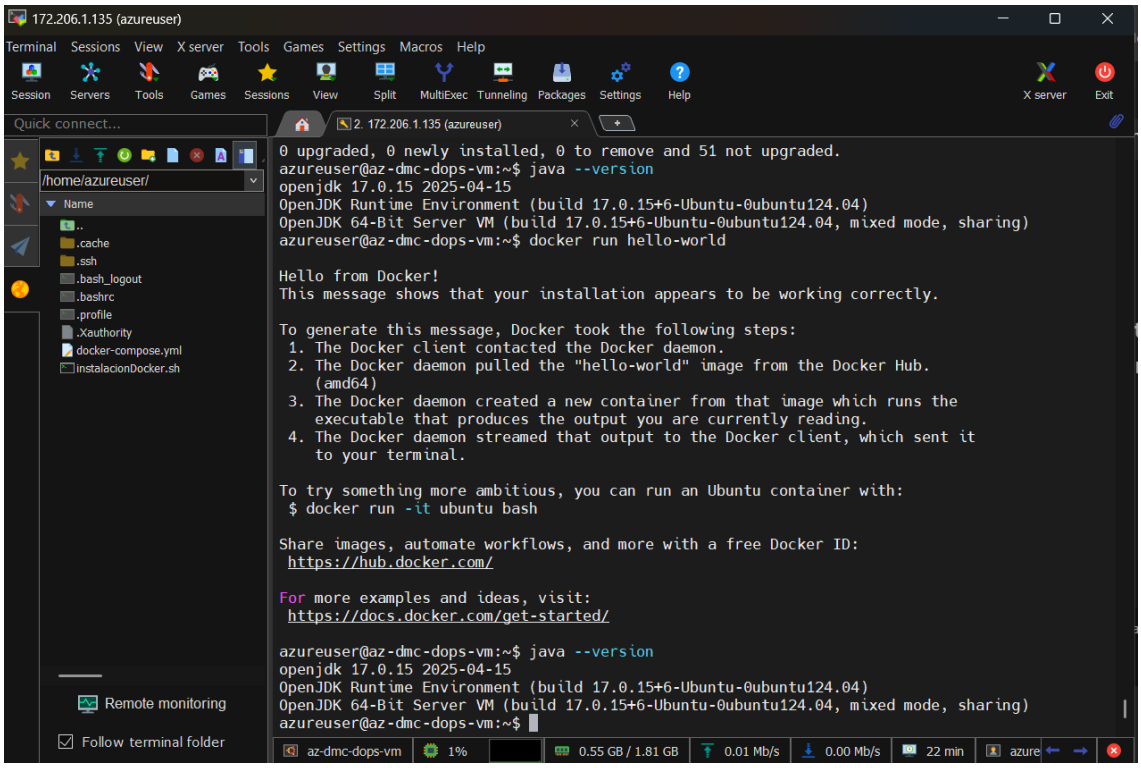
Para este proyecto se utilizó una máquina virtual con sistema operativo Linux (Ubuntu) en la que se instaló Docker, Java y Git utilizando un script automatizado llamado instalacionDocker.sh. Este archivo contiene todos los comandos necesarios

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



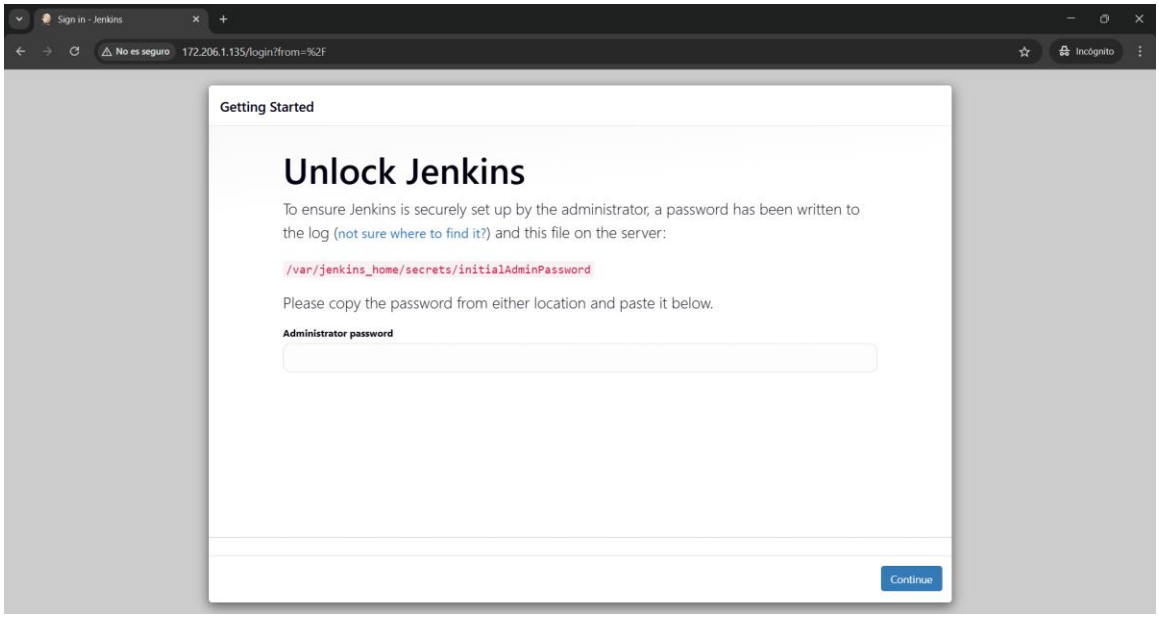

```
azureuser@az-dmc-dops-vm:~$ sh instalacionDocker.sh
-------------------------------------------------
> Instalando Java y Herramientas
-------------------------------------------------
git is already the newest version (1:2.43.0-1ubuntu7.2).
git set to manually installed.
The following additional 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-amber-dri libgl1-mesa-dri libglapi-mesa libglvnd0
  libglx-mesa0 libglx0 libgraphite2-3 libgtk2.0-0t64 libgtk2.0-bin
  libgtk2.0-common libharfbuzz0b libice-dev libice6 libjbig0 libjpeg-turbo8
  libjpeg8 liblcms2-2 liblerc4 libllvm19 libpango-1.0-0 libpangocairo-1.0-0
  libpangoft2-1.0-0 libpciaccess0 libpcsclite1 libpixman-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
```

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

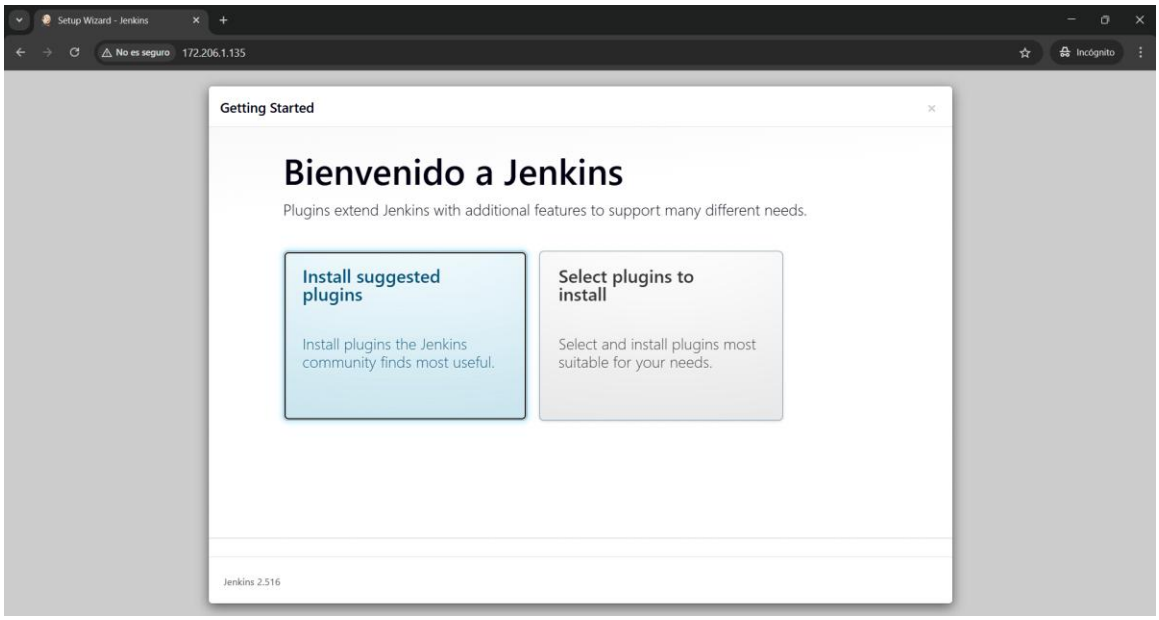



```
azureuser@az-dmc-dops-vm:~$ docker compose up -d
[+] Running 13/13
 ✔ jenkins Pulled                                                    20.5s
   ✔ 0c01110621e0 Pull complete                                       6.5s
   ✔ bbbe19972478 Pull complete                                      13.8s
   ✔ 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
```

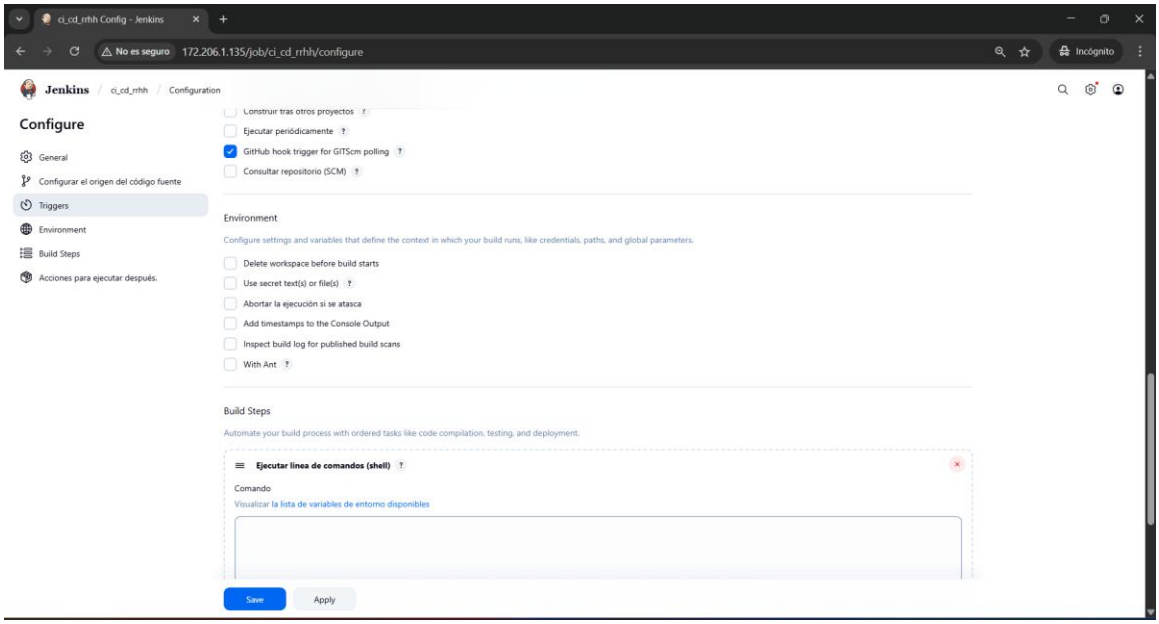
Una vez completada la instalación, Jenkins se desplegó como contenedor utilizando el archivo docker-compose.yml. Este archivo levanta el servicio en el puerto 80, accediendo a Jenkins desde el navegador.



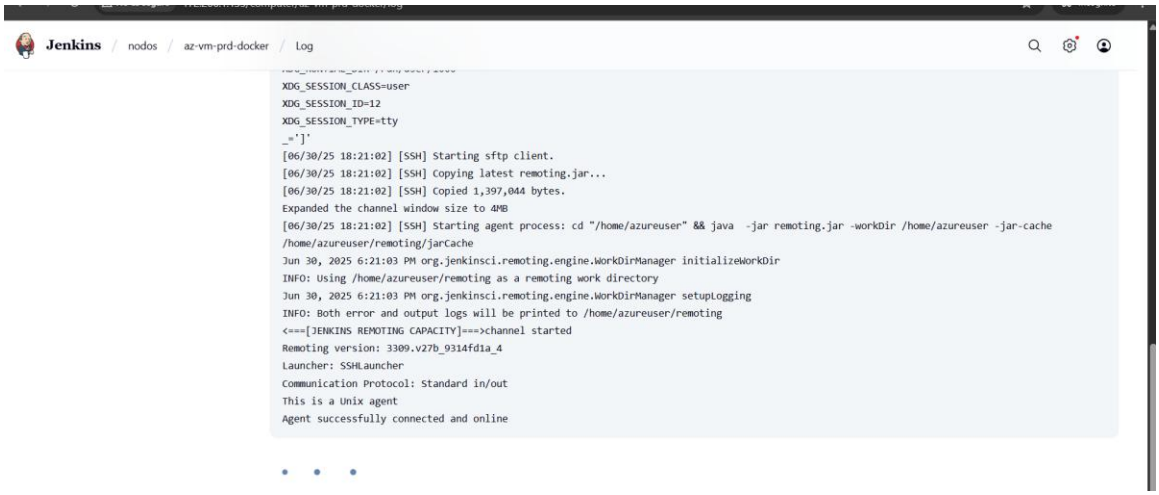
Se obtuvo la clave ejecutando la línea *docker logs jenkins*



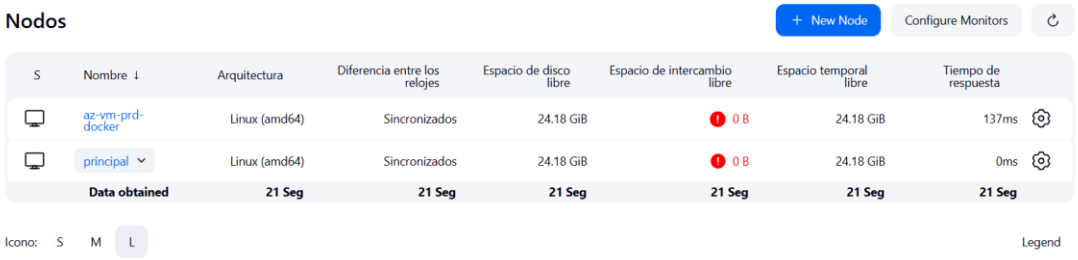
Desde la interfaz web de Jenkins se creó manualmente un proyecto, replicando el procedimiento visto en clase. Asegurándonos de marcar la casilla Github hook trigger for GITScm polling



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).

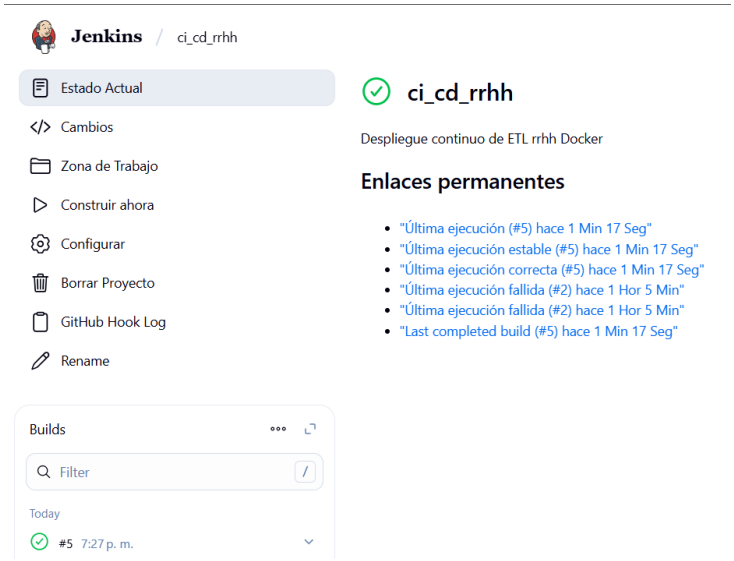


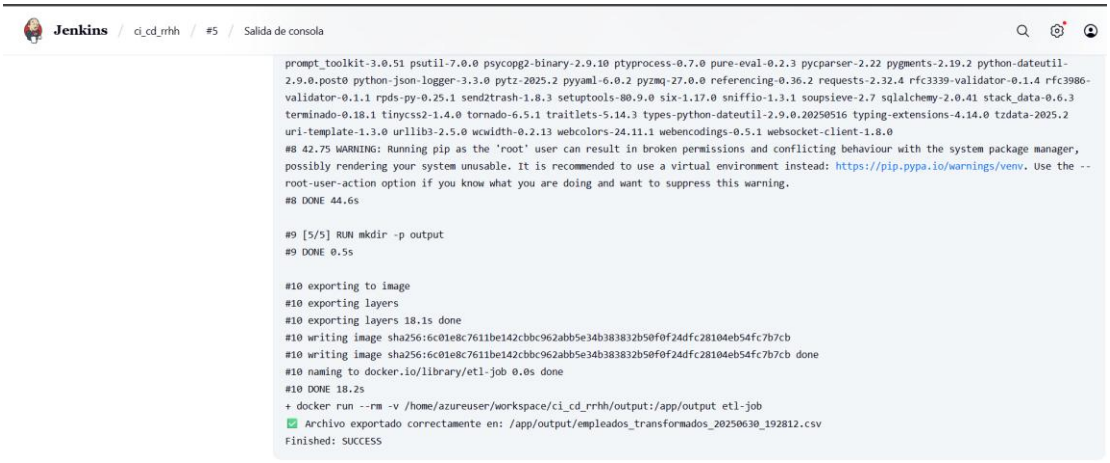
Se creó un segundo nodo az-vm-prd-docker, y se configuro el nodo principal para que solo ejecutara tareas asociadas al mismo.

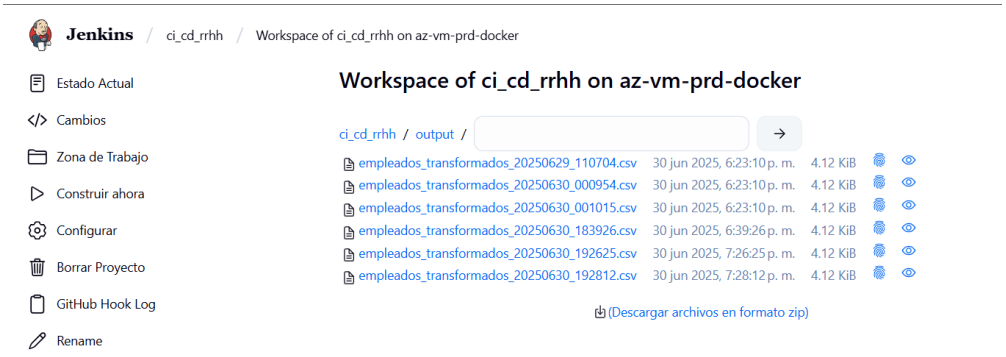


Se comprobó el funcionamiento de la tarea:

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)







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.