

Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e
Ingenierías

Computación Tolerante a Fallas

Mtro: Michell Lopez Franco

Juan Antonio Pérez Juárez

INCO

215660996

Airflow

Introducción:

En la actualidad, el mundo empresarial y tecnológico se enfrenta a desafíos cada vez más complejos en cuanto al manejo y procesamiento de datos. La automatización de procesos y la gestión eficiente de flujos de trabajo se han convertido en elementos fundamentales para cualquier organización que busque mantener su competitividad en la era digital.

Las empresas modernas generan y procesan cantidades masivas de datos diariamente, lo que ha creado la necesidad de desarrollar herramientas y sistemas que permitan gestionar estos procesos de manera eficiente, confiable y escalable. La orquestación de tareas y la automatización de procesos se han vuelto cruciales para mantener operaciones fluidas y reducir errores humanos.

Y debido a esas necesidades fue que nació Apache Airflow. Pero... ¿Y eso qué es?

Desarrollo:

Apache Airflow es una plataforma de gestión de flujo de trabajo de código abierto escrita en Python, donde los flujos de trabajo se crean a través de scripts de Python. Fue creada por Airbnb en octubre de 2014 como solución para la gestión de flujos de trabajo dentro de la empresa.

Al crear Airflow, Airbnb logró manejar más fácilmente sus flujos de trabajo y controlarlos gracias a la interfaz de usuario incluida en Airflow.

Desde el principio, el proyecto fue distribuido como código abierto, se convirtió en un proyecto de la Incubadora de Apache en marzo de 2016 y un proyecto de software de nivel superior de la Fundación Apache en enero de 2019.

Un pequeño inciso, Apache Software Foundation es una comunidad descentralizada de desarrolladores que trabajan cada uno en sus propios proyectos de código abierto. Los proyectos Apache se caracterizan por un modelo de desarrollo basado en el consenso y la colaboración y en una licencia de software abierta y pragmática. Cada proyecto es gestionado por un grupo auto seleccionado de expertos técnicos que son participantes activos en dicho proyecto. La ASF es una meritocracia, de lo que se deriva que la pertenencia a la fundación se permite solo a voluntarios que hayan contribuido significativamente a proyectos Apache.

Entre los objetivos de la ASF se encuentran el de proporcionar protección legal a los voluntarios que trabajan en proyectos Apache, y al propio nombre Apache de ser empleado por otras organizaciones. El proyecto Apache es el origen de la Licencia Apache y de todas las licencias que siguen un esquema similar (llamadas licencias "estilo Apache").

Me encanta la comunidad del código abierto, no por nada caí redondito en las redes de Linux, en fin, sigamos.

Airflow está escrito en Python, y los workflows son creados vía scripts en Python. Airflow Está diseñado bajo el principio de "configuración como código". Si bien hay otras plataformas de flujos de trabajo que también

trabajan bajo el principio "configuración como código", la mayoría utiliza XML. Al utilizarse Python en Airflow, los desarrolladores pueden importar bibliotecas y clases para ayudarles crear sus flujos de trabajo.

Airflow utiliza grafos acíclicos dirigidos (DAG) para gestionar la orquestación del flujo de trabajo. Las tareas y dependencias se definen en Python y luego Airflow gestiona la programación de tareas y la ejecución. Los DAG se pueden ejecutar en un horario definido (por ejemplo, cada hora o cada día) o en función de la ocurrencia de eventos externos (por ejemplo, un archivo que aparece en Hive). Los programadores anteriores basados en DAG como Oozie y Azkaban tendían a depender de múltiples archivos de configuración y árboles del sistema de archivos para crear un DAG, mientras que en Airflow, los DAG a menudo se pueden escribir en un solo archivo Python.

Airflow se puede utilizar para cualquier circuito de datos por lotes, por lo que sus casos de uso son tantos como diversos. Debido a su extensibilidad, esta plataforma destaca particularmente para organizar tareas con dependencias complejas en múltiples sistemas externos.

Al escribir circuitos en código y utilizar los diversos plugins disponibles, se puede integrar Airflow en cualquier sistema dependiente desde una plataforma unificada para la gestión y la supervisión.

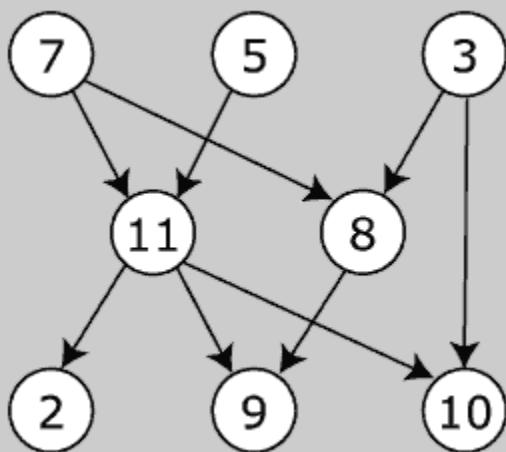
Como ejemplo, se puede utilizar Airflow para añadir las actualizaciones diarias de los equipos de ventas de Salesforce para enviar un informe diario a los ejecutivos de la empresa.

Además, la plataforma se puede utilizar para organizar y ejecutar tareas de Machine Learning que se ejecutan en clústeres Spark externos. También permite cargar datos de aplicaciones o sitios web en un Data Warehouse una vez por hora.

La arquitectura Airwave se basa en varios elementos. Estos son los principales.

Los DAG

Un DAG (Directed Acyclic Graph) es un circuito de datos definido en código Python. Cada DAG representa una secuencia de tareas por ejecutar, organizada para indicar las relaciones entre las tareas en la interfaz de usuario de Airflow.



Cada una de las tres palabras del acrónimo DAG corresponde a una propiedad de estas tareas. Están "directed" (dirigidas) porque las tareas deben tener al menos una tarea ascendente y una tarea descendente.

«Acyclic» (acíclicos), porque no se permite que las tareas creen datos de autorreferencia para evitar la creación de bucles infinitos. Finalmente, “Graph” (gráficos), porque las tareas se presentan en una estructura clara indicando sus relaciones.

Cada nodo de un DAG representa **una tarea**. Es una representación visual de los trabajos que se están ejecutando en cada etapa del flujo de trabajo. Los operadores son quienes definen los trabajos.

Los operadores son los componentes básicos de la plataforma Airflow. Permiten determinar el trabajo realizado. Se puede tratar de una tarea individual o del nodo de un DAG, que definirán cómo se ejecutará la tarea ejecutará la tarea.

El DAG ayuda a garantizar que los operadores se programen y ejecuten en un orden preciso, mientras que los operadores definen los trabajos que se realizarán en cada paso del proceso.

Hay tres categorías principales de operadores. En primer lugar, los operadores de acción realizan una función. Como ejemplo, citemos PythonOperator o BashOperator.

Los operadores de transferencia, por su parte, pueden transferir datos de una fuente a un destino, como S3ToRedshiftOperator.

Finalmente, los operadores de captura permanecen pasivos hasta que se detecta un evento. Este es el caso del ExternalTaskSensor.

Cada operador se define individualmente. Sin embargo, los operadores pueden comunicar información entre sí mediante XComs.

En Airflow, **los Hooks** permiten la interfaz con sistemas de terceros. Permiten conectarse a API y bases de datos externas como Hive, S3, GCS, MySQL, Postgres, etc.

La información confidencial, como las credenciales de inicio de sesión, se mantiene fuera de Hooks. Se almacenan en una base de metadatos cifrada asociada con la instancia actual de Airflow.

Los plugins de Airflow se pueden describir como una combinación de Hooks y operadores. Se utilizan para efectuar determinadas tareas específicas que implican a una aplicación externa.

Las «**conexiones**» permiten que Airflow almacene información, lo que permite conectarse a sistemas externos como identificadores o tokens API.

Se gestionan directamente desde la interfaz de usuario de la plataforma. Los datos se cifran y almacenan como metadatos en una base de datos Postgres o MySQL.

Ahora, como es de código abierto, no es una herramienta nativa de Windows, por lo que no es sencillo de ejecutar en dicho sistema operativo.

Y viendo un curso de Airflow y leyendo distintos repositorios y foros llegué a la conclusión de que lo mejor será ejecutarlo en un ambiente linux, que lo mas sencillo sería hacerlo en una laptop vieja que tengo con Bodhi Linux, pero es demasiado vieja y no tiene acceso a los repositorios nuevos de debian, así que navegando por ahí leí que incluso en linux, la mejor manera de utilizar Airflow es dentro de un contenedor, así que ahora tocará hablar de Docker.

Docker es una plataforma... Podría ser un vago y de este trabajo hacer 2 entregas, pero por el momento nos centraremos en Airflow, para el siguiente trabajo donde el uso de docker es un requerimiento. Lo desarrollaré más a fondo, así que sigamos con el desarrollo de Airflow

Por lo que para poder hacer se necesita la instalación de Docker, así que le dejo un pequeño material didáctico que me sirvió de Maravilla para poder hacerlo.

▶ Aprende Docker ahora! curso completo gratis desde cero! (Funciona mejor si lo pones en velocidad x 1.75)

Así que veamos como me quedó a mí. (No tomé capturas de pantalla, por lo que se muestra es el estado actual de mis contenedores)

The screenshot shows the Docker Desktop application interface. The left sidebar has options: Ask Gordon (BETA), Containers (selected), Images, Volumes, Builds, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' with a 'Give feedback' link. It displays usage statistics: Container CPU usage (3.39% / 800% (8 CPUs available)) and Container memory usage (1.4GB / 7.5GB). A search bar and a 'Only show running containers' toggle are present. A table lists four running containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last stat	Actions
<input type="checkbox"/>	airflow-webser	18a2c8f41061	apache/airf		0%	1 day ag	
<input type="checkbox"/>	airflow-webser	470b4bcc0d20	apache/airf		0%	1 day ag	
<input type="checkbox"/>	airflow-webser	408e291d2afa	apache/airf		0%	1 day ag	
<input checked="" type="checkbox"/>	airflow-docker	-	-		2.74%	1 day ag	

At the bottom, it says 'Showing 4 items'. Below the table is a 'Walkthroughs' section with a close button. The taskbar at the bottom shows various icons and the system tray indicates 'Engine running', 'RAM 5.99 GB CPU 0.37%', 'Disk: 3.91 GB used (limit 1006.85 GB)', 'Terminal v4.40.0', and the date/time '5/10/2025 12:08 AM'.

El diseño de la UI es precioso, bueno, pero lo importante ocurre desde la terminal. Eso lo puedes hacer desde la CMD de windows, aquí algunos pantallazos de mi desarrollo.

```
C:\Users\AnthemKGR\Desktop\airflow-docker>docker-compose run airflow-webserver airflow dags list
time="2025-05-09T00:01:02-06:00" level=warning msg="C:\\Users\\AnthemKGR\\Desktop\\\\airflow-docker\\\\docker-compose
.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
time="2025-05-09T00:01:03-06:00" level=warning msg="Found orphan containers ([airflow-docker-airflow-webserver-ru
n-565acd67d5d5 airflow-docker-airflow-webserver-run-39c83aceadf8]) for this project. If you removed or renamed th
e service in your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Creating 1/1
  Container airflow-docker-postgres-1  Running          0.0s
Container CPU usage/58% / 800%
/home/airflow/.local/lib/python3.8/site-packages/airflow/configuration.py:829 DeprecationWarning: The sql_alchemy_
_conn option in [core] has been moved to the sqlalchemy_conn option in [database] - the old setting has been used,
but please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/configuration.py:735 DeprecationWarning: The sql_alchemy_
_conn option in [core] has been moved to the sqlalchemy_conn option in [database] - the old setting has been used,
but please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/settings.py:195 DeprecationWarning: The sqlalchemy_conn
option in [core] has been moved to the sqlalchemy_conn option in [database] - the old setting has been used, bu
t please update your config.
/home/airflow/.local/lib/python3.8/site-packages/airflow/models/base.py:71 DeprecationWarning: The sqlalchemy_co
nn option in [core] has been moved to the sqlalchemy_conn option in [database] - the old setting has been used,
but please update your config.

dag_id                                | filepath                               | owner   | paused
=====+=====+=====+=====
analisis_datos_ejemplo                 | ejemplo_analitica.py                  | tu_nombre | True
dataset_consumes_1                      | /home/airflow/.local/lib/python3.8/site-pac
ages/airflow/example_dags/example_datasets.p
y
dataset_consumes_1_and_2                | /home/airflow/.local/lib/python3.8/site-pac
ages/airflow/example_dags/example_datasets.p
y
dataset_consumes_1_never_scheduled     | /home/airflow/.local/lib/python3.8/site-pac
ages/airflow/example_dags/example_datasets.p
y
dataset_consumes_unknown_never_scheduled| /home/airflow/.local/lib/python3.8/site-pac
ages/airflow/example_dags/example_datasets.p
y
dataset_produces_1                     | /home/airflow/.local/lib/python3.8/site-pac
ages/airflow/example_dags/example_datasets.p
y
```

```
Command Prompt

example_time_delta_sensor_async           _decorator.py          airflow    True
                                         /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_time_delta
                                         _sensor_async.py
example_trigger_controller_dag            /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_trigger_co
                                         ntroller_dag.py
example_trigger_target_dag                /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_trigger_ta
                                         rget_dag.py
example_weekday_branch_operator           /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_branch_day
                                         _of_week_operator.py
example_xcom                            /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_xcom.py
example_xcom_args                         /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_xcomargs.p
                                         y
example_xcom_args_with_operators         /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_xcomargs.p
                                         y
latest_only                             /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_latest_onl
                                         y.py
latest_only_with_trigger                 /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/example_latest_onl
                                         y_with_trigger.py
tutorial                                /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/tutorial.py
tutorial_dag                            /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/tutorial_dag.py
tutorial_taskflow_api                   /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/tutorial_taskflow_
                                         _api.py
tutorial_taskflow_api_virtualenv        /home/airflow/.local/lib/python3.8/site-pac
                                         ages/airflow/example_dags/tutorial_taskflow_
                                         _api_virtualenv.py

.61% C:\Users\AnthemKGR\Desktop\airflow-docker>
```

Que para el desarrollo de esta tarea utilicé Cursor que es un editor de texto de Microsoft con una IA integrada supuestamente mejor que GitHub Copilot, lo que es relativamente cierto.
Pero me sigo quedando con visual Studio Code.

Ahora veamos la raíz de mis archivos que utilicé para Docker.

```

AIRFLOW-DOCKER
  dags
    __pycache__
    ejemplo_analitica.py
  logs
    dag_id=analisis_datos_ejemplo
    dag_processor_manager
    scheduler
  plugins
  docker-compose.yaml

```

```

services:
  airflow-init:
    <<: *airflow-common
    entrypoint: /bin/bash
    command:
      - -c
      - |
        mkdir -p /opt/airflow/logs
        chown -R "airflow:airflow" /opt/airflow/logs
        exec airflow init
        airflow user
          --username airflow
          --firstname airflow

```

En la imagen anterior podemos ver que tengo varios warnings, que python no reconoce por que no tengo instaladas las dependencias necesarias, pero eso no importa, por que no lo estoy desplegando en mi pc, sinó que la estoy desplegando en un contenedor el cual parte de una distribución de linux y en la cual está corriendo Airflow, lo único que tengo que hacer es inicializar mi contenedor y exponer el puerto donde me voy a conectar de ambos lados, de uno el mío y de otro el del contenedor al cual me acoplaré. Por así decir.

Para ello automatizo la creación del contenedor y la configuración con un archivo Docker-Compose.yaml. veamos el código.

```

Python
# Versión de Docker Compose a utilizar
version: '3'

# Definición de configuraciones comunes para los servicios de Airflow
x-airflow-common:
  &airflow-common # Ancla para referenciar esta configuración en otros servicios
  image: apache/airflow:2.7.1 # Imagen base de Airflow a utilizar
  environment:
    &airflow-common-env # Variables de entorno comunes
    AIRFLOW__CORE__EXECUTOR: LocalExecutor # Tipo de ejecutor (local en este caso)
    AIRFLOW__CORE__SQLALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
  # Conexión a la base de datos
  AIRFLOW__CORE__FERNET_KEY: '' # Clave para encriptar conexiones sensibles
  AIRFLOW__CORE__LOAD_EXAMPLES: 'true' # Cargar DAGs de ejemplo
  AIRFLOW__WEBSERVER__SECRET_KEY: 'this_is_a_secret_key' # Clave secreta para el webserver

  user: "${AIRFLOW_UID}:0" # Usuario con el que se ejecutará Airflow

```

```
# Mapeo de volúmenes para persistencia de datos
volumes:
  - ./dags:/opt/airflow/dags # Directorio para los DAGs
  - ./logs:/opt/airflow/logs # Directorio para logs
  - ./plugins:/opt/airflow/plugins # Directorio para plugins

# Dependencias necesarias antes de iniciar
depends_on:
  &airflow-common-depends-on
  postgres:
    condition: service_healthy

# Definición de los servicios
services:
  # Servicio de base de datos PostgreSQL
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data # Volumen para persistencia de datos
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"] # Comprueba si PostgreSQL está listo
      interval: 5s
      retries: 5
    restart: always

  # Servicio del servidor web de Airflow
  airflow-webserver:
    <<: *airflow-common # Hereda la configuración común
    command: webserver
    ports:
      - "8080:8080" # Puerto para acceder a la interfaz web
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always
    depends_on:
      <<: *airflow-common-depends-on

  # Servicio del planificador de Airflow
  airflow-scheduler:
    <<: *airflow-common # Hereda la configuración común
    command: scheduler
```

```

healthcheck:
  test: ["CMD-SHELL", 'airflow jobs check --job-type SchedulerJob --hostname
"${HOSTNAME}"']
    interval: 10s
    timeout: 10s
    retries: 5
  restart: always
  depends_on:
    <<: *airflow-common-depends-on

# Servicio de inicialización de Airflow
airflow-init:
  <<: *airflow-common # Hereda la configuración común
  entrypoint: /bin/bash
  command:
    - -c
    - |
      # Script de inicialización
      mkdir -p /sources/logs /sources/dags /sources/plugins
      chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
      # Inicializa la base de datos y crea el usuario administrador
      exec airflow db init && \
      airflow users create \
        --username admin \
        --firstname Admin \
        --lastname User \
        --role Admin \
        --email admin@example.com \
        --password admin
  environment:
    <<: *airflow-common-env
  depends_on:
    <<: *airflow-common-depends-on

# Definición de volúmenes persistentes
volumes:
  postgres-db-volume: # Volumen para almacenar datos de PostgreSQL

```

Simple y sencillo, la mayor parte de este código lo desarollé solo a partir del video anteriormente citado, necesité algunas ayudas de IA al momento del login del usuario.

Y una vez configurado, me apareció el home page de Airflow.

Ya dentro puedes correr varios DAGS de ejemplo que tienen para que los puedas utilizar como referencia.

Así que busqué un script que fuera más como una animación para poder ver el como funciona, por lo que lo agregué a la carpeta de dags en la raíz y lo ejecuté lo que en airflow me muestra es eso:

Cuyo código es este:

```
Python
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta
import random
import time

# Argumentos por defecto para el DAG
default_args = {
```

```
'owner': 'tu_nombre',
'depends_on_past': False,
'start_date': datetime(2024, 5, 8),
'email': ['tu_correo@ejemplo.com'],
'email_on_failure': False,
'email_on_retry': False,
'retries': 1,
'retry_delay': timedelta(minutes=5),
}

# Crear el DAG
dag = DAG(
    'analisis_datos_ejemplo',
    default_args=default_args,
    description='Un DAG de ejemplo para análisis de datos',
    schedule_interval=timedelta(days=1),
    catchup=False,
    tags=['ejemplo', 'tutorial']
)

# Función para simular extracción de datos
def extraer_datos(**context):
    time.sleep(5) # Simular proceso
    return {'datos_extraidos': random.randint(100, 1000)}

# Función para simular procesamiento
def procesar_datos(**context):
    ti = context['task_instance']
    datos_extraidos = ti.xcom_pull(task_ids='extraccion_datos')['datos_extraidos']
    time.sleep(3) # Simular proceso
    return {'datos_procesados': datos_extraidos * 2}

# Función para simular análisis
def analizar_datos(**context):
    ti = context['task_instance']
    datos_procesados = ti.xcom_pull(task_ids='procesamiento_datos')['datos_procesados']
    time.sleep(4) # Simular proceso
    return {'resultado_analisis': f'Análisis completado con valor: {datos_procesados}'}

# Función para generar reporte
def generar_reporte(**context):
    ti = context['task_instance']
    analisis = ti.xcom_pull(task_ids='analisis_datos')['resultado_analisis']
    time.sleep(3) # Simular proceso
    return {'reporte': f'Reporte generado basado en: {analisis}'}

# Tareas
inicio = BashOperator(
```

```

    task_id='inicio',
    bash_command='echo "Iniciando pipeline de análisis: $(date)"',
    dag=dag
)

extraccion = PythonOperator(
    task_id='extraccion_datos',
    python_callable=extraer_datos,
    dag=dag
)

procesamiento = PythonOperator(
    task_id='procesamiento_datos',
    python_callable=procesar_datos,
    dag=dag
)

analisis = PythonOperator(
    task_id='analisis_datos',
    python_callable=analizar_datos,
    dag=dag
)

reporte = PythonOperator(
    task_id='generar_reporte',
    python_callable=generar_reporte,
    dag=dag
)

fin = BashOperator(
    task_id='fin',
    bash_command='echo "Pipeline completado exitosamente: $(date)"',
    dag=dag
)

# Definir el orden de las tareas
inicio >> extraccion >> procesamiento >> analisis >> reporte >> fin

```

Lo que me arroja en Airflow es este resultado:

Cursos Entretimiento ASI CONSE Tamash Resumen d Airflow is Ejemplo ut COMPUTA SharkEng analis + - X

http://localhost:8080/dags/analisis_datos_ejemplo/grid

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs

Schedule: 1 day, 0:00:00 Next Run: 2025-05-10, 06:00:57

DAG: analisis_datos_ejemplo Un DAG de ejemplo para análisis de datos

Grid Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code Audit Log

05/10/2025 06:30:38 AM 25 All Run Types All Run States Clear Filters

Duration 00:00:31 00:00:15 00:00:00

Press Shift + / for Shortcuts

defered failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status

DAG analisis_datos_ejemplo

Details Graph Gantt Code

DAG Runs Summary

Total Runs Displayed	3
■ Total success	3

First Run Start 2025-05-09, 06:01:28 UTC

Last Run Start 2025-05-10, 06:00:58 UTC

Max Run Duration 00:00:31

Mean Run Duration 00:00:29

Min Run Duration 00:00:29

DAG Summary

Total Tasks	6
-------------	---

inicio extraccion_datos procesamiento_datos analisis_datos generar_reporte fin

Eso como vista general, en cambio, sí lo muestro cómo grafo se ve algo así:

Cursos Entretimiento ASI CONSE Tamash Resumen d Airflow is Ejemplo ut COMPUTA SharkEng analis + - X

http://localhost:8080/dags/analisis_datos_ejemplo/grid?tab=graph&dag_run_id=manual_20...

Auto-refresh

05/10/2025 06:31:48 AM 25 All Run Types All Run States Clear Filters

defered failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status

DAG analisis_datos_ejemplo / Run 2025-05-09, 06:01:38 UTC

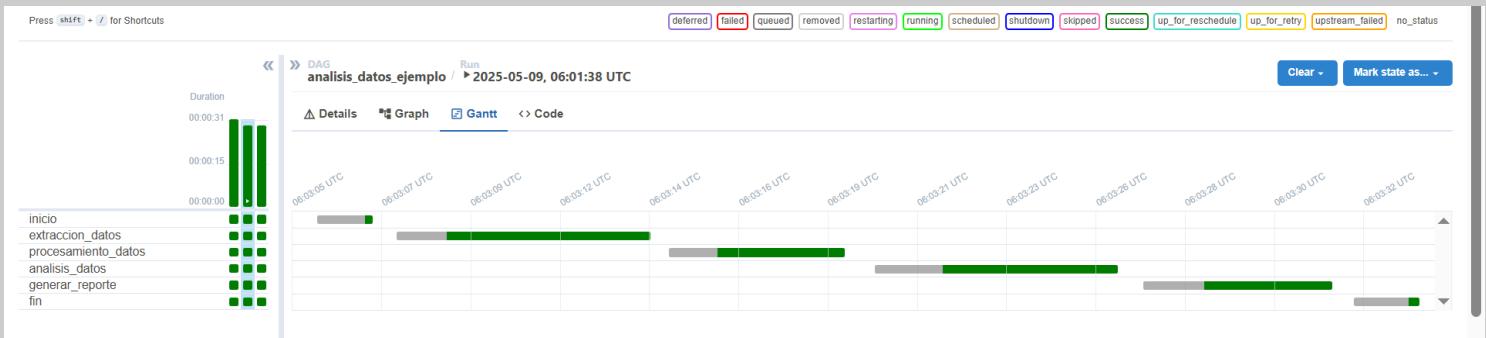
Details Graph Gantt Code

Layout: Left -> Right

inicio extraccion_datos procesamiento_datos analisis_datos generar_reporte fin

```
graph LR; inicio[BEGIN] --> extraccion_datos[extraccion_datos<br/>PythonOperator]; extraccion_datos --> procesamiento_datos[procesamiento_datos<br/>PythonOperator]; procesamiento_datos --> analisis_datos[analisis_datos<br/>PythonOperator]; analisis_datos --> generar_reporte[generar_reporte<br/>PythonOperator]; generar_reporte --> fin[END]
```

Y tiene algo que me impresionó, es una tontería pero para mí fue increíble:



De Locos...

Pero en fin.

Solo es un pequeño script que trata de simular varias tareas, las tareas solo tienen un proceso de espera, le dicen que espere cierto tiempo antes de pasar a la siguiente por lo que es super sencillo de hacer y de emular, lo más difícil sería hacer el contenedor desde 0. Lo de airflow es super sencillo.

Conclusión:

Esta actividad me encantó, desde que me puse serio a hacerla me encantó.

Y me dió la oportunidad de matar dos pájaros de un tiro, por que puedo hacer un docker con más facilidad lo cual me ayuda a hacer la actividad siguiente con mayor facilidad.

Aprendí mucho, siento que esa es la manera en que debes aprender, por curiosidad auténtica, ya que para este trabajo casi no utilicé IA, lo cual me sorprendió mucho.

Y quizá no sea lo que yo quiera utilizar como profesional por qué no me agradan mucho las ciencias de datos ni el big data, pero ahora que lo conozco desde dentro creo que debería darle una oportunidad para probarme y quizá ganar mucho dinero de ello.

Referencias:

colaboradores de Wikipedia. (2025, January 19). Apache Software Foundation. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Apache_Software_Foundation

colaboradores de Wikipedia. (2024, March 2). Apache Airflow. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Apache_Airflow

HolaMundo. (2022, July 7). Aprende Docker ahora! curso completo gratis desde cero! [Video]. YouTube. <https://www.youtube.com/watch?v=4Dko5W96WHg>

Daniel. (2023, October 30). Apache Airflow. DataScientest. <https://datascientest.com/es/todo-sobre-apache-airflow>

Public Interface of Airflow – Airflow Documentation. (n.d.). <https://airflow.apache.org/docs/apache-airflow/stable/public-airflow-interface.html>