

ESPECIALIZACIÓN EN MACHINE LEARNING ENGINEER

Tema: Fundamentos de MLE – parte 3

Docente: Daniel Chavez Gallo

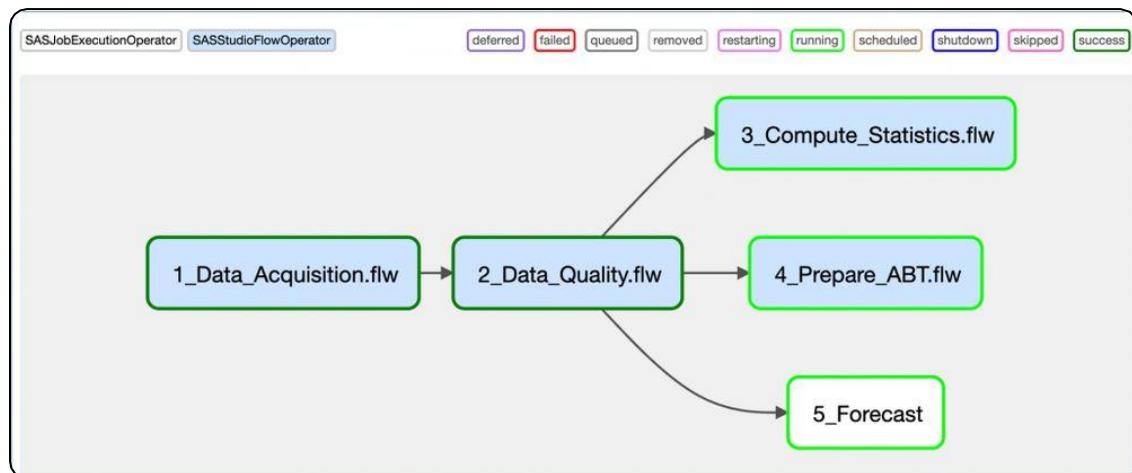


Apache Airflow : Definición

Apache Airflow es una potente **plataforma de código abierto** diseñada para la **orquestación y automatización de flujos de trabajo** complejos y tareas programadas. En el corazón de Apache Airflow se encuentran los **DAGs (Directed Acyclic Graphs)**, que permiten a los usuarios definir una colección de tareas y sus dependencias de una manera clara y lógica.

Airflow se ha convertido en una herramienta esencial para **Data Engineer** y **Machine Learning Engineer**, ya que facilita la **programación, organización y monitoreo** de flujos de trabajo complejos de manera eficiente y escalable.

Los **DAGs** de Airflow permiten a los usuarios **establecer secuencias de tareas** que pueden ejecutarse de forma paralela o secuencial, asegurando que cada tarea se ejecute en el **orden correcto** y solo después de que se hayan completado todas sus dependencias. Esta capacidad de gestionar y automatizar tareas interdependientes hace de Airflow una herramienta indispensable para proyectos que van desde simples pipelines de datos hasta complejos ecosistemas.





Apache Airflow : Historia

Apache Airflow fue creado por **Maxime Beauchemin en Airbnb en 2014** como una solución interna para **manejar las crecientes necesidades de orquestación de flujos de trabajo de la compañía**. En sus primeros días, Airbnb enfrentaba desafíos significativos en la gestión de tareas interdependientes y flujos de trabajo relacionados con el procesamiento de datos y la ingeniería de machine learning. Estos desafíos requerían una solución que no solo pudiera programar tareas de manera confiable y eficiente, sino que también proporcionara **visibilidad en la ejecución** de estas tareas y permitiera la **fácil reutilización y mantenimiento** de los flujos de trabajo.



Los **flujos de trabajo se definen en archivos Python**, lo que permite una gran **flexibilidad, mantenibilidad y versionabilidad**. Esta aproximación fue un cambio significativo respecto a otras herramientas de orquestación que requerían configuraciones estáticas y menos flexibles.

Debido a su diseño robusto y flexibilidad, Airflow ganó rápidamente popularidad y fue adoptado por una variedad de empresas y organizaciones de diferentes tamaños e industrias. En **2016, Airflow fue donado a la Apache Software Foundation**, donde continuó evolucionando y ganando características gracias a una comunidad activa y creciente. Bajo el paraguas de Apache, Airflow alcanzó su estatus como un proyecto de nivel superior, reflejando su madurez, estabilidad y la amplia adopción en la industria.



Apache Airflow : Conceptos

DAGs (Directed Acyclic Graphs)

Descripción: Los DAGs son el corazón de Apache Airflow. Un DAG es un **conjunto de tareas con direccionalidad** que indica el orden en el que deben ejecutarse, y sin ciclos, asegurando que **no hayan dependencias circulares** que podrían causar ejecuciones infinitas. Cada nodo en el grafo representa una tarea, y las aristas definen las dependencias entre estas tareas.

Uso en Airflow: En Airflow, un DAG es **definido en Python** y representa un flujo de trabajo que quieras automatizar y monitorear. Dentro del archivo Python, se **definen las tareas individuales y sus dependencias**. Airflow luego se encarga de programar y ejecutar estas tareas de acuerdo con las dependencias establecidas, asegurando que se ejecuten en el orden correcto.

Operadores

Descripción: Los operadores en Airflow son las clases predefinidas que **definen la tarea que se va a ejecutar**. Cada operador está diseñado para realizar una función específica.

Tipos de Operadores:

BashOperator : Ejecuta un comando Bash.

PythonOperator : Ejecuta una función Python.

SqlOperator : Ejecuta una consulta SQL.

DockerOperator : Ejecuta un contenedor Docker.

Y muchos otros, cada uno adaptado para integrarse con diferentes herramientas y sistemas.

Uso en Airflow: Seleccionas el operador que coincide con la **tarea que necesitas realizar dentro de tu DAG**. Por ejemplo, si necesitas ejecutar un script Python como parte de tu flujo de trabajo, usarías el PythonOperator.



Apache Airflow : Conceptos

Tareas y dependencias

Tareas: En el contexto de un DAG, **una tarea es una instancia de un operador**. Es la unidad básica de trabajo en Airflow.

Definición de Dependencias: Las dependencias entre tareas se definen utilizando los **operadores bitshift >> y <<**. Por ejemplo, tarea1 >> tarea2 indica que tarea1 debe completarse antes de que tarea2 pueda comenzar.

Ejemplo: Esto permite una gran flexibilidad en la definición de flujos de trabajo, pudiendo fácilmente **orquestar tareas que dependen de la finalización de otras**, incluyendo la capacidad de definir múltiples tareas que pueden ejecutarse en paralelo si no tienen dependencias entre sí.

Programación y triggers

Programación de DAGs: Los DAGs en Airflow se pueden programar para **ejecutarse en intervalos regulares**, lo que se define mediante el parámetro **schedule_interval** en la definición del DAG. Este parámetro puede tomar varios formatos, desde cron-like strings hasta timedelta objects de Python, permitiendo una gran precisión en la frecuencia de ejecución.

Triggers Externos: Además de la programación basada en intervalos, Airflow permite la activación (o "trigger") de DAGs en **respuesta a eventos externos**. Esto se puede lograr mediante la **API de Airflow** o **mediante el uso de sensores**, que son tipos especiales de tareas que esperan a que se cumpla una condición externa antes de completarse y permitir que continúe el flujo de trabajo.

start_date: Es otro parámetro importante en la definición de un DAG, que indica la **fecha y hora en que el DAG debe comenzar a ejecutarse**. Airflow no ejecutará tareas en un DAG antes de su start_date, lo que ofrece control sobre el inicio de las operaciones automatizadas.



Apache Airflow : Instalación

<https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html>

The screenshot shows the Apache Airflow documentation page for Docker Compose. The header includes the Apache Airflow logo and navigation links for Community, Meetups, Documentation, Use-cases, Announcements, Blog, and Ecosystem. A sidebar on the left shows the version as 2.8.3 and provides a search bar for 'Search docs'. The main content area is titled 'Fetching docker-compose.yaml' and instructs users to fetch the file using the command 'curl -Lfo 'https://airflow.apache.org/docs/apache-airflow/2.8.3/docker-compose.yaml''. Below this, there is a terminal window showing the command being run.

Version: 2.8.3

Search docs

CONTENT

Fetching `docker-compose.yaml`

To deploy Airflow on Docker Compose, you should fetch [docker-compose.yaml](https://airflow.apache.org/docs/apache-airflow/2.8.3/docker-compose.yaml).

```
curl -Lfo 'https://airflow.apache.org/docs/apache-airflow/2.8.3/docker-compose.yaml'
```

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC
$ cd Airflow_example

usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)
$ curl -Lfo 'https://airflow.apache.org/docs/apache-airflow/2.8.3/docker-compose.yaml'
% Total    % Received % Xferd  Average Speed   Time   Time     Time Current
          Dload  Upload Total Spent   Left Speed
100 10940  100 10940    0      0  26884      0 --:--:-- --:--:-- 27079
```



Apache Airflow : Instalación

Sección x-airflow-common

Define un conjunto de configuraciones comunes que se aplicarán a varios servicios de Airflow en este docker-compose. Esta es una técnica en docker-compose para **evitar la repetición** y mantener el archivo limpio.

Cualquier servicio que necesite estas configuraciones puede referenciarlas con `<<: *airflow-common.`

image: Define la imagen de Docker que se usará para los contenedores de Airflow. Aquí se puede especificar una imagen personalizada o usar una existente.

environment: Establece variables de entorno necesarias para la configuración de Airflow, como el tipo de ejecutor, las conexiones a la base de datos PostgreSQL y Redis, y otros ajustes.

volumes: Monta directorios del host en el contenedor para DAGs, registros, configuraciones y plugins de Airflow.

user: Define el usuario y grupo bajo el cual se ejecutarán los servicios.

depends_on: Especifica las dependencias de otros servicios (como redis y postgres) que deben estar operativos antes de iniciar este servicio.

```
x.airflow-common :  
  &airflow-common  
image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.8.3}  
environment :  
  &airflow-common-env  
    AIRFLOW__CORE__EXECUTOR : CeleryExecutor  
    AIRFLOW__DATABASE__SQLALCHEMY_CONN :  
      postgresql+psycopg2://airflow:airflow@postgres/airflow  
    AIRFLOW__CELERY__RESULT_BACKEND :  
      db+postgresql://airflow:airflow@postgres/airflow  
    AIRFLOW__CELERY__BROKER_URL : redis://:@redis:6379/0  
    AIRFLOW__CORE__FERNET_KEY : ''  
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION : 'true'  
    AIRFLOW__CORE__LOAD_EXAMPLES : 'true'  
    AIRFLOW__API__AUTH_BACKENDS :  
      airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'  
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK : 'true'  
  _PIP_ADDITIONAL_REQUIREMENTS : ${_PIP_ADDITIONAL_REQUIREMENTS:-}  
volumes :  
  - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags  
  - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs  
  - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config  
  - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins  
user: "${AIRFLOW_UID:-50000}:0"  
depends_on :  
  &airflow-common-depends-on  
  redis:  
    condition: service_healthy  
  postgres:  
    condition: service_healthy
```



Apache Airflow : Instalación

Sección services

Define los servicios individuales que componen tu aplicación Airflow.

postgres: Es la base de datos para Airflow. Usa la imagen oficial de PostgreSQL, establece las credenciales de la base de datos y define un volumen para la persistencia de datos.

redis: Actúa como broker de mensajes para el CeleryExecutor de Airflow. Usa la imagen oficial de Redis y configura un chequeo de salud.

airflow-webserver: Es el servidor web de Airflow que proporciona la UI. Utiliza las configuraciones comunes y establece comandos específicos y puertos, además de configuraciones de salud.

airflow-scheduler: El programador de Airflow que gestiona la programación de los DAGs. También usa las configuraciones comunes y tiene su propio chequeo de salud.

airflow-worker: Los workers que ejecutan las tareas. Necesarios cuando se usa CeleryExecutor. Incluye configuraciones para la correcta terminación y reinicio.

airflow-triggerer: Para la versión de Airflow que utiliza triggerers, este servicio gestiona los triggers.

airflow-init: Se usa para inicializar o actualizar la base de datos de Airflow antes de que se inicien los otros servicios.

airflow-cli: Un servicio para interactuar con la CLI de Airflow. Puede ser útil para depurar o ejecutar comandos de Airflow manualmente.

flower: Una herramienta de monitoreo para Celery. Proporciona una interfaz para monitorear la cola de tareas de Celery usada por Airflow.

```
services :  
postgres :  
    image: postgres:13  
    ...  
redis :  
    image: redis:latest  
    ...  
airflow-webserver :  
    <<: *airflow-common  
    command: webserver  
    ports :  
        - "8080:8080"  
    ...  
airflow-scheduler :  
    <<: *airflow-common  
    command: scheduler  
    ...  
airflow-worker :  
    <<: *airflow-common  
    command: celery worker  
    ...  
airflow-triggerer :  
    <<: *airflow-common  
    command: triggerer  
    ...  
airflow-init :  
    <<: *airflow-common  
    entrypoint: /bin/bash  
    ...  
airflow-cli :  
    <<: *airflow-common  
    ...  
flower :  
    <<: *airflow-common  
    ...
```



Apache Airflow : Instalación

Sección volumes

Define los volúmenes de Docker que se utilizan para la persistencia de datos. Por ejemplo, postgres-db-volume se usa para almacenar los datos de la base de datos de PostgreSQL de manera persistente, de modo que no se pierdan cuando el contenedor se detenga o se reinicie.

```
volumes:  
  postgres-db-volume:
```



Apache Airflow : Instalación

docker-compose.yaml U X

Airflow_example > docker-compose.yaml

```
47      x-airflow-common:  
72          volumes:  
73              - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags  
74              - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs  
75              - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config  
76              - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins  
77          user: "${AIRFLOW_UID:-50000}:0"  
78      depends_on.
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)  
$ mkdir ./dags ./logs ./config ./plugins
```

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)  
$ mkdir ./dags ./logs ./config ./plugins
```

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)  
$ echo -e "AIRFLOW_UID=$(id -u)" > .env
```

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)  
$ docker compose up airflow-init
```

```
[+] Running 42/4  
✓ redis 6 layers [██████] 0B/0B Pulled  
✓ postgres 13 layers [██████████] 0B/0B Pulled  
- airflow-init 22 layers [██████████] 0B/0B Pulling
```



Apache Airflow : Instalación

usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)

```
$ docker compose up -d
```

[+] Running 7/7

✓ Container airflow_example-redis-1

Healthy

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)
$ docker compose up -d
[+] Running 7/7
✓ Container airflow_example-redis-1
✓ Container airflow_example-postgres-1
✓ Container airflow_init-1
✓ Container airflow-triggerer-1
✓ Container airflow_webserver-1
✓ Container airflow_scheduler-1
✓ Container airflow_worker-1

Containers
```

Containers

Container CPU usage 206.06% / 1000% (10 cores available)

Search Only show running containers

	Name	Image	Status	CPU Usage
airflow_example	redis-1	redis:latest	Running	2.88G
redis-1	postgres-1	postgres:13	Running	0.00%
postgres-1	airflow-init-1	apache/airflow:2.8.3	Running	0.00%
airflow-init-1	airflow-triggerer-1	apache/airflow:2.8.3	Exited	0.00%
airflow-triggerer-1	airflow-webserver-1	apache/airflow:2.8.3	Running	100.84%
airflow-webserver-1	airflow-scheduler-1	apache/airflow:2.8.3	Running	0.23%
airflow-scheduler-1	airflow-worker-1	apache/airflow:2.8.3	Running	3.74%
airflow-worker-1	c4aa30de5160	apache/airflow:2.8.3	Running	98.81%

Sign In

Enter your login and password

Username:

Password:

8080:8080

Sign In

Enter your login and password below:

Username:



Password:



Sign In



Apache Airflow : Instalación



Apache Airflow : Instalación



Pipeline_1

tensorflow == ...
pandas == ...

...

tensorflow == ...
pandas == ...

...



Apache Airflow : Instalación



Pipeline_1

tensorflow == ...
pandas == ...

...

Imagen base (custom_airflow:latest)

tensorflow == ...
pandas == ...

...



Apache Airflow : Instalar dependencias adicionales

x-airflow-common:
&airflow-common

```
# In order to add custom dependencies or upgrade pro  
# Comment the image line, place your Dockerfile in t  
# and uncomment the "build" line below, Then run `do  
image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.8.3}
```

docker-compose.yaml

x-airflow-common:
&airflow-common

```
# In order to add custom dependencies or upgrade pro  
# Comment the image line, place your Dockerfile in t  
# and uncomment the "build" line below, Then run `do  
image: ${AIRFLOW_IMAGE_NAME:-custom_airflow:latest}
```

docker-compose.yaml

requirements.txt

```
scikit-learn==1.3.2  
...  
numpy==1.24.4  
pandas==2.0.3  
joblib==1.3.1  
...
```

Dockerfile

```
FROM apache/airflow:2.8.3  
COPY requirements.txt /requirements.txt  
RUN pip install --user --upgrade pip  
RUN pip install --no-cache-dir --user -r /requirements.txt|
```

docker build . --tag custom_airflow:latest

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)  
$ docker build . --tag custom_airflow:latest  
[+] Building 26.5s (9/9) FINISHED  
=> [internal] load .dockerignore  
=> => transferring context: 2B  
=> [internal] load build definition from Dockerfile  
=> => transferring dockerfile: 203B  
=> [internal] load metadata for docker.io/apache/airflow:2.8.3  
=> [internal] load build context  
=> => transferring context: 94B  
=> CACHED [1/4] FROM docker.io/apache/airflow:2.8.3  
=> [2/4] COPY requirements.txt /requirements.txt  
=> [3/4] RUN pip install --user --upgrade pip  
=> [4/4] RUN pip install --no-cache-dir --user -r /requirement  
=> => exporting to image  
=> => exporting layers  
=> => writing image sha256:ee546ac485d13ef1746bf0d747a2cfca03  
=> => naming to docker.io/library/custom airflow:latest
```

docker compose up -d --build

```
usuario@DESKTOP-4GL9NGT MINGW64 /d/DMC/Airflow_example (main)  
$ docker compose up -d --build  
[+] Running 7/7  
✓ Container airflow_example-postgres-1 Healthy  
✓ Container airflow_example-redis-1 Healthy  
✓ Container airflow_example-airflow-init-1 Exited  
✓ Container airflow_example-airflow-webserver-1 Started  
✓ Container airflow_example-airflow-scheduler-1 Started  
✓ Container airflow_example-airflow-triggerer-1 Started  
✓ Container airflow_example-airflow-worker-1 Started
```

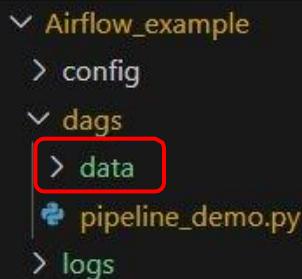


Apache Airflow : Crear pipeline.py

PS D:\DMC> docker ps	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	9042d2f67364	custom_airflow:latest	"/usr/bin/dumb-init ..."	4 hours ago	Up 4 hours (healthy)	8080/tcp	airflow_example-airflow-worker-1
	0e6bae99d88f	custom_airflow:latest	"/usr/bin/dumb-init ..."	4 hours ago	Up 4 hours (healthy)	8080/tcp	airflow_example-airflow-scheduler-1
	4d53a845cf63	custom_airflow:latest	"/usr/bin/dumb-init ..."	4 hours ago	Up 4 hours (healthy)	0.0.0.0:8080->8080/tcp	airflow_example-airflow-webserver-1
	276a4806b8f8	custom_airflow:latest	"/usr/bin/dumb-init ..."	4 hours ago	Up 4 hours (healthy)	8080/tcp	airflow_example-airflow-triggerer-1
	611f3d73b15b	redis:latest	"docker-entrypoint.s..."	24 hours ago	Up 24 hours (healthy)	6379/tcp	airflow_example-redis-1
	7b58ccb5c649	postgres:13	"docker-entrypoint.s..."	24 hours ago	Up 24 hours (healthy)	5432/tcp	airflow_example-postgres-1

```
docker exec -it 4d53a845cf63 bash
```

Este comando permite **abrir una sesión interactiva de Bash** dentro del contenedor 4d53a845cf63, lo que puede ser útil para la depuración o la inspección del estado del contenedor.



```
PS D:\DMC> docker exec -it 4d53a845cf63 bash
default@4d53a845cf63:/opt/airflow$ ls -la
total 120
drwxrwxr-x 1 airflow root 4096 Mar 26 00:12 .
drwxr-xr-x 1 root root 4096 Mar 12 18:27 ..
-rw-r--r-- 1 default root 3 Mar 26 00:12 airflow-webserver.pid
-rw----- 1 default root 97299 Mar 26 00:12 airflow.cfg
drwxrwxrwx 1 root root 4096 Mar 25 03:33 config
drwxrwxrwx 1 default root 4096 Mar 26 04:22 dags
drwxrwxrwx 1 default root 4096 Mar 25 04:12 logs
drwxrwxrwx 1 default root 4096 Mar 25 03:33 plugins
-rw-rw-r-- 1 default root 4761 Mar 26 00:12 webserver_config.py
default@4d53a845cf63:/opt/airflow$ cd dags
default@4d53a845cf63:/opt/airflow/dags$ ls -la
total 8
drwxrwxrwx 1 default root 4096 Mar 26 04:22 .
drwxrwxr-x 1 airflow root 4096 Mar 26 00:12 ..
-rwxrwxrwx 1 root root 2028 Mar 26 04:26 pipeline.py
```



Apache Airflow : Crear pipeline.py

Importaciones y configuraciones iniciales

Se importan las **librerías necesarias para el script**, incluyendo componentes de Airflow, herramientas de fecha y hora, y componentes de scikit-learn necesarios para el flujo de trabajo de aprendizaje automático.

Argumentos predeterminados del DAG

default_args contiene un conjunto de **argumentos predeterminados** que se aplicarán a cada instancia del DAG. Incluye el propietario del DAG, políticas de reintentos, y otros parámetros de configuración como start_date, que indica cuándo comenzar a ejecutar las tareas.

```
import airflow
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from joblib import dump, load

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 3, 26),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

https://airflow.apache.org/docs/apache-airflow/1.10.12/_api/airflow/models/index.html#airflow.models.BaseOperator



Apache Airflow : Crear pipeline.py

Definición del DAG

Aquí se crea una **instancia del objeto DAG** que define cómo se deben organizar y ejecutar las tareas. Los `default_args` se pasan como parámetro, junto con la identificación del DAG, una descripción y un intervalo de programación.

Funciones de Tareas

Cada función define una tarea en el flujo de trabajo

`load_data()`

Esta función se encarga de **cargar un conjunto de datos para su uso posterior** en el pipeline de machine learning. En este caso específico, carga el conocido conjunto de datos Iris, que es un conjunto de datos clásico en el campo del aprendizaje automático utilizado frecuentemente para clasificación y pruebas.

`preprocess_data(ti)`

Esta función realiza el preprocesamiento de los datos cargados, lo que es un paso crucial antes de cualquier entrenamiento de modelos en machine learning.

`X, y = ti.xcom_pull(task_ids='load_data')`: Esta línea **recupera los datos y las etiquetas devueltas por la tarea load_data** utilizando XCom, una característica de Airflow que permite a las tareas intercambiar datos. `ti` es una instancia de `TaskInstance` que proporciona el método `xcom_pull()` para este propósito. Aquí `X` recibirá los datos de las flores Iris y `y` las etiquetas correspondientes.

```
dag = DAG(  
    'ml_workflow',  
    default_args=default_args,  
    description='A simple ML pipeline',  
    schedule_interval=timedelta(days=1),  
)  
  
def load_data():  
    data = load_iris()  
    return data.data.tolist(),  
    data.target.tolist()  
  
def preprocess_data(ti):  
    X, y = ti.xcom_pull(task_ids='load_data')  
    scaler = StandardScaler().fit(X)  
    X_scaled = scaler.transform(X).tolist()  
    return X_scaled, y
```

En el contexto de Apache Airflow, `ti` se refiere a una instancia de `TaskInstance`. En las funciones de Python que utilizas como tareas de Python en Airflow (mediante el `PythonOperator` o similares), `ti` se pasa automáticamente como argumento a la función cuando se ejecuta la tarea.



Apache Airflow : Crear pipeline.py

Funciones de Tareas

Cada función define una tarea en el flujo de trabajo

train_model(ti)

Esta función se ocupa de **entrenar un modelo** de regresión logística utilizando los datos preprocesados.

X_scaled, y = ti.xcom_pull(task_ids='preprocess_data'): Esta línea extrae los datos y etiquetas preprocesados del paso anterior (preprocess_data) utilizando XCom. X_scaled contiene las características de los datos ya estandarizadas, y y contiene las etiquetas correspondientes.

evaluate_model(ti)

Esta función se utiliza para **evaluar el rendimiento del modelo** entrenado en el conjunto de datos preprocesados.

model_path = ti.xcom_pull(task_ids='train_model'): Recupera la ruta del archivo del modelo entrenado que fue guardado por la tarea anterior (train_model) usando XCom.

```
def train_model(ti):
    X_scaled, y = ti.xcom_pull(task_ids='preprocess_data')
    model = LogisticRegression(random_state=42)
    model.fit(X_scaled, y)
    model_path =
        '/opt/airflow/dags/data/logistic_model.joblib'
    dump(model, model_path)
    return model_path

def evaluate_model(ti):
    model_path = ti.xcom_pull(task_ids='train_model')
    model = load(model_path)
    X_scaled, y = ti.xcom_pull(task_ids='preprocess_data')
    predictions = model.predict(X_scaled)
    accuracy = accuracy_score(y, predictions)
    return accuracy
```



Apache Airflow : Crear pipeline.py

Tareas de Airflow

Cada tarea de Airflow se crea utilizando **PythonOperator**, pasando la función Python correspondiente como argumento. Estas tareas se asignan a las funciones definidas anteriormente.

Dependencias de las Tareas

Esto establece el **orden en el que se deben ejecutar las tareas**. Aquí, load_data_task debe completarse antes de preprocess_data_task, que a su vez debe completarse antes de train_model_task, y así sucesivamente.

Using Operators:

<https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/index.html>

```
load_data_task = PythonOperator(  
    task_id='load_data',  
    python_callable=load_data,  
    dag=dag)  
  
preprocess_data_task = PythonOperator(  
    task_id='preprocess_data',  
    python_callable=preprocess_data,  
    dag=dag)  
  
train_model_task = PythonOperator(  
    task_id='train_model',  
    python_callable=train_model,  
    dag=dag)  
  
evaluate_model_task = PythonOperator(  
    task_id='evaluate_model',  
    python_callable=evaluate_model,  
    dag=dag)
```

```
load_data_task >> preprocess_data_task >> train_model_task >> evaluate_model_task
```



GCP: Crear cuenta de GCP

Recibe 300 USD en crédito gratis y usa más de 20 productos gratuitos.

La nueva forma de la nube comienza aquí

Desarrolla aplicaciones con IA generativa, desplégalas rápidamente y analiza datos en cuestión de segundos, todo con una seguridad a la altura de Google.

[Empezar gratis](#) [Contactar con Ventas](#)

Paso 1 de 2 Información de la cuenta

Jose DMC MLE
jose.dmc.mle@gmail.com [CAMBIAR DE CUENTA](#)

País
Perú

Actualizaciones por correo electrónico
 Quiero recibir correos electrónicos periódicos sobre novedades, actualizaciones de productos y ofertas especiales de Google Cloud y los socios de Google Cloud.

Cuando usas esta aplicación, aceptas las [Condiciones del Servicio de Google Cloud Platform](#), la [Prueba gratuita complementaria](#) y cualquier servicio y APIs aplicables.

[ACEPTAR Y CONTINUAR](#)

Paso 2 de 2 Verificación de información de pago

Tu información de pago nos permite reducir los fraudes y abusos. Si usas una tarjeta de crédito o débito, no se te cobrará hasta que actives tu cuenta de forma manual.

Tipo de cuenta
Persona física

Solo las cuentas comerciales pueden tener varios usuarios. No puedes cambiar el tipo de cuenta después de registrarte. En algunos países, esta selección influye en las opciones de impuestos. Si eliges el tipo de cuenta Persona física, aceptas que su uso sea para tu comercio, negocio, oficio o profesión. [Más información](#)

Forma de pago

Número de tarjeta
MM / AA CVC
Se requiere el número de tarjeta

Nombre del titular
El nombre del titular de la tarjeta es obligatorio.
Línea 1 de la dirección
El campo Línea 1 de la dirección es obligatorio.
Línea 2 de la dirección
Distrito
El campo Distrito es obligatorio.
Código postal
Provincia/ región
Municipalidad Metropolitana de Lima

Bienvenido a Google!

My First Project [Buscar \(/\) recursos, documentos, productos y más](#) [Buscar](#)

Te damos la bienvenida

Estás trabajando en My First Project

Número de proyecto: 694915007051 [ID del proyecto: swift-library-412504](#)

[Panel](#) [Recomendaciones](#)

[Crea una VM](#) [Ejecuta una consulta en BigQuery](#) [Crea un clúster de GKE](#) [Crea un bucket de almacenamiento](#)



Cloud Composer: Conceptos

Cloud Composer es un **servicio de orquestación de flujos de trabajo totalmente gestionado**, basado en Apache Airflow, ofrecido por Google Cloud. Permite a los usuarios **componer, programar, y monitorizar flujos de trabajo complejos** que abarcan desde el procesamiento de datos y la analítica hasta la integración de datos y la inteligencia artificial. Su integración nativa con servicios de Google Cloud y su capacidad para orquestar servicios de otras plataformas lo hacen versátil para diversas aplicaciones de automatización y ETL (Extract, Transform, Load).

Beneficios

Integración con Servicios: Facilita la automatización de flujos de trabajo al integrarse de manera fluida con servicios de Google Cloud y sistemas externos, apoyando entornos de nube híbrida o multi-nube.

Gestión Centralizada: Proporciona una interfaz única para la programación, monitorización y gestión de flujos de trabajo, simplificando la supervisión y el manejo de procesos complejos.

Flexibilidad y Escalabilidad: Ajusta automáticamente los recursos según las necesidades de los flujos de trabajo, asegurando eficiencia y capacidad de adaptación a diferentes cargas de trabajo.

Facilidad de Uso: Permite a los usuarios definir flujos de trabajo complejos de forma intuitiva con Python, aprovechando la estructura de DAGs (Directed Acyclic Graphs) para simplificar la creación y mantenimiento de flujos de trabajo.

Fiabilidad y Seguridad: Como servicio gestionado, ofrece alta disponibilidad y robustas medidas de seguridad, liberando a los usuarios de la gestión de infraestructura y mantenimiento.

Consistencia y Reproducibilidad: Asegura la ejecución consistente de tareas y facilita la reproducción de flujos de trabajo, mejorando la calidad y la fiabilidad de los procesos automatizados.

Costo-Efectividad: Reduce la necesidad de invertir en infraestructura y mantenimiento dedicados a la ejecución de flujos de trabajo, optimizando los costos operativos y de recursos.



Cloud Composer: Conceptos

En Cloud Composer, un "**Environment**" (Entorno) representa una **instancia aislada** y completamente gestionada de Apache Airflow en la nube. Es el **espacio de trabajo central** donde se **configuran, ejecutan y monitorean** los flujos de trabajo de Airflow. Cada entorno de Cloud Composer se compone de **recursos y servicios gestionados** que están optimizados para ejecutar, programar y orquestar tus flujos de trabajo de manera eficiente.

Componentes Principales de un Entorno

Servidor web de Airflow: Una interfaz de usuario web para visualizar y gestionar tus flujos de trabajo de Airflow.

Scheduler de Airflow: El componente que se encarga de programar las tareas de los flujos de trabajo basándose en sus dependencias y en el cronograma definido.

Base de datos de metadatos: Una base de datos (usualmente PostgreSQL) que Airflow utiliza para almacenar el estado y la información de los flujos de trabajo, tareas y ejecuciones.

Worker nodes: Nodos que ejecutan las tareas definidas en tus DAGs (Directed Acyclic Graphs). Estos nodos son gestionados por Kubernetes en el clúster de GKE.

Cloud Storage: Cada entorno utiliza un bucket de Cloud Storage como almacenamiento persistente para logs, DAGs y otros archivos necesarios para la ejecución de flujos de trabajo.



Cloud Composer: Permisos

Screenshot of the Google Cloud IAM & Admin interface showing permissions for the project "My First Project".

The sidebar shows the following navigation:

- IAM & Admin
- IAM (selected)
- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer (NEW)

The main content area displays the following information:

Permissions for project "My First Project"

These permissions affect this project and all of its resources. [Learn more](#)

2 service accounts with highly privileged roles Owner / Editor have excess permissions.
Improve security by applying recommendations to these accounts.
[Learn more about recommendations.](#)

[Tell me more](#) [VIEW RECOMMENDATIONS IN TABLE](#)

[Include Google-provided role grants](#) [?](#)

[VIEW BY PRINCIPALS](#) [VIEW BY ROLES](#)

[GRANT ACCESS](#) [REMOVE ACCESS](#)

Filter [Enter property name or value](#)

Type	Principal	Name	Role	Security insights	Conditions
<input type="checkbox"/>	service-608375853532@cloudcomposer-accounts.iam.gserviceaccount.com	Cloud Composer Service Agent	Cloud Composer API Service Agent	?	
<input type="checkbox"/>			Cloud Composer v2 API Service Agent Extension		



Cloud Composer: Permisos

Edit access to "My First Project"

Principal ?
service-608375853532@cloudcomposer-accounts.iam.gserviceaccount.com

Project
My First Project

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

Role
Cloud Composer API Service Age... ▾

IAM condition (optional) ?
[+ ADD IAM CONDITION](#)

Cloud Composer API service agent can manage environments.

+ ADD ANOTHER ROLE

SAVE **TEST CHANGES** ⓘ **CANCEL**

Filter composer | X

Cloud Composer API Service Agent
Cloud Composer API service agent can manage environments.

Cloud Composer v2 API Service Agent Extension
Cloud Composer v2 API Service Agent Extension is a supplementary role required to manage Composer v2 environments.

Composer Administrator
Full control of Composer resources.

Composer Shared VPC Agent
Role that should be assigned to Composer Agent service account in Shared VPC host project.

MANAGE ROLES

Edit access to "My First Project"

Principal ?
service-608375853532@cloudcomposer-accounts.iam.gserviceaccount.com

Project
My First Project

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

Role
Cloud Composer API Service Age... ▾

IAM condition (optional) ?
[+ ADD IAM CONDITION](#)

Cloud Composer API service agent can manage environments.

Role
Cloud Composer v2 API Service ... ▾

IAM condition (optional) ?
[+ ADD IAM CONDITION](#)

Cloud Composer v2 API Service Agent Extension is a supplementary role required to manage Composer v2 environments.

+ ADD ANOTHER ROLE

SAVE **TEST CHANGES** ⓘ **CANCEL**



Cloud Composer: Create environment

Cloud Composer API

Google Enterprise API

Manages Apache Airflow environments on Google Cloud Platform.

[ENABLE](#) [TRY THIS API](#)

Cloud Composer Environments

With Composer, you can easily create and manage Airflow environments. Get started by creating your first environment.

[CREATE ENVIRONMENT](#) [LEARN MORE](#)

Composer 2 Autoscaling, Airflow 2	
Composer 1 (end of support) No autoscaling, supports Airflow 1 and Airflow 2	
Read about Cloud Composer versions	

Environments are self-contained Airflow deployments based on Google Kubernetes Engine, and they work with other Google Cloud services using connectors built into Airflow. You can create one or more environments in a single Google Cloud project. You can create Cloud Composer environments in any supported region.

Name *

Location * [?](#)

Image version * [?](#)

Environment resources

[SMALL](#) [MEDIUM](#) [LARGE](#) [CUSTOM](#)

Workloads configuration

Scheduler [?](#)
1 scheduler with 0.5 vCPU, 2 GB memory, 1 GB storage

Triggerer [?](#)
1 triggerer with 0.5 vCPU, 0.5 GB memory, 1 GB storage

Web server [?](#)
0.5 vCPU, 2 GB memory, 1 GB storage

Worker [?](#)
Autoscaling between 1 and 3 workers, with 0.5 vCPU, 2 GB memory, 1 GB storage each

Core infrastructure

Environment size [?](#)
Small

Service account —
608375853532-compute@developer.gserviceaccount.com [?](#)

1 Composer requires additional permissions to use the selected service account. Please grant the permissions below or use another service account.

Grant required permissions to Cloud Composer service account

Cloud Composer relies on [Workload Identity](#) as Google API authentication mechanism for Airflow.

In order to support Workload Identity, Cloud Composer creates additional IAM role bindings which requires the [Cloud Composer v2 API Service Agent Extension](#) role.

Grant the Cloud Composer v2 API Service Agent Extension role to the service 608375853532@cloudcomposer-accounts.iam.gserviceaccount.com service account. Service account 608375853532-compute@developer.gserviceaccount.com will be used as a resource.

[GRANT](#)



Cloud Composer: Instalar dependencias

Composer [← Environment details](#) [OPEN AIRFLOW UI](#) [OPEN DAGS FOLDER](#) [SAVE SNAPSHOT](#) [LOAD SNAPSHOT](#) [REFRESH](#) [DELETE](#)

custom-dmc This environment is running

MONITORING LOGS DAGS ENVIRONMENT CONFIGURATION AIRFLOW CONFIGURATION OVERRIDES ENVIRONMENT VARIABLES LABELS **PYPI PACKAGES**

EDIT

Required libraries from the Python Package Index (PyPI)
No PyPI packages added

Specify any required libraries from the Python Package Index (PyPI). If a library cannot be found, it will be removed.

PyPI packages

Package name 1 * scikit-learn	Extras and version 1 ==1.3.2
Package name 2 * numpy	Extras and version 2 ==1.24.4
Package name 3 * pandas	Extras and version 3 ==2.0.3
Package name 4 * joblib	Extras and version 4 ==1.3.1

+ ADD PACKAGE

SAVE CANCEL



Cloud Storage: Create bucket

The screenshot illustrates the process of creating a new Cloud Storage bucket in the Google Cloud Platform.

Step 1: Main Cloud Storage Dashboard

The top navigation bar shows "Google Cloud" and "My First Project". Below it, the "Buckets" section is selected, displaying a "CREATE" button which is highlighted with a red box. A "REFRESH" button is also present.

Step 2: Create a Bucket Dialog

A modal dialog titled "Create a bucket" is open. It contains a list item "Name your bucket" and a text input field containing "bucket-dmc-jose". A tip below the input says "Tip: Don't include any sensitive information". There is also a "LABELS (OPTIONAL)" section and a "CONTINUE" button at the bottom.

Step 3: Bucket Details View

The "Bucket details" view for "bucket-dmc-jose" is shown. Key details include:

- Location:** us (multiple regions in United States)
- Storage class:** Standard
- Public access:** Not public
- Protection:** None

The navigation bar at the top of this view includes tabs for OBJECTS, CONFIGURATION, PERMISSIONS, PROTECTION, LIFECYCLE, OBSERVABILITY, INVENTORY REPORTS, and OPERATIONS. Below the tabs, there are buttons for UPLOAD FILES, UPLOAD FOLDER, CREATE FOLDER, TRANSFER DATA, MANAGE HOLDS, EDIT RETENTION, DOWNLOAD, and DELETE.

At the bottom, there is a search/filter bar with the placeholder "Filter by name prefix only" and a "Filter" button, along with columns for Name, Size, Type, Created, Storage class, Last modified, Public access, Version history, Encryption, and Object retention.

No rows are displayed in the main content area.



Cloud Composer: Añadir variables de entorno

custom-dmc This environment is running.

MONITORING LOGS DAGS ENVIRONMENT CONFIGURATION AIRFLOW CONFIGURATION OVERRIDES **ENVIRONMENT VARIABLES** LABELS PYPI PACKAGES

EDIT

Optional additional environment variables to provide to the Airflow scheduler, worker, and webserver processes.

No environment variables added

Optional additional environment variables to provide to the Airflow scheduler, worker, and webserver processes.

Environment variables

Key 1 *	BUCKET_NAME	Value 1	bucket-dmc-jose
---------	-------------	---------	-----------------

+ ADD ENVIRONMENT VARIABLE

SAVE CANCEL



Cloud Composer: Agregar DAG

Cambios: pipeline_demo.py

```
def train_model(ti):
    X_scaled, y = ti.xcom_pull(task_ids='preprocess_data')
    model = LogisticRegression(random_state=42)
    model.fit(X_scaled, y)
    model_path = '/opt/airflow/dags/data/logistic_model.joblib'
    dump(model, model_path)
    return model_path
```



```
def train_model(ti):
    X_scaled, y = ti.xcom_pull(task_ids='preprocess_data')
    model = LogisticRegression(random_state=42)
    model.fit(X_scaled, y)

    #####
    ##### Using Cloud Composer #####
    #####

    # Use Airflow's GCSHook to upload the model
    gcs_hook = GCSHook()
    bucket_name = os.environ.get('BUCKET_NAME')
    object_name = 'airflow_demo_dmc/logistic_model.joblib'

    with tempfile.NamedTemporaryFile(delete=False) as tmp:
        # Save the model to a temporary file
        dump(model, tmp.name)
        # Upload the temporary file to GCS
        gcs_hook.upload(bucket_name, object_name, tmp.name)

    return f'gs://{bucket_name}/{object_name}'
```

<https://airflow.apache.org/docs/apache-airflow-providers-google/stable/index.html>



Cloud Composer: Agregar DAG

Cambios: pipeline_demo.py

```
def evaluate_model(ti):
    model_path = ti.xcom_pull(task_ids='train_model')
    model = load(model_path)
    X_scaled, y = ti.xcom_pull(task_ids='preprocess_data')
    predictions = model.predict(X_scaled)
    accuracy = accuracy_score(y, predictions)
    return accuracy
```



```
def evaluate_model(ti):
    model_path = ti.xcom_pull(task_ids='train_model')

    ##### Using Cloud Composer #####
    # Extract bucket name and object name from the model_path
    _, _, bucket_name, object_name = model_path.split('/', 3)

    gcs_hook = GCSHook()

    with tempfile.NamedTemporaryFile(delete=False) as tmp:
        # Download the model from GCS to a temporary file
        gcs_hook.download(bucket_name, object_name, tmp.name)
        # Load the model from the temporary file
        model = load(tmp.name)

    #####
    X_scaled, y = ti.xcom_pull(task_ids='preprocess_data')
    predictions = model.predict(X_scaled)
    accuracy = accuracy_score(y, predictions)
    return accuracy
```



Cloud Composer: Agregar DAG



custom-dmc This environment is running

MONITORING

LOGS

DAGS

ENVIRONMENT CONFIGURATION

AIRFLOW CONFIGURATION OVERRIDES

Name

custom-dmc

Location

us-central1

Service account

60837583532-compute@developer.gserviceaccount.com

Image version

composer-2.6.6-airflow-2.6.3 UPGRADE

ⓘ New version available. [Learn more](#) ↗

Python version

3

DAGs folder

<gs://us-central1-custom-dmc-1b72fc20-bucket/dags>

Airflow web UI

<https://dddb5bc85e294b2f99ea33df89d6cefd-dot-us-central1.cloudfront.net/>

Logging

[view logs](#)

Maintenance windows

4 hours starting 11:00 PM UTC-5 on Thursday, Friday, and Saturday

us-central1-custom-dmc-1b72fc20-bucket

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Subject to object ACLs	None

OBJECTS CONFIGURATION PERMISSIONS PROTECTION LIFECYCLE OB

Buckets > us-central1-custom-dmc-1b72fc20-bucket > dags ↗

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA ▾ MANAGE HOLDS

Filter by name prefix only ▾ Filter Filter objects and folders

<input type="checkbox"/>	Name	Size	Type	Created
<input type="checkbox"/>	airflow_monitoring.py	809 B	text/x-python	Mar 31, 2024, 2:08:31 PM



Cloud Composer: Agregar DAG

The diagram illustrates the workflow for adding a Data Access Generator (DAG) to a Cloud Composer environment. It consists of three main components:

- Google Cloud Storage Bucket:** A screenshot of the Google Cloud Storage console showing a bucket named "us-central1-custom-dmc-1b72fc20-bucket". Inside the bucket, there are two files: "airflow_monitoring.py" and "pipeline_demo.py". A red arrow points from the "pipeline_demo.py" file towards the Cloud Composer interface.
- Cloud Composer Environment:** A screenshot of the Cloud Composer environment named "custom-dmc". The "DAGS" tab is selected, showing a list of active DAGs: "airflow_monitoring" and "ml_workflow_demo". The "airflow_monitoring" DAG is currently running, as indicated by a green checkmark icon.
- Cloud Composer DAG Details:** A detailed view of the "airflow_monitoring" DAG. It shows the DAG ID is "airflow_monitoring", the state is "Active", the description is "liveness monitoring dag", the schedule interval is "/10 * * * *", and the last completed run was 8 minutes ago.



Cloud Composer: Agregar DAG

Composer [← Environment details](#) [OPEN AIRFLOW UI](#)

 custom-dmc This environment is running

Airflow DAGs Datasets Browse Admin Docs Composer

custom-dmc

All 2 Active 2 Paused 0

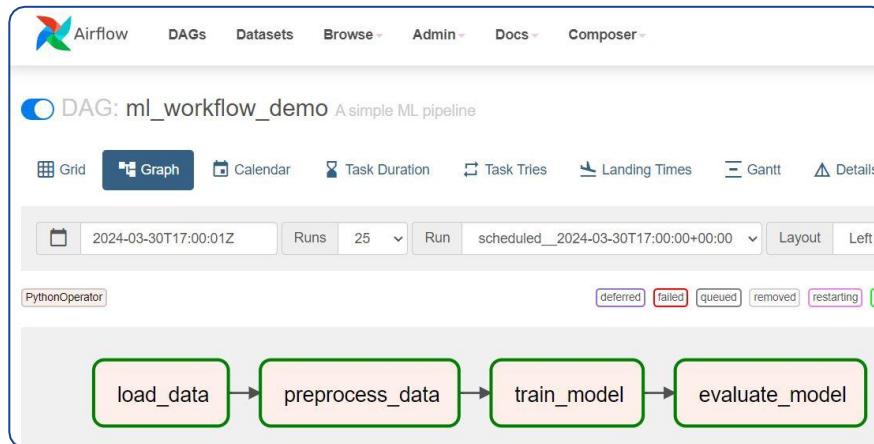
Filter DAGs by tag Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
airflow_monitoring	airflow	6	*/*/*/*/*	2024-03-31, 19:40:00	2024-03-31, 19:50:00	1
ml_workflow_demo	airflow	5	0 17 * * *	2024-03-30, 17:00:00	2024-03-31, 17:00:00	3 2 15

« ‹ 1 › »



Cloud Composer: Agregar DAG



bucket-dmc-jose

Location us (multiple regions in United States) Storage class Standard

OBJECTS CONFIGURATION PERMISSIONS

Buckets > bucket-dmc-jose

UPLOAD FILES UPLOAD FOLDER

Filter by name prefix only

Name airflow_demo_dmc/

bucket-dmc-jose

Location us (multiple regions in United States) Storage class Standard

OBJECTS CONFIGURATION PERMISSIONS

Buckets > bucket-dmc-jose > airflow_demo_dmc

UPLOAD FILES UPLOAD FOLDER CREATE FOLD

Filter by name prefix only

Name logistic_model.joblib



AWS: Crear cuenta de AWS

Nivel gratuito de AWS

Adquiera experiencia práctica y gratuita con los productos y servicios de AWS

Más información sobre el nivel gratuito de AWS

[Crear una cuenta gratuita](#)

aws

Registrarse en AWS

Explore los productos de la capa gratuita con una cuenta de AWS nueva.

Para obtener más información, visite aws.amazon.com/free.



Dirección de correo electrónico del usuario raíz
Se utiliza para la recuperación de cuentas y algunas funciones administrativas

Nombre de la cuenta de AWS
Elija un nombre para la cuenta. Podrá cambiarlo en la configuración de la cuenta después de registrarse.

[Verificar la dirección de correo electrónico](#)

O

[Iniciar sesión en una cuenta de AWS existente](#)



AWS: Managed Workflows Apache Airflow

Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

US East (Ohio) us-east-2

Bucket name Info

dmc-esp-4

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

ACLs disabled (recommended)

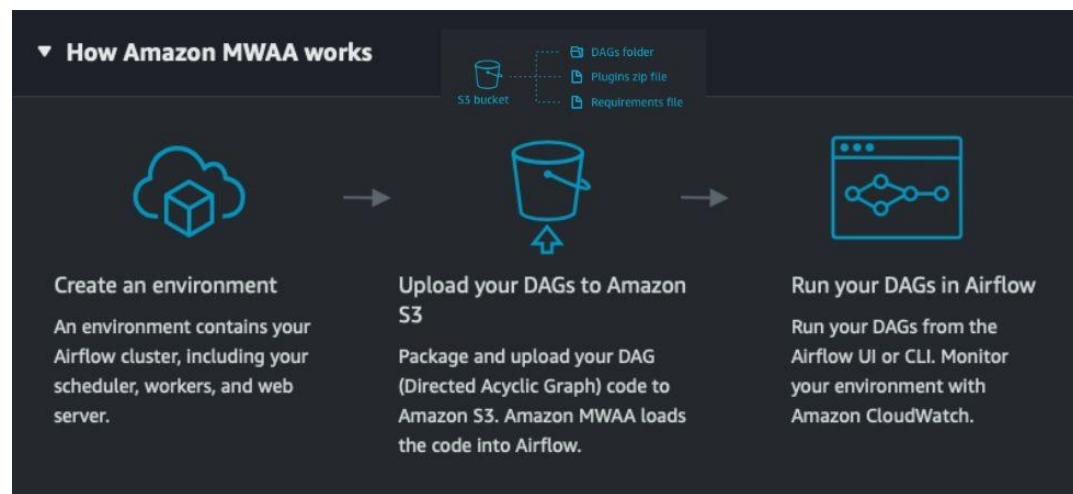
All objects in this bucket are owned by this account.
Access to this bucket and its objects is specified using only policies.

ACLs enabled

Objects in this bucket can be owned by other AWS accounts.
Access to this bucket and its objects can be specified using ACLs.

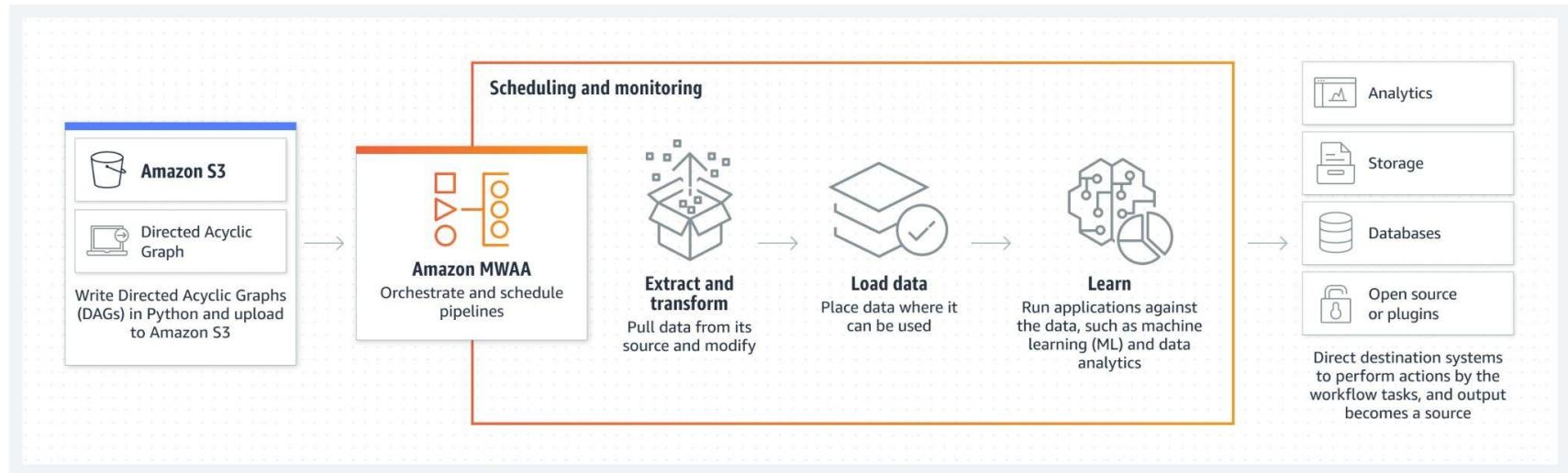
Object Ownership

Bucket owner enforced





AWS: Managed Workflows Apache Airflow





AWS: Managed Workflows Apache Airflow

Subnet settings
Specify the CIDR blocks and Availability Zone for the subnet.

Subnet 1 of 1

Subnet name
Create a tag with a key of 'Name' and a value that you specify.
The name can be up to 256 characters long.

Availability Zone [Info](#)
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

IPv4 VPC CIDR block [Info](#)
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.

IPv4 subnet CIDR block
 4096 IPs
< > ^ ▾

Tags - optional

Key	Value - optional
<input type="text" value="Name"/> <input type="button" value="X"/>	<input type="text" value="private-subnet-aa-01"/> <input type="button" value="X"/>

You can add 49 more tags.

Subnet settings
Specify the CIDR blocks and Availability Zone for the subnet.

Subnet 1 of 1

Subnet name
Create a tag with a key of 'Name' and a value that you specify.
The name can be up to 256 characters long.

Availability Zone [Info](#)
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

IPv4 VPC CIDR block [Info](#)
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.

IPv4 subnet CIDR block
 4096 IPs
< > ^ ▾

Tags - optional

Key	Value - optional
<input type="text" value="Name"/> <input type="button" value="X"/>	<input type="text" value="private-subnet-aa-02"/> <input type="button" value="X"/>

You can add 49 more tags.



AWS: Managed Workflows Apache Airflow

Application Integration

Amazon Managed Workflows for Apache Airflow (MWAA)

Run Apache Airflow without provisioning or managing servers

Environment details Info

Name
 Use only letters, numbers, dashes, or underscores. Max 80 characters.

Airflow version

Weekly maintenance window start (UTC)

Create an Airflow environment

Launch a complete, auto-scaling Airflow environment in minutes.

Create environment

Configure advanced settings

Networking

Virtual private cloud (VPC) Info
Defines the networking infrastructure setup of your Airflow environment. An environment needs 2 private subnets in different availability zones. To create a new VPC with private subnets, choose Create MWAA VPC. [Learn more](#)

vpc-027a3c688e2156669 Default

Subnet 1
Private subnet for the first availability zone. Each environment occupies 2 availability zones.
subnet-0904cc2efcae0cd4 us-west-1c Private

Subnet 2
Private subnet for the second availability zone. Each environment occupies 2 availability zones.
subnet-010da3c9d223e3956 us-west-1a Private



AWS: Managed Workflows Apache Airflow

Web server access | [Info](#)

Private network (No internet access)

Additional setup required. Your Airflow UI can only be accessed by secure login behind your VPC. Choose this option if your Airflow UI is only accessed within a corporate network and you do not require a public repository for webserver requirements installation. IAM must be used to handle user authentication.

Public network (Internet accessible)

Your Airflow UI can be accessed by secure login over the Internet. Choose this option if your Airflow UI is accessed outside of a corporate network. IAM must be used to handle user authentication.

Security group(s) | [Info](#)

A VPC security group is required to allow traffic between your environment and your web server.

Create new security group

Allow MWAA to create a VPC security group with inbound and outbound rules based on your selection for web server access.

Existing security group(s)

You can choose 1 or more existing security groups to configure the inbound and outbound rules for your environment.

Choose security group ▾

sg-07fc8aad0e240ed87
default VPC security group
default

Max 5 security groups

Endpoint management | [Info](#)

Service managed endpoints (Recommended)

Amazon MWAA will create the VPC endpoints required to connect your VPC to the single-tenant VPC hosting the Airflow UI and meta database.

Customer managed endpoints

Amazon MWAA will create resources in the single-tenant VPC and then wait for you to create the required endpoint(s) to your VPC (required when using shared VPCs)

Environment class | [Info](#)

Each Amazon MWAA environment includes 2 schedulers, 2 web servers, and 1 worker. Workers and web servers auto-scale up and down according to system load. You can monitor the load on your environment and modify its class at any time.

DAG capacity*	Scheduler CPU	Worker CPU	Web server CPU
<input checked="" type="radio"/> mw1.small	Up to 50	1 vCPU	1 vCPU
<input type="radio"/> mw1.medium	Up to 250	2 vCPU	1 vCPU
<input type="radio"/> mw1.large	Up to 1000	4 vCPU	2 vCPU
<input type="radio"/> mw1.xlarge	Up to 2000	8 vCPU	4 vCPU
<input type="radio"/> mw1.2xlarge	Up to 4000	16 vCPU	8 vCPU

*under typical usage

Maximum worker count

The maximum number of workers your environment is permitted to scale up to.

4

Must be between 1 and 25.

Minimum worker count

The minimum number of workers always present in your environment.

1

Must be less than or equal to maximum workers. Minimum 1 worker.

Maximum web server count

The maximum number of web servers your environment is permitted to scale up to.

2

Must be between 2 and 5.

Minimum web server count

The minimum number of web servers always present in your environment.

2

Must be less than or equal to maximum web servers. Minimum 2 web servers.

Scheduler count

The number of schedulers your environment will use.

2

Must be between 2 and 5.



AWS: Managed Workflows Apache Airflow

Permissions [Info](#)

Execution role
The IAM role used by your environment to access your DAG code, write logs, and perform other actions.

[Create a new role](#) [C](#)

Role name
 [C](#)

Use alphanumeric and '+,-,@,_' characters. Maximum 64 characters.

Info Amazon MWAA will create and assume the execution role in IAM named **AmazonMWAA-MyAirflowEnvironment-jemg-AV1vqQ** on your behalf. This role is configured with permission to retrieve code from your Amazon S3 bucket, use your KMS key, and send data to Amazon CloudWatch. You must add permissions to your execution role if your Airflow DAGs require access to any other AWS services.[Info](#)

[Cancel](#) [Previous](#) [N](#)

Amazon MWAA > Environments > MyAirflowEnvironment-jemg

MyAirflowEnvironment-jemg

[Edit](#) [Delete](#) [Open Airflow UI](#)

Details

Status: Available

ARN: arn:aws:airflow:us-west-1:533267339745:environment/MyAirflowEnvironment-jemg

Airflow UI: <https://c91t081-8c5c-4bd3-842d-bbd7c6eb293b.r6.us-west-1.airflow.amazonaws.com>

Weekly maintenance window start (UTC): Tuesday 10:30

Last update

Status: SUCCESS

Created at: 2024-10-07T19:57:50.000Z



AWS: Managed Workflows Apache Airflow

The screenshot shows the AWS IAM Roles page. A new role named "AmazonMWAA-MyAirflowEnvironment-jeng-AV1vqQ" has been created. The "Permissions" tab is selected, showing one managed policy attached: "MWAA-Execution-Policy-42bcd018-9389...". The "Add permissions" button is highlighted.

Permissions policies (1) Info
You can attach up to 10 managed policies.

Other permissions policies (972)
Filter by Type: All types
Policy name: AmazonDMSRedshiftS3Role, AmazonSFFullAccess

AmazonSFFullAccess
Provides full access to all buckets via the AWS Management Console.

```
1+ {
2+     "Version": "2012-10-17",
3+     "Statement": [
4+         {
5+             "Effect": "Allow",
6+             "Action": [
7+                 "s3:*",
8+                 "s3-object-lambda:*"
9+             ],
10+            "Resource": "*"
11+        }
12+    ]
13+ }
```