Apache Airflow

Introducción a la Orquestación de Flujos de Datos

Flavio Cesar Sandoval

Data Enginner Manager

- SQL Developer
- Data Analyst
- Marketing Science Analyst
- Data Engineer 3 años

Profesor en los Bootcamp dedicados a la Ingeniería y Ciencia de Datos en Código Facilito

Redes Sociales

- LinkedIn: flaviocesarsandoval
- Twitter: DSandovalFlavio
- GitHub: DSandovalFlavio

Conocimientos necesarios para este curso

- Python
- Bash Basico
- SQL Basico

¿Que vas a aprender?

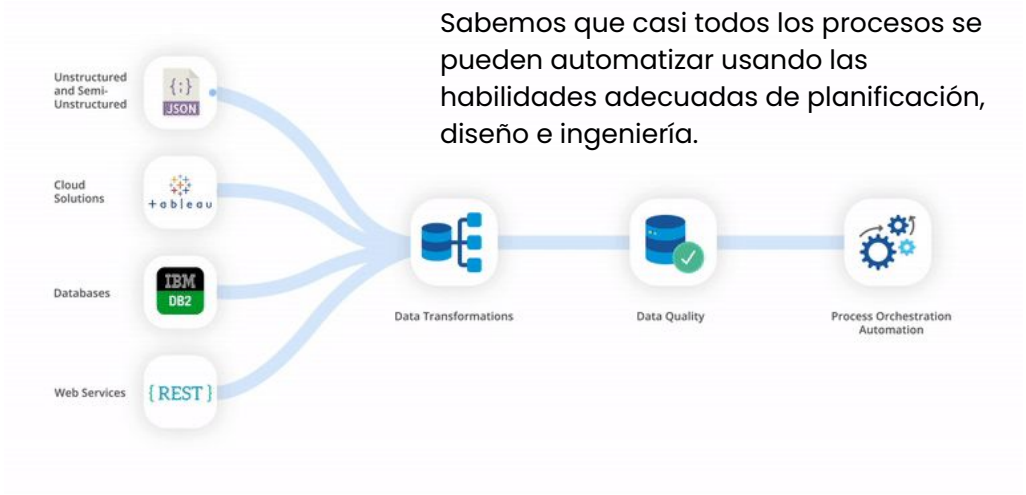
- Que es la orquestación de data pipelines o data workflows
- Por qué es un punto crucial en la ingeniería de datos
- Que es un DAG
- Que es Apache Airflow
- Cuáles son los componentes de Airflow
- Que es un Operador
- Que son los providers en Airflow
- Como orquestar procesos en GCP utilizando Airflow
- Que es un sensor
- Que es un branchOperator

Los Desafíos Iniciales en la Ingeniería de Datos

Datos, fuentes, arquitecturas, procesos y más datos

- Limpieza de datos
- Disponibilidad de los datos
- Estandarización de procesos
- Silos de información
- Cuellos de botella en el procesamiento
- Calidad de datos
- Integración de datos
- Integración de procesos

El 80% del esfuerzo de un equipo de data se enfoca en recopilar, limpiar y transformar los datos.

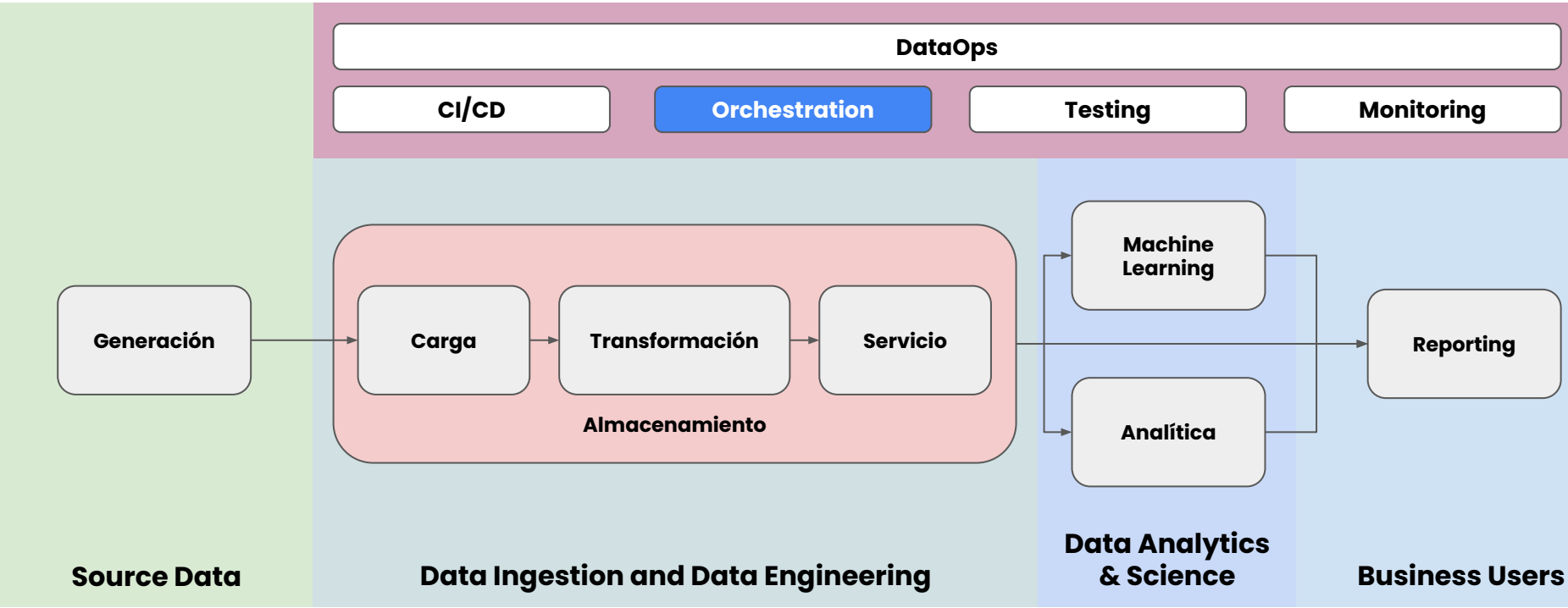


Sabemos que casi todos los procesos se pueden automatizar usando las habilidades adecuadas de planificación, diseño e ingeniería.

Datos y más datos = Procesos y más procesos

La unificación de procesos es una de las principales tareas que deben de priorizar los equipos de ingeniería de datos

¿Por qué necesitamos unificar y orquestar los sistemas de datos?



Flujo de Vida de la Ingenieria de Datos

Orquestación de Flujos de Datos

¿Qué es un orquestador de flujos de datos?

Características de un Orquestador de Flujos de Datos

- Automatización
- Planificación y Programación
- Gestión de Dependencias
- Monitorización y Registro
- Tolerancia a Fallos
- Escalabilidad
- Integración con Ecosistemas Tecnológicos

Data workflow

- Abstracción
- Enfoque Lógico
- Visión de Alto Nivel

Data pipeline

- Implementación Técnica
- Enfoque Técnico
- Detalles Técnicos

Introducción a Apache Airflow

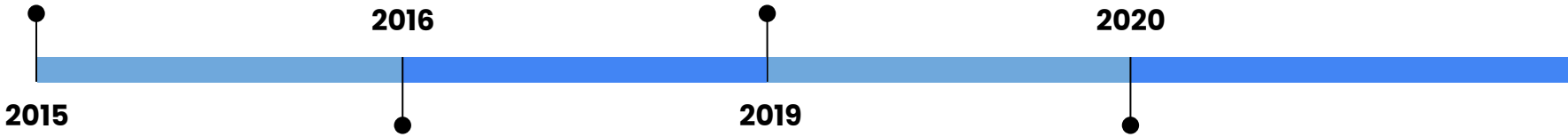
como herramienta de orquestación de flujos de datos.

Orquestación de flujos de datos con Airflow

¿Que es Apache Airflow?

Apache Airflow es un framework diseñado para la creación y monitoreo de data pipelines. Se destaca por su flexibilidad y su capacidad de programación en Python.

Historia de Apache Airflow



AirBnB

Maxime Beauchemin creó una herramienta para orquestar procesos de datos en AirBnB llamada Airflow

Fundación Apache

Airflow se incorpora a la incubadora oficial de la Fundación Apache, marcando un hito significativo en su desarrollo y reconocimiento en la comunidad de código abierto.

Máxima Categoría

Airflow alcanza el estatus de proyecto de "*máxima categoría*" dentro de la Fundación Apache.

Versión 2.0

La versión 2.0 se lanza al público, introduciendo nuevas funciones y mejoras significativas.

¿Para que sirve?

Coordinación de Procesos de Datos: Airflow actúa como el conductor en el centro de sus procesos de datos. Coordina y programa la ejecución de tareas en sistemas distribuidos.

¿Para que no sirve?

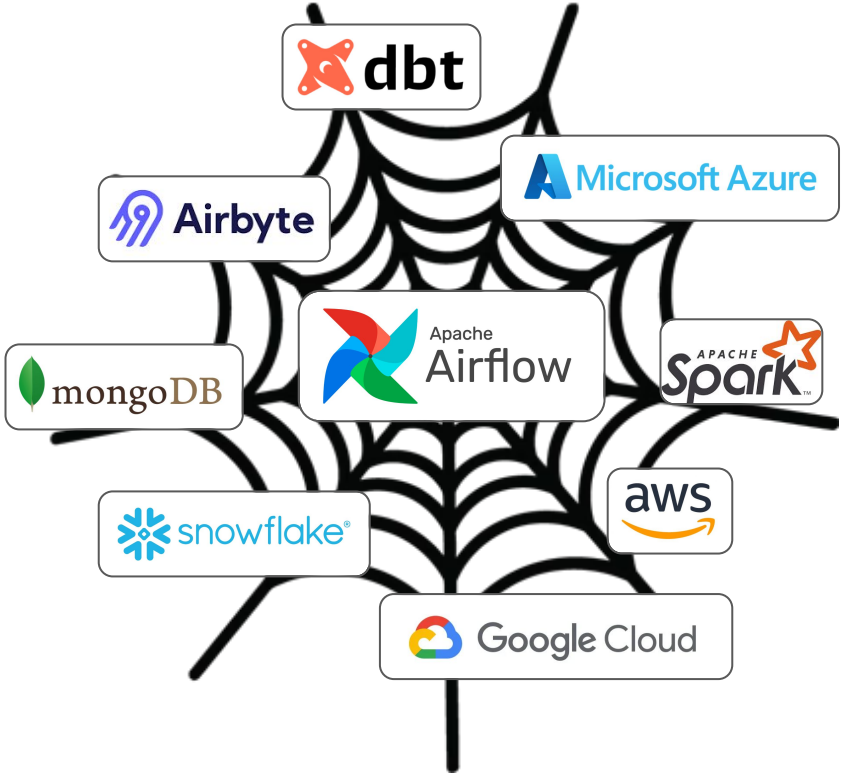
No es una Herramienta de Procesamiento: A diferencia de las herramientas de procesamiento de datos, Airflow no realiza la manipulación directa de los datos. En cambio, se encarga de orquestar los diversos componentes que procesan los datos en las canalizaciones.

Rol de Apache Airflow en la Orquestación de Canalizaciones de Datos

Características:

Orquestación de Componentes: Airflow coordina y programa componentes que son responsables del procesamiento de datos en las canalizaciones. Estos componentes pueden incluir extracción, transformación, carga (ETL), almacenamiento, y más.

Facilita la Automatización: Al estar en el centro de la orquestación, Airflow permite la automatización de flujos de trabajo complejos, lo que ahorra tiempo y reduce la probabilidad de errores en la gestión de datos.



Apache Airflow NO es una herramienta de ETL

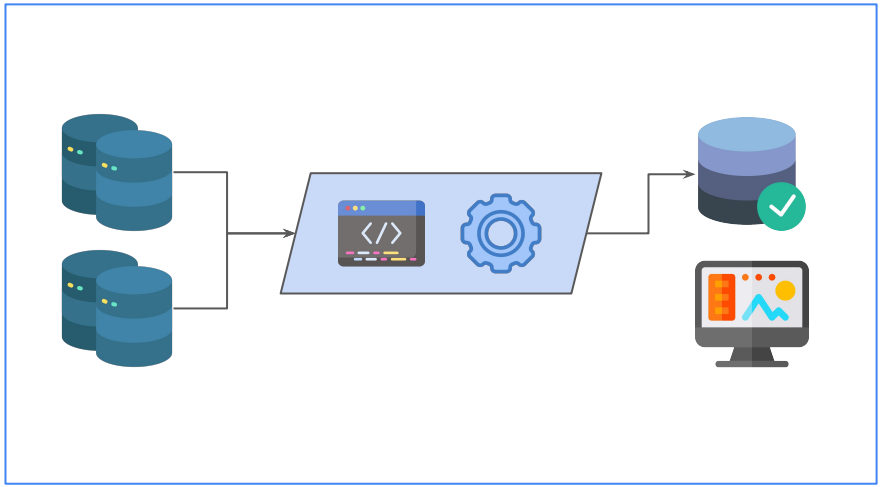
Data pipelines como graficos

Los data pipelines consisten en varias tareas o acciones que deben ejecutarse para lograr un resultado esperado

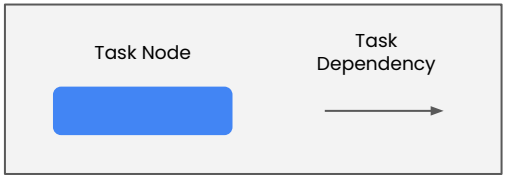
Ejemplo: Supongamos que queremos crear un pequeño dashboard de ventas que nos diga como se ven las ventas de la semana.

Pasos a seguir:

- Obtener los datos de las ventas desde sus fuentes
- Limpiar y/o transformar los datos para que se ajusten a nuestro propósito aplicando reglas de negocio
- Enviar los datos transformados a una fuente donde la pueda leer el dashboard



Graficos Dirigidos



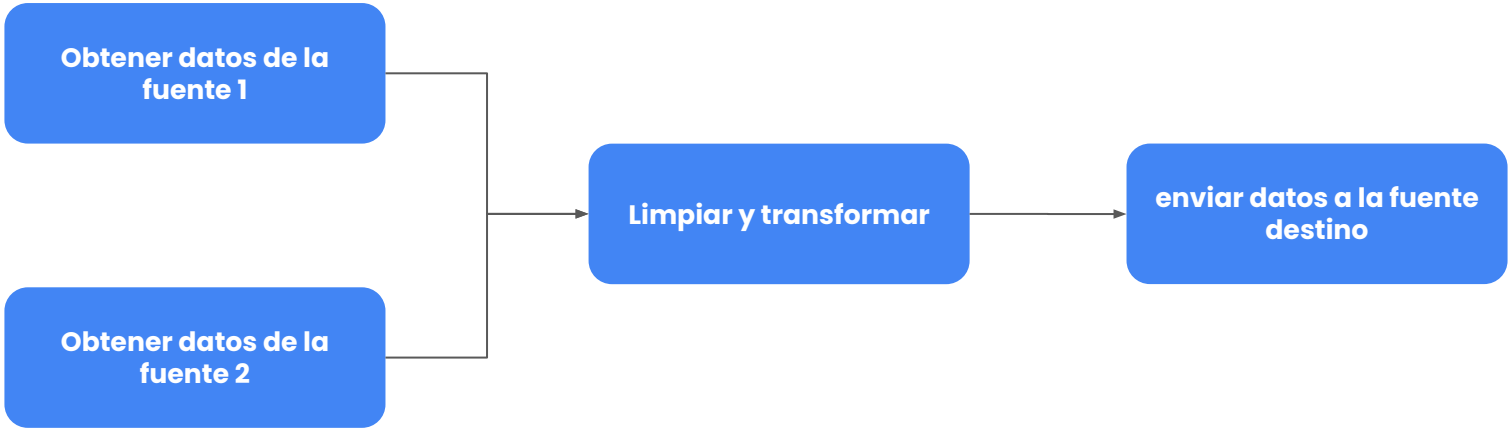


Gráfico Acíclico Dirigido (DAG) de tareas

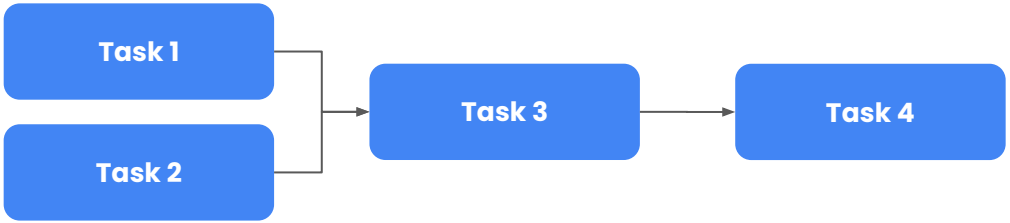
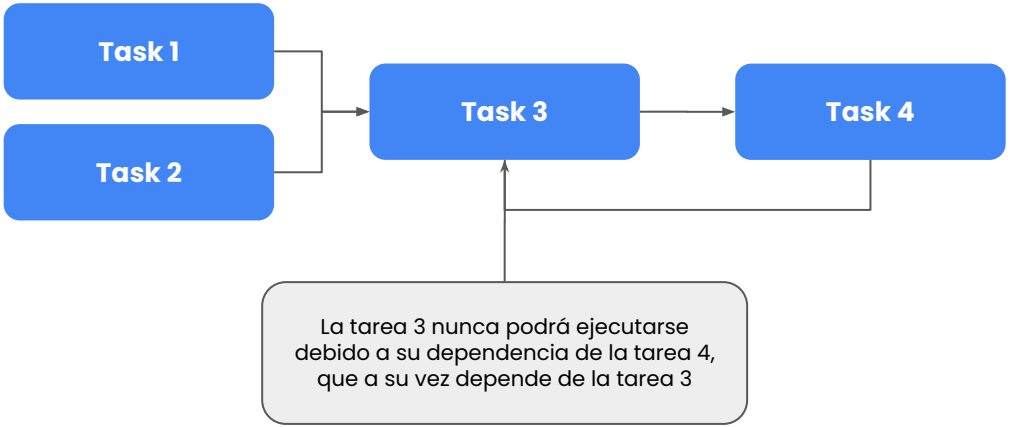


Gráfico Cíclico Dirigido de tareas



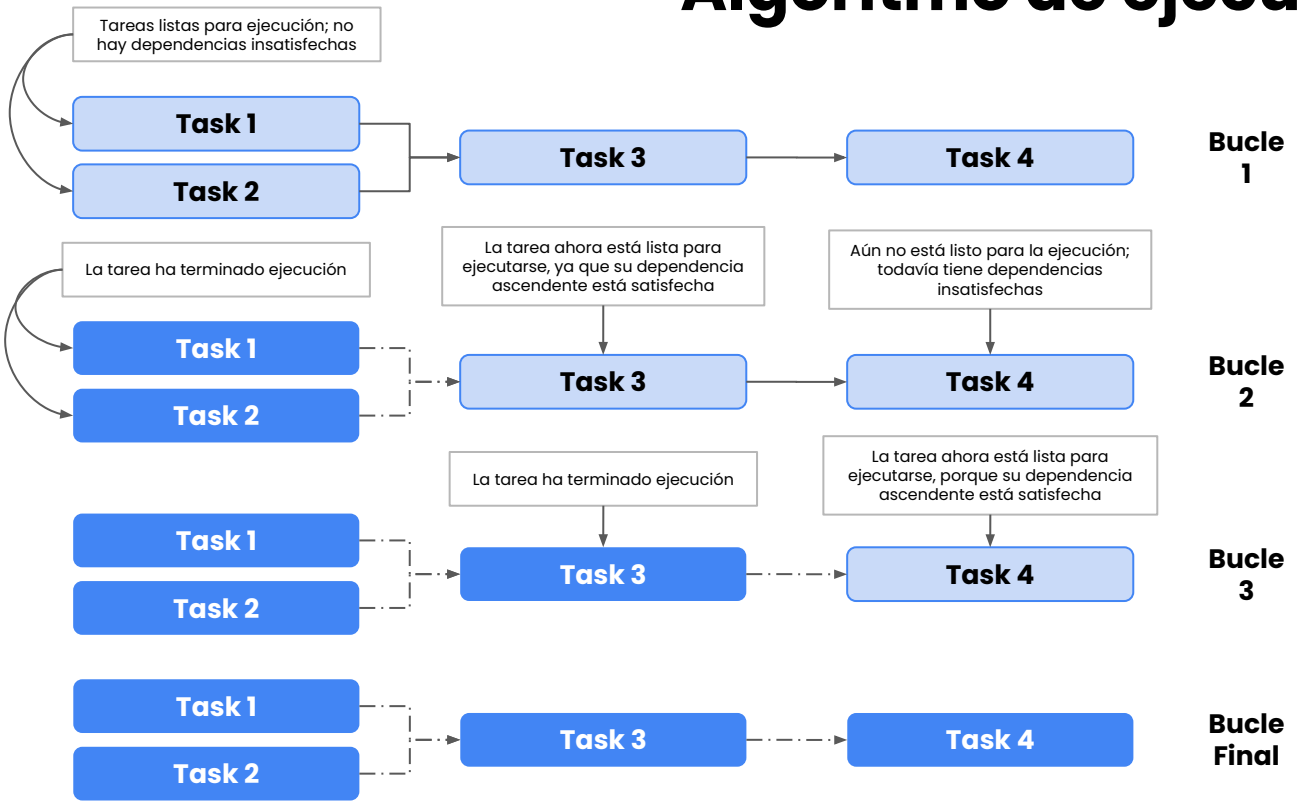
DAG

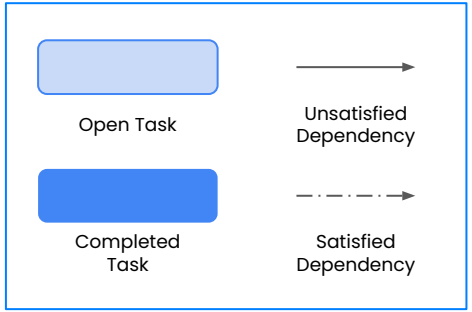
Gráfico Acíclico Dirigido

Un Gráfico Acíclico Dirigido (DAG) es un gráfico que contiene bordes dirigidos y no tiene ciclos ni bucles, lo que significa que no hay dependencias circulares entre las tareas. Esta propiedad de ser acíclico es fundamental, ya que evita situaciones en las que una tarea depende de otra y viceversa, lo que se conoce como una dependencia circular.

¿Cómo se ejecuta un DAG?

Algoritmo de ejecución de un DAG

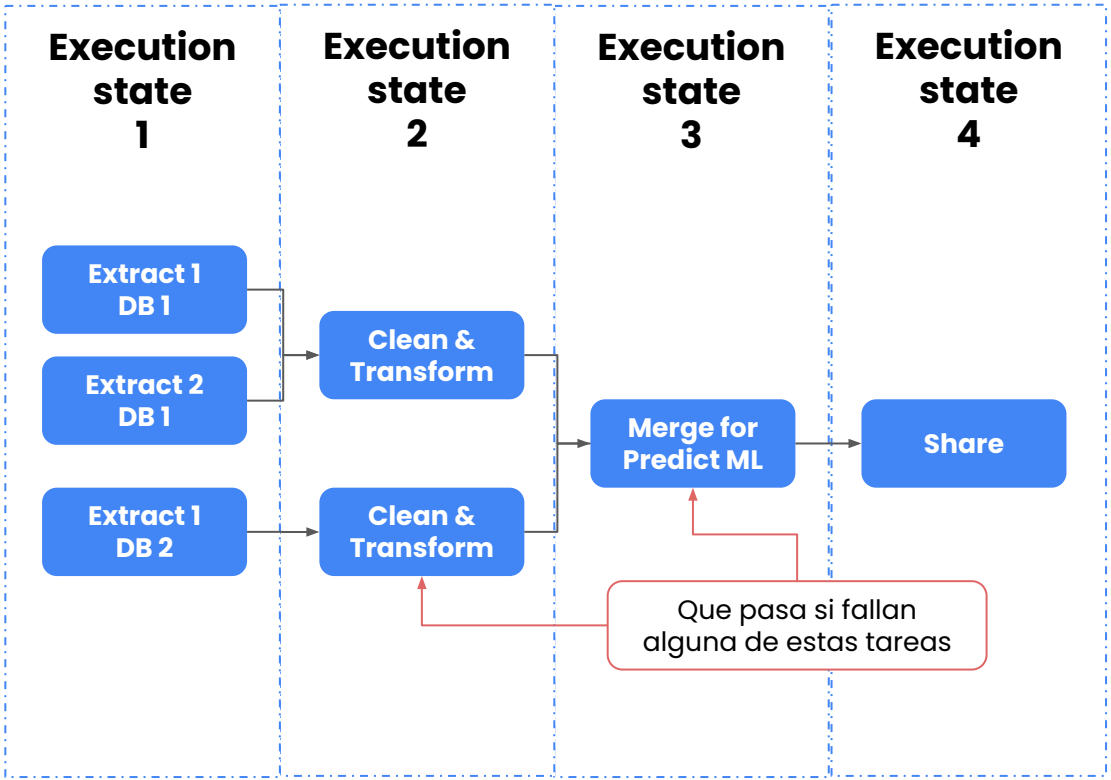




DAG vs. Scripts Secuenciales

Caso de Uso: Implementación de Aprendizaje Automático (ML)

Imaginemos un escenario en el que el propietario de una empresa busca utilizar el aprendizaje automático (ML) para mejorar la eficiencia de su operación. El objetivo es utilizar un modelo ML que correlacione las ventas de la empresa con los patrones de tráfico de personas en todas sus tiendas y así mejorar las promociones para vender más.



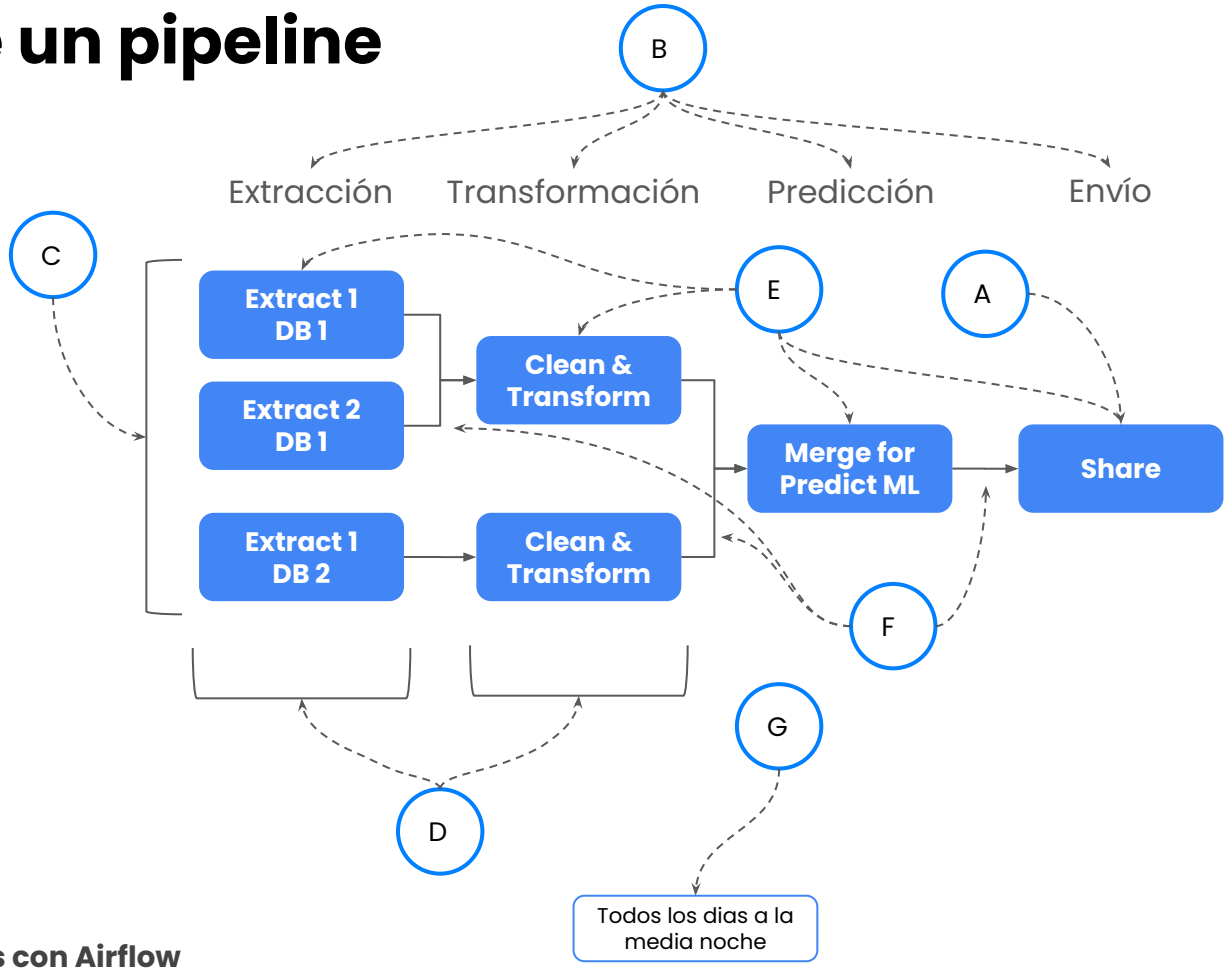
Ejemplo: Implementación de Aprendizaje Automático (ML)

¿Como pasar de un pipeline a un DAG?

Script Pipeline

Pasos a seguir:

A. Identificación de Objetivos
B. Desglose en Etapas
C. Identificación de Fuentes y Dependencias
D. Análisis de Paralelismo
E. Definición de Tareas
F. Secuenciación y Dependencias
G. Programación Temporal



Práctica: Creación de DAG’s

La empresa "CineFlow" está desarrollando un sistema de recomendación personalizada de películas.

El proceso se inicia con la recopilación de datos, que provienen de dos fuentes principales: la base interna de registros de comportamiento de usuarios dentro de la plataforma y los datos extraídos de una API pública que proporciona críticas de películas y otros metadatos relevantes.

Una vez recopilados, estos datos se procesan mediante un modelo de aprendizaje automático alojado en la nube que se consume desde una API. Este modelo analiza los patrones de comportamiento de los usuarios, las preferencias de visualización y otros datos relevantes para generar recomendaciones personalizadas para cada usuario.

Las recomendaciones resultantes se almacenan en una base de datos centralizada, que se actualiza mensualmente.

Por último, se implementa un sistema de notificación por correo electrónico para comunicar las recomendaciones a los usuarios.

Orquestación de flujos de datos con Airflow

Sistema de Recomendación de Películas

Vamos a Miro

Analisis de Redes Sociales para una Marca de Moda

La reconocida marca de moda "ChicTrends" busca mejorar su presencia en redes sociales mediante un proceso estructurado de análisis y mejora continua. El flujo de datos comienza con la extracción de métricas de múltiples plataformas sociales, como Instagram, Twitter y Facebook. Estas métricas incluyen datos de interacciones de usuarios, como likes, comentarios, shares, y otros indicadores relevantes para evaluar el rendimiento de las publicaciones y la percepción del público hacia la marca.

Posteriormente, se lleva a cabo un análisis de sentimientos utilizando un modelo de procesamiento de lenguaje natural (NLP) previamente entrenado, que se encuentra alojado en el servidor de la empresa.

Los resultados de este análisis se almacenan en una base de datos centralizada, donde se realiza la actualización automática de tableros de métricas para el equipo ejecutivo, permitiéndoles monitorear de manera continua el desempeño de la marca en las redes sociales y tomar decisiones estratégicas basadas en datos.

Orquestación de flujos de datos con Airflow

Pausa la clase y realiza el ejercicio

Sistema de Seguimiento de Inventario para una Tienda en Línea

La tienda en línea "E-Shopper" está buscando mejorar la gestión de su inventario para garantizar la disponibilidad de productos y optimizar sus operaciones.

Se lleva a cabo un proceso de reconciliación de inventario, donde se contrastan las ventas realizadas con las reposiciones y nuevas llegadas de productos para actualizar el estado actual del inventario. Durante este proceso, se utiliza un algoritmo de aprendizaje automático para predecir cuándo es probable que se agoten los productos disponibles en función de las tendencias de ventas y el ritmo de reposición.

Los resultados de este proceso se actualizan en una base de datos en la nube, lo que proporciona a "E-Shopper" una visión del estado de su inventario y de posibles problemas de escasez que puedan surgir.

Además, se implementa un sistema de alertas automáticas por correo electrónico para notificar al equipo de logística y gestión de inventario en caso de niveles críticos de existencias.

Pausa la clase y realiza el ejercicio

Monitoreo de Sensores en una Planta de Manufactura

En una planta de manufactura moderna, se implementa un sistema de monitoreo basado en sensores para supervisar el rendimiento de los equipos de producción. El flujo de datos comienza con la extracción de datos de los sensores distribuidos en la planta, los cuales se recopilan cada hora y se transmiten al sistema central para su procesamiento.

Una vez recopilados, los datos se someten a un análisis de calidad para identificar posibles problemas de producción o cualquier otra anomalía que pueda afectar la calidad o eficiencia del proceso de manufactura.

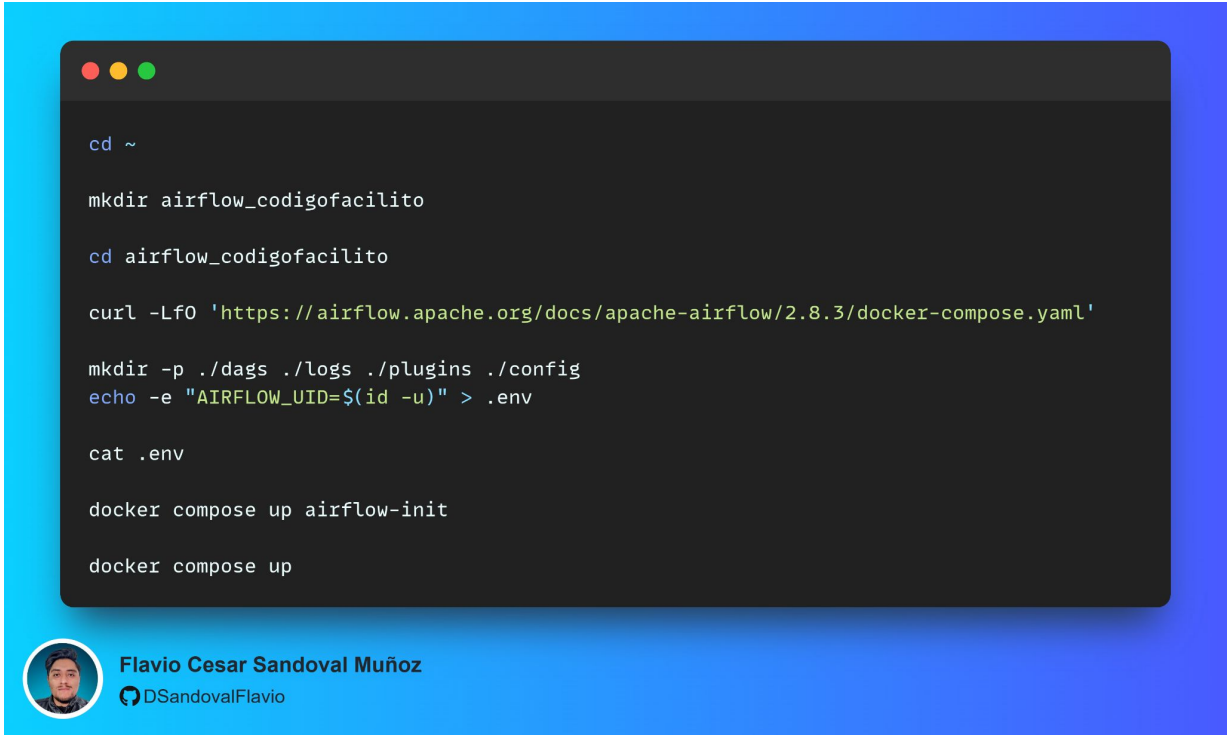
Los resultados de este análisis se resumen automáticamente y se envían a los supervisores y al equipo de mantenimiento a través de correos electrónicos.

Además, se actualizan tableros específicos para los departamentos de producción y mantenimiento, donde se visualizan de manera clara y concisa los datos recopilados, los hallazgos del análisis de calidad y el estado actual de los equipos.

Pausa la clase y realiza el ejercicio

Instalación de Apache Airflow

Componentes de Airflow - Docker compose



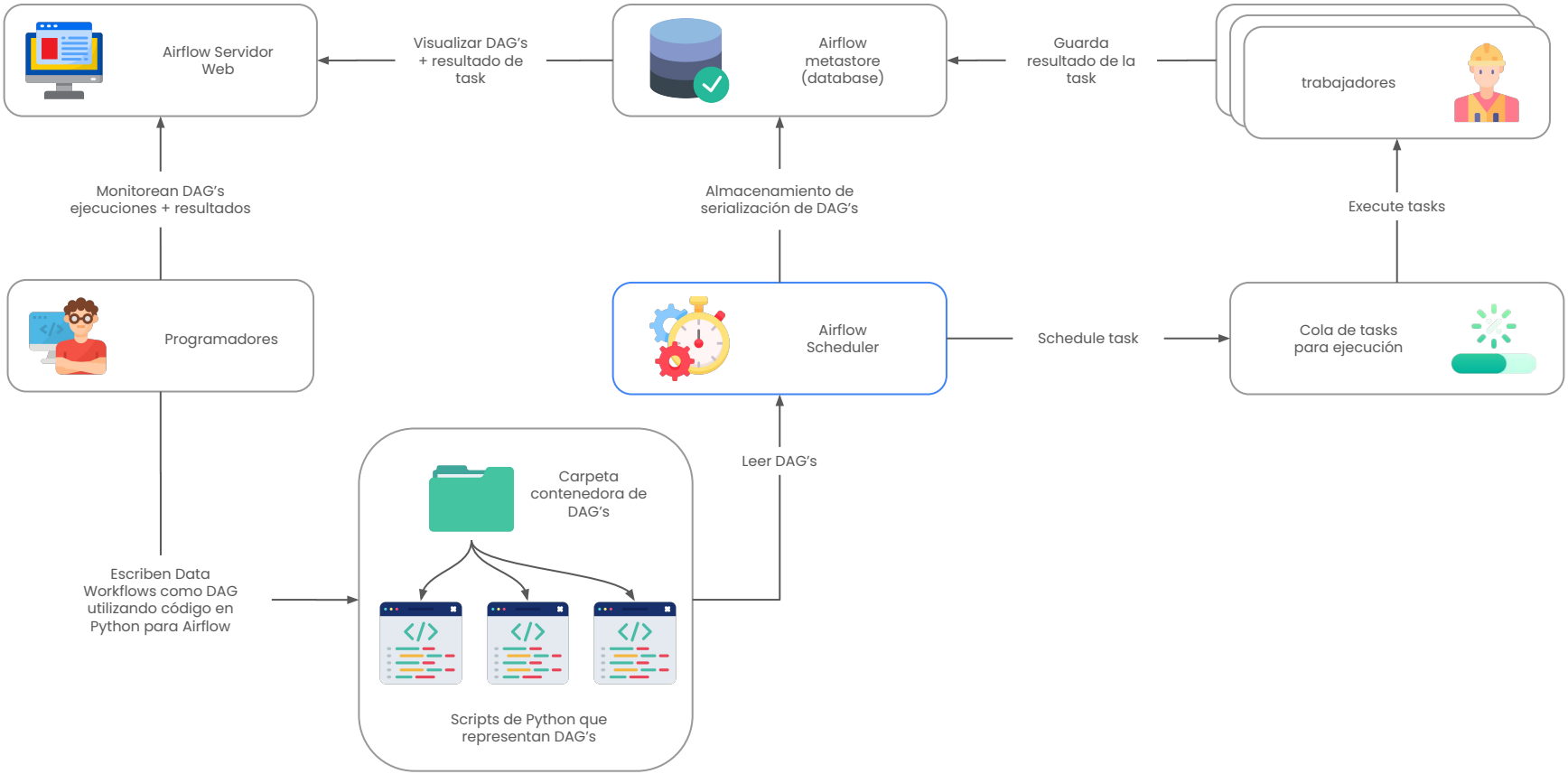
- airflow-scheduler
- airflow-webserver
- airflow-worker
- airflow-triggerer
- airflow-init
- postgres
- redis

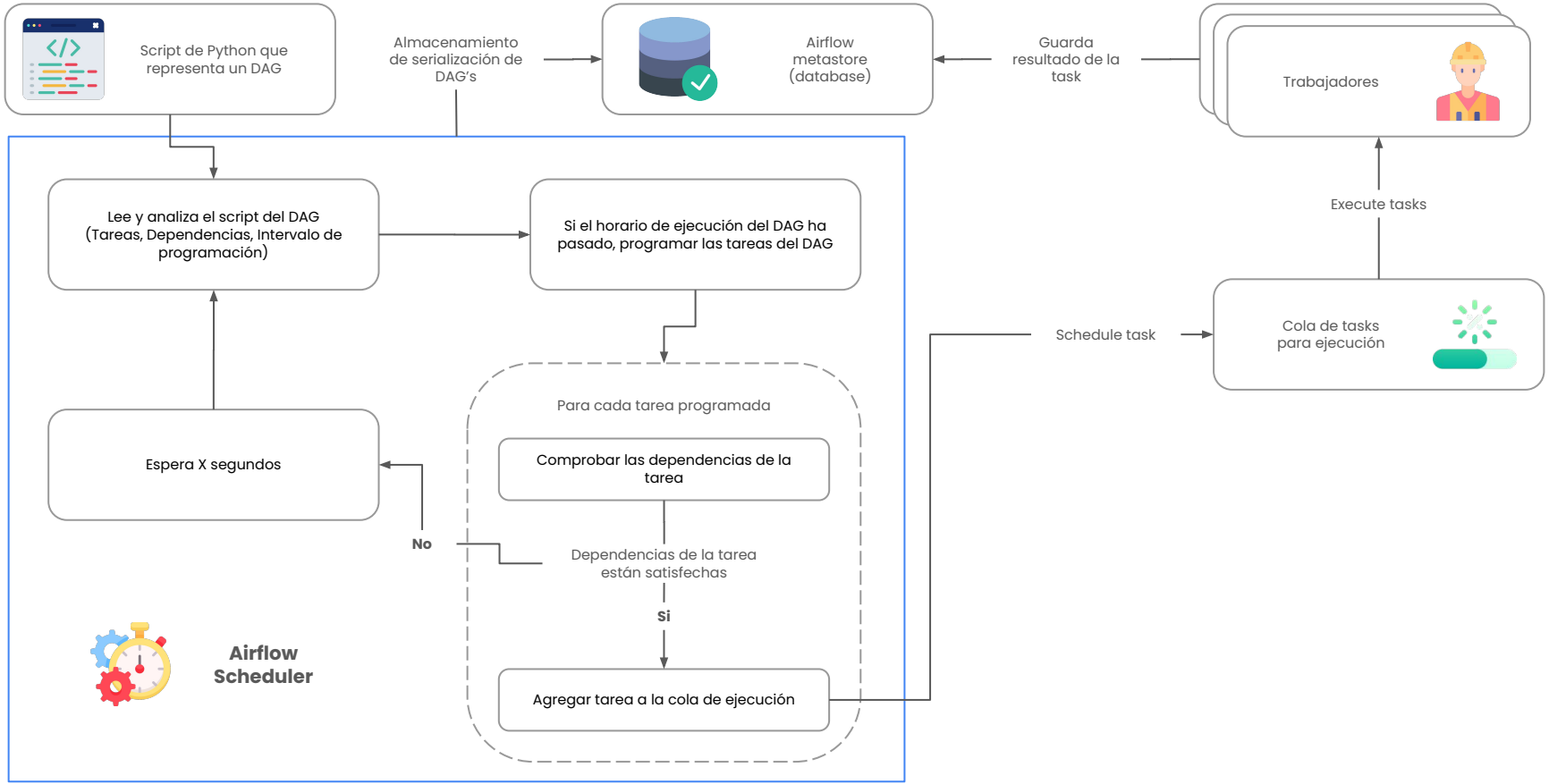
Programación y Ejecución de DAG's en Airflow

Para comprender cómo Airflow lleva a cabo la ejecución de los DAG, es esencial examinar el proceso general que involucra el desarrollo y la ejecución en Airflow.

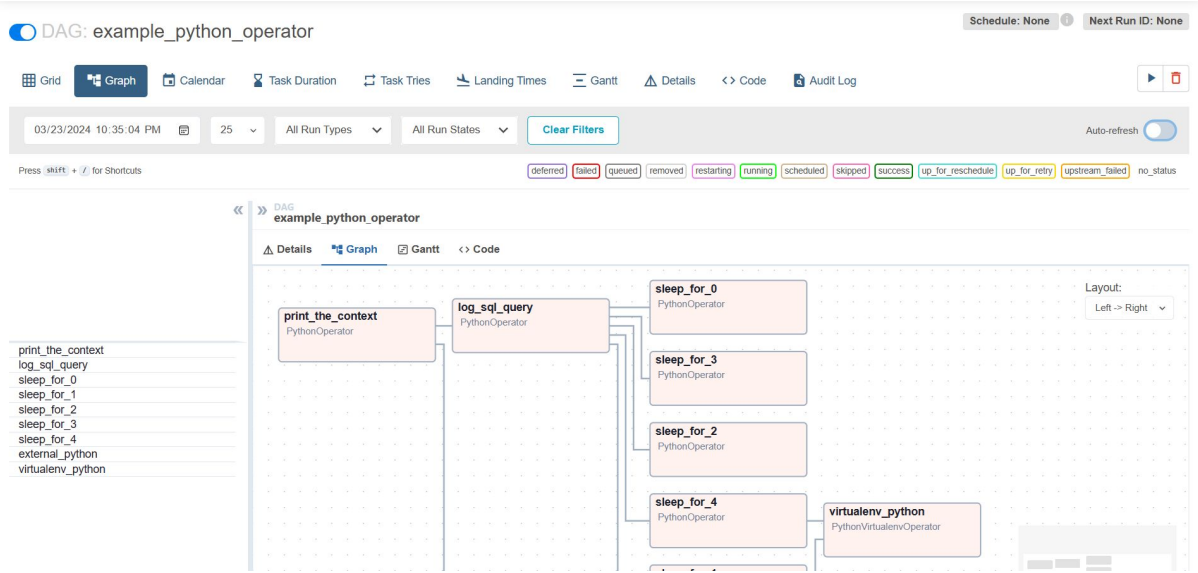
A un nivel más alto, Airflow se organiza en tres componentes fundamentales:

- **Programador de Airflow**
- **Trabajadores de Airflow**
- **Servidor web de Airflow**.


Airflow Servidor Web
Visualizar DAG's + resultado de task
Airflow metastore (database)
Guarda resultado de la task
trabajadores
Monitorean DAG's ejecuciones + resultados
Almacenamiento de serialización de DAG's
Execute tasks
Programadores
Airflow Scheduler
Schedule task
Cola de tasks para ejecución
Leer DAG's
Carpeta contenedora de DAG's
Escriben Data Workflows como DAG utilizando código en Python para Airflow
Scripts de Python que representan DAG's

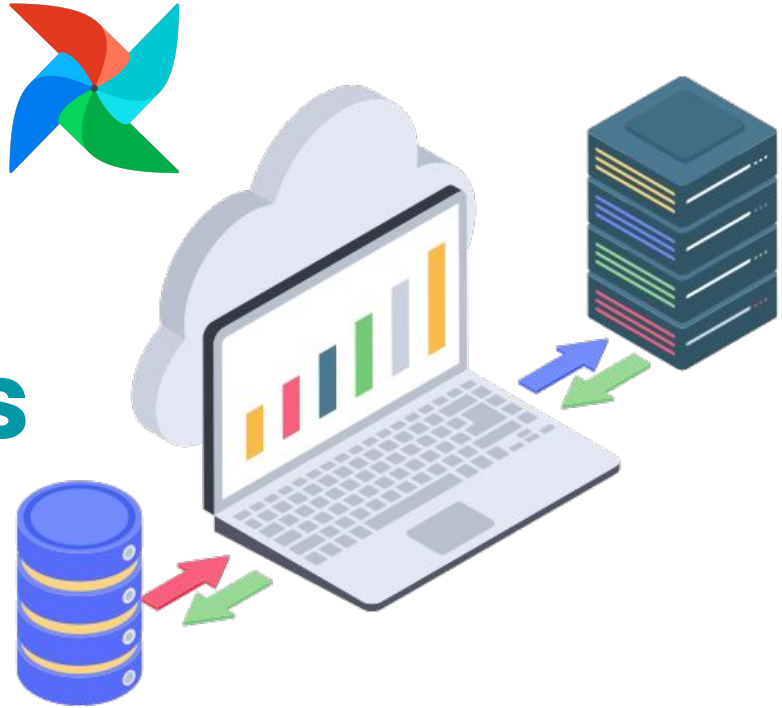

Script de Python que representa un DAG
Almacenamiento de serialización de DAG's
Airflow metastore (database)
Guarda resultado de la task
Trabajadores
Execute tasks
Lee y analiza el script del DAG (Tareas, Dependencias, Intervalo de programación)
Si el horario de ejecución del DAG ha pasado, programar las tareas del DAG
Schedule task
Cola de tasks para ejecución
Para cada tarea programada
Comprobar las dependencias de la tarea
Espera X segundos
No
Dependencias de la tarea están satisfechas
Si
Agregar tarea a la cola de ejecución
Airflow Scheduler

Funcionalidades de la interface web de Airflow



- Visualización de DAGs
- Monitoreo y la gestión de las ejecuciones de los DAG
- Vista de Grafos
- Ejecución Manual de DAG
- Administración de conexiones y variables

Cuándo Utilizar Apache Airflow Consideraciones y Ventajas



Consideraciones y Razones Para Elegir Airflow

- Flexibilidad en la Implementación
- Ampliación e Integración
- Semántica de Programación Enriquecida
- Funciones Avanzadas como Backfilling
- Interfaz Web
- Código Abierto y Soluciones Gestionadas

Consideraciones y Razones Para No Elegir Airflow

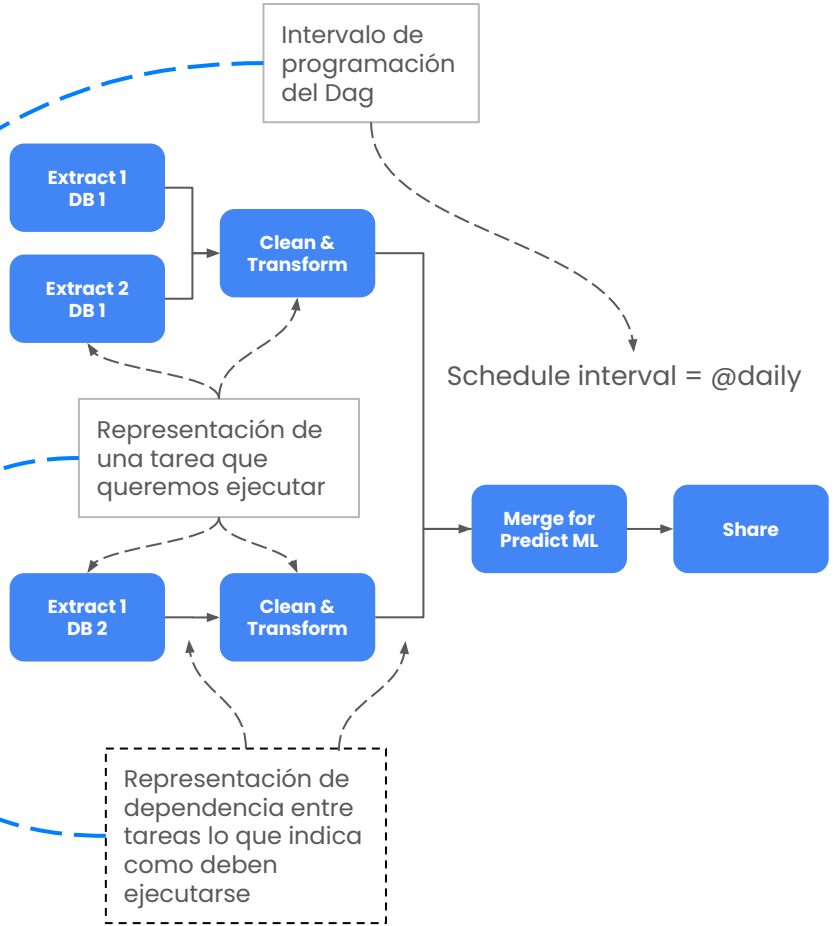
- Cargas de Trabajo de Transmisión
- Canalizaciones Altamente Dinámicas
- Poca Experiencia en Programación (Python)
- Complejidad para Grandes Casos de Uso
- Limitaciones en Funcionalidades Más Amplias

Anatomía y Desarrollo de un DAG con Python

DAG

```python
with DAG(
    'Dag_1',
    default_args=default_args,
    description='Dag',
    schedule_interval=timedelta(days=1),
) as dag:

    extract1_db1 = DummyOperator(task_id='extract1_db1')
    extract2_db1 = DummyOperator(task_id='extract2_db1')

    extract1_db2 = DummyOperator(task_id='extract1_db2')

    transform_db1 = DummyOperator(task_id='transform_db1')
    transform_db2 = DummyOperator(task_id='transform_db2')

    merge_db1_db2 = DummyOperator(task_id='merge_db1_db2')

    load_db = DummyOperator(task_id='load_db')

    [extract1_db1, extract2_db1] >> transform_db1
    extract1_db2 >> transform_db2
    [transform_db1, transform_db2] >> merge_db1_db2 >> load_db
```

Flavio Cesar Sandoval Muñoz
DSandovalFlavio



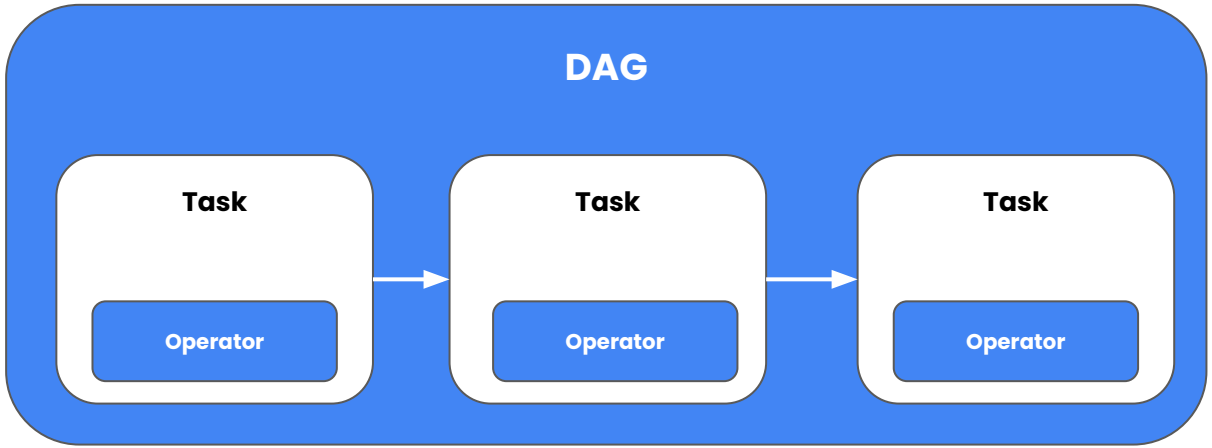
```python
default_args = {
    "owner": "airflow", # Dueño del DAG
    "depends_on_past": True, # Dependencia del DAG con respecto a la ejecución anterior
    "email" : ["airflow@example.com"], # Correo electrónico para recibir notificaciones
    "email_on_failure": False, # Enviar correo en caso de fallo
    "email_on_retry": False, # Enviar correo en caso de reintentos
    "retries": 1, # Número de reintentos en caso de fallo
    "retry_delay": timedelta(minutes=2), # Tiempo de espera entre reintentos
    "on_failure_callback": task_fail_slack_alert, # Función a ejecutar en caso de fallo
    "on_retry_callback": task_retry_slack_alert, # Función a ejecutar en caso de reintentos
    "on_success_callback": task_success_slack_alert, # Función a ejecutar en caso de éxito
    "tags": ["CF"] # Etiquetas para identificar el DAG
}
```

Flavio Cesar Sandoval Muñoz
DSandovalFlavio

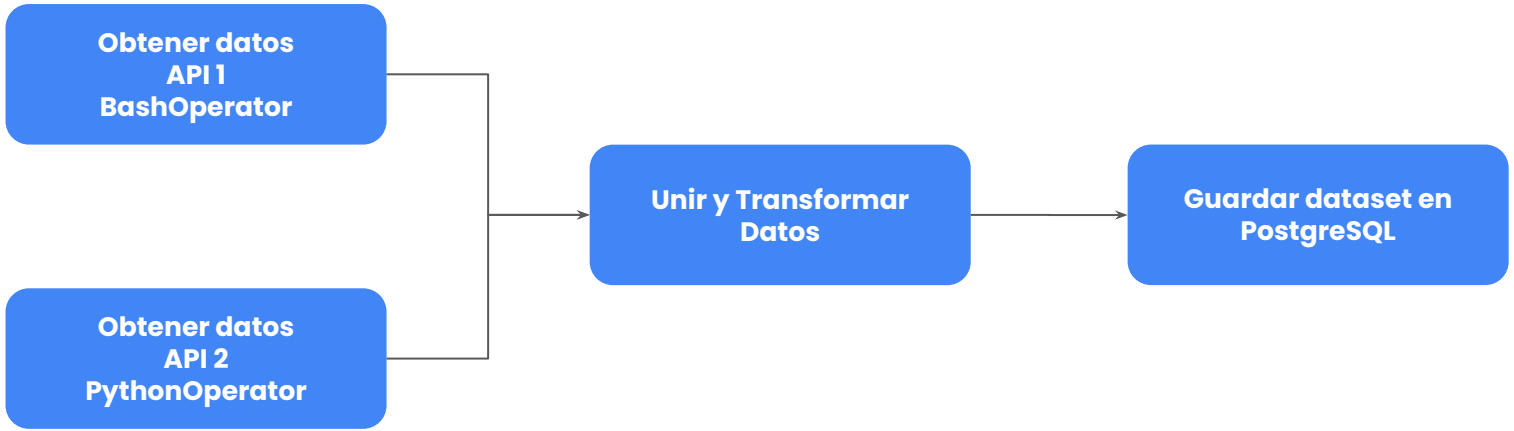
¿Qué es un Operador?

Operadores en Apache Airflow

Cuando se trabaja con Apache Airflow, dos conceptos cruciales que a menudo se entrelazan son “tareas” y “operadores”. Comprender la diferencia entre ambos es esencial para desarrollar flujos de trabajo efectivos y aprovechar al máximo la potencia de Airflow.

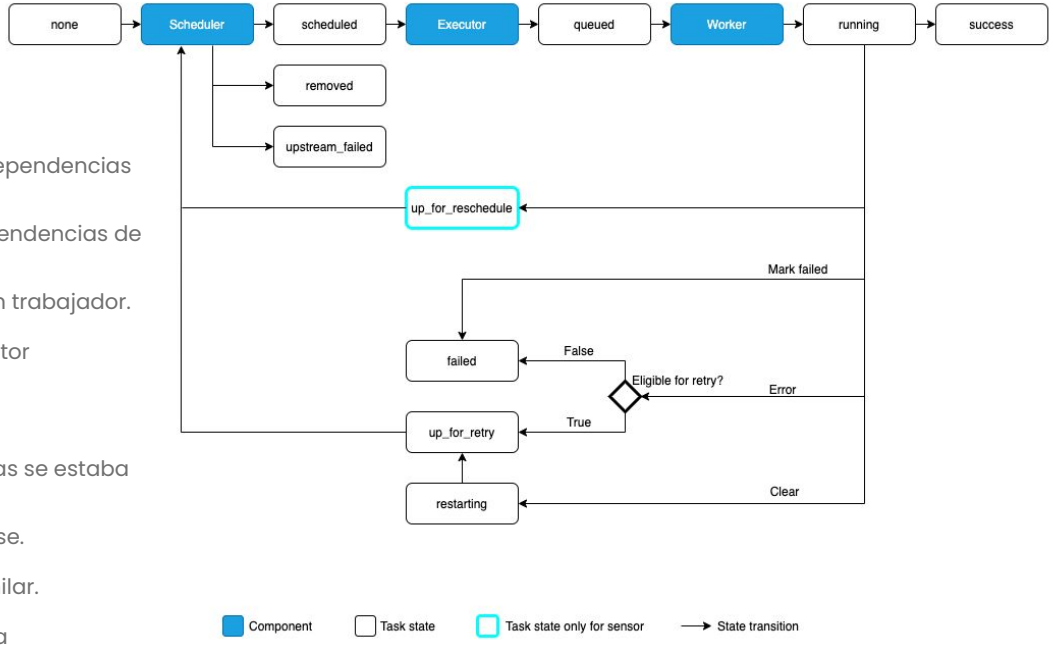


Tu primer DAG en código



¿Que pasa si algo falla?

¿Y si una tarea falla?



none: La tarea aún no se ha puesto en cola para su ejecución (sus dependencias aún no se cumplen)

scheduled: El programador ha determinado que se cumplen las dependencias de la tarea y debe ejecutarse.

queued: La tarea se ha asignado a un ejecutor y está esperando a un trabajador.

running: La tarea se está ejecutando en un trabajador (o en un ejecutor local/síncrono).

success: La tarea terminó de ejecutarse sin errores.

restarting: Se solicitó externamente que la tarea se reiniciara mientras se estaba ejecutando.

failed: La tarea tuvo un error durante la ejecución y no pudo ejecutarse.

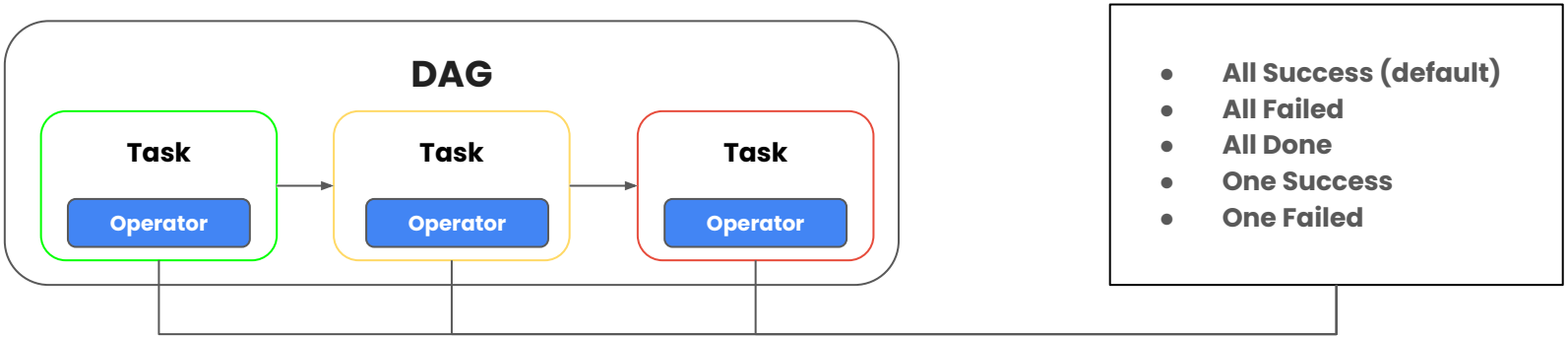
skipped: La tarea se omitió debido a ramificaciones, LatestOnly o similar.

upstream_failed: Una tarea anterior falló y el Trigger Rule dice que la necesitábamos.

up_for_retry: La tarea falló, pero le quedan intentos de reintento y se volverá a programar.

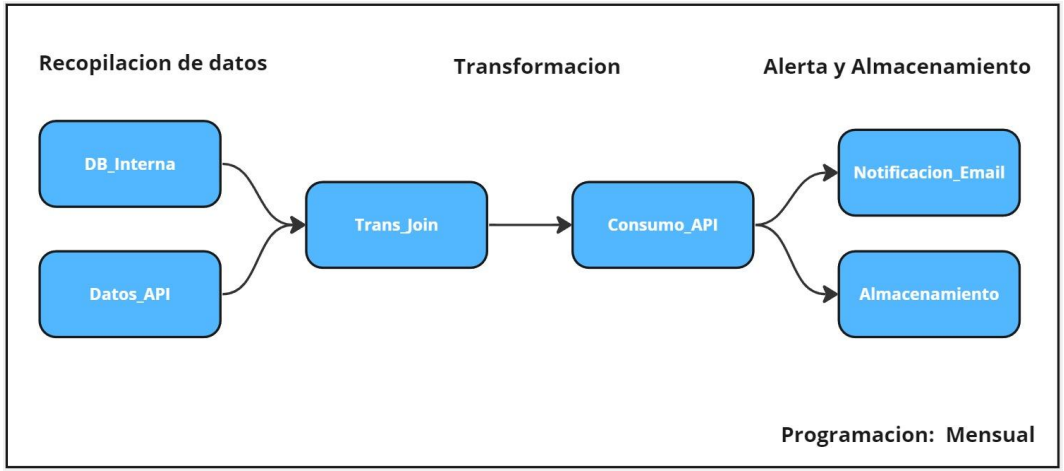
up_for_reschedule: La tarea es un Sensor que está en modo eschedule.

Trigger Rules



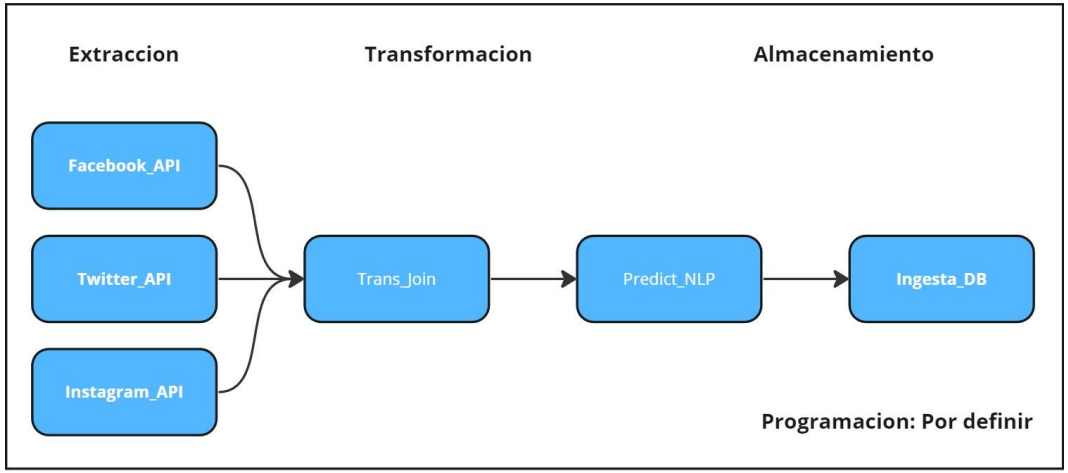
- `all_success` : (predeterminado) todos los padres lo han logrado
- `all_failed` : todos los padres están en un estado `failed` o `upstream_failed`
- `all_done` : todos los padres han terminado con su ejecución
- `one_failed` : se activa tan pronto como al menos uno de los padres falla, no espera a que todos los padres hayan terminado
- `one_success` : se activa tan pronto como al menos uno de los padres tiene éxito, no espera a que todos los padres terminen
- `none_failed` : todos los padres no han fallado (`failed` o `upstream_failed`), es decir, todos los padres han tenido éxito o se han omitido
- `none_failed_or_skipped` : todos los padres no han fallado (`failed` o `upstream_failed`) y al menos uno de los padres ha tenido éxito.
- `none_skipped` : ningún padre está en un `skipped` estado, es decir, todos los padres están en un estado `success` , `failed` o `upstream_failed`
- `dummy` : las dependencias son sólo para mostrar, se activan a voluntad

Prácticas: Creación de DAGs con Python y Apache Airflow



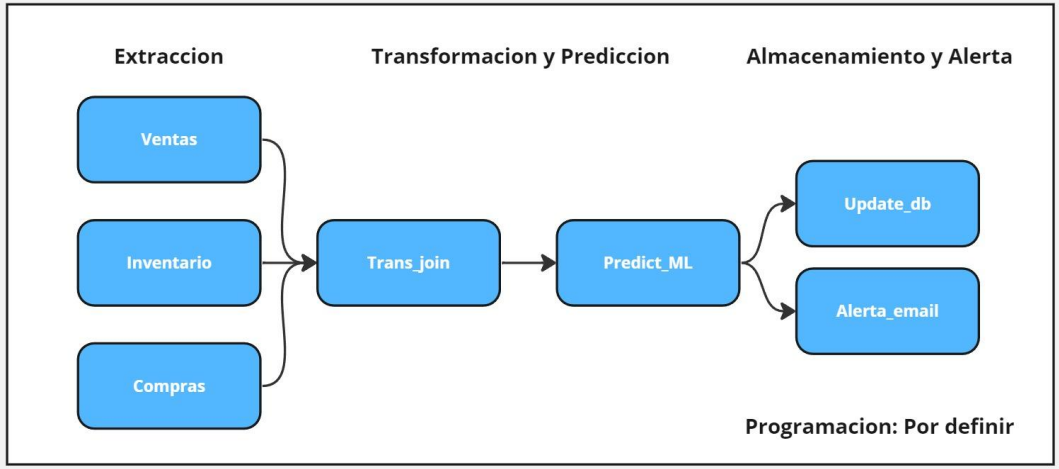
Sistema de Recomendación de Películas

Orquestación de flujos de datos con Airflow



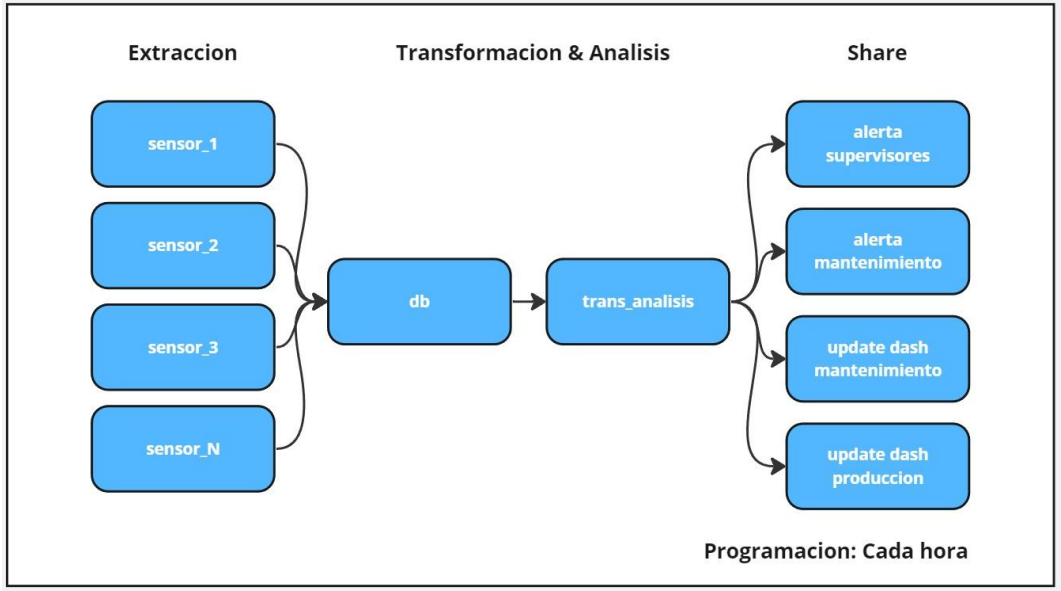
Análisis de Redes Sociales para una Marca de Moda

Orquestación de flujos de datos con Airflow



Sistema de Seguimiento de Inventario para una Tienda en Línea

Orquestación de flujos de datos con Airflow



Monitoreo de Sensores en una Planta de Manufactura

Orquestación de flujos de datos con Airflow

Providers para la ampliación de funcionalidades de Airflow

Google Cloud

databricks

Práctica: Creación de DAGs utilizando elementos de GCP

Tipos de operadores que podríamos utilizar

Operadores de Google Cloud Storage (GCS):

- GCSToBigQueryOperator: Transfiere datos desde Google Cloud Storage a BigQuery.
- BigQueryToGCSOperator: Transfiere datos desde BigQuery a Google Cloud Storage.
- GCSToGCSOperator: Transfiere datos entre dos ubicaciones de Google Cloud Storage.
- LocalFilesystemToGCSOperator: Transfiere datos desde Local PC a Google Cloud Storage.

Estos operadores están diseñados para interactuar específicamente con servicios de Google Cloud Platform (GCP) y son útiles cuando trabajas con datos almacenados en GCS o BigQuery.

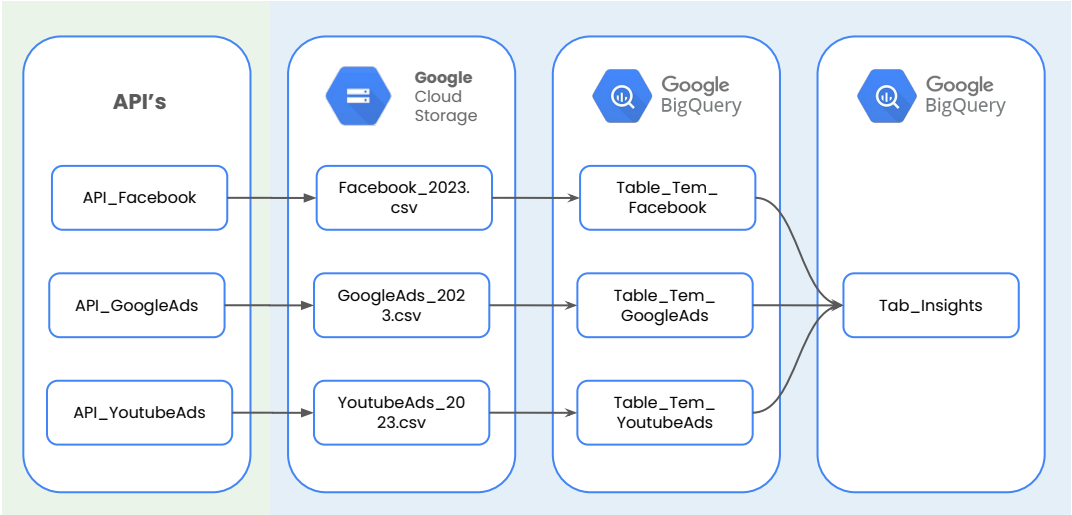
Google Cloud

Google Big Query

Google Cloud Storage

Automatización de Integración y Análisis de Datos de Campañas de Marketing

La agencia digital "Elevate Marketing" ha ejecutado con éxito ocho campañas de marketing a lo largo del último año en tres plataformas principales: Facebook, Google Ads y YouTube Ads. Cada campaña ha generado archivos de datos específicos, nombrados de acuerdo con la plataforma y el año, como Facebook_2023.csv, GoogleAds_2023.csv y YoutubeAds_2023.csv. La agencia busca consolidar y analizar estos datos para obtener insights valiosos cada mes.



Orquestación de flujos de datos con Airflow

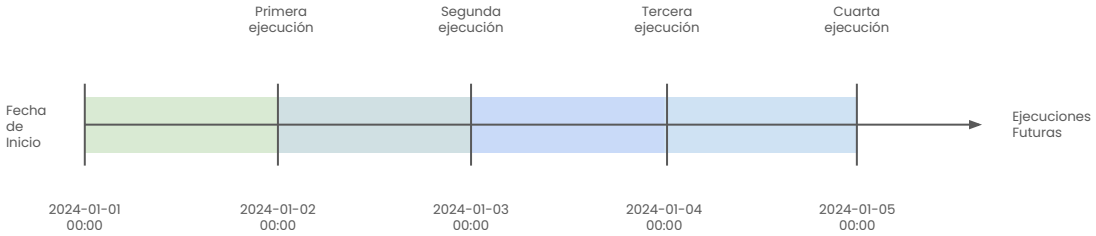
Vamos al código

Intervalos de Programación en Airflow

Automatización Programada con Apache Airflow DAGs

El argumento **schedule_interval** al inicializar un DAG tiene, por defecto, un valor de **None**. En consecuencia, el DAG no se programará automáticamente y solo se ejecutará cuando se active manualmente desde la interfaz de usuario o la API. No obstante, al establecer la frecuencia de ejecución con la macro **@daily**, se generarán ejecuciones diarias a medianoche.

Otro aspecto importante a tener en cuenta es la fecha de inicio, ya que desempeña un papel crucial en la determinación de la primera ejecución y los intervalos subsiguientes.



Intervalos de programación Cron

La sintaxis cron consta de cinco componentes (minuto, hora, día del mes, mes y día de la semana) y permite una representación detallada del momento en que debe ejecutarse un trabajo cron. **Los asteriscos (*) se utilizan para definir campos sin restricciones**, lo que significa que no se tiene en cuenta el valor de ese campo.

Ejemplos de expresiones cron y sus significados incluyen:

- **0 * * * ***: Cada hora, ejecutando en la hora.
- **0 0 * * ***: Todos los días, ejecutando a la medianoche.
- **0 0 * * 0**: Semanal, ejecutando a la medianoche del domingo.

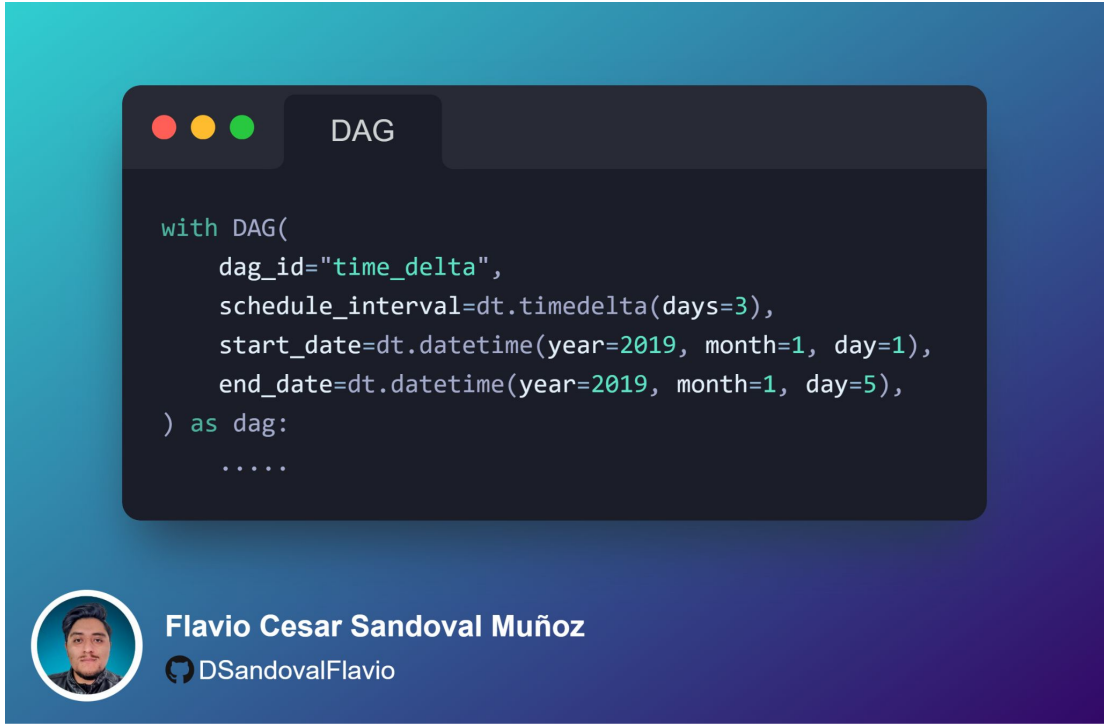
La sintaxis cron también permite definir expresiones más complejas, como:

- **0 0 1 * ***: La medianoche del primero de cada mes.
- **45 23 * * SAT**: A las 23:45 todos los sábados.

Además, las expresiones cron permiten definir colecciones de valores mediante comas (,) para listas de valores y guiones (-) para rangos de valores. Ejemplos incluyen:

- **0 0 * * MON,WED,FRI**: Correr todos los lunes, miércoles, y viernes a la medianoche.
- **0 0 * * MON-FRI**: Ejecutar todos los días de la semana a la medianoche.
- **0 0,12 * * ***: Correr todos los días a las 00:00 y 12:00.

Intervalos de programación basados en frecuencia



El DAG se ejecutará cada tres días después de la fecha de inicio, es decir, en los días 4, 7, 10 y así sucesivamente de enero de 2019. Este enfoque proporciona una solución sencilla y flexible para definir intervalos de tiempo basados en la frecuencia sin preocuparse por las limitaciones de las expresiones cron.

Adicionalmente, esta técnica también es aplicable para programar DAGs en intervalos más cortos o largos, como ejecutar el DAG cada 10 minutos (timedelta(minutes=10)) o cada dos horas (timedelta(hours=2)).

DAG

```python
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime

dag = DAG(
    'mi_dag_bash',
    schedule_interval='@daily',
    start_date=datetime(2023, 1, 1)
)

# Utilizando plantillas en el operador Bash
tarea_bash = BashOperator(
    task_id='tarea_bash',
    bash_command='echo {{ params.mensaje }}',
    params={'mensaje': '¡Hola, Airflow!'},
    dag=dag
)
```

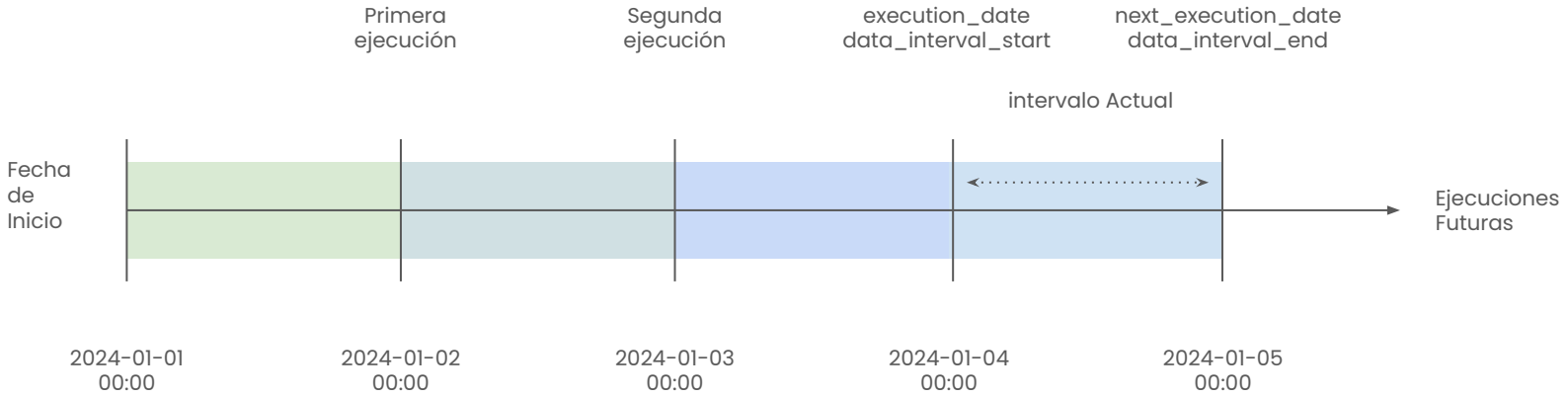
Flavio Cesar Sandoval Muñoz
DSandovalFlavio

Uso de Plantillas en la Definición de Tareas con Operadores en Airflow

Variable	Type	Description
`{{ data_interval_start }}`	pendulum.DateTime	Start of the data interval. Added in version 2.2.
`{{ data_interval_end }}`	pendulum.DateTime	End of the data interval. Added in version 2.2.
`{{ ds }}`	str	The DAG run's logical date as `YYYY-MM-DD`. Same as `{{ dag_run.logical_date | ds }}`.
`{{ ds_nodash }}`	str	Same as `{{ dag_run.logical_date | ds_nodash }}`.
`{{ ts }}`	str	Same as `{{ dag_run.logical_date | ts }}`. Example: `2018-01-01T00:00:00+00:00`.
`{{ ts_nodash_with_tz }}`	str	Same as `{{ dag_run.logical_date | ts_nodash_with_tz }}`. Example: `20180101T000000+0000`.
`{{ ts_nodash }}`	str	Same as `{{ dag_run.logical_date | ts_nodash }}`. Example: `20180101T000000`.
`{{ prev_data_interval_start_success }}`	pendulum.DateTime | `None`	Start of the data interval of the prior successful `DagRun`. Added in version 2.2.
`{{ prev_data_interval_end_success }}`	pendulum.DateTime | `None`	End of the data interval of the prior successful `DagRun`. Added in version 2.2.
`{{ prev_start_date_success }}`	pendulum.DateTime | `None`	Start date from prior successful `DagRun` (if available).
`{{ dag }}`	DAG	The currently running `DAG`. You can read more about DAGs in DAGs.
`{{ task }}`	BaseOperator	The currently running `BaseOperator`. You can read more about Tasks in Opera
`{{ macros }}`		A reference to the macros package. See Macros below.
`{{ task_instance }}`	TaskInstance	The currently running `TaskInstance`.
`{{ ti }}`	TaskInstance	Same as `{{ task_instance }}`.

Plantillas de Airflow

Ejecuciones temporales dinámicas utilizando las fechas de ejecución



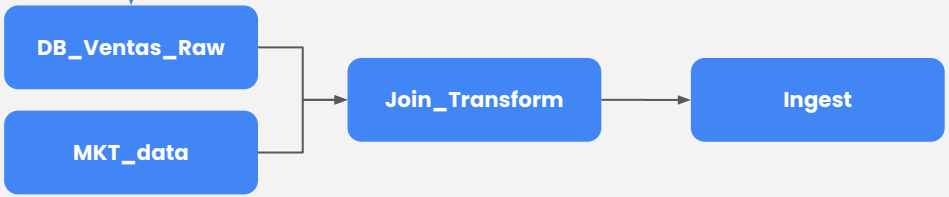
Vamos al código

Sensores en Airflow

ExternalTaskSensor

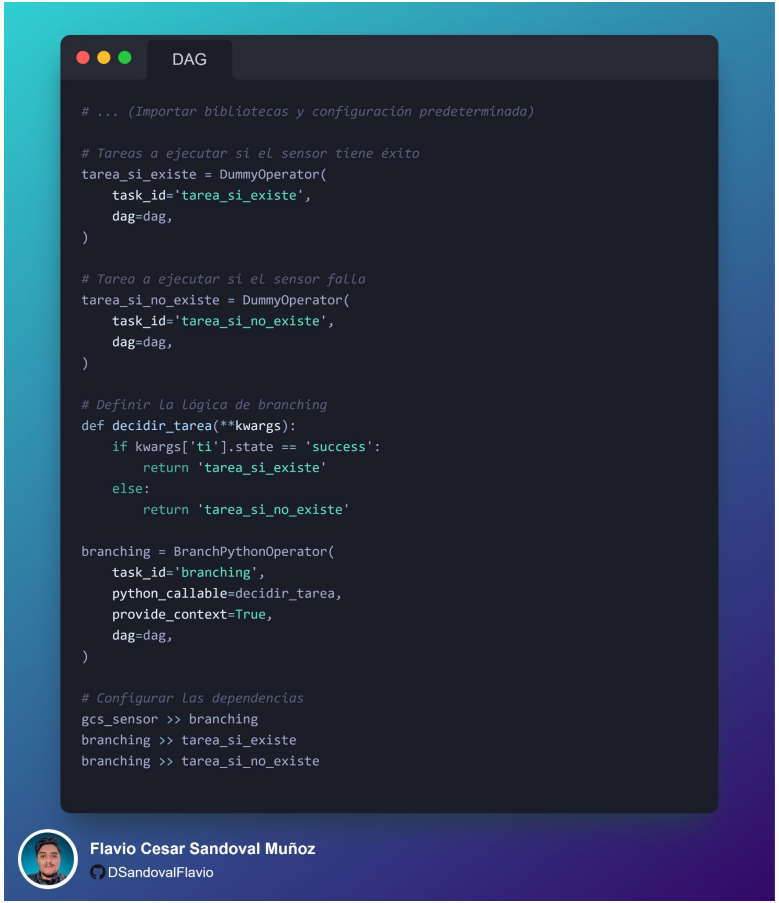
DAG_Ventas

DAG_Analitica_MKT



Vamos al código

Branch Operator



BranchPythonOperator

Es un operador en Apache Airflow que permite tomar decisiones en tiempo de ejecución dentro de un DAG (Grafo Acíclico Dirigido). Este operador se utiliza para crear ramificaciones en el flujo de tareas basándose en el resultado de una función Python. Dependiendo del resultado de esta función, el flujo del DAG puede dirigirse por diferentes caminos.

Vamos al código

Apache
Airflow

En resumen, Airflow no solo ofrece programación y ejecución eficientes de DAG, sino que también proporciona herramientas integrales de monitoreo y manejo de fallas a través de su interfaz web, permitiendo a los usuarios comprender, depurar y mejorar continuamente sus canalizaciones de manera efectiva.