



ACTIVIDAD 07
APLICACIÓN DE AIRFLOW

MATERIA:

Computación tolerante a fallas
(I7036)

NRC:

179961

SECCIÓN:

D06

MAESTRO:

López Franco, Michel
Emmanuel

HORARIO:

Lunes y Miércoles, 11:00 –
12:55

FECHA DE ENTREGA:

14/04/2023

ALUMNO:

J. Emmanuel Ortiz Renteria
(219747611)

Contenido

Introducción	3
Objetivos y descripción del programa	3
Código del programa.....	4
Ejemplo 1.....	4
Ejemplo 2.....	6
Capturas y explicación de ejecución.....	8
Ejemplo 1:.....	8
Ejemplo 2:.....	9

Introducción

Apache Airflow es un manejador de flujo de trabajo basado en Python de código abierto que permite diseñar, programar y monitorear canalizaciones de datos (data pipelines). La herramienta representa procesos en forma de gráficos acíclicos dirigidos (DAG) que visualizan relaciones casuales entre tareas y el orden de su ejecución.



Airflow funciona con canalizaciones por lotes, que son secuencias de tareas finitas, con un inicio y un final claros, iniciados en ciertos intervalos o mediante activadores. No es una herramienta de procesamiento de datos en sí misma, sino más bien un instrumento para gestionar múltiples componentes del procesamiento de datos. Tampoco está diseñado para flujos de trabajo de transmisión continua.

Las aplicaciones más comunes de la plataforma son:

- Envío de datos, o tomar datos de un origen y trasladarlos a un almacén de datos local, un lago de datos o una plataforma de datos basada en la nube.
- Canales de aprendizaje automático de un extremo a otro.
- Integración de datos a través de canales complejos ETL/ELT (extracción-transformación-carga/extracción-carga-transformación).
- Generación automatizada de informes.
- Tareas de DevOps, por ejemplo, crear copias de seguridad programadas y restaurar datos a partir de ellas.
- Es especialmente útil en situaciones donde el flujo de trabajo usa Big Data.

Los Directed Acyclic Graphs (DAGs), o Grafos Acíclicos Dirigidos, son una parte integral de Airflow. Representan visualmente los flujos de trabajo como una serie de tareas interconectadas, donde cada nodo del grafo representa una tarea y las flechas indican la dirección de las dependencias entre ellas. Los DAG permiten al usuario definir la lógica de ejecución de sus flujos de trabajo de manera clara y concisa, lo que facilita la comprensión, la gestión y la escalabilidad de los procesos automatizados. Con Airflow, los DAGs se convierten en una poderosa herramienta para manejar, representar y reportar el flujo de tareas complejas de manera eficiente.

Objetivos y descripción del programa

Los programas son ejemplos sencillos de ejecución para manejar las herramientas de Airflow y aprender acerca de una herramienta que ayuda al manejo de monitoreo y solución de errores. Los principales temas a tratar en el reporte son:

- Mostrar la construcción de un DAG detectado por Airflow
- Aprovechar la herramienta DAG para construir Grafos o Diagrama de Grantt, que representan el flujo de trabajo determinado.
- Usar los recursos de monitoreo en tiempo real, y guardados para generar informes de ejecución guardadas en la base de datos integradas en la plataforma.

Código del programa

Ejemplo 1

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta

def print_hello():
    return 'Hello, World!'

def imprimir_calculo():
    a = 10
    b = 5
    return (a*b)

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 2, 10),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(seconds=10),
}

dag = DAG(
    dag_id='hello_world',
    default_args=default_args,
    'start_date': datetime(2024, 2, 10),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay':
timedelta(minutes=5),
    description='A simple "Hello, World!" DAG',
    schedule_interval=timedelta(days=1),
)

hello_task = PythonOperator(
    task_id='print_hello',
    python_callable=print_hello,
    dag=dag,
)

calculo_task = PythonOperator(
    task_id='print_calculo',
    python_callable=imprimir_calculo,
    dag=dag,
)

hello_task>>>calculo_task
```

Descripción del programa:

El DAG se llama 'hello_world' y se programa para ejecutarse diariamente. Contiene dos tareas:

1. Regresar la cadena “Hello, World!”.
2. Regresar el valor de una operación de multiplicación.

Ejemplo 2

```
from datetime import datetime, timedelta
import random
from airflow import DAG
from airflow.operators.python import PythonOperator

def print_hello():
    print('¡Hola Mundo!')

def generate_random_numbers():
    num1 = random.randint(0, 50)
    num2 = random.randint(0, 50)
    resultado = num1+num2
    if (resultado%2 == 0):
        return 'print_resultado_par'
    else:
        return 'print_resultado_impar'

def imprimir_par():
    print("El resultado es par.")

def imprimir_impar():
    print("El resultado es impar")

args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 12, 4),
    'retries': 1,
    'retry_delay': timedelta(seconds=10),
}

# Crear un objeto DAG
dag = DAG(
    catchup=False,
    dag_id='ejemplo_suma',
    default_args=args,
    start_date=datetime(2024,1,1),
    description='DAG que genera dos números al azar y determina si la suma es par o impar.',
    schedule_interval='*/5 * * * *',
)

# Definir las tareas del DAG
task1 = PythonOperator(
    task_id='print_hello',
    python_callable=print_hello,
    dag=dag
)

task2 = PythonOperator(
    task_id='generate_random_numbers',
    python_callable=generate_random_numbers,
    dag=dag
)
```

```

task3_par = PythonOperator(
    task_id='print_resultado_par',
    python_callable=imprimir_par,
    trigger_rule='none_failed',
    dag=dag
)

task3_impar = PythonOperator(
    task_id='print_resultado_impar',
    python_callable=imprimir_impar,
    trigger_rule='none_failed',
    dag=dag
)

task1 >> task2 >> [task3_par, task3_impar]

```

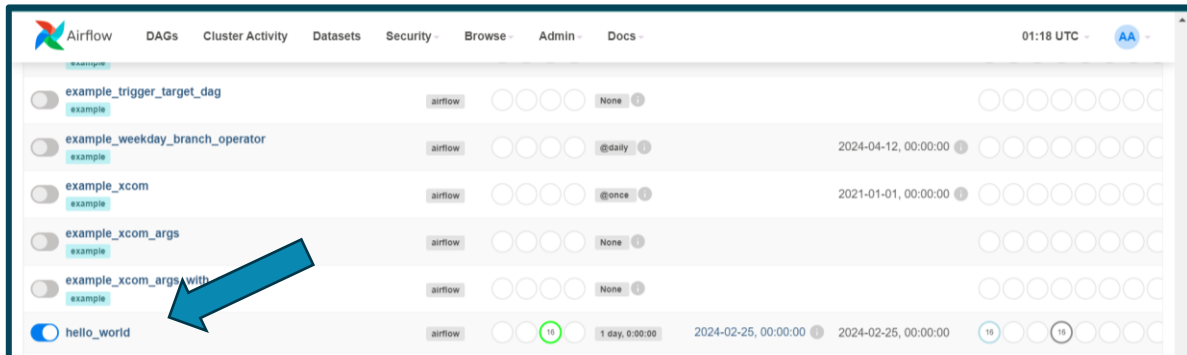
Descripción del programa:

El DAG se llama 'ejemplo_suma' y se programa para ejecutarse diariamente, cada 5 minutos, y en caso de haber saltado las tareas programadas (a cierta hora) estas se ignoraran solo realizando el ultimo debido y en adelante. Contiene dos tareas:

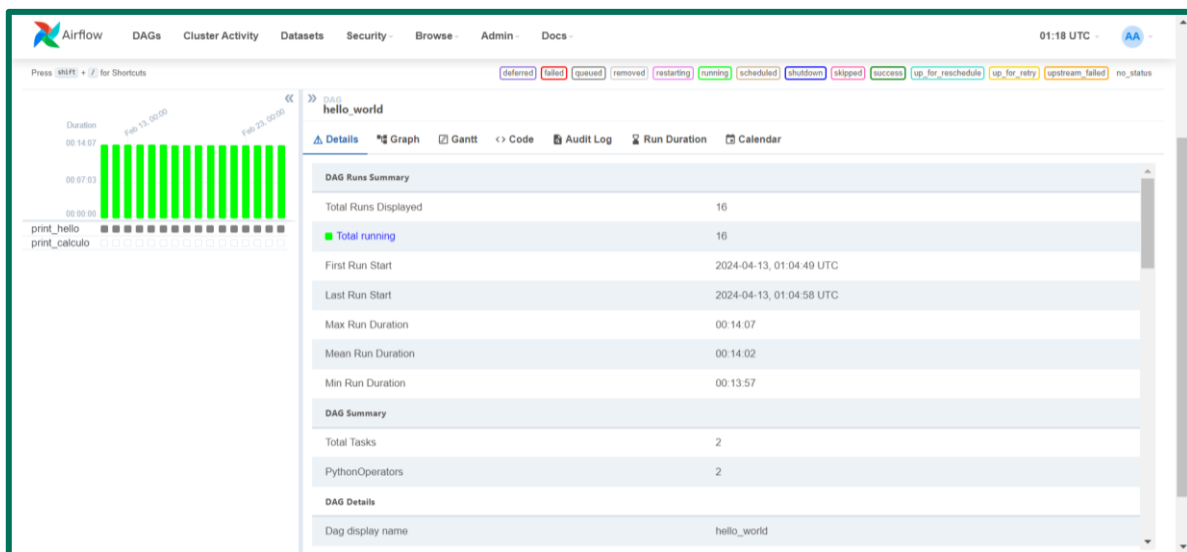
1. Imprimir en consola la cadena (Hello, World!).
2. Generar dos números aleatorios en un rango de números enteros entre 0 y 50, realizar una suma de ambos. El resultado es puesto en una condicional if-else, preguntando si este es par o impar. Dependiendo del resultado se mandará a llamar una tarea especifica.
3. Dependiendo si el resultado fue par o impar se ejecutará uno de dos tareas llamadas imprimir_par e imprimir_impar. Estos imprimen en consola el resultado de la comparación, terminando el proceso.

Capturas

Ejemplo 1:



This screenshot shows the Airflow web interface's 'DAGs' tab. A list of DAGs is displayed, including 'example_trigger_target_dag', 'example_weekday_branch_operator', 'example_xcom', 'example_xcom_args', 'example_xcom_args_with', and 'hello_world'. The 'hello_world' DAG is selected, indicated by a blue toggle switch and a blue arrow pointing to it. The interface includes a top navigation bar with links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The right side shows the current time as 01:18 UTC and a user profile icon.

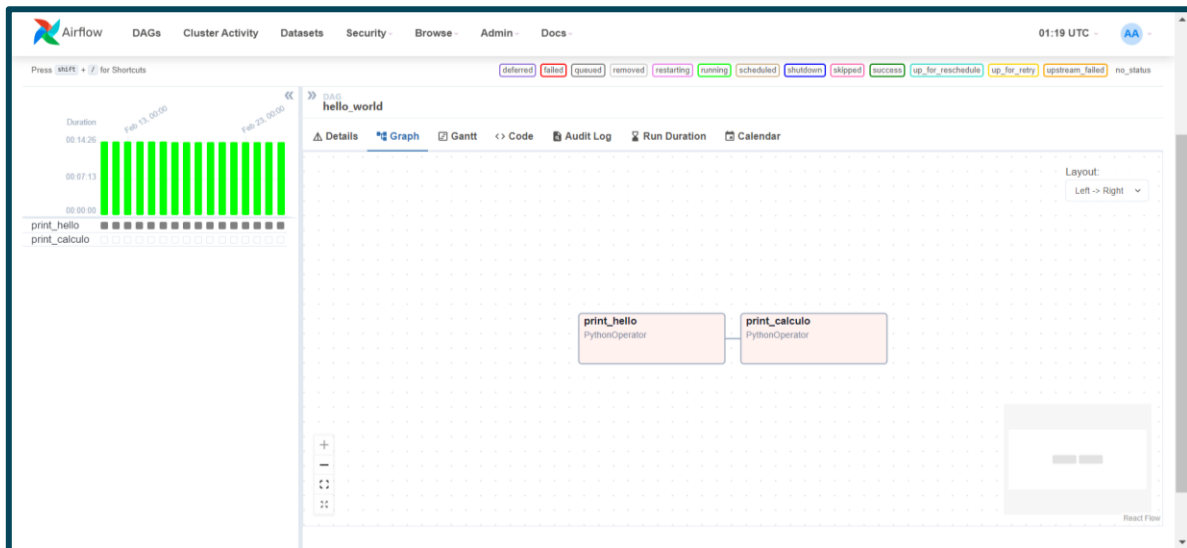


This screenshot shows the 'hello_world' DAG details in the Airflow web interface. The 'Details' tab is active, displaying a summary of the DAG's performance. On the left, a Gantt chart shows the execution timeline with green bars representing successful runs. The summary table provides the following data:

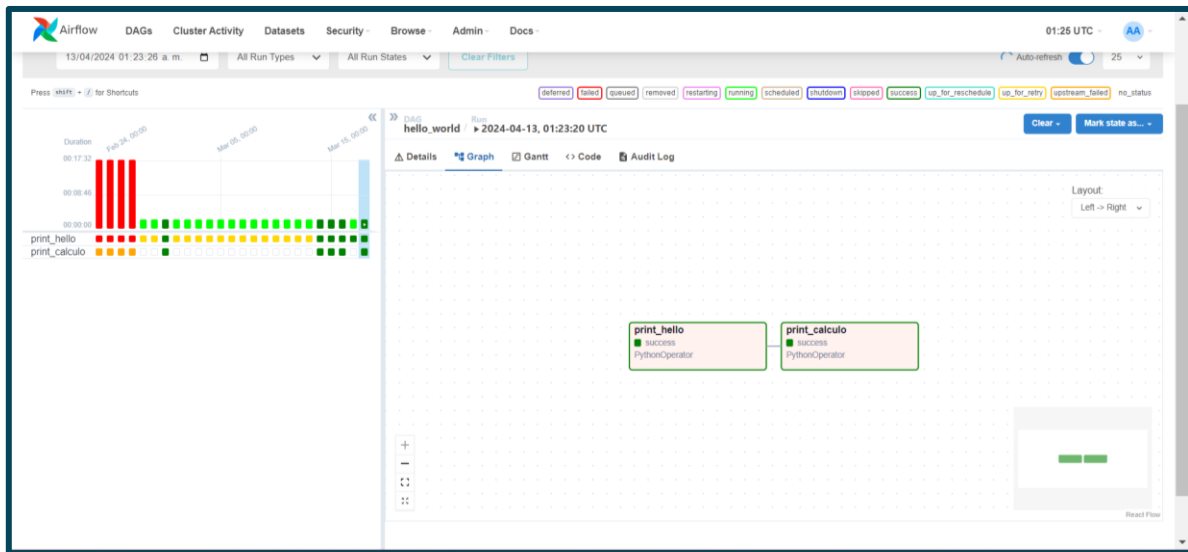
DAG Runs Summary	
Total Runs Displayed	16
Total running	16
First Run Start	2024-04-13, 01:04:49 UTC
Last Run Start	2024-04-13, 01:04:58 UTC
Max Run Duration	00:14:07
Mean Run Duration	00:14:02
Min Run Duration	00:13:57

DAG Summary	
Total Tasks	2
PythonOperators	2

DAG Details	
Dag display name	hello_world



This screenshot shows the 'hello_world' DAG details in the Airflow web interface, with the 'Graph' tab active. The graph displays two tasks: 'print_hello' and 'print_calculo', both identified as 'PythonOperator'. The tasks are connected by a flow line, indicating a sequential execution. The interface includes a top navigation bar and a right sidebar with a 'Layout' dropdown set to 'Left -> Right'. The Gantt chart on the left shows the execution timeline with green bars representing successful runs.



Ejemplo 2:

