

## ¿Qué es la programación basada en datos en Airflow?

El aspecto revolucionario de los conjuntos de datos es su capacidad para activar DAG en respuesta a los cambios. Imagine un escenario en el que está procesando registros de usuarios y aparece un nuevo **archivo de registro** disponible. Tradicionalmente, puede ejecutar su canalización según una programación, independientemente de si existen datos nuevos. Con la programación basada en datos, su canalización puede prepararse para ejecutarse solo cuando se actualiza el conjunto de datos (en este caso, el nuevo **archivo de registro**). Esto elimina el procesamiento redundante, lo que garantiza que sus recursos computacionales se asignen de manera óptima.

### *Entendamos cómo pueden ser útiles los conjuntos de datos.*

Los conjuntos de datos pueden ayudar a resolver problemas comunes. Por ejemplo, considere un equipo de ingeniería de datos con un DAG que crea un conjunto de datos y un equipo de análisis con un DAG que analiza el conjunto de datos. Al utilizar conjuntos de datos, el DAG de análisis de datos se ejecuta solo cuando el DAG del equipo de ingeniería de datos publica el conjunto de datos.

Los conjuntos de datos allanan el camino para una coordinación avanzada entre los DAG. Al configurar DAG para escuchar cambios en conjuntos de datos específicos, puede crear DAG maestros que orquesten el flujo de tareas de manera inteligente. Cuando una tarea de productor actualiza un conjunto de datos, el DAG maestro puede activar DAG de consumidor que dependen de estos datos nuevos. Esta transferencia orquestada garantiza que las tareas se ejecuten solo cuando se cumplan los requisitos previos, lo que fomenta un flujo de datos eficiente en sus canales.

## Comprender los conjuntos de datos en Airflow

Un conjunto de datos de Airflow es un sustituto de una agrupación lógica de datos. Los conjuntos de datos pueden actualizarse mediante tareas de **productor** ascendentes, y las actualizaciones de los conjuntos de datos contribuyen a programar los DAG de **consumidor** descendentes.

Un conjunto de datos se define mediante un **identificador uniforme de recursos (URI)** :

```
from airflow.datasets import Dataset

""" s3://dataset-bucket/example.csv: - Ruta del archivo que se utilizará para
la programación. """

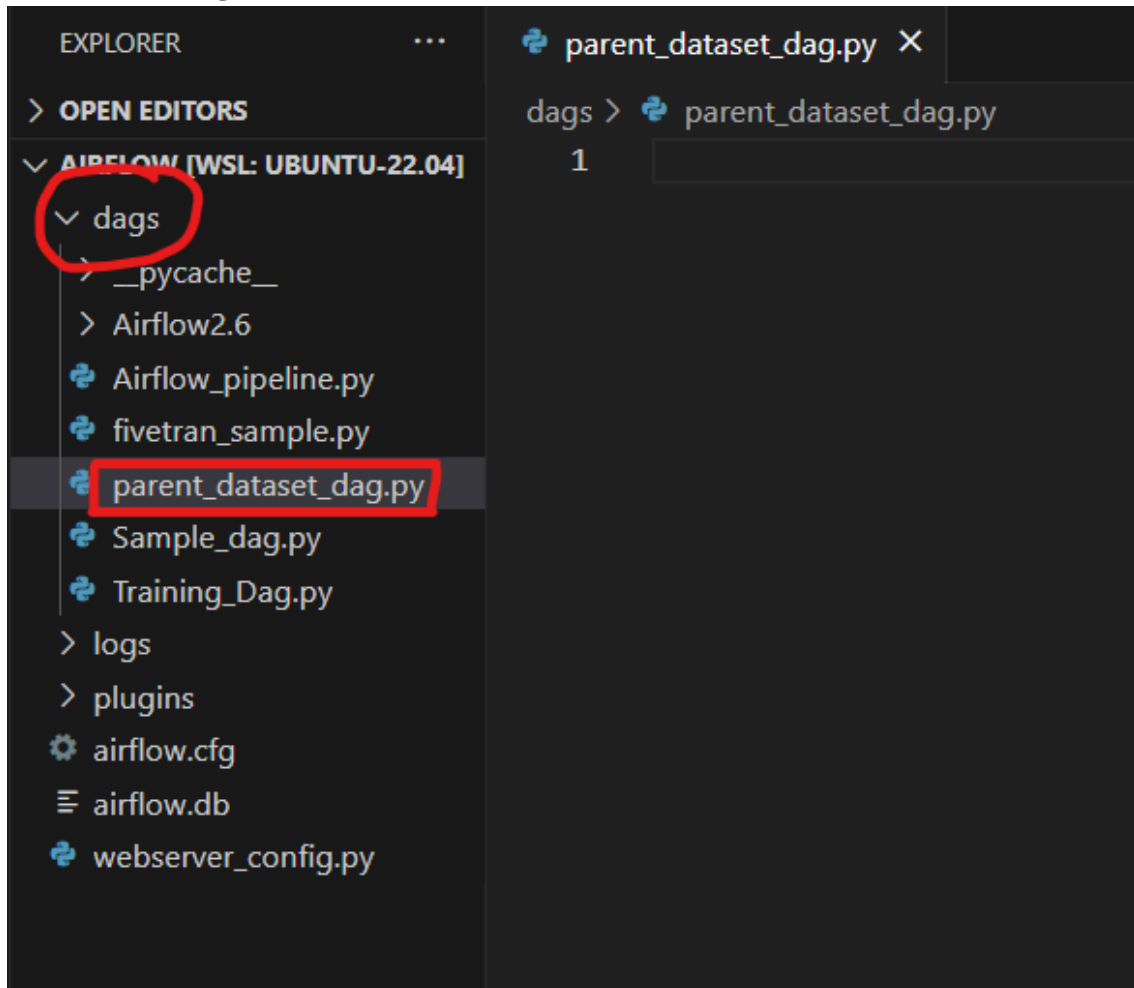
conjunto_datos_ejemplo = Conjunto de datos( "s3://conjunto-de-datos/ejemplo.csv"
)
```

Airflow no hace suposiciones sobre el contenido o la ubicación de los datos representados por el URI. Se trata como una cadena, por lo que cualquier uso de expresiones regulares (p. ej. `input_\d+.csv`) o patrones globales de archivos (p. ej. `input_2022*.csv`) como intento de crear múltiples conjuntos de datos a partir de una declaración no funcionará.

Debemos pasar la ruta específica del archivo que desea modificar, una vez que el DAG principal lo haya logrado. Si el archivo no existe, creará un archivo nuevo en la misma ubicación que pasó en el URI.

*Siga los pasos a continuación para programar su DAG utilizando conjuntos de datos.*

1. Creemos nuestro primer DAG, `parent_dataset_dag` en la carpeta DAG



Crear archivo `parent_dataset_dag.py`

## 2. Copie el siguiente código en `parent_dataset_dag.py` el archivo

```
from airflow import Dataset, DAG, task
from datetime import datetime, timedelta
import requests

# Variable to store the source file path. We can use Variables or secret
variables to store the location of the file for more security.
my_file = Dataset('/home/jorge/airflow/source_file.txt')

default_args = {
    'retries': 1,
    'start_date':datetime(2023, 1, 2),
}

with DAG(
    'Parent Dataset DAG',
    default_args=default_args,
    description='Parent DAG for Datasets',
    schedule_interval=None,
    tags=['Datasets'],
) as dag:

    @task
    def print_task():
        print('This is my first task')

''' OUTLETS - This parameter is passed to indicate that this
task will update the Dataset once completed.
```

```
'''
@task(outlets = [my_file])
def get_data_from_api():
    url = 'https://api.publicapis.org/entries' # public API

    response = requests.get(url).json()

    with open(my_file.uri, "a+") as file: # my_file.uri is the dataset name
and .uri is the extension for datasets
        file.write(str(response)) # Updating the Source File

    print('\nData from the API is written in the Source file\n')
```

En el código anterior: -

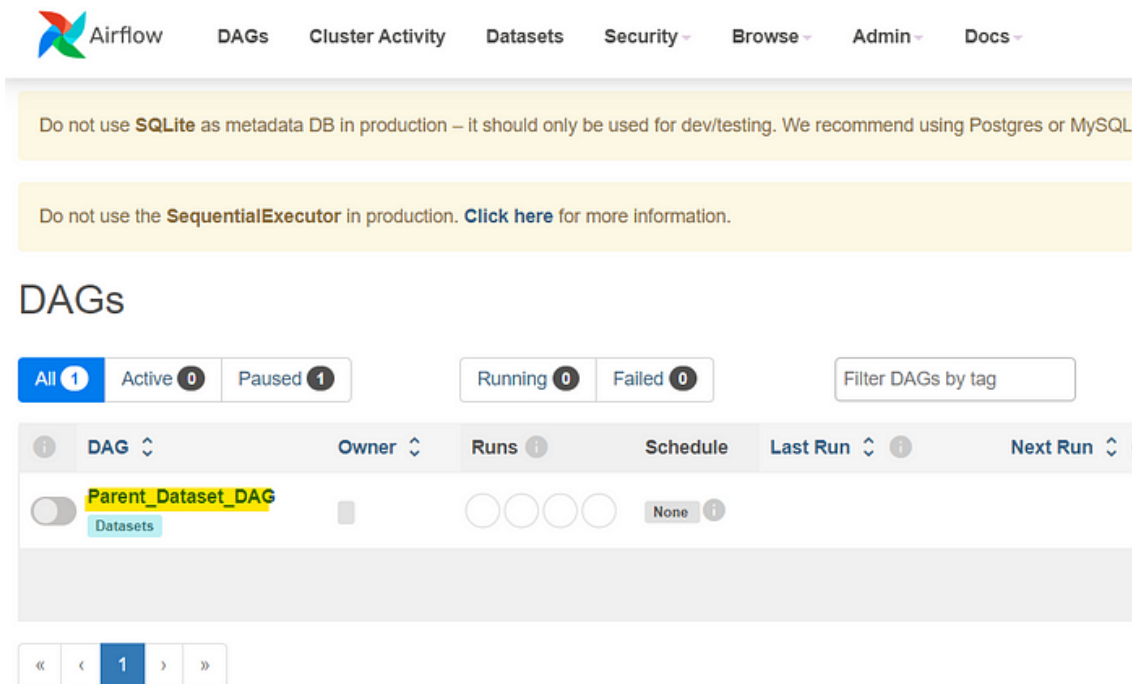
- `outlets`— Puede hacer referencia al conjunto de datos en una tarea pasándolo al parámetro de la tarea. Este operador indica que la tarea a la que está asignado actualizará el conjunto de datos una vez que se ejecute correctamente.
- `my_file.uri`— Es el nombre del conjunto de datos y `.uri` es la extensión del conjunto de datos.

El código DAG anterior recupera datos de una API pública, los convierte a JSON y los escribe en el archivo fuente. El `outlets` argumento que dimos indica que el conjunto de datos se actualizará después de la ejecución de la `get_data_from_api()` tarea.

Con el fin de programar más DAG dependientes, generamos un conjunto de datos utilizando el método Dataset y especificamos la ubicación de un archivo que se actualizará más adelante y se utilizará para programar otros DAG dependientes.

```
my_file = Dataset('/home/raj/airflow/source_file.txt')
```

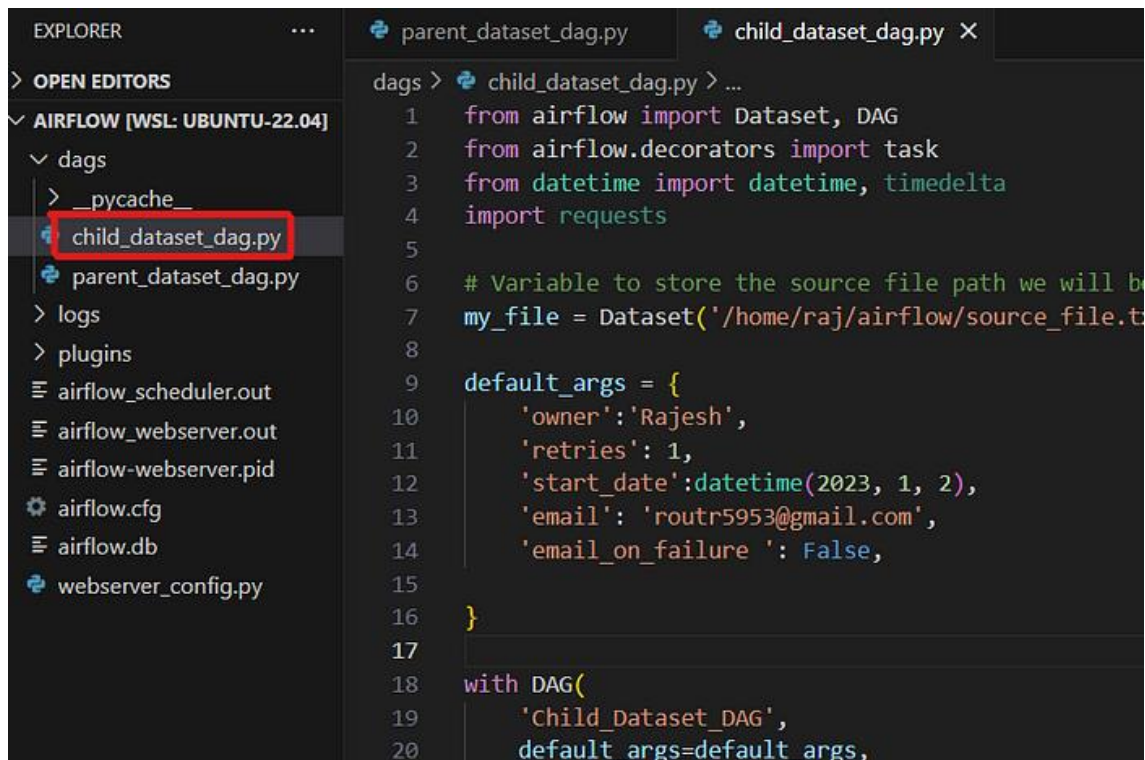
### 3. Guarde el archivo y actualice la interfaz de usuario de Airflow para ver el Dag.



The screenshot shows the Airflow web interface. At the top, there's a navigation bar with links: Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. Below the navigation bar, there are two yellow warning boxes. The first one says: "Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL..". The second one says: "Do not use the **SequentialExecutor** in production. [Click here](#) for more information." Below the warnings, the main heading is "DAGs". Underneath, there are filters for DAG status: "All 1", "Active 0", "Paused 1", "Running 0", and "Failed 0". There is also a "Filter DAGs by tag" input field. The main table lists DAGs with columns: DAG, Owner, Runs, Schedule, Last Run, and Next Run. The first DAG listed is "Parent\_Dataset\_DAG" with a toggle switch, a "Datasets" tag, and four empty run status circles. The "Schedule" column for this DAG shows "None". At the bottom, there is a pagination bar showing "1" of 1 items.

Conjunto de datos principal Dag

### 3. De manera similar, crearemos otro DAG: `Child_Dataset_dag`



Crear archivo **child\_dataset\_dag.py**

#### 4. Copie el siguiente código en child\_dataset\_dag.py el archivo

```
from airflow import Dataset, DAG
from airflow.decorators import task
from datetime import datetime, timedelta
import requests
```

```
# Variable to store the source file path which will be updated by the Parent
DAG.
```

```
my_file = Dataset('/home/raj/airflow/source_file.txt')
```

```
default_args = {
    'owner': 'Rajesh',
    'retries': 1,
    'start_date': datetime(2023, 1, 2),
```

```

'email': 'routr5953@gmail.com',
'email_on_failure ': False,

}

with DAG(
    'Child_Dataset_DAG',
    default_args=default_args,
    description='A Child DAG for Datasets',
    schedule = [my_file], # we are adding [my_file] dataset as the schedule
    for the Child DAG
    tags=['Datasets'],
) as dag:

    @task
    def print_task():
        print('This is my child task')

    @task()
    def read():

        with open(my_file.uri, "r+") as file:
            print(file.read())

```

- **Schedule : [my\_file]**— Programa todo según el conjunto de datos que hemos producido. El DAG solo se ejecutará en caso de que se cambie o modifique el conjunto de datos.



En el código anterior, hemos creado el mismo conjunto de datos con la misma ubicación de archivo, ya que programaremos el DAG secundario según las actualizaciones/modificaciones del conjunto de datos del DAG principal.

```
my_file = Dataset('/home/raj/airflow/source_file.txt')
```

En el parámetro de programación, pasamos el nombre del conjunto de datos.

```
with DAG(  
    'Child Dataset DAG',  
    default_args=default_args,  
    description='A Child DAG for Datasets',  
    schedule = [my_file], # we are adding [my_file] dataset as the schedule for the child DAG  
    tags=['Datasets'],  
) as dag:
```

horario = [mi\_archivo]

*Tanto en el DAG principal como en el secundario, debemos definir el conjunto de datos antes de usarlo como se muestra a continuación.*

```
from airflow import Dataset, DAG  
from airflow.decorators import task  
from datetime import datetime, timedelta  
import requests  
  
# Variable to store the source file path we will be updated by the Parent DAG.  
my_file = Dataset('/home/raj/airflow/source_file.txt')  
  
default_args = {  
    'owner': 'Rajesh',  
    'retries': 1,  
    'start_date': datetime(2023, 1, 2),  
    'email': 'routr5953@gmail.com',  
    'email_on_failure': False,
```

**6. Guarde el archivo y actualice la interfaz de usuario de Airflow para ver ambos DAG.**

Para `Child_Dataset_DAG`, la programación se basa en el conjunto de datos como se muestra a continuación.

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. [Click here](#) for more information.

Do not use the **SequentialExecutor** in production. [Click here](#) for more information.

## DAGs

All **2** Active **2** Paused **0** Running **0** Failed **0** Filter DAGs by tag Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
<b>Child_Dataset_DAG</b> Datasets	Rajesh	0	Dataset		On /home/raj/airflow/source_file.txt	
<b>Parent_Dataset_DAG</b> Datasets	Rajesh	0	None			

1

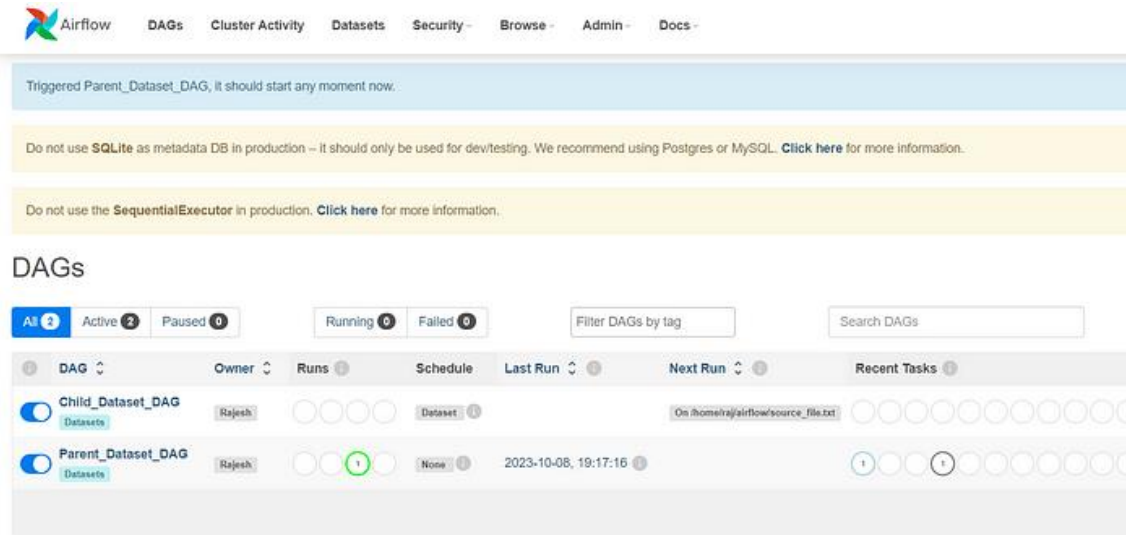
La programación del DAG secundario está configurada en Conjunto de datos

Child\_Dataset\_DAG se ejecutará después de Parent\_Dataset\_DAG haber modificado el archivo/conjunto de datos. Podemos definir un horario para el Parent DAG en función de nuestras necesidades. Por el momento lo tengo configurado en **Ninguno**.

```
with DAG(
    'Parent Dataset DAG',
    default_args=default_args,
    description='Parent DAG for Datasets',
    schedule_interval=None,
    tags=['Datasets'],
) as dag:
```

Programación DAG principal establecida en Ninguno

## 7. Ahora activemos el DAG principal y veamos cómo ejecuta el DAG secundario.



Trigged Parent\_Dataset\_DAG, it should start any moment now.

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. [Click here](#) for more information.

Do not use the **SequentialExecutor** in production. [Click here](#) for more information.

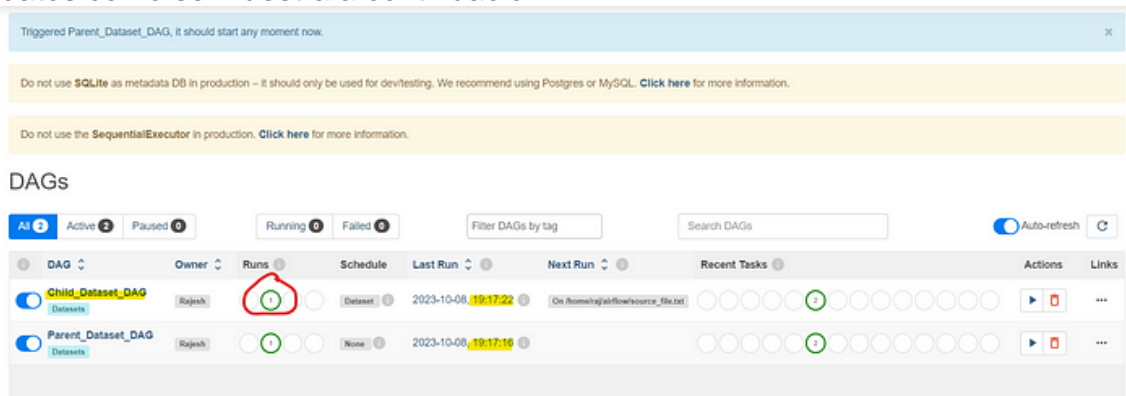
### DAGs

All 2 Active 2 Paused 0 Running 0 Failed 0 Filter DAGs by tag Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
Child_Dataset_DAG	Rajesh	0	Dataset		On /home/raj/airflow/source_file.txt	
Parent_Dataset_DAG	Rajesh	1	None	2023-10-08, 19:17:16		1

DAG principal activado a las **19:17:16**

El DAG secundario también se activó debido a la modificación del conjunto de datos como se muestra a continuación.



Trigged Parent\_Dataset\_DAG, it should start any moment now.

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. [Click here](#) for more information.

Do not use the **SequentialExecutor** in production. [Click here](#) for more information.

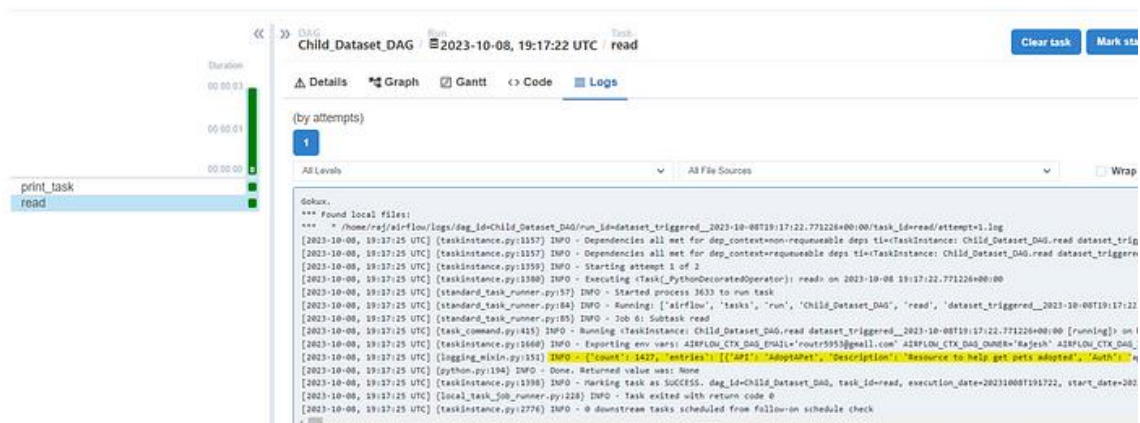
### DAGs

All 2 Active 2 Paused 0 Running 0 Failed 0 Filter DAGs by tag Search DAGs Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
Child_Dataset_DAG	Rajesh	1	Dataset	2023-10-08, 19:17:22	On /home/raj/airflow/source_file.txt	2		
Parent_Dataset_DAG	Rajesh	1	None	2023-10-08, 19:17:16		2		

DAG infantil ejecutado a las 19:17:22

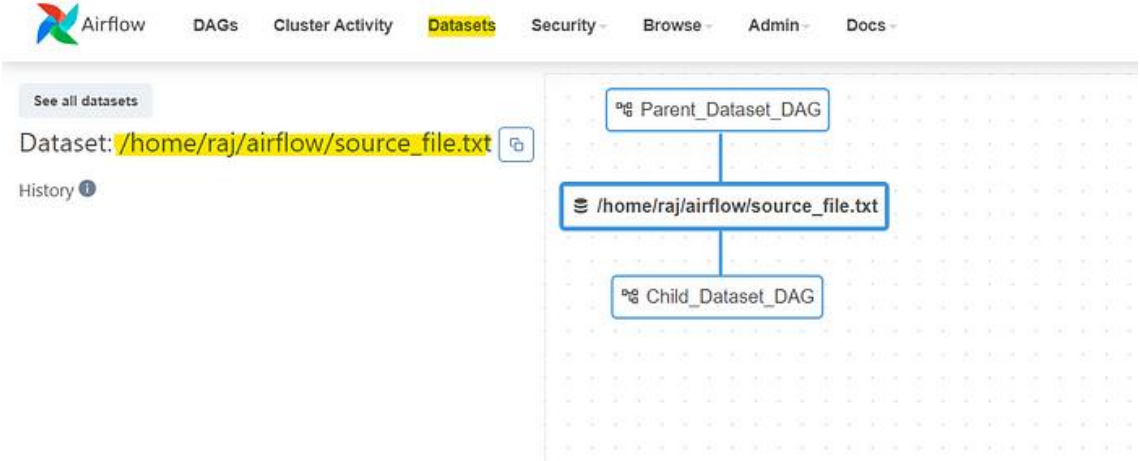
read()Para Child DAG, también podemos ver los registros de la tarea.



```
2023-10-08, 19:17:25 UTC [taskinstance.py:1557] INFO - Dependencies all met for dep_context=non-requeueable deps ti=TaskInstance: Child_Dataset_DAG.read dataset_trig
2023-10-08, 19:17:25 UTC [taskinstance.py:1557] INFO - Dependencies all met for dep_context=queueable deps ti=TaskInstance: Child_Dataset_DAG.read dataset_trigge
2023-10-08, 19:17:25 UTC [taskinstance.py:1558] INFO - Starting attempt 1 of 1
2023-10-08, 19:17:25 UTC [taskinstance.py:1558] INFO - Executing <TaskPythonDecoratedOperator>: read on 2023-10-08 19:17:22.771226+00:00
2023-10-08, 19:17:25 UTC [standard_task_runner.py:157] INFO - Started process 3633 to run task
2023-10-08, 19:17:25 UTC [standard_task_runner.py:184] INFO - Running: ['airflow', 'tasks', 'run', 'Child_Dataset_DAG', 'read', 'dataset_triggered_2023-10-08T19:17:22
2023-10-08, 19:17:25 UTC [standard_task_runner.py:185] INFO - Job 6: Subtask read
2023-10-08, 19:17:25 UTC [task_command_runner.py:145] INFO - Running <TaskInstance: Child_Dataset_DAG.read dataset_triggered_2023-10-08T19:17:22.771226+00:00 [running]> on 1
2023-10-08, 19:17:25 UTC [taskinstance.py:1568] INFO - Exporting env vars: AIRFLOW_CTX_DAG_EMAIL='routr595@gmail.com' AIRFLOW_CTX_DAG_OWNER='Rajesh' AIRFLOW_CTX_DAG_
2023-10-08, 19:17:25 UTC [logging_mixin.py:133] INFO - {'count': 1427, 'entries': [['API', 'AdoptAPet', 'Description: Resource to help get pets adopted', 'Auth:']]
2023-10-08, 19:17:25 UTC [python.py:194] INFO - Done. Returned value was: None
2023-10-08, 19:17:25 UTC [taskinstance.py:1398] INFO - Marking task as SUCCESS. dag_id=Child_Dataset_DAG, task_id=read, execution_date=20231008T191722, start_date=202
2023-10-08, 19:17:25 UTC [local_task_job_runner.py:228] INFO - Task exited with return code 0
2023-10-08, 19:17:25 UTC [taskinstance.py:12776] INFO - 0 downstream tasks scheduled from follow-on schedule check
```

Leyó los datos de la API del archivo fuente que habíamos creado al principio.

Para verificar la dependencia de los DAG, haga clic en **Datasets** en la interfaz de usuario de Airflow.



Dependencia de DAG con conjunto de datos

En la figura anterior, podemos ver que Parent Dag se ejecutará primero y modificará el conjunto de datos después de una ejecución exitosa. La modificación del archivo dará lugar a la ejecución del Child DAG.

Al aprovechar los conjuntos de datos, hemos aprendido cómo orquestar canales de datos que respondan dinámicamente a los cambios en la disponibilidad de los datos, reduciendo las ejecuciones innecesarias y maximizando la utilización de los recursos. Este cambio de rutinas estáticas a DAG dinámicos e inteligentes abre un nuevo ámbito de posibilidades en el campo de la ingeniería de datos.