

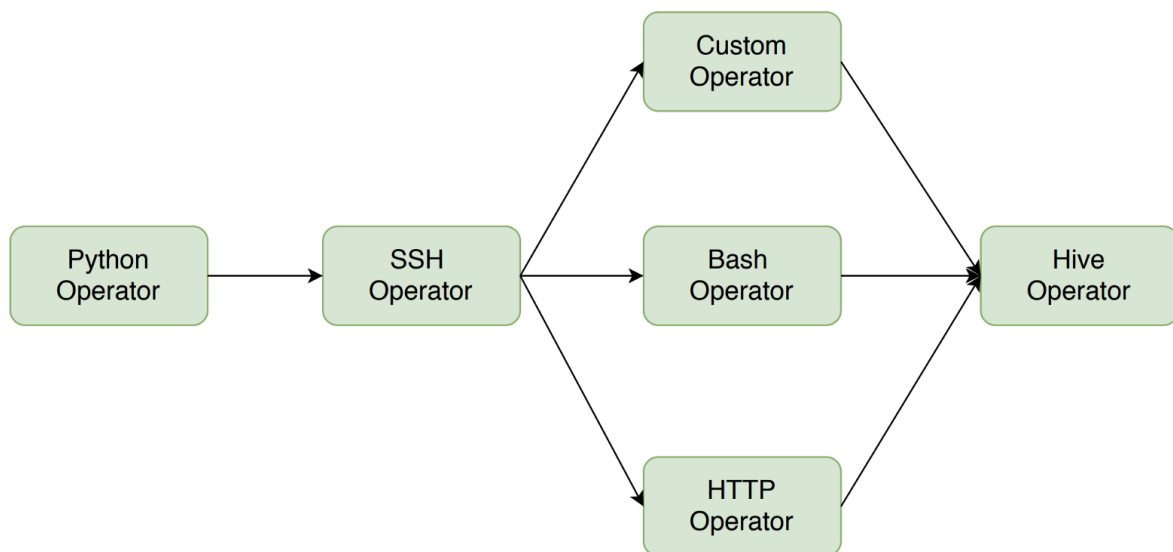
## Minería de Datos II

### Clase 12 - Flujos de Trabajo

#### -Workflows

Un proyecto de Big Data implica realizar múltiples tareas en diferentes sistemas en un orden específico.

Es por este motivo que existe la necesidad de contar con orquestadores de flujos de trabajo que permitan automatizar el movimiento y la transformación de los datos.

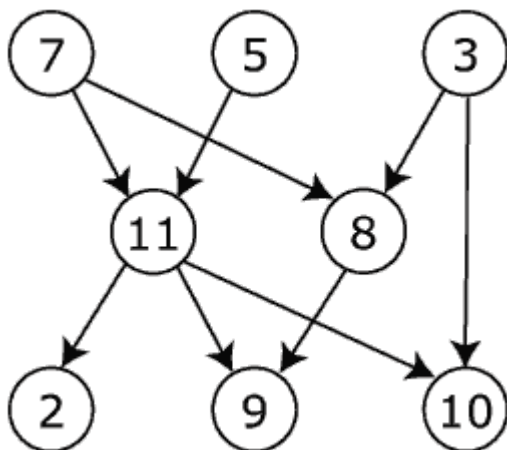


#### -DAG

##### Directed Acyclic Graph

Es una representación conceptual de una serie de actividades

- Dirigido: cada relación entre nodos tiene un sentido único
- Acíclico: no hay ningún camino que nos permita volver al nodo inicial



## -Notación Cron

```
# |----- minute (0 - 59)
# |----- hour (0 - 23)
# |----- day of the month (1 - 31)
# |----- month (1 - 12)
# |----- day of the week (0 - 6) (Sunday to Saturday;
# |                          7 is also Sunday on some systems)
# |
# * * * * * <command to execute>
```

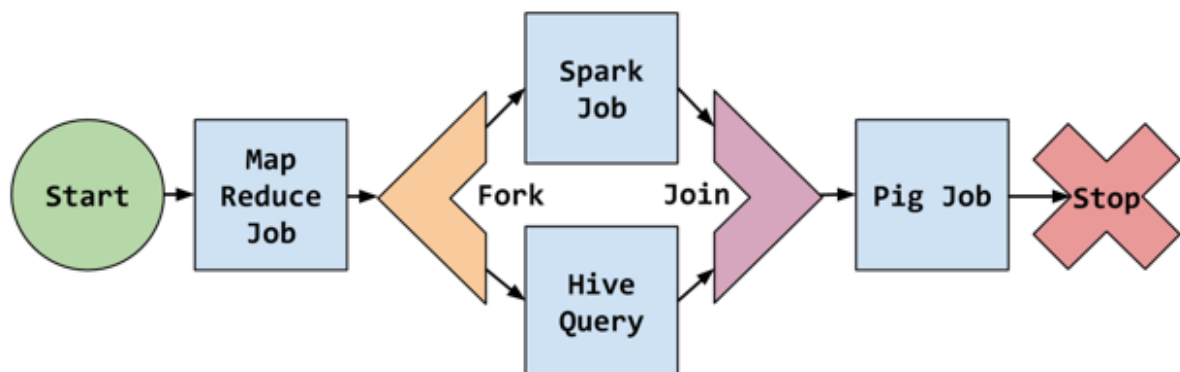
## -Enlaces sugeridos:

- <https://crontab.guru/>
- <https://blog.desdelinux.net/cron-crontab-explicados/>

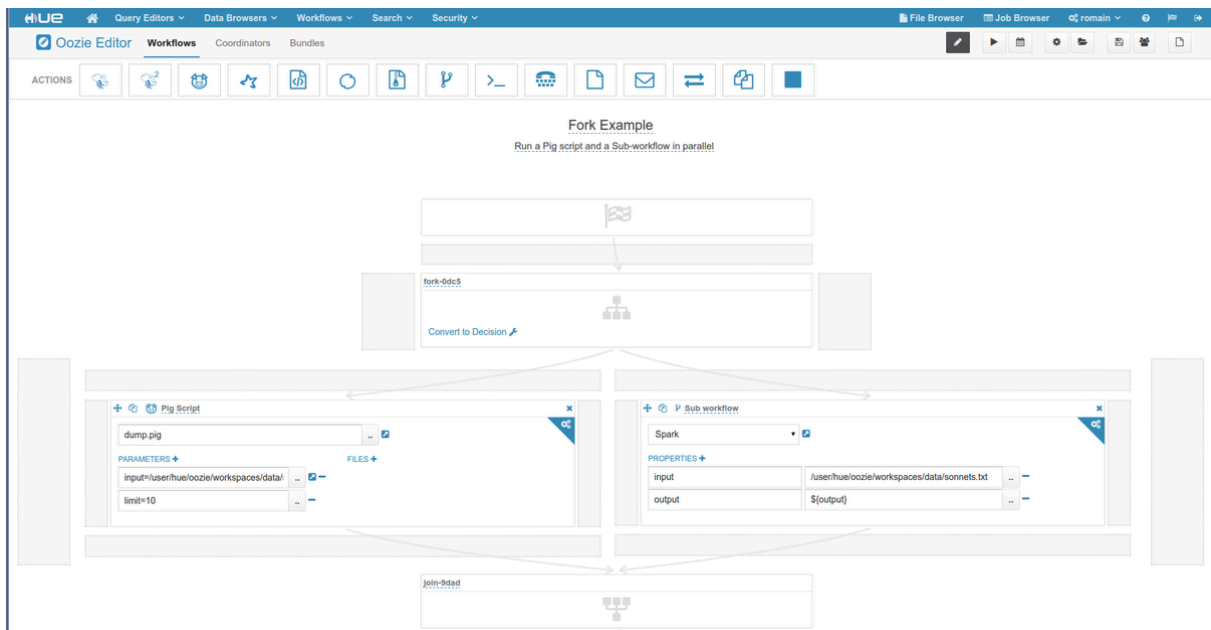
## -Oozie

Es un sistema de programación de workflows incluido en distribuciones de Hadoop  
Los flujos de trabajo en Oozie están definidos como una colección de tareas representadas en un DAG.

Acciones soportadas: MapReduce, Shell, Pig, Hive, Spark, Java, entre otros.



## -Oozie en Hue

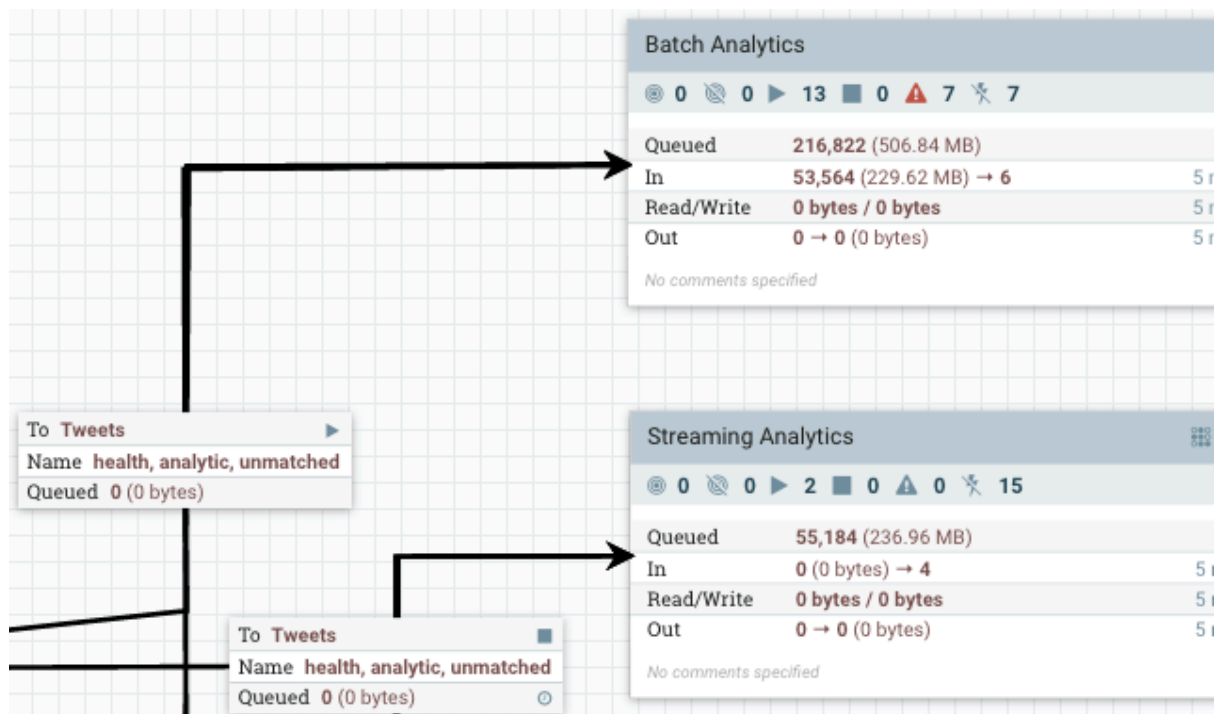


-Nifi

Es una herramienta desarrollada por la NSA (actualmente es un proyecto de alto nivel de la ASF) que permite automatizar flujos de datos entre sistemas.

Posee una interfaz web que permite crear flujos sin necesidad de escribir código.

Brinda funcionalidades de seguridad, monitoreo y linaje de datos en movimiento.



-Processors

## Add Processor

Source: all groups ▼

Displaying 11 of 219

Filter

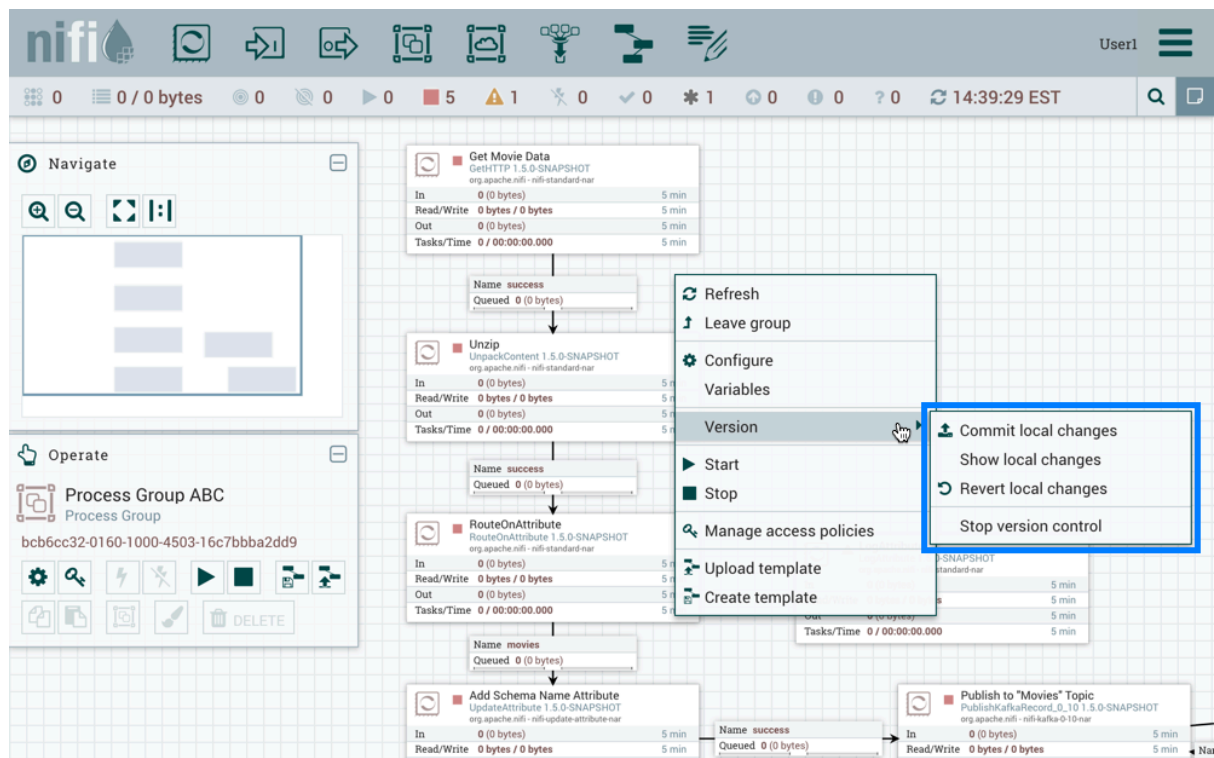
| Type      | Version | Tags  |
|-----------|---------|---|
| FetchFTP  | 1.2.0   | input, ftp, get, fetch, retrieve, files, source, remote, ingest           |
| FetchFile | 1.2.0   | ingress, input, restricted, get, files, source, local, filesystem, ingest |
| FetchSFTP | 1.2.0   | input, get, fetch, retrieve, files, sftp, source, remote, ingest          |
| GetFTP    | 1.2.0   | input, FTP, get, fetch, retrieve, files, source, remote, ingest           |
| GetFile   | 1.2.0   | ingress, input, restricted, get, files, source, local, filesystem, ingest |
| GetHDFS   | 1.2.0   | restricted, get, fetch, HDFS, hadoop, source, filesystem, ingest          |
| GetSFTP   | 1.2.0   | input, get, fetch, retrieve, files, sftp, source, remote, ingest          |
| ListFTP   | 1.2.0   | input, ftp, files, source, list, remote, ingest                           |
| ListFile  | 1.2.0   | file, get, source, list, filesystem, ingest                               |
| ListHDFS  | 1.2.0   | get, HDFS, hadoop, source, list, filesystem, ingest                       |
| ListSFTP  | 1.2.0   | input, files, sftp, source, list, remote, ingest                          |

amazon attributes  
avro aws consume  
csv database fetch  
files get hadoop  
ingest input insert  
json listen logs  
message put remote  
restricted source sql  
text update

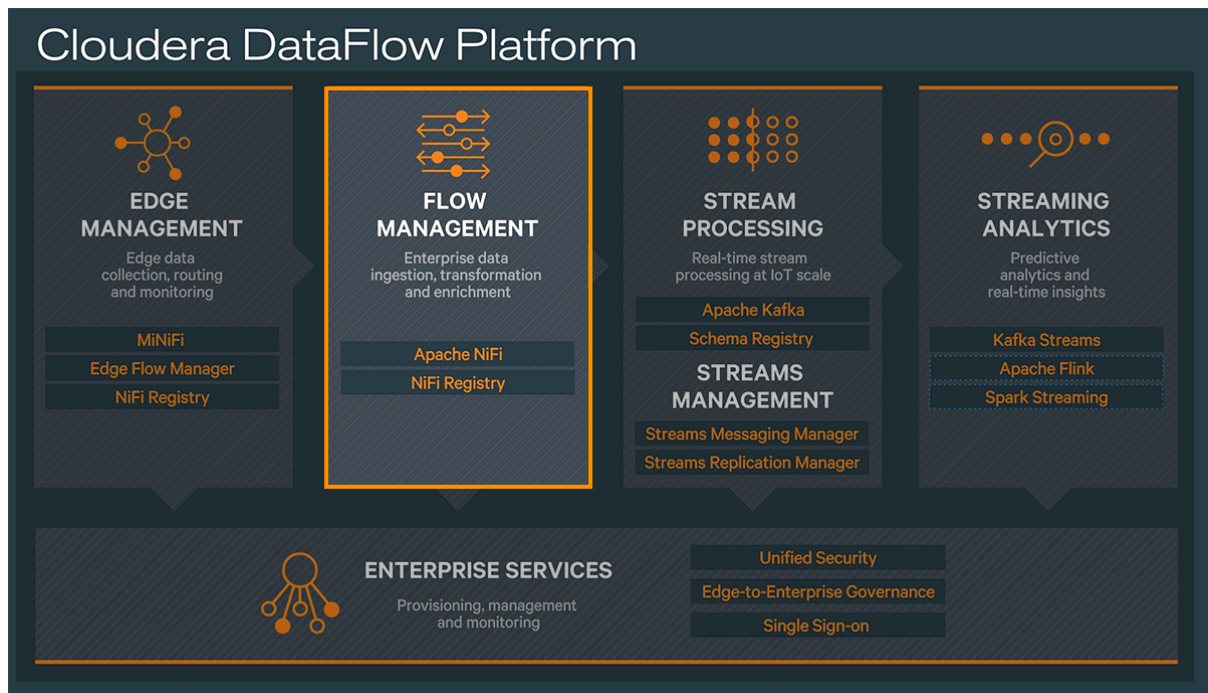
**FetchFTP 1.2.0** org.apache.nifi - nifi-standard-nar

Fetches the content of a file from a remote SFTP server and overwrites the contents of an incoming FlowFile with the content of the remote file.

-Ejemplo



-Cloudera Enterprise

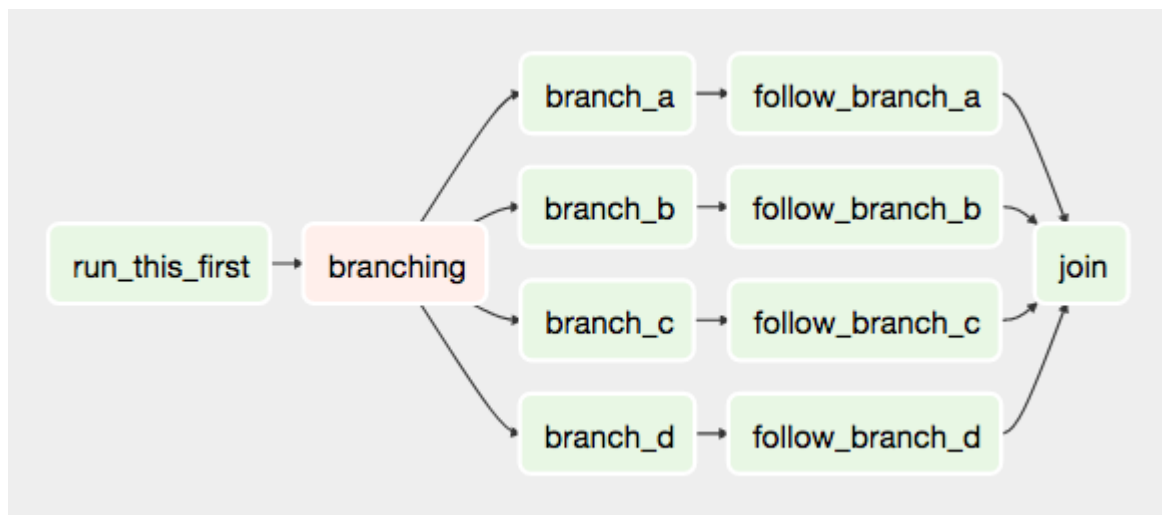


-Airflow

Es una plataforma de gestión de flujos de trabajo de código abierto desarrollada por Airbnb (actualmente es un proyecto ASF).

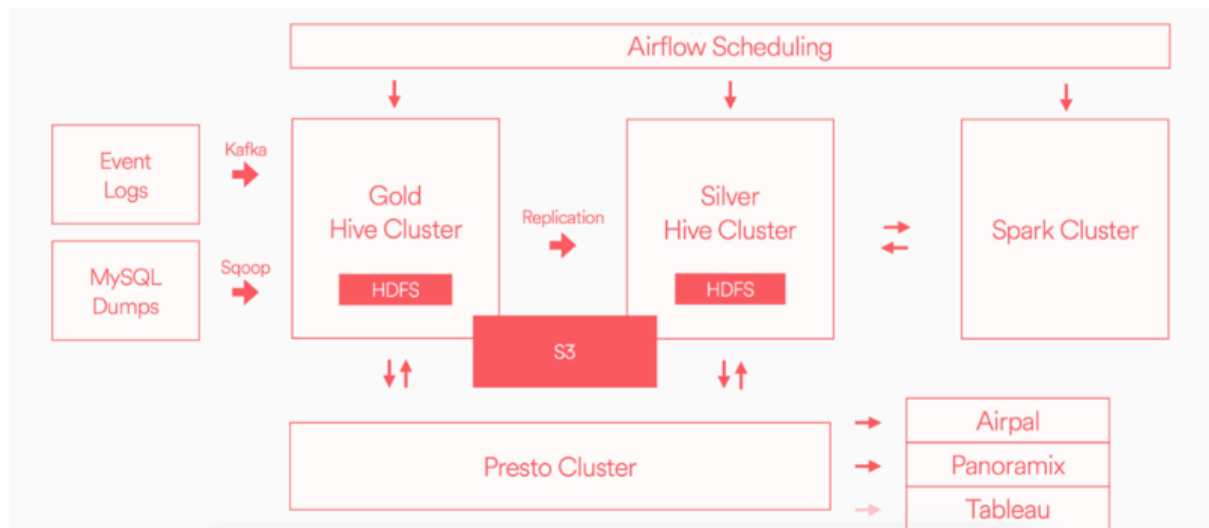
Las tareas y dependencias se representan como DAG's definidos en scripts Python.

Los DAG's pueden ser programados para ejecutarse en un horario predefinido o en función de la ocurrencia de eventos.



<https://airflow.apache.org/docs/stable/concepts.html>

-Airbnb



-Google Cloud Composer - Código

```

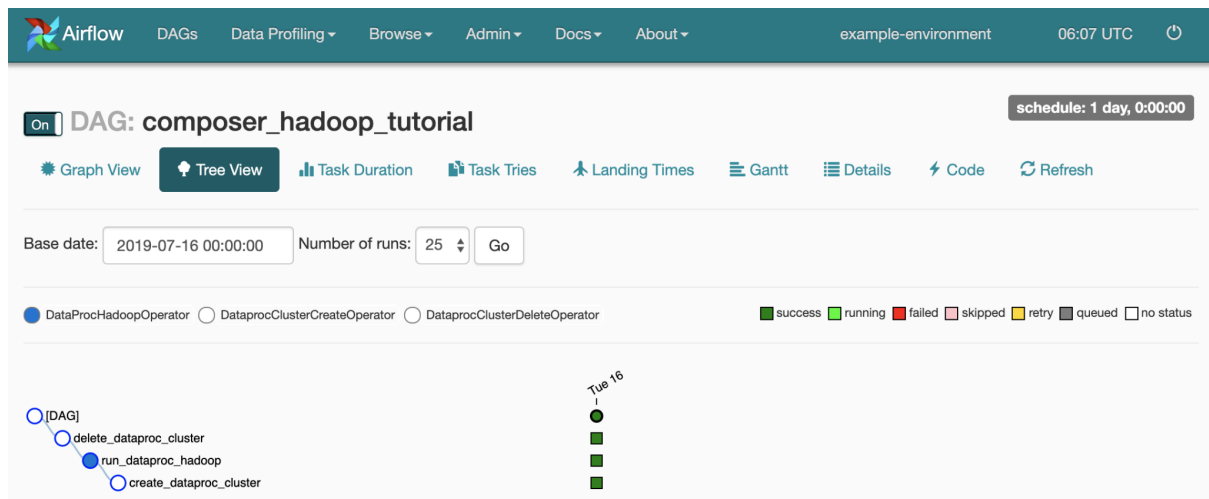
# Run the Hadoop wordcount example installed on the Cloud Dataproc cluster
# master node.
run_dataproc_hadoop = dataproc_operator.DataProcHadoopOperator(
    task_id='run_dataproc_hadoop',
    main_jar=WORDCOUNT_JAR,
    cluster_name='composer-hadoop-tutorial-cluster-{{ ds_nodash }}',
    arguments=wordcount_args)

# Delete Cloud Dataproc cluster.
delete_dataproc_cluster = dataproc_operator.DataprocClusterDeleteOperator(
    task_id='delete_dataproc_cluster',
    cluster_name='composer-hadoop-tutorial-cluster-{{ ds_nodash }}',
    # Setting trigger_rule to ALL_DONE causes the cluster to be deleted
    # even if the Dataproc job fails.
    trigger_rule=trigger_rule.TriggerRule.ALL_DONE)

# Define DAG dependencies.
create_dataproc_cluster >> run_dataproc_hadoop >> delete_dataproc_cluster
  
```

<https://cloud.google.com/composer/docs/tutorials/hadoop-wordcount-job>

-Google Cloud Composer - DAG



## -Alternativas



## -Links de referencia

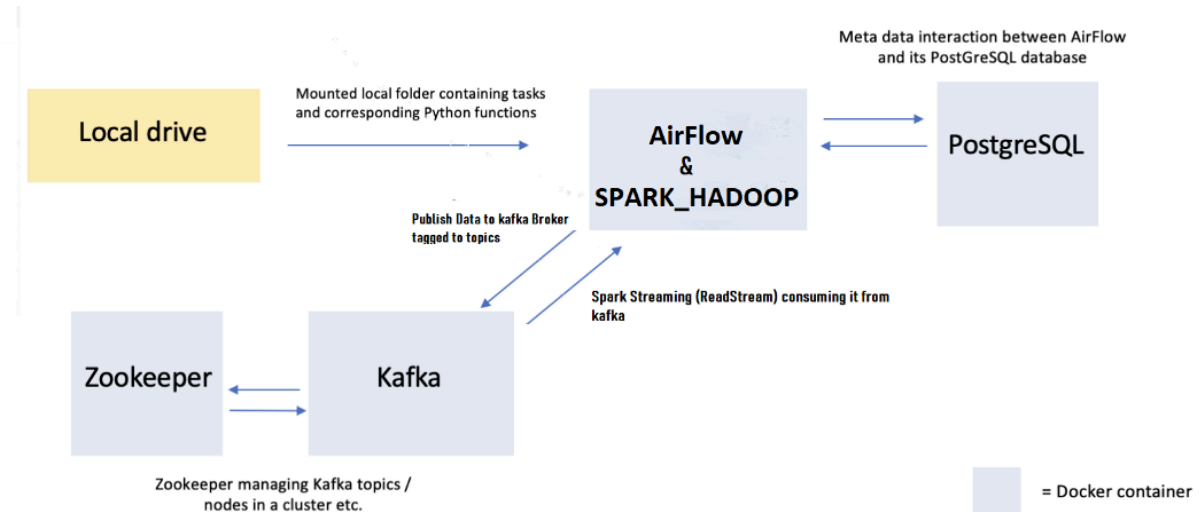
- Nifi: <https://nifi.apache.org/docs/nifi-docs/html/getting-started.html>
- Oozie: [https://oozie.apache.org/docs/5.2.0/DG\\_Overview.html](https://oozie.apache.org/docs/5.2.0/DG_Overview.html)
- Airflow: <https://airflow.apache.org/docs/stable/>

## Laboratorio:

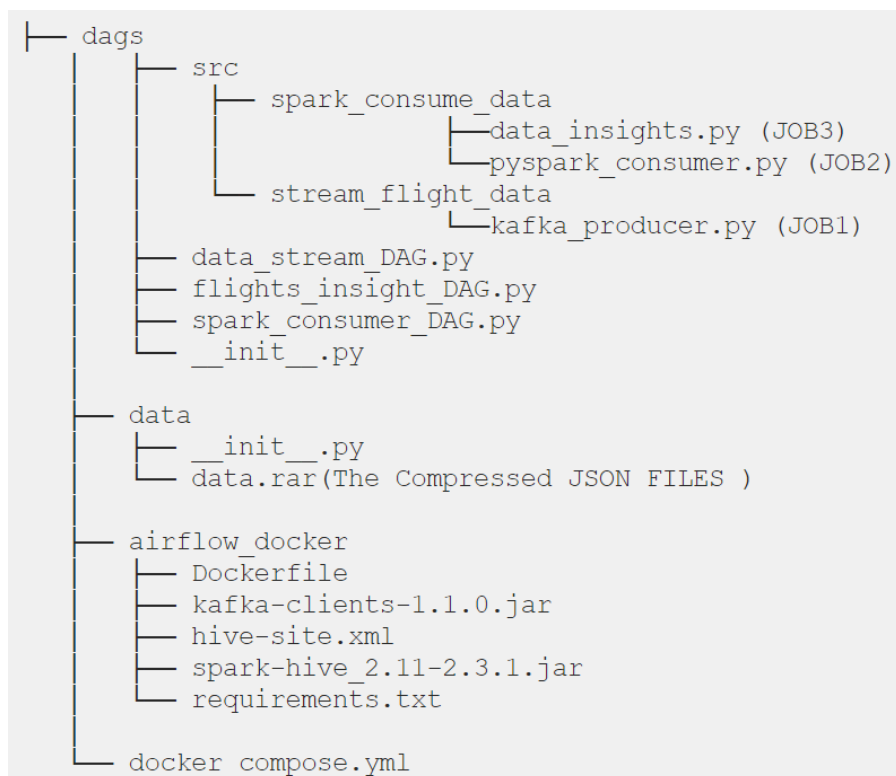
Vamos a utilizar un ambiente generado con Docker para ejecutar un workflow en Airflow.

Utilizar los siguientes links de referencia para configurar el ambiente:

- <https://medium.com/@rose4earn/docker-compose-ing-kafka-airflow-spark-b2ea66993c50>
- [https://github.com/KumarRoshandot/AirFlow\\_Kafka\\_Spark\\_Docker/tree/master/Project\\_Flight\\_Docker2](https://github.com/KumarRoshandot/AirFlow_Kafka_Spark_Docker/tree/master/Project_Flight_Docker2)

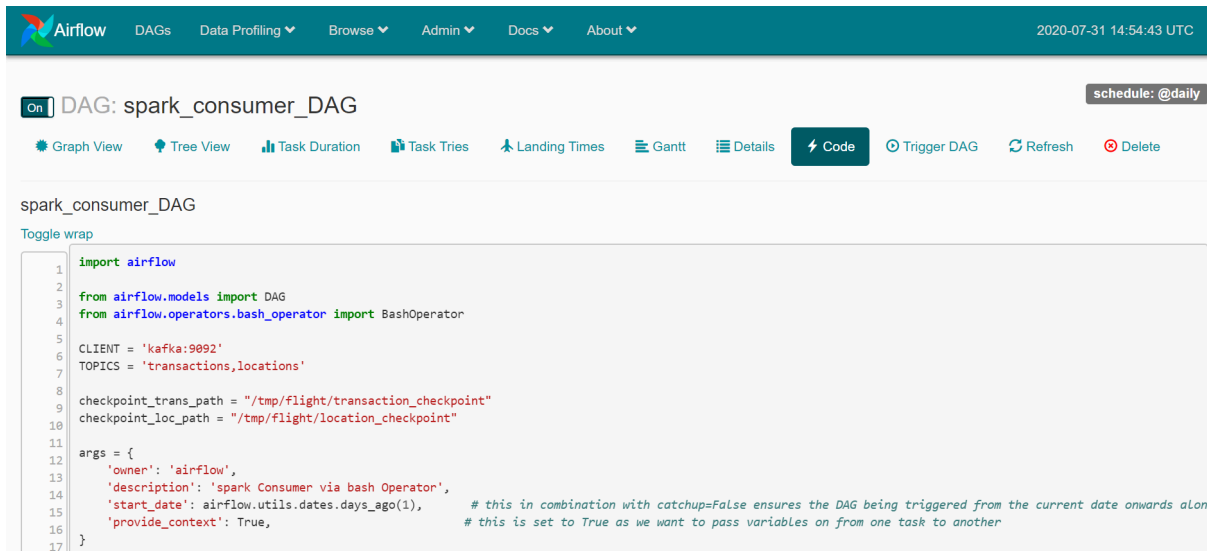


## Estructura del Proyecto



## DAG



The screenshot shows the Apache Airflow web interface. At the top, there's a navigation bar with links for DAGs, Data Profiling, Browse, Admin, Docs, and About. The current view is for a DAG named 'spark\_consumer\_DAG'. Below the navigation bar, there are tabs for Graph View, Tree View, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, Trigger DAG, Refresh, and Delete. The 'Code' tab is selected, showing the DAG's Python code. The code defines a DAG with a single task named 'spark\_consumer\_DAG' using the 'BashOperator'. The task's command is a shell script that interacts with Kafka and writes data to HDFS. The code includes comments explaining the purpose of the DAG and the variables used.

```
1 import airflow
2
3 from airflow.models import DAG
4 from airflow.operators.bash_operator import BashOperator
5
6 CLIENT = 'kafka:9092'
7 TOPICS = 'transactions,locations'
8
9 checkpoint_trans_path = "/tmp/flight/transaction_checkpoint"
10 checkpoint_loc_path = "/tmp/flight/location_checkpoint"
11
12 args = {
13     'owner': 'airflow',
14     'description': 'spark Consumer via bash Operator',
15     'start_date': airflow.utils.dates.days_ago(1), # this in combination with catchup=False ensures the DAG being triggered from the current date onwards along
16     'provide_context': True, # this is set to True as we want to pass variables on from one task to another
17 }
```

## Proyecto Integrador:

### Carga incremental con Spark

[https://github.com/lopezdar222/herramientas\\_big\\_data/tree/main?tab=readme-ov-file#7-carga-incremental-con-spark](https://github.com/lopezdar222/herramientas_big_data/tree/main?tab=readme-ov-file#7-carga-incremental-con-spark)

Ahora resta evaluar qué sucede cuando en los sistemas fuente, se genere más dato, es decir, siguiendo los datos de esta práctica, qué pasa cuando se carguen más ventas. Se debería tomar las novedades e ingestar en el modelo existente cada día, de modo que la tabla venta, irá creciendo en cantidad de registro de manera diaria. Para este fin, se provee un script en spark que realiza la generación de nuevas ventas, de manera aleatoria, para poder crear una situación, donde se cuenta con novedades para la tabla de venta. El script "Paso06\_GeneracionVentasNuevasPorDia.py" utiliza los datasets provistos en la carpeta "Datasets\data\_nvo" para generar las novedades de forma automática. Revisar la variable "fecha\_nvo" que contiene la fecha para la que se quiere generar información, como tenemos datos hasta el año 2020, la fecha de ejemplo tomada es '2021-01-01'. Es necesario entonces generar, un script tal que tome las novedades en csv, y las cargue al modelo.

```
$ sudo docker exec -it spark-master /spark/bin/spark-submit --master
spark://spark-master:7077 /home/Paso06_GeneracionVentasNuevasPorDia.py
```

Supongamos que tenemos nuestro script, y ahora se quiere programar su ejecución:

```
$ /spark/bin/spark-submit --master spark://spark-master:7077 Paso06_IncrementalVentas.py
```

Con crontab, para que ejecute cada día a las 5 AM:

```
$ crontab -e
5 0 * * * /home/CargaIncremental.sh
$ crontab -l
```