

# Curso Data Engineer: Creando el pipeline de datos

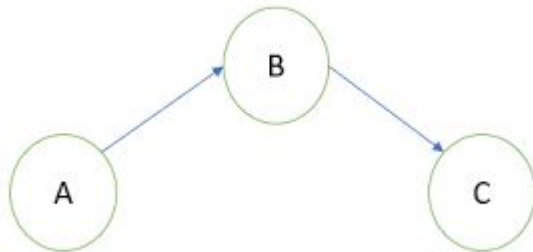
MÓDULO E



# Data pipelines

# Data pipelines

Concepto que captura la idea del flujo de datos, desde un proceso de estado a otro.



# Data pipelines

- ¿Cómo se representan?

Directed Acyclic Graphs (DAGS)

- ¿Qué etapas existen?

Ingesta, transformación, Almacenamiento y Análisis

- ¿Qué tipos de pipelines existen?

DW pipeline, streaming pipeline, ML pipeline



# Directed Acyclic Graphs (DAGS)



- ¿Qué son?

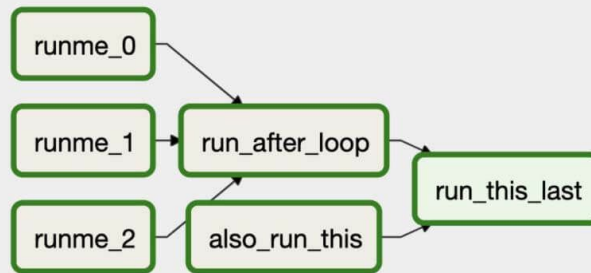
Son nodos unidos por aristas

- ¿Para qué se utilizan?

Se utilizan para modelar un pipeline de datos

- ¿Cómo se utilizan?

Se modela el flujo de datos de mi proyecto de Big Data



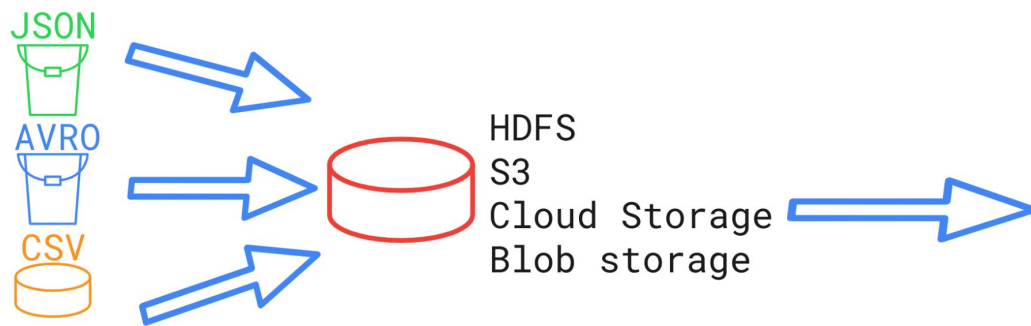
# Etapas

- Ingesta
- Transformación
- Almacenamiento
- Análisis



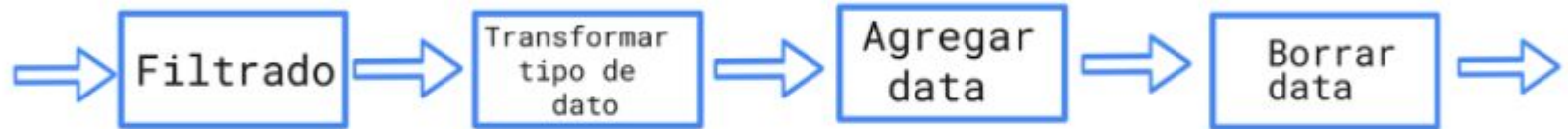
# Etapas

## Ingesta



# Etapas

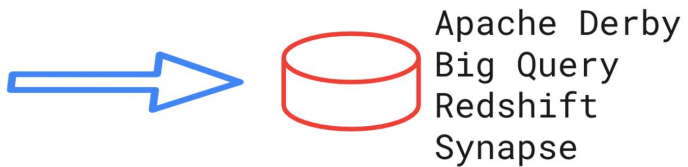
## Transformación





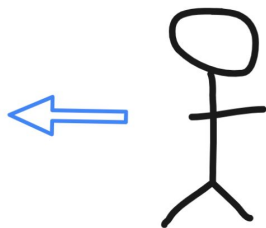
# Etapas

## Almacenamiento



# Etapas

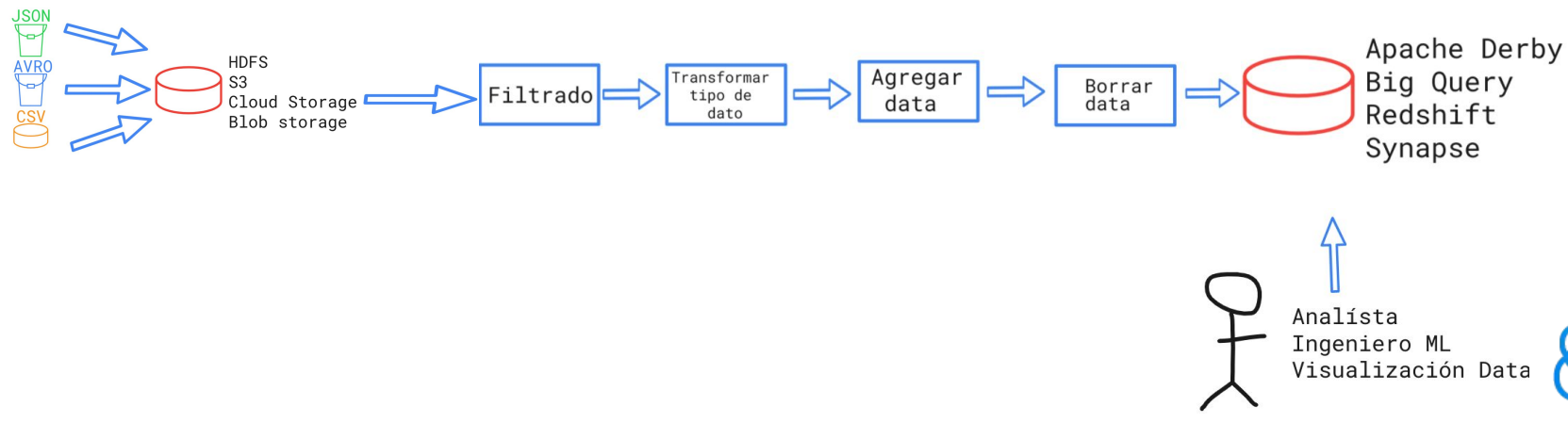
## Análisis



Analísta  
Ingeniero ML  
Visualización Data



# Etapas



# Tipos de data pipelines

## Data Warehouse pipeline

- Data desde diferente fuentes
- Data desnormalizada
- Data para contestar eficientemente consultas analíticas



# Tipos de data pipelines

## Data streaming pipeline

- Flujo continuo de data
- Pueden provenir de sensores, logs, monitoreo
- Puede almacenar cierto periodo de tiempo y luego tomar un promedio



# Resumen

- Data Pipelines
- Directed Acyclic Graphs (DAGS)
  - Etapas: Ingesta, transformación, Almacenamiento y Análisis
  - Tipos; DW pipeline, streaming pipeline, ML pipeline



# Orquestación

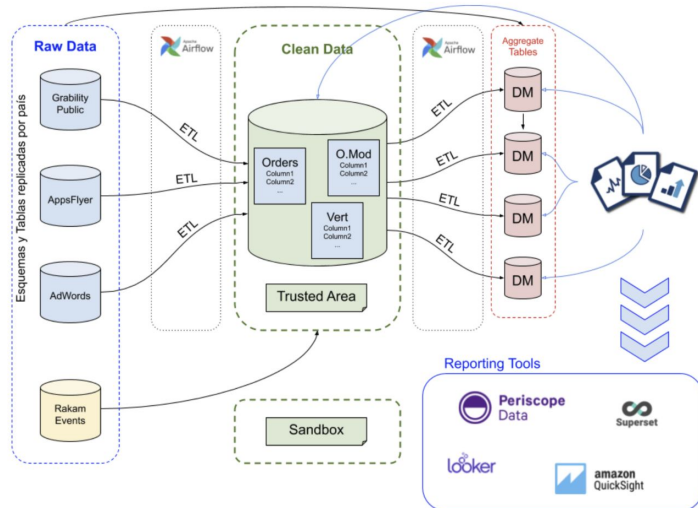


¿Qué es la  
orquestración?



# Orquestación

La orquestación de datos automatiza los procesos relacionados con la gestión de datos, como ingerir datos de múltiples fuentes, combinarlos y prepararlos para el análisis de datos. Adicionalmente realiza supervisión de las tareas.





¿Cómo funciona?

# Data orchestration

- En la orquestación de datos se encuentra la creación de pipelines de datos y flujos de trabajo para mover datos de una ubicación a otra mientras se coordina la combinación, verificación y almacenamiento de esos datos para que sean útiles.
- Debido a la cantidad y complejidad de pipelines se comenzaron a automatizar.
- Tiempo atrás se ejecutaban con cron (utilidad de Linux para ejecutar tareas)
- La orquestación de datos moderna implica definir las tareas básicas dentro de un sistema de datos y ejecutar lo que se conoce como un grafo acíclico dirigido (DAG) que ilustra todas las tareas relevantes y su relación entre sí.



Utilidad

# Utilidad data orchestration

- Limpiar, transformar, organizar y publicar data en un datawarehouse
- Visualización de métricas empresariales
- Aplicar reglas para involucrar a usuarios a través de campañas de correo electrónico
- Mantenimiento de la infraestructura de datos como web scraping
- Correr una tarea de TensorFlow para entrenar un modelo de machine learning



# Apache Airflow

# Apache Airflow

Apache Airflow es una herramienta de tipo workflow manager, donde gestiona, monitorea y planifica flujos de trabajo, usada como orquestador de servicios.

Se usa para automatizar trabajos dividiéndolos en subtarefas.

Permite su planificación y monitorización desde una herramienta centralizada.



# Apache Airflow - DAGS

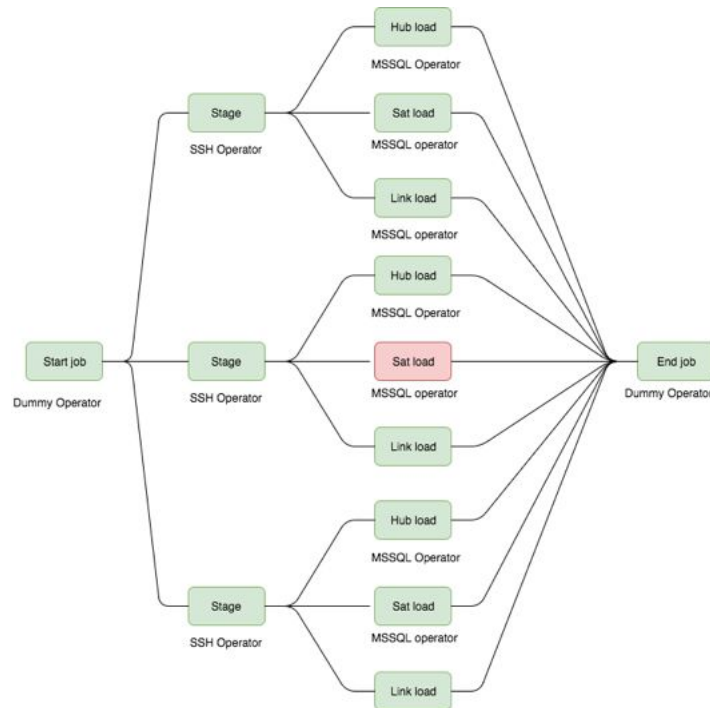
REAL \* IA PARA UN MUNDO

Airflow modela estos workflows o conjuntos de tareas como grafos acíclicos dirigidos, DAG (Directed Acyclic Graphs). Estos grafos tienen la peculiaridad de que cumplen dos condiciones:

**Son dirigidos:** las uniones entre los diferentes nodos tienen un sentido

**Son acíclicos:** no podemos formar ciclos y por lo tanto volver a un nodo por el que ya hayamos pasado

La creación de DAGS se realiza a través de Python.





# Apache Airflow - Arquitectura



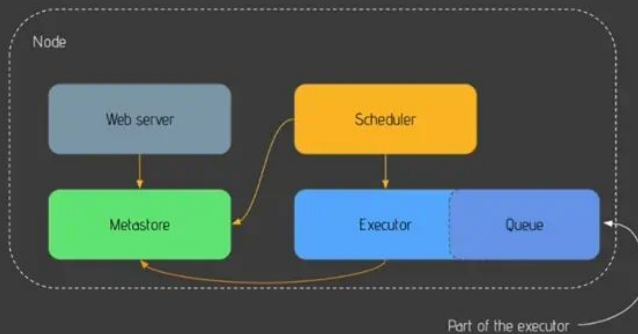
Según el tipo de arquitectura (un nodo o varios nodos) y los tipos de ejecutor que se elija, los componentes principales tendrán un lugar diferente en los nodos:

- Web server – provee la UI Web
- Scheduler – agenda las tareas, pipelines y workers
- Metastore – almacena la metadata de las tareas (usualmente usa PostgreSQL)

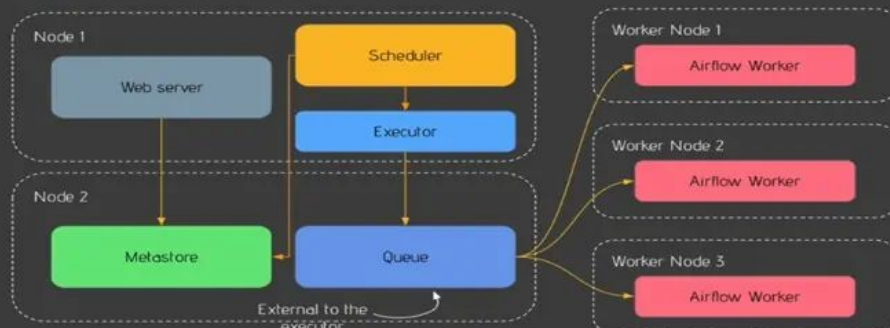
# Apache Airflow - Arquitectura

- Executor – es la definición de cómo las tareas serán ejecutadas
- Queue – define el orden en el cual las tareas serán ejecutadas. Es parte de el ejecutor en la arquitectura de un solo nodo
- Worker – este es realmente el proceso cuando la tarea es ejecutada

## One Node Architecture



## Multi Nodes Architecture (Celery)



# Apache Airflow - parámetros

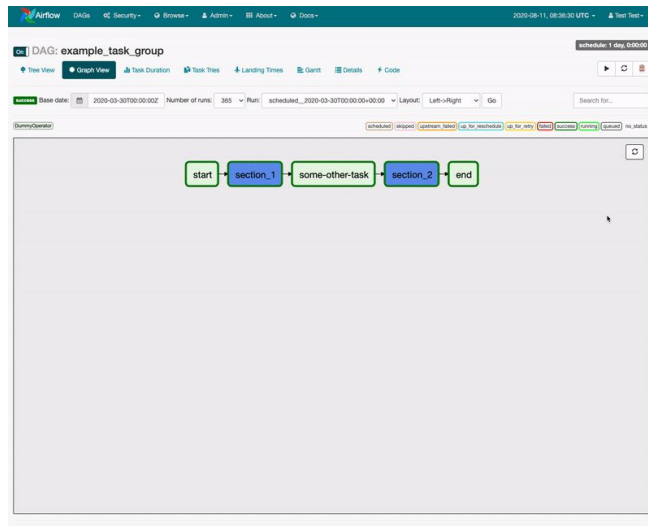
Los principales parámetros de Airflow son:

- **Catchup (si está en TRUE)** – Los DAG tienen una variable `schedule_interval` que determina el intervalo en el que se ejecutan. Con el parámetro `catchup`, se programará una ejecución de DAG para cualquier intervalo que no se haya ejecutado desde la última ejecución programada regularmente
- **Parallelism (default es 32)** – determina el máximo número de tareas que pueden correrse en paralelo para toda la instancia de Airflow
- **DAG\_concurrency (default es 16)** – máximo número de tareas que pueden correrse concurrentemente por DAG
- **MAX\_ACTIVE\_RUNS\_PER\_DAG (default es 16)** – máximo número de DAG corriendo por DAG

# Apache Airflow - V2.0


REAL \* IA PARA UN MUNDO

- Soporte full de API
- Permite utilizar almacenamiento externo para pasar información entre tareas (XCOM)
- Permite actualizar el versionado de los operadores independientemente de la versión de Airflow
- Agrupación de tareas



# Apache Airflow - Dags

REAL \* IA PARA  
UN MUNDO

 Airflow

DAGs

Security

Browse

Admin

Docs

20:25 UTC

AA

DAGs

All 33

Active 0

Paused 33

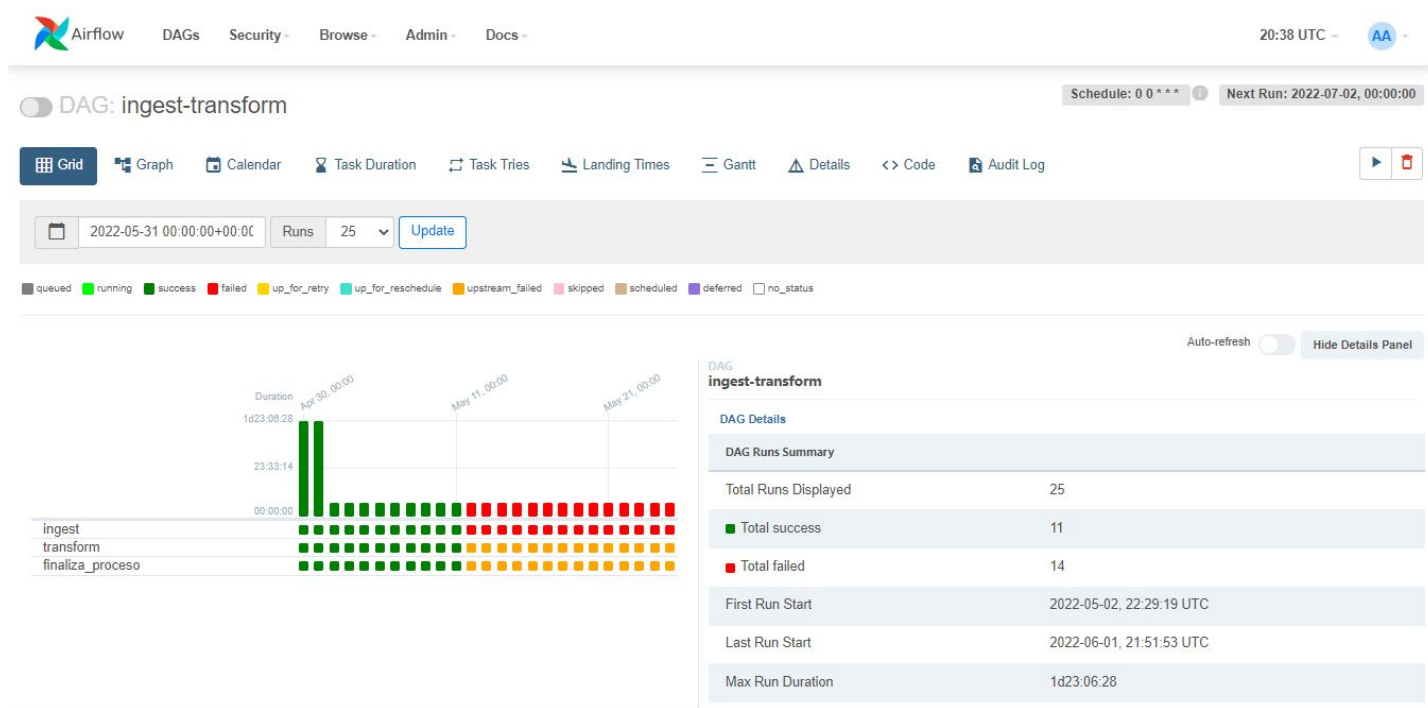
Filter DAGs by tag

Search DAGs

<div><div></div></div> DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<div><div></div><div>example-DAG</div><div>ingesttransform</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>0 0 * * *</div>		2022-06-26, 00:00:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_bash_operator</div><div>exampleexample2</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>0 0 * * *</div>		2022-06-26, 00:00:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_branch_datetime_operator_2</div><div>example</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>@daily</div>		2022-06-26, 00:00:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_branch_dop_operator_v3</div><div>example</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>* * ? * * *</div>		2022-06-27, 23:42:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_branch_labels</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>@daily</div>		2022-06-27, 00:00:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_branch_operator</div><div>exampleexample2</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>@daily</div>		2022-06-26, 00:00:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_branch_python_operator_decorator</div><div>exampleexample2</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>@daily</div>		2022-06-27, 00:00:00	<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_complex</div><div>exampleexample2example3</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>None</div>			<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>
<div><div></div><div>example_dag_decorator</div><div>example</div></div>	airflow	<div><div></div><div></div><div></div></div>	<div>None</div>			<div><div></div><div></div><div></div></div>	<div><div></div><div></div></div>	<div>...</div>

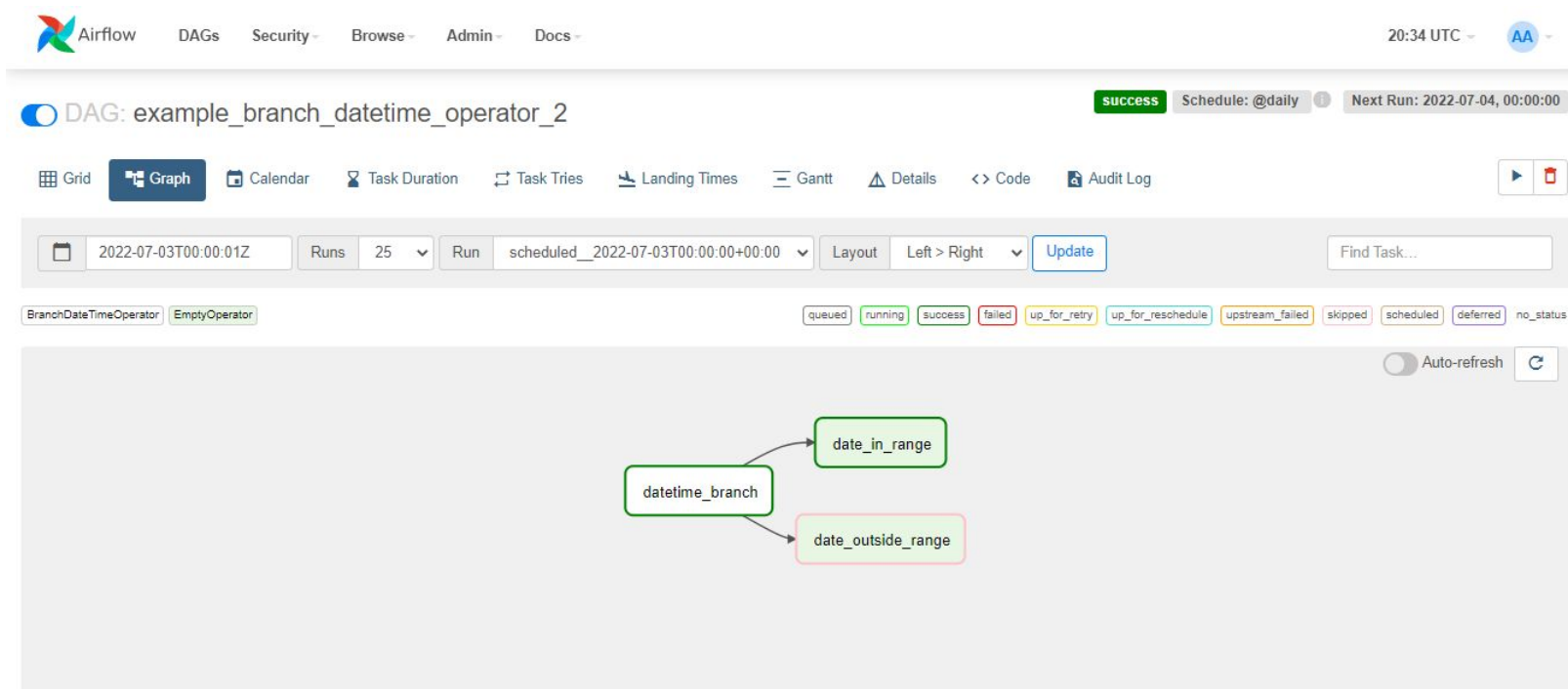
# Apache Airflow - Vista grilla

REAL \* IA PARA UN MUNDO



# Apache Airflow - Vista gráfica

REAL \* IA PARA UN MUNDO



# Apache Airflow - Vista calendario

REAL \* IA PARA UN MUNDO



Airflow

DAGs

Security

Browse

Admin

Docs

20:35 UTC



DAG: example\_branch\_datetime\_operator\_2

Schedule: @daily

Next Run: 2022-07-04, 00:00:00

Grid

Graph

Calendar

Task Duration

Task Tries

Landing Times

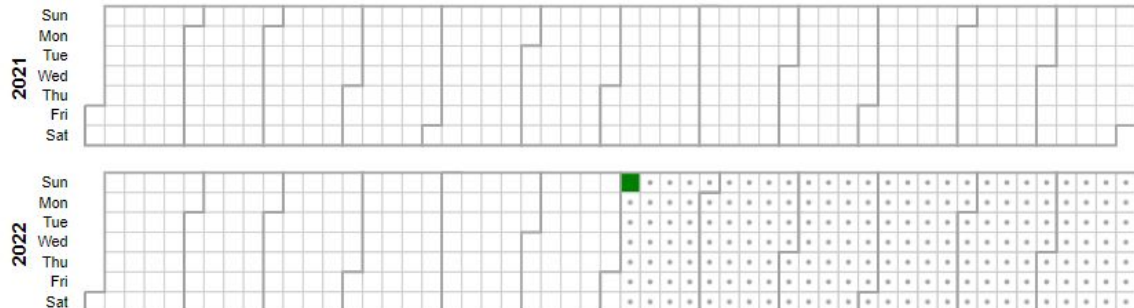
Gantt

Details

Code

Audit Log

success failed running planned no\_status





# Apache Airflow - Duración tareas

REAL \* IA PARA UN MUNDO



# Apache Airflow - Código DAG

REAL \* IA PARA UN MUNDO



Airflow

DAGs

Security

Browse

Admin

Docs

20:40 UTC



DAG: ingest-transform

Schedule: 0 0 \* \* \*

Next Run: 2022-07-02, 00:00:00

Grid

Graph

Calendar

Task Duration

Task Tries

Landing Times

Gantt

Details

Code

Audit Log



```
1 from datetime import timedelta
2 from airflow import DAG
3 from airflow.operators.bash import BashOperator
4 from airflow.operators.dummy import DummyOperator
5 from airflow.utils.dates import days_ago
6
7 args = {
8     'owner': 'airflow',
9 }
10
11 with DAG(
12     dag_id='ingest-transform',
13     default_args=args,
14     schedule_interval='0 0 * * *',
15     start_date=days_ago(2),
16     dagrun_timeout=timedelta(minutes=60),
17     tags=['ingest-transform'],
18 ):
```

Toggle Wrap

# Apache Airflow - Operadores

REAL \* IA PARA UN MUNDO

Airflow DAGs Security Browse Admin Docs

20:32 UTC AA

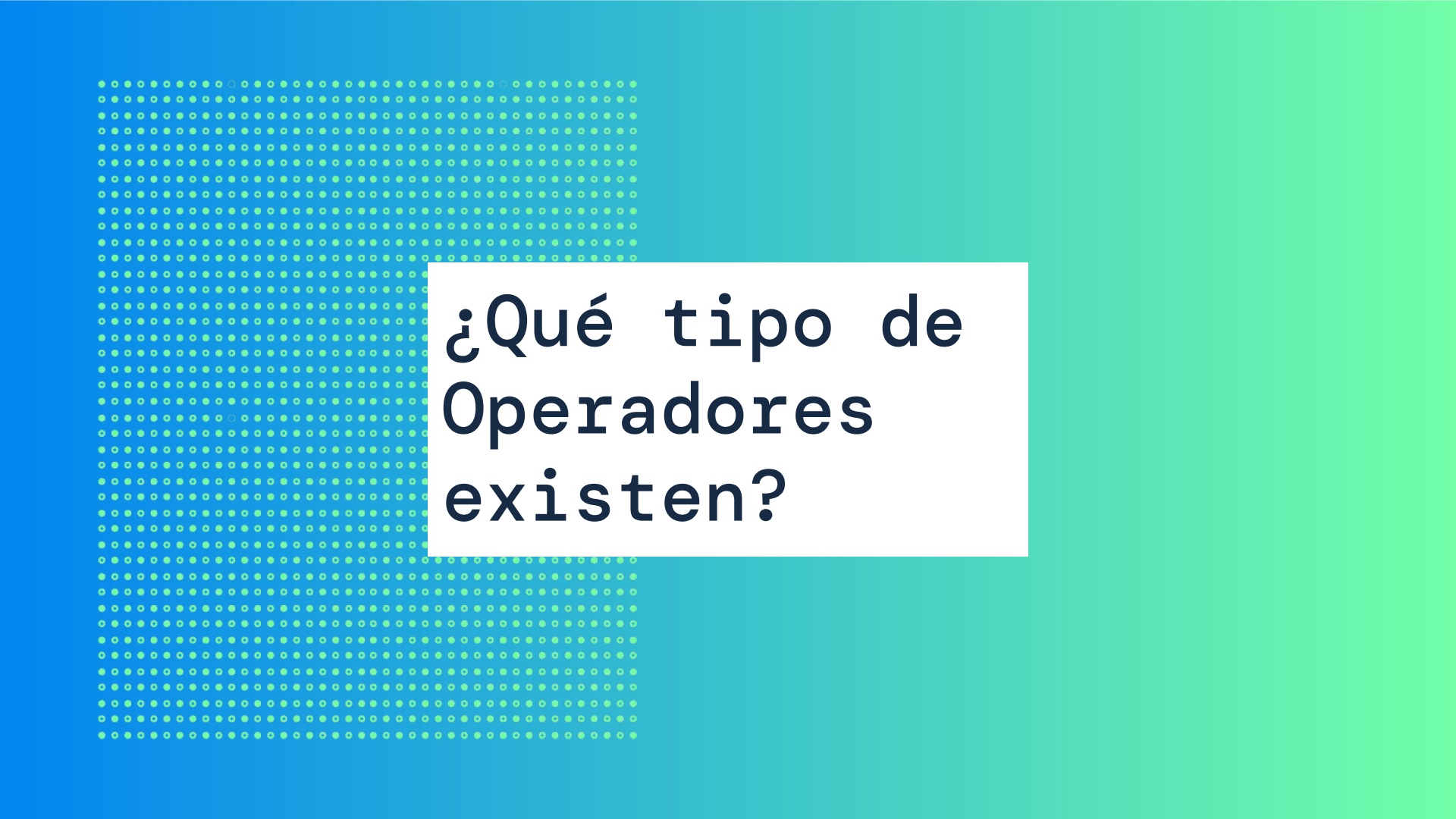
```
15 # K1NU, either express or implied. See the License for the
16 # specific Language governing permissions and limitations
17 # under the License.
18
19 """
20 Example Airflow DAG that shows the complex DAG structure.
21 """
22 import pendulum
23
24 from airflow import models
25 from airflow.models.baseoperator import chain
26 from airflow.operators.bash import BashOperator
27
28 with models.DAG(
29     dag_id="example_complex",
30     schedule_interval=None,
31     start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
32     catchup=False,
33     tags=['example', 'example2', 'example3'],
34 ) as dag:
35     # Create
36     create_entry_group = BashOperator(task_id="create_entry_group", bash_command="echo create_entry_group")
37
38     create_entry_group_result = BashOperator(
39         task_id="create_entry_group_result", bash_command="echo create_entry_group_result"
40     )
41
42     create_entry_group_result2 = BashOperator(
43         task_id="create_entry_group_result2", bash_command="echo create_entry_group_result2"
44     )
45
46     create_entry_gcs = BashOperator(task_id="create_entry_gcs", bash_command="echo create_entry_gcs")
47
48     create_entry_gcs_result = BashOperator(
49         task_id="create_entry_gcs_result", bash_command="echo create_entry_gcs_result"
50     )
```



¿Qué es un  
operador?

# Apache Airflow - operadores


- Los operadores son los componentes principales de los DAG de Airflow. Son clases que encapsulan la lógica para hacer una unidad de trabajo.
- Cuando se crea una instancia de un operador en un DAG y se le proporciona los parámetros requeridos, se convierte en una tarea.
- Cuando Airflow ejecuta esa tarea para una fecha de ejecución determinada, se convierte en una instancia de tarea.



¿Qué tipo de  
Operadores  
existen?

# Apache Airflow - operadores

- BashOperator: ejecuta un comando bash.
- PythonOperator: invoca una función Python.
- EmailOperator: envía un email.
- SimpleHttpOperator: hace una petición HTTP.
- DatabaseOperator: MySQLOperator, SqliteOperator, PostgresOperator, MsSqlOperator, OracleOperator, JdbcOperator, etc. (ejecuta una query SQL)
- Sensor: espera por un tiempo, fichero, fila de base de datos, objeto en S3...
- Dummy: no realizan ninguna acción, se utilizan para comenzar o finalizar un job



¿Qué otras  
alternativas  
existen?



# Alternativas

Otras herramientas de orquestación que existe en el mercado son:

- Luigi (creado por Spotify)
- Apache Nifi
- Aws Step Functions
- Apache Oozie

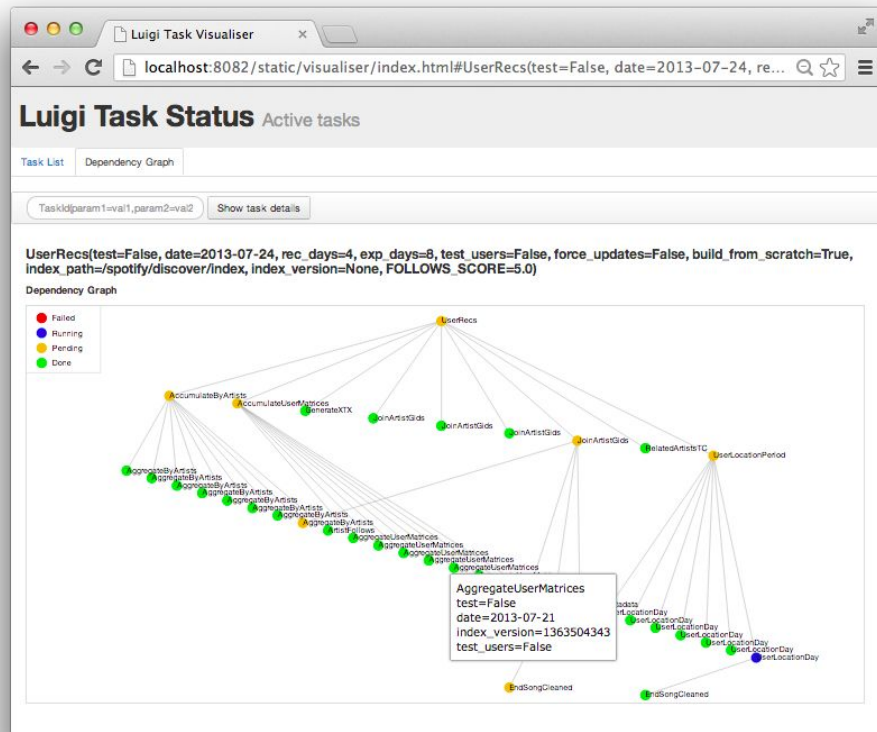
# Alternativas

## Luigi

El gráfico de dependencia se especifica dentro de Python.

Esto facilita la creación de gráficos de dependencias complejas de tareas.

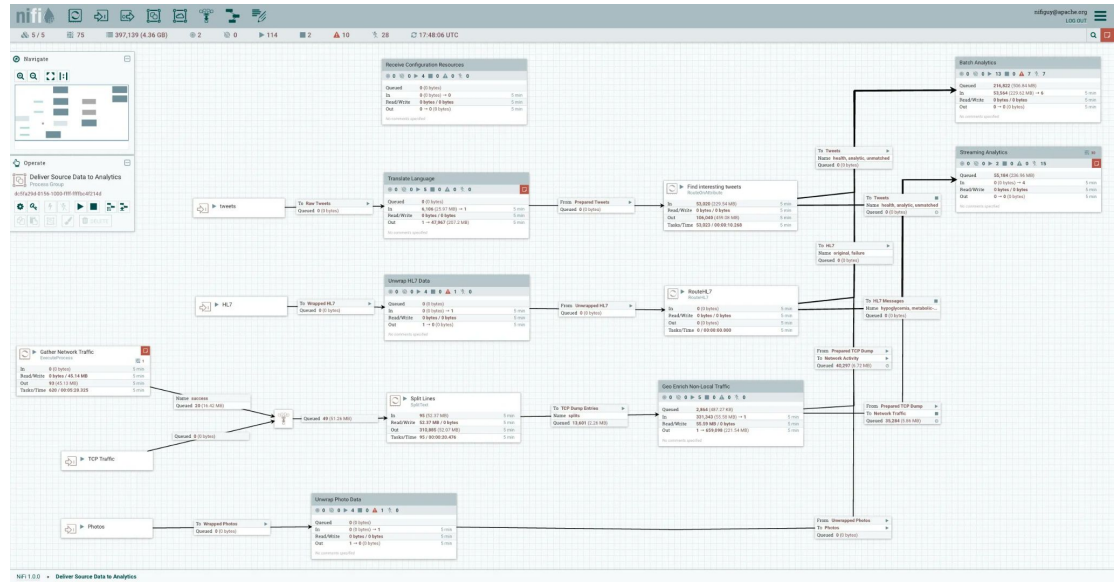
El flujo de trabajo puede desencadenar cosas que no están en Python, cómo ejecutar scripts de Pig o archivos de scripting.



# Alternativas

## Apache NiFi

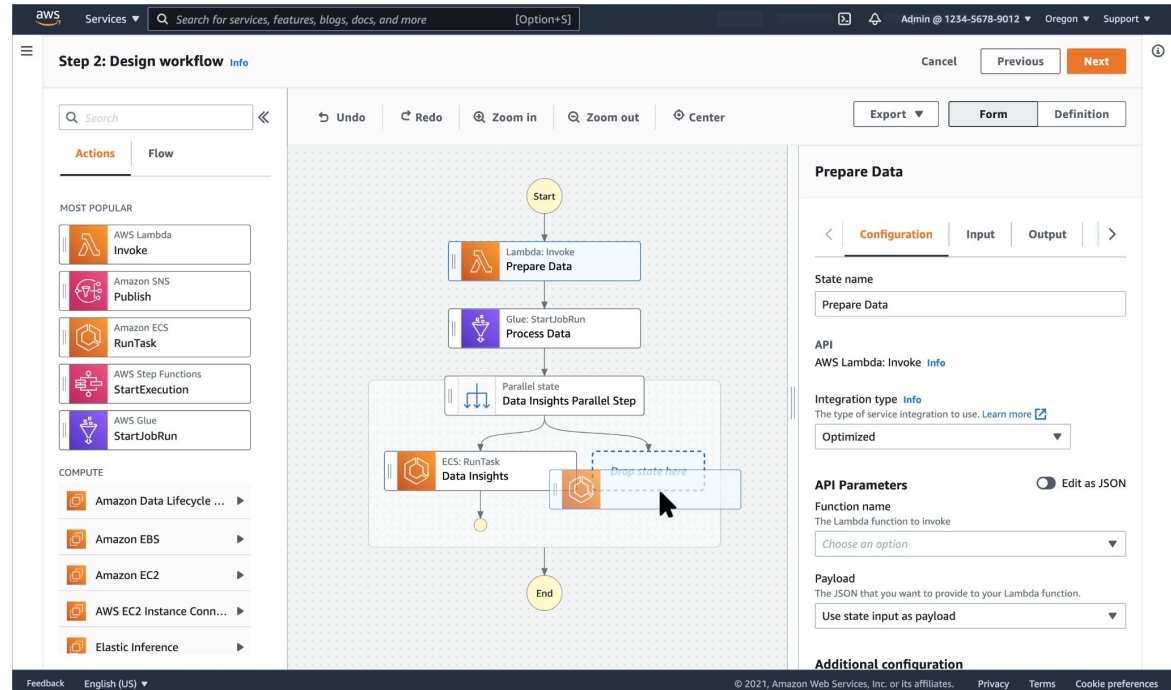
NiFi se creó para automatizar el flujo de datos entre sistemas.



# Alternativas

## Aws Step Functions

Step Functions es un servicio de orquestación sin servidor que le permite combinar funciones Lambda y otros servicios para crear aplicaciones críticas para el negocio.



The screenshot displays the AWS Step Functions console interface. At the top, there's a navigation bar with the AWS logo, a search bar, and user information. The main heading is "Step 2: Design workflow". Below this, there's a left sidebar with "Actions" and "Flow" tabs. The "Actions" tab is active, showing a list of "MOST POPULAR" services: AWS Lambda Invoke, Amazon SNS Publish, Amazon ECS RunTask, AWS Step Functions StartExecution, and AWS Glue StartJobRun. Under the "COMPUTE" section, there are links to Amazon Data Lifecycle, Amazon EBS, Amazon EC2, AWS EC2 Instance Connect, and Elastic Inference. The central area shows a workflow diagram starting with a "Start" node, followed by a "Lambda: Invoke Prepare Data" action, then a "Glue: StartJobRun Process Data" action. This is followed by a "Parallel state Data Insights Parallel Step" which branches into two parallel actions: "ECS: RunTask Data Insights" and a "Drop state here" placeholder. The workflow ends at an "End" node. On the right, the "Prepare Data" configuration panel is visible, showing fields for "State name" (Prepare Data), "API" (AWS Lambda: Invoke), "Integration type" (Optimized), and "API Parameters" (Function name, Payload).

REAL \* IA PARA UN MUNDO

# Alternativas

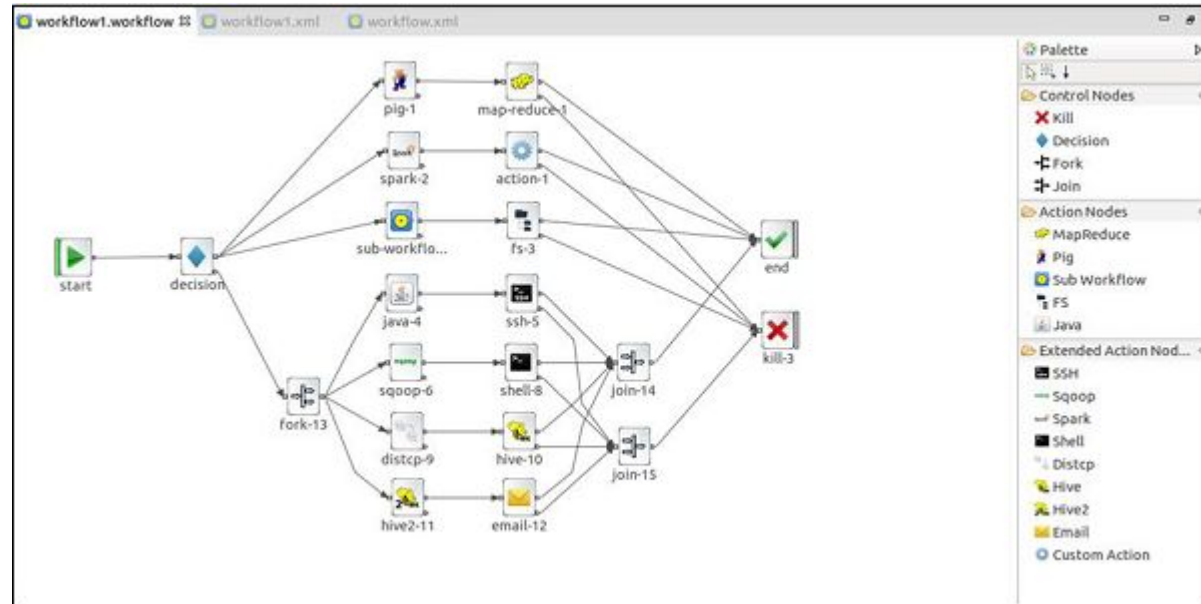
REAL \* IA PARA UN MUNDO

## Apache Oozie

Oozie es un sistema de programación de flujo de trabajo para administrar trabajos de Apache Hadoop.

Los trabajos de flujo de trabajo de Oozie son DAGs.

Oozie está integrado con el resto de la pila de Hadoop y admite varios tipos de trabajos de Hadoop (Hive, Pig y Sqoop)





# Data pipelines