

Introducción a Apache Airflow

Qué es y por qué es útil en procesos
de datos

¿Qué es Apache Airflow?

- Framework open-source creado en Airbnb (2014)
- Orquestador de workflows: coordina tareas en distintos sistemas
- Pipelines definidos como DAGs (Directed Acyclic Graphs) en Python
- Componentes principales:
 - Scheduler (programa tareas)
 - Workers (ejecutan tareas)
 - Webserver (visualización y monitoreo)

¿Qué es un DAG?

- Grafo dirigido sin ciclos (Directed Acyclic Graph)
- Nodos = tareas (ej. extraer datos, transformarlos, cargarlos)
- Aristas = dependencias (ej. limpiar datos → entrenar modelo)
- Ventajas:
 - Ejecución paralela
 - Reejecución selectiva de tareas fallidas

Ejemplo gráfico: DAG simple

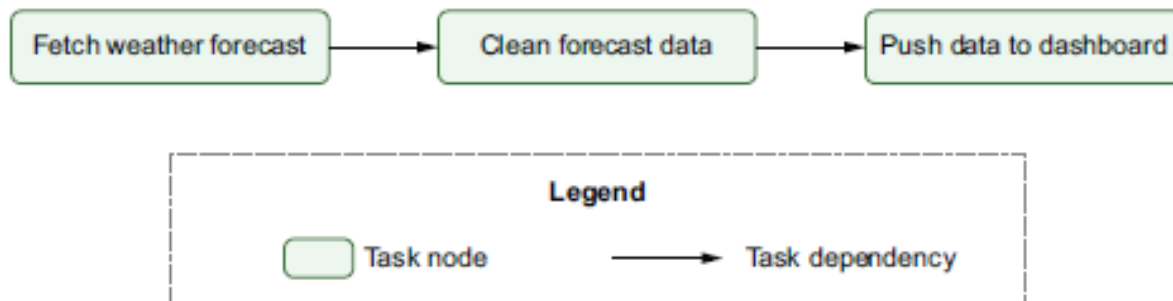


Figure 1.2 Graph representation of the data pipeline for the weather dashboard. Nodes represent tasks and directed edges represent dependencies between tasks (with an edge pointing from task A to task B, indicating that task A needs to be run before task B).

Ejemplo gráfico: DAG con ramas paralelas

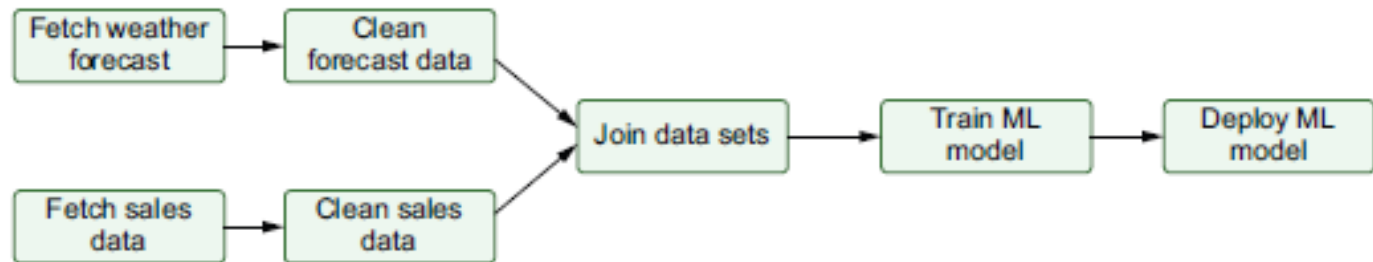


Figure 1.6 Independence between sales and weather tasks in the graph representation of the data pipeline for the umbrella demand forecast model. The two sets of fetch/cleaning tasks are independent as they involve two different data sets (the weather and sales data sets). This independence is indicated by the lack of edges between the two sets of tasks.

¿Por qué usar Airflow en procesos de datos?

- Flexibilidad: pipelines en Python
- Integraciones con DBs, Big Data y Cloud
- Escalabilidad con múltiples workers
- Scheduling avanzado (cron-like)
- Monitoreo con UI rica
- Backfilling para regenerar datasets históricos

Casos de uso típicos

- ETL / ELT para Data Warehouses
- Ingesta incremental de datos
- Entrenamiento y despliegue de modelos ML
- Orquestación de tareas en múltiples servicios cloud
- Construcción de Data Lakes y Data Platforms

¿Cuándo NO usar Airflow?

- Procesos streaming (mejor Kafka, Flink)
- Workflows muy dinámicos que cambian en cada ejecución
- Equipos sin experiencia en Python
- Si se requiere lineage/versionado de datos → usar herramientas adicionales

Conclusión

- Apache Airflow:
 - No procesa datos directamente, los orquesta
 - Es el 'cerebro' de los pipelines batch
 - Facilita escalabilidad, monitoreo y resiliencia
 - Estándar en Data Engineering y MLOps

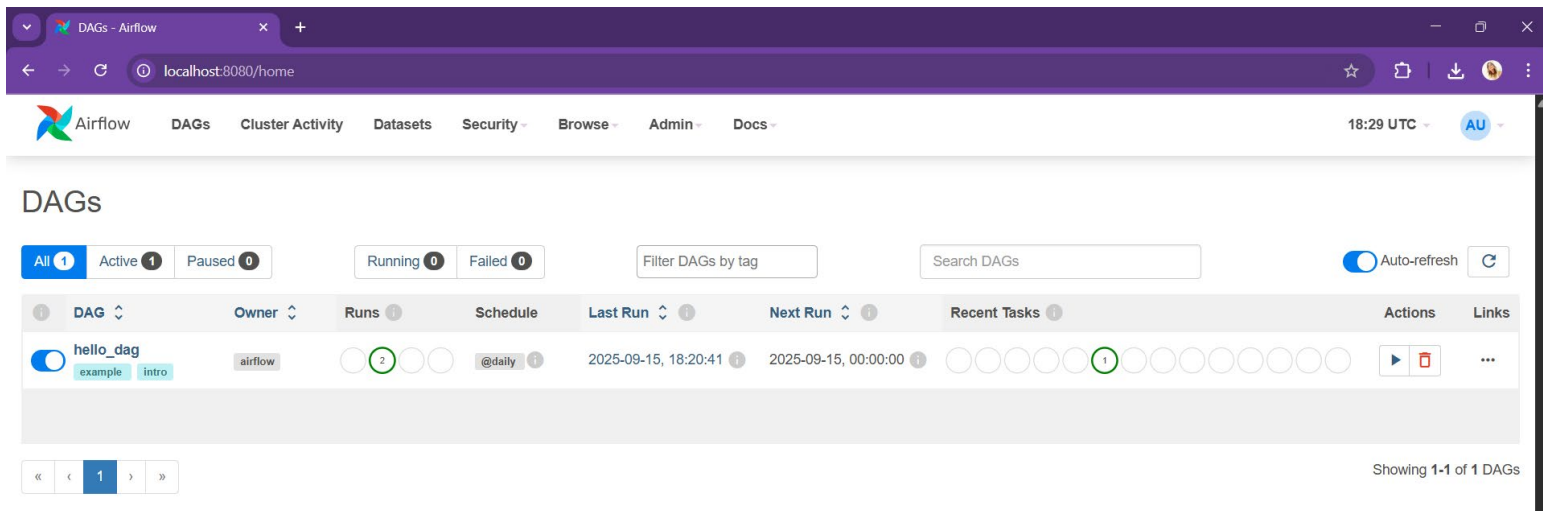
<https://github.com/darkanita/apache-airflow-workshop>

Demo: tu primer DAG en Airflow

Explorando la interfaz web y
ejecutando tu primer pipeline

Lista de DAGs

- Aquí vemos todos los DAGs disponibles
- Un DAG = Directed Acyclic Graph (pipeline de tareas)
- Podemos activarlos o pausarlos desde aquí



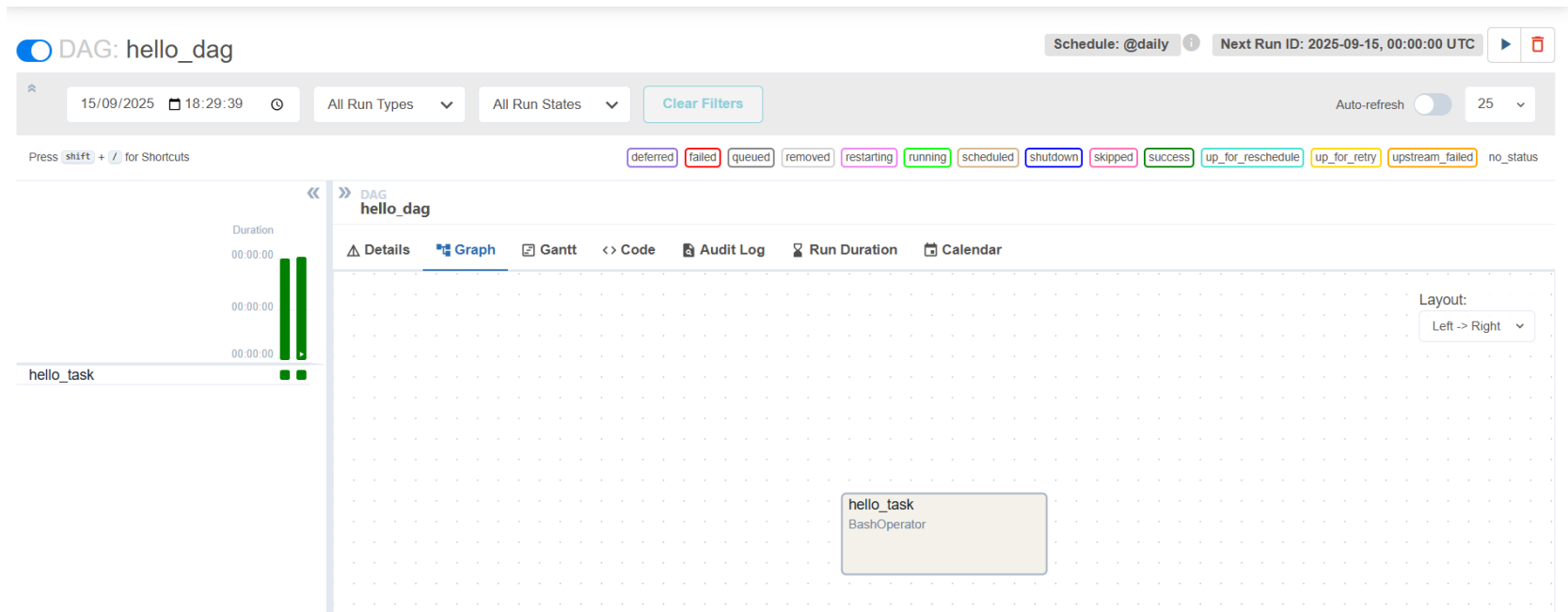
The screenshot shows the Apache Airflow web interface. The top navigation bar includes links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The main heading is "DAGs". Below this, there are filters for "All" (1), "Active" (1), and "Paused" (0). There are also buttons for "Running" (0) and "Failed" (0), a "Filter DAGs by tag" input, and a "Search DAGs" input. A toggle for "Auto-refresh" is set to "On".

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<input checked="" type="checkbox"/> hello_dag example intro	airflow	<input checked="" type="radio"/> 2 <input type="radio"/>	@daily 1	2025-09-15, 18:20:41 1	2025-09-15, 00:00:00 1	<input checked="" type="radio"/> 1 <input type="radio"/>	<input type="button" value="▶"/> <input type="button" value="⏏"/>	...

At the bottom, there is a pagination control showing "1" and a status message "Showing 1-1 of 1 DAGs".

Vista del DAG (Graph)

- Visualización de tareas y dependencias
- Cada nodo = una tarea
- Flechas = dependencias



Ejecución y Logs

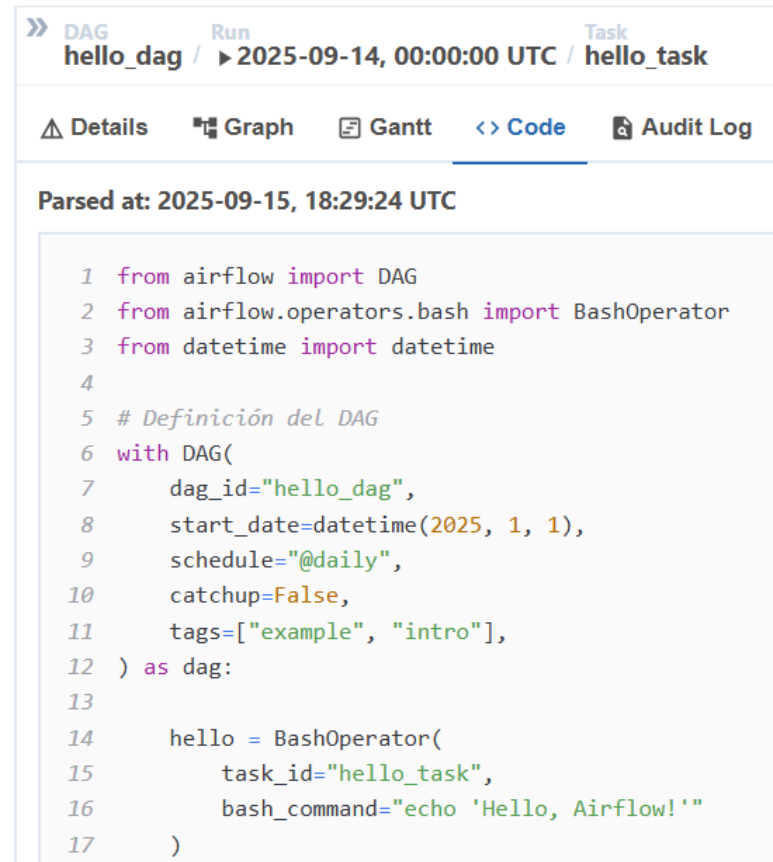
- Airflow ejecuta comandos o scripts definidos en cada tarea
- Los logs permiten auditar qué ocurrió en cada ejecución
- Ejemplo: echo 'Hello, Airflow!'

The screenshot displays the Apache Airflow web interface for a task named 'hello_task'. The breadcrumb navigation shows 'DAG hello_dag / Run 2025-09-14, 00:00:00 UTC / Task hello_task'. On the right, there are buttons for 'Clear task', 'Mark state as...', and 'Filter DAG by task'. Below the navigation bar, a menu includes 'Details', 'Graph', 'Gantt', 'Code', 'Audit Log', 'Logs' (which is selected), 'XCom', and 'Task Duration'. The main content area shows '(by attempts)' with a dropdown menu set to '1'. Below this, there are filters for 'All Levels' and 'All File Sources', along with options for 'Wrap', 'Download', and 'See More'. The log content is displayed in a light blue box and includes the following text:

```
5a366abc2ca0
*** Found local files:
***   * /opt/airflow/logs/dag_id=hello_dag/run_id>manual__2025-09-15T18:20:41.395879+00:00/task_id=hello_task/attempt=1.log
[2025-09-15, 18:20:41 UTC] {local_task_job_runner.py:120} ► Pre task execution logs
[2025-09-15, 18:20:42 UTC] {subprocess.py:63} INFO - Tmp dir root location: /tmp
[2025-09-15, 18:20:42 UTC] {subprocess.py:75} INFO - Running command: ['/usr/bin/bash', '-c', 'echo 'Hello, Airflow!']
[2025-09-15, 18:20:42 UTC] {subprocess.py:86} INFO - Output:
[2025-09-15, 18:20:42 UTC] {subprocess.py:93} INFO - Hello, Airflow!
[2025-09-15, 18:20:42 UTC] {subprocess.py:97} INFO - Command exited with return code 0
[2025-09-15, 18:20:42 UTC] {taskinstance.py:441} ► Post task execution logs
```

Conceptos clave de Airflow

- DAG → pipeline de tareas
- Task → unidad de trabajo (ej: script Bash, query SQL, función Python)
- Scheduler → decide cuándo correr cada tarea
- Web UI → interfaz para ver, ejecutar y monitorear



The screenshot displays the Apache Airflow web interface for a specific DAG. At the top, the breadcrumb navigation shows 'DAG hello_dag / Run 2025-09-14, 00:00:00 UTC / Task hello_task'. Below this, there are tabs for 'Details', 'Graph', 'Gantt', 'Code' (which is currently selected), and 'Audit Log'. The main content area shows the code that was parsed at '2025-09-15, 18:29:24 UTC'. The code is a Python script that defines a DAG with the ID 'hello_dag', starting on '2025-09-01' with a daily schedule. It includes a single task named 'hello_task' of type 'BashOperator' that executes the command 'echo 'Hello, Airflow!''.

```
1 from airflow import DAG
2 from airflow.operators.bash import BashOperator
3 from datetime import datetime
4
5 # Definición del DAG
6 with DAG(
7     dag_id="hello_dag",
8     start_date=datetime(2025, 1, 1),
9     schedule="@daily",
10    catchup=False,
11    tags=["example", "intro"],
12 ) as dag:
13
14    hello = BashOperator(
15        task_id="hello_task",
16        bash_command="echo 'Hello, Airflow!'"
17    )
```

Anatomía de un DAG en Airflow

Cómo se estructura un pipeline de
datos en Python

¿Qué es un DAG?

- DAG = Directed Acyclic Graph (grafo acíclico dirigido)
- Representa un pipeline como conjunto de tareas y dependencias
- En Airflow, los DAGs se definen en archivos Python (*.py)
- Cada DAG incluye: tareas, dependencias y metadatos (schedule, owner, etc.)

Estructura básica de un DAG en Python

- Importar librerías de Airflow
- Definir argumentos por defecto (default_args)
- Crear objeto DAG con su id y schedule
- Definir tareas (Operators o funciones @task)
- Establecer dependencias entre tareas

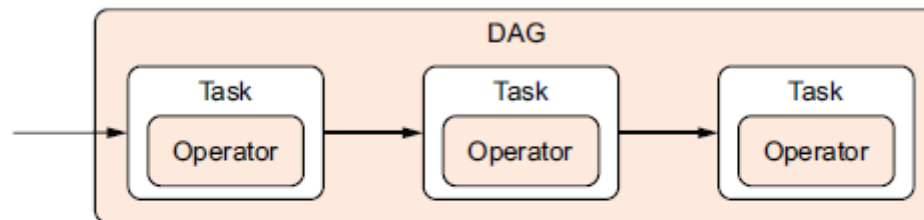


Figure 2.4 DAGs and operators are used by Airflow users. Tasks are internal components to manage operator state and display state changes (e.g., started/finished) to the user.

Ejemplo simple de DAG

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

with DAG('hello_dag', start_date=datetime(2023,1,1), schedule='@daily') as dag:

    tarea1 = BashOperator(task_id='saludo', bash_command="echo 'Hola Airflow!'")
    tarea2 = BashOperator(task_id='despedida', bash_command="echo 'Adiós Airflow!'")

    tarea1 >> tarea2
```

Elementos clave de un DAG

- DAG ID: identificador único del pipeline
- start_date: desde cuándo empieza a ejecutarse
- schedule_interval: cada cuánto corre (@daily, @hourly, cron)
- task_id: nombre único de cada tarea
- Operators: definen el tipo de tarea (BashOperator, PythonOperator, etc.)

Mejores Prácticas en Apache Airflow

Consejos para crear DAGs más
robustos y mantenibles

Principales recomendaciones

- Usar nombres claros para DAGs y tareas (dag_id, task_id)
- Definir argumentos por defecto (default_args) para evitar repetición
- Establecer dependencias explícitas entre tareas (task1 >> task2)
- Configurar retries y retry_delay para tolerancia a fallos
- Mantener DAGs simples y modulares (no sobrecargar con lógica compleja)
- Documentar DAGs y tareas con docstrings o parámetros `doc_md`
- Versionar código de DAGs con Git
- Probar DAGs localmente antes de pasarlos a producción
- Usar conexiones y variables de Airflow (no hardcodear credenciales)

Arquitecturas de Airflow para Producción

Basado en 'Data Pipelines with
Apache Airflow' (Manning, 2021)

Arquitectura básica: Standalone

- Scheduler, Webserver y Worker en una sola máquina
- Base de datos SQLite (por defecto)
- Útil para desarrollo y pruebas locales

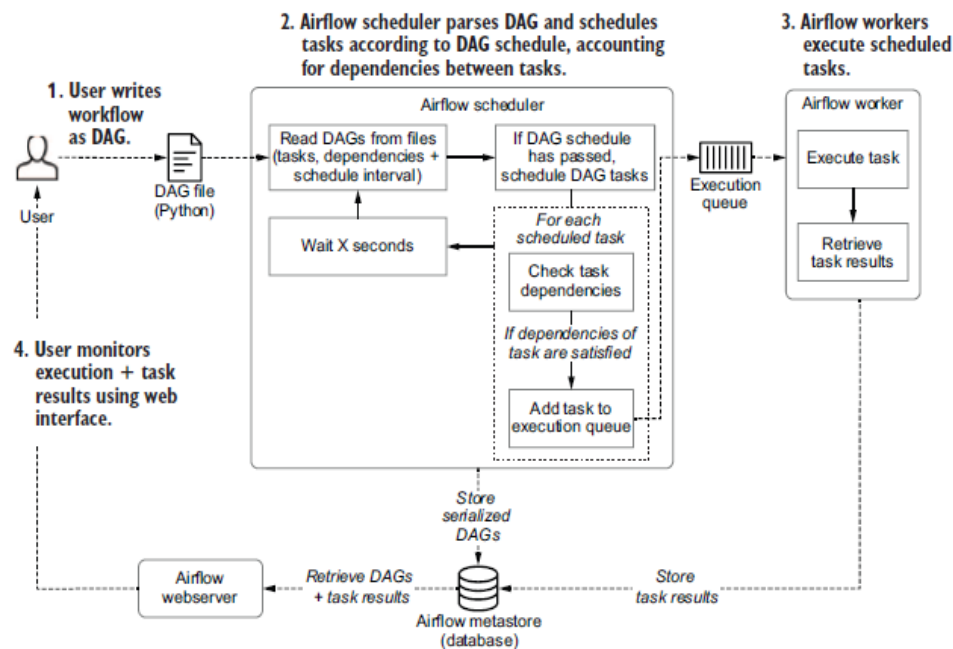


Figure 1.9 Schematic overview of the process involved in developing and executing pipelines as DAGs using Airflow

Arquitectura LocalExecutor con Postgres

- Scheduler y Webserver en un servidor
- Workers gestionados localmente en el mismo host
- Base de datos externa (ej: Postgres)
- Apta para entornos pequeños en producción

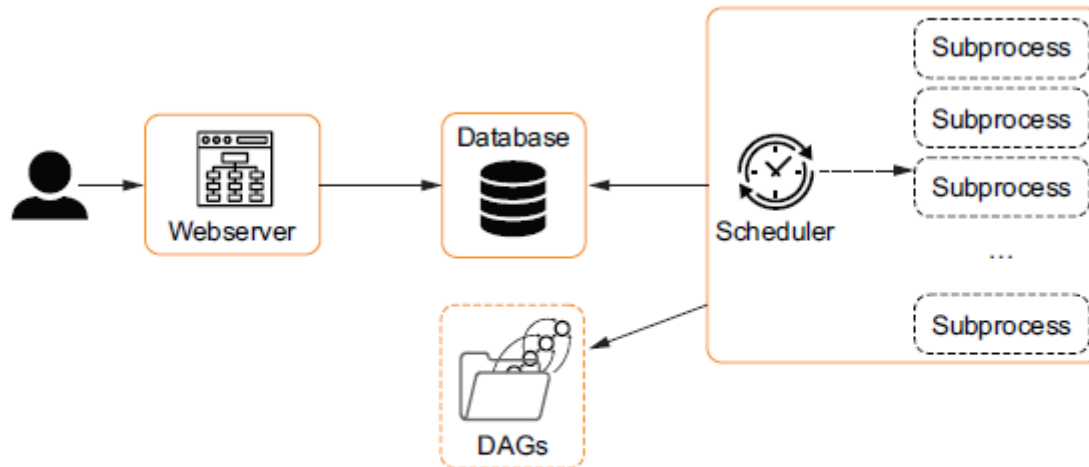


Figure 12.6 With the LocalExecutor all components can run on a separate machine. However, subprocesses created by the scheduler all run on one single machine.

Arquitectura distribuida con CeleryExecutor

- Scheduler, Webserver y múltiples Workers en nodos separados
- Mensajería con Redis o RabbitMQ
- Base de datos externa (Postgres/MySQL)
- Escalable y tolerante a fallos

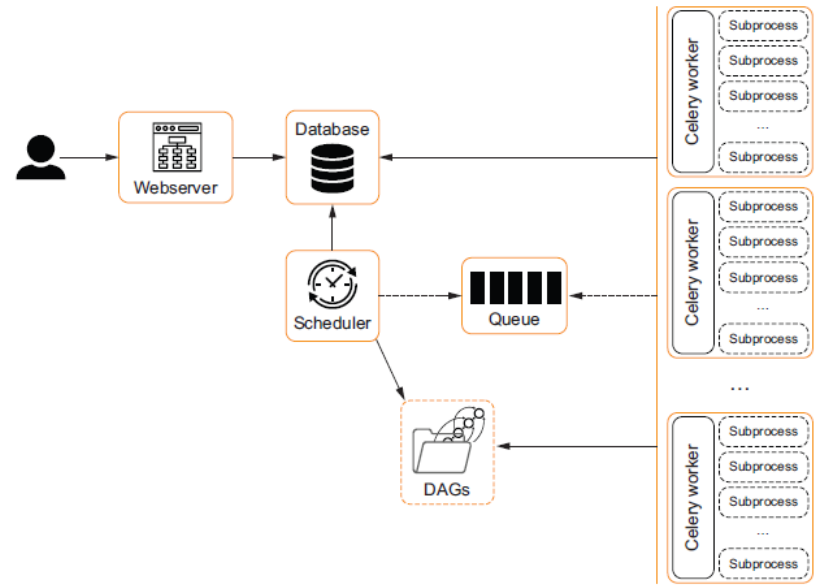


Figure 12.7 In the CeleryExecutor, tasks are divided among multiple machines running Celery workers. The workers wait for tasks to arrive on a queue.

Arquitectura en Kubernetes

- Scheduler y Webserver corren como pods en Kubernetes
- Cada tarea se ejecuta en un pod independiente
- Escalabilidad dinámica y aislamiento de tareas
- Ideal para producción en la nube

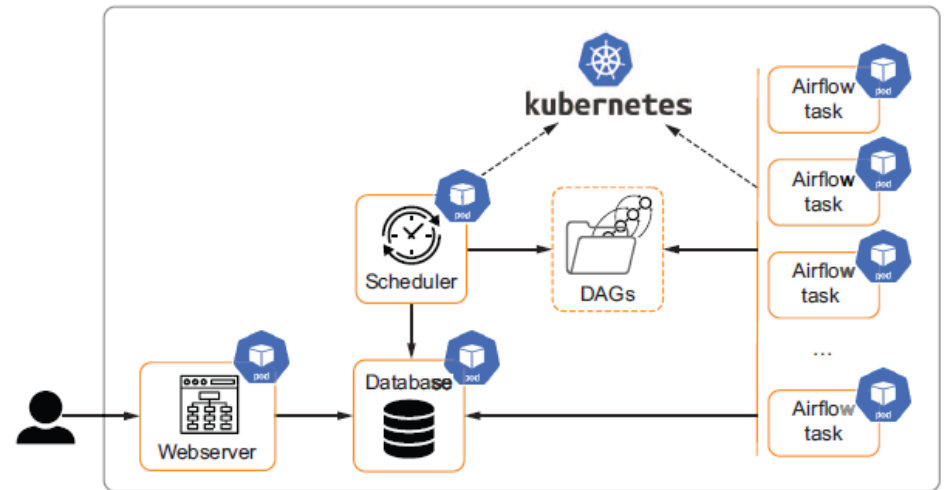


Figure 12.10 With the `KubernetesExecutor`, all tasks run in a pod in Kubernetes. While it is not necessary to run the webserver, scheduler, and database in Kubernetes, it is sensible to also run it there when using the `KubernetesExecutor`.