Apache Airflow Executors

Bavatharani S

1. What is an Executor?

- An Executor in Airflow is the component responsible for running tasks.
- It takes task instances from the scheduler and executes them, either locally or on remote workers.
- Executors are pluggable and configurable in airflow.cfg or via environment variables.

2. Types of Executors

SequentialExecutor

- Use Case: Simple development/testing
- Pros: No setup needed
- Cons: Runs one task at a time (slow)

LocalExecutor

- Use Case: Single-machine parallel execution
- Pros: Supports parallelism
- Cons: Shared resources on one machine

CeleryExecutor

- Use Case: Multi-worker scalable deployments
- Pros: High concurrency
- Cons: Requires message broker (Redis/RabbitMQ)

KubernetesExecutor

- Use Case: Cloud-native / isolated tasks
- Pros: Task-level isolation
- Cons: Requires Kubernetes cluster

Custom Executor

- Use Case: Specialized execution needs

- Pros: Fully tailored execution
- Cons: Requires development effort

3. Choosing an Executor

- Sequential Executor: Best for learning and small test setups.
- Local Executor: Ideal for small-scale single-machine parallel execution.
- Celery Executor: Suitable for production-scale, distributed environments.
- Kubernetes Executor: Recommended for cloud-native deployments with task isolation.

4. Configuring Executors

A. LocalExecutor

1. Open airflow.cfg → [core] section
2. executor = LocalExecutor
3. Or set via environment variable:
4. export AIRFLOW__CORE__EXECUTOR=LocalExecutor

B. CeleryExecutor

1. Install dependencies:
2. pip install 'apache-airflow[celery]'
3. pip install redis # or rabbitmq
4. Update airflow.cfg:
5. executor = CeleryExecutor
6. Configure Celery backend (Redis/RabbitMQ).

C. KubernetesExecutor

1. Ensure Kubernetes cluster is running.
2. Update airflow.cfg:
3. executor = KubernetesExecutor
4. Configure Kubernetes-specific settings (namespaces, pod templates).

5. Testing an Executor (Example: LocalExecutor)

1. Install Airflow:

   pip install apache-airflow

2. Initialize database:

   airflow db init

3. Configure executor in airflow.cfg:

   executor = LocalExecutor

4. Launch Airflow components:

   airflow scheduler

   airflow webserver

5. Create a test DAG (my_test_dag.py):

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime
with DAG("example_local_executor",
      start_date=datetime(2025, 8, 19),
      schedule_interval=None) as dag:
  t1 = BashOperator(task_id="task1", bash_command="echo 'Task 1'")
  t2 = BashOperator(task_id="task2", bash_command="echo 'Task 2'")
  t1 >> t2
```

Open Airflow UI (http://localhost:8080) and trigger the DAG to see tasks run concurrently.

6. Best Practices

- Avoid storing files locally between tasks in distributed setups.
- Use XComs for small data or external storage (e.g., S3) for large data.
- For production, prefer CeleryExecutor or KubernetesExecutor for scalability and isolation.
- Use SequentialExecutor only for learning or single-task debugging.