Apache Airflow Executors – Summary

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

Executor Type	Use Case	Pros	Cons / Requirements
SequentialExecutor	Simple development/testing	No setup needed	Runs one task at a time (slow)
LocalExecutor	Single-machine parallel execution	Supports parallelism	Shared resources on one machine
CeleryExecutor	Multi-worker scalable deployments	High concurrency	Requires message broker (Redis/RabbitMQ)
KubernetesExecutor	Cloud-native / isolated tasks	Task-level isolation	Requires Kubernetes cluster
Custom Executor	Specialized execution needs	Fully tailored execution	Requires development effort

3. Choosing an Executor

- **SequentialExecutor:** Best for learning and small test setups.
- **LocalExecutor:** Ideal for small-scale single-machine parallel execution.
- **CeleryExecutor:** Suitable for production-scale, distributed environments.
- **KubernetesExecutor:** Recommended for cloud-native deployments with task isolation.

4. Configuring Executors

A. Using LocalExecutor

1. Open airflow.cfg.
2. Set in [core] section:

   ```
   executor = LocalExecutor
   ```

3. Or via environment variable:

   ```
   export AIRFLOW__CORE__EXECUTOR=LocalExecutor
   ```

B. Using CeleryExecutor

1. Install dependencies: celery, redis or rabbitmq.
2. Update airflow.cfg:

   ```
   executor = CeleryExecutor
   ```

3. Configure Celery backend settings.

C. Using KubernetesExecutor

1. Ensure a Kubernetes cluster is running.
2. Update airflow.cfg:

   ```
   executor = KubernetesExecutor
   ```

3. 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 as 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
```

6. 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.