Building Automated Data Pipelines with Airflow

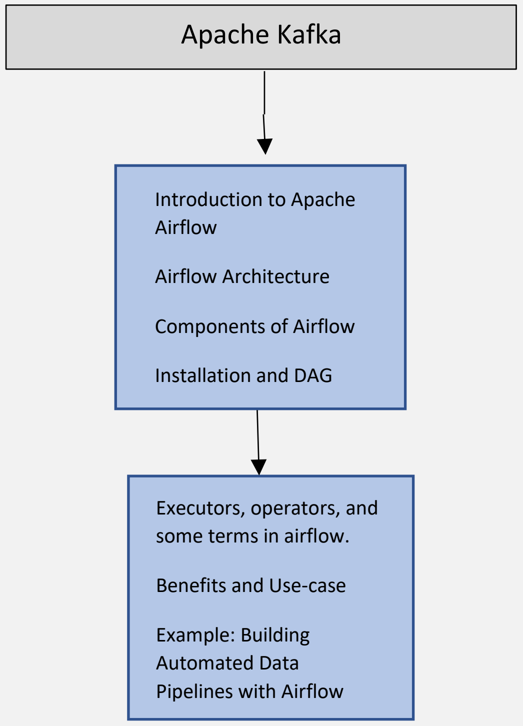
Apache Airflow is a workflow engine that will easily schedule and run your complex data pipelines.

As a part of Building Automated Data Pipelines with Airflow, you covered:

- Introduction to Apache Airflow
- Airflow Benefits and Use-cases
- Apache Airflow Architecture and Components
- Installation steps and DAG in Airflow and executors, operators in airflow
- Example: Building Automated Data Pipelines with Airflow

Common Interview Questions:

1. How do we define workflows in Apache Airflow?
2. What are the components of the Apache Airflow architecture?
3. How does airflow communicate with a third party (S3, Postgres, MySQL)?
4. What are the basic steps to create a DAG?
5. What is Branching in Directed Acyclic Graphs (DAGs)
6. What are ways to Control Airflow Workflow?
7. How are Connections used in Apache Airflow?
8. What are the ways to monitor Apache Airflow?
9. Is Apache Airflow an ETL tool?


Apache Kafka
Introduction to Apache Airflow
Airflow Architecture
Components of Airflow
Installation and DAG
Executors, operators, and some terms in airflow.
Benefits and Use-case
Example: Building Automated Data Pipelines with Airflow

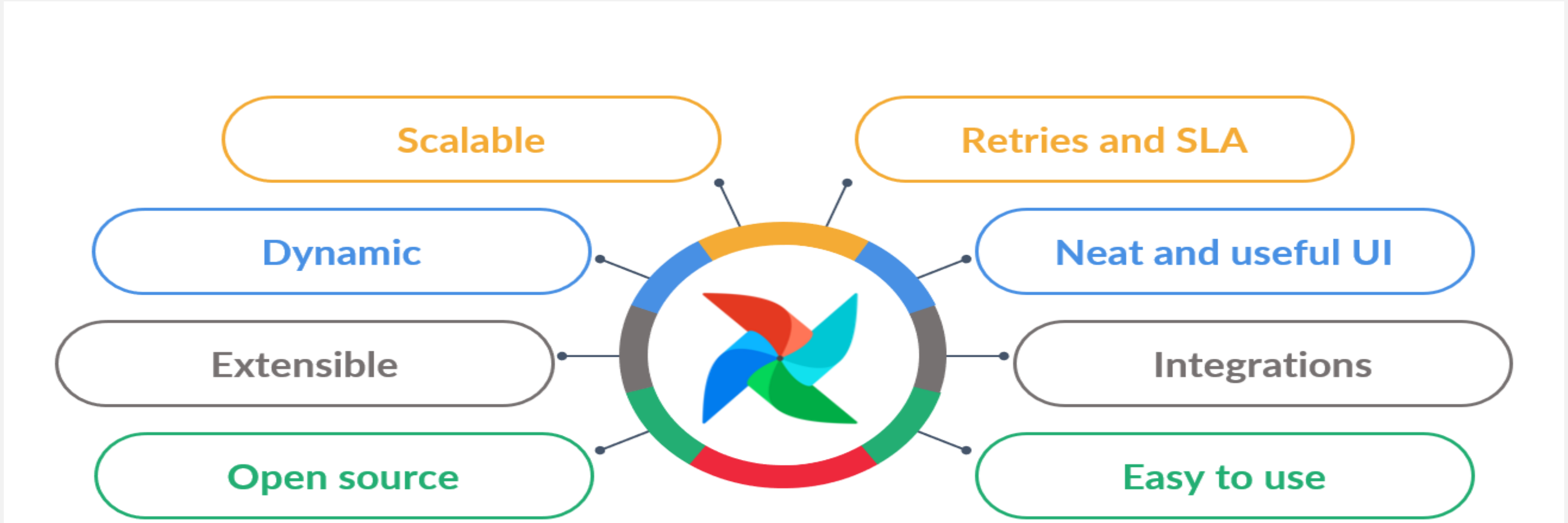
Building Automated Data Pipelines with Airflow

Apache Airflow:

Apache Airflow is a workflow engine that will easily schedule and run your complex data pipelines. It will make sure that each task of your data pipeline will get executed in the correct order and each task gets the required resources.

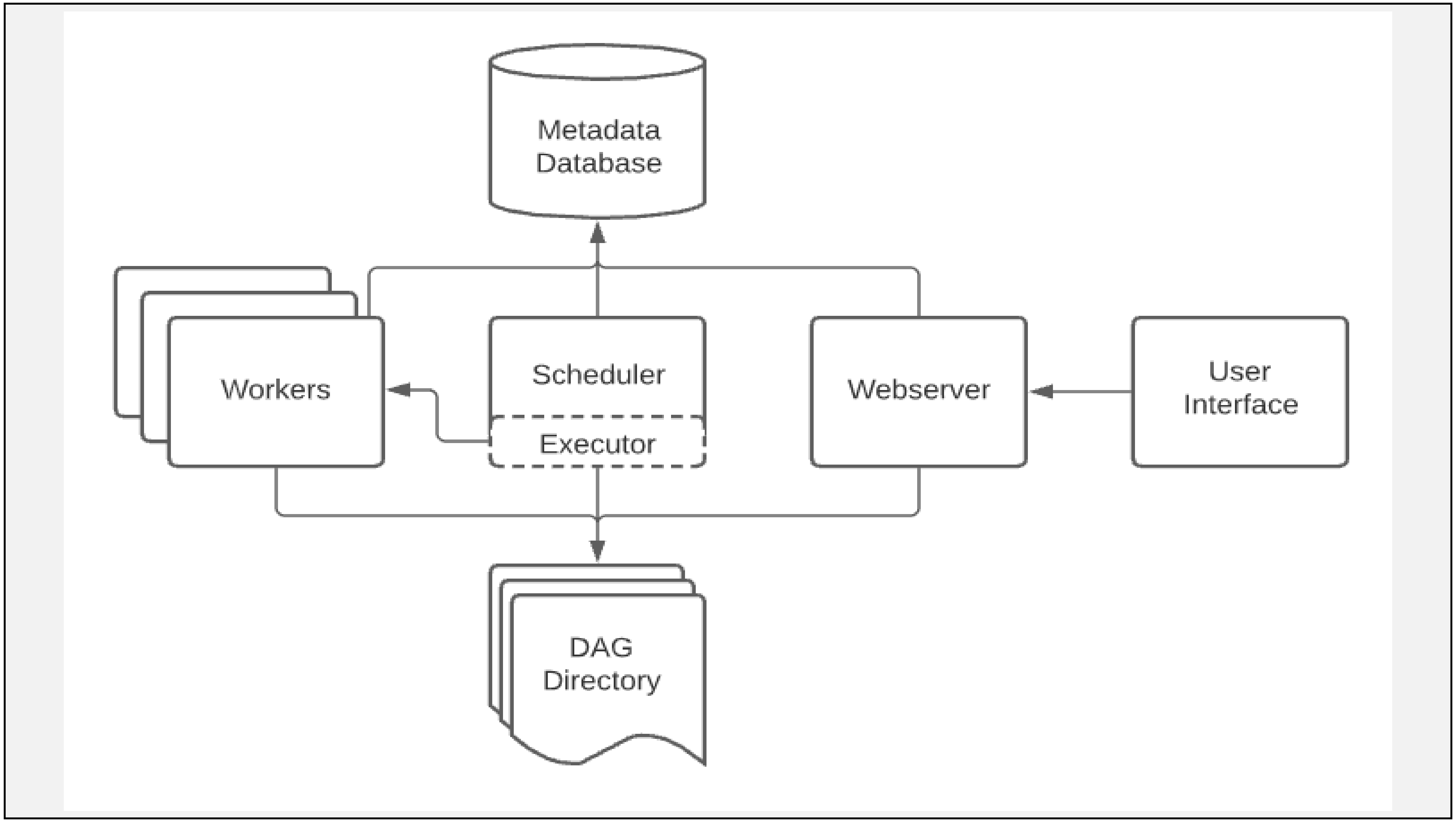
Data pipelines in Airflow are:

- Implemented in the form of Directed Acyclic Graphs (DAGs);
- Created using Python code; and
- Can be generated dynamically.



Features

Apache Airflow Architecture:



Building Automated Data Pipelines with Airflow

Components of Apache Airflow:

User interface: lets you view DAGs, Tasks and logs, trigger runs and debug DAGs. This is the easiest way to keep track of your overall Airflow installation and dive into specific DAGs to check the status of tasks.

Hooks: Airflow uses Hooks to interface with third-party systems, enabling connection to external APIs and databases (e.g. Hive, S3, GCS, MySQL, Postgres). Hooks should not contain sensitive information such as authentication credentials.

Providers: packages containing the core Operators and Hooks for a particular service. They are maintained by the community and can be directly installed on an Airflow environment

Metadata Database: Airflow stores the status of all the tasks in a database and do all read/write operations of a workflow from here.

Plugins: a variety of Hooks and Operators to help perform certain tasks, such as sending data from SalesForce to Amazon Redshift.

Connections: these contain information that enable a connection to an external system. This includes authentication credentials and API tokens. You can manage connections directly from the UI, and the sensitive data will be encrypted and stored in PostgreSQL or MySQL.

DAG: It is the Directed Acyclic Graph – a collection of all the tasks that you want to run which is organized and shows the relationship between different tasks. It is defined in a python script.

Scheduler: As the name suggests, this component is responsible for scheduling the execution of DAGs. It retrieves and updates the status of the task in the database.

Building Automated Data Pipelines with Airflow

Apache Airflow Benefits:

Apache Airflow benefits:

Ease of use: you only need a little python knowledge to get started.

Open-source community: Airflow is free and has a large community of active users.

Integrations: ready-to-use operators allow you to integrate Airflow with cloud platforms (Google, AWS, Azure, etc).

Coding with standard Python: you can create flexible workflows using Python with no knowledge of additional technologies or frameworks.

Graphical UI: monitor and manage workflows, check the status of ongoing and completed tasks.

Apache Airflow Use-case:

Apache Airflow some Use-Case:

- ETL pipelines that extract data from multiple sources, and run Spark jobs or other data transformations
- Machine learning model training
- Automated generation of reports
- Backups and other DevOps tasks

Apache Airflow Workloads:

The DAG runs through a series of Tasks, which may be subclasses of Airflow's BaseOperator, including:

Operators: predefined tasks that can be strung together quickly

Sensors: a type of Operator that waits for external events to occur

TaskFlow: a custom Python function packaged as a task, which is decorated with @tasks

Apache Airflow Installation Steps:

Step 1: To install pip run the following command in the terminal.

sudo apt-get install python3-pip

Step 2: Next airflow needs a home on your local system. By default ~/airflow is the default location but you can change it as per your requirement.

export AIRFLOW_HOME=~/airflow

Step 3: Now, install the apache airflow using the pip with the following command.

pip3 install apache-airflow

Step 4: Airflow requires a database backend to run your workflows and to maintain them. Now, to initialize the database run the following command.

airflow initdb

Step 5: To start the webserver run the following command in the terminal. The default port is 8080 and if you are using that port for something else then you can change it.

airflow webserver -p 8080

Step 6: Now, start the airflow schedular using the following command in a different terminal. It will run all the time and monitor all your workflows and triggers them as you have assigned.

airflow scheduler

Step 7: Now, create a folder name dags in the airflow directory where you will define your workflows or DAGs and open the web browser and go open: *http://localhost:8080/admin/*

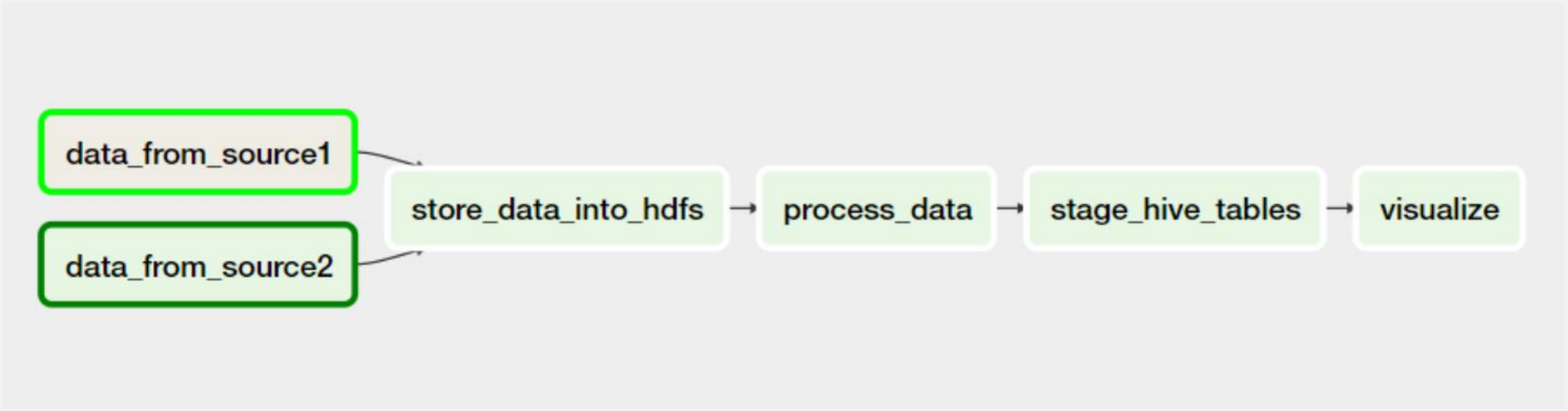
DAG in Apache Airflow:

In Apache Airflow, DAG stands for Directed Acyclic Graph. DAG is a collection of tasks organized in such a way that their relationships and dependencies are reflected. One of the advantages of this DAG model is that it gives a reasonably simple technique for executing the pipeline. Another advantage is that it clearly divides pipelines into discrete incremental tasks rather than relying on a single monolithic script to perform all the work.

The properties of a DAG are as follows:

- DAGs are graph structures (i.e., a collection of vertices and edges).
- It should be directed (i.e., the edges all have a direction indicating which task is dependent on which).
- It must be acyclic (i.e., it cannot contain cycles).

Following diagram shows what a very simple DAG in the Airflow UI looks like:



Sample DAG code:

Sample code for writing a DAG:

```
#Import modules for the DAG
from datetime import datetime
from airflow import DAG

#Define a dictionary with the default arguments of a DAG
dag_default_configs = {
'start_date': datetime(2016, 1, 1),
'owner': 'airflow'
}

#Create DAG object using the DAG()
dag_object = DAG('my_dag',
                default_args = dag_default_configs,
                description='Sample DAG',
                schedule_interval='0 12 * * *')
```

Some sample attributes with respect to the code:

- *ID - 'my_dag'*
- *Description - 'Sample DAG'*
- *Schedule - '0 12 * * *'*
- *Start Date - datetime(2016, 1, 1)*
- *Configs/default arguments - dag_default_configs*

Executors in Apache Airflow:

Sequential Executor

It is a lightweight local executor, which is available by default in airflow. It runs only one task instance at a time and is not production-ready. It can run with SQLite since SQLite does not support multiple connections.

Local Executor

The local executor can run multiple task instances. Generally, we use MySQL or PostgreSQL databases with the local executor since they allow multiple connections, which helps us achieve parallelism.

Celery Executor

Celery is a task queue, which helps in distributing tasks across multiple celery workers. The Celery Executor distributes the workload from the main application onto multiple celery workers with the help of a message broker such as RabbitMQ or Redis.

Kubernetes Executor

The Kubernetes Executor uses the Kubernetes API for resource optimization. It runs as a fixed-single Pod in the scheduler that only requires access to the Kubernetes API. MySQL or PostgreSQL database systems are required to set up the Kubernetes Executor.

CeleryKubernetes Executor

It permits users to run Celery Executor and a Kubernetes Executor simultaneously. It works when you have plenty of small tasks that Celery workers can manage but also have resource-hungry jobs that will be better to run in predefined environments.

Dask Executor

Dask is a parallel computing library in python whose architecture revolves around a sophisticated distributed task scheduler. Dask Executor allows you to run Airflow tasks in a Dask Distributed cluster.

Building Automated Data Pipelines with Airflow

Some famous Apache Airflow Operators:

Apache Airflow Bash Operator

It executes a bash command BashOperator in Apache Airflow provides a simple method to run bash commands in your workflow.

```
t1 = BashOperator(
        task_id=t1,
        dag=dag,
        bash_command='echo "Text"'
        )
```

Apache Airflow Python Operator

The Airflow PythonOperator Calls an arbitrary python function.

```
def print_string():
    print("Test String")

t2 = PythonOperator(
        task_id="t3",
        dag=dag,
        python_callable=print_string,
      )
```

Apache Airflow PostgresOperator

The Postgres Operator interface defines tasks that interact with the PostgreSQL database. It will be used to create tables, remove records, insert records, and more.

```
with DAG(
    dag_id="postgres_operator_dag",
    start_date=datetime.datetime(2021, 10, 11),
    schedule_interval="@once",
    catchup=False,
) as dag:

t4= PostgresOperator(
        task_id="t4",
        sql="""
            CREATE TABLE IF NOT EXISTS pet (
            table_id SERIAL PRIMARY KEY,
            name VARCHAR NOT NULL,
            table_type VARCHAR NOT NULL,
            birth_date DATE NOT NULL,
            OWNER VARCHAR NOT NULL);
          """,
    )
```

Apache Airflow SSH Operator

SSHOperator is used to execute commands on a given remote host using the ssh_hook.

```
t5 = SSHOperator(
        task_id='SSHOperator',
        ssh_conn_id='ssh_connectionid',
        command='echo "Text from SSH Operator"'
    )
```

Apache Airflow Spark Operators

Apache Spark is a general-purpose cluster computing solution that is quick and scalable.

```
t9= SparkJDBCOperator(
    cmd_type='spark_to_jdbc',
    jdbc_table="foo",
    spark_jars="${SPARK_HOME}/jars/postgresql-42.2.12.jar",
    jdbc_driver="org.postgresql.Driver",
    metastore_table="bar",
    save_mode="append",
    task_id="t9",
)
```

Building Automated Data Pipelines with Airflow

Apache Airflow Xcom:

Xcom, an abbreviation of “cross-communication”, lets you pass messages between tasks. Although tasks are suggested to be atomic and not to share resources between them, sometimes exchanging messages between them become useful. Xcom has two side; pusher and puller. The pusher sends a message and the puller receives the message.

Apache Airflow SubDAG:

SubDAG is a pluggable module DAG that can be inserted into a parent DAG. This is useful when you have repeating processes within a DAG or among other DAGs.

One note about SubDag is that by default it uses SequentialExecutor; that means all processes will be executed in sequence regardless of an absence of a task dependency.

Apache Airflow Catchup:

Airflow will run any past scheduled intervals that have not been run. In order to avoid catchup, we need to explicitly pass the parameter catchup=False in the DAG definition.

Apache Airflow Backfill:

If for some reason we want to re-run DAGs on certain schedules manually we can use the following CLI command to do so.

```
airflow backfill -s <START_DATE> -e <END_DATE> --rerun_failed_tasks -
B <DAG_NAME>
```

This will execute all DAG runs that were scheduled between START_DATE & END_DATE irrespective of the value of the catchup parameter in airflow.cfg.

Apache Airflow Variables

Airflow Variables are useful for storing and retrieving data at runtime while avoiding hard-coding values and duplicating code in our DAGs. They are stored as Key-Value pair in Airflow metadata database. To set and get variables in airflow

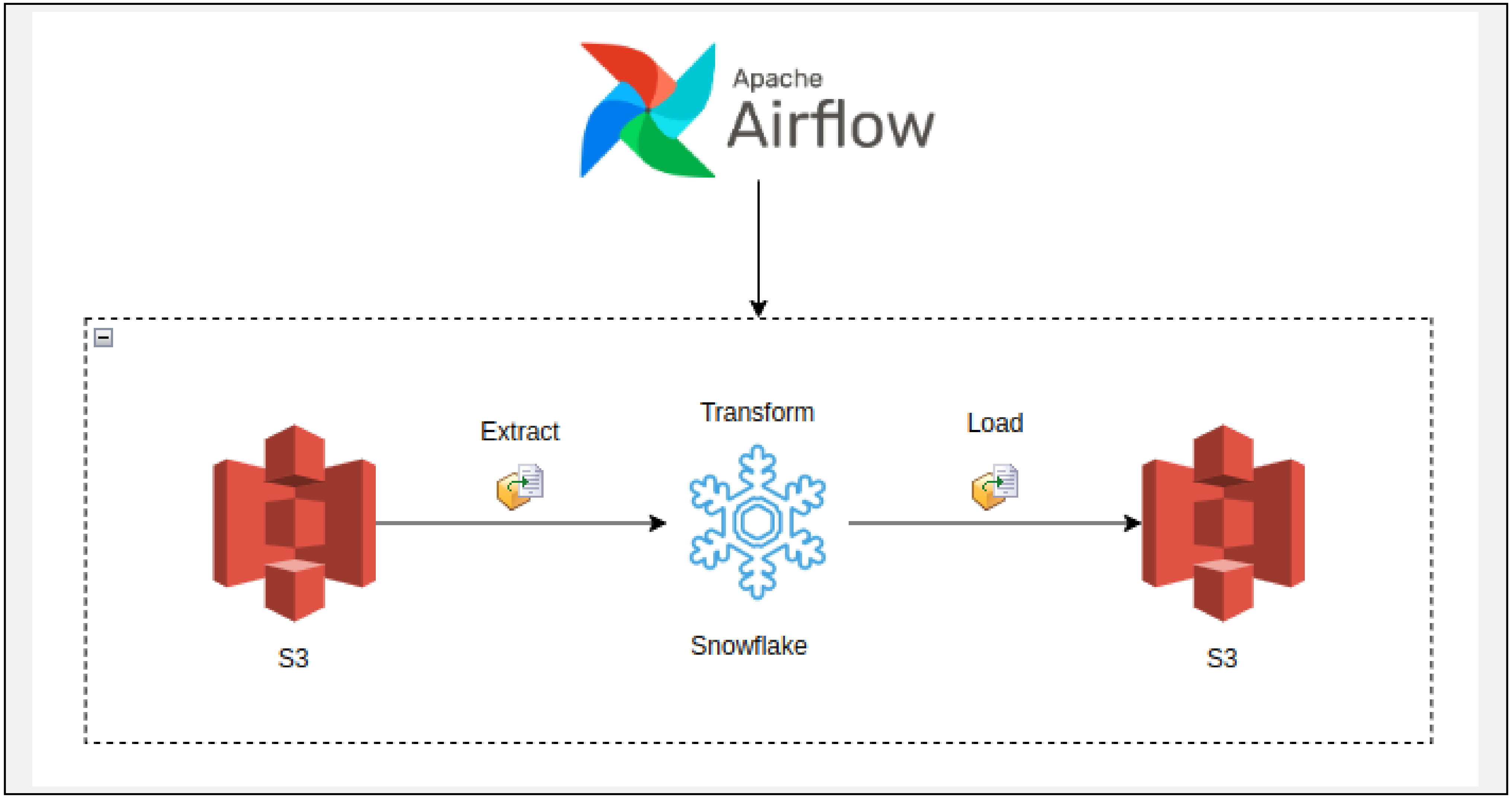
Airflow UI : Admin > Variables. Here we can set variable values individually or import a json file with list of variables.

Command line interface: The airflow variables high level command can be used to work with airflow variables. To set a variable value we can use the command airflow variables set ourvarname ourvalue

Code: We can use the .set and .get methods on airflow.model.Variable to set and get variable values in our DAG code.

Example of Building Automated Data Pipelines with Airflow:

Here, AWS S3 is the storage layer, Snowflake is the Cloud Data warehouse, and Apache airflow is the data pipeline orchestration tool.


Apache Airflow
Extract
Transform
Load
S3
Snowflake
S3