Introduction to Data Warehousing and Data Lakes

Data Lake?

A Data Lake is a storage repository that can store a large amount of structured, semi-structured, and unstructured data. It is a place to store every type of data in its native format with no fixed limits on account size or file. It offers a large amount of data quantity for increased analytical performance and native integration.

A data lake is a central location that holds a large amount of data in its native, raw format. Compared to a hierarchical data warehouse, which stores data in files or folders, a data lake uses a flat architecture and object storage to store the data. Object storage stores data with metadata tags and a unique identifier, which makes it easier to locate and retrieve data across regions, and improves performance. By leveraging inexpensive object storage and open formats, data lakes enable many applications to take advantage of the data.

Data lake challenges

Despite their pros, many of the promises of data lakes have not been realized due to the lack of some critical features: no support for transactions, no enforcement of data quality or governance, and poor performance optimizations. As a result, most of the data lakes in the enterprise have become data swamps.

1.reliability issues

Without the proper tools in place, data lakes can suffer from data reliability issues that make it difficult for data scientists and analysts to reason about the data. These issues can stem from difficulty combining batch and streaming data, data corruption and other factors.

2.Slow performance

As the size of the data in a data lake increases, the performance of traditional query engines has traditionally gotten slower. Some of the bottlenecks include metadata management, improper data partitioning and others.

3.Lack of security features

Data lakes are hard to properly secure and govern due to the lack of visibility and ability to delete or update data. These limitations make it very difficult to meet the requirements of regulatory bodies.

Data Warehouse?

Data Warehouse is a blend of technologies and components for the strategic use of data. It collects and manages data from varied sources to provide meaningful business insights. It is the electronic storage of a large amount of information designed for query and analysis instead of transaction processing. It is a process of transforming data into information.

Difference between Data Lake and Data Warehouse

key differences between data lakes vs data warehouse:

Parameters	Data Lake	Data Warehouse
Storage	In the data lake, all data is kept irrespective of the source and its structure. Data is kept in its raw form. It is only transformed when it is ready to be used.	A data warehouse will consist of data that is extracted from transactional systems or data which consists of quantitative metrics with their attributes. The data is cleaned and transformed
History	Big data technologies used in data lakes is relatively new.	Data warehouse concept, unlike big data, had been used for decades.
Data Capturing	Captures all kinds of data and structures, semi-structured and unstructured in their original form from source systems.	Captures structured information and organizes them in schemas as defined for data warehouse purposes
Data Timeline	Data lakes can retain all data. This includes not only the data that is in use but also data that it might use in the future. Also, data is kept for all time, to go back in time and do an analysis.	In the data warehouse development process, significant time is spent on analyzing various data sources.
Users	Data lake is ideal for the users who indulge in deep analysis. Such users include data scientists who need advanced analytical tools with capabilities such as predictive modelling and statistical analysis.	The data warehouse is ideal for operational users because of being well structured, easy to use and understand.
Storage Costs	Data storing in big data technologies are relatively inexpensive then storing data in a data warehouse.	Storing data in Data warehouse is costlier and time-consuming.
Task	Data lakes can contain all data and data types; it empowers users to access data prior the process of transformed, cleansed and structured.	Data warehouses can provide insights into pre-defined questions for pre-defined data types.
Processing time	Data lakes empower users to access data before it has been transformed, cleansed and structured. Thus, it allows users to get to their result more quickly compares to the traditional data warehouse.	Data warehouses offer insights into pre-defined questions for pre-defined data types. So, any changes to the data warehouse needed more time.
Position of Schema	Typically, the schema is defined after data is stored. This offers high agility and ease of data capture but requires work at the end of the process	Typically schema is defined before data is stored. Requires work at the start of the process, but offers performance, security, and integration.
Data processing	Data Lakes use of the ELT (Extract Load Transform) process.	Data warehouse uses a traditional ETL (Extract Transform Load) process.

Complain	Data is kept in its raw form. It is only transformed when it is ready to be used.	The chief complaint against data warehouses is the inability, or the problem faced when trying to make change in in them.
Key Benefits	They integrate different types of data to come up with entirely new questions as these users not likely to use data warehouses because they may need to go beyond its capabilities.	Most users in an organization are operational. These types of users only care about reports and key performance metrics.

1. **<u>Building an ETL Pipeline with Batch Processing:</u>**

   Designing a data warehousing solution for an ETL (Extract, Transform, Load) data pipeline involves several key considerations to ensure efficiency, scalability, and reliability. Here's a comprehensive guide to help you design a robust data warehousing solution for your ETL pipeline:

   1. Define Requirements:

   a. Business Requirements:

   - Understand the business goals and reporting needs.
   - Identify key performance indicators (KPIs) and reporting frequency.

   b. Technical Requirements:

   - Assess data volume, velocity, and variety.
   - Define data retention policies.
   - Consider security and compliance requirements.

2. Choose a Data Warehouse Platform:

a. Traditional vs. Cloud-Based:

Evaluate whether to use traditional on-premises data warehousing solutions or cloud-based platforms (e.g., Amazon Redshift, Google BigQuery, Snowflake).

b. Scalability:

Ensure the chosen platform can scale horizontally and vertically to accommodate growing data volumes.

c. Performance:

Assess the performance capabilities of the data warehouse for complex queries and large datasets.

3. Data Modeling:

a. Star Schema or Snowflake Schema:

Choose an appropriate dimensional modeling technique based on reporting requirements.

b. Fact and Dimension Tables:

Design fact tables for measurable data and dimension tables for descriptive attributes.

c. Indexing:

Implement proper indexing for fast query performance.

4. ETL Design:

a. Extract:

- Identify source systems and extract relevant data.
- Consider incremental extraction for efficiency.

b. Transform:

Apply necessary transformations to clean, enrich, and standardize data.

Implement data quality checks and error handling.

c. Load:

- Decide on loading strategies (e.g., full load, incremental load).
- Implement mechanisms for handling late-arriving data.

5. Metadata Management:

a. Metadata Repository:

Establish a centralized repository for storing metadata related to source systems, transformations, and data lineage.

b. Data Lineage:

Track the movement of data from source to destination to enhance transparency.

6. Monitoring and Logging:

a. Logging:

Implement comprehensive logging to track ETL job execution details and errors.

b. Monitoring:

Set up monitoring tools to proactively detect and address issues.

7. Security:

a. Access Control:

Implement role-based access control to restrict data access based on user roles.

b. Encryption: Encrypt sensitive data during transit and at rest.

8. Testing and Validation:

a. Unit Testing: Perform unit tests for individual transformations and data loading processes.

b. Integration Testing: Test the end-to-end ETL process with sample datasets.

9. Performance Tuning:

a. Query Optimization: Optimize SQL queries for efficient execution.

b. Indexing and Partitioning: Utilize indexing and partitioning strategies for better performance.

10. Disaster Recovery and Backup:

a. Backup Strategy:

Establish a robust backup and recovery strategy to prevent data loss.

b. Disaster Recovery Plan:

Develop a comprehensive plan to recover from system failures or disasters.

11. Documentation:

a. Technical Documentation:

Document the entire data warehousing architecture, ETL processes, and data models.

b. Runbooks:

Create runbooks for routine maintenance tasks and issue resolution.

12. Continuous Improvement:

a. Performance Monitoring:

Continuously monitor and analyse system performance for optimization opportunities.

b. Feedback Loop:

Establish a feedback loop with end-users to address evolving reporting needs.

Designing Data Lakes for an ETL Data Pipeline

Designing a data lake for an ETL (Extract, Transform, Load) data pipeline involves creating a scalable and flexible storage infrastructure capable of handling diverse data types. Here's a guide to help you design an effective data lake for your ETL processes:

1. Define Requirements:

a. Business Requirements:

- Understand the business goals, reporting needs, and use cases for data analysis.
- Identify the types of data to be stored in the data lake.

b. Technical Requirements:

- Assess data volume, velocity, and variety.
- Consider security, compliance, and data governance requirements.

2. Choose a Data Lake Platform:

a. Cloud-Based vs. On-Premises:

- Evaluate whether to use a cloud-based data lake (e.g., Amazon S3, Azure Data Lake Storage) or an on-premises solution.

b. Scalability:

- Ensure the chosen platform can scale horizontally to handle increasing data volumes.

c. Performance:

- Assess the performance capabilities of the data lake for large-scale data processing.

3. Data Organization and Storage:

a. Raw Data Storage:

- Store raw, unprocessed data in its native format for flexibility and future use.

b. Folder Structure:

- Design a logical and organized folder structure to categorize data based on source, type, or purpose.

c. Data Partitioning:

- Implement data partitioning to enhance query performance, especially for large datasets.

4. Data Formats:

a. Parquet, Avro, ORC:

- Choose columnar storage formats for optimized query performance.

b. Compression:

- Apply compression techniques to reduce storage costs and improve data transfer speed.

5. Metadata Management:

a. Metadata Catalog:

- Establish a metadata catalog to store information about the data, including schema, lineage, and quality.

b. Data Lineage:

- Track the movement of data within the data lake for transparency and traceability.

6. Security:

a. Access Control:

- Implement fine-grained access control to restrict data access based on user roles.

b. Encryption:

- Encrypt data at rest and in transit to ensure data security.

7. ETL Design:

a. Extract:

- Identify sources and ingest data into the data lake.
- Consider batch and streaming data ingestion based on requirements.

b. Transform:

- Apply transformations as needed for analytics and downstream processing.
- Leverage distributed processing frameworks for scalability.

c. Load:

- Store transformed data back into the data lake.
- Implement data versioning for traceability and rollback.

8. Data Governance:

a. Data Quality:

- Implement data quality checks to ensure data accuracy and consistency.

b. Metadata Tagging:

- Use metadata tags to classify and label data for governance and compliance.

9. Disaster Recovery and Backup:

a. Backup Strategy:

- Establish a backup and recovery strategy to prevent data loss.

b. Versioning:

- Implement versioning mechanisms for data to facilitate recovery and auditing.

10. Monitoring and Logging:

a. Logging:

- Implement comprehensive logging to track ETL job execution details, errors, and performance metrics.

b. Monitoring:

- Set up monitoring tools to detect and respond to issues promptly.

11. Documentation:

a. Technical Documentation:

- Document the data lake architecture, ETL processes, and data organization.

b. Data Dictionary:

- Create a data dictionary to define and describe the metadata used in the data lake.

12. Continuous Improvement:

a. Performance Optimization:

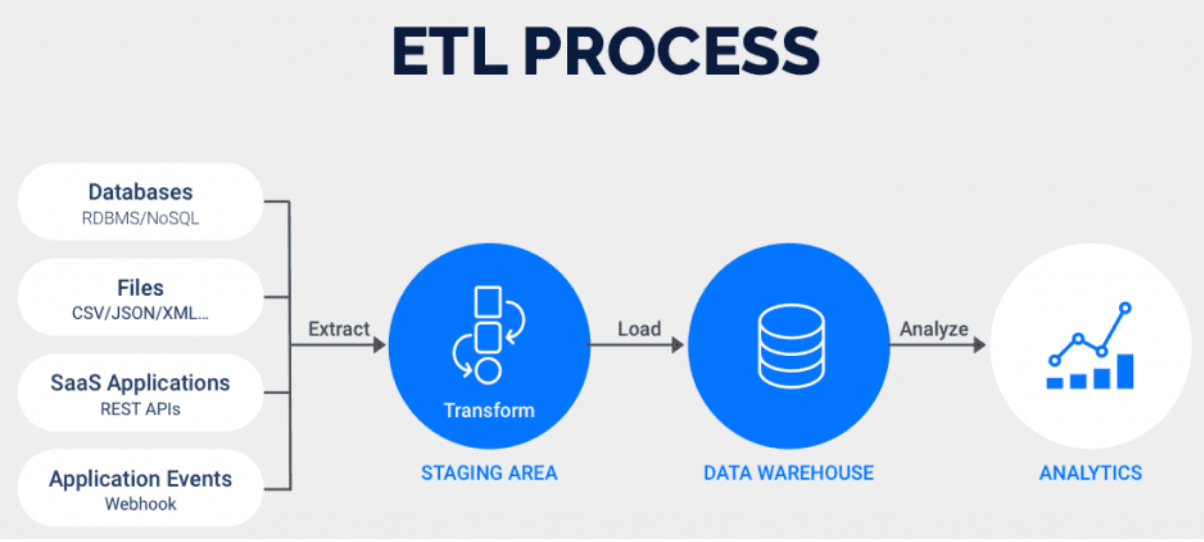
- Continuously monitor and optimize the performance of data lake queries and processes.

b. Feedback Loop:

- Establish a feedback loop with data users to adapt to changing requirements.

ELT vs ETL:

1.ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform) are both data integration processes used to move and transform data from source systems to target systems.



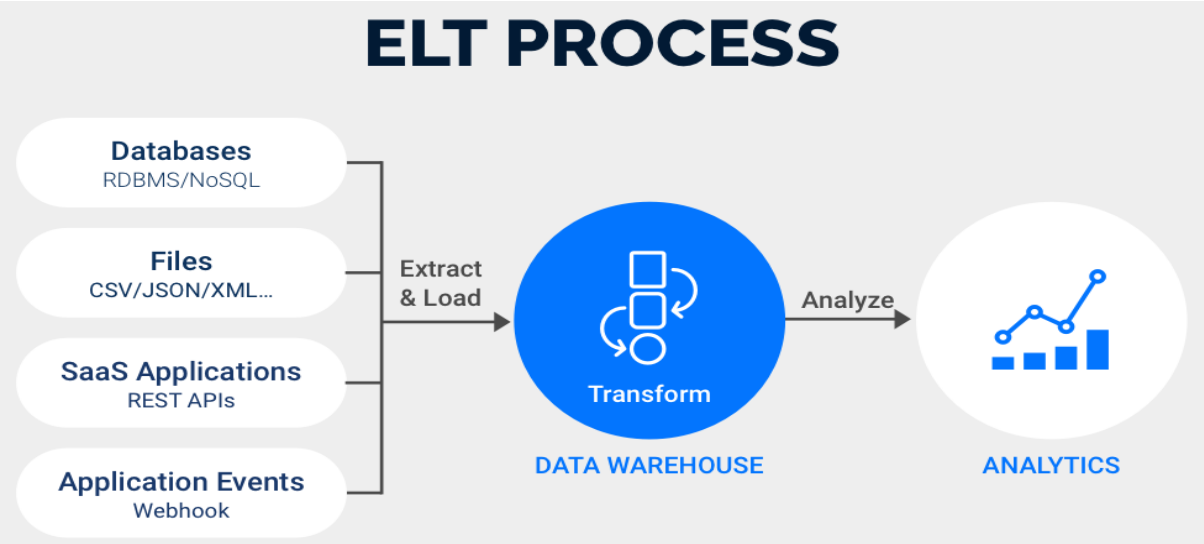
ETL (Extract, Transform, Load):

1. **Extract:**
    - Data is extracted from source systems, which could include databases, applications, or external data sources.
2. **Transform:**
    - Extracted data is then transformed into the desired format or structure. Transformations may include cleaning, filtering, aggregating, and enriching the data.
3. **Load:**
    - Transformed data is loaded into the target system, typically a data warehouse or a database optimized for reporting and analytics.

<u>Characteristics of ETL:</u>

- Traditionally associated with data warehousing.
- Transformation occurs before loading data into the target system.
- Well-suited for scenarios where data needs to be cleansed, aggregated, or enriched before being used for reporting or analytics.
- Can be resource-intensive, especially during the transformation phase.

<u>ELT (Extract, Load, Transform):</u>



1. **Extract:**
    - Data is extracted from source systems, similar to the ETL process.
2. **Load:**
    - Extracted data is loaded into the target system without significant transformation. It is often loaded into a staging area or directly into the target data store.

3. **Transform:**
    - Transformation is performed within the target system. This can involve using SQL queries, data manipulation languages, or other processing capabilities of the target environment.

Characteristics of ELT:

- Commonly associated with modern cloud-based data platforms.
- Transformation occurs after loading data into the target system.
- Well-suited for distributed and parallel processing in cloud environments.
- Offers flexibility in terms of using the processing power of the target system for transformations.

Considerations for Choosing Between ETL and ELT:

1. **Data Volume:**
    - ETL is often preferred for large-scale data transformations before loading into a target system.
    - ELT can be more suitable for scenarios where the target system can handle transformations efficiently at scale.
2. **Target System Capabilities:**
    - ETL is useful when the target system lacks the processing power or capabilities for complex transformations.
    - ELT leverages the processing capabilities of modern cloud-based target systems.
3. **Data Latency:**
    - ETL may introduce some latency as data is transformed before loading.
    - ELT can provide near real-time or real-time processing as transformation occurs after loading.
4. **Data Warehousing vs. Data Lakes:**
    - ETL is often associated with data warehousing and structured data.
    - ELT is well-suited for data lakes and environments that store raw, unstructured, or semi-structured data.
5. **ELT is faster than ETL**

Category	ETL	ELT
Definition	Data is extracted from a source system, transformed on a secondary processing server, and loaded into a destination system.	Data is extracted from a source system, loaded into a destination system, and transformed inside the destination system.
Extract	Raw data is extracted using API connectors.	Raw data is extracted using API connectors.
Transform	Raw data is transformed on a processing server.	Raw data is transformed inside the target system.
Load	Transformed data is loaded into a destination system.	Raw data is loaded directly into the target system.
Speed	ETL is a time-intensive process; data is transformed before loading into a destination system.	ELT is faster by comparison; data is loaded directly into a destination system, and transformed in-parallel.
Code-Based Transformations	Performed on secondary server. Best for compute-intensive transformations & pre-cleansing.	Transformations performed in-database; simultaneous load & transform; speed & efficiency.
Maturity	Modern ETL has existed for 20+ years; its practices & protocols are well-known and documented.	ELT is a newer form of data integration; less documentation & experience.

Privacy	Pre-load transformation can eliminate PII (helps for HIPPA).	Direct loading of data requires more privacy safeguards.
Maintenance	Secondary processing server adds to the maintenance burden.	With fewer systems, the maintenance burden is reduced.
Costs	Separate servers can create cost issues.	Simplified data stack costs less.
Requeries	Data is transformed before entering destination system; therefore raw data cannot be requeried.	Raw data is loaded directly into destination system and can be requeried endlessly.
Data Lake Compatibility	No, ETL does not have data lake compatibility.	Yes, ELT does have data lake compatibility.
Data Output	Structured (typically).	Structured, semi-structured, unstructured.
Data Volume	Ideal for small data sets with complicated transformation requirements.	Ideal for large datasets that require speed & efficiency

Fundamentals of Airflow

Apache Airflow is the leading orchestrator for authoring, scheduling, and monitoring data pipelines. Airflow started as an open-source project at Airbnb. In 2015, Airbnb was growing rapidly and struggling to manage the vast quantities of internal data it generated every day.

Airflow provides many benefits, including:

- **Dynamic data pipelines:** In Airflow, pipelines are defined as Python code. Anything you can do in Python, you can do in Airflow.
- **CI/CD for data pipelines:** With all the logic of your workflows defined in Python, it is possible to implement CI/CD processes for your data pipelines.
- **Tool agnosticism:** Airflow can connect to any application in your data ecosystem that allows connections through an API.
- **High extensibility:** For many commonly used data engineering tools, integrations exist in the form of provider packages, which are routinely extended and updated.
- **Infinite scalability:** Given enough computing power, you can orchestrate as many processes as you need, no matter the complexity of your pipelines.
- **Visualization**: The Airflow UI provides an immediate overview of your data pipelines.
- **Stable REST API:** The Airflow REST API allows Airflow to interact with RESTful web services.
- **Ease of use:** With the Astro CLI, you can run a local Airflow environment with only three bash commands.
- **Active and engaged OSS community:** With millions of users and thousands of contributors, Airflow is here to stay and grow.

When to use Airflow

Airflow can be used for almost any batch data pipeline,

Core Airflow concepts

- **DAG**: Directed Acyclic Graph. An Airflow DAG is a workflow defined as a graph, where all dependencies between nodes are directed and nodes do not self-reference. For more information on Airflow DAGs.
- **DAG run**: The execution of a DAG at a specific point in time. A DAG run can be scheduled or manually triggered.
- **Task**: A step in a DAG describing a single unit of work.
- **Task instance**: The execution of a task at a specific point in time.

Airflow components

When working with Airflow, it is important to understand the underlying components of its infrastructure. Knowing which components are available and their purpose can help you develop your DAGs, troubleshoot issues, and run Airflow successfully.

The following Airflow components must be running at all times:

- **Webserver**: A Flask server running with Gunicorn that serves the Airflow UI.
- **Scheduler**: A Daemon responsible for scheduling jobs. This is a multi-threaded Python process that determines what tasks need to be run, when they need to be run, and where they are run.
- **Database**: A database where all DAG and task metadata are stored. This is typically a Postgres database, but MySQL, MsSQL, and SQLite are also supported.
- **Executor**: The mechanism that defines how the available computing resources are used to execute tasks. An executor is running within the scheduler whenever Airflow is up.

Apache Airflow: A Workflow Management Platform

Apache Airflow is a workflow management platform that schedules and monitors the data pipelines.

Airflow helps us build, schedule, and monitor the data pipelines using the python-based framework. It captures data processes activities and coordinates the real-time updates to a distributed environment. Apache Airflow is not a data processing or ETL tool it orchestrates the various tasks in the data pipelines. The data pipeline is a set of several tasks that need to be executed in a given flow with dependencies to achieve the main objective.

We can introduce the dependencies between tasks using graphs in the data pipeline. In graph-based solutions, tasks are represented as nodes and dependencies as directed edges between two tasks. This directed graph may lead to a deadlock situation so it is necessary to make it acyclic. Airflow uses Directed Acyclic Graphs (DAGs) to represent a data pipeline that efficiently executes the tasks as per graph flow.

Features of Apache Airflow:

- It is an Open-Source tool.
- It is a Batch-oriented framework.
- It uses Directed Acyclic Graph(DAG) for creating workflows.
- Web Interface for monitoring pipeline.
- Python-based tool
- Scheduling data pipelines.
- Robust Integrations with cloud platforms.

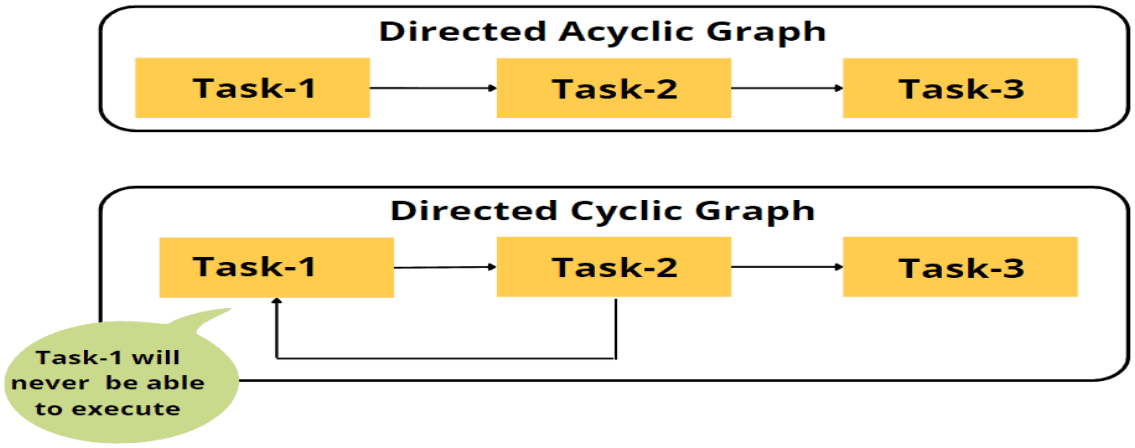
Principles of Apache Airflow

- **Extensible:** In Airflow, we can define operators and extend libraries.

- **Scalable:** In Airflow, we can scale it to infinity.
- **Dynamic:** In Airflow, we can execute the pipelines dynamically.
- **Elegant:** In Airflow, we can create lean and explicit pipelines.

Why Directed Acyclic Graphs?

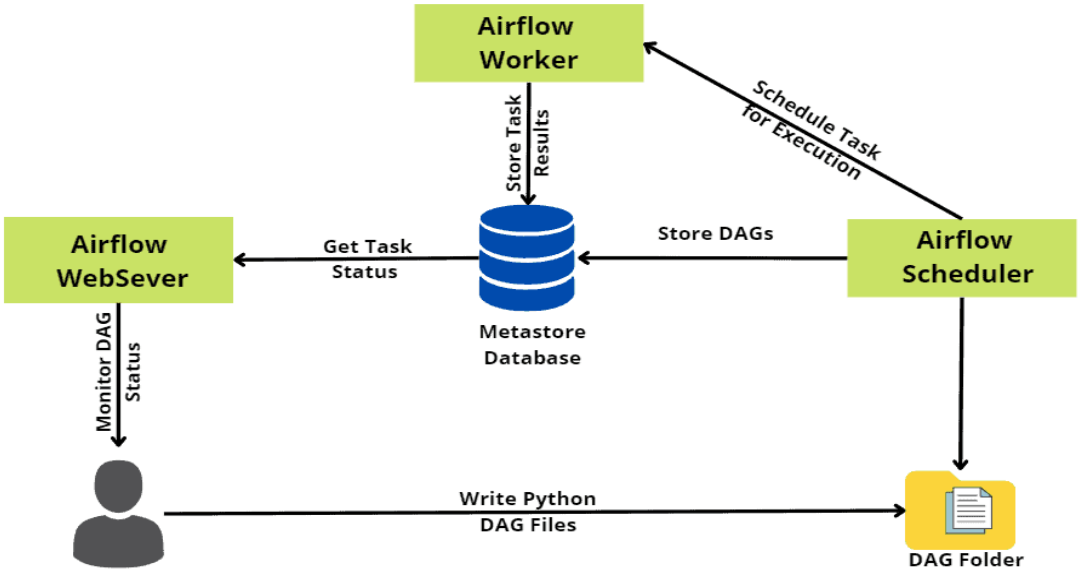
Directed Acyclic Graph (DAG) comprises directed edges, nodes, and no loop or cycles. Acyclic means there are no circular dependencies in the graph between tasks. Circular dependency creates a problem in task execution. for example, if task-1 depends upon task-2 and task-2 depends upon task-1 then this situation will cause deadlock and leads to logical inconsistency.



Data Pipeline Execution in Airflow

In Airflow, we can write our DAGs in python and schedule them for execution at regular intervals such as every minute, every hour, every day, every week, and so on. Airflow is comprising four main components:

- **Airflow scheduler:** It parses the DAGs and schedules the task as per the scheduled time. After scheduling, it submits the task for execution to Airflow workers.
- **Airflow workers:** It selects the scheduled tasks for execution. Workers are the main engine that is responsible for performing the actual work.
- **Airflow webserver:** It visualizes the DAGs on the UI web interface and monitors the DAG runs.
- **Metadata Database** – It is used for storing the pipeline task status.



When to use Airflow?

- Implement the data pipelines using Python.
- Scheduling pipelines at regular intervals.
- Backfilling allows us to easily reprocess the historical data.
- Interactive Web Interface for monitoring pipelines and debugging failures.

When not to use Airflow?

- It cannot be deployed with Streaming Data Pipelines.
- Highly Dynamic Pipelines (It has the capability to deal with dynamic pipelines but not highly dynamic)
- No knowledge of programming language.
- Complex for larger use-cases
- Does not support data lineages and data versioning features.