ETL and Data Pipelines with Shell, Airflow and Kafka Notes

Chuks Okoli

05 January, 2024

Week 1

ETL using Shell Scripts

ETL and ELT Processes
ETL stands for Extract, Transform, and Load. ETL is an automated data pipeline engineering methodology, whereby data is acquired and prepared for subsequent use in an analytics environment, such as a data warehouse or data mart. ETL refers to the process of curating data from multiple sources, conforming it to a unified data format or structure, and then loading the transformed data into its new environment.

- E = Extraction: Extraction process obtains or reads the data from one or more sources
  - Some common methods include: Web scraping, where data is extracted from web pages using applications such as Python or R to parse the underlying HTML code
  - Using APIs to programmatically connect to data and query it
- T = Transformation: Transformation process wrangles the data into a format that is suitable for its destination and its intended use. Transformation is done in an intermediate working environment called a "staging area" and can include any of the following kinds of processes:
  - Cleaning: fixing errors or missing values
  - Filtering: selecting only what is needed
  - Joining disparate data sources: merging related data
  - Normalizing: converting data to common units
  - Data Structuring: converting one data format to another, such as JSON, XML, or CSV to database tables
  - Feature engineering: such as creating KPIs for dashboards or machine learning
  - Anonymizing and Encrypting: ensuring privacy and security
  - Sorting: ordering the data to improve search performance
  - Aggregating: summarizing granular data
  - Formatting and data typing: making the data compatible with its destination.
- L = Load: Loading takes the transformed data and loads it into its new environment (.e.g., database, data warehouses or data mart), ready for visualization, exploration, further transformation, and modelling
  - Data loading techniques
    - Full
    - Incremental
    - Scheduled
    - On-demand
    - Batch and stream
    - Push and pull
    - Parallel and serial

The ETL process is shown in **Fig. 1**.

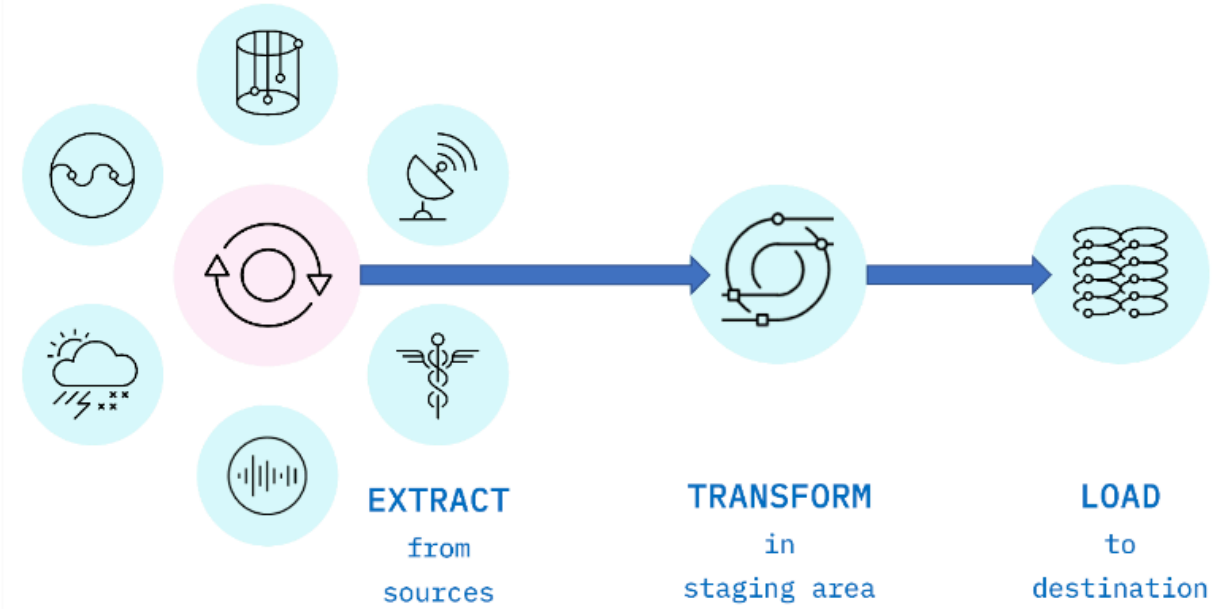


Figure 1: The ETL process

Summary of ETL and ELT Processes

- ETL stands for Extract, Transform, and Load
- Loading means writing the data to its destination environment
- Cloud platforms are enabling ELT to become an emerging trend
- The key differences between ETL and ELT include the place of transformation, flexibility, Big Data support, and time-to-insight
- There is an increasing demand for access to raw data that drives the evolution from ETL, which is still used, to ELT, which enables ad-hoc, self-serve analytics
- Data extraction often involves advanced technology including database querying, web scraping, and APIs
- Data transformation, such as typing, structuring, normalizing, aggregating, and cleaning, is about formatting data to suit the application
- Information can be lost in transformation processes through filtering and aggregation
- Data loading techniques include scheduled, on-demand, and incremental
- Data can be loaded in batches or streamed continuously

Week 2

ETL Workflows as Data Pipelines

Traditionally, the overall accuracy of the ETL workflow has been a more important requirement than speed, although efficiency is usually an important factor in minimizing resource costs. To boost efficiency, data is fed through a *data pipeline* in smaller packets (see **Fig. 2**). While one packet is being extracted, an earlier packet is being transformed, and another is being loaded. In this way, data can keep moving through the workflow without interruption. Any remaining bottlenecks within the pipeline can often be handled by parallelizing slower tasks.

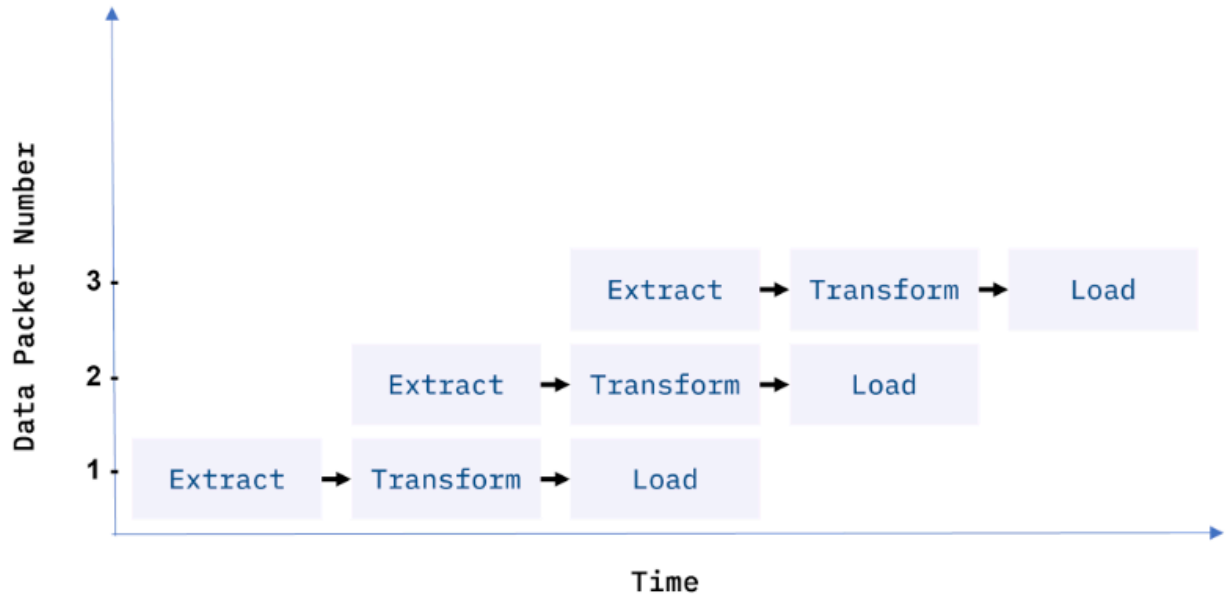


Figure 2: ETL Workflows as Data Pipelines

With conventional ETL pipelines, data is processed in *batches*, usually on a repeating schedule that ranges from hours to days apart. For example, records accumulating in an Online Transaction Processing System (OLTP) can be moved as a daily batch process to one or more Online Analytics Processing (OLAP) systems where subsequent analysis of large volumes of historical data is carried out.

Staging Areas

ETL pipelines are frequently used to integrate data from disparate and usually siloed systems within the enterprise. These systems can be from different vendors, locations, and divisions of the company, which can add significant operational complexity. As an example, (see **Fig. 3**) a cost accounting OLAP system might retrieve data from distinct OLTP systems utilized by the separate payroll, sales, and purchasing departments.

ETL pipelines are frequently used to integrate data from disparate and usually *siloed* systems within the enterprise. These systems can be from different vendors, locations, and divisions of the company, which can add significant operational complexity. As an example, (see **Fig. 3**) a cost accounting OLAP system might retrieve data from distinct OLTP systems utilized by the separate payroll, sales, and purchasing departments.

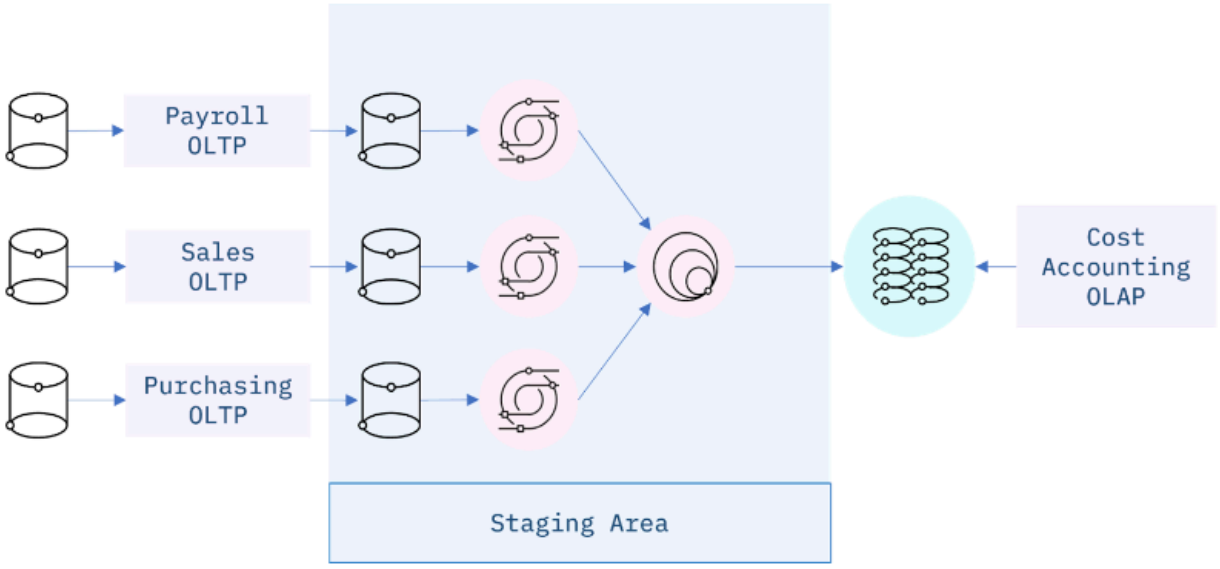


Figure 3: An ETL data integration pipeline concept for a Cost Accounting OLAP

ETL Workflows as DAGs

ETL workflows can involve considerable complexity. By breaking down the details of the workflow into individual tasks and dependencies between those tasks, one can gain better control over that complexity. Workflow orchestration tools such as Apache Airflow do just that.

Airflow represents your workflow as a directed acyclic graph (DAG). A simple example of an Airflow DAG is illustrated in **Fig. 4**. Airflow tasks can be expressed using predefined templates, called operators. Popular operators include Bash operators, for running Bash code, and Python operators for running Python code, which makes them extremely versatile for deploying ETL pipelines and many other kinds of workflows into production.

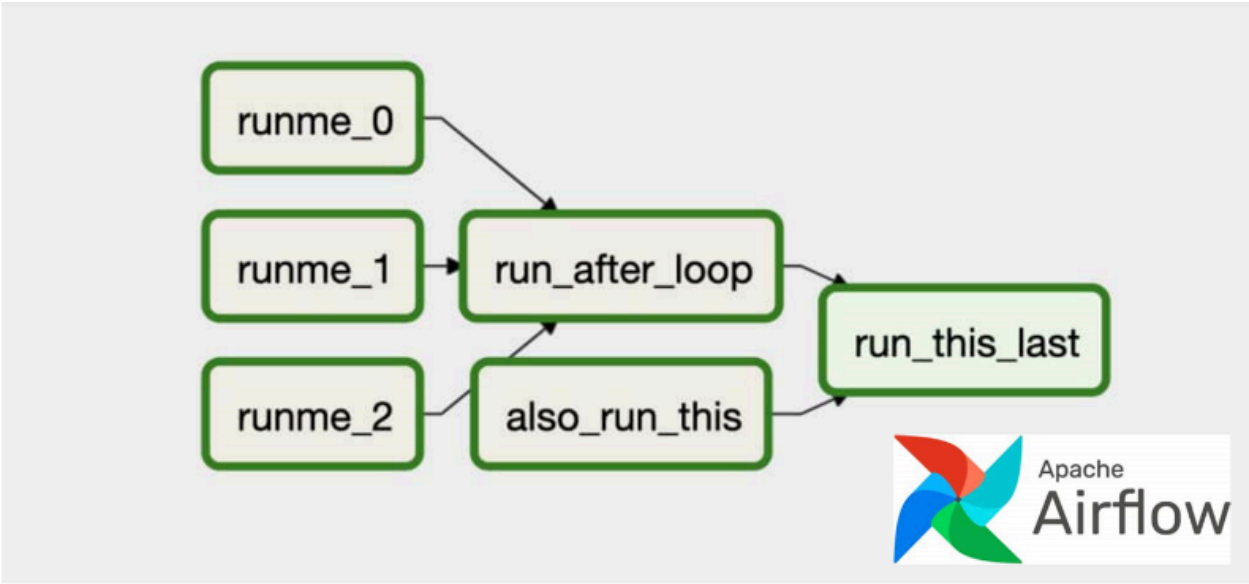


Figure 4: An Apache Airflow DAG representing a workflow

An Introduction to Data Pipelines

What is a data pipeline

Data pipelines are pipelines that specifically move or modify data. The purpose of a data pipeline is to move data from one place or form to another. A data pipeline is a system which extracts data and passes it along to optional transformation stages for final loading. This includes low-level hardware architectures, but our focus here is on data pipelines as architectures driven by software processes, such as commands, programs, and processing threads. The simple Bash pipe command in Linux can be used as the 'glue' that connects such processes together.

We can think of data flowing through a pipeline in the form of data packets (see **Fig. 5**), a term which we will use to broadly refer to units of data. Packets can range in size from a single record, or event, to large collections of data. Here, we have data packets queued for ingestion to the pipeline. The length of the data pipeline represents the time it takes for a single packet to traverse the pipeline. The arrows between packets represent the throughput delays, or the times between successive packet arrivals.

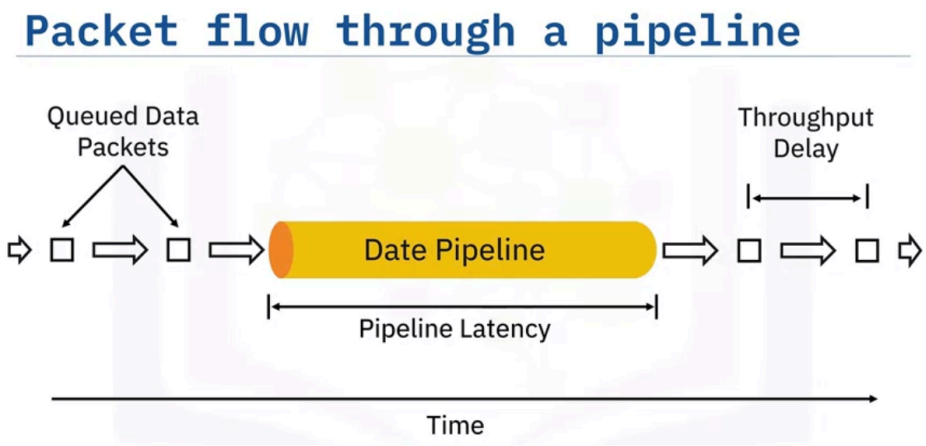


Figure 5: Packet flow through a pipeline

There are two key performance considerations regarding data pipelines. The first is *latency*, which is the total time it takes for a single packet of data to pass through the pipeline as shown in **Fig. 6**. Equivalently, latency is the sum of the individual times spent during each processing stage within the pipeline. Thus overall latency is limited by the slowest process in the pipeline. For example, no matter how fast your internet service is, the time it takes to load a web page will be decided by the server's speed. The second performance consideration is called *throughput*. It refers to how much data can be fed through the pipeline per unit of time. Processing larger packets per unit of time increases throughput.

Some of the applications of data pipelines from the multitude of use cases includes:

- The simplest pipeline is one which has no transformations and is used to copy data from one location to another, as in file backups.
- Integrating disparate raw data sources into a data lake.
- Moving transactional records to a data warehouse.
- Streaming data from IoT devices to make information available in the form of dashboards or alerting systems.
- Preparing raw data for machine learning development or production.
- Message sending and receiving, such as with email, SMS, or online video meetings.

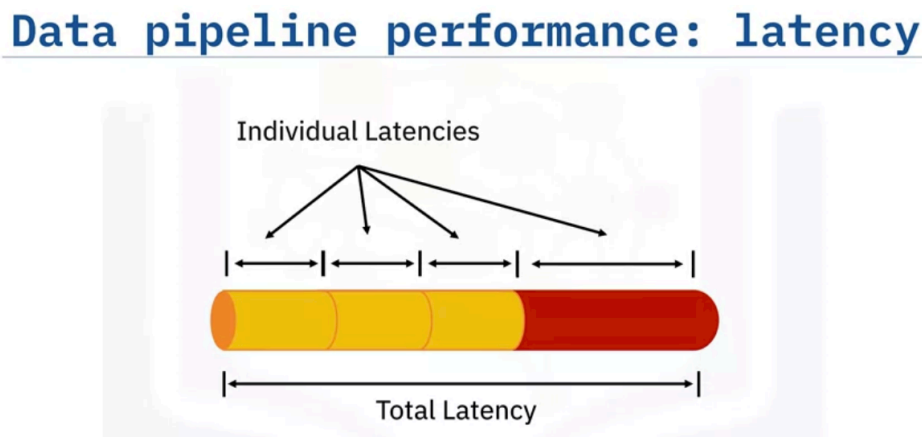
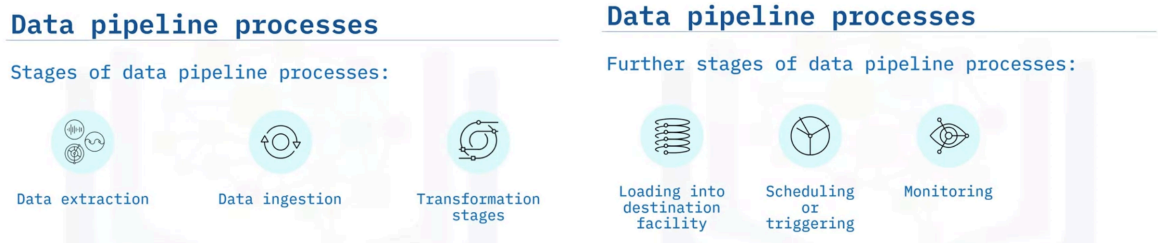


Figure 6: Data pipeline performance: latency

Key Data Pipeline Processes
Data pipeline processes typically have the following stages in common as shown in **Fig. 7**:

- Extraction of data from one or more data sources.
- Ingestion of the extracted data into the pipeline.
- Optional data transformation stages within the pipeline
- Loading of the data into a destination facility.
- A mechanism for scheduling or triggering jobs to run.
- Monitoring the entire workflow, and Maintenance and optimization as required to keep the pipeline up and running smoothly.



(a) Stages of data pipeline

(b) Further stages

Figure 7: Data Pipeline Process

The data pipeline needs to be monitored (see **Fig. 8**) once it is in production to ensure data integrity. Some key monitoring considerations include:

- Latency, or the time it takes for data packets to flow through the pipeline.
- Throughput demand, the volume of data passing through the pipeline over time.
- Errors and failures, caused by factors such as network overloading, and failures at the source or destination systems.
- Utilization rate, or how fully the pipeline's resources are being utilized, which affects cost.
- Lastly, the pipeline should also have a system for logging events and alerting administrators when failures occur.

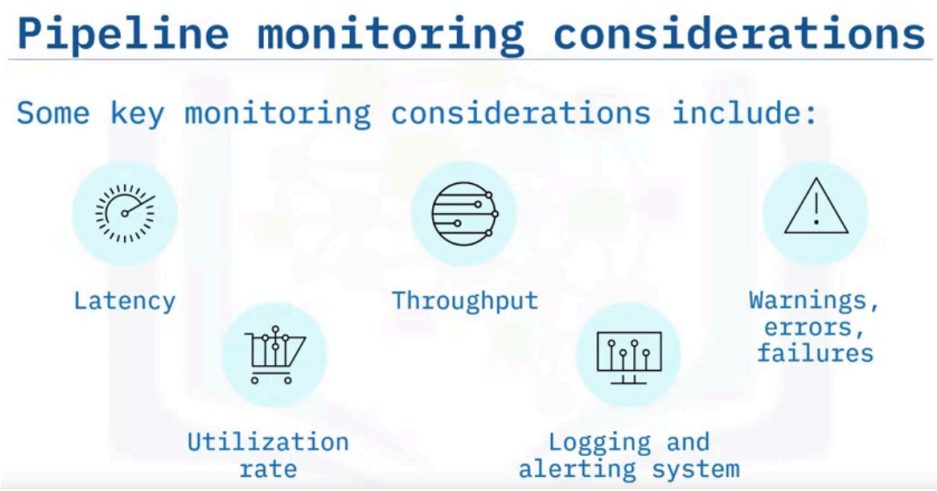


Figure 8: Data pipeline monitoring considerations

Handling unbalanced loads

- Pipelines typically contain bottlenecks
- Slower stages may be parallelized to speed up throughput
- Processes can be replicated on multiple CPUs/cores/threads
- Data packets are then distributed across these channels
- Pipelines that incorporate parallelism are referred to as being dynamic or non-linear, as opposed to "static," which applies to serial pipelines

Stage synchronization

- I/O buffers can help synchronize stages
- An I/O buffer is a holding area for data, placed between processing stages having different or varying delays
- Buffers regulate the flow of data, may improve throughput
- I/O buffers used to distribute loads on parallelized stages

Summary of Introduction to Data Pipelines

- Data pipelines move data from one place, or form, to another
- Data flows through pipelines as a series of data packets
- Latency and throughput are key design considerations for data pipelines
- Data pipeline processes include scheduling or triggering, monitoring, maintenance, and optimization
- Parallelization and I/O buffers can help mitigate bottlenecks
- Batch pipelines extract and operate on batches of data
- Batch processing applies when accuracy is critical, or the most recent data isn't required
- Streaming data pipelines ingest data packets one-by-one in rapid succession
- Streaming pipelines apply when the most current data is needed
- Micro-batch processing can achieve near-real-time data streaming
- Lambda architecture applies when access to earlier data is required, and speed is important
- Examples of streaming data pipelines use cases, such as social media feeds, fraud detection, and real-time product pricing
- Modern data pipeline technologies include schema and transformation support, drag-and-drop GUIs, and security features
- Stream-processing technologies include Apache Kafka, IBM Streams, and SQLStream

Week 3

Building Data Pipelines using Apache Airflow

What is Apache Airflow

- Great open source workflow orchestration tool
- A platform that lets you build and run workflows
- A workflow is represented as a DAG
- Apache Airflow is not a data streaming solution but a workflow manager

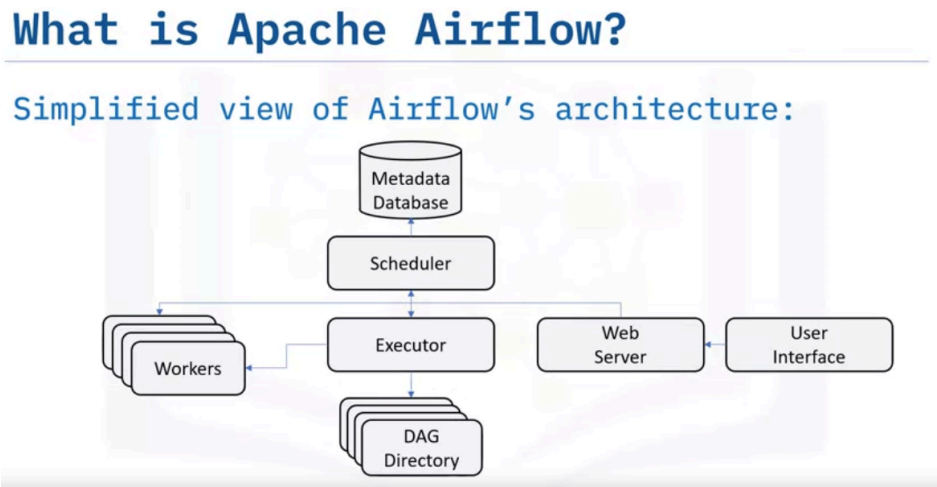


Figure 9: Simplified view of Airflow's architecture

Summary of Using Apache Airflow to build Data Pipelines

- Apache Airflow is scalable, dynamic, extensible, and lean
- The five main features of Apache Airflow are pure Python, useful UI, integration, easy to use, and open source
- A common use case is that Apache Airflow defines and organizes machine learning pipeline dependencies
- Tasks are created with Airflow operators
- Pipelines are specified as dependencies between tasks
- Pipeline DAGs defined as code are more maintainable, testable, and collaborative
- Apache Airflow has a rich UI that simplifies working with data pipelines
- You can visualize your DAG in graph or tree mode
- Key components of a DAG definition file include DAG arguments, DAG and task definitions, and the task pipeline
- The 'schedule_interval' parameter specifies how often to re-run your DAG
- You can save Airflow logs into local file systems and send them to cloud storage, search engines, and log analyzers
- Airflow recommends sending production deployment logs to be analyzed by Elasticsearch or Splunk
- With Airflow's UI, you can view DAGs and task events
- The three types of Airflow metrics are counters, gauges, and timers
- Airflow recommends that production deployment metrics be sent to and analyzed by Prometheus via StatsD

Week 4

Building Streaming Pipelines using Apache Kafka

ETL T

-
-
-
-