Data Engineering Day 19

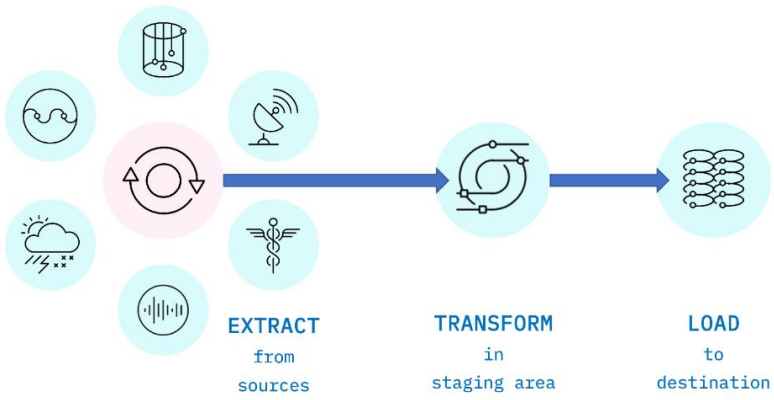
The credit for this course goes to Coursera. Click More

Another link : Azure data Engineer

ETL and Data Pipelines with Shell, Airflow and Kafka

 # ETL & Data Pipelines: Tools and techniques:

- ETL techniques:
  - ETL stands for Extract, Transform, and Load, and refers to the process of curating *(select, organize, and present)* data from multiple sources, conforming it to a unified data format or structure, and loading the transformed data into its new environment.

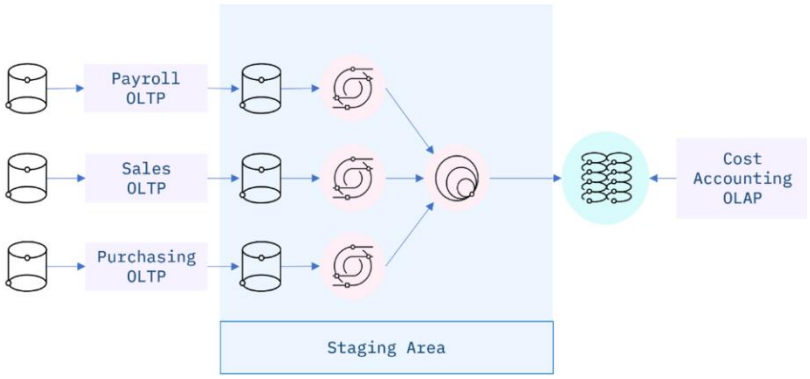


- Extract:
  - Data extraction is the first stage of the ETL process, where data is acquired from various source systems. The data may be completely raw, such as sensor data from IoT devices, or perhaps it is unstructured data from scanned medical documents or company emails.
  - It may be streaming data coming from a social media network or near real-time stock market buy/sell transactions, or it may come from existing enterprise databases and data warehouses.
- Transform
  - The transformation stage is where rules and processes are applied to the data to prepare it for loading into the target system.
  - This is normally done in an intermediate working environment called a “staging area.” Here, the data are cleaned to ensure reliability and conformed to ensure compatibility with the target system.
  - Many other transformations may be applied, including:
  - Cleaning: fixing any errors or missing values
  - Filtering: selecting only what is needed
  - Joining: merging disparate data sources
  - 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: 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

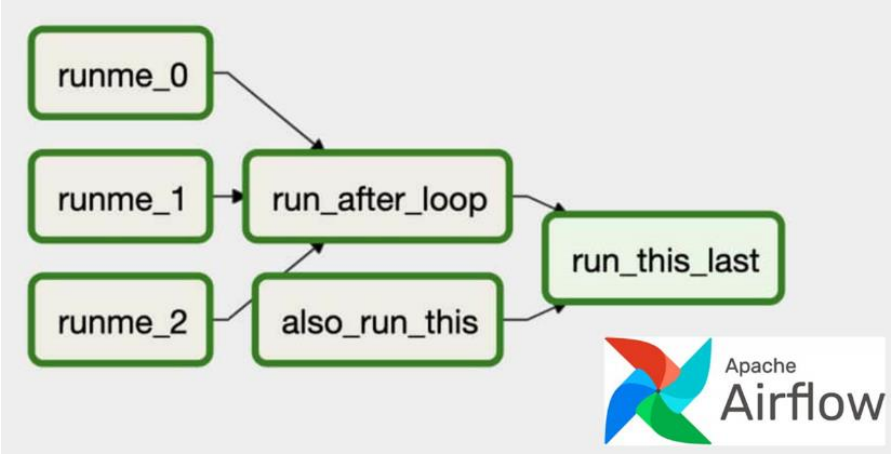
- Load
  - The load phase is all about writing the transformed data to a target system.
  - The system can be as simple as a comma-separated file, which is essentially just a table of data like an Excel spreadsheet.
  - The target can also be a database, which may be part of a much more elaborate system, such as a data warehouse, a data mart, data lake, or some other unified, centralized data store forming the basis for analysis, modeling, and data-driven decision making by business analysts, managers, executives, data scientists, and users at all levels of the enterprise.

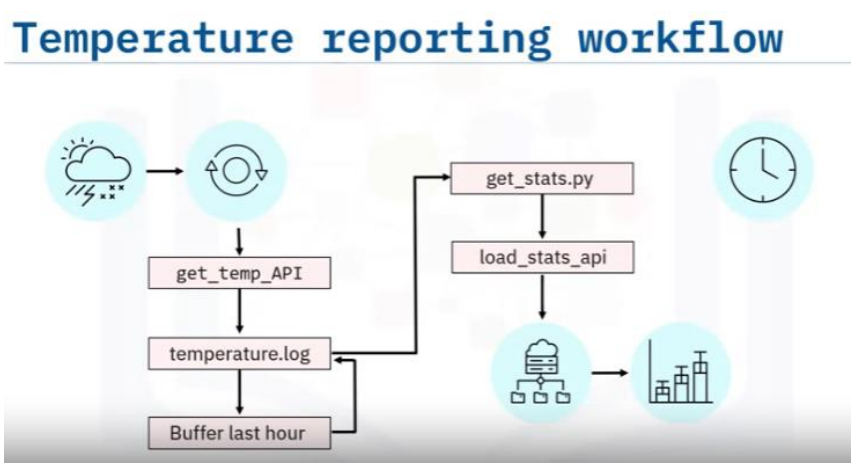
- ETL Workflows as Data Pipelines
  - Generally, an ETL workflow is a well thought out process that is carefully engineered to meet technical and end-user requirements.
  - Traditionally, the overall accuracy of the ETL workflow has been a more important requirement than speed, although efficiency is usually an important factor in minimizing resource costs.
  - To boost efficiency, data is fed through a data pipeline in smaller packets.
  - 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.

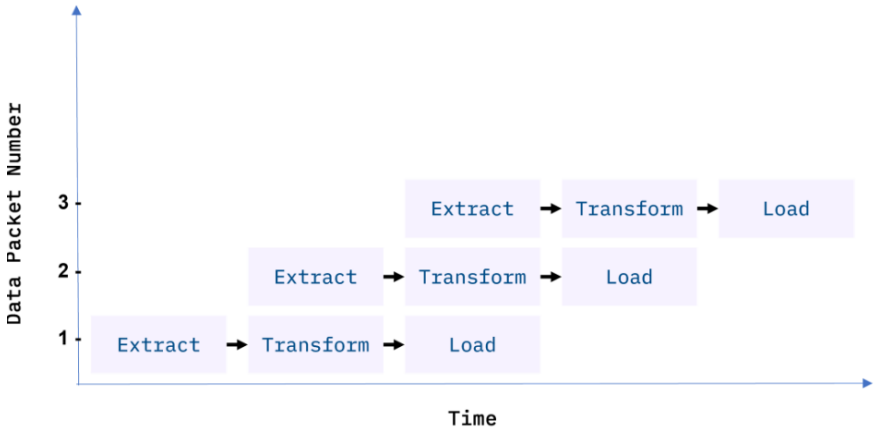
- Staging Areas
  - ETL pipelines are frequently used to integrate data from disparate and usually siloed systems within the enterprise.
  - These systems can be from different vendors, locations, and divisions of the company, which can add significant operational complexity.



- ETL Workflows as DAGs (the flow of the data has the defined directions):
  - 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
  - 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.


runme_0
runme_1
runme_2
run_after_loop
also_run_this
run_this_last
Apache
Airflow


Temperature reporting workflow
get_temp_API
temperature.log
Buffer last hour
get_stats.py
load_stats_api


Data Packet Number
3
2
1
Extract
Transform
Load
Extract
Transform
Load
Extract
Transform
Load
Time

- ETL using shell scripting:

Create an ETL shell script

```
$ touch Temperature_ETL.sh
$ gedit Temperature_ETL.sh
```

```
#! /bin/bash
# Extract reading with get_temp_API
# Append reading to temperature.log
# Buffer last hour of readings
# Call get_stats.py to aggregate the readings
# Load the stats using load_stats_api
```

ETL script: Extract and buffer

```
$ touch temperature.log
```

```
#! /bin/bash
# Extract reading with get_temp_API
# Append reading to temperature.log
get_temp_api >> temperature.log

# Buffer last hour of readings
tail -60 temperature.log > temperature.log
```

ETL script: Transform temperatures

get_stats.py

- Reads temperatures from log file
- Calculates temperature stats
- Writes temperature stats to file
- Input/output filenames specified as command line arguments

```
# Call get_stats.py to aggregate the readings
python3 get_stats.py temperature.log temp_stats.csv
```

Load the transformed data

```
# Load the stats using load_stats_api
load_stats_api temp_stats.csv
```

Set permissions

```
$ chmod +x Temperature_ETL.sh
```

Schedule your ETL job

- Open crontab editor:

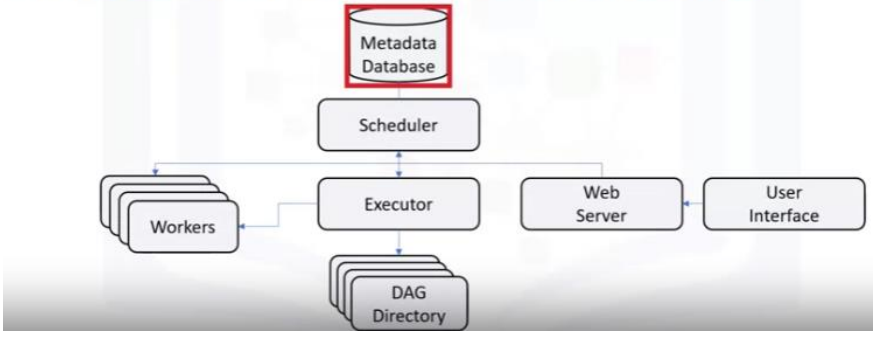
```
$ crontab -e
```

- Enter schedule:

```
1 * * * * path/Temperature_ETL.sh
```

- Close and save
- Your job is now scheduled and running in production

Simplified view of Airflow's architecture:



- Introductions to Data Pipeline:
  - Move or modify data.

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

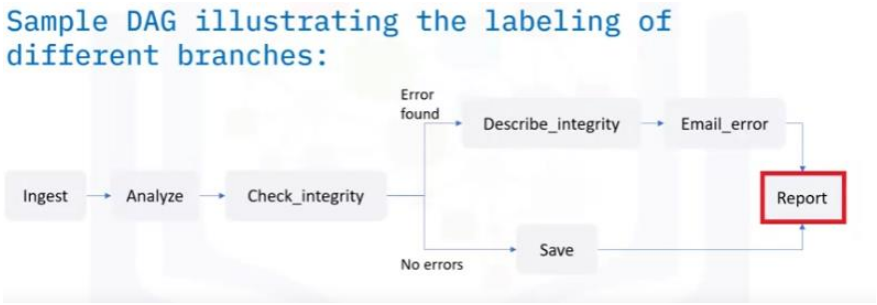
- Purpose: to move data from one place or form to another
- Any system which extracts, transforms, and loads data

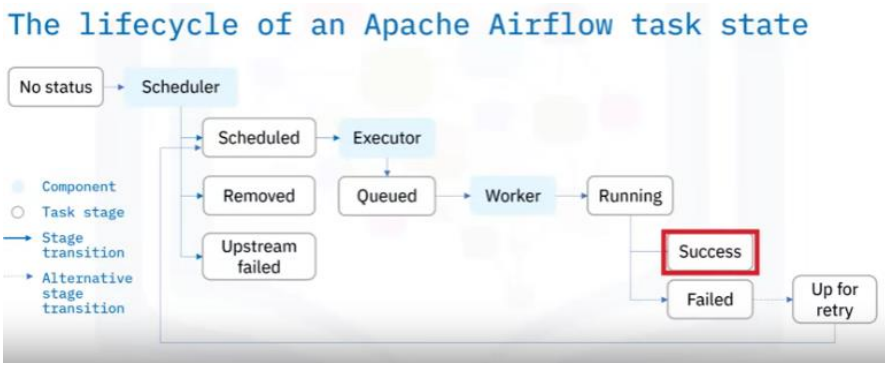
Use cases

Applications of data pipelines

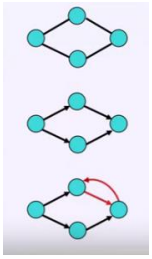
- Backing up files
- Integrating disparate raw data sources into a data lake
- Moving transactional records to a data warehouse
- Streaming data from IoT devices to dashboards
- Preparing raw data for machine learning development or production
- Messaging systems such as email, SMS, video meetings

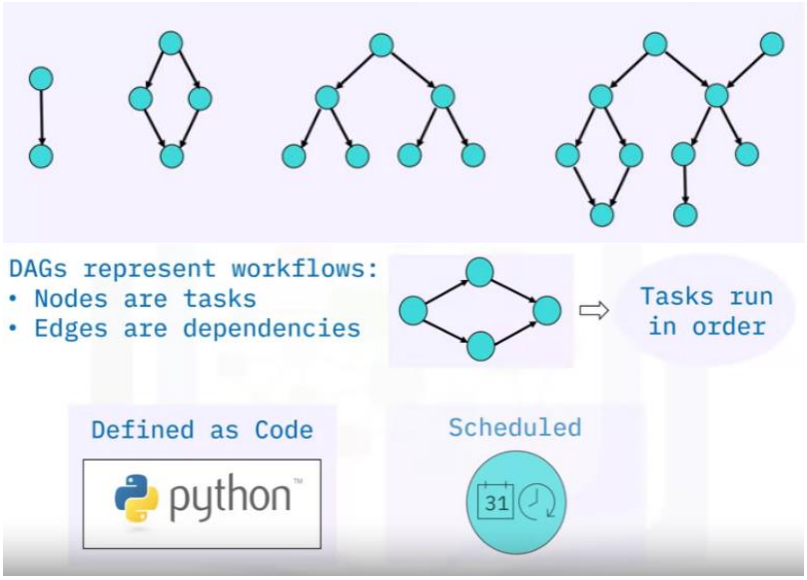
- Apache Airflow Overview:
  - It is data workflow manager.
  - DAGs specify the dependencies between the tasks and order in which to execute them.





- What are DAGs:
  - It is a special kind of graph called Directed Acyclic Graph
  - A graph contains nodes and edges.
  - The workflow of the data in those nodes and the edges are directed in one specific direction.
  - There aren't any loops, or we can also say that the graph flows only in one direction without feedback.
  - The figure below represents more examples of DAGs.





Python script blocks:
- Library imports
- DAG arguments
- DAG definition
- Task definitions
- Task pipeline

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt

default_args = {
    'owner': 'me',
    'start_date': dt.datetime(2021, 7, 28),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5),
}

dag = DAG('simple_example',
          description='A simple example DAG',
          default_args=default_args,
          schedule_interval=dt.timedelta(seconds=5),
)
```

 # Building DAGs using Airflow (steps for writing code for DAGs pipeline):

- Importing python libraries.

Airflow pipeline script

Python script blocks:
- Python library imports

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
import datetime as dt
```

- Writing DAGs arguments.

- DAG arguments

```
default_args = {
    'owner': 'me',
    'start_date': dt.datetime(2021, 7, 28),
    'retries': 1,
    'retry_delay': dt.timedelta(minutes=5),
}
```

- Writing DAGs definitions:

- DAG definition

```
dag=DAG('simple_example',
        description='A simple example DAG',
        default_args=default_args,
        schedule_interval=dt.timedelta(seconds=5),
        )
```

- Tasks definitions:

```
task1 = BashOperator(
    task_id='print_hello',
    bash_command='echo \'Greetings. The date and time are \'',
    dag=dag
)

task2 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag
)
```

- Tasks pipelines: