Data Pipelines

A set of tools and processes that cover the entire journey of the data from source to destination

Tools:

Apache Beam
Airflow
Data Flow

Data Repository

Data repository is a general term used to refer to data that has been collected, organized and isolated.

Relational Database

Data that is organized into a tabular format with rows and columns.
SQL - Structured Query Language is used to querying the RDB

Non-Relational Databases

Data can be stored in a schemaless format
NOSQL - Not Only SQL is used in these databases

Data Warehouse

Consolidates data through ETL process into comprehensive database for analytics and BI (Business Intelligence)

Data Lake

Is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to first structure the data.

ETL (Extract Load Transfer)

Tools:

- *IBM Infosphere*
- *AWS Glue*
- *Skyvia*
- *HEVO*
- *Improvado*
- *Informatica*

Data Integration

Is a discipline comprising the practices, architectural techniques and tools that allow organizations to ingest, transform, combine and provision data across various data types.

Big Data

The 5 V's:

- Velocity
- Volume
- Variety
- Veracity - quality/ origin of data
- Value

Hadoop

Distributed storage and processing of large datasets across clusters of computers

- Node - Single computer
- Clusters - connection of nods
- HDFS - Hadoop Distributed File System for big data storage

Hive

Built on top of the hadoop has high latency thus not suitable for applications requiring fast response

Better suited for data warehousing and tasks such as ETL, reporting and data analysis.

Spark

Is a general - purpose data processing engine designed to extract and process large volumes of data for a wide range of applications.

- In memory processing

Data Architecture

Data Ingestion / collection

- Connect to data source.
- Transfer data from data sources to the data platform in streaming and batch modes.
- Maintain information about the data collected in the metadata repository.

Tools :

Google Cloud DataFlow
IBM Streams
IBM Streaming Analytics on cloud
Amazon Kinesis
Apache Kafka

Data Storage and Integration Layer

- Store data for processing and long term use.
- Transform and merge extracted data either logically or physically
- Make data available for processing in both streaming and batch modes.
- The storage must be reliable, High performance, Scalable Cost-efficient.

Tools:

IBM db2
Microsoft SQL server
My SQL
Oracle Database
Postgresql

Cloud Based Relational Databases

IBM db2 cloud
Amazon RDS
Google Cloud SQL
Microsoft Azure SQL

Non-Relational Databases

MongoDB
Neo 4j

Non-Relational Databases On Cloud

Cloudant
Redis
MongoDB
Cassandra
Neo 4j

Integration Tools

IBM Cloud Pak for Data
IBM Cloud Pak Integration
Talend Data Fabric
Open Studio

Open Source Integration Tools

Dell boomi
SnapLogic

Cloud based Integration tools

Adeptia Integration suite
Google Clouds Cooperation
IBM Application Integration Suite Cloud
Informatica's Integration Cloud

Data Processing

- Read data in batch or streaming modes from storage and apply transformation.
- Support popular querying tools and programming languages.
- Scale to meet processing demand of a growing dataset.
- Provide a way for analysis and data scientists to work with the data in a data platform.

  Tools:

  Spreadsheet
  Trifacta Wrangler
  Open Refine
  Google Dataprep
  Watson Studio Refinery
  Python - pyspark, pandas
  R

- In relation Databases storage and processing can occur in the same layer.
- In Big data the data can first be stored in hadoop File Distribution System and then process data using spark engine.

Designing Data Storage

Considerations

- Type of Database
  - Relational Database
  - Non-relational Database

- Volume Data
  - Data Lakes
  - Big Data repositories

- Intended use of Data
  - Consider transactions, updates, operations, response time and backup/recovery
  - Design for high speed transactions or complex queries for analytical purposes

- Scalability
    - Design considering growth/ Scalability.

- Storage Consideration
    - Throughput, latency
    - Availability, No downtime, continues access to data.
    - Integrity - Data safety from corruption, Recoverability, Ability to recover data in case of failures or disasters.

- Security and Governance
    - Access control ,Encryption, data management, monitoring.
    - Compliance with regulation (GDPR, CCPA, HIPAA).
    - Data privacy, security and governance.
    - Considerations include data type, volume intended use, scalability, storage, performance, availability, integrity,recoverability and adherence to privacy and security regulations.

Common Linux Shell commands

Commands for getting information

- whoami - username
- id - user ID and group ID (-u returns numeric id) (-n returns the name)
- uname -(unix name) operating system name (-s returns the system name) (-r returns the version) (-v for more detailed information)
- ps - running processes (-e shows all the processes)
- top - resource usage (-n 3 shows the top three process)
- df - mounted file systems (-h make the output more readable)
- man - reference manual (-k . to list all the commands in the system)
- date - today's date (to format date "+%j day of the year %Y")

Commands for working with files

- cp - copy file cp source destination (cp test.txt Documents/test) (to copy folders you need to specify the -r)
- mv - change file name or path (mv test.txt. Documents/test)
- rm - remove file
- touch - create empty file, update file timestamp
- chmod - change/modify file permissions . chmod +x test.sh (-x removes the executable permission) (Gives executable permissions) (You can remove permission from others by chmod go-r test.sh) g for group and o for other -r remove read permission
- wc - get count of lines, words, character in file. (-l to view only line characters) (-w to view only word counts) (-c for only character count)
- grep - return lines in file matching pattern.
  grep ch people.txt (returns the line containing 'ch') (-i for case insensitivity)
- touch - to create files touch filename.txt
- echo - to add content to file echo "text" >> filename.txt

Commands for navigating and working with directories

- ls - list files and directories (-l more additional information)
- find - find files in directory tree find . -name "a.txt" (-iname for can insensitive search)
- pwd - get present working directory
- cd - change directory
- mkdir - make directory
- rmdir - remove directory (can only be used to remove empty directory)
- rm - remove file or directory (-r force remove directory with all child)

Commands for printing file contents or strings, common commands

- cat - printing file contents
- more - print file contents page by page
- head - print first N lines of file
- tail - print last N lines of file
- echo - print string or variable value
- sort - sorts the text or number (-r reverse order)
- uniq - gives only consecutive unique lines (sort the file first) (-c count the number of occurrences) (-d display only the repeated) (-u display only the unique values)
- cut - can extract specific character
  $ cut -c 2-9 people.txt
  (-d cut at delimiter -f1 for first half -f2 for second half)
  $ cut -d ' ' -f2 people.txt
- paste - Merge line from different files (-d use this delimiter " , ")
  $ paste file1.txt file2.txt
  $ paste -d ',' file1.txt file2.txt
- export - convert any variable into environment variable
  $ export firstname

Commands related to file compression and archiving

- tar - archives a set of files
- zip - compress a set of files
- unzip - extract files from compressed zip archive

Commands for networking applications

- hostname - print hostname (-i gives ip address)
- ping - send packets to URL and print response (-c 5 pings the target exactly 5 times)
- ifconfig - display or configure system network interfaces
- curl - display contents of file at a URL (-o filename.txt puts output to file)
- wget - download file from URL

Bash (Bourne again shell)

Archive

- Store rarely used information and preserve it
- It is a collection of data files and directives stored as single file
- Make the collection more portable and serve as a backup in case of loss or corruption

  $ tar -cf notes.tar /notes
- Archive with compression

  $ tar -czf notes.tar.gz /notes

  -tf to display the archive structure
  -xf to extract the archive
  -xzf to extract compressed archive

File Compression

- Reduces file size by reducing information redundancy
- Preserves storage space speeds up data transfer and reduces bandwidth load

  $ zip notes.zip /notes
  $ unzip notes.zip

Variable

$ firstname = "jeff"
$ unset firstname

$ read lastname - prompts to input and once the input is given stores in the variable

| pipe

To pass the output of one command as input to another command

Metacharacters

- precedes a comment
; - command separator (multiple commands in single line)
* - filename expansion wildcard

```
ls /bin/ba*
```

\ - escape unique character (metacharacters)
“ ” - interpret literally but evaluate metacharacter
‘ ’ - interpret literally but ignore metacharacter interpretation

I/O redirection

> - Redirect output to a file, create file if it does not exist, overwrite file if it exists.
>> - Append output to the file
2> - Append standard error to file
2>> - Append standard error to file
< - Redirect file contents to standard input

Command substitution

```
$ here=$ ( pwd )
```

Command line argument

- Program arguments specified on the command line
- A way to pass arguments to shell script
- Usage

```
$ ./mybashscript.sh arg1 arg2
```

Batch vs concurrent modes

- Batch mode commands run sequentially

```
$ command1 ; command2
```

- Commands run in parallel

```
$ command1 & command2
```

- Conditional

```
If [ condition ]
Then
        statement-block1
else
        statement-block2
fi
```

- Example

```
If [ [ $# ==2 ] ]
Then
        echo "Number of arguments are equal to two"
else
        echo "Number of arguments are less then one"
fi
```

  - " $# " represents the number of arguments passed to the script or a function.
  - The use of double square brackets, which is the syntax required for making integer comparison in the condition
    [[$# == 2]]
  - Only need single square brackets when making string comparison
    [$string_var == 'yes']

AND , OR

- AND " && "
- OR " || "

Arithmetic calculation

You can perform addition, subtraction, multiplication and division using notation

```
$( ( ) )
$( ( 3 + 2) )
```

Arrays

```
$ My_array = ( 1 2 'yes' 'no' )
```

- Empty array
  ```
  declare -a empty_array
  ```
- Add items to array
  ```
  empty_array += ( 'six' )
  ```
- Print first element
  ```
  echo ${ my_array[0] }
  ```

- Print all elements
  ```
  echo ${my_array[ @ ]}
  ```

For loop

```
for item in ${ my_array [ @ ] }; do
    echo $item
done
```

Ex:

```
N = 6

for ( ( i = 0 ; i <= $N ; i++) ); do
    echo $i
done
```

Cron

- Cron is a service that runs jobs
- Crontab interprets ' crontab files '
- Crontab enables to edit crontab file
- Crontab contains jobs and schedule date
- To access crontab (editor)
  ```
  $ crontab -e
  ```
- Job syntax

M	H	DOM	MON	DOW	Command
30	3	*	*	*	./script.sh (run everyday at 3:30)

M : minute
H : hour
DOM : Day Of Month
MON : Month
DOW : Day Of Week

Data pipeline performance

- latency - the total time taken for a single packet to pass through the pipeline.
- Throughput - How much data can be fed through the pipe per unit of time.

Applications of pipeline

- 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

Stages of data pipeline processes

- Data pipeline processes include
    - scheduling or triggering,
    - monitoring,
    - maintenance,
    - optimization

- Pipeline monitoring
    - tracking latency
    - throughput
    - resource utilization
    - failures

Parallelization and I/o buffers can help mitigate bottlenecks.

Popular ETL Tools

- Talend
    - Supports big data, data warehousing and profiling
    - Includes collaboration, monitoring, and scheduling
    - Drag-and-drop GUI allows you to create ETL pipelines
    - Automatically generatesJava code
    - Integrates with many data warehouse

- AWS Glue

- IBM Infosphere
    - A data integration tool for designing, developing and running ETL and ELT jobs
    - The data integration component of IBM InfoSphere Information Server
    - Drag and drop graphical interface
    - Uses parallel processing and enterprise connectivity in a highly scalable platform

- Alteryx
    - Self-service data analytics platform
    - Drag-and-drop accessibility to ETL tools
    - No SQL or coding required to create pipelines

- Panoply
    - An ELT - specific platform
    - No-code data integration
    - SQL-based view creating
    - Shifts emphasis from data pipeline development to data analytics
    - Integrates with dashboard and Bi tools such as Tableau and PowerBI

- Airflow
    - Apache Airflow and python
    - Versatile "configuration" as code data pipeline platform
    - Open-sourced by Airbnb
    - Programmatically author, schedule and monitor workflows
    - Scales to Big Data
    - Integrates with cloud platforms

- Pandas
  - Does Not readily scale to big data
  - Libraries with similar API : Vaex, Dask and Spark help with scalping up
  - Consider Sql-like alternatives such as postgreSQL for Big Data applications

- IBM Streams
  - Build real-time analytical applications using SPL, plus Java, Python or C++
  - Combine data in motion and at rest to deliver intelligence in real time
  - Achieve ene-to-end processing with sub-millisecond latency
  - Includes IBM Streams Flows, a drag and drop interface for building workflows

- Apache Storm
- sqlstream
- samza
- Apache Spark
- Azure Stream Analytics
- Apache Kafka

Batch data pipelines

- Operate on batches of data
- Usually run periodically - hours, days, weeks apart
- Can be initiated based on data size or other triggers
- when latest data isn't needed
- Typical choice when accuracy is critical

Use cases

- Data backups
- Transaction history loading
- Billing and order processing
- Data modeling
- Forecasting sales or weather
- Retrospective data analysis
- Diagnostic medical image processing

Streaming data pipelines

- Ingest data packets in rapid succession
- for real-time results
- Records/ events processed as they happen
- Event streams can be loaded to storage
- Users publish/subscribe to event streams

Use cases

- Media streaming
- social media feeds, sentiment analysis
- Fraud detection
- User behavior, advertising
- Stock market trading
- Real-time product pricing
- Recommender systems

Micro-batch data pipelines

- Tiny micro-batches and faster processing simulate real-time processing
- Smaller batches improve load balancing, lower latency
- When short windows of data are required

Batch vs Stream

- tradeoff between accuracy (Batch) and latency (Stream)
- Data cleaning improves quality, but increases latency (Batch)
- Lowering latency increases potential for errors (Stream)

Lambda architecture

- Historical data is delivered in batches to batch layer
- Streaming data in delivered via speed layer
- Then the data is integrated in Serving layer

- Here the data streams fills the latency gap while batch loading
- Used when data is needed but speed is critical
- Drawback is logical complexity
- Lambda architecture = accuracy and speed

Apache Airflow

- Apache Airflow is primarily designed as a batch processing system.
- It is a platform for orchestrating complex workflows and data pipelines in a batch-oriented manner.
- In a batch processing model, data is collected, processed, and then the results are produced at scheduled intervals or in response to trigger events.

DAG (Directed Acyclic Graph)

Workflows in Apache Airflow are defined using directed acyclic graphs, where nodes represent tasks and edges represent dependencies between tasks. These workflows are often scheduled to run at specific intervals or triggered by external events.

Schedulers and Executors

Airflow's scheduler determines when to execute tasks based on their defined schedules. Executors are responsible for running those tasks. This is more aligned with the batch processing model, where tasks are executed at predetermined times.

Task Execution

Each task within an Airflow DAG represents a unit of work. Tasks are executed in a specific order based on their dependencies. The results of one task can be used as input for another, allowing for a sequential flow of operations.

Note: Apache Airflow is not designed as a dedicated stream processing system like Apache Kafka or Apache Flink.

The Dag python code (pure python) syntax

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator

                                                    → Define default arguments for
                                                  the DAG
default_args = {
    'owner': 'your_name',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

                                                    → Instantiate the DAG object
dag = DAG(
    'your_dag_id',
    default_args=default_args,
    description='Description of your DAG',
    schedule_interval=timedelta(days=1),  # DAG runs daily
)

# Define tasks
start_task = DummyOperator(
    task_id='start_task',
    dag=dag,
)

def your_python_function():
    # Your Python code here                    → this is a operator definition
    pass

python_task = PythonOperator(
    task_id='python_task',
    python_callable=your_python_function,     → this a a task definition
    dag=dag,
)

end_task = DummyOperator(                     → A dummy operator used to
    task_id='end_task',                         end dag
```

```
    dag=dag,
)

# Set up task dependencies
start_task >> python_task >> end_task
```

1. Import
2. Default args
3. Dag instantiation
4. Python Operator or function definition
5. Task definition (python_callable)
6. Order of task execution

The Dag Python, Bash syntax

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.operators.dummy_operator import DummyOperator

default_args = {
    'owner': 'your_name',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

→ Instantiate the DAG object

```
dag = DAG(
    'bash_example_dag',
    default_args=default_args,
    description='A DAG with a BashOperator example',
    schedule_interval=timedelta(days=1),  # DAG runs daily
)

start_task = DummyOperator(
    task_id='start_task',
    dag=dag,
)
```

→ Bash command to be executed instead of python function and the relative task definition.

```
bash_command = """
echo "Executing Bash Command"
# Your actual bash command here
echo "Bash Command Completed"
"""

bash_task = BashOperator(
    task_id='bash_task',
    bash_command=bash_command,
    dag=dag,
)
```

```
end_task = DummyOperator(
    task_id='end_task',
    dag=dag,
)
```

→ Task Flow

```
start_task >> bash_task >> end_task
```

1. Import
2. Default args
3. Dag instantiation
4. Bash command (bash_command)
5. Task definition
6. Order of task execution