**WEEK-2: ETL & DATA PIPELINES: TOOLS AND TECHNIQUES**

1. **ETL USING SHELL SCRIPTS**

**ETL Techniques:**

ETL stands for Extract, Transform, and Load, and refers to the process of curating data from multiple sources, conforming it to a unified data format or structure, and loading the transformed data into its new environment.

,A diagram of a diagram

Description automatically generated

*ETL is used to describe the main processes behind a data pipeline design methodology that stands for Extract-Transform-Load. Data extracted from disparate sources to an intermediate staging area where it is integrated and prepared for loading into a destination such as a data warehouse.*

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 def 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.

A diagram of a process

Description automatically generated

*Data packets being fed in sequence or piped through the ETL data pipeline. Ideally, by the time the third packet is ingested, all three ETL processes are running simultaneously on different packets.*

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 OLTP system can be moved as a daily processes to one or more OLAP systems where subsequent analysis of large volumes of historical data is carried out.

Batch processing intervals need not be periodic and can be triggered by events such as:

* When the source data reaches a certain size
* When an event of interest occurs and is detected by a system such as intruder alert
* On-demand, with web-apps such as music, video streaming services

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.

A diagram of a stage

Description automatically generated

*An ETL data integration pipeline concept for a Cost Accounting OLAP, fed by disparate OLTP systems within the enterprise. The staging area is used in this example to manage change detection of new or modified data from the source systems, data updates, and any transformations required to conform and integrate the data prior to loading to the OLAP.*

ETL Workflow as DAGs (Directed Acyclic Grpah).

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

*Example of an Airflow DAG:*

A diagram of a running process

Description automatically generated

Green-boxes 🡪 Individual tasks

Arrows 🡪 Dependencies between tasks

3 tasks on left 🡪 runs “runme\_j” are jobs that run simultaneously along with also\_run\_this

Once the runme\_j tasks complete, the ‘run\_after\_loop’ task starts.

Funally ‘run\_this\_last’ engages once all tasks have finished successfully

Popular ETL tools:

* Automation – Fully automated pipelines
* Ease of use: ETL rule recommendations
* Drag and Drop interface – “0 code” rules and data flows
* Transformation support – Assistance with complex calculations
* Security and compliance – Data encryption and HIPAA, GDPR compliance

Commercial and open-source ELT tools:

* Talend
* AWS Glue
* IBM Infosphere DataStage
* Alteryx
* Apache Airflow
* Python Pandas library

**LAB -1**

Consider the following Usecase Scenario: TEMPERATURE REPORTING SCENARIO

Task: *Report temperature statistics for a remote location*

* + *Hourly average, min and max temperature*
  + *Remote temperature sensor*
  + *Update every minute*

Given:

* + ‘get\_temp\_api’ – read temperature from remote sensor
  + ‘load\_stats\_api’ – load stats to dashboard repo

Workflow:

A diagram of a software application

Description automatically generated

|  |  |
| --- | --- |
| Step – 1 | **Create an ETL shell script**  $touch Temperature\_ETL.sh  $Gedit Temperature\_ETL.sh  Use Bash shebang  #! /bin.bash |
| Step – 2 | **Extract reading with get\_temp\_API** |
| Step – 3 | **Append reading to temperature.log** |
| Step – 4 | **Buffer last hour of readings** |
| Step – 5 | **Call get\_stats.py to aggregate the readings** |
| Step – 6 | **Load the stats using load\_stats\_api** |
| Step – 7 | **Schedule the bash script to run every minute** |

**ETL SCRIPT:**

Step-1: Extract and Buffer

$ touch temperature.log

In the text editor:

#! /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

Step-2: Transform temperatures

get\_stats.py

* Reads temperatures from log files
* Calculates temperature stats
* Writes temperature stats to file
* I/O filenames specified as CLI args

*#call get\_stats.py to aggregate the readings – Calls get\_stats python file using temperature.log and writes to temp\_stats.csv*

Python3 get\_stats.py temperature.log temp\_stats.csv

Step-3: Loading

*#load the stats using load\_stats\_api (temp\_stats.csv -> CLI arg)*

load\_stats\_api temp\_stats.csv

Step-4: Set Permissions

$ chmod +x Temperature\_ETL.sh

Step-5: Schedule your ETL job

#*Open cron tab editor and enter schedule*

$ crontab -e

1 \* \* \* \* path/Temperature\_ETL.sh

Close the editor and save your edits.

**NOTE:**

* ETL pipelines are created with BASH scripts too.
* ETL jobs can be run on a schedule using cron.

**LAB Hands-On:**

Start PostgreSQL.

Open new terminal:

* $ echo”database” | cut -c1-4 (first 4 characters)
* $ echo “database” | cut -c5-8 (5th to 8th characters)
* $ echo “database” | cut -c1,5 (1st and 5th characters)

*Extracting fields and columns:*

*-d delimiter -f field number*

* $ cut -d”:” -f1 /etc/passwd (extracts usernames – the first field)
* $ cut -d”:” -f1,3,6 /etc/passwd (extracts multiple fields 1st, 3rd and 6th – username, userid and home dir from /etc/passwd)
* $ cut -d”:” -f3-6 /etc/passwd (extracts a range of fields 3rd to 6th - userid, groupid, user description and home dir from /etc/passwd)

*Transforming data using tr:*

*tr – Filter command used to translate, squeeze and/or delete characters.*

* $ echo “Shell Scripting” | tr “[a-z]” “[A-Z]” (Translates all lowercase to uppercase)
* $ echo “Shell Scripting” | tr “[:lower:]” “[:upper:]”
* $ echo “Shell Scripting” | tr “[A-Z]” “[a-z]” (Translates all uppercase to lowercase)

*Squeezing repeating occurrences of characters:*

*-s : replaces a sequence of repeated characters with a single occurrence of that character*

* $ ps | tr -s “ “ (Replaces repeat occurrences of space in the output of ps command with one space)
* $ echo "My login pin is 5634" | tr -d "[:digit:]" (Delete specific characters using -d: Deletes all digits)

**Staring PostgreSQL DB:**

* Creating table:

**create table users(username varchar(50), userid int, homedirectory varchar(100));**

Open new terminal:

* touch csv2db.sh

Open csv2db.sh and insert the following lines of code:

# This script

# Extracts data from /etc/passwd file into a CSV file.

# The csv data file contains the user name, user id and

# home directory of each user account defined in /etc/passwd

# Transforms the text delimiter from ":" to ",".

# Loads the data from the CSV file into a table in PostgreSQL database.

# Extract phase

echo "Extracting data"

# Extract the columns 1 (user name), 2 (user id) and 6 (home directory path) from /etc/passwd into extracted-data.txt

cut -d":" -f1,3,6 /etc/passwd > extracted-data.txt

#run bash csv2db.sh

# cat extracted-data.txt

# Transform phase

echo "Transforming data"

# read the extracted data and replace the colons with commas.

tr ":" "," < extracted-data.txt > transformed-data.csv

#run bash csv2db.sh

# cat transformed-data.csv

# Load phase

echo "Loading data"

# Send the instructions to connect to 'template1' and

# copy the file to the table 'users' through command pipeline.

echo "\c template1;\**COPY users FROM '/home/project/transformed-data.csv' DELIMITERS ',' CSV;"** | psql --username=postgres --host=localhost

**Note:**

To load data from a shell script, you will use the **psql** client utility in a non-interactive manner. This is done by sending the database commands through a command pipeline to **psql** with the help of **echo** command.

The basic structure of the command which we will use in out script is:

**Copy table\_name from ‘filename’ DELIMITERS ‘delimiter\_character’ format**

# run csv2db.sh

*Go to Postgre SQL CLI that you used in the run command to verify that the table users is populated with the data*

# echo '\c template1; \\SELECT \* from users;' | psql --username=postgres --host=localhost

**PRACTICE EXERCISES:**

Copy the data in the file ‘web-server-access-log.txt.gz’ to the table ‘access\_log’ in the PostgreSQL database ‘template1’.

The file is available at the location : https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Bash%20Scripting/ETL%20using%20shell%20scripting/web-server-access-log.txt.gz

The following are the columns and their data types in the file:

a. timestamp - TIMESTAMP

b. latitude - float

c. longitude - float

d. visitorid - char(37)

e. accessed\_from\_mobile - boolean

f. browser\_code - int

The columns which we need to copy to the table are the first four coumns : timestamp, latitude, longitude and visitorid.

NOTE: The file comes with a header. So use the ‘HEADER’ option in the ‘COPY’ command.

The problem may be solved by completing the following tasks:

1. **Go to the SkillsNetwork Tools menu and start the Postgres SQL server if it is not already running.**
2. **Create a table named access\_log to store the timestamp, latitude, longitude and visitorid.**

* \c template1; (To connect with database template)
* create table access\_log(timestamp TIMESTAMP, latitude float, longitude float, visitorid char(37))

1. **Task 3. Create a shell script named cp-access-log.sh and add commands to complete the remaining tasks to extract and copy the data to the database.**

Create a shell script to add commands to complete the rest of the tasks.

Open new terminal and type:

* $ touch cp-access-log.sh

1. **Task 4. Download the access log file.**

Add the wget command to the script to download the file.

wget “<https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0250EN-SkillsNetwork/labs/Bash%20Scripting/ETL%20using%20shell%20scripting/web-server-access-log.txt.gz>”

run bash cp-access-log.sh

1. **Task 5. Unzip the gzip file.**

Add the code, to run the gunzip command to unzip the .gz file and extract the .txt file, to the script.

# Unzip the file to extract the .txt file.

**gunzip -f web-server-access-log.txt.gz**

The -f option of gunzip is to overwrite the file if it already exists.

1. **Task 6. Extract required fields from the file.**

Extract timestamp, latitude, longitude and visitorid which are the first four fields from the file using the cut command.

The columns in the web-server-access-log.txt file is delimited by ‘#’.

echo "Extracting data"

cut -d"#" -f1-4 web-server-access-log.txt

1. **Task 7. Redirect the extracted output into a file.**

Redirect the extracted data into a file named extracted-data2.txt

cut -d"#" -f1-4 web-server-access-log.txt > extracted-data2.txt

1. **Task 8. Transform the data into CSV format.**

The extracted columns are separated by the original “#” delimiter.

We need to convert this into a “,” delimited file.

echo "Transforming data"

tr "#" "," <extracted-data2.txt > transformed-data2.csv

**check cat transformed-data2.csv**

1. **Task 9. Load the data into the table access\_log in PostgreSQL**

PostgreSQL command to copy data from a CSV file to a table is COPY.

The basic structure of the command is,

**COPY table\_name FROM 'filename' DELIMITERS 'delimiter\_character' FORMAT;**

The file comes with a header. So use the ‘HEADER’ option in the ‘COPY’ command.

Invoke this command from the shellscript, by sending it as input to ‘psql’ filter command.

echo "Loading data"

echo "\c template1;\COPY access\_log  FROM '/home/project/transformed-data2.csv' DELIMITERS ',' CSV HEADER;" | psql --username=postgres --host=localhost

1. **Task 10. Execute the final script.**

Run the final script.

bash cp-access-log.sh

1. **Task 11. Verify by querying the database.**

In PostgreSQL CLI type:

select \* from access\_logs;

1. **AN INTRODUCTION TO DATA PIPELINES**

**Getting to Know a Data Pipeline:**

Series of connected processes.

Output of one process 🡪 Input of the next process.

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. It is any system which extracts, transforms, and loads the data.

This includes “low-level hardware architectures” driven by software-driven-processes like “commands, programs, and threads”. Bash ‘pipe’ command can be used as the glue that can connect such processes together.

Packet flow through a pipeline:

Data flowing through a pipeline can be considered as data packets.

Packets size can range from single record or event to large collections of data.

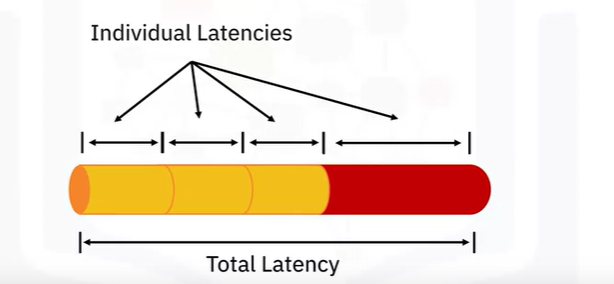
A yellow rectangular object with black text

Description automatically generated

* Queued Data Packets -> Data that is ready for ingestion.
* Pipeline Latency -> Time taken by a data packet to come out of the data pipeline.
* Throughput delay -> The time between successive packet arrivals.

Data pipeline performance key considerations:

* **Pipeline latency**: Time taken by a single packet of data to pass through a pipeline



Latency -> Sym of individual latencies (i.e., sum of individual times spent during each processing stage within the pipeline).

Thus, the overall latency is limited by the slowest process in the pipeline.

* **Throughput:** The amount of data that can be fed through the pipe per unit of time.

Processing larger packets per time unit 🡪 greater throughput.

Applications/ Use Cases of Data Pipelines:

* Backing up files from one location to another.
* 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 from ML development and production.
* Message sending and receiving systems like Email, SMS, video meetings.

**Key Data Pipeline Processes**

Stages of Data pipeline processes:

1. Data Extraction
2. Data Ingestion
3. Data Transformation stages within the pipeline
4. Final loading of data into destination facility
5. Scheduling/triggering mechanisms for jobs to run
6. Monitoring the entire workflow
7. Maintenance and optimization as required to keep the pipeline up running smoothly

Some key monitoring considerations include:

* Latency
* Throughput
* Warnings, errors caused by network overloading, failure of src or dest systems
* Utilization rate: How fully the pipeline resources are made use which effects cost
* Logging and alerting system when failures occur.

Load balanced pipelines:

* Just-in-time data packet relays
* No upstream data flow bottlenecks
* Uniform packet throughput for each stage
* Such a pipeline is called “Load Balanced”

Handling unbalanced loads:

A diagram of a latency

Description automatically generated

Suppose you have a datapipeline and it has a bottleneck in one of its stages, like the red section here, which has more latency than the other stages in the pipeline. If it is possible to parallelize that stage, for example by splitting the data into two concurrent stages, then you can reduce this stage’s latency.

There will be little overhead in managing the parallelization and recombination of the output back into the pipeline, but the overall latency will be reduced.

* Pipeline typically contain bottlenecks. /simple way to handle this is by parallelization. Slower stages may be parallelized to speedup throughput. Processes can be replicated on multiple CPUs/ cores/ threads. Data packets are then distributed across these channels in an alternating fashion amongst the replicated channels. Such pipelines are called dynamic or non-linear pipelines.
* **Stage Synchronization:** I/O buffers can help synchronize stages to smooth out the data flow. Holding area for data between processing stages. Buffers can also regulate the flow of data that may improve processing rate or throughput. Single I/O buffers can also be used to distribute or gather loads on parallelized stages.

**Batch Versus Streaming Data Pipeline – Use Cases:**

**Batch Data pipelines:**

Are used when data needs to be operated as one big batch of data.

They are usually run periodically – hours/ days/ weeks apart on a fixed schedule.

They can be initiated based on data size or other triggers.

Can be used in cases when the latest data is not needed.

They are a typical choice when accuracy is critical.

**Streaming Data Pipelines:**

Ingest data packets in rapid succession or dynamically.

For real-time results

Record/ events processed as they happen.

Event streams can be loaded to storage as history for later use.

Users publish/ subscribe to event streams.

**Micro-batch Data pipelines:**

By decreasing the batch size or increasing the processes you can achieve near real time processing. i.e., tiny micro-batches and faster processing simulate real-time processing.

Smaller batches improve load balancing, lower latency.

Useful only when very short windows of data are required.

**Batch vs Stream requirements:**

* Tradeoff between accuracy and latency requirements.
* Data cleaning improves quality but increases latency.
* Lowering latency increases potential for errors.

**Lambda architecture:**

It is a **hybrid-architecture** designed for handling big data. They combine batch and streaming data pipeline methods.

**Historical data** is delivered to batches through **batch layer**.

**Real time data** is streamed through a **speed layer**.

These two layers are then integrated in the **Serving layer**.

Data streams fill in “latency gap”.

Lambda can be used in cases where earlier data is important and data window is needed but speed is also important.

The drawback is the logical complexity involved in the design.

Mainly we consider lambda architecture when we are looking for **accuracy** and **speed**.

**Use Cases:**

|  |  |
| --- | --- |
| **Batch Data Pipelines** | **Stream Data Pipelines** |
| Data Backups  Transaction history loading  Billing and order processing  Data modelling  Forecasting sales or weather  Retrospective/ historical data analysis  Diagnostic medical image processing | Watching movies, listening to music/ podcasts  Social media feeds, Sentiment analysis  Fraud detection  User behavior, Advertising  Stock market trading  Real time product pricing  Recommender systems |

**Data Pipeline Tools and Technologies**

Features of modern pipeline tools:

* **Automation:** Fully automated pipelines
* **Ease of use:** ETL rule recommendations
* **Drag and drop GUI:** “no-code” rules and data flows
* **Transformation support:** Assistance with complex calculations
* **Security and Compliance:** Data encryption and compliance with HIPAA and GDPR

Open source Data pipeline tools:

1. **Pandas Python library:**

Versatile and popular programming tool

Based on DataFrames – table like structures

Great for ETL, data exploration, and prototyping

Doesn’t readily scale to Big DATA

1. **Libraries with similar APIs:**

Vaex, Dask, and Spark help with scaling up

1. Consider **SQL-**like alternatives such as **PostgreSQL** for Big Data applications
2. **Apache Airflow** (Configuration as a Code Data Pipeline platform) and Python
3. **Talend Open Studio:**

Supports big data, DWH, and profiling

Includes collaboration, monitoring and scheduling

Drag and drop GUI allows you to create ETL pipelines

Automatically generates Java code

Integrates with many DWHs

1. **AWS Glue:**

Fully managed ETL service that simplifies data prep for analytics.

Suggests schemas for storing your data.

Create ETL jobs from AWS console.

1. **Panoply:**

ELT specific platform

No-code data integration

SQL based view creation

Shifts emphasis from data pipeline development to data analytics

Integrates with dashboards and BI tools such as PowerBI and Tableau

1. **Alteryx:**

Self-service data analytics platform

Frag and drop accessibility to ETL tools

No SQL or coding required to create pipelines

1. **IBM InfoSphere DataStage:**

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 GUI

Uses parallel processing and enterprise connectivity in a highly scalable platform.

1. **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 end-to-end processing with sub-millisecond latency.

Includes IBM Streams Flows, a drag and drop interface for building workflows.

Other Stream processing technologies:

Apache storm sqlstream samza

Apache spark Apache kafka Azure Stream analytics

A screenshot of a computer screen

Description automatically generated