ETL Techniques

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 (see Figure 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.

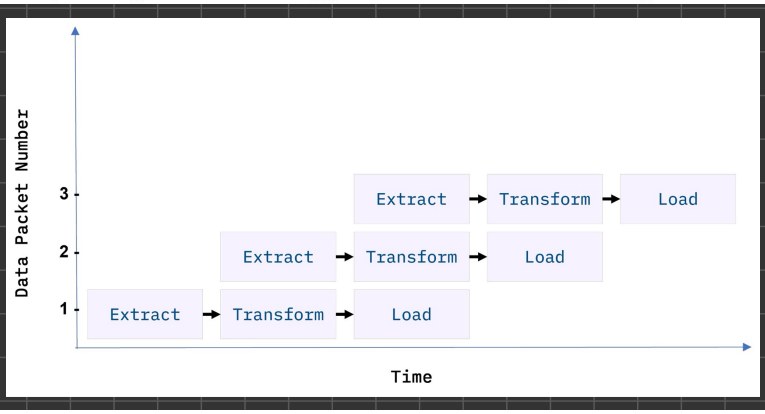


Fig 2. Data packets being fed in sequence, or "piped" through the ETL data pipeline. Ideally, by the time the third packet is ingested, all three ETL processes are running simultaneously on different packets.

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 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.

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

- when the source data reaches a certain size, or
- when an event of interest occurs and is detected by a system, such as an intruder alert, or
- on-demand, with web apps such as music or 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. As an example, (see Figure 3) a cost accounting OLAP system might retrieve data from distinct OLTP systems utilized by the separate payroll, sales, and purchasing departments.

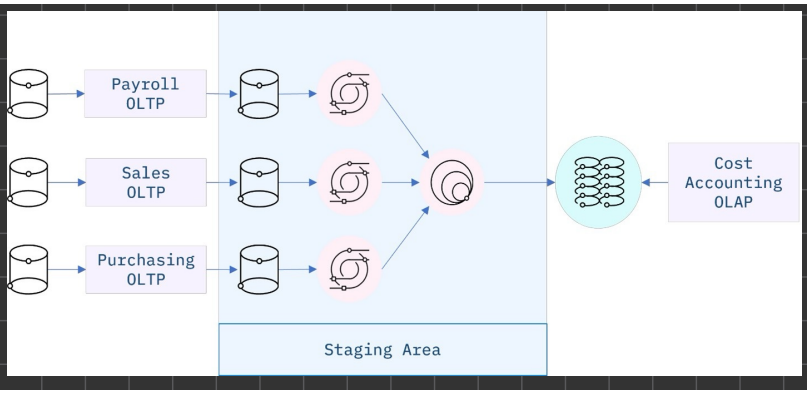


Fig 3. An ETL data integration pipeline concept for a Cost Accounting OLAP, fed by disparate OLTP systems within the enterprise. The staging area is used in this example to manage change detection of new or modified data from the source systems, data updates, and any transformations required to conform and integrate the data prior to loading to the OLAP.

ETL 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 Figure 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.

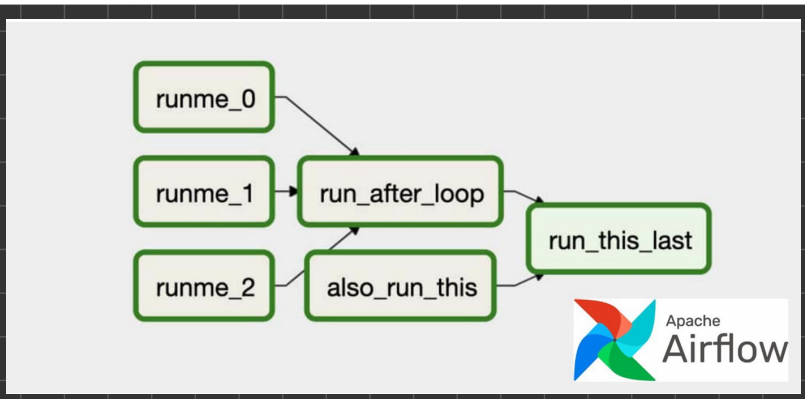


Fig 4. An Apache Airflow DAG representing a workflow. The green boxes represent individual tasks, while the arrows show dependencies between tasks. The three tasks on the left, 'runme_j' are jobs that run simultaneously along with the 'also_run_this' task. Once the 'runme_j' tasks complete, the 'run_after_loop' task starts. Finally, 'run_this_last' engages once all tasks have finished successfully.

Popular ETL tools

There are many ETL tools available today. Modern enterprise grade ETL tools will typically include the following features:

- Automation: Fully automated pipelines
- Ease of use: ETL rule recommendations
- Drag-and-drop interface: "o-code" rules and data flows
- Transformation support: Assistance with complex calculations
- Security and Compliance: Data encryption and HIPAA, GDPR compliance

Some well-known ETL tools are listed below, along with some of their key features. Both commercial and open-source tools are included in the list.

talend

Talend Open Studio

- Supports big data, data warehousing, and profiling
- Includes collaboration, monitoring, and scheduling
- Drag-and-drop GUI for ETL pipeline creation
- Automatically generates Java code
- Integrates with many data warehouses
- Open-source

alteryx

Alteryx

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

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

AWS Glue

AWS Glue

- ETL service that simplifies data prep for analytics
- Suggests schemas for storing your data
- Create ETL jobs from the AWS Console

IBM InfoSphere

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 graphical interface
- Uses parallel processing and enterprise connectivity in a highly scalable platform

pandas

The Pandas Python library

- Versatile and popular open-source programming tool
- Based on data frames – table-like structures
- Great for ETL, data exploration, and prototyping
- Doesn't readily scale to Big Data

ETL using Shell Scripting

Scenario :

This is an outline of how this can be Achieved using bash scripting

Temperature reporting scenario

Task: Report temperature statistics for a remote location

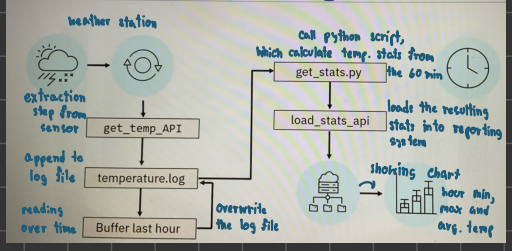
- Hourly average, min. and max. temperature
- Remote temperature sensor
- Update every minute

Given:

- 'get_temp_api' – read temperature from remote sensor
- 'load_stats_api' – load stats to dashboard repo

sketch workflow for ETL pipeline

Temperature reporting workflow



Create an ETL shell script

```
$ touch Temperature_ETL.sh   creating file
$ gedit Temperature_ETL.sh   open the file with text editor
```

```
#! /bin/bash   ← in editor form to Bash Shell script
# Extract reading with get_temp_API
# Append reading to temperature.log
# Buffer last hour of readings   → keep the most recent 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   → fill detail in Task
```

```
#! /bin/bash
# Extract reading with get_temp_API
# Append reading to temperature.log
get_temp_api >> temperature.log
  call api → append to it

# Buffer last hour of readings
  keep the last hour or 60 lines   overwriting
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
                     using the readings in
```

and writes the temp. stats to a .csv file

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

Execising

Exercise 1 - Extracting data using 'cut' command

The filter command `cut` helps us extract selected characters or fields from a line of text.

1. Extracting characters.

The command below shows how to extract the first four characters.

```
echo "database" | cut -c1-4
```

set 1-4 character on database
word

You should get the string `data` as output.

The command below shows how to extract 5th to 8th characters.

```
echo "database" | cut -c5-8
```

You should get the string `base` as output.

Non-contiguous characters can be extracted using the comma.

The command below shows how to extract the 1st and 5th characters.

```
echo "database" | cut -c1,5
```

You get the output : `db`

get index 1 and 5

2. Extracting fields/columns

We can extract a specific column/field from a delimited text file, by mentioning

- the delimiter using the `-d` option, or
- the field number using the `-f` option.

The /etc/passwd is a `":"` delimited file.

The command below extracts usernames (the first field) from /etc/passwd.

```
cut -d":" -f1 /etc/passwd
```

in

The command below extracts multiple fields 1st, 3rd, and 6th (username, userid, and home directory) from /etc/passwd.

```
cut -d":" -f1,3,6 /etc/passwd
```

The command below extracts a range of fields 3rd to 6th (userid, groupid, user description and home directory) from /etc/passwd.

```
cut -d":" -f3-6 /etc/passwd
```

Exercise 2 - Transforming data using 'tr'

`tr` is a filter command used to translate, squeeze, and/or delete characters.

1. Translate from one character set to another

The command below translates all lower case alphabets to upper case.

```
echo "Shell Scripting" | tr "[a-z]" "[A-Z]"
```

lower to upper

You could also use the pre-defined character sets also for this purpose:

```
echo "Shell Scripting" | tr "[:lower:]" "[:upper:]"
```

The command below translates all upper case alphabets to lower case.

```
echo "Shell Scripting" | tr  "[A-Z]" "[a-z]"
```

2. Squeeze repeating occurrences of characters

The `-s` option replaces a sequence of a repeated characters with a single occurrence of that character.

The command below replaces repeat occurrences of `'space'` in the output of `ps` command with one `'space'`.

```
ps | tr -s " "
```

In the above example, the space character within quotes can be replaced with the following : `"[\:space\:]"` .

3. Delete characters

We can delete specified characters using the `-d` option.

The command below deletes all digits.

```
echo "My login pin is 5634" | tr -d "[:digit:]"
```

The output will be : `'My login pin is'`

Exercise 4 - Create a table

In this exercise we will create a table called `users` in the PostgreSQL database using PostgresSQL CLI. This table will hold the user account information.

The table `users` will have the following columns:

1. uname
2. uid
3. home
4. You will connect to `template1` database which is already available by default. To connect to this database, run the following command at the `'postgres=#'` prompt.

```
\c template1
```

You will get the following message.

```
You are now connected to database "template1" as user "postgres".
```

Also, your prompt will change to `'template1=#'` .

2. Run the following statement at the `'template1=#'` prompt to create the table.

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

continue

If the table is created successfully, you will get the message below.

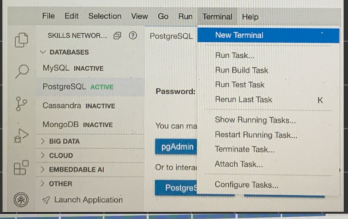
```
CREATE TABLE
```

Exercise 5 - Loading data into a PostgreSQL table.

In this exercise, you will create a shell script which does the following.

- Extract the user name, user id, and home directory path of each user account defined in the /etc/passwd file.
- Save the data into a comma separated (CSV) format.
- Load the data in the csv file into a table in PostgreSQL database.

1. Open a new Terminal.



2. In the terminal, run the following command to create a new shell script named `csv2db.sh`.

```
touch csv2db.sh
```

3. Open the file in the editor. Copy and paste the following lines into the newly created file.

Open **csv2db.sh** in IDE

```
# 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/pass

# Transforms the text delimiter from ":" to ",".
# Loads the data from the CSV file into a table in P re
```

4. Save the file by presseing `Ctrl+s` or by using the **File->Save** menu option.

5. You need to add lines of code to the script that will xtract user name (field 1), user id (field 3), and home directory path (field 6) from /etc/passwd file using the `cut` command.

Copy the following lines and paste them to the end of the script and save the file.

```
# Extract phase

echo "Extracting data"

# Extract the columns 1 (user name), 2 (user id) and
# 6 (home directory path) from /etc/passwd

cut -d":" -f1,3,6 /etc/passwd
```

Extract

6. Run the script.

```
bash csv2db.sh
```

7. Verify that the output contains the three fields, that you extracted.

8. Change the script to redirect the extracted data into a file named `extracted-data.txt`

Replace the cut command at end of the script with the following command.

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

9. Run the script.

```
bash csv2db.sh
```

10. Run the command below to verify that the file `extracted-data.txt` is created, and has the content.

```
cat extracted-data.txt
```

11. The extracted columns are separated by the original `":"` delimiter. You need to convert this into a `","` delimited file. Add the below lines at the end of the script and save the file.

```
Transform phase
cho "Transforming data"
read the extracted data and replace the colons with commas.

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

Transform

12. Run the script.

```
bash csv2db.sh
```

13. Run the command below to verify that the file `transformed-data.csv` is created, and has the content.

```
cat transformed-data.csv
```

14. 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.

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

The basic structure of the command which we will use in our script is,

```
COPY table_name FROM 'filename' DELIMITERS 'delimiter_character' FORMAT;
```

Now, add the lines below to the end of the script 'csv2db.sh' and save the file.

```
# Load phase
echo "Loading data"
# Send the instructions to connect to 'template1' and
# copy the file to the table 'users' through command pipel

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

Load

Exercise 6 - Execute the final script

1. Run the script.

```
bash csv2db.sh
```

2. Go to Postgres SQL CLI that you used in the Run the command below to verify that the table users is populated with the data.

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

Congratulations! You have created an ETL script using shell scripting.

csv2db.sh

```
# Load phase
echo "Loading data"
# Send the instructions to connect to 'template1' and
# copy the file to the table 'users' through command pipeline.

```

theia@theiadocker-tanaphatsa2: /home/project

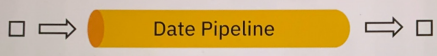
```
You are now connected to database "template1" as user "postgres".
    username      | userid |   homedirectory
------------------+--------+--------------------
root              |      0 | /root
daemon            |      1 | /usr/sbin
bin               |      2 | /bin
sys               |      3 | /dev
sync              |      4 | /bin
games             |      5 | /usr/games
man               |      6 | /var/cache/man
lp                |      7 | /var/spool/lpd
mail              |      8 | /var/mail
news              |      9 | /var/spool/news
uucp              |     10 | /var/spool/uucp
proxy             |     13 | /bin
www-data          |     33 | /var/www
backup            |     34 | /var/backups
list              |     38 | /var/list
irc               |     39 | /run/ircd
gnats             |     41 | /var/lib/gnats
nobody            |  65534 | /nonexistent
_apt              |    100 | /nonexistent
systemd-network   |    101 | /run/systemd
systemd-resolve   |    102 | /run/systemd
messagebus        |    103 | /nonexistent
```

Introduction to Data Pipelines

What is a pipeline?

- Series of connected processes (connected to each other sequentially)
- Output of one process is input of the next
- For example, take a box, pass it to your neighbor, and so on, until the box arrives at the end of the line
- Mass production – parts pass along conveyor belts between manufacturing stages

What is a data pipeline?

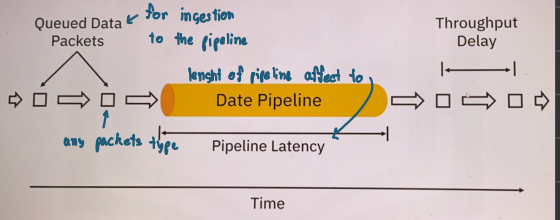


- Purpose: to move data from one place or form to another
- Any system which extracts, transforms, and loads data

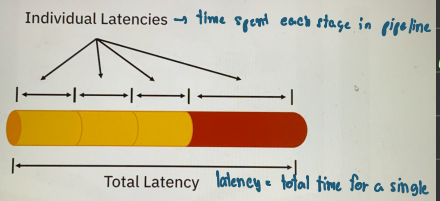
What is a data pipeline?

- Includes low-level hardware architectures
- Software-driven processes – commands, programs and threads
- Bash 'pipe' command can connect such processes together

Packet flow through a pipeline



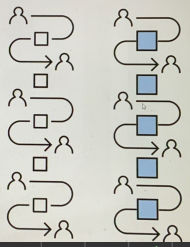
Data pipeline performance: latency



mark:
no matter how fast internet is, to load a web page will be decided by the server's speed

Pipeline performance: throughput

- How much data can be fed through the pipe per unit of time?
- Larger data packets per time unit equals greater throughput



increase productivity

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

Key Data Pipeline Processes

Data pipeline processes

Stages of data pipeline processes:

- Data extraction
- Data ingestion into the pipeline
- Optional: Transformation stages within pipeline

Data pipeline processes

Further stages of data pipeline processes:

- Loading into destination facility
- Scheduling or triggering
- Monitoring
- Maintenance and optimization

Pipeline monitoring considerations

↳ to ensure data integrity

Some key monitoring considerations include:

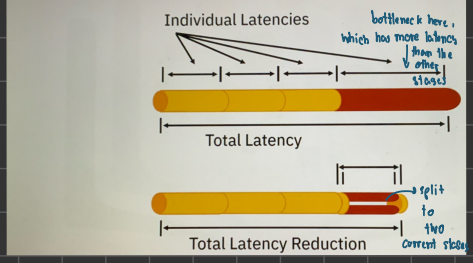
- Latency
- Throughput — volume of data passing through the pipeline overtime
- Warnings, errors, failures — such as network overloading when failure occur
- Utilization rate
- Logging and alerting system

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"

all stages should take the same amount of time to process a packet

Handling unbalanced loads



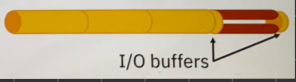
Handling unbalanced loads

pipelines are rarely perfectly load balanced

- Pipelines typically contain bottlenecks
- Slower stages may be parallelized to speed up throughput → A simple way to parallelize a process is to replicate
- Processes can be replicated on multiple CPUs/cores/threads
- Data packets are then distributed across these channels
- Such pipelines are called dynamic or non-linear

Stage synchronization

- I/O buffers can help synchronize stages
- Holding area for data between processing stages
- Buffers regulate the flow of data, may improve throughput
- I/O buffers used to distribute loads on parallelized stages



Bath vs Streaming Data Pipeline Use Cases

Batch data pipelines

- Operate on batches of data → as one big unit
- Usually run periodically – hours, days, weeks apart → fix schedule
- Can be initiated based on data size or other triggers → don't depend on recency of data
- When latest data isn't needed
- Typical choice when accuracy is critical → rapidly maturing

Note: Streaming alternatives are emerging

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

Micro-batch data pipelines

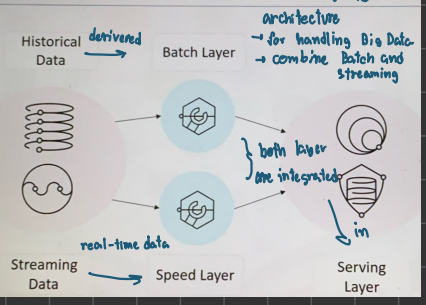
↳ by decreasing batch size and increasing the refresh rate of individual batch process

- 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 requirements

- Tradeoff between accuracy (batch) and latency (stream) requirements
- Data cleaning improves quality, but increases latency → for batch
- Lowering latency increases potential for errors → for stream

Lambda architecture → is a hybrid architecture → for handling Big Data → combine Batch and Streaming



Lambda architecture

- Data stream fills in "latency gap"
- Used when data window is needed but speed is critical
- Drawback is logical complexity → in design
- Lambda architecture = accuracy and speed

Batch data pipeline use cases

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

Streaming data pipeline use cases

- Watching movies, listening to music or podcasts
- Social media feeds, sentiment analysis
- Fraud detection
- User behavior, advertising

Streaming data pipeline use cases

- Stock market trading
- Real-time product pricing
- Recommender systems

Data Pipeline Tools and Technology

Features of modern pipeline tools

Typical enterprise grade technologies:

- Automation: Fully automated pipelines → from Extract to Load
- Ease of use: ETL rule recommendations
- Drag-and-drop interface: "no-code" (also known as) rules and data flows → specifying rules and data pipeline flows
- Transformation support: Assistance with complex calculations, storing, and merging data
- Security and compliance: Data encryption and compliance with HIPAA and GDPR

Open source data pipeline tools

- The Pandas Python library

pandas

  - Versatile and popular programming tool
  - Based on data frames – table-like structures
  - Great for ETL, data exploration, and prototyping
  - Doesn't readily scale to Big Data
- Libraries with similar APIs: Vaex, Dask, and Spark help with scaling up → to big data
- Consider SQL-like alternatives such as PostgreSQL for Big Data applications

Open source data pipeline tools

- Apache Airflow and Python

Airflow

  - 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 AWS, IBM, GCP

Open source data pipeline tools

- Talend Open Studio

talend

  - Supports big data, data warehousing and profiling
  - Includes collaboration, monitoring, and scheduling
  - Drag-and-drop GUI allows you to create ETL pipelines
  - Automatically generates Java code → no need to write code
  - Integrates with many data warehouses

Enterprise data pipeline tools

- AWS Glue

AWS Glue

  - ETL service that simplifies data prep for analytics
  - Suggests schemas for storing your data
  - Create ETL jobs from the AWS Console

Enterprise data pipeline tools

- Panoply

PANOPLY

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

Enterprise data pipeline tools

- Alteryx

alteryx

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

Enterprise data pipeline tools

- IBM InfoSphere DataStage

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

Streaming data pipeline tools

- IBM Streams

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

More streaming data pipeline tools

Stream-processing technologies include:

APACHE STORM · sqlstream · samza · Spark · kafka