Case Study 1 - Case (Part 2)

Summary	
URL	NA
Category	Web
Status	Unpublished
Author	Anshul Chaudhary, Narayani Arun Patil, Agash Uthayasuriyan

Research and Understanding
- Requirements for Pipeline
- Challenges
- Research Summary
- ETL - Extract, Transform and Load
- ELT - Extract, Load, Transform
- EtLT - Extract, transform, Load, Transform
- Some Design Patterns Implemented by Microsoft

Design Development
- Architecture Diagram
- Design process, decisions, and rationale:

Assumptions and Use Cases
- Assumptions
- Use cases

Discussion and Justification
- Why ETL?
- Why not ELT or EtLT?
- Why Directed Acyclic Graph + Foreach pattern?
- Why not other design patterns?
- Why CQRS and not others?

References

Research and Understanding

Requirements for Pipeline

1. Get raw review data from the source (site) and obtain the available meaningful data into individual components.
2. Have a columnar-type database, so that a large amount of data can be read at once.
3. Avoid duplicate data and perform deduplication.
4. Periodically collecting data to accommodate newly added data in the source.
5. Have the final output of the data pipeline suitable for use in BI tools like Tableau and also suitable for analysts to query the database directly.

Challenges

1. Determining which design pattern to select.
2. How do we ensure all the reviews are extracted?
3. Should all the processes run synchronously?
4. How do we avoid duplication of data?

Research Summary

There are three main components in building a data pipeline, i.e. Extraction, Transformation, and Loading.

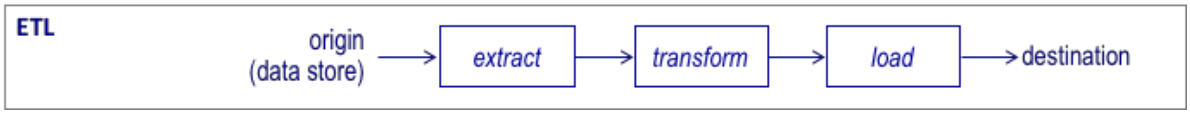
- **Extract**: Extraction refers to the process of retrieving or pulling data from a source system. This source could be a database, a file, an application, or any other data repository. Extraction involves identifying and collecting relevant data for further processing.
- **Transform**: Transformation involves the cleaning, restructuring, and converting of extracted data into a format that is suitable for analysis or storage. This step may include tasks such as filtering out unnecessary information, aggregating data, handling missing values, and applying various transformations to meet the requirements of the target system.
- **Load**: Loading involves the insertion of the transformed data into a target system, which could be a data warehouse, a database, or another storage

system. Loading ensures that the processed and transformed data is available for analysis, reporting, or any other business use.

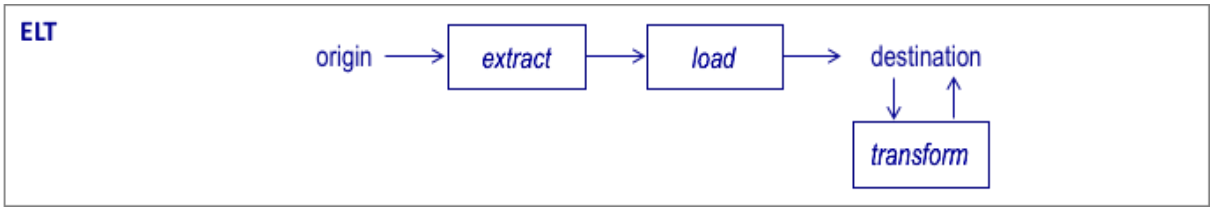
Based on the pipeline design, there could be three patterns possible:

- ETL
- ELT
- EtLT

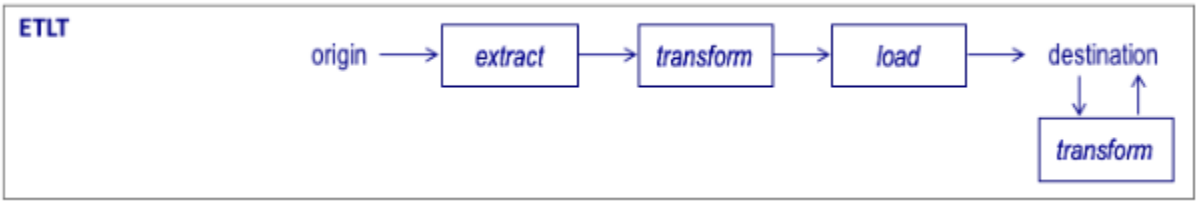
ETL - Extract, Transform and Load



ELT - Extract, Load, Transform

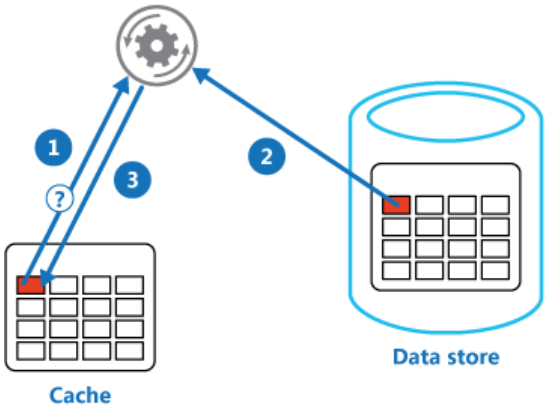


EtLT - Extract, transform, Load, Transform



Some Design Patterns Implemented by Microsoft

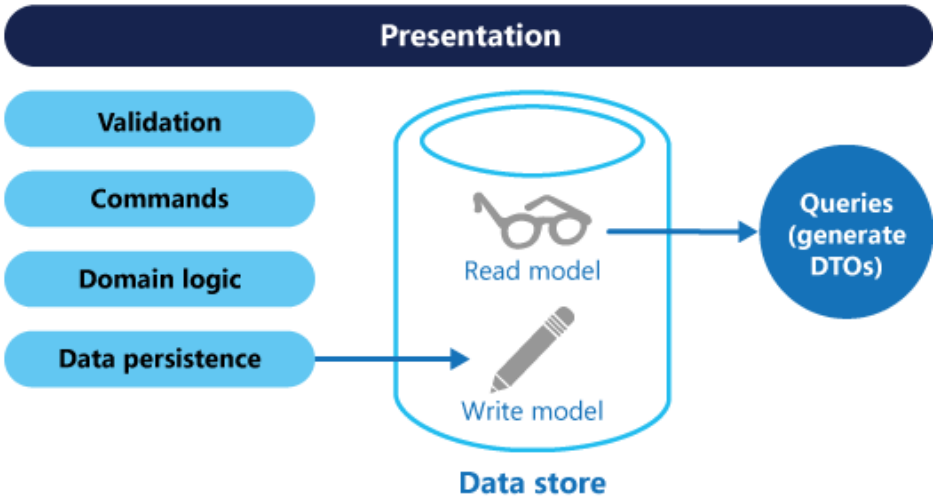
- **Cache-Aside pattern**
  Load data on demand into a cache from a data store. This can improve performance and also help to maintain consistency between data held in the cache and data in the underlying data store.



1: Determine whether the item is currently held in the cache.
2: If the item is not currently in the cache, read the item from the data store.
3: Store a copy of the item in the cache.

- **CQRS pattern**

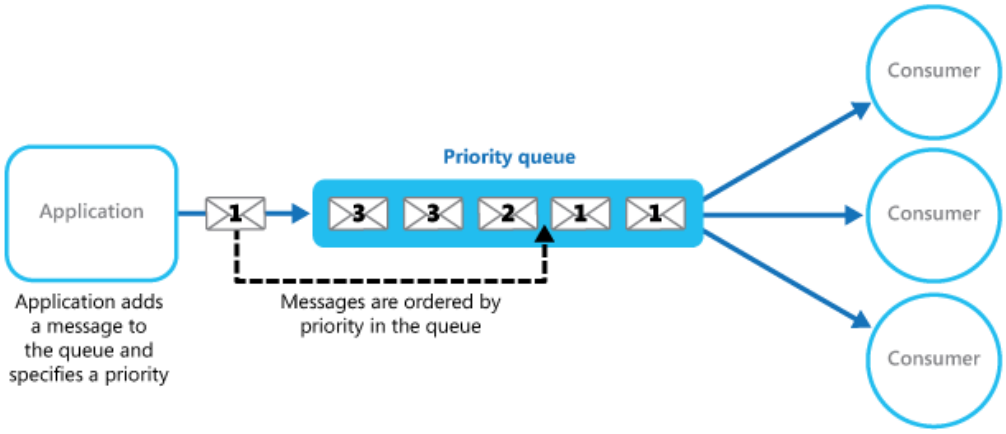
  CQRS stands for Command and Query Responsibility Segregation, a pattern that separates read and update operations for a data store. Implementing CQRS in your application can maximize its performance, scalability, and security. The flexibility created by migrating to CQRS allows a system to better evolve and prevents update commands from causing merge conflicts at the domain level.



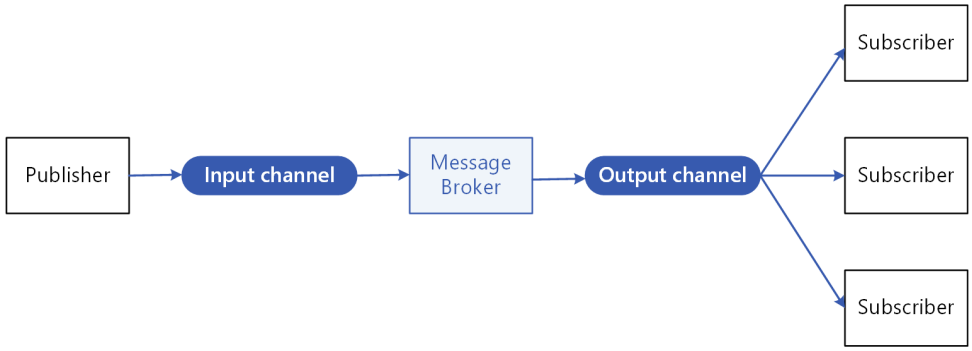
- **Priority Queue pattern**

  Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority. This

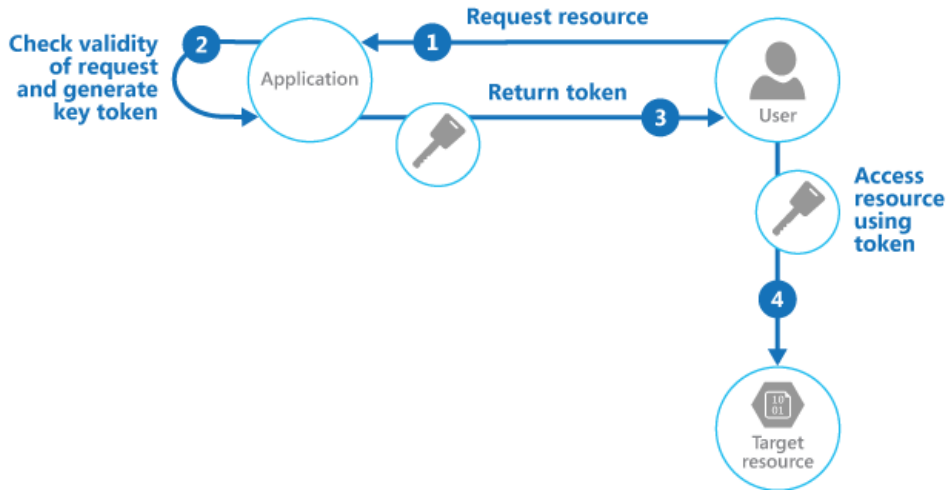
pattern is useful in applications that offer different service-level guarantees to individual clients.



- **Publisher-Subscriber pattern**
  Enable an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers.

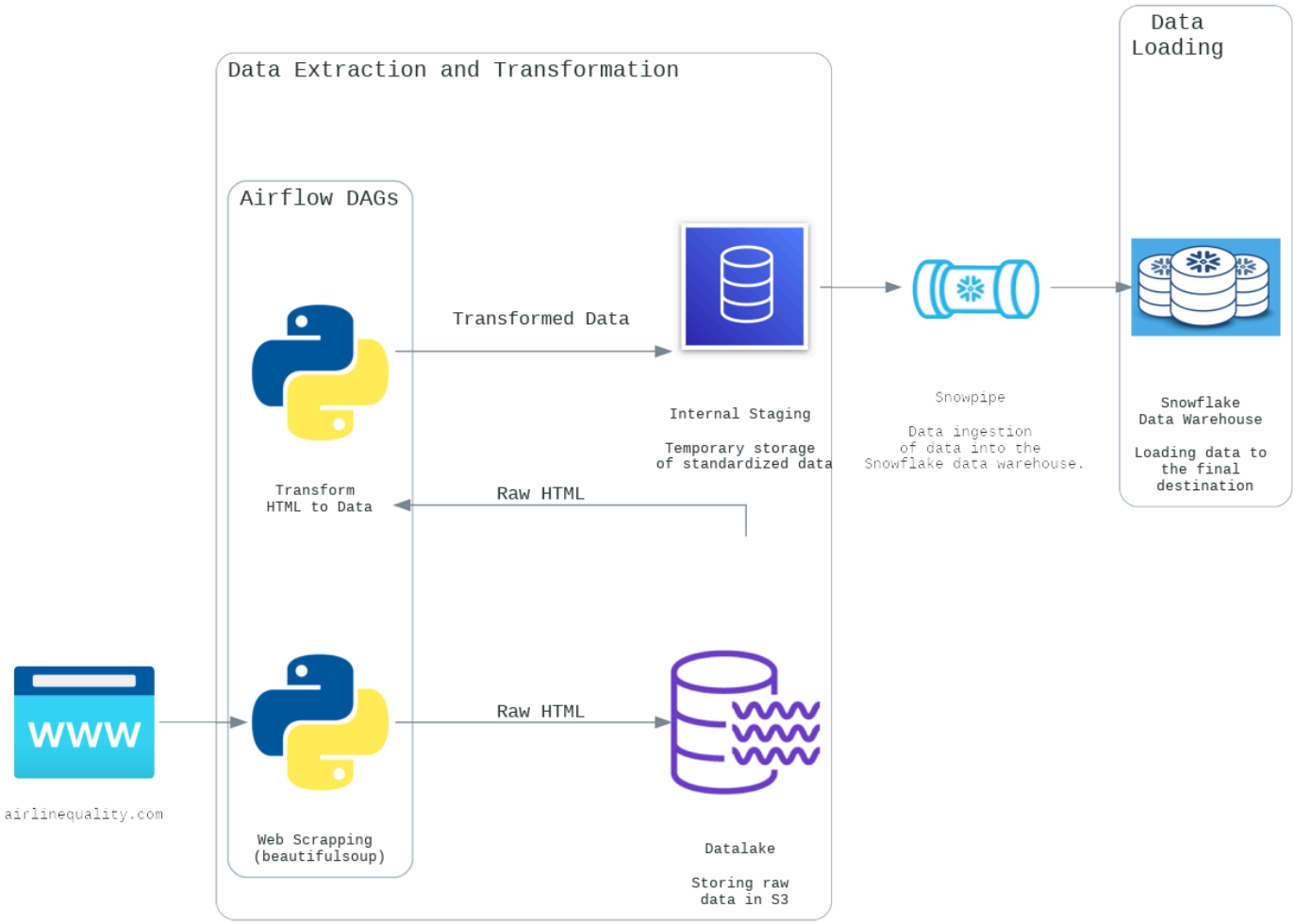


- **Valet Key pattern**
  Use a token that provides clients with restricted direct access to a specific resource, to offload data transfer from the application. This is particularly useful in applications that use cloud-hosted storage systems or queues and can minimize cost and maximize scalability and performance.



Design Development

Architecture Diagram



Data Architecture Diagram
Airlines Service Review System

Design Flow:

Airflow: Airflow is leveraged to schedule and orchestrate Directed Acyclic Graphs (DAGs) for efficient extraction of data. Data extraction occurs every month.

DAG-1: Data Extraction

- Used to execute a python script to scrape HTML data from a review site.
- Beautifulsoup library is used to extract the HTML contents.
- Raw Data is stored in an Amazon S3 bucket.

DAG-2: Data Processing

- Usable data is collected from the raw data that was loaded into the snowflake internal stage.

Snowflake Internal Stage: The internal stage is used to store all of the processed data. When the stage is updated the snowpipe is triggered.

Snowpipe: Snowpipe is used to load data into a snowflake warehouse. Snowpipe also ensures that no duplicate values are loaded.

Snowflake Warehouse: This is the final destination of data.

Design process, decisions, and rationale:

1. Data Extraction from "airlinequality.com": This involves extracting airline service review data from the "airlinequality.com" website. The pipeline accommodates this by using a Python script, assisted by BeautifulSoup, to perform web scraping.

Rationale: For the data collection from the website we opted for web scraping with Beautiful Soup library in python script because it gives more control over the underlying infrastructure. BeautifulSoup's robustness allows it to handle both well-structured and imperfect HTML, enabling the extraction of data straightforwardly.

2. Storing Raw Data in Data Lake (S3): After extraction, the raw data is stored in a Data Lake in Amazon S3 (DataLakeResource). The pipeline accommodates this by utilizing S3 as a scalable and durable storage solution, ensuring the raw data is securely stored and ready for further processing.

Rationale: We have proposed to store raw data in a Data Lake because it ensures preservation and flexibility for future analyses, allowing us to maintain unaltered records and adapt to evolving data needs efficiently. Additionally, centralized storage promotes data governance, reusability, and cost efficiency in managing large datasets.

3. Data Transformation and Standardization: The pipeline includes a transformation cluster to standardize and scale the extracted data. The Python script will handle the standardization and scaling processes.

Rationale: The reason behind this step is to handle diverse review formats, ranging from text to numeric, with varying scales. This process ensures uniformity, enabling effective comparison and analysis across different review sections.

4. Internal staging: It provides temporary storage for the standardized data, facilitating efficient data processing.

Rationale: This step serves as temporary storage for standardized data, optimizing cost. It allows for streamlined operations, reducing the need for extensive computational resources and enabling more cost-effective data management.

5. Snowpipe (Data Ingestion into Snowflake Data Warehouse): Snowpipe is employed for data ingestion into the Snowflake data warehouse from the staging area. The pipeline accommodates this by leveraging Snowpipe's capabilities for near-real-time ingestion from internal/external stage into Snowflake, automating the loading process.

Rationale: We opted snowpipe for data ingestion because snowpipe allows real-time data ingestion by loading data into Snowflake as soon as it becomes available, thereby minimizing data latency and ensuring timely availability for analysis. And the snowpipe also maintains a metadata cache of data loaded into the table which will help with deduplication.

6. Loading Data to the Final Destination (Snowflake Data Warehouse): The final step involves loading the standardized data into the Snowflake Data Warehouse.

Rationale: Snowflake is preferred for storing final tabular data due to its optimized data warehousing. It provides a purpose-built environment for structured data analytics, ensuring faster and more streamlined analytical workflows.

7. Airflow DAGs: Airflow DAGs in the proposed architecture will automate the execution of tasks such as extracting and transformation of data as per predefined schedules.
Rationale: In the architecture, we have scheduled tasks i.e. data extraction every month from the website (airlinequality.com). These airflow DAGs will ensure timely task execution which will lead to minimal errors.

Assumptions and Use Cases

Assumptions:

1. **Funding Availability:**
   Assumption: There is enough funding to support the project's operational costs.
   This implies that the project will not face financial constraints in terms of maintaining and operating the data pipeline.

2. **Active Source Website:**
   Assumption: The source website (site) is active, and users keep adding new reviews of customers.
   This assumes a continuous stream of raw data, allowing the data pipeline to be regularly updated with new customer reviews.

3. **Stable HTML Configuration:**
   Assumption: The HTML configuration of the website does not change over time.
   This implies that the web scraping or data extraction process will remain consistent, reducing the need for frequent adjustments to the data pipeline.

4. **Scaling Option:**
   Assumption: There is an option for scaling the computing and storage resources if required.
   This suggests that the data pipeline is designed to handle increased loads and data volumes by scaling resources as needed.

Use cases

Building a Data Pipeline for extracting Airline Customer Reviews and storing in tabular format which will be used for BI reports:

1. **Data Extraction from Source Website:**
   Extract the latest customer reviews from the source website (site).

Temporarily store the raw HTML data in a scalable storage solution, such as a cloud-based object storage service.

2. **Preprocessing and Data Cleaning:**
   Process the raw HTML data to extract meaningful information, such as review text, rating, date, and other details.
   Standardize values (e.g., convert ratings to a common scale, handle missing data) to ensure consistency and ease of analysis.

3. **Deduplication Process:**
   Implement a deduplication mechanism to avoid redundancy in the final dataset.
   Ensure that only unique reviews are considered, preventing the storage of duplicate information in the final destination.

4. **Storage in a Tabular Database:**
   Store the processed and standardized data in a tabular database for efficient querying and reporting.
   Choose a database that supports the requirements of the BI reporting process, such as relational databases or data warehouses.

Discussion and Justification

Suitable design: An ETL pipeline that follows a Directed Acyclic Graph + Foreach pattern in a Command and Query Responsibility Segregation setup.

Why ETL?

The ETL architecture is selected due to the sequential flow, where the extraction of raw data precedes transformation and loading. This ensures a logical and structured approach to processing data. It allows data transformation before loading, ensuring that only relevant and structured information is stored for analysis.

By performing transformation after extraction, the ETL design minimizes redundant processing. This enhances efficiency by streamlining the data modification process, avoiding repeated transformations post-loading.

Why not ELT or EtLT?

It is a mandatory step to separate the meaningful data from the raw data immediately after extracting it. Therefore, “Loading” into storage after extraction would not be an ideal option, ruling out the ELT design.
Also, the necessary transformation of data is required only once in the pipeline. The aim is to have the raw data modified into a structured version that is suitable for analysts to interpret. Therefore, a transformation step after the loading phase is not required in the data pipeline architecture, ruling out the EtLT design.

Why Directed Acyclic Graph + Foreach pattern?

Directed Acyclic Graphs (DAG) were chosen since our desired pipeline does not have any circular loops between any two nodes and there is a sequence of specific activities to achieve the system’s objective. In our pipeline, there is a linear arrangement of nodes that only moves forward and the workflow is static with predefined dependencies & objectives of each node.

The retrieval of raw data from the source can be done by parsing through each of the subpages – that had reviews. Until all the subpages of the source are visited, the process of retrieving raw data from web pages has to be repeated before proceeding to the next node. Therefore, the Foreach pattern is followed in the extraction part of our pipeline.

DAG + Foreach pattern is the most suitable design pattern that our data pipeline could follow to suit the pipeline requirements. Additionally, following this pattern makes the pipeline robust, scalable, and manageable.

Why not other design patterns?

· Single leader: There is no requirement for master/slave nodes in the pipeline and also that there is no node that is aware of the system’s overall state.

· Embeddings: The data does not involve high dimensions
· Data Parallelism: The computation involved is manageable and simple and multiple compute resources are not necessary.
· Model Parallelism: The computing is not intensive to have multiple CPUs or GPUs.
· Federated Learning: No transactional or sensitive information is handled, since there is no such kind of data present in the source to work with.
· Parameter server and Ring all reduce pattern is unnecessary to handle the requirements of the data pipeline

Why CQRS and not others?

CQRS – Command and Query Responsibility Segregation is the most suitable architecture pattern for our pipeline, as the reading and updating of data are handled separately. Since our requirement is the have the final output of the data to be suitable for BI tools that generate reports, and also that the data underlying in the database could get increased/modified over time – based on source, it is crucial to have a data retrieval at user side and data updating at the backend to be isolated. Our pipeline would also face more data reading than data writes. The other architectures' use cases are different and not solely relevant to our data pipeline requirements.

References

- https://coda.io/@colin-bryar/working-backwards-how-write-an-amazon-pr-faq
- https://diagrams.mingrammer.com/docs/getting-started/examples
- https://github.com/googlecodelabs/tools
- https://docs.google.com/document/d/1E6XMcdTexh5O8JwGy42SY3Ehzi8gOfUGiqTiUX6N04o/edit#heading=h.odt7tpbn267k
- https://aws.amazon.com/solutions/implementations/data-lake-solution/
- https://www.airlinequality.com/airline-reviews/british-airways#google_vignette
- https://neptune.ai/blog/ml-pipeline-architecture-design-patterns
- https://www.eckerson.com/articles/data-pipeline-design-patterns
- https://learn.microsoft.com/en-us/azure/architecture/patterns/
- https://docs.snowflake.com/en/user-guide/databases