## **DATA Acquisition Design for Big Data Pipelines**

### **Data Acquisition Patterns**

1. **For a 5-year-old**:

* **File Transfer**: It's like your friend's mom putting all your Legos in a bag and giving them to your mom. Your mom checks if all Legos are there and then brings them to your house.
* **Extraction Client**: It's like you going to your friend's house, picking up a few Legos each time and bringing them to your house.
* **Ingestion API**: It's like your friend throwing Legos one-by-one and you catching them and putting them in your box.
* **Publish-Subscribe**: It's like your friend telling you whenever they have new Legos and you go get them.

1. **For a 15-year-old**:

* **File Transfer**: It's like downloading a whole album from a website, checking if the download was successful, and then adding it to your playlist.
* **Extraction Client**: This is like you going to different websites, searching for your favorite songs, and adding each one to your playlist.
* **Ingestion API**: It's like having a music app where artists can upload their songs, and those songs automatically appear in your playlist.
* **Publish-Subscribe**: It's like subscribing to an artist's channel and every time they release a new song, it automatically appears in your playlist.

1. **For a University student**:

* **File Transfer**: It's like a batch exchange of files between two systems, where the source system sends files to the destination system, then the destination system verifies and acknowledges the receipt.
* **Extraction Client**: This pattern involves a client system periodically querying a source system for data. The client then stores the extracted data in its own system.
* **Ingestion API**: In this pattern, the data pipeline exposes an API for data producers to send their data. The API then stores the received data in the destination system.
* **Publish-Subscribe**: This pattern involves a source system running a publisher service that pushes new data to its subscribers. The subscribers then save the data in their own systems.

#### **1. File Transfer Pattern**

This pattern involves the batch exchange of files, where the file is pushed from the source to the destination.

* Key Features:
  + Acknowledgments and verification
  + Archive/Delete after sending files
  + Receiving server may trigger post processing
* Advantages:
  + Simplicity and ease
  + Retry on failures
  + Encryption
* Disadvantages:
  + Latency
  + Scaling issues
  + Difficulty with incremental transfer
* Use Case:  
  This pattern is best for when the client and server have different software, batch processing, when the volume of data is small, or for remote clients.
* Technologies:
  + FTP/FTPS, SCP, HTTP PUT, SFTP, Apache Flume

#### **2. Extraction Client Pattern**

In this pattern, data resides in a database at the source and an API/query interface provides access to the source data.

* Key Features:
  + An extraction client runs periodically to extract incremental data
  + Extracted data is persisted in a destination database or message queue
* Advantages:
  + Controlled by pipeline
  + Retry on failure
  + Easy verification
  + Encryption
* Disadvantages:
  + Latency
  + Difficulty with parallelism
  + Adaptation to source protocol required
  + Each source needs a separate client
* Use Case:  
  This pattern is best for batch processing, internal databases, and external sources with APIs.
* Technologies: Sqoop, Data Replication, Service SDKs

#### **3. Ingestion API Pattern**

Here, the data pipeline exposes an ingestion API to receive data.

* Key Features:
  + Data producers build clients to push data to the Ingestion APIs from their Source Databases
  + Ingestion API then stores the received data to destination database or message queue
* Advantages:
  + Handles a large number of clients
  + Parallelism
  + Client conformity to API
  + Low latency
  + Enforced security
* Disadvantages:
  + Payload sizes
  + Significant effort required on the client side
  + Fluctuating workloads
  + Interface change management
* Use Case:  
  This pattern is best for batch/real-time processing, small payloads, a large number of clients, and public APIs.
* Technologies:
  + REST APIs, RPC/gRPC, WebSocket, GraphQL

#### **4. Publish-Subscribe Pattern**

In this pattern, the source runs a publisher service and subscribers subscribe to the publisher for data consumption.

* Key Features:
  + When a new message arrives, the publisher pushes it to its subscribers
  + Subscriber saves it to the destination database or message queue
* Advantages:
  + Real-time streaming
  + Low latency
  + Asynchronous
  + Multiple publishers and subscribers
  + Scalability
* Disadvantages:
  + Offset management
  + Exactly-once processing
  + Large payload sizes
  + Wide area networks
  + Public APIs
* Use Case:  
  This pattern is best for real-time processing, streaming, known publishers, multiple consumers, and small payloads.
* Technologies:
  + Kafka, Pulsar, RabbitMQ, ActiveMQ

## **Best Practices in Data Acquisition**

1. **Understand the use case:** Determine whether the processing is batch or real-time, acceptable latencies, incremental data requirements, and data accuracy expectations.
2. **Understand the data source:** Understand the format and protocol of the data, data frequency, delivery guarantees, and support for fault tolerance.
3. **Be open to multiple patterns based on source:** Unique source types might require different patterns. There is no one-size-fits-all approach. Make independent choices and design in a decoupled manner.
4. **Design for scalability and performance:** Consider the expected volume and velocity of data, and design the system accordingly.
5. **Use appropriate technologies:** Choose the right technologies for the job, and use them correctly.
6. **Test and monitor the system:** Test the system thoroughly before deploying it, and monitor it in production to ensure that it is meeting expectations.