Problem:

Imagine that for providing data to fuel this service, you need to receive and insert big batches of new prices, ranging within tens of thousands of items, conforming to a similar format. Each batch of items needs to be processed together, either all items go in, or none of them do.

Both the incoming data updates and requests for data can be highly sporadic - there might be large periods without much activity, followed by periods of heavy activity.

High availability is a strict requirement from the customers.

•    How would you design the system?

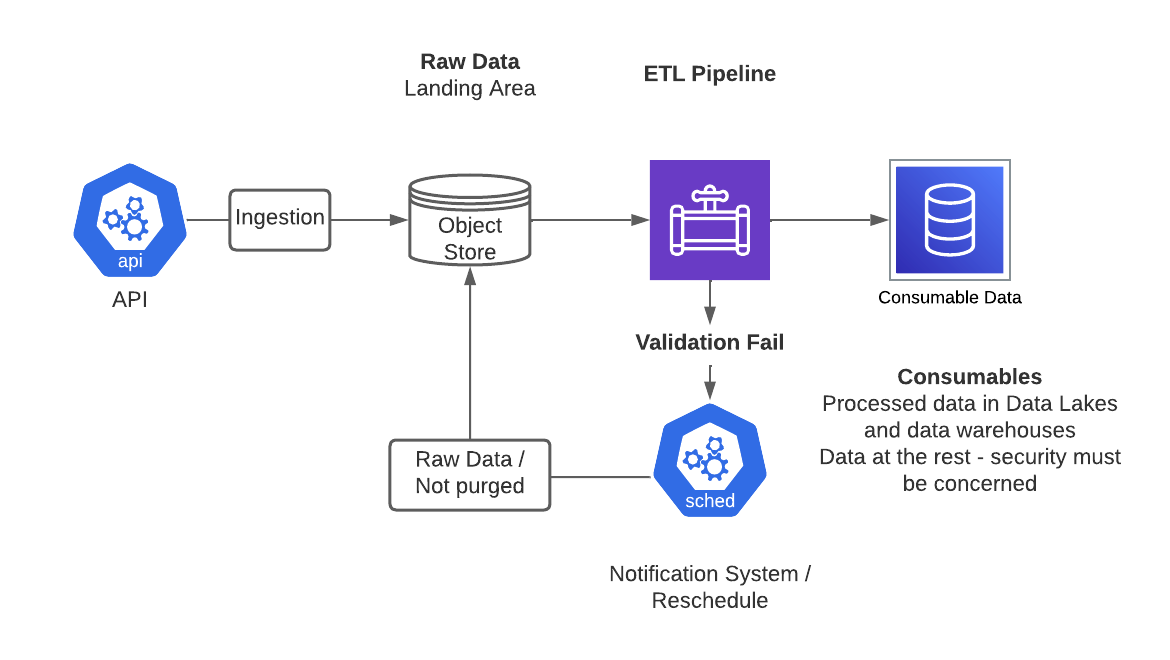
•    How would you set up monitoring to identify bottlenecks as the load grows?

•    How can those bottlenecks be addressed in the future?

Provide a high-level diagram, along with a few paragraphs describing the choices you've made and what factors you need to take into consideration.

Requirement Analysis:

1. Occasional or random incoming data - No guarantee for the peak/off peak timings.
   1. High availability and scalability must be in place.
2. Occasional or random requests for data - No guarantee for the peak/off peak timings.
   1. High availability and scalability must be in place.
3. Batch processing required.
   1. Batch item processed together.
   2. If data failed, whole batch will be failed
      1. Need fall back mechanism or retry for the incoming data.
4. Incoming data must be accepted for the processing.
5. Data ingression should be happened without any issue for the external party.
6. Batch validation must be in place before processing and persisting.
7. Monitoring must be in place to alert high cpu, high memory, restart of service.
8. Notification alerts for the failed batches and **re-trigger them manually**.



Diagram

Description automatically generated

Pipeline:

Input Data:

Assuming that, **no** **real time data processing** is required (because we don’t get stream data) in this data ingestion pipeline. Hence Input raw data must be stored to the object store or data store for processing. Those are uncleaned, non-validated data as it is coming from the up streams.

ETL pipeline:

Let’s assume that batch data process is running on the **spark cluster** and this job starts with the **Apache airflow schedule**.

Apache airflow triggering the scheduler to process the batch (Apache spark job) of all the items and prices.

1. Data validation flow must be considered here.
   1. Data format validation
   2. Information validation against required, items vs prices.
2. If any validation errors or data issues from the data received from upstreams.
   1. Whole batch will be failed and notified to the relevant teams.
   2. Or **trigger data cleaning pipeline and filter** the data.

Since the input data structure is **similar flow all data**, **hive meta store** uses to keep the **catalog information.**

Enabling autoscaling of sparkjob can be manage though the Kubernetes to support the load. Auto scaling will help with it in a cost-effective way.

We can use Airflow **to monitor and well defined DAG (Directed Acyclic Graph**) can be used to ensure the **atomicity** of the batch process (All Success mechanism)

**Monitoring**:

Enabling **sparkmeasure** is supporting to understanding the bottlenecks in each tasks, each stages for particular jobs by measuring processing time, required resources etc..

Publishing **sparkmeasure metrics to Prometheus** is an easy way to monitor the bottlenecks and understand the load.

**Mechanisms to Optimize**

Most optimum way is handling **ingestion skewness**. That can we identify by looking at **RDD partition size in the spark l**ogs. That can be addressed **by re-partitioning in the future**.

When processing the data store more **optimized formats like parquet.**

Use predicate mechanisms like **predicate pushdown** to optimize the data processing.