ETL Solution for Vestiaire collective

Since Vestiaire Collective's mission is that of transforming the fashion industry towards a more sustainable future, my End-to-End Data Pipeline approach will also focus on extracting valuable insights from raw data to empower data-driven decision making sustainably.

This pipeline will transform diverse data sources into actionable business intelligence, eg: including product listings, user behavior, and sales trends.

The Basics

Data Sources:
For me, understanding the data format (CSV, SQL database) and volume from each source is crucial. This is because, different tools handle formats and have data volume limitations. So I am identifying data coming from main platform, CRM, marketing automation, and financial spreadsheets.

Initially, the data from different sources is captured via **AWS DMS**. It is secure and has very less downtime or data loss. It captures small changes through its logs and updates it automatically. Then the data is fetched into Amazon relational databases (RDS) or just S3.

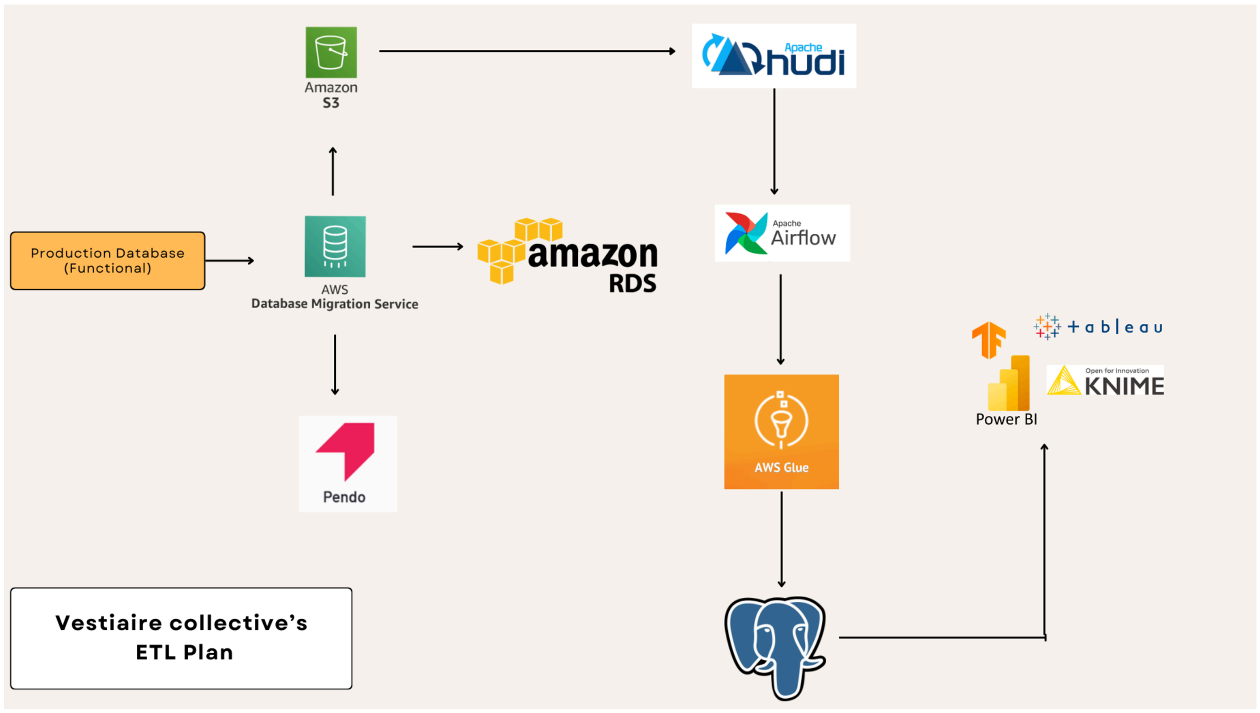
The data is then moved to **Apache hudi** where its delta streamer can build a streaming **data lake** with *incremental* pipelines working as an independent and self-managing database layer.

This means that it can keep an account of the **number of files loaded** and can detect which are not loaded. It keeps track of recent files using a **hoodie table**, and we can refer to this maintained table without any manual work.

From **glue catalog,** a glue job can fetch the data. Since it is serverless, we don't need to bother about other related resources. For custom transformations or requirements, a function can be written inside the logic containers. Eg: Check the validity of data types, and make specific changes to phone number changes according to the country at which the client is (add '+00' against those phone numbers which are not starting with '0')

Inbuilt triggers can be employed to run the glue job however, airflows offers a winning position as it has greater transparency eg: rolls-out emails, and tracks how many jobs have been executed or not.
Eg: In situation when the dag fails it automatically retries after a specific period. The loaded data can be then pushed into postgres as it is a quick and cheap scalable option.

The data is then put into postgres and used for further data applications using ML or visualization tools to gain insights for business.


Amazon S3
Apache hudi
Production Database (Functional)
AWS Database Migration Service
amazon RDS
Apache Airflow
tableau
KNIME
Power BI
Pendo
AWS Glue
Vestiaire collective's ETL Plan

Explaining Main Choices

Why chose Glue?

- I opted for Glue because it is **serverless** and **fully managed server by AWS.** This implies scalability won't be much of an issue. It has a data catalog which allows for efficient data transformations and query processing.
For instance, if the database schema changes from any of the Vestiaire Collective's client then it can keep a history of data schema changes, and saves other control information about the company's ETL environment.

- It has a **simple user interface** for data ingestion, and allows for Automatic ETL Code Generation. Glue has connector for **Apache Hudi and spark.** We can just specify source and destination and it will generate code in Python/Scala. Also, if Vestiaire Collective's side of schema is same for all tenants. Glue catalog can make the data and **metadata** available. Furthermore, it is compatible with Apache Spark thereby allowing for **parallelize heavy workloads.**

- **Quality assurance techniques** of data is incorporated in it. Eg: data validation and sanity checks would be done as glue uses Machine Learning algos for data cleaning and deduplication.It is simpler than **EMR where cluster management** becomes a separate task and has issues with memory issues which causes a job to fail.

Why chose Pendo?

- I found it to be a viable option. It's easy to adopt new features in it.
- It is more user centric than **Google Analytics** with options of **in-app surveys** and **polls**, user activity tracking by account type etc. This can help engineers at Vestiaire Collective to improve their product incrementally. Pendo's segment **user-activity** should help to analyze the retentions and usage of clients.

Why using Hudi?

- Hudi effortlessly brings together database functionaliuties and warehouses into a datalake. To make Vestiaire Collective's analytical workloads faster it should be included for performance optimisations. It works well with variety of query engines like Hive, Spark, Flink etc.

  For instance, hudi resolves small files problem in data lake. During building of data lakefrequent updates/inserts result in small file problem which further causes issue while querying.

- If we run a query to extract or transform Vestiaire Collective's clients, driver node will collect metadata of each file causing the performance overhead during the transformation process.Hudi supports synchronous and asynchronous compaction for running regular compaction of small files.
- It serves well for **large** workloads and has **record** level change streams. Moreover, it has transactions, rollbacks and concurrency control options as well.
- In another scenario, the time taken by **File listing** can be saved. It is a **heavy process**, and generally updates on distributed file system is expensive. Primarily because of their immutable nature. **Hudi** should be used to identify **sub-set** of files that needs to be updated. It then overwrites the files with new version which contains the latest record.It works by storing the **metadata** of each slick of file and track records of its operation.

Why using Postgres?

- It is a free, **open-source database** and ideal for ideal for processing transactional data. Postgres uses **a row-ordered approach** to building tables. It is easy to query many rows just what Reewa wants to do - *extract a few million rows in a da*y and that too in **lowest** cost.

- Imagine we want to perform a quick **EDA(Exploratory Data Analysis)** and want a bird's eye view, of all the data we have. In that case postgres will allow to see unique data patterns. Furthermore, Postgres is resilient **99.9%** of the time in the face of **infrastructure failure**. Its clusters ensures that a **secondary server** will take over if the primary server crashes.

Load balancing and System Design Integration

- Taking into account the **processing** and scalability is extremely critical. We as data engineers must know the constraints such as *data read per seconds*, *requests types*, *data written per second* and *requests per second*. If Vestiaire Collective's system has microservice architecture, thus it must be containerized and employs kubernettes.

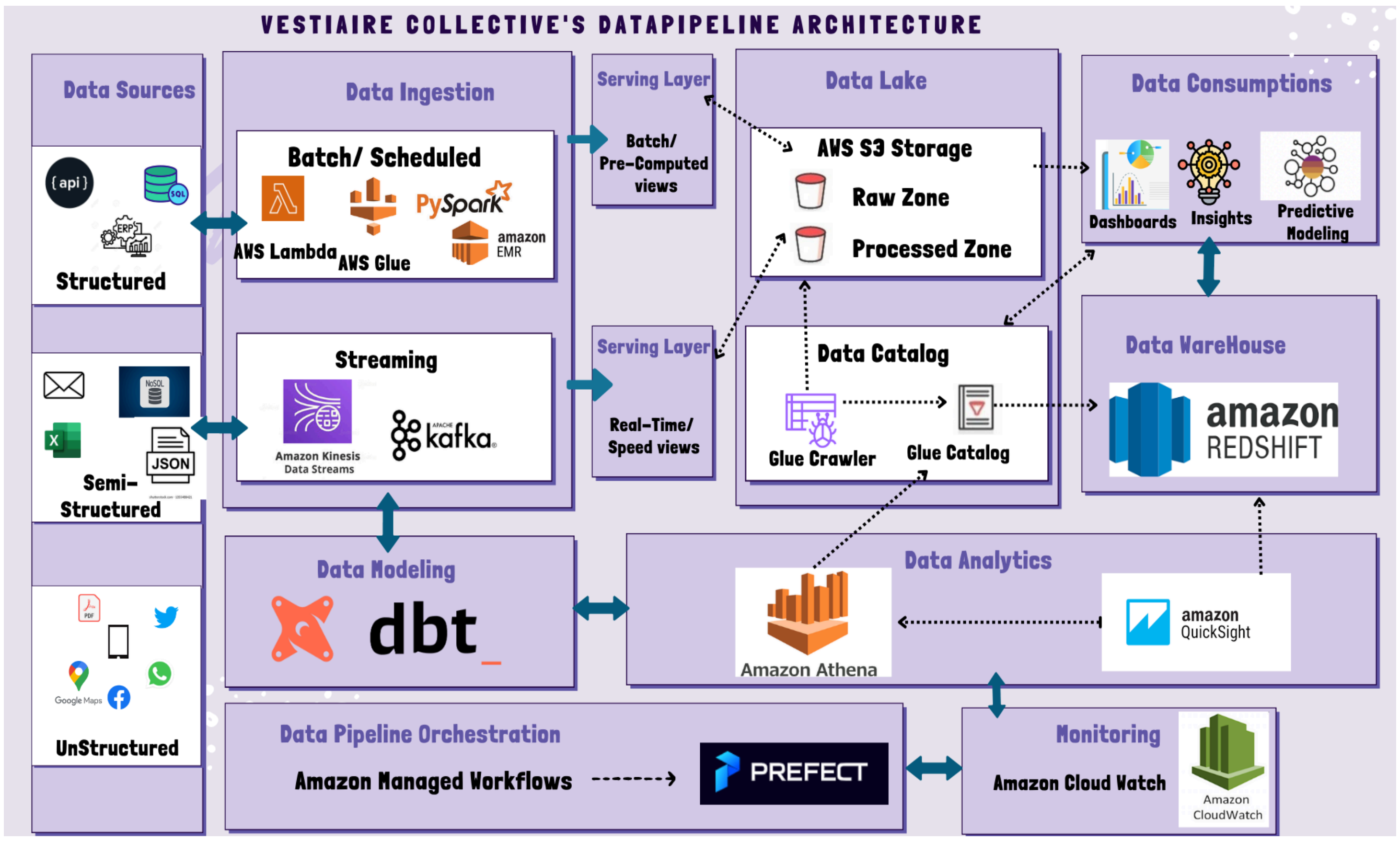
- For this reason I think using **Karpenter** will be an ideal autoscaler in this scenario. It is opensource kubernettes built with AWS. Whenever the application load will change it can provide right-sized compute resources.

- The existing applications and services of Vestiaire Collective like **Datadog** and **Pendo** should be linked with Karpenter.In situation when Karpenter is idle, it mentions it in a log of helm chart. It works on the concept of nodes. It launches new ones when the load is high and terminates them when not needed to reduce infrastructure cost and latencies.

  Basically we are left with less to worry about the CAP- **consistency**, **availability** and **partition** tolerance in our system. This covers **CAP theorem** and takes care of it

Alternate Proposal

Ps: I have mentioned 2-3 options for each of the step in the diagram, please refer to my explanation to see the reasoning behind my specific choice.
In this alternate solution I decided to break different functionalities into more granular states. I have followed through Lambada architecture which includes both Real-time and Batch Processing. Here is the high-level design:



Vestiaire Collective can collect source data from its client by variety of origins. Mainly structured (relational databases, APIs, CRM, ERPs), semi-structured (NoSQL, emails, JSON,Flat files) and lastly unstructured(from mainly social media and maps- as it was mentioned it has **physical stores** as well.

One immediate drawback with my above proposal is that of finances. On a single day it bills out to be around or more than **$1500.** If cost is of little importance then EMR can be used instead of Glue.

EMR is also ideal when for example: You want to run hadoop on the cloud .
So instead of running the server downloading hadoop in that and confiurgiring every single step.
We can go ahead with Amazon EMR.
Just type in the name of your **cluster** , choose the software and big data tools which you need.
Once you are done with that , we can mention how many nodes we want eg: 1 Master and 2 slaves.

Then provide a role and thats it -> click on " *create cluster button*" and we are good to go!

We can have EMR on EKS for fast processing and can be easily deployed over kubernetes. Again, auto-scalars must be connected with it as per the need eg: AWS **Karpenter**

Batch -> is scheduled hence periodic. It should write to a dedicated serving layer before any analytical activity can occur. Used when historical data is of a lot of significance. It is processed in serving layer with precomputed views and stored in datalake(s3) and finally sent to downstream for consumption.

Real-time Streaming -> First flow is when the request is catered in real time by **in-memory** processing with the help of computing a **real-time view**.
Typically this makes use of capabilities hosted at analytical layer but can function without using those capabilities for a limited no of use cases where speed is everything. Kafka is ideal for this.

Chosen **Amazon Redshift** for datawarehouse as it is known for "massive parallel processing" having capacity to store many **petabytes** of data.
AWS Redshift is a fully managed AWS service that is **scalable** as it uses clusters. It gives the option to join **multiple databases** and give full access control to the owners. This means engineers at Vestiaire Collective can offer **grant** and **revoke** rights with end to end permissions.

I have chosen **DBT** for modeling because it works in SQL. No need to learn a new language for data abstraction. Easy apply data quality checks like- *accepted value range,null,foreign keys, uniqueness, etc.*
It aligns all data transformations into separate **models**. Each model is a stand-alone select statement, making collaboration and version control more efficient as it is integratable with **git**.

For analyzing the data **AWS Athena** would be suitable as it is a **serverless interactive analytics** tool and has **decoupled architecture.**Unlike redshift, it needs to infrastructure setup.This means we can leverage inexpensive s3 storage as Athena runs directly on top of it.

Airflow is the most popular orchestration tool but **prefect** is the key competitor to it. It offers cloud with **hybrid** architecture. It has an intuitive **real-time UI** and at the same time is secure. Prefect solves many of the reported issues against airflow which includes **DAGS** delaying the exchange of data and having **complex** branching. Moreover **Prefect's task** can be authenticated with other AWS services.
Cloudwatch can be used instantaneously and easy to use with majority of the AWS services for the collection of **metrics + logs.** It also offers **alerting** via notification if the job fails, and **state handler API** is best suited to inform us about the status of the job.

At the end **data consumption** is all about taking out data driven insights which can help

Vestiaire Collective to **optimize product curation and pricing strategies** or **identify high-value customer segments** or **predict fashion trends** or just **gain insights into user behavior**.

Closing Note

There are many plenty of solutions that are coming to my mind. I have just justified the 2 possible ways in which Vestiaire Collective can use them. **Data Governance** can later on be imposed after this ETL process. It is ideal to have developed this solution made as **Infrastructure as Code**. **Terraform** is one great option for it as it eases the creation of *account*, *buckets*, *machines* and *roles*.The proposal is scalable, highly available, and informs us about the status of overrun pipelines to check for failed pipelines.

Whenever choosing the solution, it is important that we keep Vestiaire Collective's business use-case, budget and performance requirements as the priority.

Thank You!