Pipeline Components

There are two core components for a data ingestion pipeline.

- Ingestion and Curation
- Loading to RDS

Assumptions:

There are the following assumptions that I have made while designing this architecture.

1. The solution is designed using AWS services
2. There will be a remote location from where we will be getting our data. It could be any remote/ on-premise database or an endpoint.
3. There will be a job written in any programming language that has all the logic to pull the data.
4. Data is not loaded into target database in real time

Ingestion and Curation

As per architecture, this stage has the following components responsible for ingestion of data, cleaning and applying transformation (if any).

1. Step Functions
2. Fargat Job
3. S3 Raw
4. Glue ETL job
5. S3 Curated
6. Glue Catalog and Athena

Explanation of Architecture:

We are using step functions to orchestrate the flow of our ingestion pipeline. For every state failure we are using **SNS**.
The first state is **igestion state**. In ingestion state, a job will be triggered hosted in **AWS ECS Fargat** that will have all the logic to pull the data from a remote host and dump it as it is, without modifying, in **S3 Raw** bucket. The data could be in any format, xml, json, csv etc. I used ECS Fargat because the amount of data could vary and we might not need to run the job all the time. Fargat is serverless so we are not worried about managing the infrastructure instead we can focus on the logic. Also the autoscaling could help us in case there is a spike in data retrieval. We can set alarms based on memory consumption to scale in and out.

The next state is the **curation state**. During this stage, we will be using our Glue ETL job to process, clean and transform (if needed) the data to required structure. I used Glue ETL because it's serverless and we can process huge amounts of data without worrying about the

underlying resources (though some configurations are still needed like DPUs). Finally the processed/ curated data will be stored in a different **S3 Curated** bucket in Parquet format (column based file storage).

Optionally I have used Glue Catalog and Athena. I believe this will greatly help us understand and audit the data. We can query anytime to the curated data using Athena.

Loading to RDS

This is the second part of our pipeline. I, on purpose, keep this component independent from the ingestion. Because as per our assumption, the data will be received in the form of batches. And as per instructions, we do not want to disturb the actual underlying data.
This stage will be scheduled and triggered during off-peak hours. We, after looking at our load stats, can schedule this job.
It has following components

1. Lambda Function
2. Database Migration Service (DMS)
3. Relational Database Service (RDS)

A lambda function will be used as a scheduler to trigger the DMS task.
DMS service will be used to do the migration of data from **S3 Curated** bucket to our target database RDS.

Questions

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

DMS uses the compute power but it is not fully managed, So we have to manually set up the compute requirements according to the need and load. So if the load fluctuates we cannot scale up or down resources so it can bottleneck our current data pipeline.

How can those bottlenecks be addressed in the future?

There are multiple options.

1. We can monitor the workload on the DMS replication instance metrics and based on that we can set the maximum instance type.
2. We can completely replace DMS. We can write our own application to dump the curated data to RDS and run in fargate serverless that will scale out/ in based on the need.

The batch updates have started to become very large, but the requirements for their processing time are strict.

In the solution we are using aws fargate which is a fully managed service. So if the load increases, It will scale up the compute resources, so it can handle load. And processing keeps on going.

Code updates need to be pushed out frequently. This needs to be done without the risk of stopping a data update already being processed, nor a data response being lost.

For that we have a lambda scheduler, which will be triggering the DMS jobs in the off development hours. So no databases will be updated in the development hours.

For development and staging purposes, you need to start up a number of scaled-down versions of the system

Fargate will be taking care of itself, as its fully managed service so it will be using low compute resources according to the load and requirements. That's how this mechanism is very scalable and cost effective.

Bottlenecks or problems that might make it incompatible with the new requirements?

I have designed the architecture keeping an eye on the future requirements. But incorporation of new data sources and business logic could affect the existing architecture. But we might not need to update too many components. I believe we might need to update the existing source code but the rest of the architecture, more or less, would remain the same.

How would you restructure and scale the system to address those?

It totally depends upon the use-case.