Q2. Case: Data ingestion pipeline

Assumptions made when answering the questions,

1. Cloud based solution + AWS is the vendor

2. Question is focused on how to ingest data

Data ingestion can mainly categories in to two parts

1. Batch processing
2. Stream processing

About the nature of the use case, this is obviously batch processing.

By looking at the whole scenario and the behavior of the data, the obvious solution is to go with serverless.

Why?

Mostly if we want to build a solution from a scratch with the server setup for this type of data behavior. This solution will become more costly and same time it will be a huge operation overhead at the end of the day.

Issues you may face,

- Due to unpredictable updates and data requests scaling will be a crucial

- I/O handling

- Cost + operation overhead (license costs, Maintenance, Operational time, security, resources cost (compute + human))

- Knowledge about specific technologies (eg, kafka, pub/sub,rabit MQ, Hive)

- High Availability

When it comes to the serverless-based solution, The following are the options available but not all of them are the best,

Some solutions pipeline with technologies -: AWS batch, AWS Glue, Kinesis, EMR, EMR serverless

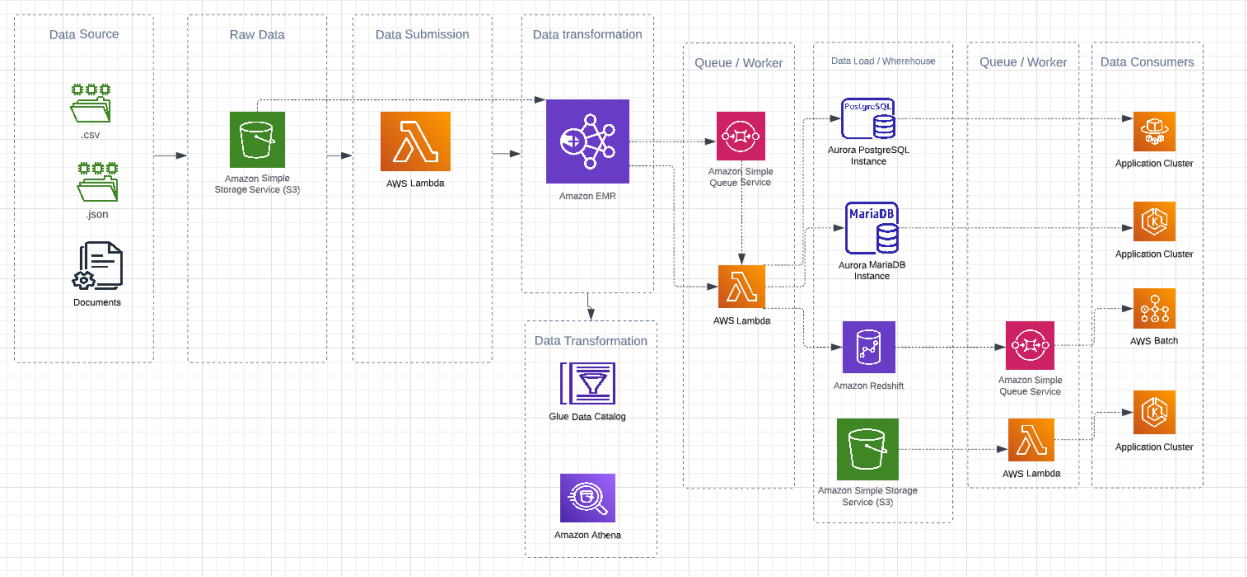
Kenisis more of a streaming process

AWS Glue is slow, costly but flexible and handy when it comes to the ETL

But when it comes to following factors, I'll go with EMR with managed scaling-based solution,

1. Can handle larger number of batch processing
2. Not ideal ETL but still performing fast (this can optimize)
3. Automatically identify the resources required for application + able scale up and down based the volumes of requests.
4. Support for opensource solutions
5. Less operational overhead and straight forward cost control

Hence here I would like to suggest a solution for the given scenario and the same time please note that this will address all the additional questions as well



Walk through of the architecture,

Here usually the data batches we received will first store on the S3 bucket and then it will go through the lambda layer to do some formatting stuff or which job files will take place. But of course, EMR can directly work with S3 as well. So once EMR receives it will do the necessary data processing and transforming based the logics define inside of it. Then once data is ready it can provide processed data to endpoints such as AWS glue catalog (if further filtering is required), directly written to RDS or lambda or queue services.

Bottlenecks and Monitoring,

1. Mainly bottlenecks come around when EMR writing data to databases.
   - But yet again in this architecture, those were addressed. That's why there is a queue service introduced. So, if there is anything writing a high volume of data directly to the DB we can write it through SQS, and then the lambda function and lambda function able to write data into RDS
2. EMR write into lambda function and lambda functions time limit
   - Here also queue provides the backup but still we can use lambda invocations and run lambda on parallel. Of course, here we need more caution since the pricing factor is largely impacted if we do not properly optimize the lambda functions (e.g. -: memory allocation + time).
3. Also, there are some rare scenarios when EMR writes to s3 you will come across some errors. Here most of the time you have to limit the rates.
4. EMR slowness when writing process data to S3 – largely due to logical issue define on the pyspark scripts
5. Scaling
   - EMR (No issue at all based on the input it scales up and scales down. There is no issue of over-provisioning or under-provisioning here),
   - Lambda (the tricky point is you have to carefully identify when to use Asynchronous invocation and synchronous invocation)
   - RDS and ECS, EKS (Obliviously they have enough time and they can scale accordingly)
6. EMR is not supported for interactive workloads but in that case, it's more like streaming so better to move on a kinesis-based solution.

Monitoring

EMR comes with comprehensive monitoring you can monitor it at job level and both glue and EMR provide fully log output with EMR studio + dashboard capability

Lambda functions also having its own logging consoles and monitoring dashboards like x-ray

But Personally, I would like to go with Datadog monitoring. So, it can provide whole view of entire pipeline performance and issues.

But still you can use AWS cloudwatch dashboard for a less costly solution.

Additional questions

Here are a few possible scenarios where the system requirements change or the new functionality is required:

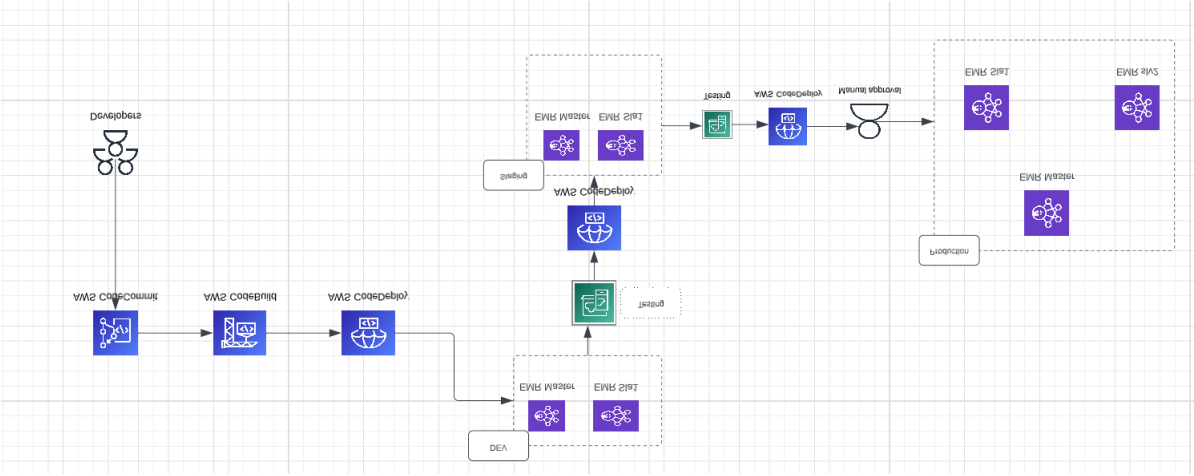
1.**The batch updates have started to become very large, but the requirements for their processing time are strict.**

This is will be a not problem, when it comes to the AWS EMR. EMR has proven stats for 250 Billion batches per day.

When it comes to the EMR serverless this is more enhanced.

2.**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.**

AWS codepipeline could be a good solution,



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

EMR serverless can address this in EMR + terraform (any IAC tool). Most of this solution designed based on serverless technologies and same templates you can use for larger or scale down version.

Another good solution for staging and development environments is to use EMR serverless.

Similar to bellow setup, same with EMR serverless