**Problem statement**

1. **For a given set of data , we would need to add a new field called "Nutritional Content" and set it to (for now, assume a model returns this information for you)**
   1. High Protein
   2. Low Carb
   3. High Fiber
   4. Balanced

Given a model which classifies the Nutritional content of a given food item based on its ingredients, a typical choice of processing and compute for this type of a requirement would be Spark.

1. **We would start with 1 million data and move toward 100 million , how do you go about doing the same transformation in an optimized manner ?**

Given that we are getting a CSV data file as source data which can range from 1 million rows to greater than 100 million.

Assumption: Source file lands on AWS S3 bucket called **Pre-landing** bucket

**Step: 1** Create an AWS Glue job **IngestSourceData** with trigger type set as **EventBridge** event. The job is started upon the occurrence of a single Amazon EventBridge event, that is the arrival of a new object in an Amazon S3 bucket (the S3 PutObject operation).

**Step: 2** Glue job executes a Pyspark code which essentially :

* Extracts the s3 file URI on which object created event was triggered
* Executes a Pyspark code to read CSV data from source S3 location
* Upon reading source file, it executes Data Quality checks to ensure file sanity such as :
  + File schema checks - To detect deviation in schema of file
  + File Delimiter check - To detect misses in attributes leading to data issues

**Step: 3** Post successful file sanity checks, using Pyspark code the source file is written to a **landing** S3 location having Parquet as file format.

**Step: 4** Landing Hive table is created on top of a landing S3 location as External Hive table for easy querying on incoming source data. This table also adds audit columns such as create\_date, update\_date to the data

Also, a target Hive external table with below schema is created post processing entire data

1. UUID

2. Name

3. Ingredients

4. Diet

5. Prep\_time

6. Cook\_time

7. Flavor\_profile

8. Course

9. State

10. Region

11. Nutritional Content

**Step: 5** Next step is to detect, Adds and Updates and perform UpSert logic. This is being done to reduce the volume of data on which we invoke model to return the Nutritional Content

**Step: 6**

* **Insert** - If a given food item exists in the Target table, it is checked for updates in Ingredients , otherwise considered as a New record. All such **Add** records are prepared as a add\_data\_dataframe
* **Update** - For an existing food item, if the ingredients is changed we consider it as a Update and prepare a update\_dataframe with all such records
* Insert and Update records prepared in respective dataframes is then combined to prepare a common data (**Change\_dataframe**) against which we would call model to return “Nutritional Content” which is used to populate this required column

**Step: 7** Anti-join is performed on Target table with this UpSert dataframe to pick all records not part of Adds or Updates i.e. No change records and combined with **Change\_dataframe** to write as parquet on an S3 location

**Step: 8** Target external hive table is now altered to this new S3 location

1. **DataSource where the above csv file can be postgres db, elastic search or AWS s3 or GCP's GCS**

**a. Which data source would you prefer and why ?**

* 1. The data should be stored in the AWS s3 in parquet format.
     1. **Advantages**
        1. It is a columnar storage.It will be easy to query data and
        2. get the aggregated results with less cost.
        3. It accesses less memory to output data. Similar types of
        4. data stored together in the same memory/disk.
        5. **Data compression** -Snappy used as a default
        6. compression algorithm. Compression of similar data types
        7. happens together.
        8. **Data security** - It uses AES 256 data security to store the
        9. data in the s3 location which can be set using the bucket
        10. policy.
        11. **Data Retention** - we can set the rules in AWS s3 for
        12. archiving the data which contributed toward cost saving.
  2. **How would design to ensure data source is not a bottleneck ?** 
     1. In order to ensure, the data source is not a bottleneck, we could have a process to read data in batches based on total rows. Say, if total rows is 1M and we choose to have a batch of 20K records each then total batches shall be 50. This would allow for multiple processing and concurrently 50 batches can be executed to perform the same task.

1. **Please have a design document with various components**

