**Design Document for Review Ingestion System**

This document covers the design of Restaurants Review Ingestion System, its implementation in Google Cloud Platform .

System Design has been implemented on below Scenario and assumptions   
  
### Scenario

You are working for a company that specializes in restaurant reviews and are tasked with designing the new Review Ingestion System. Provide a design document proposing a solution that fulfills the following requirements.

### Requirements

The reviews we receive might contain inappropriate words, so we need to make sure that we filter these words out and replace them with asterisks (`\*\*\*\*`). If a review contains too many inappropriate words in proportion to the overall text, we consider the entire review as inappropriate and need to filter it out. Furthermore, we are not interested in outdated reviews, as they might no longer reflect the reality of the restaurant today.

The Product department requests that we deliver the following aggregated metrics per restaurant ID:

\* Number of reviews

\* Average score

\* Average length of review

\* Review age (in days)

\* Oldest

\* Newest

\* Average

### Constraints

Design your solution under the following assumptions and constraints

\* The reviews are provided as JSONL (new-line delimited JSON) files uploaded into a BLOB storage system (S3, GCS, Azure Blob Storage, HDFS, CephFS, etc.)

\* You are free to choose which BLOB storage you want to use in your design.

\* Assume the reviews land in a dedicated bucket/namespace in the BLOB storage system and are prefixed by date.

\* All reviews are in English

\* A review is considered outdated if it is more than 3 years old from the date of processing.

\* The threshold for "inappropriate words per review" until it is considered inappropriate as a whole is 20% of words of the entire text.

\* The reviews are supposed to adhere to the [included Review JSON schema](schemas/review.json).

\* You are not in control of the upstream producer of these reviews, so consider the following:

\* Frequency of arrivals of new files in the storage.

\* Maximum file size or number of lines in the files (there is at least one line per file).

\* You may receive duplicates.

\* Completeness. The incoming file can't be considered a full import.

\* The incoming reviews might not adhere to the specified JSON schema.

\* Reviews may be updated over time.

\* The list of inappropriate words is provided in the same BLOB storage system in a dedicated bucket/namespace as a one-word-per-line UTF-8 encoded txt file.

\* The file is updated independently of your solution's deployment.

\* The file is always guaranteed to be complete and valid.

\* You are free to choose your deployment target

\* Public cloud environment like (AWS, Azure or GCP)

\* Private cloud/servers

\* Assuming necessary services are available, please focus on the Data Engineering aspects of your solution rather than the operational aspects.

\* Your solution needs to be scalable.

\* The aggregations should be written to a Data Warehouse / Data Lakehouse of your choice.

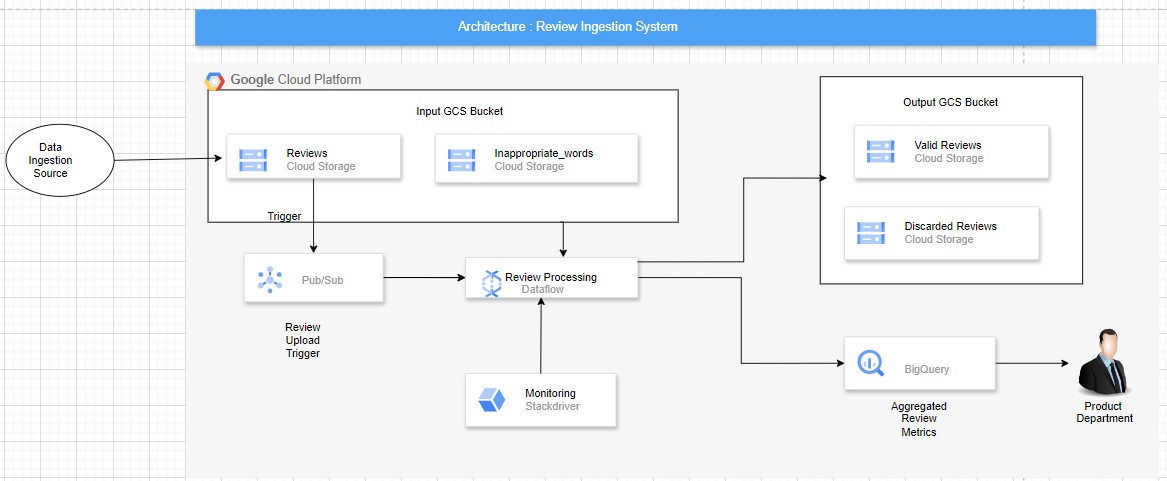
\* You are free to choose between a streaming or a batch solution. The minimum frequency is once a day.

\* The processed reviews must be written to a dedicated bucket/namespace of your BLOB storage system using a date(time) prefix.

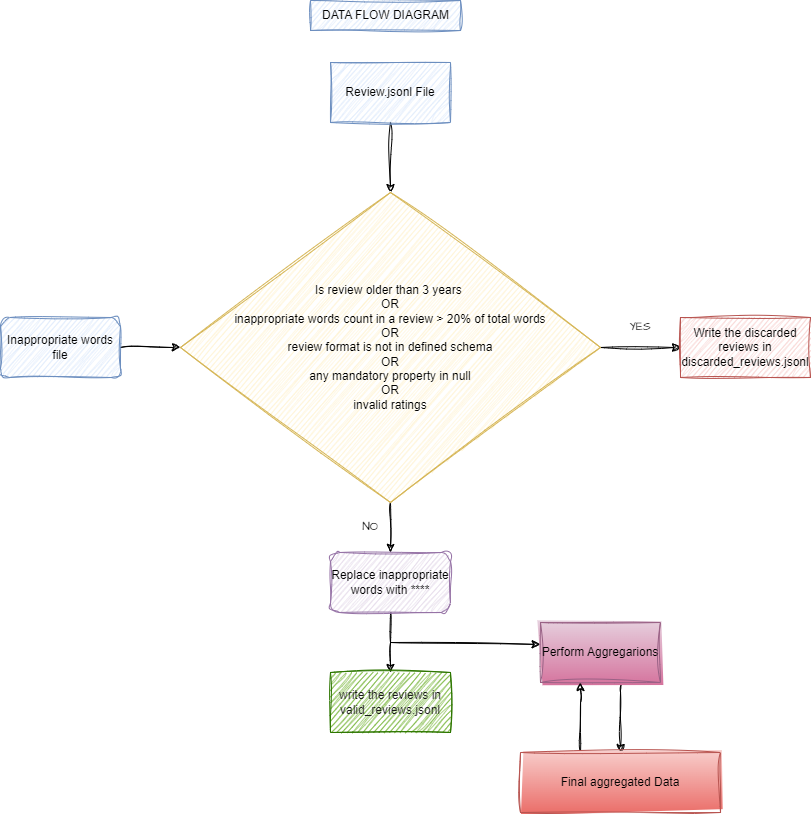
\* Discarded reviews should also be written to another dedicated bucket/namespace of your BLOB storage system.

**Considering all above assumptions, I have chosen GCP as my platform to implement this system.**

1. **Design Architecture**

****

**2. Data Flow Diagram**

**

**3. Solution Overview**

**Components**

1. **Google Cloud Storage (GCS)**: For storing incoming JSONL files, inappropriate words list, and processed results.
2. **Google Cloud Pub/Sub**: To handle event-driven ingestion of review files.
3. **Google Dataflow**: For processing and transforming review data.
4. **BigQuery**: For storing aggregated metrics and enabling analytics.
5. **Process Overview**
6. **File Arrival**: Review files are uploaded to a GCS bucket. New files trigger a Pub/Sub event.
7. **Ingestion Processing**:
   * **Load Inappropriate Words**: A dedicated Dataflow job reads the inappropriate words list from GCS.
   * **Review Filtering**: Dataflow processes incoming reviews to:
     + Filter out outdated reviews (older than 3 years).
     + Count inappropriate words and filter reviews exceeding 20%.
     + Replace inappropriate words with asterisks(\*\*\*\*).
8. **Aggregation**: Aggregate metrics are calculated per restaurant ID.
9. **Storage**:
   * Valid reviews are written back to a separate GCS bucket.
   * Discarded reviews are also stored in a different GCS bucket.
   * Aggregated metrics are stored in BigQuery for analytics.

**5. Detailed Design**

**Ingestion and Processing Workflow**

1. **Review Arrival**:
   * Reviews uploaded to GCS with a naming convention like reviews/yyyy-mm-dd.jsonl.
   * A Pub/Sub notification triggers the Dataflow job for processing.
2. **Dataflow Job**:
   * Reads the incoming JSONL files.
   * Loads the inappropriate words from GCS.
   * Filters reviews based on:
     + Age (using the publishedAt field).
     + Proportion of inappropriate words (calculates inappropriate count and compares to total words).
   * Replaces inappropriate words using regex.
   * Aggregates data for each restaurant and calculates the required metrics.
3. **Writing Results**:
   * Valid reviews are saved to GCS with a date prefix in a separate bucket, e.g., processed\_reviews/yyyy-mm-dd/.
   * Discarded reviews are saved in another dedicated bucket, e.g., discarded\_reviews/yyyy-mm-dd/.
   * Aggregated metrics are written to BigQuery table structured to accommodate the required fields. Also table is partitioned on ingestion\_time

**6.Scaling Strategies**

* **Batch Processing**: Dataflow can scale based on input file sizes and complexity. We can use autoscaling to handle varying loads.
* **BigQuery**: Designed to handle large datasets and can scale based on query complexity and load.
* **GCS**: Scales seamlessly as files are added.

**7. Deployment Strategy**

1. **Infrastructure as Code**: Use Terraform or Google Deployment Manager to provision resources (GCS buckets, Pub/Sub, Dataflow jobs, BigQuery datasets).
2. **CI/CD Pipeline**: Use Cloud Build to automate testing and deployment of Dataflow jobs.
3. **Monitoring and Alerts**: Set up Stackdriver for monitoring Dataflow jobs and creating alerts for failures or performance issues.

**8. Conclusion**

With this design we can implement a scalable and efficient Review Ingestion System on GCP that addresses all requirements while ensuring proper data handling and storage. In addition to these we have to ensure that all components comply with security best practices (IAM roles, encryption). Consider implementing a data retention policy for GCS to manage storage cost. We can also define quota and limits in Big Query to make efficient use of resources, prevent abuse and maintain the overall performance of the system.