Handling Write Data

Application (Rest API) gets the data and produces message to Kafka Stream

Website/

App

Load balancer gets data and distributes across applications

Kafka Stream 1

Yes

Is Write to DB Successful

Kafka Consumer/ consumes data and writes to DB and cache

cache

Consume Failed message and log its data

Yes

Kafka Stream 3

Yes

No

No

Failed for Nth time send to different stream

If failed publish back to Kafka.

Process message to write to DB.

Kafka Stream 2

Publish to a Kafka Stream For reprocessing

Cosmos or Cassandra NoSql DB

Read Data

Consume Kafka message and update the cache

cache

Kafka Stream 4

Get from Db, and send an event to Kafka for cache repair.

If cache miss

Cosmos or Cassandra NoSql DB

cache

Application (Rest API) gets the data.

Load balancer gets data and distributes across applications

Website/

App

To maintain a delay of less than one hour, scale the systems or applications such that all events are processed within one hour. If the lag of updates increases by more than an hour, keep alerts in place so that engineers are notified and can take necessary actions to control the lag.

**Maintain minimum downtime**

Run the application in different availability zones; therefore, all applications will not be down simultaneously. The preferred availability zones can be the one’s in close proximity of customers, taking operations into consideration.

**Maintain historical data**

After updating DB, push data to Hadoop clusters using spark jobs. Data can be retrieved from there whenever required.

Writing to DB is success

Cosmos or Cassandra NoSql DB

Applications Write to DB

Hadoop cluster

Spark Jobs to publish to Hadoop

The technology and framework that I will be using for the application are the Java language and Spring Boot Framework.