Enabling financial data in near real-time

Business vision was to enable financial data available in Mainframe DB2 tables in near real-time to have a **single master view,** to ensure **synchronization between downstream systems** and to **create a re-usable data asset**.

To achieve above business vision, below technical design was implemented

**Technical Design**

**1) Change Data Capture (CDC)** tool was implemented to capture the latest updates happening in DB2 tables and write it onto input Kafka topic in JSON format having before and after image of the data.

**Sample JSON message on input Kafka topic**

{

“tablename”:”VBNUK067F”,

“before”:

{

“id”:8

**"fruit": "Apple",**

"size": "Large",

"color": "Red"

}

“after”:

{

“id”:8

**"fruit": "Banana",**

"size": "Large",

"color": "Red"

}

}

**2) Apache Storm application integrated with Kafka** was developed to read the data from Kafka topic, a callback to DB2 was performed by calling a particular method inside micro service API (Connection to DB2 was made and having SQL queries with joins on various tables) with ‘id’ as input to bring out the historical data for a particular ‘id’, transform the data into JSON format and write into output Kafka topic for a downstream application to process.

Please note that micro service API URL was not called because of high latency and frequent failures due to network issues. Instead of API URL, a particular method was called by resolving dependency via pom.xml file.

**Important:** Here data on output Kafka topic is acting as single master view and a highly reusable component as it can be consumed by ‘n’ number of consumers.

**Code Snippet for calling micro service specific method**

**getFinancialDetails(id)** – Inside this method all required DB2 connection parameters and SQL queries were available to bring out the data from DB2 tables for a particular ‘id’.

**Code Snippet for DB2 connection**

**DB2SimpleDataSource class** was used for implementing dataSource and **org.apache.tomcat.jdbc.pool.DataSource** was used for implementing connection pooling in order to avoid increase even a single MIPS usage *(Connection Pooling was finalized and implemented after performing 4-5 months of non-functional testing by using a code without connection pooling which resulted in very high MIPS usage)* whenever callbacks to DB2 were performed.

To make a connection to DB2, key store and trust store were used by creating SSL certificates for passwordless connection to DB2 and to provide encryption during transit of data from DB2 to Kafka topic.

**Below dataSource DB2 parameters were used:**

.setTrustStoreLocation

.setKeyStoreLocation

.setSecurityMechanism

.setSSLCipherSuites

.setServerName

.setPortNumber

.setDatabaseName

.setCurrentFunctionPath

.setCurrentSchema

.setUser

**Implement Connection Pooling**

PoolProperties pool = new PoolProperties ()

pool.setDataSource (dataSource)

pool.setMaxActive (5)

pool.setJdbcInterceptors ()

apacheDataSource.setPoolProperties (pool)

**Sample JSON message on output Kafka topic (historical data)**

{

financial

{

[

**“id”:8**

**"fruit": "Apple",**

"size": "Large",

"color": "Red"

]

[

**“id”:8**

**"fruit": "Banana",**

"size": "Large",

"color": "Red"

]

}

}

**3) Spring Boot application integrated with Kafka (Downstream application)** was developed to read the data from output Kafka topic and update the target database if data is not already available in target database by applying various business rules.

**Important:** Here data in target database ensures the synchronization in near-real time with the data in source database DB2.

**4) Alerting script** was developed using UNIX shell scripting language to check the health of storm application after every regular intervals.

**Health of Storm application**

Consumer lag is calculated by subtracting committed offsets in zookeeper from Kafka offsets of a particular Kafka topic.

Threshold limit is set. If consumer lag breaches the threshold limit, then an alert is raised in order to start working on an issue as early as possible.

**Code Snippet for Alerting script**

Committed offset

/user/hadoop/current/zookeeper/bin/zk.sh –server “${zookeeper list}” get “${Kafka consumer group}”/”${partition number}” | grep ‘offset’ | cut –d’,’ –f3 | cut –d’:’ –f2)”

Kafka Offset of a particular Kafka topic

/user/hadoop/current/Kafka/bin/Kafka-run.sh Kafka.tools.GetOffsetShell –broker-list $Kafka brokers –topic topic\_name –security-protocol PLAINTEXTSASL –time -1 | grep topic\_name | cut –d’:’ –f3