The major components of this pipeline are the HBase tables, the StreamInsight framework to relay CDC updates to HBase, the Hive metatables and queries, and the Excel BI tools to report on the results of the Hive queries.

**HBase Table**

Steps to set up HBase table

1. Declare table with desired column families
   1. To access hbase shell:
      1. In Azure Management Portal, click on cluster -> configuration -> Connect (enable remote if necessary)
      2. Open Hadoop command line

**cd %HBASE\_HOME%/bin**

**hbase shell**

* 1. To create table as target for SQL Table/data to be translated and inserted into:

**create ‘ActivityPointerBase’, ‘cf’**

* 1. I only declared 1 column family, **cf**, for each table that I made. Then naming convention for HBase columns were then the respective column name in the SQL table, prefixed by **“cf:”**. For example, **ActivityId 🡪 cf:ActivityId, OwnerName 🡪 cf:OwnerName**, etc.

1. Each cell *value* of an HBase table is accessed by a *key, columnFamily:columnName.* 
   1. For example:

**scan ‘ActivityPointerBase’, {STARTROW=> ‘000002d5-06eb-e211-9’, ENDROW=> ‘000002d5-06eb-e211-9’}**

will return all values associated with that row key, which represents one row in ActivityPointerBase SQL table.

* 1. Likewise:

**put ‘ActivityPointerBase’, ‘000002d5-06eb-e211-9’, ‘cf:OwnerName’, ‘Andrew’**

will insert the string ‘Andrew’ into the associated key and columnFamiy:columnName cell.

* 1. In all tables set up, I used the SQL table’s primary key as the row key for HBase. In the case of ActivityPointerBase, it was ActivityId.

**StreamInsight/VS**

**Program.cs –** contains method main(). Takes care of server and process setup.

**SetUpDatabase.cs -** Before starting the process of polling the CDC table to reflect those changes in the HBase table, an instance of this class is used to take a “snapshot” of a table and transfer it into HBase. This should be a one-time execution. It must be properly parameterized with the Event type, HBase table name, HBase key, SqlConnection, and SqlQuery. The **capacity** passed into the begin() method determines how many rows are inserted per PUT requested. This should be higher for narrow tables, and lower for wide tables so as to optimize rows/second uploaded to HBase.

Example setup and execution:

SetUpDatabase<ActivityPointerBaseEventSrc> activitySetup = new SetUpDatabase<ActivityPointerBaseEventSrc>("ActivityPointerBase", "ActivityId", SQL\_COMMANDS.TIGER\_REPL\_CONN, SQL\_COMMANDS.ACTIVITY\_BASE);

Action<object> activityAction = (object obj) =>

{

activitySetup.begin(myApp, 200, "activityProcess");

};

Task activity = new Task(activityAction, "thisStringDoesntMatter");

activity.Start();

**ChangeDatabase.cs -** Instance to set up and execute the polling of a CDC table. An example of setting one up is as follows:

ChangeDatabase<PhoneBaseEventChange> phoneChange = new ChangeDatabase<PhoneBaseEventChange>("PhoneCallBase", "ActivityId", SQL\_COMMANDS.TIGER\_REPL\_CONN, SQL\_COMMANDS.PHONE\_CALL\_BASE\_CHANGE);

Action<object> phoneAction = (object obj) =>

{

phoneChange.begin(myApp, 200, "phoneChangeProcess");

};

Task t2 = new Task(phoneAction, "thisStringDoesntMatter");

t2.Start();

Note\*\*\*:

For each table in the SQL database that is being simultaneously maintained in HBase, an **\*Event\*.cs** class representing the table schema/columns. This allows proper serialization of incoming rows from SQL queries, and outgoing HBase inserts.

This project currently contains \*EventSrc and \*EventChange to serialize queries from “snapshot” transfers and polling/CDC transfers respectively. The \*EventChange class only contains two extra fields compared to \*EventSrc: TransationTime and Operation (update, insert, delete…). Ideally, these classes should be abstracted to a parent class to avoid code duplication, but I ran into serialization errors at runtime. I decided to keep separate classes to avoid this problem for the time being.

**ObservablePoller.cs** – This class takes care of querying the CDC table, interfacing with and delivering query results to StreamInsight, and scheduling reoccurring queries (polling). It is set up inside the ChangeDatabase.begin() method as the StreamInsight source.

**RequestManager.cs –** Class to perform requests against HBase REST API. It is set up and utilized in the ChangeDatabase.Sink() method.

**SQL\_COMMANDS.cs -** class to hold string constants. Consists of SQL queries and SQL DB connection strings.

**Hive - creating HBase backed external tables**

1. Once the table is created in HBase, and populated with data, you must create an *external* hive table backed by an HBase table. This makes the HBase table queryable using Hive Query Language.
2. Creating external Table for *ActivityPointerBase*

SET hbase.zookeeper.quorum=zookeepernode0,zookeepernode1,zookeepernode2;

CREATE EXTERNAL TABLE ActivityPointerBase(

ActivityId string ,

PhoneCategory string,

PhoneSubcategory string,

PhoneNumber string,

OwningBusinessUnit string,

ActualEnd string,

VersionNumber string,

IsBilled boolean,

….*[column name] [type],…..*

EmailSenderYomiName string,

SendersAccountYomiName string)

STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'

WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key, cf:PhoneCategory, cf:PhoneSubcategory, cf:PhoneNumber, cf:OwningBusinessUnit, cf:ActualEnd, cf:VersionNumber, cf:IsBilled, ………*[cf:ColumnName]….*, cf:EmailSenderYomiName, cf:SendersAccountYomiName')

TBLPROPERTIES ('hbase.table.name' = 'ActivityPointerBase');

**Setting up Hive Queries and Oozie**

Create table files: **create\_activity\_pointer\_base.hql, create\_phone\_call\_base.hql**

The create table statements were done through the (azure manage cluster 🡪 hive editor) window since they are only needed to be executed one time.

The query files and following \*.xml are uploaded via **oozie\_prep.ps1**.

HQL Query files: **phone\_call\_base\_query.hql, join\_activity\_phone.hql**

**workflow.xml** dictates the order in which the the uploaded query files are executed, and **coordinator.xml** acts as the scheduler to set the frequency, start time, and stop time for when these jobs are to be run.

The oozie job outlined with coordinator/workflow is started by running **oozie\_run.ps1**.

The hive queries will be run as part of the oozie job. The results will be stored in the directory path stated in the hive query, i.e.

‘insert overwrite directory /pathToResults’

**Power Query to Power Pivot / Power View**

The following steps are how to import Hive Query results using Power Query

1. Under Power Query tab, click **From Other Sources 🡪 From Windows Azure HDInsight**
2. Enter the name of the storage account associated with the cluster
3. Enter the key associated with the storage account
   1. The key can be found in the Azure Management Portal. Click on the storage account, and click “Manage Keys”
4. In the resulting Navigator Pane, click on the cluster name
5. A list of cluster contents will appear. Since the Hive results are written into specified directories, like “JoinActivityPhoneQuery” for example, you can filter by folder path to find the files that contains those specific query results.
6. After filtering, the contents will be part of the binaries labeled **00000\_0, 00000\_1**, etc., depending on how large the query is.
7. Click on the **binary** of the contents, and open the file as a CSV document.
8. This view will show the contents in a table form.
9. You must rename the columns so they are more readable.
10. Most of the types are as strings, so some of them can be changed so they work better with power pivot and power view down the line, like text 🡪 DateTime for example.
11. Rename the results, and “Apply and Close” to load contents into Excel Workbook.
12. Once the relevant queries are loaded into the workbook, you can add relationships and calculated metrics using Power Pivot, and generate reports using Power View.