INCREMENTAL INGESTION PIPELINE POC (STREAMSET AND AIRFLOW)
Overview

A common problem for client on-boarding is getting data into the Hadoop ecosystem quickly and incrementally so it can be queried as soon as possible. The default tooling of Sqoop, Flume, HDFS file manipulation is hard to work with, in part because it stops data ingestion the moment it encounters any error. Insight provides a template to quickly assist clients in ingesting data and bringing that data into Hadoop for Hive/Impala query.

Insight’s ingestion pipeline uses StreamSets and Airflow to provide a reliable data import mechanism with some useful properties. The final Hive/Impala table populated by the incremental update template can be queried for the latest data by doing a self join to get the rows with the most current timestamp. This lets customers quickly learn about the most recent data received.
In addition, ingestion won’t be halted at the first sign of trouble.

- Built into the StreamSets toolset is the idea of absorbing field "drift". Input data fields change over time. If new fields are added, StreamSets keeps working. If fields change names, StreamSets simply stops mapping the changed fields, but it keeps working.
- StreamSets has record-based error handling so bad record errors do not stop the pipeline. Error records are "smartly" handeled and put into a log, queue, or any store that is desired.
- StreamSets data preview features allow the quick diagnosis of issues with incoming data.
- When the next new source of data comes, StreamSets’ multiple input plugins and standardized way of handling pipeline data allow a quick swap of the input and reuse of any other existing pipeline steps.
Airflow + StreamSets?

A good question to ask at this point is why not choose a single tool to do everything? Maybe Airflow or just StreamSets is complete on its own.

- Airflow provides the key job management tools often critical but missing from the traditional Hadoop ingestion pipeline. Once data is loaded on the Hadoop ecosystem, data usually requires some final bulk manipulation using Impala, Hive, or HDFS commands. Airflow can schedule, execute, and track any task using a variety of generic CLI or Python-based tools.
• Airflows great UI allows excellent runtime management of your Hadoop ecosystem jobs.

• StreamSets fills a very specific niche in the Hadoop ecosystem for ETL (Extract, Transform and Load).
All the common sources and destinations to and from Hadoop are included without the need to code connection clients, transformers, and clients to load data. The transport format of data in the pipeline is handled by StreamSets without additional development. It also provides monitoring and record-level error handling. StreamSets also provides the ability to do robust transforms out of the box and to use dynamic languages to handle more complex transforms. All nodes in the flow can introspect the data using the expression language built into the tool. Quick preview testing is built into the tool, so the user is not left guessing if there are any errors in the complete ingest chain when making changes. End to end integration tests of single records can be run through the chain without mutating the input or output systems at either end.

StreamSets, however, is not a generic orchestration engine. Doing gross manipulation of whole datasets and tables is not part of the tool. StreamSets pipeline nodes are all focused on record level "stream" data, so while there are generic Javascript and Jython (aka Java Python) pipeline processors, their core purpose is to process records in the pipeline as they arrive in small batches.

Large batch queries like inserting a table into a partition are not efficient to do as individual records. StreamSets is very fast at writing data directly into HDFS in known formats. Hundreds of thousands of records per minute is common. This is a big advantage over the streaming insert mechanism for Hive, HBase, or Cassandra where handling a few records at a time may only process 300 records a minute. Taken in parallel using many processes this might be tolerable, but it is not an efficient solution from a hardware and management standpoint.

StreamSets provides a good way of getting data sanitized with no errors onto HDFS and then from there Airflow can pick up that data in bulk and do native bulk data manipulation queries to move that data into final locations.

**Other Ingestion Tools Examined**

Cask Hydrator and Luigi were also examined as possible ingestion pipeline tools.

Cask Hydrator operates within the Cask ecosystem, not as a stand-alone ingestion tool. Cask within the scope of a Cloudera Hadoop instance has a very large footprint, requiring 10 processors and 14 Gb of memory. Such large server hardware isn't appropriate for some clients and limits the possible install options. This ruled out Cask Hydrator as the ingestion pipeline, though it does many of the same things as StreamSets.

Luigi is an alternative to Airflow for generic job scheduling. Airflow presents individual jobs and tasks in a format that’s easier to explore and with log data attached directly to runs of job/task. Setup is faster, and general job definition is easier with Airflow. Both tools are python-based job schedulers, but Airflow is just easier to use.
Proof of Concept

The POC pipeline consists of a StreamSets pipeline that takes the data from the MySQL table and looks at its update_date column to select records for incremental updates. This pipeline writes this data into an HDFS as a set of AVRO files. This stream is configured to write data into a "YYYY-MM-DD-mm" directory and create a new sub file every 10k records. The files are then picked up by a Airflow DAG (Directed Acyclic Graph - aka a job).

The end target table is the partitioned AVRO table below. The Airflow DAG schedules the run of a python job script.

The python job script uses Python clients for HDFS, Hive, and Impala to:

1. read the HDFS file system to list all the directories written out into the /user/hive/warehouse/madeup directory by StreamSets
2. process the contents of every directory that does not end with the ".bak" suffix. This consists of
   1. creating a Temp table pointing at the directory as a "external table"
   2. inserting the contents of that Temp table as a partition into the "madeuptable" below
   3. dropping the Temp table
   4. renaming HDFS directory to add the ".bak" suffix
CREATE TABLE madeuptable (  
id BIGINT,  
foo STRING,  
bar STRING,  
update_epoch BIGINT,  
delete_epoch BIGINT  
)  
partitioned by (dt string)  
STORED AS AVRO  
TBLPROPERTIES ('avro.schema.url'='hdfs://quickstart/user/schemas/madeup/avroschema.json')

The StreamSets Ingestion Pipeline

StreamSets has 3 very useful capabilities that simplify the creation of ingestion pipelines.

1. StreamSets handles the incremental update without any custom code. The two most common mechanisms are tracking an identifier that always increments up on a "table" and keeping track of new "entries" in a filesystem directory. The choice depends on the input needs of the client. StreamSets manages this record offset in its own meta-store, so there is no need keep track of processed files or keeping track of offsets in other stores. It is built in.
2. StreamSets has record-level error handling, so if a record doesn't process for any reason it can be directed to an error handler for later review. Error handlers can be logs, queues, files, or something custom if you want to write it. This is a vast improvement over killing a whole batch because a single record fails, which is how other ingestion tools like Sqoop, Flume, and Hive work.
3. StreamSets does name-to-name data mapping automatically with no need for manual mapping.

The StreamSets workflow can poll the database table for updates and feed them to the HDFS filesystem with minimal configuration. The simplest case is when the Database Table has a modification datetime column that can be used as the offset "column" to pick up all updated rows in a table since the last execution of the pipeline.

Note that in the POC the update_date column of the MySQL table is a datetime type while the ingested table is a BIGINT using an epoch time instead. There are 2 reasons this translation was done:

1. AVRO format files don't have a datetime type, so any dates either need to converted to BIGINT or STRING.
2. StreamSets offset increment tracking works best with integer type increments. Integers provide a clear "counting up" sequence that may not be guaranteed by other types.
   1. When querying the MySQL table for incremental updates, all dates are translated into epoch times. This provides an integer value for comparison with the offset.
   2. Files can be written into known file locations on HDFS in whatever format is easy to ingest for Hive or HDFS. The POC uses the AVRO file format since Impala Hadoop can be used to create External Tables pointed at AVRO files.
# ${OFFSET} is a replacement variable used by StreamSets of feed the offset into the query for the next run. Note all queries of the JDBC # StreamSet source need to be minimally ordered by the "offset" column which in this case the the "update_epoch" select * from (select id, foo, bar, unix_timestamp(delete_date) as delete_epoch, unix_timestamp(update_date) as update_epoch from madeuptable) as fred where fred.update_epoch > ${OFFSET} order by fred.update_epoch;

**Airflow DAG**

The Airflow DAG has a simple job: it runs the Python Job Script and keeps track of that task’s status. If more granularity or visibility is required, it would be simple to break the Python job script into 3 separate tasks that would be displayed and tracked individually in Airflow.

**Python Main Script**

The Python Job Script does bulk data movement to put data ingested by StreamSets into a Hive/Impala store. At the end of the StreamSet pipeline HDFS contains AVRO files, a well known data format used by Impala and Hive. Note that Hive and Impala are query engines and metadata managers pointing at data in files. Before they can use the data, these tools require declaration of tables through Hive/Impala DDL and loading of data through queries. The queries are required to integrate new data into the store with each incremental update. Hive/Impala cannot automatically recognize the new data just because properly formatted files were deposited in the directory structure for Hive/Impala.

The Python script uses standard Python client libraries for Hive, Impala, and HDFS to create temporary tables from the StreamSets AVRO files, do bulk data movement into partitions, and move files and clean up temporary tables. The POC partitioning strategy is based on the time of ingestion but this could just as easily be changed to a date-based partitioning strategy related to last modified time of the data in the source system. As mentioned above, this script is managed by Airflow.
Value to Insight Clients

StreamSets and Airflow enables developers to spend less time determining how to ingest data and spend more time querying data.

The ingestion POC is a template that can be followed for any Insight client that needs incremental ingestion of data. StreamSets and Airflow present a small install footprint (memory and CPU requirements are low), which lowers the investment in hardware. Both tools give excellent visibility into the status of ingestion to engineers in the form of monitoring dashboards and alerting. When errors do occur, the tools handle issues gracefully without breaking ingestion. The POC with StreamSets and Airflow provides a fast to implement, fault tolerant and well monitored pipeline to any Insight client from any StreamSet source.
About Clairvoyant

Clairvoyant is a product and technology services organization based out of Chandler, Arizona. Clairvoyant was established in 2012 by four engineers, with a goal to build high performing engineering talent in the Phoenix, Arizona area. The Governor of Arizona awarded Clairvoyant start-up of the year for the year 2014 for the remarkable growth and community presence in Arizona. Clairvoyant grew to a team of about 60 members based out of Arizona and the San Francisco Bay area as well as a team of 30 offshore resources operating from India.

Clairvoyant has a very active presence in the local Phoenix engineering community. Clairvoyant conducts a monthly Hadoop meet-up group as well as an annual data conference focusing on analytics, Hadoop and machine learning. Several Clairvoyant engineers are frequent speakers at various engineering-focused venues in Arizona.

Clairvoyant provides a suite of services in building large-scale analytical solutions using Hadoop, various open source technologies, cloud computing and data products. Clairvoyant currently provides services to higher education, legal analytics, health care, mortgage, and banking sectors focusing on building enterprise applications, consumer products and platforms.

In slightly more than two years of existence Clairvoyant has established a reputation for executing complex projects very efficiently and in a timely fashion. Clairvoyant resources mainly operate from its Chandler, Arizona location helping its clients to have a lower cost of engagement.