Overview

The Internet of Things (IoT) generates a large amount of data that can be used to provide real-time and/or historic insights into sensor measurements. The design and operating environment of these devices presents a unique set of challenges for ingesting data into big data systems.

This paper describes a reference architecture for using StreamSets Data Collector to move IoT sensor data into Hadoop. Specifically, we will cover two patterns, Real-Time Streaming of IoT Events into Hadoop and Batch-Oriented Transfer of Historic or Master Data.

Pattern 1: Real-Time Streaming of IoT Events into Hadoop

IoT devices generate a large amount of event data. There are a number of challenges you must contend with when ingesting this data:

- You may have to handle multiple device firmware versions as you perform rolling upgrades in the field, and each firmware revision may have slightly different data payloads with different schemas.
- You may have to deal with occasional bad data if sensors on some devices are damaged or have drifted out of spec.
- As you stream the data into your destination system you may want to perform lookups to enrich the data such as determining which product or location corresponds to a given device.
- You may be required to sanitize data and filter out personally identifiable information (PII) that is not allowed to be stored in some governance zones.

In this paper we will show you how to simplify IoT data movement with StreamSets Data Collector so you can avoid hand coding pipelines, which can be brittle in the face of change.
Figure 2. Real-time Processing of Streaming IoT Data

No matter what IoT devices and sensors produce the data, the backend architecture remains largely the same. There are three main components as shown in Figure 2:

1. A “first mile” solution, usually an MQTT message broker or pub/sub mechanism such as RabbitMQ, Solace Systems, Kafka or MapR Streams, that accepts incoming data from devices.
2. A StreamSets Data Collector pipeline that routes, cleans and enriches data.
3. A Hadoop cluster that allows for further processing and analysis of the data.

Figure 3. A StreamSets Data Collector IoT ingest pipeline shown in execution mode as it validates and transforms data streams before dropping them into Hadoop.

StreamSets Data Collector provides a drag and drop UI to design, test and operate data flow pipelines. The system is built for continuous operations and can consume data from any number of streaming sources including Cloudera Kafka, Apache Kafka, MapR Streams, RabbitMQ, AWS Kinesis and others. There are built-in transformation processors that let you apply higher-order sanitization functions in-stream such as field conversions, splitting, merging, hashing, masking, parsing and lookups. This palette of processors is constantly growing; check the documentation for the latest list. If you want to apply custom business logic you can use the Groovy, JavaScript or Jython processors.
Finally, Java-based processor stages can be built and deployed using an API (see Building Customer Processor Stages Using the StreamSets Maven Archetype for details).

The StreamSets Data Collector pipeline performs all of its transformations in memory and guarantees ordered delivery using At Least Once or At Most Once delivery semantics. The DevOps friendly integrated development environment (IDE) lets you create, test and run pipelines that transform your raw streaming IoT data into a consumption-ready dataset on-the-fly, immediately accessible for analysis or visualization.

In execution mode, you get excellent runtime visibility into the health of your data flows, including throughput, error rates, and processing time for each stage in the pipeline. You can set up threshold-based rules and alerts for warnings on when processing rates have slowed or your data values have become anomalous.

Figure 4. Some of the StreamSets Data Collector runtime metrics as shown in execution mode.

Common Issues with IoT Streaming Ingest

Some of the issues you will face in a large IoT deployment include bad data seeping in as devices age, the reality of numerous device versions spread across the installed base, the need to enrich the data before it lands in the data store, and of course the requirement that the system scale of extremely large numbers of sensor streams.
Handling bad data

With hundreds of thousands of devices in the field it is inevitable that occasionally a sensor will be poorly calibrated or due to age or damage drift out of spec, producing erroneous data. You will want to address such errors before the data reaches the data store.

*Figure 5: Bad data can occur due sensor damage or degradation.*

StreamSets Data Collector will detect the problem and display helpful error messages, the specific error record and a stack-trace of the failure condition if applicable. It does this without stopping the main pipeline. All error records can be saved to disk or sent to a secondary pipeline that flows to a Kafka/MapR Streams topic or Elasticsearch index for remediation.

The Reality of Firmware Multiplicity in the Field

In IoT it is inevitable you have a mix of devices in the field. Consider devices logging temperature and humidity data on freight containers transporting perishable produce. Firmware Version 1 of the device reports its firmware version, unique device id, unique container id and temperature/humidity sensor readings. After a larger number of these devices are deployed in the field a new firmware revision or a device upgrade will likely be pushed out on a rolling basis. Perhaps Firmware Version 2 adds a new set of sensors (Accelerometer, Gyroscope and Compass) that are used to calculate roll, pitch and yaw to detect if the package has tipped over. A R&D group may also be field testing Firmware Version 3, that includes GPS, adding even more fields to the data payload. The operational reality is that all three device versions (shown in Figure 6) will be concurrently deployed and sending data.

*Figure 6. Different firmware versions produce different data payloads.*
Since StreamSets Data Collector is schema-agnostic, it will allow for this variation across payloads. The ‘intent oriented’ nature of the pipeline means you only specify conditions for the data that you are interested in processing; other fields and new fields are passed through without stopping the pipeline. While this example uses JSON files, StreamSets Data Collector also works with a large number of structured, semi-structured and unstructured data formats such as Text, CSV, Delimited Files, XML, Avro, protobuf and many more.

**Enriching Data in Stream**

To assist in making your sensor data consumption-ready, you may want to enrich it with data from other sources. In the case below we want to designate the product the freight container is carrying.

```
{  
  "firmware": "1.0",  
  "device_id": "824D67",  
  "product_id": "2931355",  
  "product_description": "Flavorosa Fluots",  
  "container_id": "AD04BA42",  
  "reading_date": "1463967641",  
  "temperature": "1.5",  
  "temp_unit": "C",  
  "relative_humidity": "56"  
}
```

Instead of performing an expensive Join operation on the data in the destination store, StreamSets lets you perform a lookup to enrich the incoming data and add fields while ingesting the stream, so that it consumption ready when it reaches the data store.

*Figure 7. Adding Product ID and Description by looking up master data.*

**Processing Data at Scale**

To support thousands or millions of devices out in the field you need a scalable architecture to ensure high performance, high availability and failover, ideally one that can scale with your existing Hadoop cluster.

*Figure 6. StreamSets Data Collector - Deployment Options*

StreamSets Data Collector can work in both Standalone and In-Cluster deployments. It can execute as standalone nodes reading off a cluster of RabbitMQ nodes or directly deployed on a Hadoop cluster reading off say multiple Kafka partitions. When deployed on a Hadoop cluster StreamSets
Data Collector utilizes the underlying YARN or Mesos framework to deploy pipelines as Spark Streaming applications. This allows it to scale to virtually any processing demand.

**Pattern 2: Batch-Oriented Transfer of Historic or Master Data**

The second pattern to consider is introduction of batch data to the IoT data flow. When processing or visualizing streaming IoT data you may need to use historical data to aid in analytics. Since StreamSets Data Collector supports batch and streaming data this is a straightforward operation.

![Batch Processing Data with Data Collector](image)

**Figure 7. Batch Processing Data with Data Collector**

It makes no distinction between batch and streaming mode during pipeline design. If you use a batch-oriented Origin stage such as a file directory, database or Amazon S3 bucket, the system automatically reads data in a batch mode. The pipeline semantics remain identical.

**Processing and Visualizing Data on Hadoop**

StreamSets Data Collector can deposit files into HDFS or stream data directly into Hive (ORC Files) or Kudu. You can use fast query engines such as Impala or Kudu and use Spark, Python, R to run complex analytics or machine learning jobs on the data. You can also use visualization tools such as ZoomData to create graphs easily without any code, or Jupyter (formerly iPython) notebooks to process and visualize data.

**Final Considerations and Conclusions**

Ingesting real-time IoT data into Hadoop can be challenging. Hand coded solutions or legacy ETL tools are not designed for the combination of scale and shifting data structures that are typical in IoT use cases, nor do they offer the visibility, metrics and diagnostic capabilities required for a real-time at-scale operation. Trying to force fit them into an IoT use case can lead to longer design cycles, brittle data flow pipelines and difficulties in pinpointing and fixing operational problems. Modern tools such as StreamSets Data Collector were built to address the unique challenges of IoT and other
emerging data flow use cases. We hope these examples will stimulate your thinking around how you might build your own IoT-to-Hadoop data flow.

About StreamSets
StreamSets was founded in 2014, by Girish Pancha, former chief product officer of Informatica, and Arvind Prabhakar, an early employee and engineering leader at Cloudera. They formed the company to address the growing challenge of managing data in motion in a world marked by constant change — to data sources, to data processing infrastructure and to the data itself, a problem called data drift. StreamSets was created to re-envision the management of data flows from the ground up, avoiding the pitfalls of legacy offerings and enabling a new way to manage data in motion.

The company has offices in San Francisco and Silicon Valley and is backed by top-tier Silicon Valley venture capital firms, including Accel Partners, Battery Ventures, Ignition Partners and New Enterprise Associates (NEA). Its industry-leading partners include Cloudera, Elastic and MapR.

Its first product, StreamSets Data Collector, is used by hundreds of companies, including Fortune 500 firms across a variety of industries such as financial services, manufacturing, healthcare, media, pharmaceuticals and technology.

Download the StreamSets Data Collector
Click here to download the open source StreamSets Data Collector.