Building a Streaming Data Platform on AWS Activities 1-3

Radhika Ravirala, Solutions Architect, Amazon Web Services

10/26/2016

Pop-up Loft

Activity 1
Create and Populate a Kinesis Stream

Activity 1: Create and populate a Kinesis Stream

We are going to:

A. Create an Amazon Kinesis stream

B. Create an AWS IAM user for writing to the stream

C. Write data to a Kinesis Stream

Activity 1-A: Create an Amazon Kinesis stream

1. Go to the Amazon Kinesis Streams Console
    1. https://us-west-2.console.aws.amazon.com/kinesis/home
2. Click “Create Stream”
3. Specify a stream name and number of shards (2), click “Create”

Activity 1-B: Create an AWS IAM user

Create a AWS IAM user and attach policy

1. Go to the AWS IAM console, click “Users”
    1. https://console.aws.amazon.com/iam/home?region=us-west-2#users
2. Specify user name (i.e “kinesis_producer”), create, and download credentials
3. Go to “Users” and click the user name you just created
4. Click “Permissions” tab and then “Attach Policy”
5. Search “Kinesis” and attach the AWS Managed Policy “AmazonKinesisFullAccess” to the user.

Activity 1-C: Write data to a Kinesis Stream (pt 1)

To simplify this, we are going to use simple tool for creating sample data and writing it to a Kinesis Stream.

1. Go to the simple data producer UI http://bit.ly/kinesis-producer
2. Copy and past your credentials into the form. The credentials do not leave your client (locally hosted site). Additionally, the AWS Javascript SDK is implemented in the site which uses HTTPS. In normal situations, you *do not* want to use your credentials in this way unless you trust the provider.
3. Specify region (us-west-2) and stream name.
4. Specify a data rate between 100 – 500, pick a unique number. This will control the rate at which records are sent to the stream.

Activity 1-C: Write data to a Kinesis Stream (pt 2)

5. Copy and paste this into the text editor

```
{
  "sensorId": "sensor-{{random.number(100)}}",
  "eventTimeStamp": "{{current.timestamp}}",
  "ambientTemp": {{random.number({"min":-40, "max":60})}},
  "measuredTemp": {{random.number({"min":0, "max":100})}}
}
```

The tool will generate JSON documents based upon the above “record format”. The tool generates hundreds of these records and send them in a single put using PutRecords. Finally, the tool will drain your battery. Only keep it running while we are completing the activities if you don’t have a power cord.

Review – Kinesis Shards and Streams

You created a two shard stream in this activity.

- How much throughput does that shard support?

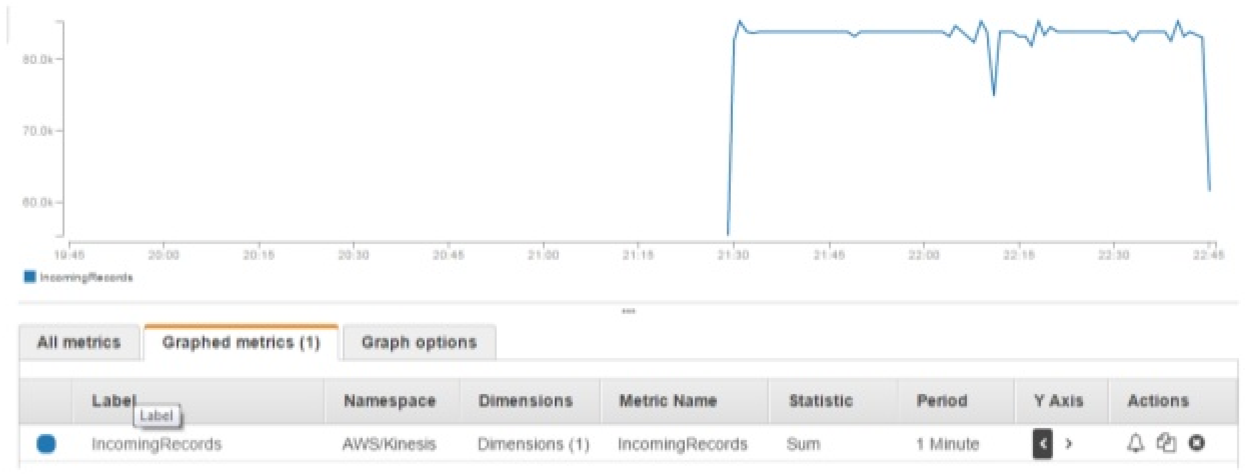
- How do we determine which shard each record is saved?

- What is a good logical partition key?

Review – Monitoring a Stream

Go to the [Amazon CloudWatch Metrics Console](#) and search “IncomingRecords”. Select this metric for your Kinesis Stream and choose a 1 Minute SUM.

How do you determine which records are going to which shards?



	Label	Namespace	Dimensions	Metric Name	Statistic	Period	Y Axis	Actions
●	IncomingRecords	AWS/Kinesis	Dimensions (1)	IncomingRecords	Sum	1 Minute	‹ ›	

Activity 2
Process Data using Kinesis Analytics

Activity 2: Process Data using Kinesis Analytics

We are going to:

A. Create a Kinesis Analytics applications that reads from the Kinesis stream with IoT Sensor data
B. Add your first query
C. Add a continuous filter
D. Calculate an aggregate metric for an interesting statistic on the incoming data
E. Stop the application

Activity 2-A: Create Kinesis Analytics Application

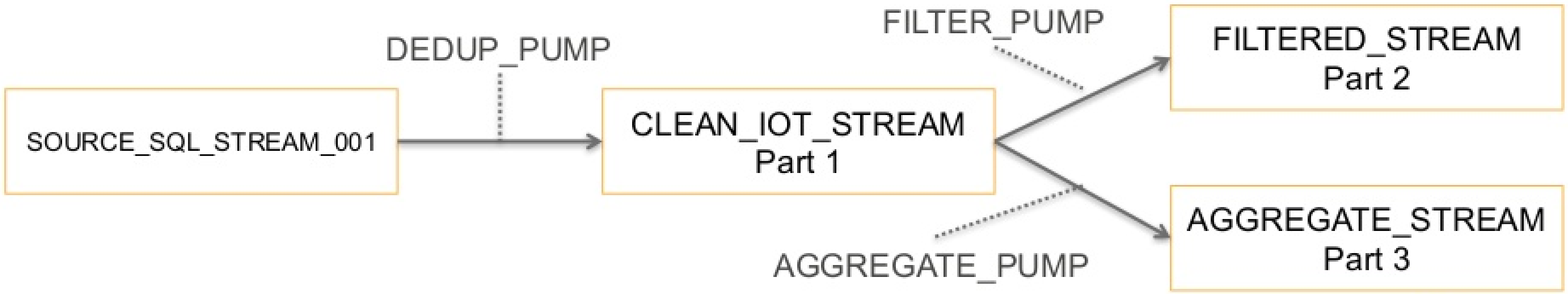
1. Go to the Amazon Kinesis Analytics Console
    1. https://us-west-2.console.aws.amazon.com/kinesisanalytics/home?region=us-west-2
2. Click “Create new application” and provide a name
3. Click “Connect to a source” and select the stream you created in the previous exercise
4. Kinesis Analytics will discover the schema for you. Verify the schema and click “Save and Continue”
5. Click “Go To SQL Editor” and then “Run Application”

You now have a running stream processing application!

Writing SQL over Streaming Data

Writing SQL over streaming data using Kinesis Analytics follows a two part model:

1. Create an in-application stream for storing intermediate SQL results. An in-application stream is like a SQL table, but is continuously updated.
2. Create a PUMP which will continuously read FROM one in-application stream and INSERT INTO a target in-application stream



Activity 2-B: Add your first query (part 1)

Our first query will use SELECT DISTINCT to remove any producer-side duplicates. Producer-side duplicates are caused by connectivity issues and producer retires. In the SQL Editor, create a target in-application stream to store the intermediate SQL results:

```
CREATE OR REPLACE STREAM CLEAN_IOT_STREAM (
    ingest_time TIMESTAMP,
    sensor_id VARCHAR(16),
    event_timestamp VARCHAR(16),
    ambient_temp DECIMAL(16,2),
    measured_temp DECIMAL(16,2),
    dedup_window TIMESTAMP);
```

Activity 2-B: Add your first query (part 2)

Perform a SELECT DISTCINT over a 1 second window, inserting into your in-application stream. Pay attention to lower and upper case when selecting columns (use quotes).

```
CREATE OR REPLACE PUMP DEDUP_PUMP AS
INSERT INTO CLEAN_IOT_STREAM
SELECT STREAM DISTINCT APPROXIMATE_ARRIVAL_TIME,
"sensorId", "eventTimeStamp", "ambientTemp",
"measuredTemp",
   FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO SECOND)
FROM SOURCE_SQL_STREAM_001;
```

Activity 2-B: Add your first query (part 3)

This query removes any producer side duplicates over a one second window. (In a production setting, this window would likely be larger like 10 seconds). Producer side duplicates are most often created when an HTTP request times out due to connectivity issues on the producer.

The window is defined bye the following statement in the SELECT statement. Note that the ROWTIME column is implicitly included in every stream query, and represents the processing time of the application.

```
    FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO SECOND)
```

Activity 2-C: Add a continuous filter (part 1)

A continuous filter is a common query that uses a WHERE clause to select a portion of your data stream. Create a target in-application stream for the filtered data.

```
CREATE OR REPLACE STREAM FILTERED_STREAM (
    ingest_time TIMESTAMP,
    sensor_id VARCHAR(16),
    event_timestamp VARCHAR(16),
    ambient_temp DECIMAL(16,2),
    measured_temp DECIMAL(16,2),
    dedup_window TIMESTAMP);
```

Activity 2-C: Add a continuous filter (part 2)

Create a pump that INSERTs into your FILTERED_STREAM, selects FROM your CLEAN_IOT_STREAM, and uses a WHERE clause on the difference between ambient and measured temperature.

```
CREATE OR REPLACE PUMP FILTER_PUMP AS
INSERT INTO FILTERED_STREAM
SELECT STREAM ingest_time, sensor_id, event_timestamp,
ambient_temp, measured_temp, dedup_window
FROM CLEAN_IOT_STREAM
WHERE abs(measured_temp - ambient_temp) > 20;
```

Activity 2-D: Calculate an aggregate metric (pt 1)

The last part of this activity is free form. Calculate a count using a tumbling window and a GROUP BY clause.

A tumbling window is similar to a periodic report, where you specify your query and a time range, and results are emitted at the end of a range. (EX: COUNT number of items by key for 10 seconds)

Activity 2-D: Calculate an aggregate metric (pt 2)

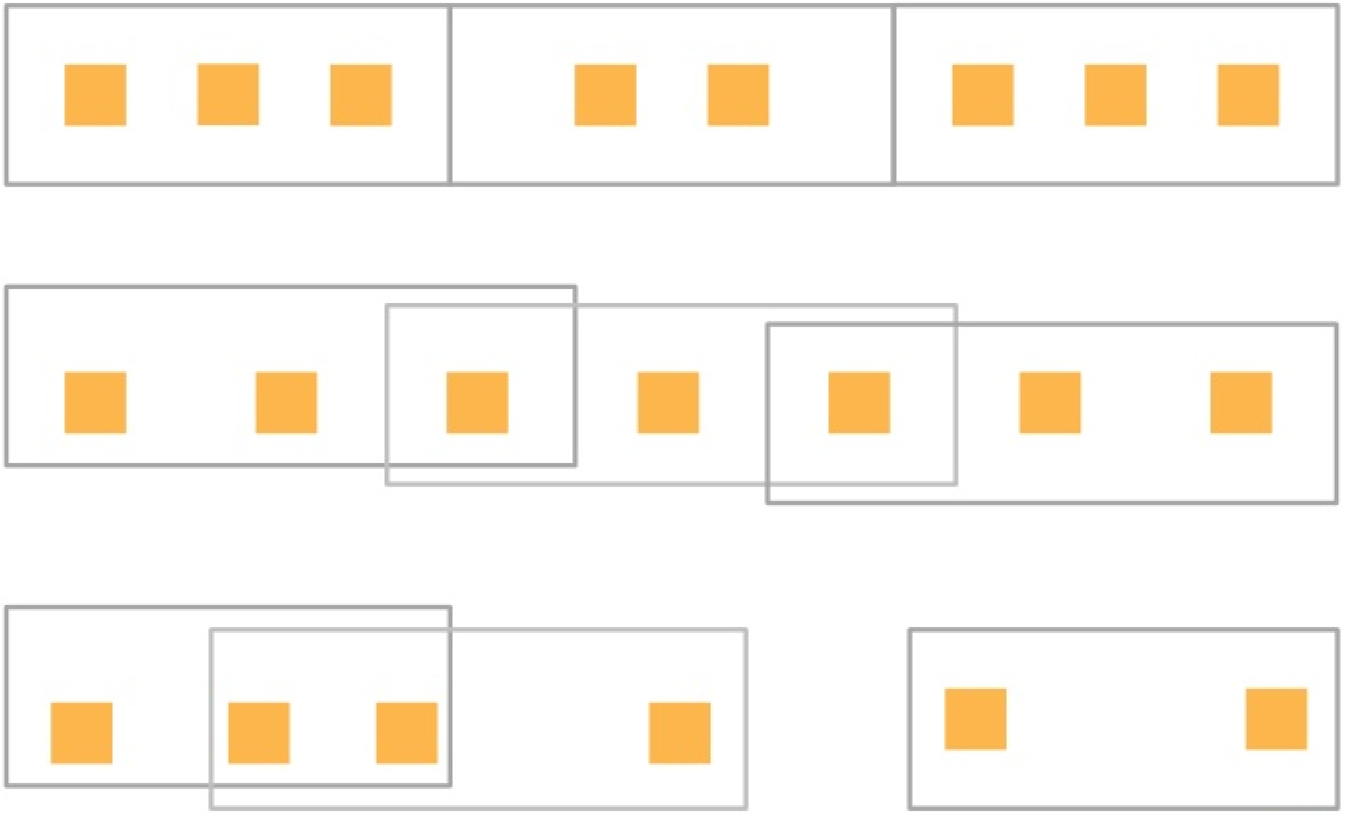
```
CREATE OR REPLACE STREAM AGGREGATE_STREAM (sensor_id
VARCHAR(16), avg_ambient_temp DECIMAL(16,2),
avg_measured_temp DECIMAL(16,2), measurements INTEGER);

CREATE OR REPLACE PUMP AGGREGATE_PUMP AS INSERT INTO
AGGREGATE_STREAM
SELECT STREAM sensor_id, AVG(ambient_temp),
AVG(measured_temp), COUNT(sensor_id)
FROM CLEAN_IOT_STREAM
GROUP BY sensor_id, FLOOR((CLEAN_IOT_STREAM.ROWTIME -
TIMESTAMP '1970-01-01 00:00:00') SECOND / 10 TO SECOND);
```

Different types of windows

Tumbling

Sliding

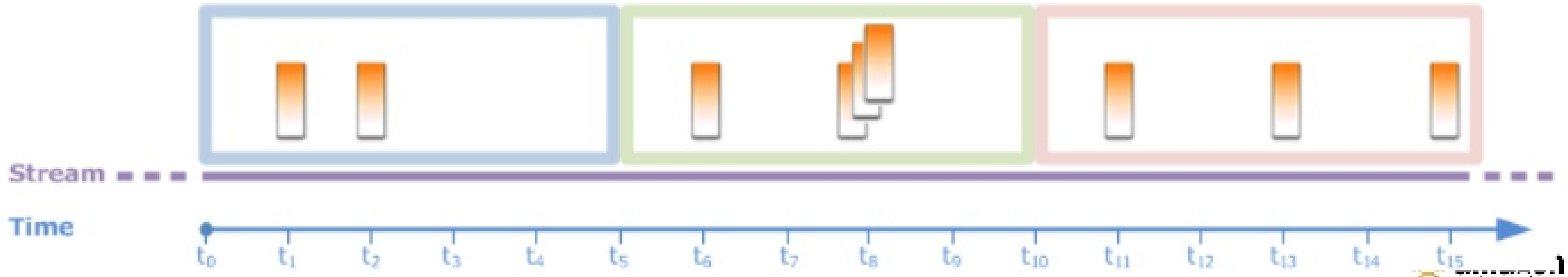
Custom



Tumbling Windows

Tumbling windows are useful for periodic reports. You can use a tumbling window to compute an average number of visitors to your website in the last 5 minutes, or the maximum over the past hour. A single result is emitted for each key in the group as specified by the clause at the end of the defined window.

An important characteristic of a tumbling window is that the bounds do not overlap; the start of a new tumbling window begins with the end of the old window. The below image visually captures how a 5-minute tumbling window would aggregate results, in separate windows that are not overlapping. The blue, green, and red windows represent minutes 0 to 5, 5 to 10, and 10 to 15, respectively.



Tumbling Windows in Group By

Tumbling windows are always included in a GROUP BY clause and use the FLOOR function. The FLOOR function takes the lowest possible integer for a given input. To create a tumbling time window, convert a timestamp to an integer representation and put it in a FLOOR function to create the window bound. Three examples are shown below:

Activity 2-E: Stop your application

1. Go to the Amazon Kinesis Analytics main dashboard.
2. Select your application.
3. Click “Actions” and then “Stop Application”.

Review – In-Application SQL streams

Your application has multiple in-application SQL streams including CLEAN_IOT_STREAM, FILTERED_STREAM, and AGGREGATE_STREAM. These in-application streams which are like SQL tables that are continuously updated.

What else is unique about an in-application stream aside from its continuous nature?

Activity 3

Deliver streaming data to S3

Activity 3: Deliver Data to S3 using Firehose

We are going to:

A. Create a Firehose delivery stream

B. Update Kinesis Analytics app

C. Check the S3 bucket for results

D. Clean up

Activity 3-A: Create Firehose delivery stream

1. Go to the [Amazon Kinesis Firehose](https://console.aws.amazon.com/firehose/home?region=us-west-2#/intro) dashboard, click “Create Delivery Stream”.
    1. https://console.aws.amazon.com/firehose/home?region=us-west-2#/intro
    2. Use a different name then the one you used for your Kinesis stream.
2. Select Amazon S3 for your destination, create a new bucket in Oregon, and name it (i.e. <yourname>-filtered-iot-data). Click “Next”
3. Choose Configuration options
    1. Choose a buffer size of 5 MB, and 60 seconds
    2. Choose No compression or encryption
    3. Turn on Error Logging
    4. Select “Create new IAM Role” and then “Allow”
    5. Click “Next”
4. Review and click “Create Delivery Stream”

Activity 3-B: Update Kinesis Analytics app (part 1)

1. Go to the Kinesis Analytics Console, select your application
2. Go to the SQL Editor, click “No, I’ll do this later”
3. Identify the name of the in-application stream you used for the filtered results (i.e. FILTERED_STREAM).
4. Click “Exit (done editing)”.
5. Click “Connect to a destination”.

Activity 3-B: Update Kinesis Analytics app (part 2)

5. Configure your destination
    1. Choose the Firehose delivery stream you just created
    2. **Edit the ".. refer to this stream as" to the in-application stream containing the filtered results.**
    3. Choose CSV as the "Output format"
    4. Click "Save and continue"
6. Go back to the Kinesis Analytics dashboard, select your application, and click "Run Application"

Activity 3-C: Check the S3 bucket for results

1. Go to the [Amazon S3 console](https://console.aws.amazon.com/s3/home?region=us-west-2#)
    1. https://console.aws.amazon.com/s3/home?region=us-west-2#
2. Select the bucket you created in Activity 3-C (i.e. filtered-iot-data
3. Navigate down the folder hierarchy
4. Select an object and click “Action”, “Download”
5. Open up the file with a text editor

Review – Reliable delivery to a destination

You are successfully delivering data to S3.

What are possible issues that would prevent Firehose from delivering data?

If something went wrong, where would you look?

Activity 3-D: Clean up

1. Stop the data producer
2. Go to the Kinesis Analytics console, delete your application
3. Go to the Kinesis Streams console, delete your stream
4. Go to the Kinesis Firehose console, delete your deliver stream
5. Go to the Amazon S3 console, right-click on your bucket and choose “Delete Bucket”

Some Next Steps

We have many AWS Big Data Blogs which cover more examples. Full list here. Some good ones:

1. Kinesis Streams
    1. Implement Efficient and Reliable Producers with the Amazon Kinesis Producer Library
    2. Presto and Amazon Kinesis
    3. Querying Amazon Kinesis Streams Directly with SQL and Sparking Streaming
    4. Optimize Spark-Streaming to Efficiently Process Amazon Kinesis Streams
2. Kinesis Firehose
    1. Persist Streaming Data to Amazon S3 using Amazon Kinesis Firehose and AWS Lambda
    2. Building a Near Real-Time Discovery Platform with AWS
3. Kinesis Analytics
    1. Writing SQL on Streaming Data With Amazon Kinesis Analytics Part 1 | Part 2
    2. Real-time Clickstream Anomaly Detection with Amazon Kinesis Analytics

Reference

- **Technical documentations**
  - Amazon Kinesis Agent
  - Amazon Kinesis Streams and Spark Streaming
  - Amazon Kinesis Producer Library Best Practice
  - Amazon Kinesis Firehose and AWS Lambda
  - Building Near Real-Time Discovery Platform with Amazon Kinesis
- **Public case studies**
  - Glu mobile – Real-Time Analytics
  - Hearst Publishing – Clickstream Analytics
  - How Sonos Leverages Amazon Kinesis
  - Nordstorm Online Stylist