Install a Maven library with the coordinates listed below. These coordinates have been taken from MVNRepository, which can be found through the following URL: (https://mvnrepository.com/artifact/com.microsoft.azure/azure-eventhubs-spark):

com.microsoft.azure:azure-eventhubs-spark\_2.12:2.3.15

The following code will build a connection string using the IoT Hub connection details and start the structured stream. Replace the IoT Hub connections in the following example before running the code. Also, remember to verify the consumer group in the code based on what was defined in the IoT Hub in Azure Portal.

import org.apache.spark.eventhubs.\_

import org.apache.spark.eventhubs.{ ConnectionStringBuilder, EventHubsConf, EventPosition }

import org.apache.spark.sql.functions.{ explode, split }

// To connect to an Event Hub, EntityPath is required as part of the connection string.

// Here, we assume that the connection string from the Azure portal does not have the EntityPath part.

val connectionString = ConnectionStringBuilder("—Event Hub Compatible Endpoint--")

.setEventHubName("—Event Hub Compatible Name--")

.build

val eventHubsConf = EventHubsConf(connectionString)

.setStartingPosition(EventPosition.fromEndOfStream)

.setConsumerGroup("delta")

val eventhubs = spark.readStream

.format("eventhubs")

.options(eventHubsConf.toMap)

.load()

Run the following code to display stream details.

display(eventhubs)

Since data is now streaming, create a Spark SQL Table by running the following code. Remember to define the columns based on your IoT Device data. For this demo, use columns body and sequenceNumber from the device simulator.

import org.apache.spark.sql.types.\_

import org.apache.spark.sql.functions.\_

val schema = (new StructType)

.add("body", DoubleType)

.add("sequence\_number", DoubleType)

val df = eventhubs.select(($"enqueuedTime").as("Enqueued\_Time"),($"systemProperties.iothub-connection-device-id")

.as("Device\_ID"),(from\_json($"body".cast("string"), schema)

.as("telemetry\_json"))).select("Enqueued\_Time","Device\_ID", "telemetry\_json.\*")

Run the following code to create the Spark SQL table to store the device telemetry data.

df.createOrReplaceTempView("device\_telemetry\_data")

Write the stream to a Delta Table and start by running the following code to define the final Dataframe.

val finalDF = spark.sql("Select Date(Enqueued\_Time) Date\_Enqueued, Hour(Enqueued\_Time) Hour\_Enqueued, Enqueued\_Time, Device\_ID, body AS Body,sequence\_number as Sequence\_Number from device\_telemetry\_data")

This next block of code shown in Figure 17-17 will write the stream to the delta table. Notice that you can define the partitions, format, checkpoint location and output mode. A default checkpoint location is being used, which is defined and managed by Databricks, but you could just as easily define this location yourself and persist the data to a different folder.

finalDF.writeStream

.outputMode("append")

.option("checkpointLocation", "/delta/events/\_checkpoints/device\_delta")

.format("delta")

.partitionBy("Date\_Enqueued", "Hour\_Enqueued")

.table("delta\_telemetry\_data")

Here are a few examples of [triggers](https://spark.apache.org/docs/2.3.0/structured-streaming-programming-guide.html#triggers). Apache Spark’s documentation contains more details about triggers and can be found within the following URL: https://spark.apache.org/docs/2.3.0/structured-streaming-programming-guide.html#triggers

.trigger(Trigger.ProcessingTime("2 seconds"))

.trigger(Trigger.Once())

.trigger(Trigger.Continuous("1 second"))

Write the following SQL query shown in Figure 17-19 to retrieve the delta table that the structured streaming data is writing into. This table can then be used to perform additional advanced analytics and/or build machine learning models to gain further valuable real-time insights into the IoT Device data.

%sql

SELECT \*  
FROM   delta\_telemetry\_data