complex-nested-structured(Scala)

**Five Spark SQL Helper Utility Functions to Extract and Explore Complex Data Types**

* get\_json\_object()
* from\_json()
* to\_json()
* explode()
* selectExpr()

The takeaway from this short tutorial is myriad ways to slice and dice nested JSON structures with Spark SQL utility functions.

Let's create a simple JSON schema with attributes and values, without any nested structures.

**import** org.apache.spark.sql.types.**\_** // include the Spark Types to define our schema

**import** org.apache.spark.sql.functions.**\_** // include the Spark helper functions

**val** jsonSchema = **new** StructType()

.add("battery\_level", LongType)

.add("c02\_level", LongType)

.add("cca3",StringType)

.add("cn", StringType)

.add("device\_id", LongType)

.add("device\_type", StringType)

.add("signal", LongType)

.add("ip", StringType)

.add("temp", LongType)

.add("timestamp", TimestampType)

import org.apache.spark.sql.types.\_ import org.apache.spark.sql.functions.\_ jsonSchema: org.apache.spark.sql.types.StructType = StructType(StructField(battery\_level,LongType,true), StructField(c02\_level,LongType,true), StructField(cca3,StringType,true), StructField(cn,StringType,true), StructField(device\_id,LongType,true), StructField(device\_type,StringType,true), StructField(signal,LongType,true), StructField(ip,StringType,true), StructField(temp,LongType,true), StructField(timestamp,TimestampType,true))

Using the schema above, create a Dataset, represented as a Scala case type, and generate some JSON data associated with it. In all likelihood, this JSON might as well be a stream of device events read off a Kafka topic. Note that the case class has two fields: integer (as a device id) and a string (as a JSON string representing device events).

**Create a Dataset from the above schema**

// define a case class

**case** **class** DeviceData (id: Int, device: String)

// create some sample data

**val** eventsDS = Seq (

(0, """{"device\_id": 0, "device\_type": "sensor-ipad", "ip": "68.161.225.1", "cca3": "USA", "cn": "United States", "temp": 25, "signal": 23, "battery\_level": 8, "c02\_level": 917, "timestamp" :1475600496 }"""),

(1, """{"device\_id": 1, "device\_type": "sensor-igauge", "ip": "213.161.254.1", "cca3": "NOR", "cn": "Norway", "temp": 30, "signal": 18, "battery\_level": 6, "c02\_level": 1413, "timestamp" :1475600498 }"""),

(2, """{"device\_id": 2, "device\_type": "sensor-ipad", "ip": "88.36.5.1", "cca3": "ITA", "cn": "Italy", "temp": 18, "signal": 25, "battery\_level": 5, "c02\_level": 1372, "timestamp" :1475600500 }"""),

(3, """{"device\_id": 3, "device\_type": "sensor-inest", "ip": "66.39.173.154", "cca3": "USA", "cn": "United States", "temp": 47, "signal": 12, "battery\_level": 1, "c02\_level": 1447, "timestamp" :1475600502 }"""),

(4, """{"device\_id": 4, "device\_type": "sensor-ipad", "ip": "203.82.41.9", "cca3": "PHL", "cn": "Philippines", "temp": 29, "signal": 11, "battery\_level": 0, "c02\_level": 983, "timestamp" :1475600504 }"""),

(5, """{"device\_id": 5, "device\_type": "sensor-istick", "ip": "204.116.105.67", "cca3": "USA", "cn": "United States", "temp": 50, "signal": 16, "battery\_level": 8, "c02\_level": 1574, "timestamp" :1475600506 }"""),

(6, """{"device\_id": 6, "device\_type": "sensor-ipad", "ip": "220.173.179.1", "cca3": "CHN", "cn": "China", "temp": 21, "signal": 18, "battery\_level": 9, "c02\_level": 1249, "timestamp" :1475600508 }"""),

(7, """{"device\_id": 7, "device\_type": "sensor-ipad", "ip": "118.23.68.227", "cca3": "JPN", "cn": "Japan", "temp": 27, "signal": 15, "battery\_level": 0, "c02\_level": 1531, "timestamp" :1475600512 }"""),

(8 ,""" {"device\_id": 8, "device\_type": "sensor-inest", "ip": "208.109.163.218", "cca3": "USA", "cn": "United States", "temp": 40, "signal": 16, "battery\_level": 9, "c02\_level": 1208, "timestamp" :1475600514 }"""),

(9,"""{"device\_id": 9, "device\_type": "sensor-ipad", "ip": "88.213.191.34", "cca3": "ITA", "cn": "Italy", "temp": 19, "signal": 11, "battery\_level": 0, "c02\_level": 1171, "timestamp" :1475600516 }"""),

(10,"""{"device\_id": 10, "device\_type": "sensor-igauge", "ip": "68.28.91.22", "cca3": "USA", "cn": "United States", "temp": 32, "signal": 26, "battery\_level": 7, "c02\_level": 886, "timestamp" :1475600518 }"""),

(11,"""{"device\_id": 11, "device\_type": "sensor-ipad", "ip": "59.144.114.250", "cca3": "IND", "cn": "India", "temp": 46, "signal": 25, "battery\_level": 4, "c02\_level": 863, "timestamp" :1475600520 }"""),

(12, """{"device\_id": 12, "device\_type": "sensor-igauge", "ip": "193.156.90.200", "cca3": "NOR", "cn": "Norway", "temp": 18, "signal": 26, "battery\_level": 8, "c02\_level": 1220, "timestamp" :1475600522 }"""),

(13, """{"device\_id": 13, "device\_type": "sensor-ipad", "ip": "67.185.72.1", "cca3": "USA", "cn": "United States", "temp": 34, "signal": 20, "battery\_level": 8, "c02\_level": 1504, "timestamp" :1475600524 }"""),

(14, """{"device\_id": 14, "device\_type": "sensor-inest", "ip": "68.85.85.106", "cca3": "USA", "cn": "United States", "temp": 39, "signal": 17, "battery\_level": 8, "c02\_level": 831, "timestamp" :1475600526 }"""),

(15, """{"device\_id": 15, "device\_type": "sensor-ipad", "ip": "161.188.212.254", "cca3": "USA", "cn": "United States", "temp": 27, "signal": 26, "battery\_level": 5, "c02\_level": 1378, "timestamp" :1475600528 }"""),

(16, """{"device\_id": 16, "device\_type": "sensor-igauge", "ip": "221.3.128.242", "cca3": "CHN", "cn": "China", "temp": 10, "signal": 24, "battery\_level": 6, "c02\_level": 1423, "timestamp" :1475600530 }"""),

(17, """{"device\_id": 17, "device\_type": "sensor-ipad", "ip": "64.124.180.215", "cca3": "USA", "cn": "United States", "temp": 38, "signal": 17, "battery\_level": 9, "c02\_level": 1304, "timestamp" :1475600532 }"""),

(18, """{"device\_id": 18, "device\_type": "sensor-igauge", "ip": "66.153.162.66", "cca3": "USA", "cn": "United States", "temp": 26, "signal": 10, "battery\_level": 0, "c02\_level": 902, "timestamp" :1475600534 }"""),

(19, """{"device\_id": 19, "device\_type": "sensor-ipad", "ip": "193.200.142.254", "cca3": "AUT", "cn": "Austria", "temp": 32, "signal": 27, "battery\_level": 5, "c02\_level": 1282, "timestamp" :1475600536 }""")).toDF("id", "device").as[DeviceData]

defined class DeviceData eventsDS: org.apache.spark.sql.Dataset[DeviceData] = [id: int, device: string]

Our Dataset is collection of Scala case classes, and when displayed as DataFrame, there are two columns (id, string)

display(eventsDS)

|  |  |
| --- | --- |
| 0 | {"device\_id": 0, "device\_type": "sensor-ipad", "ip": "68.161.225.1", "cca3": "USA", "cn": "United States", "temp": 25, "signal": 23, "battery\_level": 8, "c02\_level": 917, "timestamp" :1475600496 } |
| 1 | {"device\_id": 1, "device\_type": "sensor-igauge", "ip": "213.161.254.1", "cca3": "NOR", "cn": "Norway", "temp": 30, "signal": 18, "battery\_level": 6, "c02\_level": 1413, "timestamp" :1475600498 } |
| 2 | {"device\_id": 2, "device\_type": "sensor-ipad", "ip": "88.36.5.1", "cca3": "ITA", "cn": "Italy", "temp": 18, "signal": 25, "battery\_level": 5, "c02\_level": 1372, "timestamp" :1475600500 } |
| 3 | {"device\_id": 3, "device\_type": "sensor-inest", "ip": "66.39.173.154", "cca3": "USA", "cn": "United States", "temp": 47, "signal": 12, "battery\_level": 1, "c02\_level": 1447, "timestamp" :1475600502 } |
| 4 | {"device\_id": 4, "device\_type": "sensor-ipad", "ip": "203.82.41.9", "cca3": "PHL", "cn": "Philippines", "temp": 29, "signal": 11, "battery\_level": 0, "c02\_level": 983, "timestamp" :1475600504 } |
| 5 | {"device\_id": 5, "device\_type": "sensor-istick", "ip": "204.116.105.67", "cca3": "USA", "cn": "United States", "temp": 50, "signal": 16, "battery\_level": 8, "c02\_level": 1574, "timestamp" :1475600506 } |
| 6 | {"device\_id": 6, "device\_type": "sensor-ipad", "ip": "220.173.179.1", "cca3": "CHN", "cn": "China", "temp": 21, "signal": 18, "battery\_level": 9, "c02\_level": 1249, "timestamp" :1475600508 } |
| 7 | {"device\_id": 7, "device\_type": "sensor-ipad", "ip": "118.23.68.227", "cca3": "JPN", "cn": "Japan", "temp": 27, "signal": 15, "battery\_level": 0, "c02\_level": 1531, "timestamp" :1475600512 } |
| 8 | {"device\_id": 8, "device\_type": "sensor-inest", "ip": "208.109.163.218", "cca3": "USA", "cn": "United States", "temp": 40, "signal": 16, "battery\_level": 9, "c02\_level": 1208, "timestamp" :1475600514 } |
| 9 | {"device\_id": 9, "device\_type": "sensor-ipad", "ip": "88.213.191.34", "cca3": "ITA", "cn": "Italy", "temp": 19, "signal": 11, "battery\_level": 0, "c02\_level": 1171, "timestamp" :1475600516 } |
| 10 | {"device\_id": 10, "device\_type": "sensor-igauge", "ip": "68.28.91.22", "cca3": "USA", "cn": "United States", "temp": 32, "signal": 26, "battery\_level": 7, "c02\_level": 886, "timestamp" :1475600518 } |
| 11 | {"device\_id": 11, "device\_type": "sensor-ipad", "ip": "59.144.114.250", "cca3": "IND", "cn": "India", "temp": 46, "signal": 25, "battery\_level": 4, "c02\_level": 863, "timestamp" :1475600520 } |
| 12 | {"device\_id": 12, "device\_type": "sensor-igauge", "ip": "193.156.90.200", "cca3": "NOR", "cn": "Norway", "temp": 18, "signal": 26, "battery\_level": 8, "c02\_level": 1220, "timestamp" :1475600522 } |
| 13 | {"device\_id": 13, "device\_type": "sensor-ipad", "ip": "67.185.72.1", "cca3": "USA", "cn": "United States", "temp": 34, "signal": 20, "battery\_level": 8, "c02\_level": 1504, "timestamp" :1475600524 } |
| 14 | {"device\_id": 14, "device\_type": "sensor-inest", "ip": "68.85.85.106", "cca3": "USA", "cn": "United States", "temp": 39, "signal": 17, "battery\_level": 8, "c02\_level": 831, "timestamp" :1475600526 } |
| 15 | {"device\_id": 15, "device\_type": "sensor-ipad", "ip": "161.188.212.254", "cca3": "USA", "cn": "United States", "temp": 27, "signal": 26, "battery\_level": 5, "c02\_level": 1378, "timestamp" :1475600528 } |
| 16 | {"device\_id": 16, "device\_type": "sensor-igauge", "ip": "221.3.128.242", "cca3": "CHN", "cn": "China", "temp": 10, "signal": 24, "battery\_level": 6, "c02\_level": 1423, "timestamp" :1475600530 } |
| 17 | {"device\_id": 17, "device\_type": "sensor-ipad", "ip": "64.124.180.215", "cca3": "USA", "cn": "United States", "temp": 38, "signal": 17, "battery\_level": 9, "c02\_level": 1304, "timestamp" :1475600532 } |
| 18 | {"device\_id": 18, "device\_type": "sensor-igauge", "ip": "66.153.162.66", "cca3": "USA", "cn": "United States", "temp": 26, "signal": 10, "battery\_level": 0, "c02\_level": 902, "timestamp" :1475600534 } |
| 19 | {"device\_id": 19, "device\_type": "sensor-ipad", "ip": "193.200.142.254", "cca3": "AUT", "cn": "Austria", "temp": 32, "signal": 27, "battery\_level": 5, "c02\_level": 1282, "timestamp" :1475600536 } |
| **id** | **device** |

Printing the schema atests to two columns of type integer and string, reflecting the Scala case class.

eventsDS.printSchema

root |-- id: integer (nullable = false) |-- device: string (nullable = true)

**How to use get\_json\_object()**

This method extracts a JSON object from a JSON string based on JSON path specified, and returns a JSON string as the extracted JSON object. Take the small example of the above dataset, from which we wish to extract portions of data values that make up the JSON object string. Say you want to extract only id, device\_type, ip, and CCA3 code. Here's a quick way to do it using get\_json\_object().

**val** eventsFromJSONDF = Seq (

(0, """{"device\_id": 0, "device\_type": "sensor-ipad", "ip": "68.161.225.1", "cca3": "USA", "cn": "United States", "temp": 25, "signal": 23, "battery\_level": 8, "c02\_level": 917, "timestamp" :1475600496 }"""),

(1, """{"device\_id": 1, "device\_type": "sensor-igauge", "ip": "213.161.254.1", "cca3": "NOR", "cn": "Norway", "temp": 30, "signal": 18, "battery\_level": 6, "c02\_level": 1413, "timestamp" :1475600498 }"""),

(2, """{"device\_id": 2, "device\_type": "sensor-ipad", "ip": "88.36.5.1", "cca3": "ITA", "cn": "Italy", "temp": 18, "signal": 25, "battery\_level": 5, "c02\_level": 1372, "timestamp" :1475600500 }"""),

(3, """{"device\_id": 3, "device\_type": "sensor-inest", "ip": "66.39.173.154", "cca3": "USA", "cn": "United States", "temp": 47, "signal": 12, "battery\_level": 1, "c02\_level": 1447, "timestamp" :1475600502 }"""),

(4, """{"device\_id": 4, "device\_type": "sensor-ipad", "ip": "203.82.41.9", "cca3": "PHL", "cn": "Philippines", "temp": 29, "signal": 11, "battery\_level": 0, "c02\_level": 983, "timestamp" :1475600504 }"""),

(5, """{"device\_id": 5, "device\_type": "sensor-istick", "ip": "204.116.105.67", "cca3": "USA", "cn": "United States", "temp": 50, "signal": 16, "battery\_level": 8, "c02\_level": 1574, "timestamp" :1475600506 }"""),

(6, """{"device\_id": 6, "device\_type": "sensor-ipad", "ip": "220.173.179.1", "cca3": "CHN", "cn": "China", "temp": 21, "signal": 18, "battery\_level": 9, "c02\_level": 1249, "timestamp" :1475600508 }"""),

(7, """{"device\_id": 7, "device\_type": "sensor-ipad", "ip": "118.23.68.227", "cca3": "JPN", "cn": "Japan", "temp": 27, "signal": 15, "battery\_level": 0, "c02\_level": 1531, "timestamp" :1475600512 }"""),

(8 ,""" {"device\_id": 8, "device\_type": "sensor-inest", "ip": "208.109.163.218", "cca3": "USA", "cn": "United States", "temp": 40, "signal": 16, "battery\_level": 9, "c02\_level": 1208, "timestamp" :1475600514 }"""),

(9,"""{"device\_id": 9, "device\_type": "sensor-ipad", "ip": "88.213.191.34", "cca3": "ITA", "cn": "Italy", "temp": 19, "signal": 11, "battery\_level": 0, "c02\_level": 1171, "timestamp" :1475600516 }""")).toDF("id", "json")

eventsFromJSONDF: org.apache.spark.sql.DataFrame = [id: int, json: string]

display(eventsFromJSONDF)

|  |  |
| --- | --- |
| 0 | {"device\_id": 0, "device\_type": "sensor-ipad", "ip": "68.161.225.1", "cca3": "USA", "cn": "United States", "temp": 25, "signal": 23, "battery\_level": 8, "c02\_level": 917, "timestamp" :1475600496 } |
| 1 | {"device\_id": 1, "device\_type": "sensor-igauge", "ip": "213.161.254.1", "cca3": "NOR", "cn": "Norway", "temp": 30, "signal": 18, "battery\_level": 6, "c02\_level": 1413, "timestamp" :1475600498 } |
| 2 | {"device\_id": 2, "device\_type": "sensor-ipad", "ip": "88.36.5.1", "cca3": "ITA", "cn": "Italy", "temp": 18, "signal": 25, "battery\_level": 5, "c02\_level": 1372, "timestamp" :1475600500 } |
| 3 | {"device\_id": 3, "device\_type": "sensor-inest", "ip": "66.39.173.154", "cca3": "USA", "cn": "United States", "temp": 47, "signal": 12, "battery\_level": 1, "c02\_level": 1447, "timestamp" :1475600502 } |
| 4 | {"device\_id": 4, "device\_type": "sensor-ipad", "ip": "203.82.41.9", "cca3": "PHL", "cn": "Philippines", "temp": 29, "signal": 11, "battery\_level": 0, "c02\_level": 983, "timestamp" :1475600504 } |
| 5 | {"device\_id": 5, "device\_type": "sensor-istick", "ip": "204.116.105.67", "cca3": "USA", "cn": "United States", "temp": 50, "signal": 16, "battery\_level": 8, "c02\_level": 1574, "timestamp" :1475600506 } |
| 6 | {"device\_id": 6, "device\_type": "sensor-ipad", "ip": "220.173.179.1", "cca3": "CHN", "cn": "China", "temp": 21, "signal": 18, "battery\_level": 9, "c02\_level": 1249, "timestamp" :1475600508 } |
| 7 | {"device\_id": 7, "device\_type": "sensor-ipad", "ip": "118.23.68.227", "cca3": "JPN", "cn": "Japan", "temp": 27, "signal": 15, "battery\_level": 0, "c02\_level": 1531, "timestamp" :1475600512 } |
| 8 | {"device\_id": 8, "device\_type": "sensor-inest", "ip": "208.109.163.218", "cca3": "USA", "cn": "United States", "temp": 40, "signal": 16, "battery\_level": 9, "c02\_level": 1208, "timestamp" :1475600514 } |
| 9 | {"device\_id": 9, "device\_type": "sensor-ipad", "ip": "88.213.191.34", "cca3": "ITA", "cn": "Italy", "temp": 19, "signal": 11, "battery\_level": 0, "c02\_level": 1171, "timestamp" :1475600516 } |
| **id** | **json** |

**val** jsDF = eventsFromJSONDF.select($"id", get\_json\_object($"json", "$.device\_type").alias("device\_type"),

get\_json\_object($"json", "$.ip").alias("ip"),

get\_json\_object($"json", "$.cca3").alias("cca3"))

jsDF: org.apache.spark.sql.DataFrame = [id: int, device\_type: string ... 2 more fields]

display(jsDF)

|  |  | |  |  | |
| --- | --- | --- | --- | --- | --- |
| 0 | sensor-ipad | | 68.161.225.1 | USA | |
| 1 | sensor-igauge | | 213.161.254.1 | NOR | |
| 2 | sensor-ipad | | 88.36.5.1 | ITA | |
| 3 | sensor-inest | | 66.39.173.154 | USA | |
| 4 | sensor-ipad | | 203.82.41.9 | PHL | |
| 5 | sensor-istick | | 204.116.105.67 | USA | |
| 6 | sensor-ipad | | 220.173.179.1 | CHN | |
| 7 | sensor-ipad | | 118.23.68.227 | JPN | |
| 8 | sensor-inest | | 208.109.163.218 | USA | |
| 9 | sensor-ipad | | 88.213.191.34 | ITA | |
| **id** | | | **device\_type** | **ip** | **cca3** |

**How to use from\_json()**

A variation of get\_json\_object(), this function uses schema to extract individual columns. Using *from\_json()* helper function within the *select()* Dataset API call, I can extract or decode data's attributes and values from a JSON string into a DataFrame as columns, dictated by a schema. As well using the schema, I ascribe all associated atrributes and values within this JSON to represent as an entity *devices*. As such, not only can you use the device.attribute to retrieve its respective value but also all values using the *\** notation.

In example below:

* Uses the schema above to extract from the JSON string attributes and values and represent them as individual columns as part of devices
* select() all its columns
* Filters on desired attributes using the . notation

Once you have extracted data from a JSON string into its respective DataFrame columns, you can apply DataFrame/Dataset APIs calls to select, filter, and subsequtly display, to your satisfaction.

**val** devicesDF = eventsDS.select(from\_json($"device", jsonSchema) as "devices")

.select($"devices.\*")

.filter($"devices.temp" > 10 and $"devices.signal" > 15)

devicesDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [battery\_level: bigint, c02\_level: bigint ... 8 more fields]

display(devicesDF)

0.002040608010023182516262017270.002040608010023182516262017270.002040608010023182516262017270.002040608010023182516262017270.002040608010023182516262017270.00204060801002318251626201727TOOLTIPUSANORITACHNINDAUTsignaltempcca3USANORITACHNINDAUT

**val** devicesUSDF = devicesDF.select($"\*").where($"cca3" === "USA").orderBy($"signal".desc, $"temp".desc)

devicesUSDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [battery\_level: bigint, c02\_level: bigint ... 8 more fields]

display(devicesUSDF)

2623201716100%0%0%0%0%22%20%27%31%0%0%0%0%49%51%0%0%0%0%100%signal2623201716sensor-igaugesensor-ipadsensor-inestsensor-istick

**How to use to\_json()**

Now, let's do the reverse: you can convert or encode our filtered devices into JSON string using to\_json(). That is, convert a JSON struct into a string. The result can be republished, for instance, to Kafka or saved on disk as parquet files. To learn how to write to Kafka and other sinks, read this blog and our series on Structured Streaming blogs.

**val** stringJsonDF = eventsDS.select(to\_json(struct($"\*"))).toDF("devices")

stringJsonDF: org.apache.spark.sql.DataFrame = [devices: string]

display(stringJsonDF)

|  | |
| --- | --- |
| {"id":0,"device":"{\"device\_id\": 0, \"device\_type\": \"sensor-ipad\", \"ip\": \"68.161.225.1\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 25, \"signal\": 23, \"battery\_level\": 8, \"c02\_level\": 917, \"timestamp\" :1475600496 }"} | |
| {"id":1,"device":"{\"device\_id\": 1, \"device\_type\": \"sensor-igauge\", \"ip\": \"213.161.254.1\", \"cca3\": \"NOR\", \"cn\": \"Norway\", \"temp\": 30, \"signal\": 18, \"battery\_level\": 6, \"c02\_level\": 1413, \"timestamp\" :1475600498 }"} | |
| {"id":2,"device":"{\"device\_id\": 2, \"device\_type\": \"sensor-ipad\", \"ip\": \"88.36.5.1\", \"cca3\": \"ITA\", \"cn\": \"Italy\", \"temp\": 18, \"signal\": 25, \"battery\_level\": 5, \"c02\_level\": 1372, \"timestamp\" :1475600500 }"} | |
| {"id":3,"device":"{\"device\_id\": 3, \"device\_type\": \"sensor-inest\", \"ip\": \"66.39.173.154\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 47, \"signal\": 12, \"battery\_level\": 1, \"c02\_level\": 1447, \"timestamp\" :1475600502 }"} | |
| {"id":4,"device":"{\"device\_id\": 4, \"device\_type\": \"sensor-ipad\", \"ip\": \"203.82.41.9\", \"cca3\": \"PHL\", \"cn\": \"Philippines\", \"temp\": 29, \"signal\": 11, \"battery\_level\": 0, \"c02\_level\": 983, \"timestamp\" :1475600504 }"} | |
| {"id":5,"device":"{\"device\_id\": 5, \"device\_type\": \"sensor-istick\", \"ip\": \"204.116.105.67\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 50, \"signal\": 16, \"battery\_level\": 8, \"c02\_level\": 1574, \"timestamp\" :1475600506 }"} | |
| {"id":6,"device":"{\"device\_id\": 6, \"device\_type\": \"sensor-ipad\", \"ip\": \"220.173.179.1\", \"cca3\": \"CHN\", \"cn\": \"China\", \"temp\": 21, \"signal\": 18, \"battery\_level\": 9, \"c02\_level\": 1249, \"timestamp\" :1475600508 }"} | |
| {"id":7,"device":"{\"device\_id\": 7, \"device\_type\": \"sensor-ipad\", \"ip\": \"118.23.68.227\", \"cca3\": \"JPN\", \"cn\": \"Japan\", \"temp\": 27, \"signal\": 15, \"battery\_level\": 0, \"c02\_level\": 1531, \"timestamp\" :1475600512 }"} | |
| {"id":8,"device":" {\"device\_id\": 8, \"device\_type\": \"sensor-inest\", \"ip\": \"208.109.163.218\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 40, \"signal\": 16, \"battery\_level\": 9, \"c02\_level\": 1208, \"timestamp\" :1475600514 }"} | |
| {"id":9,"device":"{\"device\_id\": 9, \"device\_type\": \"sensor-ipad\", \"ip\": \"88.213.191.34\", \"cca3\": \"ITA\", \"cn\": \"Italy\", \"temp\": 19, \"signal\": 11, \"battery\_level\": 0, \"c02\_level\": 1171, \"timestamp\" :1475600516 }"} | |
| {"id":10,"device":"{\"device\_id\": 10, \"device\_type\": \"sensor-igauge\", \"ip\": \"68.28.91.22\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 32, \"signal\": 26, \"battery\_level\": 7, \"c02\_level\": 886, \"timestamp\" :1475600518 }"} | |
| {"id":11,"device":"{\"device\_id\": 11, \"device\_type\": \"sensor-ipad\", \"ip\": \"59.144.114.250\", \"cca3\": \"IND\", \"cn\": \"India\", \"temp\": 46, \"signal\": 25, \"battery\_level\": 4, \"c02\_level\": 863, \"timestamp\" :1475600520 }"} | |
| {"id":12,"device":"{\"device\_id\": 12, \"device\_type\": \"sensor-igauge\", \"ip\": \"193.156.90.200\", \"cca3\": \"NOR\", \"cn\": \"Norway\", \"temp\": 18, \"signal\": 26, \"battery\_level\": 8, \"c02\_level\": 1220, \"timestamp\" :1475600522 }"} | |
| {"id":13,"device":"{\"device\_id\": 13, \"device\_type\": \"sensor-ipad\", \"ip\": \"67.185.72.1\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 34, \"signal\": 20, \"battery\_level\": 8, \"c02\_level\": 1504, \"timestamp\" :1475600524 }"} | |
| {"id":14,"device":"{\"device\_id\": 14, \"device\_type\": \"sensor-inest\", \"ip\": \"68.85.85.106\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 39, \"signal\": 17, \"battery\_level\": 8, \"c02\_level\": 831, \"timestamp\" :1475600526 }"} | |
| {"id":15,"device":"{\"device\_id\": 15, \"device\_type\": \"sensor-ipad\", \"ip\": \"161.188.212.254\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 27, \"signal\": 26, \"battery\_level\": 5, \"c02\_level\": 1378, \"timestamp\" :1475600528 }"} | |
| {"id":16,"device":"{\"device\_id\": 16, \"device\_type\": \"sensor-igauge\", \"ip\": \"221.3.128.242\", \"cca3\": \"CHN\", \"cn\": \"China\", \"temp\": 10, \"signal\": 24, \"battery\_level\": 6, \"c02\_level\": 1423, \"timestamp\" :1475600530 }"} | |
| {"id":17,"device":"{\"device\_id\": 17, \"device\_type\": \"sensor-ipad\", \"ip\": \"64.124.180.215\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 38, \"signal\": 17, \"battery\_level\": 9, \"c02\_level\": 1304, \"timestamp\" :1475600532 }"} | |
| {"id":18,"device":"{\"device\_id\": 18, \"device\_type\": \"sensor-igauge\", \"ip\": \"66.153.162.66\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 26, \"signal\": 10, \"battery\_level\": 0, \"c02\_level\": 902, \"timestamp\" :1475600534 }"} | |
| {"id":19,"device":"{\"device\_id\": 19, \"device\_type\": \"sensor-ipad\", \"ip\": \"193.200.142.254\", \"cca3\": \"AUT\", \"cn\": \"Austria\", \"temp\": 32, \"signal\": 27, \"battery\_level\": 5, \"c02\_level\": 1282, \"timestamp\" :1475600536 }"} | |
| **devices** |

Assuming you have a Kafka cluster running at specified port and the respective topic, let's write to Kafka topic.

// stringJsonDF.write

// .format("kafka")

// .option("kafka.bootstrap.servers", "your\_host\_name:9092")

// .option("topic", "iot-devices")

// .save()

**How to use selectExpr()**

Another way to convert or encode a column into a JSON object as string is to use the *selectExpr()* utility function. For instance, I can convert the "device" column of our DataFrame from above into a JSON String

**val** stringsDF = eventsDS.selectExpr("CAST(id AS INT)", "CAST(device AS STRING)")

stringsDF: org.apache.spark.sql.DataFrame = [id: int, device: string]

stringsDF.printSchema

root |-- id: integer (nullable = false) |-- device: string (nullable = true)

display(stringsDF)

|  |  |
| --- | --- |
| 0 | {"device\_id": 0, "device\_type": "sensor-ipad", "ip": "68.161.225.1", "cca3": "USA", "cn": "United States", "temp": 25, "signal": 23, "battery\_level": 8, "c02\_level": 917, "timestamp" :1475600496 } |
| 1 | {"device\_id": 1, "device\_type": "sensor-igauge", "ip": "213.161.254.1", "cca3": "NOR", "cn": "Norway", "temp": 30, "signal": 18, "battery\_level": 6, "c02\_level": 1413, "timestamp" :1475600498 } |
| 2 | {"device\_id": 2, "device\_type": "sensor-ipad", "ip": "88.36.5.1", "cca3": "ITA", "cn": "Italy", "temp": 18, "signal": 25, "battery\_level": 5, "c02\_level": 1372, "timestamp" :1475600500 } |
| 3 | {"device\_id": 3, "device\_type": "sensor-inest", "ip": "66.39.173.154", "cca3": "USA", "cn": "United States", "temp": 47, "signal": 12, "battery\_level": 1, "c02\_level": 1447, "timestamp" :1475600502 } |
| 4 | {"device\_id": 4, "device\_type": "sensor-ipad", "ip": "203.82.41.9", "cca3": "PHL", "cn": "Philippines", "temp": 29, "signal": 11, "battery\_level": 0, "c02\_level": 983, "timestamp" :1475600504 } |
| 5 | {"device\_id": 5, "device\_type": "sensor-istick", "ip": "204.116.105.67", "cca3": "USA", "cn": "United States", "temp": 50, "signal": 16, "battery\_level": 8, "c02\_level": 1574, "timestamp" :1475600506 } |
| 6 | {"device\_id": 6, "device\_type": "sensor-ipad", "ip": "220.173.179.1", "cca3": "CHN", "cn": "China", "temp": 21, "signal": 18, "battery\_level": 9, "c02\_level": 1249, "timestamp" :1475600508 } |
| 7 | {"device\_id": 7, "device\_type": "sensor-ipad", "ip": "118.23.68.227", "cca3": "JPN", "cn": "Japan", "temp": 27, "signal": 15, "battery\_level": 0, "c02\_level": 1531, "timestamp" :1475600512 } |
| 8 | {"device\_id": 8, "device\_type": "sensor-inest", "ip": "208.109.163.218", "cca3": "USA", "cn": "United States", "temp": 40, "signal": 16, "battery\_level": 9, "c02\_level": 1208, "timestamp" :1475600514 } |
| 9 | {"device\_id": 9, "device\_type": "sensor-ipad", "ip": "88.213.191.34", "cca3": "ITA", "cn": "Italy", "temp": 19, "signal": 11, "battery\_level": 0, "c02\_level": 1171, "timestamp" :1475600516 } |
| 10 | {"device\_id": 10, "device\_type": "sensor-igauge", "ip": "68.28.91.22", "cca3": "USA", "cn": "United States", "temp": 32, "signal": 26, "battery\_level": 7, "c02\_level": 886, "timestamp" :1475600518 } |
| 11 | {"device\_id": 11, "device\_type": "sensor-ipad", "ip": "59.144.114.250", "cca3": "IND", "cn": "India", "temp": 46, "signal": 25, "battery\_level": 4, "c02\_level": 863, "timestamp" :1475600520 } |
| 12 | {"device\_id": 12, "device\_type": "sensor-igauge", "ip": "193.156.90.200", "cca3": "NOR", "cn": "Norway", "temp": 18, "signal": 26, "battery\_level": 8, "c02\_level": 1220, "timestamp" :1475600522 } |
| 13 | {"device\_id": 13, "device\_type": "sensor-ipad", "ip": "67.185.72.1", "cca3": "USA", "cn": "United States", "temp": 34, "signal": 20, "battery\_level": 8, "c02\_level": 1504, "timestamp" :1475600524 } |
| 14 | {"device\_id": 14, "device\_type": "sensor-inest", "ip": "68.85.85.106", "cca3": "USA", "cn": "United States", "temp": 39, "signal": 17, "battery\_level": 8, "c02\_level": 831, "timestamp" :1475600526 } |
| 15 | {"device\_id": 15, "device\_type": "sensor-ipad", "ip": "161.188.212.254", "cca3": "USA", "cn": "United States", "temp": 27, "signal": 26, "battery\_level": 5, "c02\_level": 1378, "timestamp" :1475600528 } |
| 16 | {"device\_id": 16, "device\_type": "sensor-igauge", "ip": "221.3.128.242", "cca3": "CHN", "cn": "China", "temp": 10, "signal": 24, "battery\_level": 6, "c02\_level": 1423, "timestamp" :1475600530 } |
| 17 | {"device\_id": 17, "device\_type": "sensor-ipad", "ip": "64.124.180.215", "cca3": "USA", "cn": "United States", "temp": 38, "signal": 17, "battery\_level": 9, "c02\_level": 1304, "timestamp" :1475600532 } |
| 18 | {"device\_id": 18, "device\_type": "sensor-igauge", "ip": "66.153.162.66", "cca3": "USA", "cn": "United States", "temp": 26, "signal": 10, "battery\_level": 0, "c02\_level": 902, "timestamp" :1475600534 } |
| 19 | {"device\_id": 19, "device\_type": "sensor-ipad", "ip": "193.200.142.254", "cca3": "AUT", "cn": "Austria", "temp": 32, "signal": 27, "battery\_level": 5, "c02\_level": 1282, "timestamp" :1475600536 } |
| **id** | **device** |

Another use of selectExpr() is its ability, as the function name suggests, take expressions as arguments and convert them into respective columns. For instance, say I want to express c02 levels and temperature ratios.

display(devicesDF.selectExpr("c02\_level", "round(c02\_level/temp) as ratio\_c02\_temperature").orderBy($"ratio\_c02\_temperature" desc))

|  | |  | |
| --- | --- | --- | --- |
| 1372 | | 76 | |
| 1220 | | 68 | |
| 1249 | | 59 | |
| 1378 | | 51 | |
| 1413 | | 47 | |
| 1504 | | 44 | |
| 1282 | | 40 | |
| 917 | | 37 | |
| 1304 | | 34 | |
| 1574 | | 31 | |
| 1208 | | 30 | |
| 886 | | 28 | |
| 831 | | 21 | |
| 863 | | 19 | |
| **c02\_level** | **ratio\_c02\_temperature** | |

The above query could as easily be expressed in Spark SQL as in DataFrame API. The power of *selectExpr()* lies in dealing with or working with numerical values. Let's try to create a tempoary view and express the same query, except this time we use SQL.

devicesDF.createOrReplaceTempView("devicesDFT")

Notice the output from **cmd 42** is not different from **cmd** 38. Both undergo the same Spark SQL engine's Catalyst and generate equivalent underlying compact code.

%sql **select** c02\_level,

round(c02\_level/**temp**) **as** ratio\_c02\_temperature

**from** devicesDFT

**order** **by** ratio\_c02\_temperature **desc**

|  | |  | |
| --- | --- | --- | --- |
| 1372 | | 76 | |
| 1220 | | 68 | |
| 1249 | | 59 | |
| 1378 | | 51 | |
| 1413 | | 47 | |
| 1504 | | 44 | |
| 1282 | | 40 | |
| 917 | | 37 | |
| 1304 | | 34 | |
| 1574 | | 31 | |
| 1208 | | 30 | |
| 886 | | 28 | |
| 831 | | 21 | |
| 863 | | 19 | |
| **c02\_level** | **ratio\_c02\_temperature** | |

**To verify that all your string conversions are preserved in the above DataFrame stringJsonDF, let's save to blob storage as Parquet.**

stringJsonDF

.write

.mode("overwrite")

.format("parquet")

.save("/tmp/iot")

Check if all files were written.

%fs ls /tmp/iot

|  | |  | |  | |
| --- | --- | --- | --- | --- | --- |
| dbfs:/tmp/iot/\_SUCCESS | | \_SUCCESS | | 0 | |
| dbfs:/tmp/iot/\_committed\_4820070580060822340 | | \_committed\_4820070580060822340 | | 840 | |
| dbfs:/tmp/iot/\_committed\_7472410303440732077 | | \_committed\_7472410303440732077 | | 1666 | |
| dbfs:/tmp/iot/\_committed\_vacuum5255432591358387481 | | \_committed\_vacuum5255432591358387481 | | 96 | |
| dbfs:/tmp/iot/\_started\_7472410303440732077 | | \_started\_7472410303440732077 | | 0 | |
| dbfs:/tmp/iot/part-00000-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78688-1-c000.snappy.parquet | | part-00000-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78688-1-c000.snappy.parquet | | 1701 | |
| dbfs:/tmp/iot/part-00001-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78689-1-c000.snappy.parquet | | part-00001-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78689-1-c000.snappy.parquet | | 1764 | |
| dbfs:/tmp/iot/part-00002-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78690-1-c000.snappy.parquet | | part-00002-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78690-1-c000.snappy.parquet | | 1714 | |
| dbfs:/tmp/iot/part-00003-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78691-1-c000.snappy.parquet | | part-00003-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78691-1-c000.snappy.parquet | | 1772 | |
| dbfs:/tmp/iot/part-00004-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78692-1-c000.snappy.parquet | | part-00004-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78692-1-c000.snappy.parquet | | 1709 | |
| dbfs:/tmp/iot/part-00005-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78693-1-c000.snappy.parquet | | part-00005-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78693-1-c000.snappy.parquet | | 1774 | |
| dbfs:/tmp/iot/part-00006-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78694-1-c000.snappy.parquet | | part-00006-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78694-1-c000.snappy.parquet | | 1720 | |
| dbfs:/tmp/iot/part-00007-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78695-1-c000.snappy.parquet | | part-00007-tid-7472410303440732077-1b5f39de-6616-4c21-b825-003bb8a2179f-78695-1-c000.snappy.parquet | | 1791 | |
| **path** | **name** | | **size** | |

Now let's verify what was saved—devices as each indivdual strings encoded from above—are actual strings.

**val** parquetDF = spark.read.parquet("/tmp/iot")

parquetDF: org.apache.spark.sql.DataFrame = [devices: string]

Let's check the schema to ensure what was written is not different from what is read, namely the JSON string.

parquetDF.printSchema

root |-- devices: string (nullable = true)

display(parquetDF)

|  | |
| --- | --- |
| {"id":12,"device":"{\"device\_id\": 12, \"device\_type\": \"sensor-igauge\", \"ip\": \"193.156.90.200\", \"cca3\": \"NOR\", \"cn\": \"Norway\", \"temp\": 18, \"signal\": 26, \"battery\_level\": 8, \"c02\_level\": 1220, \"timestamp\" :1475600522 }"} | |
| {"id":13,"device":"{\"device\_id\": 13, \"device\_type\": \"sensor-ipad\", \"ip\": \"67.185.72.1\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 34, \"signal\": 20, \"battery\_level\": 8, \"c02\_level\": 1504, \"timestamp\" :1475600524 }"} | |
| {"id":14,"device":"{\"device\_id\": 14, \"device\_type\": \"sensor-inest\", \"ip\": \"68.85.85.106\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 39, \"signal\": 17, \"battery\_level\": 8, \"c02\_level\": 831, \"timestamp\" :1475600526 }"} | |
| {"id":2,"device":"{\"device\_id\": 2, \"device\_type\": \"sensor-ipad\", \"ip\": \"88.36.5.1\", \"cca3\": \"ITA\", \"cn\": \"Italy\", \"temp\": 18, \"signal\": 25, \"battery\_level\": 5, \"c02\_level\": 1372, \"timestamp\" :1475600500 }"} | |
| {"id":3,"device":"{\"device\_id\": 3, \"device\_type\": \"sensor-inest\", \"ip\": \"66.39.173.154\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 47, \"signal\": 12, \"battery\_level\": 1, \"c02\_level\": 1447, \"timestamp\" :1475600502 }"} | |
| {"id":4,"device":"{\"device\_id\": 4, \"device\_type\": \"sensor-ipad\", \"ip\": \"203.82.41.9\", \"cca3\": \"PHL\", \"cn\": \"Philippines\", \"temp\": 29, \"signal\": 11, \"battery\_level\": 0, \"c02\_level\": 983, \"timestamp\" :1475600504 }"} | |
| {"id":15,"device":"{\"device\_id\": 15, \"device\_type\": \"sensor-ipad\", \"ip\": \"161.188.212.254\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 27, \"signal\": 26, \"battery\_level\": 5, \"c02\_level\": 1378, \"timestamp\" :1475600528 }"} | |
| {"id":16,"device":"{\"device\_id\": 16, \"device\_type\": \"sensor-igauge\", \"ip\": \"221.3.128.242\", \"cca3\": \"CHN\", \"cn\": \"China\", \"temp\": 10, \"signal\": 24, \"battery\_level\": 6, \"c02\_level\": 1423, \"timestamp\" :1475600530 }"} | |
| {"id":0,"device":"{\"device\_id\": 0, \"device\_type\": \"sensor-ipad\", \"ip\": \"68.161.225.1\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 25, \"signal\": 23, \"battery\_level\": 8, \"c02\_level\": 917, \"timestamp\" :1475600496 }"} | |
| {"id":1,"device":"{\"device\_id\": 1, \"device\_type\": \"sensor-igauge\", \"ip\": \"213.161.254.1\", \"cca3\": \"NOR\", \"cn\": \"Norway\", \"temp\": 30, \"signal\": 18, \"battery\_level\": 6, \"c02\_level\": 1413, \"timestamp\" :1475600498 }"} | |
| {"id":17,"device":"{\"device\_id\": 17, \"device\_type\": \"sensor-ipad\", \"ip\": \"64.124.180.215\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 38, \"signal\": 17, \"battery\_level\": 9, \"c02\_level\": 1304, \"timestamp\" :1475600532 }"} | |
| {"id":18,"device":"{\"device\_id\": 18, \"device\_type\": \"sensor-igauge\", \"ip\": \"66.153.162.66\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 26, \"signal\": 10, \"battery\_level\": 0, \"c02\_level\": 902, \"timestamp\" :1475600534 }"} | |
| {"id":19,"device":"{\"device\_id\": 19, \"device\_type\": \"sensor-ipad\", \"ip\": \"193.200.142.254\", \"cca3\": \"AUT\", \"cn\": \"Austria\", \"temp\": 32, \"signal\": 27, \"battery\_level\": 5, \"c02\_level\": 1282, \"timestamp\" :1475600536 }"} | |
| {"id":7,"device":"{\"device\_id\": 7, \"device\_type\": \"sensor-ipad\", \"ip\": \"118.23.68.227\", \"cca3\": \"JPN\", \"cn\": \"Japan\", \"temp\": 27, \"signal\": 15, \"battery\_level\": 0, \"c02\_level\": 1531, \"timestamp\" :1475600512 }"} | |
| {"id":8,"device":" {\"device\_id\": 8, \"device\_type\": \"sensor-inest\", \"ip\": \"208.109.163.218\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 40, \"signal\": 16, \"battery\_level\": 9, \"c02\_level\": 1208, \"timestamp\" :1475600514 }"} | |
| {"id":9,"device":"{\"device\_id\": 9, \"device\_type\": \"sensor-ipad\", \"ip\": \"88.213.191.34\", \"cca3\": \"ITA\", \"cn\": \"Italy\", \"temp\": 19, \"signal\": 11, \"battery\_level\": 0, \"c02\_level\": 1171, \"timestamp\" :1475600516 }"} | |
| {"id":5,"device":"{\"device\_id\": 5, \"device\_type\": \"sensor-istick\", \"ip\": \"204.116.105.67\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 50, \"signal\": 16, \"battery\_level\": 8, \"c02\_level\": 1574, \"timestamp\" :1475600506 }"} | |
| {"id":6,"device":"{\"device\_id\": 6, \"device\_type\": \"sensor-ipad\", \"ip\": \"220.173.179.1\", \"cca3\": \"CHN\", \"cn\": \"China\", \"temp\": 21, \"signal\": 18, \"battery\_level\": 9, \"c02\_level\": 1249, \"timestamp\" :1475600508 }"} | |
| {"id":10,"device":"{\"device\_id\": 10, \"device\_type\": \"sensor-igauge\", \"ip\": \"68.28.91.22\", \"cca3\": \"USA\", \"cn\": \"United States\", \"temp\": 32, \"signal\": 26, \"battery\_level\": 7, \"c02\_level\": 886, \"timestamp\" :1475600518 }"} | |
| {"id":11,"device":"{\"device\_id\": 11, \"device\_type\": \"sensor-ipad\", \"ip\": \"59.144.114.250\", \"cca3\": \"IND\", \"cn\": \"India\", \"temp\": 46, \"signal\": 25, \"battery\_level\": 4, \"c02\_level\": 863, \"timestamp\" :1475600520 }"} | |
| **devices** |

So far this tutorial has explored ways to use get\_json\_object(), from\_json(), to\_json(), selectExpr(), and explode() helper functions handling less complex JSON objects.

Let's turn focus to a more nested structures and examine how these same APIs as applied to a complex JSON as simple one.

**Nested Structures**

It's not unreasonable to assume that your JSON nested structures may have Maps as well as nested JSON. For illustration, let's use a single string comprised of complex and nested data types, including a Map. In a real life scenario, this could be a reading from a device event, with dangerous levels of C02 emissions or high temperature readings, that needs Network Operation Center (NOC) notification for some immediate action.

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

**val** schema = **new** StructType()

.add("dc\_id", StringType) // data center where data was posted to Kafka cluster

.add("source", // info about the source of alarm

MapType( // define this as a Map(Key->value)

StringType,

**new** StructType()

.add("description", StringType)

.add("ip", StringType)

.add("id", LongType)

.add("temp", LongType)

.add("c02\_level", LongType)

.add("geo",

**new** StructType()

.add("lat", DoubleType)

.add("long", DoubleType)

)

)

)

import org.apache.spark.sql.types.\_ schema: org.apache.spark.sql.types.StructType = StructType(StructField(dc\_id,StringType,true), StructField(source,MapType(StringType,StructType(StructField(description,StringType,true), StructField(ip,StringType,true), StructField(id,LongType,true), StructField(temp,LongType,true), StructField(c02\_level,LongType,true), StructField(geo,StructType(StructField(lat,DoubleType,true), StructField(long,DoubleType,true)),true)),true),true))

Let's create a single complex JSON with complex data types.

//create a single entry with id and its complex and nested data types

**val** dataDS = Seq("""

{

"dc\_id": "dc-101",

"source": {

"sensor-igauge": {

"id": 10,

"ip": "68.28.91.22",

"description": "Sensor attached to the container ceilings",

"temp":35,

"c02\_level": 1475,

"geo": {"lat":38.00, "long":97.00}

},

"sensor-ipad": {

"id": 13,

"ip": "67.185.72.1",

"description": "Sensor ipad attached to carbon cylinders",

"temp": 34,

"c02\_level": 1370,

"geo": {"lat":47.41, "long":-122.00}

},

"sensor-inest": {

"id": 8,

"ip": "208.109.163.218",

"description": "Sensor attached to the factory ceilings",

"temp": 40,

"c02\_level": 1346,

"geo": {"lat":33.61, "long":-111.89}

},

"sensor-istick": {

"id": 5,

"ip": "204.116.105.67",

"description": "Sensor embedded in exhaust pipes in the ceilings",

"temp": 40,

"c02\_level": 1574,

"geo": {"lat":35.93, "long":-85.46}

}

}

}""").toDS()

// should only be one item

dataDS.count()

dataDS: org.apache.spark.sql.Dataset[String] = [value: string] res25: Long = 1

display(dataDS)

|  | |
| --- | --- |
| { "dc\_id": "dc-101", "source": { "sensor-igauge": { "id": 10, "ip": "68.28.91.22", "description": "Sensor attached to the container ceilings", "temp":35, "c02\_level": 1475, "geo": {"lat":38.00, "long":97.00} }, "sensor-ipad": { "id": 13, "ip": "67.185.72.1", "description": "Sensor ipad attached to carbon cylinders", "temp": 34, "c02\_level": 1370, "geo": {"lat":47.41, "long":-122.00} }, "sensor-inest": { "id": 8, "ip": "208.109.163.218", "description": "Sensor attached to the factory ceilings", "temp": 40, "c02\_level": 1346, "geo": {"lat":33.61, "long":-111.89} }, "sensor-istick": { "id": 5, "ip": "204.116.105.67", "description": "Sensor embedded in exhaust pipes in the ceilings", "temp": 40, "c02\_level": 1574, "geo": {"lat":35.93, "long":-85.46} } } } | |
| **value** |

Let's process it. Note that we have nested structure geo.

**val** df = spark // spark session

.read // get DataFrameReader

.schema(schema) // use the defined schema above and read format as JSON

.json(dataDS.rdd) // RDD[String]

command-3901575101282734:4: warning: method json in class DataFrameReader is deprecated (since 2.2.0): Use json(Dataset[String]) instead. .json(dataDS.rdd) // RDD[String] ^ df: org.apache.spark.sql.DataFrame = [dc\_id: string, source: map<string,struct<description:string,ip:string,id:bigint,temp:bigint,c02\_level:bigint,geo:struct<lat:double,long:double>>>]

Let's examine its nested and complex schema.

df.printSchema

root |-- dc\_id: string (nullable = true) |-- source: map (nullable = true) | |-- key: string | |-- value: struct (valueContainsNull = true) | | |-- description: string (nullable = true) | | |-- ip: string (nullable = true) | | |-- id: long (nullable = true) | | |-- temp: long (nullable = true) | | |-- c02\_level: long (nullable = true) | | |-- geo: struct (nullable = true) | | | |-- lat: double (nullable = true) | | | |-- long: double (nullable = true)

display(df)

|  |  |
| --- | --- |
| dc-101 | {"sensor-igauge":{"description":"Sensor attached to the container ceilings","ip":"68.28.91.22","id":10,"temp":35,"c02\_level":1475,"geo":{"lat":38,"long":97}},"sensor-ipad":{"description":"Sensor ipad attached to carbon cylinders","ip":"67.185.72.1","id":13,"temp":34,"c02\_level":1370,"geo":{"lat":47.41,"long":-122}},"sensor-inest":{"description":"Sensor attached to the factory ceilings","ip":"208.109.163.218","id":8,"temp":40,"c02\_level":1346,"geo":{"lat":33.61,"long":-111.89}},"sensor-istick":{"description":"Sensor embedded in exhaust pipes in the ceilings","ip":"204.116.105.67","id":5,"temp":40,"c02\_level":1574,"geo":{"lat":35.93,"long":-85.46}}} |
| **dc\_id** | **source** |

**How to use explode()**

The explode() function is used to show how to extract nested structures. Plus, it sheds more light when we see how it works alongside to\_json() and from\_json() functions, when extracting attributes and values from complex JSON structures. So on occasion, you will want to use explode(), alongside to\_json() and from\_json() functions. And here's one case where we do.

The explode() function creates a new row for each element in the given map column. In this case, the map column is source. Note that for each key-value in the map, you have a respective Row, in this case four.

// select from DataFrame with a single entry, and explode its column source, which is Map, with nested structure.

**val** explodedDF = df.select($"dc\_id", explode($"source"))

display(explodedDF)

|  |  |  | |
| --- | --- | --- | --- |
| dc-101 | sensor-igauge | {"description":"Sensor attached to the container ceilings","ip":"68.28.91.22","id":10,"temp":35,"c02\_level":1475,"geo":{"lat":38,"long":97}} | |
| dc-101 | sensor-ipad | {"description":"Sensor ipad attached to carbon cylinders","ip":"67.185.72.1","id":13,"temp":34,"c02\_level":1370,"geo":{"lat":47.41,"long":-122}} | |
| dc-101 | sensor-inest | {"description":"Sensor attached to the factory ceilings","ip":"208.109.163.218","id":8,"temp":40,"c02\_level":1346,"geo":{"lat":33.61,"long":-111.89}} | |
| dc-101 | sensor-istick | {"description":"Sensor embedded in exhaust pipes in the ceilings","ip":"204.116.105.67","id":5,"temp":40,"c02\_level":1574,"geo":{"lat":35.93,"long":-85.46}} | |
| **dc\_id** | | **key** | **value** |

When you look at the schema, notice that source now has been expanded.

explodedDF.printSchema

root |-- dc\_id: string (nullable = true) |-- key: string (nullable = false) |-- value: struct (nullable = true) | |-- description: string (nullable = true) | |-- ip: string (nullable = true) | |-- id: long (nullable = true) | |-- temp: long (nullable = true) | |-- c02\_level: long (nullable = true) | |-- geo: struct (nullable = true) | | |-- lat: double (nullable = true) | | |-- long: double (nullable = true)

A single string aggregated with complex data types, including a Map. This could be a recording that needs Network Operation Center (NOC) attention for action, since both the temperature and C02 levels are alarming.

Let's access the data from our exploded data using Map.

//case class to denote our desired Scala object

**case** **class** DeviceAlert(dcId: String, deviceType:String, ip:String, deviceId:Long, temp:Long, c02\_level: Long, lat: Double, lon: Double)

//access all values using getItem() method on value, by providing the "key," which is attribute in our JSON object.

**val** notifydevicesDS = explodedDF.select( $"dc\_id" as "dcId",

$"key" as "deviceType",

'value.getItem("ip") as 'ip,

'value.getItem("id") as 'deviceId,

'value.getItem("c02\_level") as 'c02\_level,

'value.getItem("temp") as 'temp,

'value.getItem("geo").getItem("lat") as 'lat, //note embedded level requires yet another level of fetching.

'value.getItem("geo").getItem("long") as 'lon)

.as[DeviceAlert] // return as a Dataset

defined class DeviceAlert notifydevicesDS: org.apache.spark.sql.Dataset[DeviceAlert] = [dcId: string, deviceType: string ... 6 more fields]

notifydevicesDS.printSchema

root |-- dcId: string (nullable = true) |-- deviceType: string (nullable = false) |-- ip: string (nullable = true) |-- deviceId: long (nullable = true) |-- c02\_level: long (nullable = true) |-- temp: long (nullable = true) |-- lat: double (nullable = true) |-- lon: double (nullable = true)

display(notifydevicesDS)

sensor-igaugesensor-ipadsensor-inestsensor-istick100%0%0%0%0%100%0%0%0%0%100%0%0%0%0%100%sensor-igaugesensor-ipadsensor-inestsensor-istick1475, 351370, 341346, 401574, 40

Suppose as part of our ETL, you have a need to notify or send alerts based on certain alarming conditions. One way is to write a user functtion at iterates over your filtered dataset and sends individual notifications. In other cases you can send the message to Kafka topic as an additional option.

Once you have exploded your nested JSON into a simple case class, we can send alerts to a NOC for action. On way to this is using a foreach() DataFrame method. But to do that we need a high-level function; given a case class it can extract its alarming attributes dispurse an alert. Although this simple example writes to stdout, in a real scenario, you will want to send alerts via SNMP or HTTP POST or some API to a PagerAlert.

**Function for alert notifications**

// define a Scala Notification Object

**object** DeviceNOCAlerts {

**def** sendTwilio(message: String): Unit = {

//TODO: fill as necessary

**println**("Twilio:" + message)

}

**def** sendSNMP(message: String): Unit = {

//TODO: fill as necessary

**println**("SNMP:" + message)

}

**def** sendKafka(message: String): Unit = {

//TODO: fill as necessary

**println**("KAFKA:" + message)

}

}

**def** logAlerts(log: java.io.PrintStream = Console.out, deviceAlert: DeviceAlert, alert: String, notify: String ="twilio"): Unit = {

**val** message = "[\*\*\*ALERT\*\*\*: %s; data\_center: %s, device\_name: %s, temperature: %d; device\_id: %d ; ip: %s ; c02: %d]" format(alert, deviceAlert.dcId, deviceAlert.deviceType,deviceAlert.temp, deviceAlert.deviceId, deviceAlert.ip, deviceAlert.c02\_level)

//default log to Stderr/Stdout

log.**println**(message)

// use an appropriate notification method

**val** notifyFunc = notify **match** {

**case** "twilio" => DeviceNOCAlerts.sendTwilio **\_**

**case** "snmp" => DeviceNOCAlerts.sendSNMP **\_**

**case** "kafka" => DeviceNOCAlerts.sendKafka **\_**

}

//send the appropriate alert

notifyFunc(message)

}

defined object DeviceNOCAlerts logAlerts: (log: java.io.PrintStream, deviceAlert: DeviceAlert, alert: String, notify: String)Unit

**Iterate over alert devices and take action**

notifydevicesDS.foreach(d => logAlerts(Console.err, d, "ACTION NEED! HIGH TEPERATURE AND C02 LEVLES", "kafka"))

To view where the messages are logged, go to the **Clusters-->Spark-->Logs**.

**Send alerts as JSON to Apache Kafka topic**

What if you wanted to write these Devices' alerts to a Kafka topic on which a monitoring subscriber is awaiting for events to take action.

Here's a simple way to nofity listeners on your Kafka topic: "device\_alerts." To read further how to use Structured Streaming in detail with Apache Kafka, read our part 3 of the blog series on Structure Streaming.

**Note:** See yet another use of selectExpr() function we explored above.

// val deviceAlertQuery = notifydevicesDS

// .selectExpr("CAST(dcId AS STRING) AS key", "to\_json(struct(\*)) AS value")

// .writeStream

// .format("kafka")

// .option("kafka.bootstrap.servers", "host1:port1,host2:port2")

// .option("toipic", "device\_alerts")

// .start()

**Nest Device Data**

Let's look at another complex real-life data from Nest's readings. A Nest devices emits many JSON events to its collector. That collector could be at a nearby data center, a neighborhood-central data collector or an aggregator, or it could be a device installed at home, which on regular intervals sends device readings to a central data center connected via a secured internet connection. For illusration, I have curbed some of the attributes, but it still shows how complex data can be processed—and relevant attributes extracted.

Let's define its complicated schema first. At close observation, you will notice it's not dissimilar to the schema we defined above, except it has not one **map** but three **maps**: thermostats, cameras, and smoke alarms.

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

// a bit longish, nested, and convoluted JSON schema :)

**val** nestSchema2 = **new** StructType()

.add("devices",

**new** StructType()

.add("thermostats", MapType(StringType,

**new** StructType()

.add("device\_id", StringType)

.add("locale", StringType)

.add("software\_version", StringType)

.add("structure\_id", StringType)

.add("where\_name", StringType)

.add("last\_connection", StringType)

.add("is\_online", BooleanType)

.add("can\_cool", BooleanType)

.add("can\_heat", BooleanType)

.add("is\_using\_emergency\_heat", BooleanType)

.add("has\_fan", BooleanType)

.add("fan\_timer\_active", BooleanType)

.add("fan\_timer\_timeout", StringType)

.add("temperature\_scale", StringType)

.add("target\_temperature\_f", DoubleType)

.add("target\_temperature\_high\_f", DoubleType)

.add("target\_temperature\_low\_f", DoubleType)

.add("eco\_temperature\_high\_f", DoubleType)

.add("eco\_temperature\_low\_f", DoubleType)

.add("away\_temperature\_high\_f", DoubleType)

.add("away\_temperature\_low\_f", DoubleType)

.add("hvac\_mode", StringType)

.add("humidity", LongType)

.add("hvac\_state", StringType)

.add("is\_locked", StringType)

.add("locked\_temp\_min\_f", DoubleType)

.add("locked\_temp\_max\_f", DoubleType)))

.add("smoke\_co\_alarms", MapType(StringType,

**new** StructType()

.add("device\_id", StringType)

.add("locale", StringType)

.add("software\_version", StringType)

.add("structure\_id", StringType)

.add("where\_name", StringType)

.add("last\_connection", StringType)

.add("is\_online", BooleanType)

.add("battery\_health", StringType)

.add("co\_alarm\_state", StringType)

.add("smoke\_alarm\_state", StringType)

.add("is\_manual\_test\_active", BooleanType)

.add("last\_manual\_test\_time", StringType)

.add("ui\_color\_state", StringType)))

.add("cameras", MapType(StringType,

**new** StructType()

.add("device\_id", StringType)

.add("software\_version", StringType)

.add("structure\_id", StringType)

.add("where\_name", StringType)

.add("is\_online", BooleanType)

.add("is\_streaming", BooleanType)

.add("is\_audio\_input\_enabled", BooleanType)

.add("last\_is\_online\_change", StringType)

.add("is\_video\_history\_enabled", BooleanType)

.add("web\_url", StringType)

.add("app\_url", StringType)

.add("is\_public\_share\_enabled", BooleanType)

.add("activity\_zones",

**new** StructType()

.add("name", StringType)

.add("id", LongType))

.add("last\_event", StringType))))

import org.apache.spark.sql.types.\_ nestSchema2: org.apache.spark.sql.types.StructType = StructType(StructField(devices,StructType(StructField(thermostats,MapType(StringType,StructType(StructField(device\_id,StringType,true), StructField(locale,StringType,true), StructField(software\_version,StringType,true), StructField(structure\_id,StringType,true), StructField(where\_name,StringType,true), StructField(last\_connection,StringType,true), StructField(is\_online,BooleanType,true), StructField(can\_cool,BooleanType,true), StructField(can\_heat,BooleanType,true), StructField(is\_using\_emergency\_heat,BooleanType,true), StructField(has\_fan,BooleanType,true), StructField(fan\_timer\_active,BooleanType,true), StructField(fan\_timer\_timeout,StringType,true), StructField(temperature\_scale,StringType,true), StructField(target\_temperature\_f,DoubleType,true), StructField(target\_temperature\_high\_f,DoubleType,true), StructField(target\_temperature\_low\_f,DoubleType,true), StructField(eco\_temperature\_high\_f,DoubleType,true), StructField(eco\_temperature\_low\_f,DoubleType,true), StructField(away\_temperature\_high\_f,DoubleType,true), StructField(away\_temperature\_low\_f,DoubleType,true), StructField(hvac\_mode,StringType,true), StructField(humidity,LongType,true), StructField(hvac\_state,StringType,true), StructField(is\_locked,StringType,true), StructField(locked\_temp\_min\_f,DoubleType,true), StructField(locked\_temp\_max\_f,DoubleType,true)),true),true), StructField(smoke\_co\_alarms,MapType(StringType,StructType(StructField(device\_id,StringType,true), StructField(locale,StringType,true), StructField(software\_version,StringType,true), StructField(structure\_id,StringType,true), StructField(where\_name,StringType,true), StructField(last\_connection,StringType,true), StructField(is\_online,BooleanType,true), StructField(battery\_health,StringType,true), StructField(co\_alarm\_state,StringType,true), StructField(smoke\_alarm\_state,StringType,true), StructField(is\_manual\_test\_active,BooleanType,true), StructField(last\_manual\_test\_time,StringType,true), StructField(ui\_color\_state,StringType,true)),true),true), StructField(cameras,MapType(StringType,StructType(StructField(device\_id,StringType,true), StructField(software\_version,StringType,true), StructField(structure\_id,StringType,true), StructField(where\_name,StringType,true), StructField(is\_online,BooleanType,true), StructField(is\_streaming,BooleanType,true), StructField(is\_audio\_input\_enabled,BooleanType,true), StructField(last\_is\_online\_change,StringType,true), StructField(is\_video\_history\_enabled,BooleanType,true), StructField(web\_url,StringType,true), StructField(app\_url,StringType,true), StructField(is\_public\_share\_enabled,BooleanType,true), StructField(activity\_zones,StructType(StructField(name,StringType,true), StructField(id,LongType,true)),true), StructField(last\_event,StringType,true)),true),true)),true))

By creating a simple Dataset, you can then use all Dataset methods to do ETL, using utility functions from above: from\_json(), to\_json(), explode() and selectExpr().

**val** nestDataDS2 = Seq("""{

"devices": {

"thermostats": {

"peyiJNo0IldT2YlIVtYaGQ": {

"device\_id": "peyiJNo0IldT2YlIVtYaGQ",

"locale": "en-US",

"software\_version": "4.0",

"structure\_id": "VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw",

"where\_name": "Hallway Upstairs",

"last\_connection": "2016-10-31T23:59:59.000Z",

"is\_online": true,

"can\_cool": true,

"can\_heat": true,

"is\_using\_emergency\_heat": true,

"has\_fan": true,

"fan\_timer\_active": true,

"fan\_timer\_timeout": "2016-10-31T23:59:59.000Z",

"temperature\_scale": "F",

"target\_temperature\_f": 72,

"target\_temperature\_high\_f": 80,

"target\_temperature\_low\_f": 65,

"eco\_temperature\_high\_f": 80,

"eco\_temperature\_low\_f": 65,

"away\_temperature\_high\_f": 80,

"away\_temperature\_low\_f": 65,

"hvac\_mode": "heat",

"humidity": 40,

"hvac\_state": "heating",

"is\_locked": true,

"locked\_temp\_min\_f": 65,

"locked\_temp\_max\_f": 80

}

},

"smoke\_co\_alarms": {

"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs": {

"device\_id": "RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs",

"locale": "en-US",

"software\_version": "1.01",

"structure\_id": "VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw",

"where\_name": "Jane's Room",

"last\_connection": "2016-10-31T23:59:59.000Z",

"is\_online": true,

"battery\_health": "ok",

"co\_alarm\_state": "ok",

"smoke\_alarm\_state": "ok",

"is\_manual\_test\_active": true,

"last\_manual\_test\_time": "2016-10-31T23:59:59.000Z",

"ui\_color\_state": "gray"

}

},

"cameras": {

"awJo6rH0IldT2YlIVtYaGQ": {

"device\_id": "awJo6rH",

"software\_version": "4.0",

"structure\_id": "VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw",

"where\_name": "Foyer",

"is\_online": true,

"is\_streaming": true,

"is\_audio\_input\_enabled": true,

"last\_is\_online\_change": "2016-12-29T18:42:00.000Z",

"is\_video\_history\_enabled": true,

"web\_url": "https://home.nest.com/cameras/device\_id?auth=access\_token",

"app\_url": "nestmobile://cameras/device\_id?auth=access\_token",

"is\_public\_share\_enabled": true,

"activity\_zones": { "name": "Walkway", "id": 244083 },

"last\_event": "2016-10-31T23:59:59.000Z"

}

}

}

}""").toDS

nestDataDS2: org.apache.spark.sql.Dataset[String] = [value: string]

Let's create a DataFrame from this single nested structure and use all the above utility functions to process and extract relevant attributes

**val** nestDF2 = spark // spark session

.read // get DataFrameReader

.schema(nestSchema2) // use the defined schema above and read format as JSON

.json(nestDataDS2.rdd)

command-3901575101282763:4: warning: method json in class DataFrameReader is deprecated (since 2.2.0): Use json(Dataset[String]) instead. .json(nestDataDS2.rdd) ^ nestDF2: org.apache.spark.sql.DataFrame = [devices: struct<thermostats: map<string,struct<device\_id:string,locale:string,software\_version:string,structure\_id:string,where\_name:string,last\_connection:string,is\_online:boolean,can\_cool:boolean,can\_heat:boolean,is\_using\_emergency\_heat:boolean,has\_fan:boolean,fan\_timer\_active:boolean,fan\_timer\_timeout:string,temperature\_scale:string,target\_temperature\_f:double,target\_temperature\_high\_f:double,target\_temperature\_low\_f:double,eco\_temperature\_high\_f:double,eco\_temperature\_low\_f:double,away\_temperature\_high\_f:double,away\_temperature\_low\_f:double,hvac\_mode:string,humidity:bigint,hvac\_state:string,... 3 more fields>>, smoke\_co\_alarms: map<string,struct<device\_id:string,locale:string,software\_version:string,structure\_id:string,where\_name:string,last\_connection:string,is\_online:boolean,battery\_health:string,co\_alarm\_state:string,smoke\_alarm\_state:string,is\_manual\_test\_active:boolean,last\_manual\_test\_time:string,ui\_color\_state:string>> ... 1 more field>]

display(nestDF2)

|  | |
| --- | --- |
| {"thermostats":{"peyiJNo0IldT2YlIVtYaGQ":{"device\_id":"peyiJNo0IldT2YlIVtYaGQ","locale":"en-US","software\_version":"4.0","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Hallway Upstairs","last\_connection":"2016-10-31T23:59:59.000Z","is\_online":true,"can\_cool":true,"can\_heat":true,"is\_using\_emergency\_heat":true,"has\_fan":true,"fan\_timer\_active":true,"fan\_timer\_timeout":"2016-10-31T23:59:59.000Z","temperature\_scale":"F","target\_temperature\_f":72,"target\_temperature\_high\_f":80,"target\_temperature\_low\_f":65,"eco\_temperature\_high\_f":80,"eco\_temperature\_low\_f":65,"away\_temperature\_high\_f":80,"away\_temperature\_low\_f":65,"hvac\_mode":"heat","humidity":40,"hvac\_state":"heating","is\_locked":"true","locked\_temp\_min\_f":65,"locked\_temp\_max\_f":80}},"smoke\_co\_alarms":{"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs":{"device\_id":"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs","locale":"en-US","software\_version":"1.01","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Jane's Room","last\_connection":"2016-10-31T23:59:59.000Z","is\_online":true,"battery\_health":"ok","co\_alarm\_state":"ok","smoke\_alarm\_state":"ok","is\_manual\_test\_active":true,"last\_manual\_test\_time":"2016-10-31T23:59:59.000Z","ui\_color\_state":"gray"}},"cameras":{"awJo6rH0IldT2YlIVtYaGQ":{"device\_id":"awJo6rH","software\_version":"4.0","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Foyer","is\_online":true,"is\_streaming":true,"is\_audio\_input\_enabled":true,"last\_is\_online\_change":"2016-12-29T18:42:00.000Z","is\_video\_history\_enabled":true,"web\_url":"https://home.nest.com/cameras/device\_id?auth=access\_token","app\_url":"nestmobile://cameras/device\_id?auth=access\_token","is\_public\_share\_enabled":true,"activity\_zones":{"name":"Walkway","id":244083},"last\_event":"2016-10-31T23:59:59.000Z"}}} | |
| **devices** |

Converting the entire JSON object above into a JSON string as above.

**val** stringJsonDF = nestDF2.select(to\_json(struct($"\*"))).toDF("nestDevice")

stringJsonDF: org.apache.spark.sql.DataFrame = [nestDevice: string]

stringJsonDF.printSchema

root |-- nestDevice: string (nullable = true)

display(stringJsonDF)

|  | |
| --- | --- |
| {"devices":{"thermostats":{"peyiJNo0IldT2YlIVtYaGQ":{"device\_id":"peyiJNo0IldT2YlIVtYaGQ","locale":"en-US","software\_version":"4.0","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Hallway Upstairs","last\_connection":"2016-10-31T23:59:59.000Z","is\_online":true,"can\_cool":true,"can\_heat":true,"is\_using\_emergency\_heat":true,"has\_fan":true,"fan\_timer\_active":true,"fan\_timer\_timeout":"2016-10-31T23:59:59.000Z","temperature\_scale":"F","target\_temperature\_f":72.0,"target\_temperature\_high\_f":80.0,"target\_temperature\_low\_f":65.0,"eco\_temperature\_high\_f":80.0,"eco\_temperature\_low\_f":65.0,"away\_temperature\_high\_f":80.0,"away\_temperature\_low\_f":65.0,"hvac\_mode":"heat","humidity":40,"hvac\_state":"heating","is\_locked":"true","locked\_temp\_min\_f":65.0,"locked\_temp\_max\_f":80.0}},"smoke\_co\_alarms":{"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs":{"device\_id":"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs","locale":"en-US","software\_version":"1.01","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7... | |
| **nestDevice** |

Given the nested JSON object with three maps, you can get fetch individual map as a columnn, and then access attributes from it using explode().

**val** mapColumnsDF = nestDF2.select($"devices".getItem("smoke\_co\_alarms").alias ("smoke\_alarms"),

$"devices".getItem("cameras").alias ("cameras"),

$"devices".getItem("thermostats").alias ("thermostats"))

mapColumnsDF: org.apache.spark.sql.DataFrame = [smoke\_alarms: map<string,struct<device\_id:string,locale:string,software\_version:string,structure\_id:string,where\_name:string,last\_connection:string,is\_online:boolean,battery\_health:string,co\_alarm\_state:string,smoke\_alarm\_state:string,is\_manual\_test\_active:boolean,last\_manual\_test\_time:string,ui\_color\_state:string>>, cameras: map<string,struct<device\_id:string,software\_version:string,structure\_id:string,where\_name:string,is\_online:boolean,is\_streaming:boolean,is\_audio\_input\_enabled:boolean,last\_is\_online\_change:string,is\_video\_history\_enabled:boolean,web\_url:string,app\_url:string,is\_public\_share\_enabled:boolean,activity\_zones:struct<name:string,id:bigint>,last\_event:string>> ... 1 more field]

display(mapColumnsDF)

|  |  | | |  | |
| --- | --- | --- | --- | --- | --- |
| {"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs":{"device\_id":"RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs","locale":"en-US","software\_version":"1.01","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Jane's Room","last\_connection":"2016-10-31T23:59:59.000Z","is\_online":true,"battery\_health":"ok","co\_alarm\_state":"ok","smoke\_alarm\_state":"ok","is\_manual\_test\_active":true,"last\_manual\_test\_time":"2016-10-31T23:59:59.000Z","ui\_color\_state":"gray"}} | {"awJo6rH0IldT2YlIVtYaGQ":{"device\_id":"awJo6rH","software\_version":"4.0","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Foyer","is\_online":true,"is\_streaming":true,"is\_audio\_input\_enabled":true,"last\_is\_online\_change":"2016-12-29T18:42:00.000Z","is\_video\_history\_enabled":true,"web\_url":"https://home.nest.com/cameras/device\_id?auth=access\_token","app\_url":"nestmobile://cameras/device\_id?auth=access\_token","is\_public\_share\_enabled":true,"activity\_zones":{"name":"Walkway","id":244083},"last\_event":"2016-10-31T23:59:59.000Z"}} | | | {"peyiJNo0IldT2YlIVtYaGQ":{"device\_id":"peyiJNo0IldT2YlIVtYaGQ","locale":"en-US","software\_version":"4.0","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Hallway Upstairs","last\_connection":"2016-10-31T23:59:59.000Z","is\_online":true,"can\_cool":true,"can\_heat":true,"is\_using\_emergency\_heat":true,"has\_fan":true,"fan\_timer\_active":true,"fan\_timer\_timeout":"2016-10-31T23:59:59.000Z","temperature\_scale":"F","target\_temperature\_f":72,"target\_temperature\_high\_f":80,"target\_temperature\_low\_f":65,"eco\_temperature\_high\_f":80,"eco\_temperature\_low\_f":65,"away\_temperature\_high\_f":80,"away\_temperature\_low\_f":65,"hvac\_mode":"heat","humidity":40,"hvac\_state":"heating","is\_locked":"true","locked\_temp\_min\_f":65,"locked\_temp\_max\_f":80}} | |
| **smoke\_alarms** | | | **cameras** | **thermostats** | |

**val** explodedThermostatsDF = mapColumnsDF.select(explode($"thermostats"))

**val** explodedCamerasDF = mapColumnsDF.select(explode($"cameras"))

//or you could use the original nestDF2 and use the devices.X notation

**val** explodedSmokedAlarmsDF = nestDF2.select(explode($"devices.smoke\_co\_alarms"))

explodedThermostatsDF: org.apache.spark.sql.DataFrame = [key: string, value: struct<device\_id: string, locale: string ... 25 more fields>] explodedCamerasDF: org.apache.spark.sql.DataFrame = [key: string, value: struct<device\_id: string, software\_version: string ... 12 more fields>] explodedSmokedAlarmsDF: org.apache.spark.sql.DataFrame = [key: string, value: struct<device\_id: string, locale: string ... 11 more fields>]

display(explodedThermostatsDF)

|  |  | | |
| --- | --- | --- | --- |
| peyiJNo0IldT2YlIVtYaGQ | {"device\_id":"peyiJNo0IldT2YlIVtYaGQ","locale":"en-US","software\_version":"4.0","structure\_id":"VqFabWH21nwVyd4RWgJgNb292wa7hG\_dUwo2i2SG7j3-BOLY0BA4sw","where\_name":"Hallway Upstairs","last\_connection":"2016-10-31T23:59:59.000Z","is\_online":true,"can\_cool":true,"can\_heat":true,"is\_using\_emergency\_heat":true,"has\_fan":true,"fan\_timer\_active":true,"fan\_timer\_timeout":"2016-10-31T23:59:59.000Z","temperature\_scale":"F","target\_temperature\_f":72,"target\_temperature\_high\_f":80,"target\_temperature\_low\_f":65,"eco\_temperature\_high\_f":80,"eco\_temperature\_low\_f":65,"away\_temperature\_high\_f":80,"away\_temperature\_low\_f":65,"hvac\_mode":"heat","humidity":40,"hvac\_state":"heating","is\_locked":"true","locked\_temp\_min\_f":65,"locked\_temp\_max\_f":80} | | |
| **key** | | | **value** |

To extract specific individual fields from map, you can use the getItem() method.

**val** thermostateDF = explodedThermostatsDF.select($"value".getItem("device\_id").alias("device\_id"),

$"value".getItem("locale").alias("locale"),

$"value".getItem("where\_name").alias("location"),

$"value".getItem("last\_connection").alias("last\_connected"),

$"value".getItem("humidity").alias("humidity"),

$"value".getItem("target\_temperature\_f").alias("target\_temperature\_f"),

$"value".getItem("hvac\_mode").alias("mode"),

$"value".getItem("software\_version").alias("version"))

**val** cameraDF = explodedCamerasDF.select($"value".getItem("device\_id").alias("device\_id"),

$"value".getItem("where\_name").alias("location"),

$"value".getItem("software\_version").alias("version"),

$"value".getItem("activity\_zones").getItem("name").alias("name"),

$"value".getItem("activity\_zones").getItem("id").alias("id"))

**val** smokedAlarmsDF = explodedSmokedAlarmsDF.select($"value".getItem("device\_id").alias("device\_id"),

$"value".getItem("where\_name").alias("location"),

$"value".getItem("software\_version").alias("version"),

$"value".getItem("last\_connection").alias("last\_connected"),

$"value".getItem("battery\_health").alias("battery\_health"))

thermostateDF: org.apache.spark.sql.DataFrame = [device\_id: string, locale: string ... 6 more fields] cameraDF: org.apache.spark.sql.DataFrame = [device\_id: string, location: string ... 3 more fields] smokedAlarmsDF: org.apache.spark.sql.DataFrame = [device\_id: string, location: string ... 3 more fields]

display(thermostateDF)

|  | |  | |  | |  |  |  | |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| peyiJNo0IldT2YlIVtYaGQ | | en-US | | Hallway Upstairs | | 2016-10-31T23:59:59.000Z | 40 | 72 | | heat | 4.0 | |
| **device\_id** | | **locale** | **location** | **last\_connected** | | **humidity** | **target\_temperature\_f** | **mode** | | **version** | | |

display(cameraDF)

|  |  | |  | |  | |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| awJo6rH | Foyer | | 4.0 | | Walkway | | 244083 |
| **device\_id** | | **location** | | **version** | | **name** | **id** |

display(smokedAlarmsDF)

|  | |  | |  |  | |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RTMTKxsQTCxzVcsySOHPxKoF4OyCifrs | | Jane's Room | | 1.01 | 2016-10-31T23:59:59.000Z | | ok | |
| **device\_id** | | **location** | **version** | **last\_connected** | | | **battery\_health** | |

Let's join two DataFrames over column version.

**val** joineDFs = thermostateDF.join(cameraDF, "version")

joineDFs: org.apache.spark.sql.DataFrame = [version: string, device\_id: string ... 10 more fields]

display(joineDFs)

|  |  | |  |  |  |  |  | |  | |  |  | |  | |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4.0 | peyiJNo0IldT2YlIVtYaGQ | | en-US | Hallway Upstairs | 2016-10-31T23:59:59.000Z | 40 | 72 | | heat | | awJo6rH | Foyer | | Walkway | | 244083 |
| **version** | **device\_id** | **locale** | **location** | **last\_connected** | **humidity** | **target\_temperature\_f** | | **mode** | | **device\_id** | | | **location** | | **name** | **id** |

**Summary**