Junction Hackathon 2017 – Handle With Care

Datalake user manual for participating teams

Table of Contents

[1 Datalake Introduction 3](#_Toc496862569)

[2 Junction hackathon datalake access and usage 4](#_Toc496862570)

[2.1 Datalake Access URL s & Team Accounts information. 4](#_Toc496862571)

[2.2 Jupyter-Spark Notebooks 5](#_Toc496862572)

[2.2.1 Usage 5](#_Toc496862573)

[2.3 Livy – Spark-jobserver 7](#_Toc496862574)

[3 Accessing and loading data 9](#_Toc496862575)

[3.1 Shared Data Sets 9](#_Toc496862576)

[3.2 Uploading Own Data sets 9](#_Toc496862577)

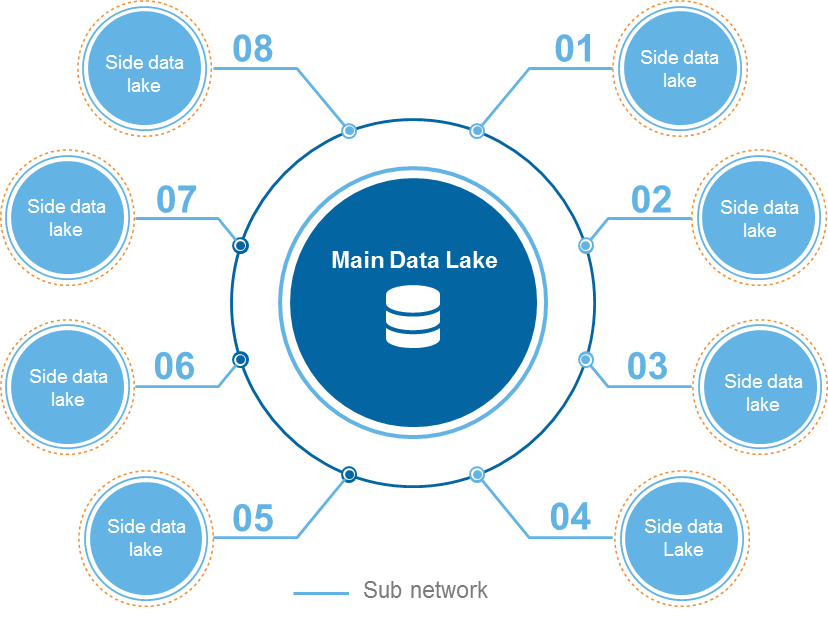
[3.3 Reading & Wrangling Data with Apache Spark 9](#_Toc496862578)

# Datalake Introduction

Junction Hackathon Datalake is based on HUS data lake solution developed by HUS (The Hospital District of Helsinki and Uusimaa). HUS datalake is a big data application developed for healthcare use. It is capable of storing and processing large amounts of different kind of data and offering a platform for data analytics. HUS datalake is a modular big data system and an applications platform that provides maximum security and at the same time being a flexible, extensible and scalable system.

Data storing in datalake is based on Microsoft’s Azure Data Lake Store. Azure Data Lake Store is HDFS compatible, hyperscale data storage that enables connections in various ways. The computing platform in the data lake system is Apache Hadoop that is provided by Microsoft Azure HDInsight. Supported data processing technologies are Apache Hive and Apache Spark.

Data lake architecture consists of a main datalake and an unlimited amount of sub datalakes (each Junction Hackathon teams gets an own sub data lake for their use). The main datalake includes all data stored to the system, and provides an environment for automated data processing. Isolated sub datalakes are created for individual users or groups and contain the data designated to that user or group. A user or group has access only to their designated sub data lake. A sub datalake is a secure work environment where the researcher can run tools and processes they want, or develop new ones by themselves. Users can also upload their own data sets to their own sub data lake.



Junction Hackathon datalake is a strongly simplified setup of the HUS datalake, including two small pre-planned data sets and a reduced data lake feature set. In Jucntion Hackathon, each hackathon team gets their own sub data lake instance (a mini cluster) for their use with the predefined data sets. The usage of Junction Hackathon datalake is described in following chapters.

Junction Hackathon teams get two separate data sets for their use:

1. synthetic health data and
2. sensor data combining well-being data and data from IoT sensors

# Junction hackathon datalake access and usage

Pre-requisites for datalake users:

* Prior knowledge of using Apache Spark with Python / Scala.
* Prior experience of working with Jupyter Notebooks

Optional, Good to know:

* Prior experience of using Livy Spark-jobserver to submit spark jobs to a remote spark cluster.
* Prior experience of using IDE(s) e.g., PyCharm for python, Intellij/ Eclipse for scala etc., for development with Apache Spark.

## Datalake Access URL s & Team Accounts information.

The participating teams receive follwing information for accesing and using the datalake capabilties.

* A single datalake username:password pair for accesing all datalake utilities.
* SSH Key pair and datalake head node hostname for ssh-login to the machines. The common username also acts as the ssh-user. However, ssh-login is only allowed using ssh-keys.
* Datalake management UIs and Apache Spark Notebooks accesible via provided username:password pair.
  + Datalake admin dashboard for managing datalake services.
  + Datalake YARN UI for checking the status of applications managed by the YARN cluster.
  + Datalake Jupyter-Spark Notebooks.
  + Spark History server to check the logs for the apache spark jobs.

Junction hackathon datalake provides two interfaces to interact with Apache Spark cluster datalake.

* Jupyter-Spark Notebooks
* Livy – Spark-jobserver

These interfaces enable users to utilize Apache Spark for accessing and analysing stored data.

## Jupyter-Spark Notebooks

Jupyter-Spark notebooks provide experimental environment for the data science teams to use Apache Spark (v 2.1.1) for data wrangling and advanced analytics. Each team in the hackathon will receive a URL, username and password for accessing the jupyter UI. Jupyter server for each team is powered by Datalake cluster with a Spark Master and 2 x Spark Worker Nodes. Each Datalake Spark cluster has a total of 32GB Memory and 18 Virtual Cores. Each team can create multiple notebooks. However, a single notebook is configured to consume 12GB Memory and 3 Vitual Cores. Teams need to be careful about the total memory usage while keeping multiple notebooks simultaneously. Jupyter server allows you to shutdown the notebooks.

It is a good practice to save and download your important notebooks to keep a local copy.

Within notebooks you can either use PySpark or Spark Scala APIs. While creating your notebooks you can choose as follows and shown in Figure 1.

* PySpark3 – For PySpark with Python 3 (Recommended)
* Spark – For Spark Scala 2.11
* PySpark - For PySpark with Python 2.7 (Available but not recommended)

### Usage

After login to the jupyter, you can either create a notebook by selecting from available options i.e., PySpark3, Spark, PySpark or upload already cretaed notebooks. Use uploading only if you have previously created the notebook withing your datalake setup. Random notebook uploading will not work.

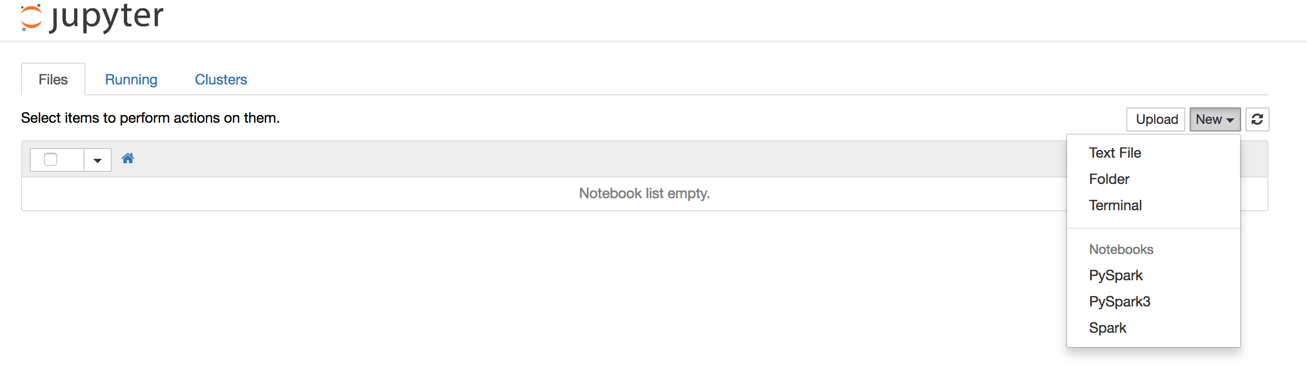
****

Figure 1: Jupyter home screen

Execution of first line of code will start a new spark application and provides the application specific information as shown in Figure 2. Please note that Spark Application creation can take few minutes for copying the dependencies and resource setup. The newly created notebook is provided with a SparkContext “sc” and a SparkSession “spark” by default.

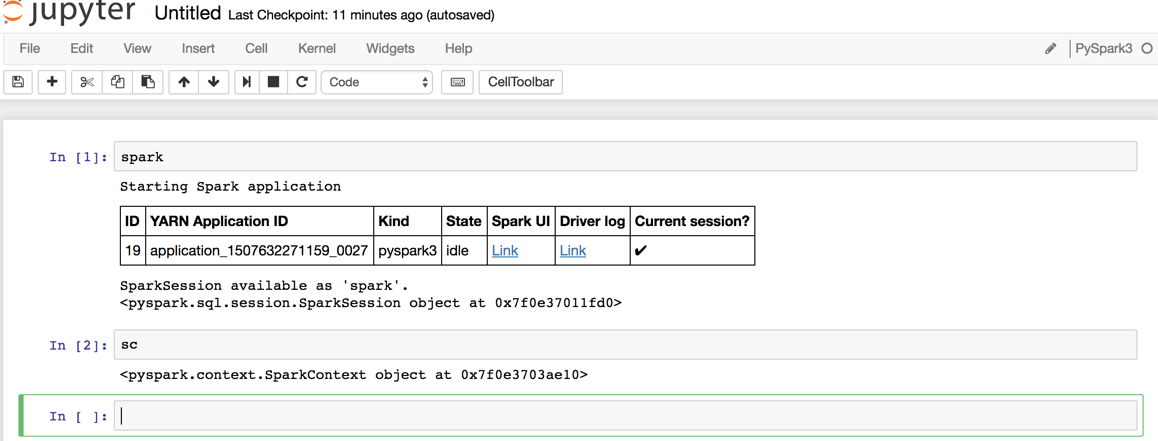


Figure 2: Jupyter notebook, Spark application init.

Once the Spark application is available, then you can start coding.

If you want to add some external spark [packages](https://spark-packages.org/) (e.g., from Spark Packages Community). You can user %%configure cell magics for that as shown in Figure 3.

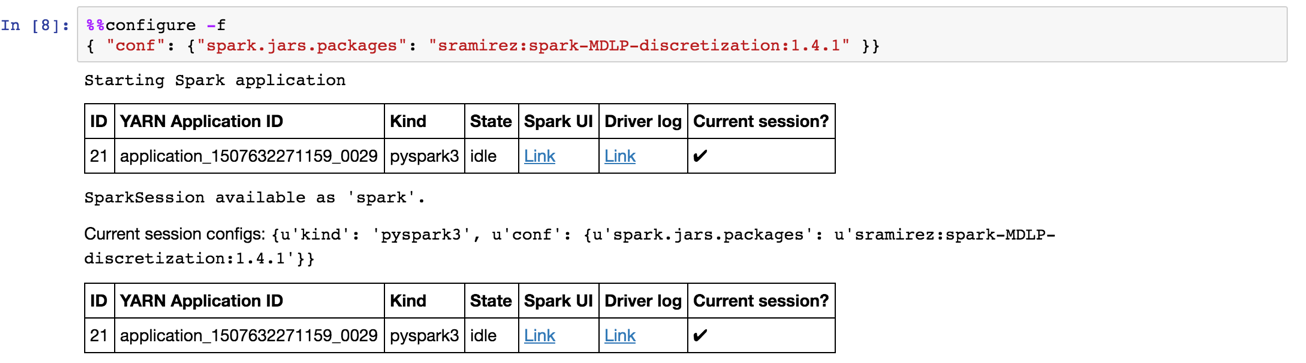


Figure 3: Adding external packages

With –f switch, %%configure will stop the application, add configurations and restart applications. You have re-excute the code. If you add %%configure in the beginning before the creation of spark application then you do not need –f switch. Please, be carefull with the packages and use only those that support scala 2.11.

If you have created temporary SparkSQL table or want to use Hive table using SQL syntax in Jupyter notebook then you can use %sql cell magic. Some default visualizations are also available with it.

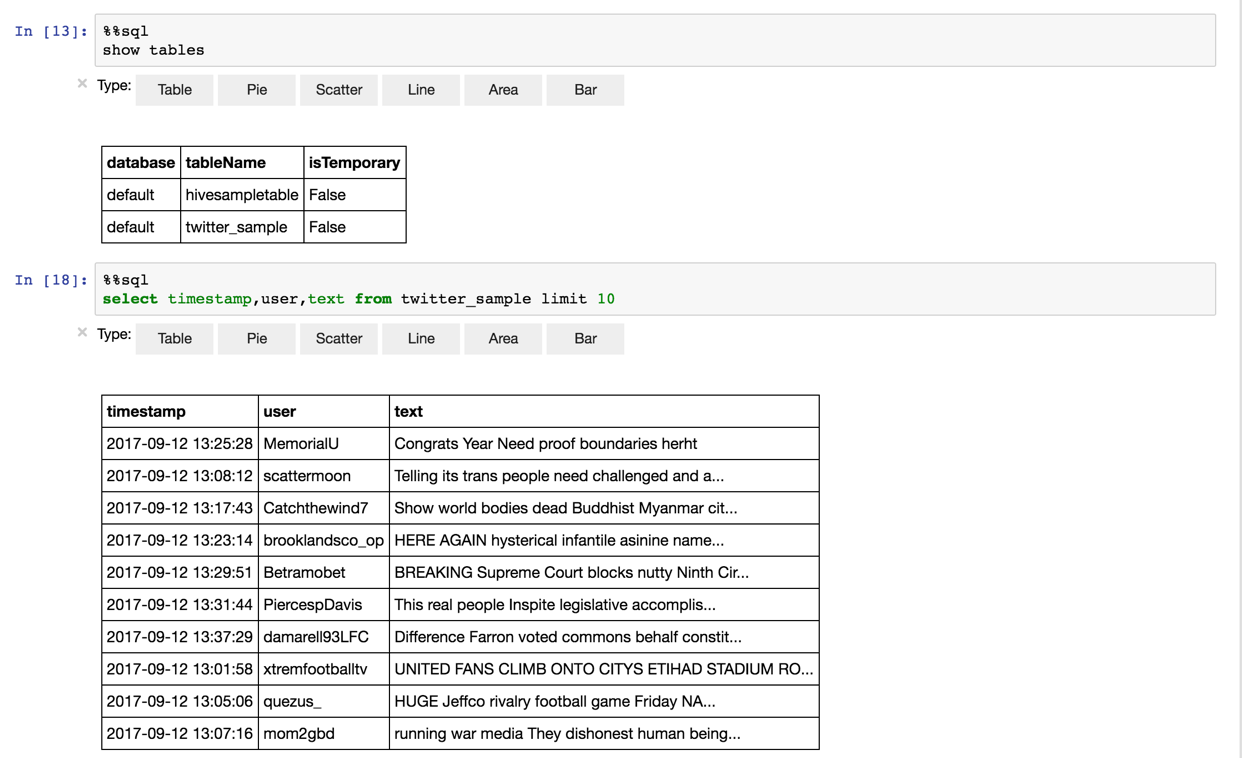


Figure 4: Using SQL syntax in Sparks

You can see complete list of cell magics from this [link](https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-apache-spark-jupyter-notebook-kernels). Please do not try to change the spark cluster parameters e.g., executor memory and vcpu. This can cause problems.

Python – Interpreter Extras:

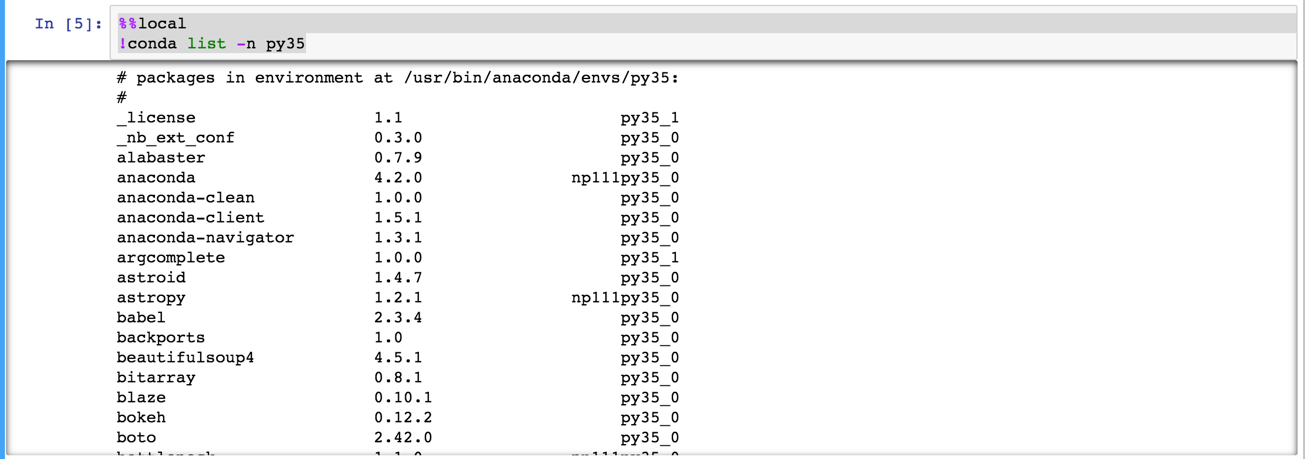
You can see the list of installed packages as in figure below. Please note that this will give you the list of locally installed packaged on the head node. %%local cell magic provides a way of working with local envrionment without pyspark distributed processing.

Figure 5: Python local packages

## Livy – Spark-jobserver

If you have coded your Spark self-contained applications, Livy spark-jobserver provides you a RESTful interface for submitting spark jobs remotely.

Livy requires a REST API client e.g., curl. You can post a job to the spark cluster available on the datalake and also get its status. Before posting a spark job you need to copy the binaries e.g., uber jars in case of Java / Scala spark applications or script sin case of PySpark.

You can copy the jars or .py files to datalake storage via FTP/SCP to headnode and then to datalake store using hadoop command line utility, e.g.,

* Copy (scp) to the datalake head node

scp -i <path\_to>/<private\_ssh\_key> <path\_to>/<spark\_jar\_script> <user>@<datalake\_hostname>:~/

* Login in to the datalake head node

ssh -i <path\_to>/<private\_ssh\_key> <user>@<datalake\_hostname>:~/

* Copy to a datalake store path e.g.,

hadoop fs -put spark-scala-app.jar /user/spark/

You can pass the spark job submission parameters as JSON with the POST request.

curl -k --user "<username>:<password>" -v -H 'Content-Type: application/json' -X POST -d '{ "file":"adl://junctionmaindls.azuredatalakestore.net/clusters/<datalake\_name>/<path>/<to>/<spark>/<jar>", "className":"<SparkApp\_ClassName>", "other\_spark\_parameter":"<parameter\_value>" }' "<Datalake URL>/livy/batches"

example, for datalake “side1datalake” the curl command will be.

curl -k --user "user:password" -v -H 'Content-Type: application/json' -X POST -d '{ "file":"adl://junctionmaindls.azuredatalakestore.net/clusters/side1datalake/user/spark/spark-scala-app.jar", "className":"com.spark.test.SparkScalaApp" }' "https://side1datalake-junction-side1-hd.azurehdinsight.net/livy/batches"

Successful submission of this job should return e.g.,

{"id":0,"state":"starting","appId":null,"appInfo":{"driverLogUrl":null,"sparkUiUrl":null},"log":[]}

You can check the status of the submitted application using a HTTP GET request.

curl -k --user "<user>:<password>" -v -X GET ""<Datalake URL>//livy/batches"

OR You can also view the datalake YARN UI as shown in Figure below.

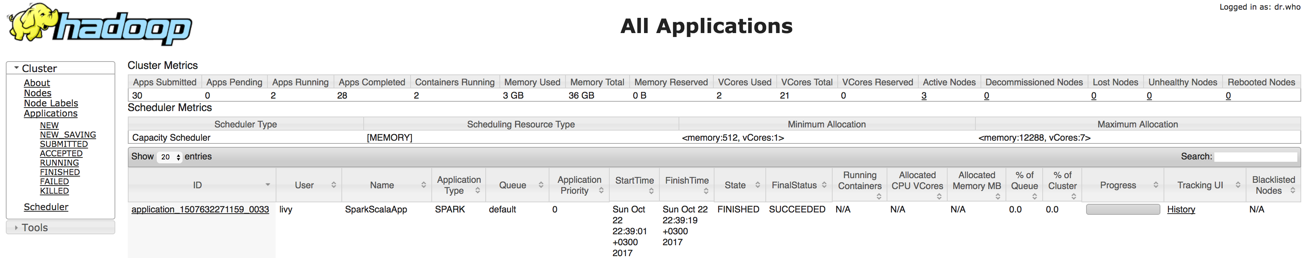


Figure 6: Spark job status on YARN UI

Please refer to Azure HDInsight [documentation](https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-apache-spark-livy-rest-interface) for further details.

# Accessing and loading data

## Shared Data Sets

Each team has access to some shared data sources, i.e.,

1. synthetic health data and
2. sensor data combining well-being data and data from IoT sensors

Teams get separate instructions with data descriptions and data lake store paths to access the data. Team can also upload their own data sets to their private side data lake stores. Such data sets will be only visible to the respective team.

## Uploading Own Data sets

For uploading data teams can FTP/SCP data to head node as described in Livy- Spark-Jobserver [Section](#_Livy_–_Spark-jobserver).

When uploading the own data sets, please remember to append the base path of your side data lake store before the path you use with hdfs command line. This path should follow the pattern as given in the example below

“adl://junctionmaindls.azuredatalakestore.net/clusters/<side\_datalake\_name>/”

## Reading & Wrangling Data with Apache Spark

Teams can use Apache Spark to read data using the provided paths or the paths of the uploaded data sets e.g., Following code snippet & Figure shows how to read a sample of synthetic health data in tab-separated format from Jupyter PySpark notebook as a dataframe.

df = spark.read.csv("adl://junctionmaindls.azuredatalakestore.net/clusters/maindatalake/staging/sample/diab\_synt\_otos\_ja\_selitteet\_lab.tsv",header=True,sep="\t")

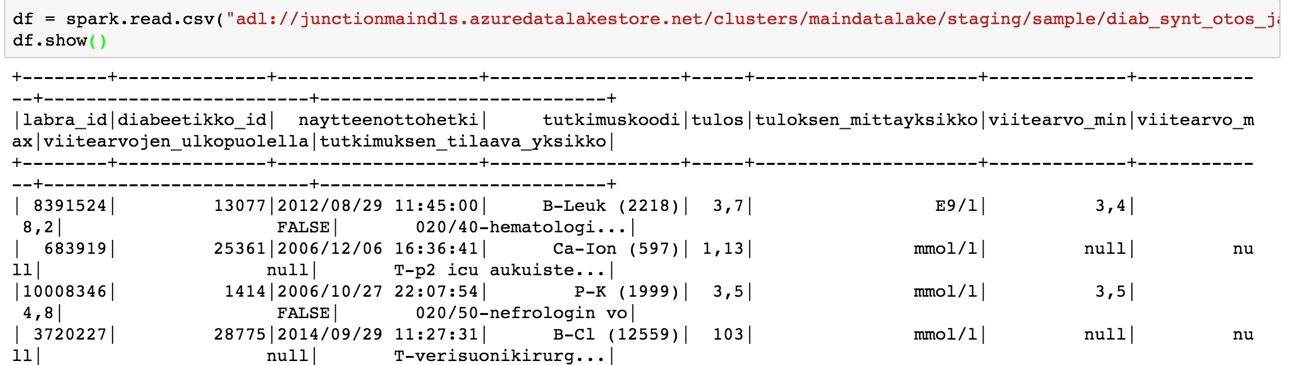


Figure 7: Read data from datalake store to PySpark Dataframe

Please note that the data path here is just as an example and the actual data paths are provided on the day 1 with the other information to access and use datalake.

The details of using Spark Dataframes are available on this [link](https://spark.apache.org/docs/2.1.1/sql-programming-guide.html). Please use documentation for Spark v2.1.1.

It is worthwhile to mention some utility things such as caching the data for in-memory processing with .cache() or persist() with different [storage levels](http://spark.apache.org/docs/2.1.1/api/python/pyspark.sql.html?highlight=persist#pyspark.sql.DataFrame.persist). You can create temporary tables and use sql like syntax to query data using createOrReplaceTempView("table\_name"). Use %%sql cell magic if you want to query data is full SQL syntax. You can also store the dataframe as Hive tables with write().saveAsTable("tableName").

In case of some complex data structures such as nested JSONs. You can apply external libaries, e.g.,

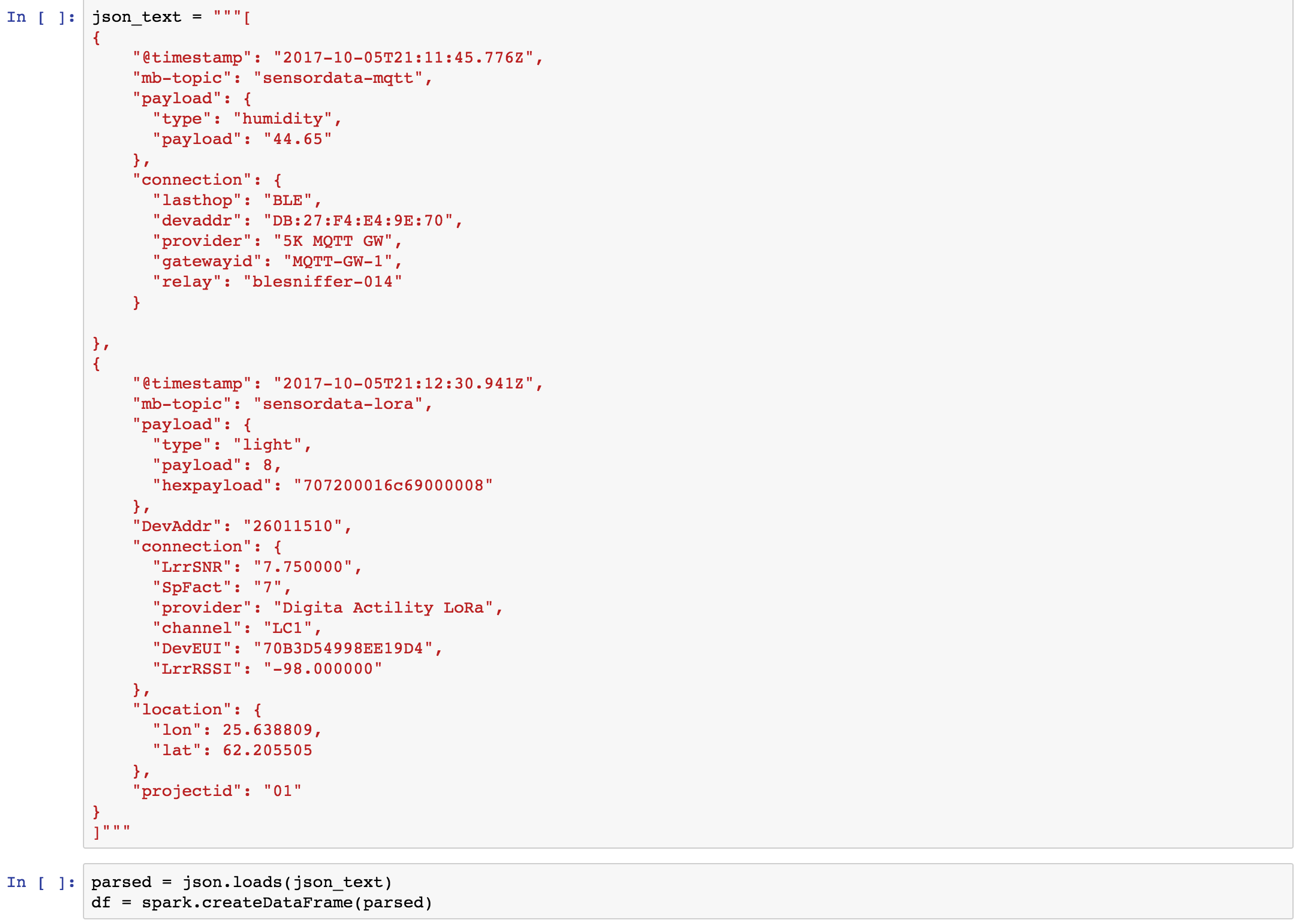


Figure 8: Nested JSONs.

There is a detailed [tutorial](https://docs.databricks.com/spark/latest/spark-sql/complex-types.html) from Databricks on handling complex data formats in Apache Spark SQL. Please refer to Apache Spark [documentation](https://spark.apache.org/docs/2.1.1/quick-start.html) for further details.