

Java-Success.com

Prepare to fast-track, choose & go places with 800+ Java & Big Data Q&As with lots of code & diagrams.

[Home](#) [Why? ▾](#) [300+ Java FAQs ▾](#) [300+ Big Data FAQs ▾](#) [Courses ▾](#)

[👤 Membership ▾](#) [Your Career ▾](#)

[Home](#) > [bigdata-success.com](#) > [Tutorials - Big Data](#) > [TUT - Spark Scala on Zeppelin](#) >

12: Spark on Zeppelin – Dataframe pivot

12: Spark on Zeppelin – Dataframe pivot

 Posted on [September 17, 2018](#)

Pre-requisite: Docker is installed on your machine for Mac OS X (E.g. \$ brew cask install docker) or Windows 10. [Docker interview Q&As](#). This extends [setting up Apache Zeppelin Notebook](#).

Step 1: Pull this from the docker hub, and build the image with the following command.

```
1 $ docker pull apache/zeppelin:0.7.3
2
```

You can verify the image with the “docker images” command.

300+ Java Interview FAQs

300+ Java FAQs



16+ Java Key Areas Q&As



150+ Java Architect FAQs



80+ Java Code Quality Q&As



150+ Java Coding Q&As



300+ Big Data Interview FAQs

300+ Big Data FAQs



Tutorials - Big Data



TUT -  Starting Big Data

TUT - Starting Spark & Scala

Step 2: Run the container with the above image.

```
1 $ docker run --rm -it -p 8080:8080 apache/zeppelin
2
```

Step 3: Open Zeppelin notebook via a web browser "http://localhost:8080". Create a notebook with "spark" as a default interpreter.

Calculate the average temperature for each station for each month

The **pivot** function to the rescue. A pivot is an aggregation where one or more of the grouping columns with distinct values (E.g. Statuses like pending, completed, etc or Types like Basic, Advanced, etc) transposed into individual columns. Pivot tables are an essential part of data analysis and reporting. In the following example let's transpose the distinct "year-month" values for each weather station.

```
1 %spark
2
3
4 import java.sql.Date
5 import org.apache.spark.sql.functions._
6
7 case class Weather (stationId: Integer, date: java
8
9 val weather = Seq(
10   Weather(1, Date.valueOf("2018-06-01"), 34.0 ),
11   Weather(2, Date.valueOf("2017-12-01"), 32.0),
12   Weather(2, Date.valueOf("2018-06-01"), 28.0),
13   Weather(2, Date.valueOf("2017-01-01"), 26.0),
14   Weather(1, Date.valueOf("2017-01-01"), 24.0),
15   Weather(1, Date.valueOf("2017-12-01"), 30.0),
16   Weather(2, Date.valueOf("2017-01-01"), 26.0),
17   Weather(1, Date.valueOf("2017-12-01"), 24.0),
18   Weather(1, Date.valueOf("2018-06-01"), 30.0)
19 )
20
```

TUT - Starting with Python

TUT - Kafka

TUT - Pig

TUT - Apache Storm

TUT - Spark Scala on Zeppelin

TUT - Cloudera

TUT - Cloudera on Docker

TUT - File Formats

TUT - Spark on Docker

TUT - Flume

TUT - Hadoop (HDFS)

TUT - HBase (NoSQL)

TUT - Hive (SQL)

TUT - Hadoop & Spark

TUT - MapReduce

TUT - Spark and Scala

TUT - Spark & Java

TUT - PySpark on Databricks

TUT - Zookeeper

800+ Java Interview Q&As

300+ Core Java Q&As



300+ Enterprise Java Q&As



150+ Java Frameworks Q&As



120+ Companion Tech Q&As



Tutorials - Enterprise Java



```

21 val weathereDF = spark.createDataFrame(
22   spark.sparkContext.parallelize(weather)
23 )
24
25 weathereDF.show()
26
27
28 val monthlyWeatherDF = weathereDF.withColumn("year",
29   year($"date"),
30   .groupBy("stationId")
31   .pivot("year_month", Seq("2017-01", "2017-12", "2018-06"))
32   .agg(avg($"temperature"))
33
34 val monthlyAvgTempDF = monthlyWeatherDF.agg(avg($"temperature"))
35 monthlyAvgTempDF.show()

```

Output:

```

1 import java.sql.Date
2 import org.apache.spark.sql.functions._
3 defined class Weather
4 weather: Seq[Weather] = List(Weather(1,2018-06-01,34.0),
5   Weather(2,2017-12-01,32.0),
6   Weather(2,2018-06-01,28.0),
7   Weather(2,2017-01-01,26.0),
8   Weather(1,2017-01-01,24.0),
9   Weather(1,2017-12-01,30.0),
10  Weather(2,2017-01-01,26.0),
11  Weather(1,2017-12-01,24.0),
12  Weather(1,2018-06-01,30.0))
13
14 weathereDF: org.apache.spark.sql.DataFrame = [stationId, date, temperature]
15
16 +-----+-----+-----+
17 |stationId|      date|temperature|
18 +-----+-----+-----+
19 |         1|2018-06-01|         34.0|
20 |         2|2017-12-01|         32.0|
21 |         2|2018-06-01|         28.0|
22 |         2|2017-01-01|         26.0|
23 |         1|2017-01-01|         24.0|
24 |         1|2017-12-01|         30.0|
25 |         2|2017-01-01|         26.0|
26 |         1|2017-12-01|         24.0|
27 |         1|2018-06-01|         30.0|
28 +-----+-----+-----+
29
30 monthlyWeatherDF: org.apache.spark.sql.RelationalDataFrame = [stationId, year_month, temperature]
31
32 +-----+-----+-----+
33 |stationId|year_month|temperature|
34 +-----+-----+-----+
35 |         1|2017-01|         24.0|
36 |         1|2017-12|         30.0|
37 |         1|2018-06|         30.0|
38 |         2|2017-01|         26.0|
39 |         2|2017-12|         26.0|
40 |         2|2018-06|         28.0|
41 +-----+-----+-----+
42
43 monthlyAvgTempDF: org.apache.spark.sql.DataFrame = [stationId, avg(temperature)]
44
45 +-----+-----+
46 |stationId|avg(temperature)|
47 +-----+-----+
48 |         1|         24.000000|
49 |         2|         26.000000|
50 +-----+-----+

```

◀ 11: Spark on Zeppelin – Dataframe groupBy, collect_list, explode & window

17+ DevOps CI/CD Jenkins interview Q&As ▶

Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty Ltd. The EmpoweringTech pty Ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances into consideration. The EmpoweringTech pty Ltd will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. [Privacy Policy](#).