

SI618: DATA MANIPULATION & ANALYSIS

SPARK DATAFRAMES

STARTING A 3-NODE CLUSTER ON AWS EMR, IMPORTING A NOTEBOOK TO S3, AND RUNNING IT

General Configuration

Cluster name

☒ **Logging** ⓘ

S3 folder ⓘ

Launch mode ☒ **Cluster** ⓘ ☐ **Step execution** ⓘ

Software configuration

Release ⓘ

Applications ☒ **Core Hadoop:** Hadoop 2.8.5 with Ganglia 3.7.2, Hive 2.3.4, Hue 4.3.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.1

☐ **HBase:** HBase 1.4.9 with Ganglia 3.7.2, Hadoop 2.8.5, Hive 2.3.4, Hue 4.3.0, Phoenix 4.14.1, and ZooKeeper 3.4.13

☐ **Presto:** Presto 0.214 with Hadoop 2.8.5 HDFS and Hive 2.3.4 Metastore

☐ **Spark:** Spark 2.4.0 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.1

☐ **Use AWS Glue Data Catalog for table metadata** ⓘ

Hardware configuration

Instance type

Number of instances (1 master and 2 core nodes)

Security and access

EC2 key pair ⓘ [Learn how to create an EC2 key pair.](#)

Permissions ☒ **Default** ☐ **Custom**

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role [EMR_DefaultRole](#) ⓘ

EC2 instance profile [EMR_EC2_DefaultRole](#) ⓘ

Cancel

Create cluster

Amazon EMR

Clusters

Security configurations

VPC subnets

Events

Notebooks

Help

What's new

Clone

Terminate

AWS CLI export

Cluster: My cluster **Waiting** Cluster ready after last step completed.

Summary

Application history

Monitoring

Hardware

Configurations

Events

Steps

Bootstrap actions

Connections:

[Enable Web Connection](#) – Zeppelin, Spark History Server, Ganglia, Resource Manager ... (View All)

Master public DNS:

ec2-3-85-144-90.compute-1.amazonaws.com [SSH](#)

Tags:

-- [View All / Edit](#)

Summary

ID: j-2OIQM4GWBCEAI

Creation date: 2019-04-09 10:41 (UTC-4)

Elapsed time: 9 minutes

Auto-terminate: No


Termination protection: Off [Change](#)

Configuration details

Release label: emr-5.29.0

Hadoop distribution: Amazon

Applications: Ganglia 3.7.2, Spark 2.4.0, Zeppelin 0.8.1

Log URI: s3://aws-logs-834790723353-us-east-1/elasticmapreduce/ 

EMRFS consistent view: Disabled

Custom AMI ID: --

Network and hardware

Availability zone: us-east-1b

Subnet ID: [subnet-c33d429f](#) 

Master: **Running** 1 m3.xlarge

Core: **Running** 2 m3.xlarge

Task: --

Create notebook

Name and configure your notebook

Name your notebook, choose a cluster or create one, and

Notebook name*

Names may only contain letters (a

Description

256 characters max.

Cluster* ☒ Choose an existing cluster

☐ Create a cluster

Security groups ☒ Use default security groups

☐ Choose security groups

AWS service role*

Notebook location* Choose an S3 location where files for this notebook are saved.

☒ Use the default S3 location
s3://aws-emr-resources-834790723353-us-east-1/notebooks/

☐ Choose an existing S3 location in us-east-1

► **Tags**

* Required

Choose a cluster



The listed clusters meet notebook requirements. They are in an EC2-VPC, running EMR 5.18.0 or later, and have Hadoop, Spark, and Livy installed. [Learn more](#)

The notebook can be opened once the cluster is in Waiting or Running status.

Filter: 1 cluster (all loaded)

	Name	ID	Status
<input checked="" type="radio"/>	My cluster	j-2OIQM4GWBCEAI	Waiting Cluster ready

Notebook: 330_21_Big_Data_III Pending Notebook is attaching to cluster j-2OIQM4GWBCEAI. Notebook can now be used in local mode.

Open

Stop

Delete

Notebook

Notebook ID: e-57P1H88BYJF47OEZIOIACZPY5

Description: --

Last modified: 1 second ago ⓘ

Last modified by: ...assumed-role/vocstartsoft/user266495=cteplovs@umich.edu ⓘ

Created on: 2019-04-09 10:51 (UTC-4)

Created by: ...assumed-role/vocstartsoft/user266495=cteplovs@umich.edu ⓘ

Service IAM role: [EMR_Notebooks_DefaultRole](#) ↗

Notebook tags: creatorUserId = AROAJEOPKG4T7CS7RAHI2:user266495=cteplovs@umich.edu [View All / Edit](#)

Notebook location: s3://aws-emr-resources-834790723353-us-east-1/notebooks/ 📁

Cluster

Cluster: My cluster

Cluster Id: [j-2OIQM4GWBCEAI](#)

Cluster status: Waiting Cluster ready after last step completed.

Cluster tags: --

Step logs: s3://aws-logs-834790723353-us-east-1/elasticmapreduce/ 📁

AWS Management Console

AWS services

Find Services

You can enter names, keywords or acronyms.

 *Example: Relational Database Service, database, RDS*

▼ Recently visited services



EMR



IAM



RDS



S3




EC2

▼ All services



Compute

EC2

Lightsail 

ECR

ECS

EKS

Lambda

Batch

Elastic Beanstalk

Serverless Application Repository



Storage

S3

EFS

FSx

S3 Glacier



Management & Governance

AWS Organizations

CloudWatch

AWS Auto Scaling

CloudFormation

CloudTrail

Config

OpsWorks

Service Catalog

Systems Manager

Trusted Advisor

Managed Services

Control Tower

AWS License Manager

AWS Well-Architected Tool

Resource Access Manager



AWS Cost Management

AWS Cost Explorer

AWS Budgets

AWS Marketplace Subscriptions



Mobile

AWS Amplify

Mobile Hub

AWS AppSync

Device Farm



AR & VR

Amazon Sumerian



Services ▾

Resource Groups ▾



vocstartsoft/user266495=ctepl... ▾

Global ▾

Support ▾

Amazon S3 > aws-emi-resources-834790723353-us-east-1 > notebooks > e-57P1H88BYJF47OEZIOIACZPY5

Overview

🔍 Type a prefix and press Enter to search. Press ESC to clear.

Upload Create folder Download Actions ▾

US East (N. Virginia)

Viewing 1 to 2

☐ Name ▾

Last modified ▾

Size ▾

Storage class ▾

☐ metadata

--

--

--

☐ 330_21_Big_Data_III.ipynb

Apr 9, 2019 10:54:02 AM GMT-0400

38.8 KB

Standard

Viewing 1 to 2

Notebook: 330_21_Big_Data_III **Ready** Notebook is ready to run jobs on cluster j-2OIQM4GWBCEAI.

Open

Stop

Delete

Notebook

Notebook ID: e-57P1H88BYJF47OEZIOIACZPY5

Description: --

Last modified: 2 minutes ago ⓘ

Last modified by: ...assumed-role/vocstartsoft/user266495=cteplovs@umich.edu ⓘ

Created on: 2019-04-09 10:51 (UTC-4)

Created by: ...assumed-role/vocstartsoft/user266495=cteplovs@umich.edu ⓘ

Service IAM role: [EMR_Notebooks_DefaultRole](#) ↗

Notebook tags: creatorUserId = AROAJEOPKG4T7CS7RAHI2:user266495=cteplovs@umich.edu [View All / Edit](#)

Notebook location: s3://aws-emr-resources-834790723353-us-east-1/notebooks/ 📁

Cluster

Cluster: My cluster

Cluster Id: [j-2OIQM4GWBCEAI](#)

Cluster status: **Waiting** Cluster ready after last step completed.

Cluster tags: --

Step logs: s3://aws-logs-834790723353-us-east-1/elasticmapreduce/ 📁

SPARK APIS

- ▶ Three APIs:
 - ▶ RDDs (last week)
 - ▶ DataSets
 - ▶ DataFrames

SPARK RDDS:

- ▶ Resilient Distributed Datasets
- ▶ low-level
- ▶ support creation, transformations, and actions
 - ▶ creation:
 - ▶ `parallelize()` an existing data structure;
load a file with `textFile()`
 - ▶ transformations return a new RDD (e.g. `map()`, `reduce()`)
 - ▶ actions return non-RDDs (e.g. `count()`, `collect()`)

SPARK DATASETS

- ▶ distributed collection of data
- ▶ only available via Scala and Java (i.e. no Python interface)
- ▶ we will not be using Datasets

SPARK DATAFRAMES

- ▶ our main focus will be on Spark DataFrames
- ▶ DataFrames are Spark Datasets organized into named columns (sound familiar?)
- ▶ they're *tables*
- ▶ conceptually very similar to pandas and R DataFrames

SPARK DATAFRAMES: GETTING STARTED

- ▶ all interaction is via a SparkSession

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .getOrCreate()
```

- ▶ entry point to programming Spark with the DataFrame API
- ▶ to create a SparkSession, use the builder pattern shown above ON NON-DATABRICKS PLATFORMS
- ▶ more or less equivalent to SparkContext from last week
- ▶ NOTE: DATABRICKS GIVES YOU THIS "FOR FREE"

CREATING SPARK DATAFRAMES

- ▶ once you have a `SparkSession` you can create a `DataFrame`
- ▶ a `DataFrame` can be created from:
 - ▶ a list
 - ▶ an RDD
 - ▶ a specially formatted JSON file

CREATING A DATAFRAME FROM A LIST

- ▶ list of tuples: include a list of column names
- ▶ list of values: specify value type

```
# create a DataFrame from a list of tuples
```

```
df_from_other_list = spark.createDataFrame(  
    [ ('Chris', 67), ('Frank', 70)], [ 'name', 'score' ] )
```

```
df_from_other_list.show()
```

name	score
Chris	67
Frank	70

```
# create a DataFrame from a list of values;  
# note that you must specify the data type
```

```
from pyspark.sql.types import FloatType
```

```
df_from_list = spark.createDataFrame(  
    [1.0, 2.0, 3.0, 4.0, 5.0], FloatType())
```

```
df_from_list.show()
```

value
1.0
2.0
3.0
4.0
5.0

CREATING A DATAFRAME FROM AN RDD

- ▶ simple if you're ok with default column names
- ▶ need to create a `pyspark.sql.Row` if you want better column names

```
# create an RDD
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lot_rdd = sc.parallelize([('Chris',67),('Frank',70)])
```

```
# create a DataFrame from an RDD
dfPeople = spark.createDataFrame(lot_rdd)
dfPeople.show()
```

```
# create a Row to include column names
from pyspark.sql import Row
```

```
lot_rdd_named_columns = lot_rdd \
    .map(lambda x: Row(name=x[0], score=int(x[1])))
dfPeople_named_columns = spark \
    .createDataFrame(lot_rdd_named_columns)
dfPeople_named_columns.show()
```

```
+-----+----+
|    _1 | _2 |
+-----+----+
| Chris | 67 |
| Frank | 70 |
+-----+----+
```

```
+-----+-----+
|  name | score |
+-----+-----+
| Chris |    67 |
| Frank |    70 |
+-----+-----+
```

CREATING A DATAFRAME FROM A JSON FILE

```
# read a specially formatted JSON file (one JSON object per line)
df = spark.read.json("business.json")
# Displays the content of the DataFrame to stdout
df.show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
|          address|          attributes|          business_id|          categories|          city|
hours|is_open|    latitude|    longitude|          name|    neighborhood|postal_code|review_count|stars|state|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
|4855 E Warner Rd,...|[true,null,null,n...|FYWN1wneV18bWNgQj...|[Dentists, Genera...|    Ahwatukee|
|7:30-17:00,7:30-...|    1|    33.3306902|    -111.9785992|    Dental by Design|
```

INFERRING SCHEMA

- ▶ from previous example, `df.printSchema()` gives:

```
root
|-- address: string (nullable = true)
|-- attributes: struct (nullable = true)
|   |-- AcceptsInsurance: boolean (nullable = true)
|   |-- AgesAllowed: string (nullable = true)
|   |-- Alcohol: string (nullable = true)
|   |-- Ambience: struct (nullable = true)
|       |-- casual: boolean (nullable = true)
|       |-- classy: boolean (nullable = true)
|       |-- divey: boolean (nullable = true)
|       |-- hipster: boolean (nullable = true)
|       |-- intimate: boolean (nullable = true)
|       |-- romantic: boolean (nullable = true)
|       |-- touristy: boolean (nullable = true)
|       |-- trendy: boolean (nullable = true)
|       |-- upscale: boolean (nullable = true)
|   |-- BYOB: boolean (nullable = true)
|   |-- BYOBCorkage: string (nullable = true)
|   |-- BestNights: struct (nullable = true)
|       |-- friday: boolean (nullable = true)
|       |-- monday: boolean (nullable = true)
|       |-- saturday: boolean (nullable = true)
|       |-- sunday: boolean (nullable = true)
|       |-- thursday: boolean (nullable = true)
|       |-- tuesday: boolean (nullable = true)
|       |-- wednesday: boolean (nullable = true)
|-- is_open: long (nullable = true)
|-- latitude: double (nullable = true)
|-- longitude: double (nullable = true)
|-- name: string (nullable = true)
|-- neighborhood: string (nullable = true)
|-- postal_code: string (nullable = true)
|-- review_count: long (nullable = true)
|-- stars: double (nullable = true)
|-- state: string (nullable = true)
```

CREATING A DATA FRAME FROM A FILE (IN GENERAL)

- ▶ spark can load a number of different formats: json, parquet, jdbc, orc, libsvm, csv, text

```
df = spark.read.load("examples/src/main/resources/people.json", format="json")
```

COLUMN SELECTION

```
# Select only the "name" column  
df.select("name").show()
```

```
+-----+  
|          name          |  
+-----+  
|    Dental by Design    |  
| Stephen Szabo Salon    |  
| Western Motor Veh...   |  
|    Sports Authority    |  
| Brick House Taver...   |  
|          Messina       |  
|          BDJ Realty     |  
|          Soccer Zone    |  
|    Any Given Sundae     |  
| Detailing Gone Mo...   |  
|    East Coast Coffee    |  
| CubeSmart Self St...   |  
| T & T Bakery and ...   |  
| Complete Dental Care   |  
| Showmars Governme...   |  
|          Alize Catering |  
|          T & Y Nail Spa |  
| Meineke Car Care ...   |  
| Senior's Barber Shop   |  
| Maxim Bakery & Re...   |  
+-----+  
only showing top 20 rows
```

COLUMN SELECTION WITH MANIPULATION

```
# Select all businesses, but increment the review_count by 1
df.select(df['name'], df['review_count'] + 1).show()
```

name	(review_count + 1)
Dental by Design	23
Stephen Szabo Salon	12
Western Motor Veh...	19
Sports Authority	10
Brick House Taver...	117
Messina	6
BDJ Realty	6
Soccer Zone	10
Any Given Sundae	16
Detailing Gone Mo...	8
East Coast Coffee	4
CubeSmart Self St...	24
T & T Bakery and ...	39
Complete Dental Care	6
Showmars Governme...	8
Alize Catering	13
T & Y Nail Spa	21
Meineke Car Care ...	10
Senior's Barber Shop	66
Maxim Bakery & Re...	35

only showing top 20 rows

FILTERING

```
# Select businesses with 4 or more stars  
df.filter(df['stars'] >= 4).show()
```

GROUPBY AND SORTING

```
# Count businesses by stars
```

```
df.groupby("stars").count().show()
```

stars	count
3.5	32038
4.5	24796
2.5	16148
1.0	3788
4.0	33492
3.0	23142
2.0	9320
1.5	4303
5.0	27540

```
# Count businesses by stars and sort the output
```

```
df.groupby("stars").count().sort("stars",ascending=False).show()
```

stars	count
5.0	27540
4.5	24796
4.0	33492
3.5	32038
3.0	23142
2.5	16148
2.0	9320
1.5	4303
1.0	3788

EXPLODE

- ▶ create a row for each value in a list/array/etc.

```
# create a DataFrame from a list of tuples
df_from_other_list2 = spark.createDataFrame(
    [('Chris',[67,42]),('Frank',[70,72])], ['name','scores']
)
df_from_other_list2.show()
```

```
+-----+-----+
| name|  scores|
+-----+-----+
|Chris|[67, 42]|
|Frank|[70, 72]|
+-----+-----+
```

```
from pyspark.sql.functions import explode
```

```
df_exploded = df_from_other_list2.withColumn('score',explode('scores'))
```

```
df_exploded.show()
```

```
+-----+-----+-----+
| name|  scores|score|
+-----+-----+-----+
|Chris|[67, 42]|   67|
|Chris|[67, 42]|   42|
|Frank|[70, 72]|   70|
|Frank|[70, 72]|   72|
+-----+-----+-----+
```

WHEN... OTHERWISE

```
import pyspark.sql.functions as F
from pyspark.sql.functions import col
df_exploded.withColumn('good', F.when(df_exploded['score']>50,1).otherwise(0)).show()
```

name	scores	score	good
Chris	[67, 42]	67	1
Chris	[67, 42]	42	0
Frank	[70, 72]	70	1
Frank	[70, 72]	72	1

JOIN

- ▶ The following code is a complex example in which we have two DataFrames: people and department. We are filtering people with age > 30 and joining to department as shown. We are then grouping by department name and (people) gender, then aggregating two variables: the average salary and the maximum age.

```
people.filter(people.age > 30).join(department, people.deptId == department.id) \
    .groupBy(department.name, "gender").agg({"salary": "avg", "age": "max"})
```

JOIN

`join(other, on=None, how=None)`

Joins with another **DataFrame**, using the given join expression.

- Parameters:**
- **other** – Right side of the join
 - **on** – a string for the join column name, a list of column names, a join expression (Column), or a list of Columns. If *on* is a string or a list of strings indicating the name of the join column(s), the column(s) must exist on both sides, and this performs an equi-join.
 - **how** – str, default `inner`. Must be one of: `inner`, `cross`, `outer`, `full`, `full_outer`, `left`, `left_outer`, `right`, `right_outer`, `left_semi`, and `left_anti`.

The following performs a full outer join between `df1` and `df2`.

```
>>> df.join(df2, df.name == df2.name, 'outer').select(df.name, df2.height).collect()
[Row(name=None, height=80), Row(name=u'Bob', height=85), Row(name=u'Alice', height=None)]
```

```
>>> df.join(df2, 'name', 'outer').select('name', 'height').collect()
[Row(name=u'Tom', height=80), Row(name=u'Bob', height=85), Row(name=u'Alice', height=None)]
```

```
>>> cond = [df.name == df3.name, df.age == df3.age]
>>> df.join(df3, cond, 'outer').select(df.name, df3.age).collect()
[Row(name=u'Alice', age=2), Row(name=u'Bob', age=5)]
```

```
>>> df.join(df2, 'name').select(df.name, df2.height).collect()
[Row(name=u'Bob', height=85)]
```

```
>>> df.join(df4, ['name', 'age']).select(df.name, df.age).collect()
[Row(name=u'Bob', age=5)]
```

SQL JOINS

