Connecting to S3

Apache Spark™ and Databricks® allow you to connect to virtually any data store including Amazon S3.

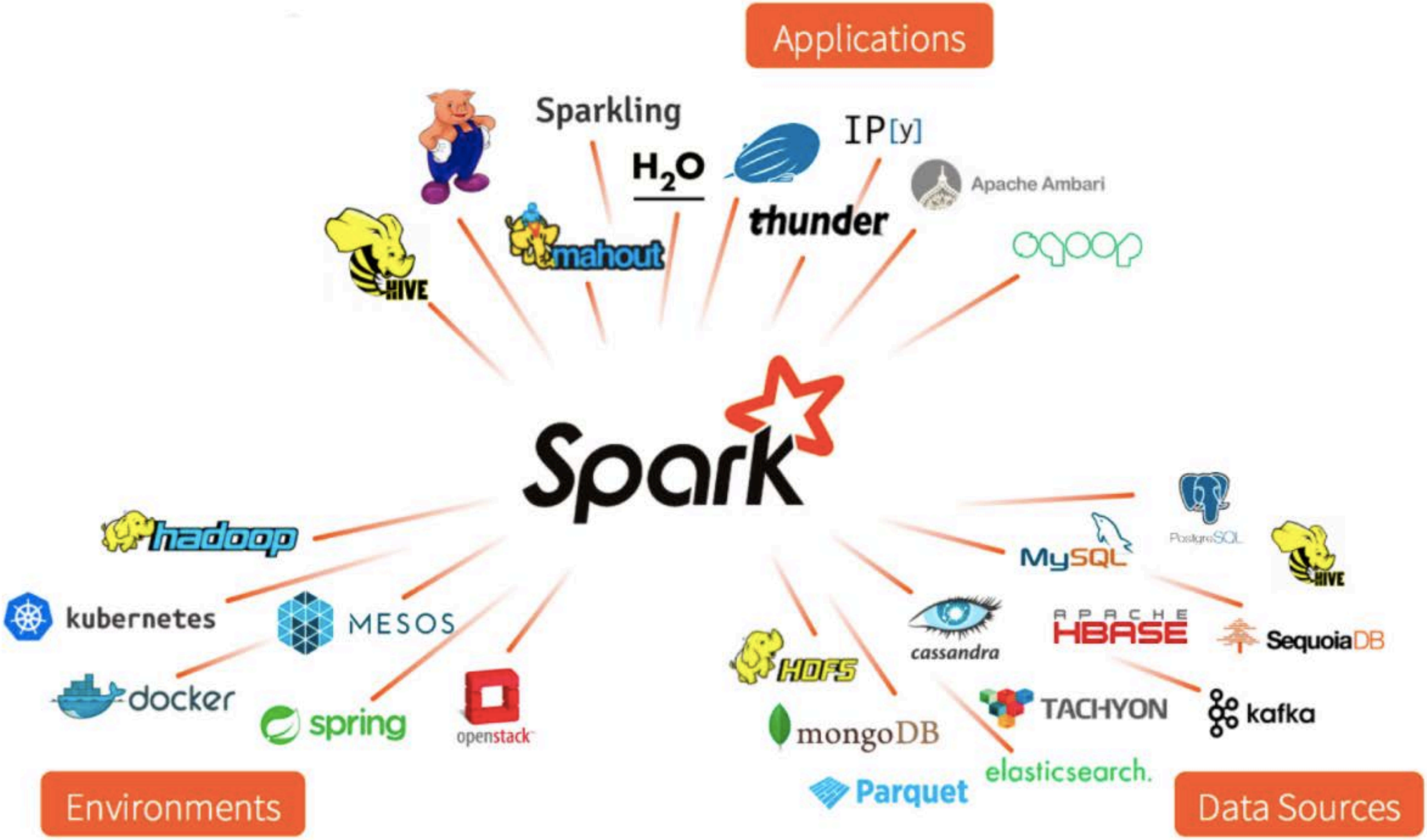
In this lesson you:

- Mount and access data in S3
- Define options when reading from S3

Spark as a Connector

Spark quickly rose to popularity as a replacement for the Apache Hadoop™ MapReduce paradigm in a large part because it easily connected to a number of different data sources. Most important among these data sources was the Hadoop Distributed File System (HDFS). Now, Spark engineers connect to a wide variety of data sources including:

- Traditional databases like Postgres, SQL Server, and MySQL
- Message brokers like Apache Kafka and Kinesis
- Distributed databases like Cassandra and Redshift
- Data warehouses like Hive
- File types like CSV, Parquet, and Avro

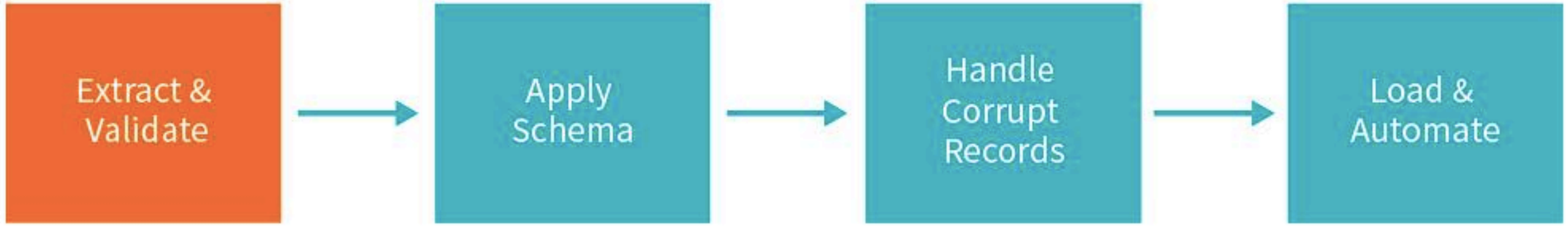

Applications
Sparkling
H2O
IP[y]
Apache Ambari
thunder
mahout
HIVE
Spark
hadoop
kubernetes
MESOS
docker
spring
openstack
Environments
MySQL
PostgreSQL
HIVE
cassandra
APACHE HBASE
SequoiaDB
HDFS
TACHYON
kafka
mongoDB
Parquet
elasticsearch.
Data Sources

DBFS Mounts and S3

Amazon Simple Storage Service (S3) is the backbone of Databricks workflows. S3 offers data storage that easily scales to the demands of most data applications and, by colocating data with Spark clusters, Databricks quickly reads from and writes to S3 in a distributed manner.

The Databricks File System, or DBFS, is a layer over S3 that allows you to mount S3 buckets, making them available to other users in your workspace and persisting the data after a cluster is shut down.

In our road map for ETL, this is the **Extract and Validate** step:



Define your AWS credentials. Below are defined read-only keys, the name of an AWS bucket, and the mount name to refer to use in DBFS.

For getting AWS keys, take a look at take a look at the AWS documentation

Cmd 10

```
awsAccessKey = "AKIAJBRYNXGHORDHZB4A"
# Encode the Secret Key to remove any "/" characters
secretKey = "a0BzE1bSegfydr3%2FGE3LSPM6uIV5A4hOUfpH8aFF".replace("/", "%2F")
awsBucketName = "databricks-corp-training/common"
```

Command took 0.04 seconds -- by huseyinyilmaz01@gmail.com at 4/26/2020, 3:01:42 PM on My Cluster

Cmd 11

In addition to the sourcing information above, we need to define a target location.

So that no two students produce the exact same mount, we are going to be a little more creative with this one.

Cmd 12

```
mountPoint = f"/mnt/etlp1s-{username}-si"
```

Command took 0.05 seconds -- by huseyinyilmaz01@gmail.com at 4/26/2020, 3:02:21 PM on My Cluster

Cmd 13

In case you mounted this bucket earlier, you might need to unmount it.

Cmd 14

```
try:
  dbutils.fs.unmount(mountPoint) # Use this to unmount as needed
except:
  print("{} already unmounted".format(mountPoint))
```

Next, explore the mount using `%fs ls` and the name of the mount.

Remember, your mount name includes your email address so you will need to uncomment and update the following FILL_IN section

Cmd 18

```
print("Hint: Your mount name is {}".format(mountPoint))
```

```
Hint: Your mount name is /mnt/etlp1s-huseyinyilmaz01@gmail.com-si
```

Command took 0.03 seconds -- by huseyinyilmaz01@gmail.com at 4/26/2020, 3:08:58 PM on My Cluster

Cmd 19

```
%python
%fs ls /mnt/etlp1-huseyinyilmaz01@gmail.com-si
```

```
UsageError: Line magic function `%fs` not found.
```

Command took 0.04 seconds -- by huseyinyilmaz01@gmail.com at 4/26/2020, 3:09:39 PM on My Cluster

Cmd 20

In practice, always secure your AWS credentials. Do this by either maintaining a single notebook with restricted permissions that holds AWS keys, or delete the cells or notebooks that expose the keys. After a cell used to mount a bucket is run, access this mount in any notebook, any cluster, and share the mount between colleagues.

Adding Options

When you import that data into a cluster, you can add options based on the specific characteristics of the data.

```
%fs head /mnt/training/Chicago-Crimes-2018.csv
```

```
[Truncated to first 65536 bytes]
ID      Case Number     Date    Block   IUCR    Primary Type    Description     Location Description    Arrest  Domestic        Beat    District        Ward    Commu
nity Area       FBI Code        X Coordinate    Y Coordinate    Year    Updated On      Latitude        Longitude       Location
23811   JB141441        02/05/2018 01:10:00 AM  118XX S INDIANA AVE     0110    HOMICIDE        FIRST DEGREE MURDER     VACANT LOT      false   false   0532    005
9       53      01A     1179707 1826280 2018    02/12/2018 03:49:14 PM  41.678585145    -87.617837834   (41.678585145, -87.617837834)
11228589        JB148990        01/23/2018 09:00:00 AM  072XX S VERNON AVE      1153    DECEPTIVE PRACTICE      FINANCIAL IDENTITY THEFT OVER $ 300     OTHER   false
false   0323    003     6       69      11                      2018    02/12/2018 03:49:14 PM
11228563        JB148931        01/31/2018 10:12:00 AM  040XX N KEYSTONE AVE    1154    DECEPTIVE PRACTICE      FINANCIAL IDENTITY THEFT $300 AND UNDER APARTMENT
false   false   1722    017     39      16      11                      2018    02/12/2018 03:49:14 PM
11228555        JB148885        02/01/2018 02:00:00 PM  017XX W CONGRESS PKWY   0820    THEFT   $500 AND UNDER  HOSPITAL BUILDING/GROUNDS      false   false   1231
012     2       28      06                      2018    02/12/2018 03:49:14 PM
11228430        JB148675        01/27/2018 09:00:00 PM  061XX S EBERHART AVE    0560    ASSAULT SIMPLE  RESIDENCE       false   true    0313    003     20      42
08A                     2018    02/12/2018 03:49:14 PM
11228401        JB148683        02/02/2018 12:00:00 PM  038XX N SAWYER AVE      1153    DECEPTIVE PRACTICE      FINANCIAL IDENTITY THEFT OVER $ 300     RESIDENCE
false   false   1733    017     33      16      11                      2018    02/12/2018 03:49:14 PM
11228347        JB148599        01/28/2018 07:00:00 PM  008XX E 45TH ST 0620    BURGLARY        UNLAWFUL ENTRY  RESIDENCE       false   false   0221    002     4
39      05                      2018    02/12/2018 03:49:14 PM
11228291        JB148591        01/10/2018 04:45:00 PM  010XX E 53RD ST 1153    DECEPTIVE PRACTICE      FINANCIAL IDENTITY THEFT OVER $ 300             false   false
0233    002     4       41      11                      2018    02/12/2018 03:49:14 PM
11228287        JB148482        01/03/2018 03:45:00 PM  0000X W C1 ST   0810    THEFT   OVER $500       AIRPORT TERMINAL LOWER LEVEL - NON-SECURE AREA  false   false
1651    016     41      76      06                      2018    02/12/2018 03:49:14 PM
```

Command took 1.32 seconds -- by huseyinyilmaz01@gmail.com at 4/26/2020, 3:28:46 PM on My Cluster

`option` is a method of `DataFrameReader`. Options are key/value pairs and must be specified before calling `.csv()`.

This is a tab-delimited file, as seen in the previous cell. Specify the `"delimiter"` option in the import statement.

:NOTE: Find a full list of parameters here.

Cmd 26

```
display(spark.read
  .option("delimiter", "\t")
  .csv("/mnt/training/Chicago-Crimes-2018.csv")
)
```

▸ (2) Spark Jobs

_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9
ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Dor
23811	JB141441	02/05/2018 01:10:00 AM	118XX S INDIANA AVE	0110	HOMICIDE	FIRST DEGREE MURDER	VACANT LOT	false	fals
11228589	JB148990	01/23/2018 09:00:00 AM	072XX S VERNON AVE	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	OTHER	false	fals
11228563	JB148931	01/31/2018 10:12:00	040XX N KEYSTONE AVE	1154	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT $300 AND	APARTMENT	false	fals

Showing the first 1000 rows.

Spark doesn't read the header by default, as demonstrated by the column names of `_c0`, `_c1`, etc. Notice that the column names are present in the first row of the DataFrame.

Fix this by setting the `"header"` option to `True`.

Cmd 28

```
display(spark.read
  .option("delimiter", "\t")
  .option("header", True)
  .csv("/mnt/training/Chicago-Crimes-2018.csv")
)
```

(2) Spark Jobs

ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	Beat	District	Ward	Community Area
23811	JB141441	02/05/2018 01:10:00 AM	118XX S INDIANA AVE	0110	HOMICIDE	FIRST DEGREE MURDER	VACANT LOT	false	false	0532	005	9	53
11228589	JB148990	01/23/2018 09:00:00 AM	072XX S VERNON AVE	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	OTHER	false	false	0323	003	6	69
11228563	JB148931	01/31/2018 10:12:00 AM	040XX N KEYSTONE AVE	1154	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT $300 AND UNDER	APARTMENT	false	false	1722	017	39	16

Showing the first 1000 rows.

Spark didn't infer the schema, or read the timestamp format, since this file uses an atypical timestamp. Change that by adding the option `"timestampFormat"` and pass it the format used in this file.

Set `"inferSchema"` to `True`, which triggers Spark to make an extra pass over the data to infer the schema.

Cmd 30

```
crimeDF = (spark.read
  .option("delimiter", "\t")
  .option("header", True)
  .option("timestampFormat", "mm/dd/yyyy hh:mm:ss a")
  .option("inferSchema", True)
  .csv("/mnt/training/Chicago-Crimes-2018.csv")
)
display(crimeDF)
```

(3) Spark Jobs

crimeDF: pyspark.sql.dataframe.DataFrame = [ID: integer, Case Number: string ... 20 more fields]

ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	Beat	District	Ward	Comm Area
23811	JB141441	2018-01-05T01:10:00.000+0000	118XX S INDIANA AVE	0110	HOMICIDE	FIRST DEGREE MURDER	VACANT LOT	false	false	532	5	9	53
11228589	JB148990	2018-01-23T09:00:00.000+0000	072XX S VERNON AVE	1153	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT OVER $ 300	OTHER	false	false	323	3	6	69
11228563	JB148931	2018-01-31T10:12:00.000+0000	040XX N KEYSTONE AVE	1154	DECEPTIVE PRACTICE	FINANCIAL IDENTITY THEFT $300 AND UNDER	APARTMENT	false	false	1722	17	39	16
11228555	JB148885	2018-01-01T14:00:00.000+0000	017XX W CONGRESS	0820	THEFT	$500 AND UNDER	HOSPITAL BUILDING/GROUNDS	false	false	1231	12	2	28

Showing the first 1000 rows.

Cmd 31

The Design Pattern

Other connections work in much the same way, whether your data sits in Cassandra, Redis, Redshift, or another common data store. The general pattern is always:

1. Define the connection point
2. Define connection parameters such as access credentials
3. Add necessary options

After adhering to this, read data using `spark.read.options(<option key>, <option value>).<connection_type>(<endpoint>)` .

Exercise 1: Read Wikipedia Data

Read Wikipedia data from S3, accounting for its delimiter and header.

Cmd 33

Step 1: Get a Sense for the Data

Take a look at the head of the data, located at `/mnt/training/wikipedia/pageviews/pageviews_by_second.tsv`.

Cmd 34

```
# TODO
display(spark.read
  .option("delimiter", "\t")
  .option("header", True)
  .csv("/mnt/training/wikipedia/pageviews/pageviews_by_second.tsv")
  .limit(3)
)
```

(2) Spark Jobs

timestamp	site	requests
2015-03-16T00:09:55	mobile	1595
2015-03-16T00:10:39	mobile	1544
2015-03-16T00:19:39	desktop	2460

Step 2: Import the Raw Data

Import the data **without any options** and save it to `wikiDF`. Display the result.

Cmd 36

```
# TODO
wikiDF = (spark.read
  .csv("/mnt/training/wikipedia/pageviews/pageviews_by_second.tsv")
)
```

▸ (1) Spark Jobs

▸ wikiDF: pyspark.sql.dataframe.DataFrame = [_c0: string]

Command took 1.01 seconds -- by huseyinyilmaz01@gmail.com at 4/26/2020, 4:17:08 PM on My Cluster

Cmd 37

Step 3: Import the Data with Options

Import the data with options and save it to `wikiWithOptionsDF`. Display the result. Your import statement should account for:

- The header
- The delimiter

Cmd 39

```
# TODO
wikiWithOptionsDF = (spark.read
  .option("delimiter", "\t")
  .option("header", True)
  .option("timestampFormat", "mm/dd/yyyy hh:mm:ss a")
  .option("inferSchema", True)
  .csv("/mnt/training/wikipedia/pageviews/pageviews_by_second.tsv")
)
display(wikiWithOptionsDF)
```

(3) Spark Jobs

wikiWithOptionsDF: pyspark.sql.dataframe.DataFrame = [timestamp: timestamp, site: string ... 1 more fields]

timestamp	site	requests
2015-03-16T00:09:55.000+0000	mobile	1595
2015-03-16T00:10:39.000+0000	mobile	1544
2015-03-16T00:19:39.000+0000	desktop	2460

Review

Question: What accounts for Spark's quick rise in popularity as an ETL tool?
Answer: Spark easily accesses data virtually anywhere it lives, and the scalable framework lowers the difficulties in building connectors to access data. Spark offers a unified API for connecting to data making reads from a CSV file, JSON data, or a database, to provide a few examples, nearly identical. This allows developers to focus on writing their code rather than writing connectors.

Question: What is DBFS and why is it important?
Answer: The Databricks File System (DBFS) allows access to scalable, fast, and distributed storage backed by S3 or the Azure Blob Store.

Question: How do you connect your Spark cluster to S3?
Answer: By mounting it. Mounts require AWS credentials and give access to a virtually infinite store for your data. Using AWS IAM roles provides added security since your keys will not appear in log files. One other option is to define your keys in a single notebook that only you have permission to access. Click the arrow next to a notebook in the Workspace tab to define access permissions.

Question: How do you specify parameters when reading data?
Answer: Using `.option()` during your read allows you to pass key/value pairs specifying aspects of your read. For instance, options for reading CSV data include `header`, `delimiter`, and `inferSchema`.

Question: What is the general design pattern for connecting to your data?
Answer: The general design pattern is as follows:

1. Define the connection point.
2. Define connection parameters such as access credentials.
3. Add necessary options such as for headers or parallelization.