

# DB Management Systems

## Distributed: Spark

Joel Klein – [jdk514@gwmail.gwu.edu](mailto:jdk514@gwmail.gwu.edu)

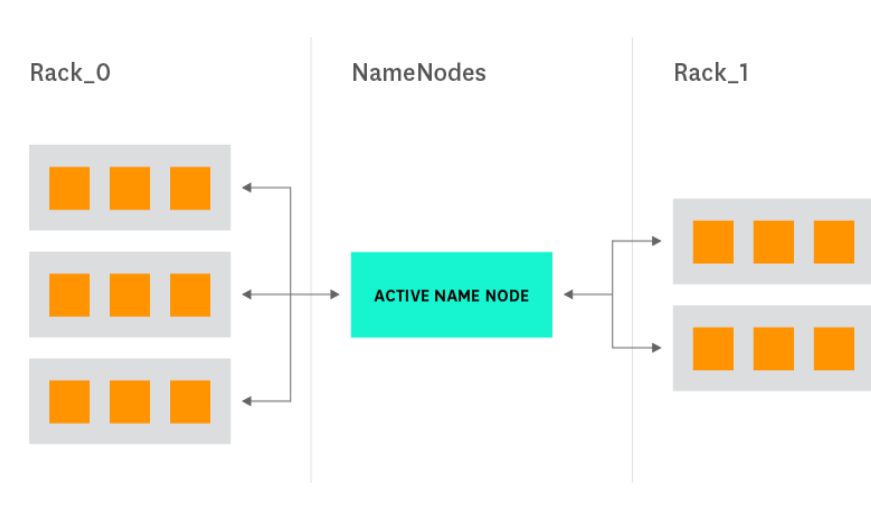
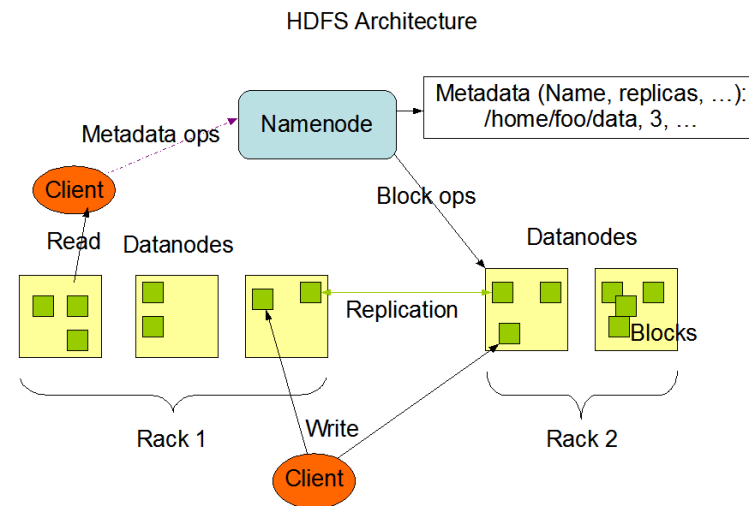
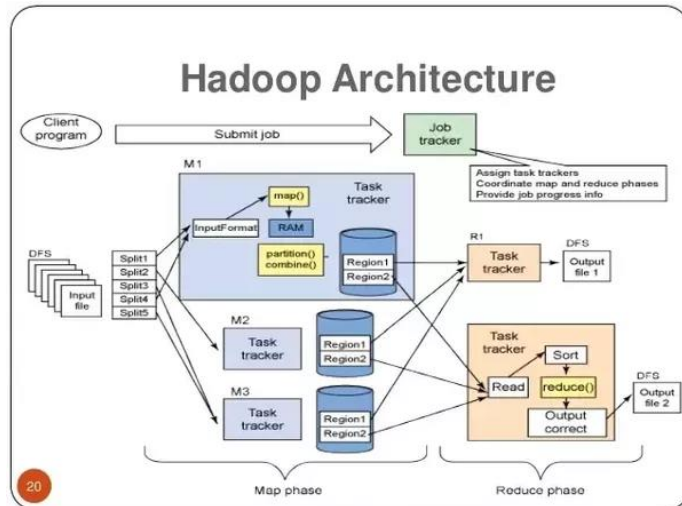
# Spark Architecture

# Spark? Architectural Diagram

Complex

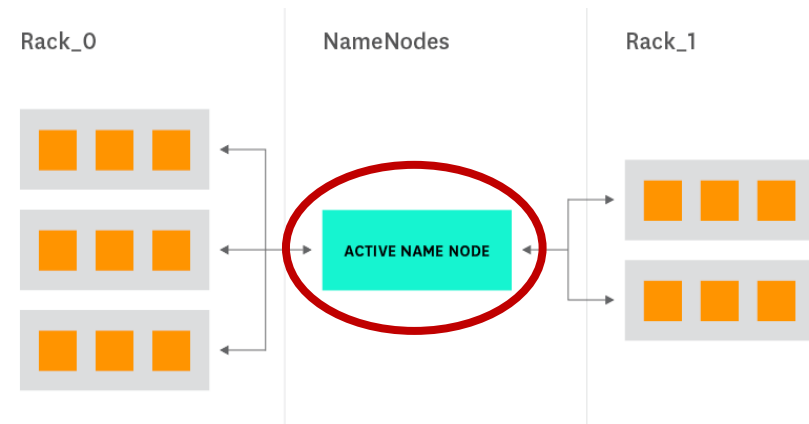


Simple

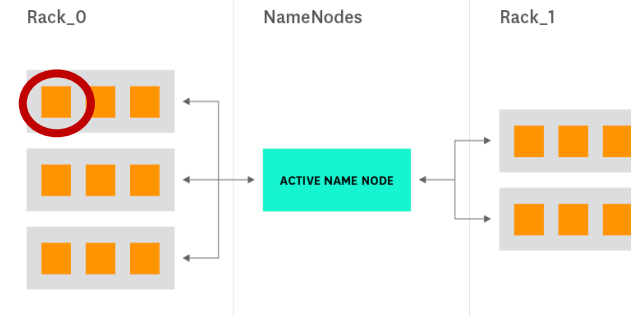


# NameNode (Spark Driver)

- Think of this as the master node.
- It is the node a user actually interfaces with to input commands/configuration
- It doesn't need to necessarily be a crazy machine (specs wise) as it doesn't do any of the heavy lifting



# DataNode (Worker/Executor)



- DataNode's represent the true brute force behind Spark.
- They are the nodes that both store and process (in memory) the data used by Sparks acyclic graph transformations
  - Executors can load data through a number of different sources
    - S3, FTP, local filesystem, Azure, Swift(??)
  - Executors load the data into memory, rather than locally on disk
- These machines are usually beefier than the NameNode, as they need to be able to process large amounts of data quickly
  - These machines typically need more RAM since spark operates entirely in memory

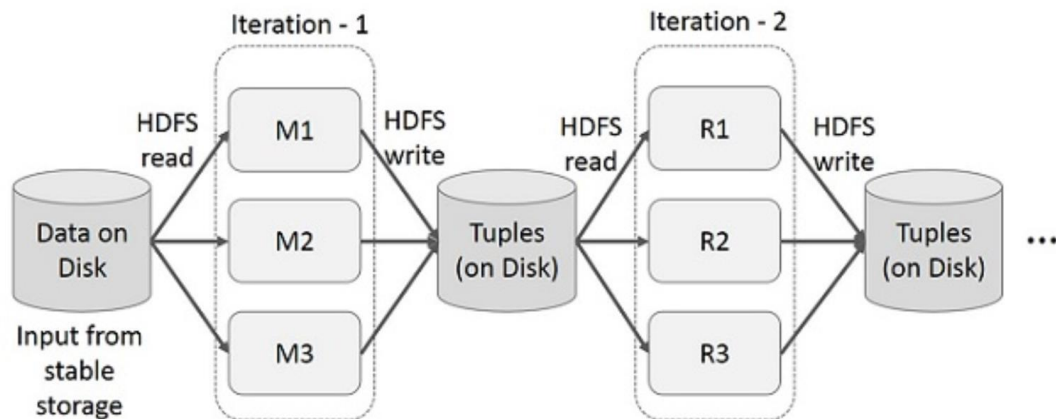
# Resilient Distributed Dataset

- RDD's are essentially like Series in Spark, designed for parallelized and distributed operations.
  - They are a collection of objects storing data
  - Some key traits are:
    - Immutability – once created they can not be changed
    - Partitioned – the dataframes are split amongst a number of executors
    - Typed – each record is statically typed (RDD[Long], RDD[String, Int, etc.] )
- RDD's can only be manipulated via transformations
  - **map**, flatmap, **filter**, **reduceByKey**, join, cogroup

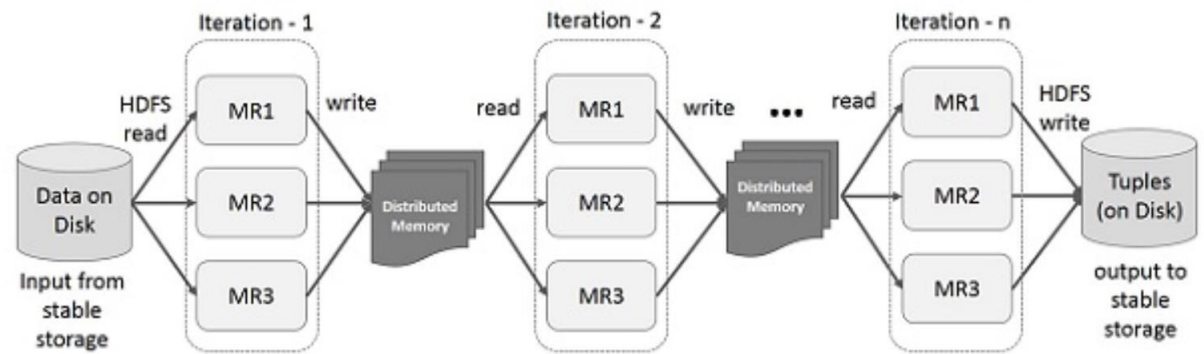
# Directed Acyclic Graph (DAG)

- DAG is Sparks version of MapReduce. It is essentially a more verbose implementation.
  - Essentially it is the process of altering an RDD through a series of transformations (as mentioned earlier)

**Hadoop MapReduce**



**Spark DAG**



## MapReduce

- Always two operations in order
  1. Map
  2. Reduce
- Information is read from disk and saved out to disk during each intermediary step
- Great for batch processing

## DAG

- Can perform any combination of transformations between RDD states
  - MapReduce can be one of these transformation processes
- Initial data is read from disk, but all further transformations take place solely in-memory
- Great for real-time processing and fast iterative processes (training ML algorithms)



# Programming in Spark

- One of the main reasons for the recent surge in Spark is due to its ease of use through flexible API's
- These API's have SDKs for the following programming languages (there may be more):
  - Scala
  - Java
  - Python
  - SQL
  - R

# Spark on AWS

# EMR (Elastic Map Reduce)

The screenshot shows the Amazon EMR console interface. At the top, there is a dark navigation bar with the AWS logo, 'Services', 'Resource Groups', and a search icon. On the left, a sidebar lists navigation options: 'Amazon EMR', 'Clusters' (highlighted with an orange bar), 'Security configurations', 'VPC subnets', 'Events', and 'Help'. The main content area has a heading 'Welcome to Amazon Elastic MapReduce' followed by a paragraph: 'Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.' Below this is the text 'You do not appear to have any clusters. Create one now:' and a blue 'Create cluster' button. A section titled 'How Elastic MapReduce Works' contains three steps: 'Upload' (with a cloud and upload icon), 'Create' (with a cluster diagram and gear icon), and 'Monitor' (with a monitor and download icon). Each step includes a brief description and a 'Learn more' link.

aws Services Resource Groups

Amazon EMR  
Clusters  
Security configurations  
VPC subnets  
Events  
Help

## Welcome to Amazon Elastic MapReduce


Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

You do not appear to have any clusters. Create one now:

[Create cluster](#)

### How Elastic MapReduce Works


**Upload**



Upload your data and processing application to S3.

[Learn more](#)


**Create**



Configure and create your cluster by specifying data inputs, outputs, cluster size, security settings, etc.

[Learn more](#)

**Monitor**



Monitor the health and progress of your cluster. Retrieve the output in S3.

[Learn more](#)

# Enabling EMR Notebooks

- Normally we could simply instantiate the spark cluster using a default software configuration

## Software configuration

Release  ⓘ

Applications

- Core Hadoop: Hadoop 2.10.1, Hive 2.3.7, Hue 4.8.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
- HBase: HBase 1.4.13, Hadoop 2.10.1, Hive 2.3.7, Hue 4.8.0, Phoenix 4.14.3, and ZooKeeper 3.4.14
- Presto: Presto 0.240.1 with Hadoop 2.10.1 HDFS and Hive 2.3.7 Metastore
- Spark: Spark 2.4.7 on Hadoop 2.10.1 YARN and Zeppelin 0.8.2

Use AWS Glue Data Catalog for table metadata ⓘ

## Create Cluster - Quick Options [Go to advanced options](#)

- However, to make things easier for us, we're going to setup Spark so it can connect to an AWS hosted Jupyter notebook or EMR Notebook

## Software Configuration

Release  ⓘ

<input checked="" type="checkbox"/> Hadoop 2.10.1	<input type="checkbox"/> Zeppelin 0.8.2
<input type="checkbox"/> JupyterHub 1.1.0	<input type="checkbox"/> Tez 0.9.2
<input type="checkbox"/> Ganglia 3.7.2	<input type="checkbox"/> HBase 1.4.13

# Custom Configuration Software

- For **EMR Notebooks** to work we need to configure a custom selection of software for things to work.
  - We can also enable **Hive** so we can take a second look at it's use cases

### Software Configuration

Release  ⓘ

<input checked="" type="checkbox"/> Hadoop 2.10.1	<input type="checkbox"/> Zeppelin 0.8.2	<input checked="" type="checkbox"/> Livy 0.7.0
<input checked="" type="checkbox"/> JupyterHub 1.1.0	<input type="checkbox"/> Tez 0.9.2	<input type="checkbox"/> Flink 1.11.2
<input type="checkbox"/> Ganglia 3.7.2	<input type="checkbox"/> HBase 1.4.13	<input checked="" type="checkbox"/> Pig 0.17.0
<input checked="" type="checkbox"/> Hive 2.3.7	<input type="checkbox"/> Presto 0.240.1	<input type="checkbox"/> ZooKeeper 3.4.14
<input checked="" type="checkbox"/> JupyterEnterpriseGateway 2.1.0	<input type="checkbox"/> MXNet 1.7.0	<input type="checkbox"/> Sqoop 1.4.7
<input type="checkbox"/> Mahout 0.13.0	<input checked="" type="checkbox"/> Hue 4.8.0	<input type="checkbox"/> Phoenix 4.14.3
<input type="checkbox"/> Oozie 5.2.0	<input checked="" type="checkbox"/> Spark 2.4.7	<input type="checkbox"/> HCatalog 2.3.7
<input type="checkbox"/> TensorFlow 2.3.1		

# Reason for Custom Configuration

- There are a couple of reasons why we need to modify the configuration
  - Adding JupyterHub and JupyterEnterpriseGateway enable us to connect via JupyterNotebooks
    - JupyterHub is a webserver-based version of Jupyter
    - JupyterEnterpriseGateway enables JupyterHub to spin up notebooks on a cluster of machines
  - Adding Livy provides a REST driven API for spark that makes handling concurrent asynchronous requests easier (basically makes the notebook interactions work from the EMR perspective)

# Creating an EMR Cluster cont.

- Defining Security:
  - We'll still setup the EC2 key pair, so we can connect directly to the master node
- Waiting:
  - This shouldn't take any longer than last time, but we still do need to wait

## Security and access

EC2 key pair **GW Course Key** [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](#) ⓘ

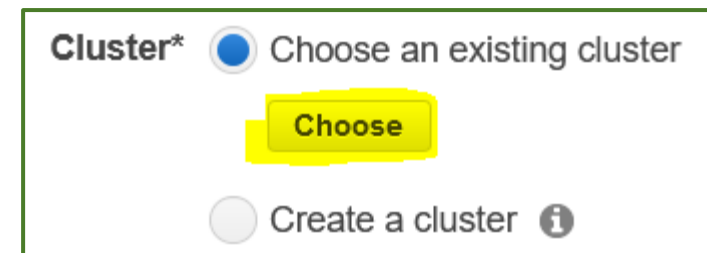
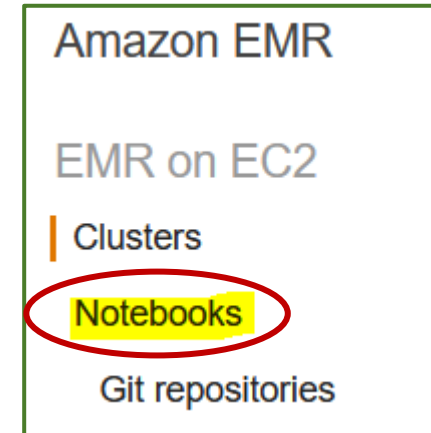
EC2 instance profile [EMR\\_EC2\\_DefaultRole](#) ⓘ

# Accessing Our EMR



# EMR Notebooks

- From the EMR service page, there is an option to setup an **EMR Notebook**, we'll be using that to connect to our cluster
- From here we can [Create notebook](#)
- The only thing we need to define (beyond providing a name), is the existing cluster for the notebook to connect

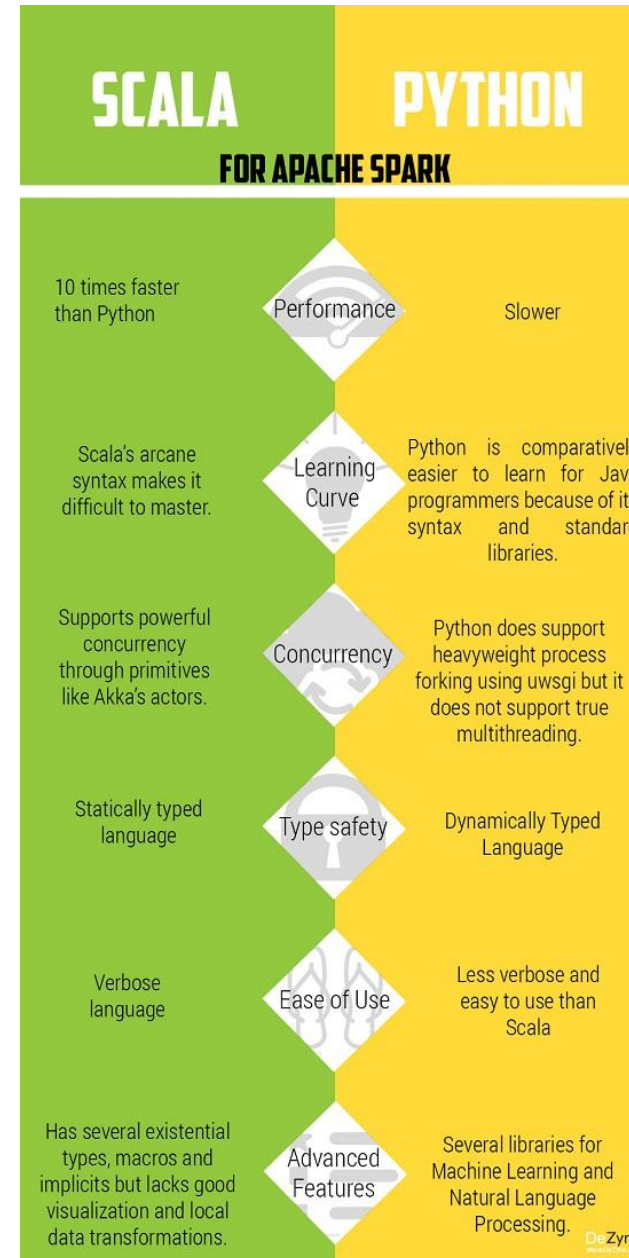


# PySpark

# PySpark

- PySpark is the name of the python specific interface for python programming in Spark
- One of its main benefits is ability to seamlessly leverage python for distributed programming
- PySpark also heavily utilizes Dataframes an abstraction of RDD's
  - PySpark Dataframes work a lot like pandas dataframes, but have some unique interfaces

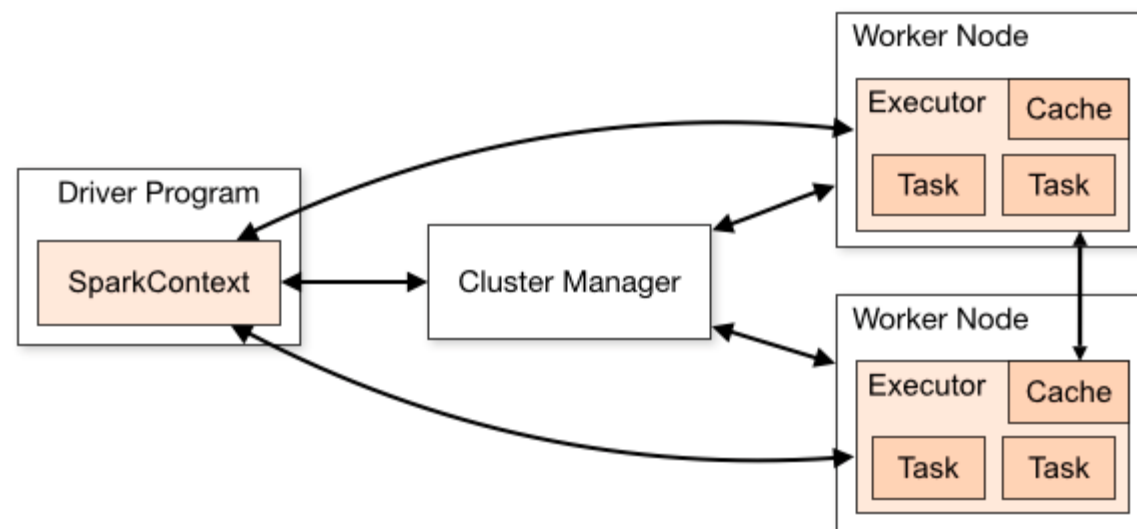
# Python vs Scala



# Working with PySpark

# Spark Session

- The **spark session** is the primary interface for working with Spark's API
  - This means that we need an active spark session to do anything within spark
  - Creating the EMR Notebooks provides us an easy mechanism to hook into the spark session



Source: [hadoopsters.com/2020/10/26/spark-starter-guide-4-2-how-to-create-a-spark-session/](https://hadoopsters.com/2020/10/26/spark-starter-guide-4-2-how-to-create-a-spark-session/)

# Working with a Spark Session

- Normally we need to build a spark session (which can require some complex networking for custom environments)
  - `spark = SparkSession.builder.appName("Python Spark SQL basic example").config("spark.some.config.option", "some-value").getOrCreate()`
  - With the **EMR Notebook** we can simply call spark directly

# Creating a spark Dataframe

- To create a spark Dataframe we need to tell spark to read in data from a data source. This can be accomplished through several different commands, like Hadoop:
- For our case, we'll read in the same data from s3 from last class:
  - `spark.read.json("s3a://{s3_bucket}/ratebeer/*.json")`
- Most datawarehousing applications use file types like parquet and avro



# Alternative File Formats - Parquet and Avro

- Parquet and Avro are file types that emulate many of the compression techniques provided by databases and archiving (think .zip or .targz)
  - This means that unlike json and csv, these file types are focused around data optimization
- Parquet is a columnar storage format, meaning that it focuses on storing columns of data (compared to csv which is row focused)
- Avro is essentially a serialized version of JSON and has similarities to protobuf
  - Field names are abstracted away and data is stored in binary formatting

# PySpark Lazy Evaluation

- RDD's, and thus Dataframes, are lazily evaluated. This means that transformations on the dataframe won't be evaluated until actually requested.
- This also means that only necessary transformations actually occur (to an extent). This is part of the advantage to columnar data, as the entire file/dataset may not need to be processed, but only a single column.

# PySpark Streaming and SQL API

# SQL in PySpark

- In addition to working with data loaded straight into a dataframe, we can also query our data leveraging SQL through Spark's SQL API
- To accomplish this we need to simply create a temporary view of our data that we can query against
  - `df.createOrReplaceTempView("beer_table")`
- Now all we need to do is create a new Dataframe by querying our existing information
  - `sql_table = spark.sql("SELECT * from beer_table LIMIT 10")`

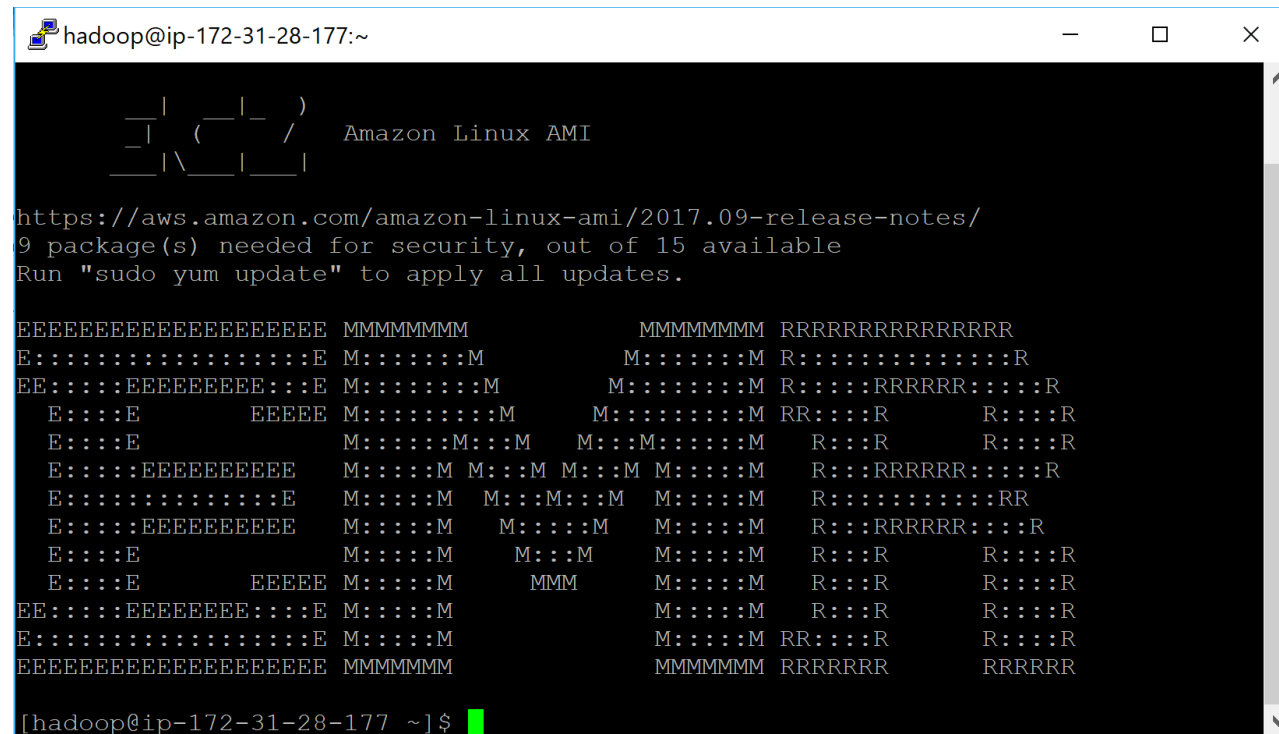
End Slide

EMSE 6992 – DBMS for Data Analytics

# PySpark Terminal (If Using SSH)

# Once Connected Via SSH

- If everything was done correctly, we should be met with the following:



```
hadoop@ip-172-31-28-177:~  
  
  _ |  _ |  )  
  _ | (  _ | /  Amazon Linux AMI  
  _ | \  _ |  _ |  
  
https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/  
9 package(s) needed for security, out of 15 available  
Run "sudo yum update" to apply all updates.  
  
EEEEEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMM RRRRRRRRRRRRRRRR  
E::::::::::::::::::E M:::::M      M:::::M R:::::R  
EE:::::EEEEEEEE:::E M:::::M      M:::::M R:::::RRRRRR:::R  
  E::::E      EEEEE M:::::M      M:::::M RR::::R      R::::R  
  E::::E      M:::::M:::M      M:::M:::::M      R::::R      R::::R  
  E:::::EEEEEEEEEE M:::::M M:::M M:::M M:::::M      R:::RRRRR:::R  
  E:::::EEEEEEEEEE M:::::M M:::::M M:::::M      R:::::RRRRR:::R  
  E::::E      M:::::M      M:::M      M:::::M      R::::R      R::::R  
  E::::E      EEEEE M:::::M      MMM      M:::::M      R::::R      R::::R  
EE:::::EEEEEEEE:::E M:::::M      M:::::M      R::::R      R::::R  
E:::::EEEEEEEEEE M:::::M      M:::::M RR::::R      R::::R  
EEEEEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMM RRRRRRR      RRRRRR  
  
[hadoop@ip-172-31-28-177 ~]$
```





# Spark REPL

- REPL stands for Read-Eval-Print-Loop
- The pyspark repl is effectively a python console
  - This means that all generic python will run in the pyspark repl
- The key difference between pyspark repl and a generic python console is the ability to leverage DAG operations to process RDD's
  - You can see when spark is being leveraged when you see the following in the console:

```
+-----+-----+
|summary|          overall|
+-----+-----+
|  count|          824|
|   mean|13.445388349514563|
| stddev| 2.156552781084682|
|    min|           4.0|
|    max|          20.0|
+-----+-----+

>>> df.select('overall').describe().show()
[Stage 4:=====>
```

(11 + 16) / 32