```
----------------------------------------------
What is Big Data?
 --> 3Vs of Big Data
         --> Velocity
         --> Variety
         --> Volume
 ----------------------------------------------

----------------------------------------------
What is Hadoop?
 --> Reliable, Scalable distributed computing framework
 --> Big Data Analytics Platform
----------------------------------------------

----------------------------------------------
Hadoop Roles

Hadoop Developer and Analyst
 - Pre requisites: Programming skills (Java | Python | Scala | SQL)
 - Hadoop Tools: MapReduce, Pig, Hive, Impala, Spark etc.

Hadoop Administrator
 - Pre requisites: Admin skills (Linux | DBA)
 - Hadoop Tools: Ambari, Cloudera Manager, ZooKeeper, Sentry etc.

Data Scientist
 - Pre requisites: Statistics, Machine Learning, Python | R
 - Hadoop Tools: Hive, Impala, Spark ML-Lib, Mahout etc.


Data Migration
 - Hadoop Tools: Sqoop, Flume, Kafka etc.
----------------------------------------------

----------------------------------------------
Generations of Hadoop

Hadoop 1.x --> Old
Hadoop 2.x --> Production
Hadoop 3.x --> New
----------------------------------------------

----------------------------------------------
Gen 2 Hadoop = FsShell + HDFS + MapReduce + YARN

FsShell - Let's a user interact with HDFS

HDFS - Distributed File System

MapReduce - Programming model for parallel processing

YARN - MapReduce Engine V2, for cluster resource management
----------------------------------------------



                https://goo.gl/n2zLTW
```

```
---------------------------------------------
Create a shared folder

$ sudo mount -t vboxsf -o uid=501,gid=501 Labs Downloads/
---------------------------------------------

---------------------------------------------
Hadoop is a master-slave architecture

Hadoop Daemons

 HDFS
        - NameNode (master)
        - DataNode(s)   (slave)
        - SecondaryNameNode (master) --> out of date --> Checkpoint node

 YARN
        - ResourceManager (master)
        - NodeManager(s)   (slave)
---------------------------------------------

---------------------------------------------
Note: 1 master / cluster

Note: HDFS is immutable (Write Once Read Many - WORM)
---------------------------------------------




---------------------------------------------
MapReduce - Programming model for parallel processing
          - Involves 2 phases
                - Map phase
                - Reduce phase

          - Other complex things are abstracted for the developer

Overall steps in MapReduce
 - Input Split
 - Map
 - Shuffle & Sort
 - Reduce
 - Final Output


Understanding MapReduce with an example

Problem Statement: Compute maximum close price / stock symbol

Mantra
 - Map --> Transformation
 - Reduce --> Aggregation



MapReduce --> <k, v>

Overall Logic (MapReduce way)
 - Read each line
 - Split based on delimiter ","
        - mark 2nd column as key (stock_symbol), and 7th column as the value (close_price)
 - Group by key
 - Sort by key
         - For each key, a list of values will be prepared
 - For each key, pick the maximum value of close_price
 - Write the output
```

```
Overall Logic (MapReduce way)

Input Split
 - Read each line

Map
 - Split based on delimiter ","
        - mark 2nd column as key (stock_symbol), and 7th column as the value (close_price)

Shuffle & Sort
 - Group by key
 - Sort by key
         - For each key, a list of values will be prepared

Reduce
 - For each key, pick the maximum value of close_price

Final Output
 - Write the output
----------------------------------------------



----------------------------------------------
Each map task generates output - Intermediate results - These are stored on the local file system of the node
where the map task was running
----------------------------------------------



----------------------------------------------
YARN Glossary

 - ResourceManager (master)
        - 2 components
                - ApplicationsManager
                - Scheduler

 - NodeManager (slave) - 1 / DN

 - Container - Compute Resource - <CPU+RAM>
        - map and reduce tasks are the actual units of execution
        - containers will be allocated for task execution
        - a container for map/reduce task is called YarnChild

 - ApplicationMaster
        - first container for the job
        - negotiate for containers
        - monitor them once allocated
----------------------------------------------



----------------------------------------------
```

```
---------------------------------------------
Understanding MapReduce with another example

Problem Statement: WordCount --> Count each word

Word,Count

Input Set

Welcome to Hadoop
Learning Hadoop is fun
Hadoop Hadoop Hadoop is the buzz

Expected Output?

Hadoop 5
Learning 1
Welcome 1
buzz 1
fun 1
is 2
the 1
to 1

Overall Logic for Word,Count (MapReduce way)

Input Split
 - Read each line

Map
 - Split based on delimiter " "
       - mark each word as the key, assign a value 1

Shuffle & Sort
 - Group by key
 - Sort by key
        - For each key, a list of values will be prepared

Reduce
 - For each key, sum the values

Final Output
 - Write the output




Mantra
 - Map --> Transformation --> Convert into words
```

```
 - Reduce --> Aggregation --> Count the words
----------------------------------------------
```

```
----------------------------------------------
Hive is a SQL interface for Hadoop datasets

Hive is not RDBMS

Schema on Read: Structure can be projected onto data already in storage
----------------------------------------------

----------------------------------------------
Sqoop --> SQL to Hadoop
 --> RDBMS to Hadoop import tool
 --> Get the data from RDBMS data sources into a Hadoop cluster
 --> Abstraction to MapReduce
----------------------------------------------

--------------------------------------------------------------------------------
Hive Datatypes
 - Hive supports most of the datatypes that were supported by RDBMS
 - 2 classifications
        - Primitive
        - Collection

 - Primitives
        - Boolean
        - Integer
                - tinyint -128 to 127
                - smallint -32768 to 32767
                - int -2^15 to 2^15-1
                - bigint -2^63 to 2^63-1
        - Decimal
                - float
                - double
                - decimal
        - String
                - string
                - char
                - varchar
        - Timestamp
                - date
                - timestamp

 - Collections
        - Array
        - Struct
        - Union
        - Map

 - Collection types can be used to store bunch of information in a single column

--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
 Hive tables
 - Hive arranges data in form of tables
 - A database in Hive is a collection of tables

 - Hive stores the table information in a metastore DB
```

```
 - The data is in HDFS
 - Hive tables are stored as directories in HDFS

 2 types of tables in Hive
 - Managed Table
        - Default behavior
        - Hive manages 'data' and 'schema'
        - When you drop a managed table, data is removed from HDFS and schema is removed from metastore

 - External Table
        - Best practice
        - Use external keyword in the table definition
        - Can also point to an existing HDFS dataset
        - When you drop an external table, only schema is removed from metastore


 - Hive also has a default warehouse directory in HDFS - /user/hive/warehouse
--------------------------------------------------------------------------------



Note: Hive is a SQL interface for Hadoop datasets, Hive is not RDBMS
--------------------------------------------------------------------------------
 Hive Vs RDBMS

- Batch processing
        - Hive can process gigabytes to petabytes of data (meant for large scale data processing)
        - Data stored here is meant for analytics
        - Since Hive is using MapReduce (under the hood), it gets the power of parallel processing
        - Hive does not go well for transactional processing (unlike RDBMS) --> Once data is written, the
purpose is read only

- Schema on Read
        - Hive does NOT validate the data against the schema while the data was loaded, instead it is
validated while we issue a select query
        - Structure can be projected on data which is already in storage (HDFS)
        - Schema (table, columns and datatypes) is not stored along with data in Hive
        - Schema is stored in a metastore DB
        - Hive does not support constraints (Primary key, Foreign key etc)
        - Delete / Updates are not supported in Hive (yet)
--------------------------------------------------------------------------------


****************************************************************************
Case Sensitivity in Pig:

--> The names (aliases) of relations and fields are case sensitive
--> The names of Pig Latin functions are case sensitive.
--> The names of parameters and all other Pig Latin keywords are case insensitive.

In the example. note the following:
--> The names (aliases) of relations A, B, and C are case sensitive.

--> The names (aliases) of fields f1, f2, and f3 are case sensitive.

--> Function names PigStorage and COUNT are case sensitive.

--> Keywords LOAD, USING, AS, GROUP, BY, FOREACH, GENERATE, STORE and DUMP are case insensitive. They can
also be written as load, using, as, group, by, etc.

--> In the FOREACH statement, the field in relation B is referred to by positional notation ($0).

****************************************************************************
```

```
--------------------------------------------------------------------------------
Pig Datatypes
 - Pig supports most of the datatypes that other programming languages support
 - 2 classifications
        - Scalar
        - Complex

 - Scalar
        - Boolean
        - Integer
                - int
                - long
                - biginteger

        - Decimal
                - float
                - double
                - bigdecimal

        - String
                - chararray

        - Timestamp
                - datetime

        - Bytearray - BLOB (Array of bytes)
                - ByteArray is the default datatype for Pig

 - Collections
        - Map
        - Tuple
        - Bag

 - Collection types can be used to store bunch of information in a single column


--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

 Pig Vs Hive
 - Pig is great to get the data which is unstructured and transform it into some structure (ETL)
 - Once the data is structured, Hive can operate on this data for reporting purposes

 - Instead of considering Pig as an alternative to Hive, a typical use case would consider Pig to complement
Hive

 - Pig is great for transformations
 - Hive is great for analytics

 Note: Pig can also do analytics, however this is a rare use case

 - Pig does not require data to be loaded as a table
 - Pig is a procedural data flow language

 - Pig has an interpreter and can understand only 1 instruction at a time

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------

Impala Vs Hive
- Like Hive, Impala allows users to query data in HDFS using an SQL-like language
- Unlike Hive, Impala does not turn queries into MapReduce jobs
- Impala returns results typically within seconds or a few minutes, rather than the many minutes or hours
  that are often required for Hive queries to complete

--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Spark
 - Spark is a general purpose, large scale, in-memory, distributed, unified data processing engine

 - Spark jobs can be written in Scala | Python | R | Java | SQL

 - Spark can read data from variety of data sources like HDFS, Cassandra, S3, HBase, local file system etc.

 - Spark can use its own cluster manager | YARN | Mesos



RDBMS Vs Hadoop MapReduce
 - With Hadoop MapReduce, process data 10x faster than RDBMS

Hadoop MapReduce Vs Spark
 - With Spark, process data 10x faster than Hadoop MapReduce


What is Scala?
 - Programming Language
 - Object Oriented Language
 - Functional Programming
 - Statically typed
 - Runs in a JVM

REPL --> Interactive Shell
     --> Read, Evaluate, Print and Loop


RDD - Resilient Distributed Dataset
    - Progamming abstraction for Spark
    - in memory objects that can be operated on in parallel
    - immutable


RDD Operations (2 types)
 - Transformations (Lazily evaluated, no data processing)
 - Actions (A job is triggered)



Let us read data from HDFS into an RDD
scala> val baseRDD = sc.textFile("/Sample")

sc --> Spark Context --> Connection to Spark cluster


Transformations
 - map(func)
 - flatMap(func)
 - reduceByKey(func, numtasks)
 - sortBy()


Actions
 - collect()
 - count()
 - first()
 - take(n)
 - saveAsTextFile(<Path>)
```

```
-------------------------------------------------------------------------------
WordCount using Spark and Scala


Overall Logic for Word,Count (MapReduce way)

 - Read each line
 - Split based on delimiter " "
 - mark each word as the key, assign a value 1
 - Group by key
 - For each key, sum the values
 - Write the output


scala> val baseRDD = sc.textFile("/Sample")

scala> baseRDD.collect()

scala> baseRDD.first()

scala> val mapRDD = baseRDD.map(str => str.split(" "))

scala> mapRDD.collect()

scala> val fmapRDD = baseRDD.flatMap(str => str.split(" "))

scala> fmapRDD.collect()

scala> val mapRDD = fmapRDD.map(str => (str, 1))

scala> mapRDD.collect()

scala> val countRDD = mapRDD.reduceByKey((sum, index) => sum + index)

scala> countRDD.collect()


Note: An RDD with <k, v> pair is called 'Pair RDD' or 'Paired RDD'

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
WordCount final code - Spark and Scala

scala> val baseRDD = sc.textFile("/Sample")

scala> val fmapRDD = baseRDD.flatMap(line => line.split(" "))

scala> val mapRDD = fmapRDD.map(word => (word, 1))

scala> val countRDD = mapRDD.reduceByKey((sum, index) => sum + index)

scala> countRDD.collect()
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------

scala> val countRDD = sc.textFile("/Sample").flatMap(l => l.split(" ")).map(w => (w, 1)).reduceByKey((s,i) =>
s+i)

scala> countRDD.collect()
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------

scala> sc.textFile("/Sample").flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).collect()

-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
WordCount final code - Spark and Python

>>> baseRDD = sc.textFile("/Sample")

>>> fmapRDD = baseRDD.flatMap(lambda line : line.split(" "))

>>> mapRDD = fmapRDD.map(lambda word : (word, 1))

>>> countRDD = mapRDD.reduceByKey(lambda sum, index : sum + index)

>>> countRDD.collect()
-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

RDD Dependency
        - Narrow
        - Wide


-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

Spark Architecture

http://spark.apache.org/docs/1.6.0/cluster-overview.html


Spark Glossary
 - Application
 - Jobs
 - Spark Context
 - Driver Program
 - Cluster Manager
 - Executor
 - Stages
 - Tasks
 - Worker
 - Cache

-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
val x= sc.parallelize(Array("b", "a", "c"))
val y= x.map(z => (z,1))
y.collect()
-------------------------------------------------------------------------------
```

Further Reading:

https://blog.cloudera.com/blog/2014/08/improving-query-performance-using-partitioning-in-apache-hive/

http://www.codecommit.com/blog/scala/quick-explanation-of-scalas-syntax

Known knowns
Known unknowns --> SQL - Analysts
Unknown unknowns --> Python | R --> Data Scientist

Big Data --> Java | C++ --> Hadoop

DataFrames

```
scala> val ordersRDD = sc.textFile("/orders")

scala> val order = ordersRDD.first()

scala> order.split(",")

scala> order.split(",")(0).toInt

scala> order.split(",")(1)

scala> order.split(",")(2).toInt

scala> order.split(",")(3)

scala> val ordersDF = ordersRDD.map(order => {

  (order.split(",")(0).toInt, order.split(",")(1), order.split(",")(2).toInt, order.split(",")(3))

}).toDF()
```

Flume Agent

 Source - tells where to get the data from

 Sink - tells where the data needs to be written

Channel - interim queue between source and sink where the events are held

In our example

- Exec Source - Exec source runs a given Unix command on start-up and expects that process to continuously produce data on standard out

- HDFS Sink - This sink writes events into the Hadoop Distributed File System (HDFS)

- Memory Channel - The events are stored in an in-memory queue with configurable max size

----

171.93.174.121 - - [13/Mar/2019:01:48:48 -0800] "GET /departments HTTP/1.1" 200 843 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko"