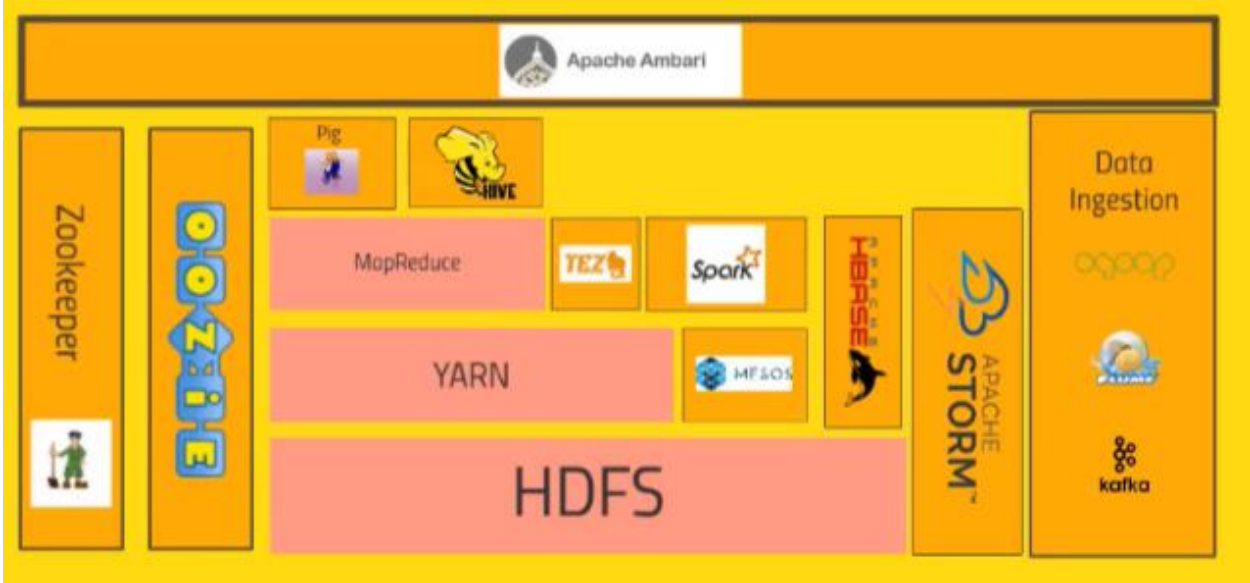
BIG DATA

My understanding of big data, its technologies and connections.

BUZZ WORDS in Big Data World:

- HDFS — Hadoop Distributed File System
- Yarn — Resource Negotiator
- Mesos — Resource Negotiator
- MapReduce — Programming model to process data across clusters
- Spark — at Level of Map Reduce, scripts to process queries on cluster, can do ML
- Tez — like Spark, uses Directed Acyclic Graphs
- Apache Hbase — NOSQL DB for large transactional data, can connect with other technologies
- Pig — High Level programming API, SQL like syntax, it coverts to MR's and process
- Hive — Like Pig but matches SQL more
- Apache Storm — to process streaming data in real-time -> HDFS, also Spark Streaming
- Oozie — Job Scheduler
- ZooKeeper — Node Manager, co-ordinates everything on cluster
- Sqoop — RDMS -> HDFS, data ingestion
- Flume — Web Logs in real-time -> HDFS
- Kafka — also a real-time data ingestion tool, can transfer any general data -> HDFS
- Apache Ambari — interface that sits on top of everything, view of the cluster
- My SQL — RDMS to store data and perform queries
- Mongo DB — NOSQL, faster querying performances
- Cassandra — NOSQL, faster querying performances
- Apache Drill — Query Engine to perform SQL queries on NOSQL DB
- Hue — Query Engine, Hue is to Cloudera like Ambari is to Horton Works
- Apache Phoenix — Query Engine
- Presto — Query Engine
- Zeppelin — Query Engine

HADOOP ECOSYSTEM



- MapReduce, Yarn, HDFS are the integral part of Hadoop.
- Rest all are added on to Hadoop.

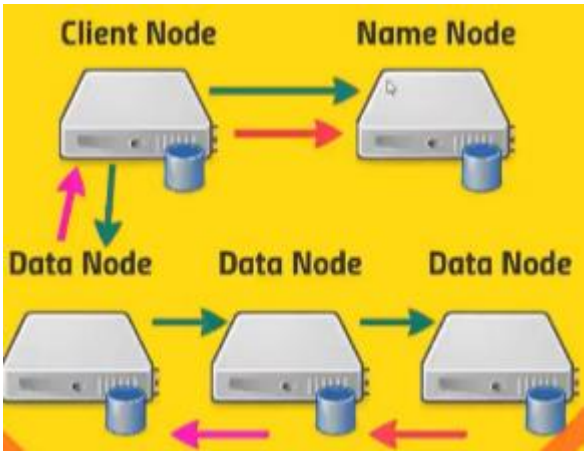
"Hadoop is an open source software platform for distributed storage and distributed processing of very large data sets on computer cluster built from commodity hardware" – Hortonworks

WHY HADOOP?

- Data's too darn big in present world – TB's per day
- Vertical Scaling cannot cut it
  - Disk seek times
  - Hardware failures
  - Processing times

So, with hadoop we cluster the files on several nodes and using different technologies on the file system to make processing accurate, fast and easy.

HDFS ARCHITECTURE



- Big files ---> broken to blocks ---> stored across several computers(replications for backups)

- Name Node writes to local disk and NFS (Backup)
- Maintains merged copy of edit log you can restore from (Secondary Name Node)
- Each name node manages a specific namespace volume

In HDFS to read or write the client node first talks to the name node which directs the client node to the appropriate data node.

Data nodes talk to themselves to keep replications.

HDFS High Availability

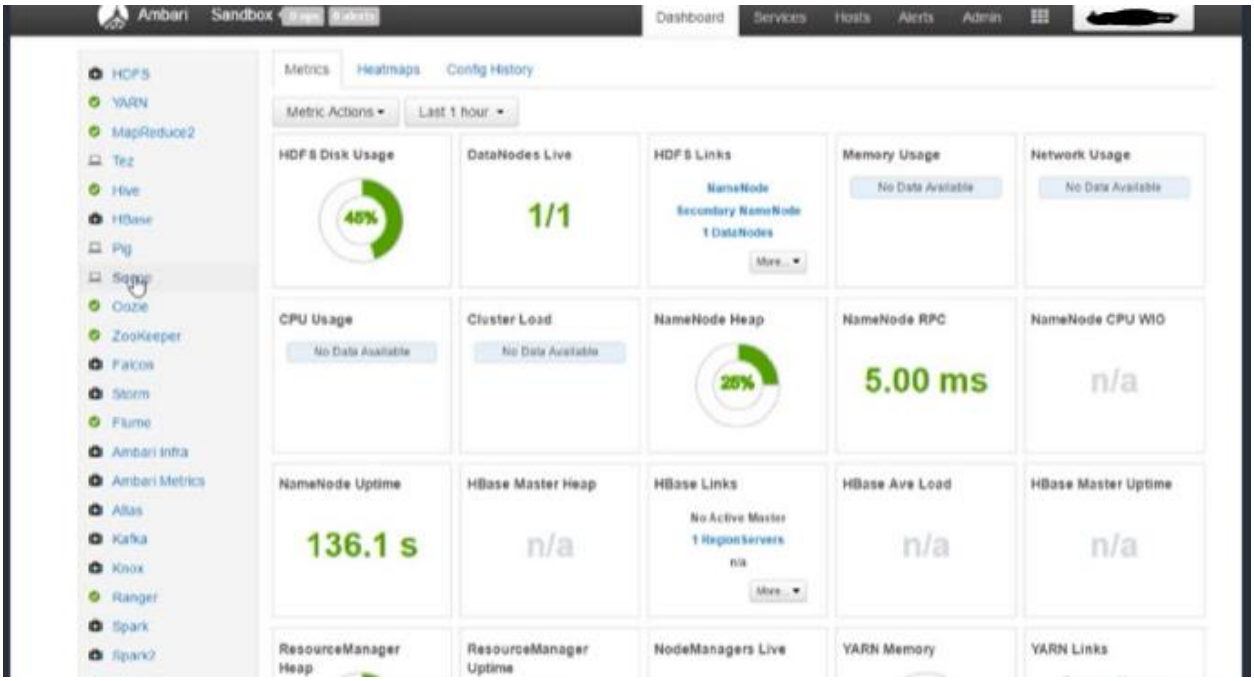
- Hot standby namenode using shared edit log
- Zookeeper tracks active namenode
- Uses extreme measures to ensure only one namenode is used at a time.

Using HDFS

- UI (Ambari)
- Command – Line Interface
- HTTP / HDFS Proxies
- Java Interface
- NFS Gateway

Installing dataset into HDFS

1) AMBARI INTERFACE



- You will have a files view in the ambari interface and can directly upload the table with appropriate delimiter and column names. This will be stored in the HDFS.

2) Using command line:

On putty connecting to the sandbox server can use commands like

- hadoop fs -mkdir directory — to create a directory on hdfs
- hadoop fs -ls — to see the list of directories
- wget http://..... link — to get data from an online source
- hadoop fs -copyFromLocal file directory/file — to copy from one place to other
- hadoop fs -rm file directory/file — to remove a file
- hadoop fs -rmdir directory — to remove a directory
- hadoop fs — to get list of commands

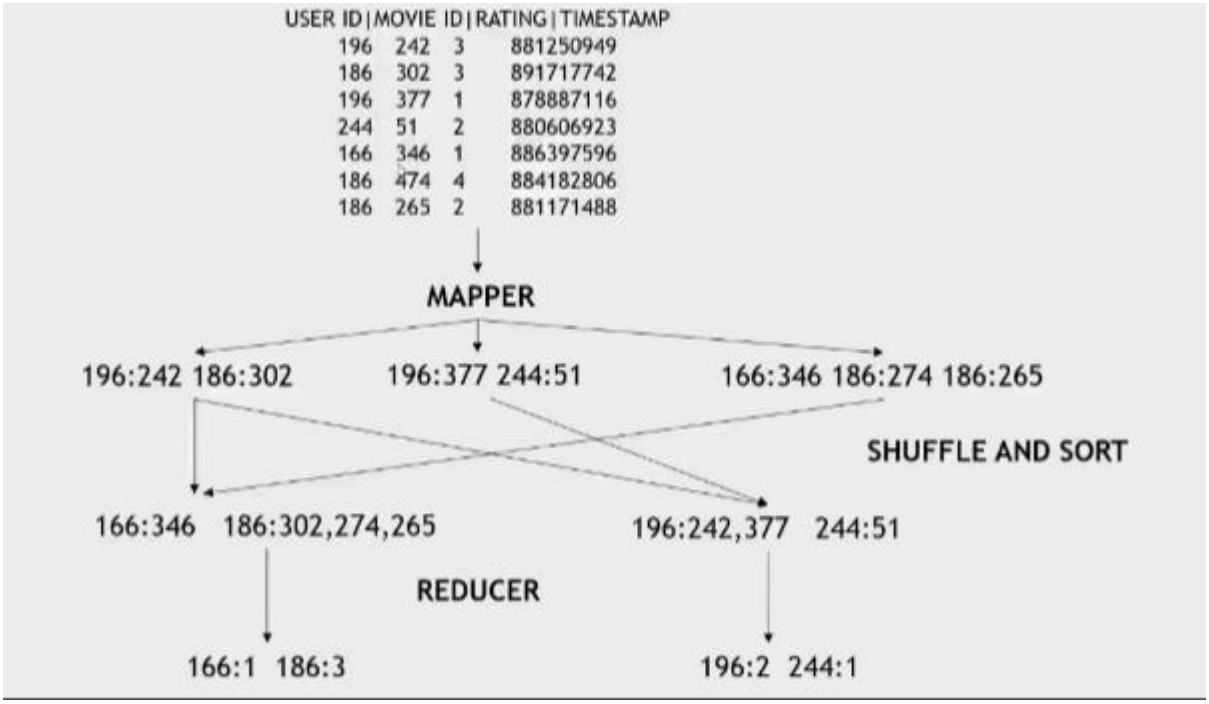
AMBARI

- We have already seen how to add data on to hdfs through the ambari view and various applications on its interface.
- **ambari -admin -password -reset**
- The above mentioned command is to be executed via super user permissions on your terminal which in turn can be got by executing the **su root** command and giving the password.
- This admin access into the Ambari interface gives you the additional options like start/stop any service in the interface.
- An additional cool feature from Ambari is that, we can set notifications to our email when something goes wrong
- So, Ambari just waits for something to go wrong for us respond or else it can handle the process for itself, if all the necessary services are started.

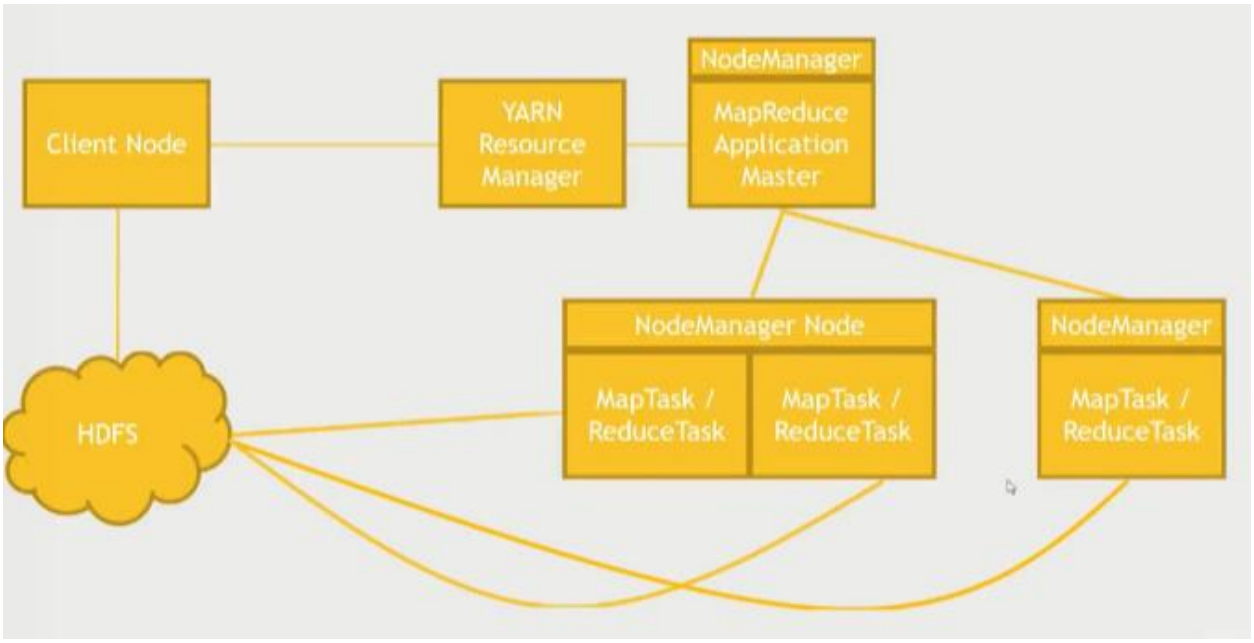
MAPREDUCE

- It distributes the processing of data on your cluster
- It divides your data into partitions that are MAPPED(transformed) and REDUCED (aggregated) by mapper and reducer functions you define.
- Resilient to failure – an application master monitors your mappers and reducers on each partition.

A simple example to show how a mapper and reducer works in visual,



- A mapper transforms the input data into key value pairs.
- At this stage there can be repeating keys.
- A Shuffle and Sort is performed where keys are merged together like in a dictionary and are sorted.
- Then the reducer here is giving out the length for each key i.e., number of elements with each key.
- Each node here can be a computer, multiple machines can work together that's the advantage.



- The Master Node assigns tasks to the worker node by interacting with the YARN, a resource manager.
- The client Node, worker nodes will be connected to the HDFS Cluster.
- Node closer to the data will be give priority at task.
- Multiple tasks can be done on one physical computer and there can be multiple computers working in parallel.

- MapReduce is natively JAVA.

- So, Streaming is a concept that allows interfacing of other languages like PYTHON to MapReduce.

- MapReduce is a fundamental for hadoop. But, SPARK, PIG, HIVE are more friendly and are taking over.

- Simple example MapReduce scripts are given below (in python), we can get the script even though its not on the local system using the 'wget' command with its path.

Map Reduce scripts can have any number of reducers in a single function.

Handling Failure

- Application master monitors worker tasks for errors or hanging
    - Restarts as needed
    - Preferably on a different node

- What if the application master goes down?
    - YARN can try to restart it

- What if an entire Node goes down?
    - This could be the application master
    - The resource manager will try to restart it

- What if the resource manager goes down?
    - Can set up “high availability” using ZooKeeper to have a hot standby

- Run locally
    - python script.py filename

- Run with Hadoop
    - python script.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar filename

  If the file is not in local system but on the hdfs cluster then

    - python script.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar hdfs://…..path

PIG

- Pig is a scripting language called Pig Latin that is on top of MapReduce and Tez in hadoop system.
- Pig can stand alone on itself, i.e., without a hdfs cluster
- Writing mappers and reducers by hand takes a long time.
- Pig introduces Pig Latin, a language with SQL like syntax to define your map and reduce steps.
- It is highly extensible with user – defined functions (UDF's)
- Pig can run
    - grunt - command line interface
    - Script file - .pig
    - Ambari/ Hue - web interfaces

Executing a pig script lets it do its own map reduce tasks.

But using Tez for executing a pig script is very fast in comparison, which can be found in the pig query view on ambari by just enabling a checkbox.

A Pig Query view on Ambari shows a window with results, a window with logs for run-time (if errors), and the actual script window.

FUNCTIONS on a relation

- LOAD STORE DUMP
- FILTER DISTINCT FOREACH/GENERATE MAPREDUCE STREAM SAMPLE
- JOIN COGROUP GROUP CROSS CUBE
- ORDER RANK LIMIT
- UNION SPLIT
- AVG CONCAT COUNT MAX MIN SIZE SUM

LOADERS

- PigStorage
- TextLoader
- JsonLoader
- AvroStorage
- ParquetLoader
- OrcStorage
- HBaseStorage

These are various functions and loaders that can be performed on the SQL like Pig queries to process data on a cluster or on a local storage.

Pig can also be executed using the Query engines in collaboration with external databases which will be further discussed.

SPARK

Many interesting features like Machine Learning, Graph Analysis, Streaming data …. are built on Spark.

- "A fast and general engine for large-scale data processing" is a general definition for spark which is true with many Hadoop technologies.
- It gives a lot of flexibility to write scripts in Java, Python or Scala to do complex queries.
- It's scalable. Spark need not work with Hadoop, it also has its own resource manager.

- Executor processes happen in parallel containing
    - Cache - one main reason for its speed.
    - Tasks
- It runs much faster than Hadoop MapReduce in memory, or 10x faster on disk.
- DAG Engine optimizes workflows, which is a key to its speed.

- It is built around one main concept: the **Resilient Distributed Dataset (RDD)**

- In Spark 2.0 it has datasets on RDD's.
- Components of Spark
    - Spark Streaming – Instead of the batch processing can process real time data
    - Spark SQL – SQL queries and SQL like functions on data
    - MLLib – library for Machine Learning and Data analysis
    - GraphX – for graph theory analysis

- Spark itself is written in Scala.
- Python is easier because we deal less with JAR's, dependencies etc..
- But, Scala is fast and reliable and is compiled to Java Byte code. uses less resources.
- But Python and Scala are very similar.

Python code to square numbers in a data set:

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
```

Scala code to square numbers in a data set:

```
val nums = sc.parallelize(List(1, 2, 3, 4))
val squared = nums.map(x => x * x).collect()
```

- Not very hard to move from Python to Scala

- RDD objects are way to
    - **Distribute** your job evenly across your cluster
    - Can handle failure in a **Resilient** manner
    - Looks like just a **dataset** in the end

- RDD objects use a spark context.

Creating RDD's

- nums = parallelize ([1, 2, 3, 4])
- sc.textFile("file:……")
  - or s3n:// , hdfs://
- hiveCtx = HiveContext(sc) rows = hiveCtx.sql("SELECT name, age FROM users")
- Can also create from:
    - JDBC
    - Cassandra
    - Hbase
    - JSON, CSV, sequence files, object files, various compressed formats

Transforming RDD's, Mapper jobs

- map – when there is one row of input for one row of output
- flatmap – if it's doesn't have one to one relation between input and output
- filter
- distinct
- sample
- union, intersection, subtract, cartesian

Functions can also be performed on RDD objects.

RDD Actions, Reducer jobs

- collect
- count
- countByValue
- take
- top
- reduce
- Nothing will kick off untill an action is called in the driver program.
- Once called, it will find the fastest path for the job through the dependencies and kicks off the job.

- When run, it by default takes SPARK 1, if need change it to SPARK2 using
  Export SPARK_MAJOR_VERSION = 2

SPARK SQL

- Extends RDD to a "DataFrame" object.
- DataFrames contain:
    - Contains Row objects.
    - Can run SQL queries
    - Has a schema
    - Read and write to JSON, Hive
    - Communicates with JDBC/ODBC, Tableau

- from pyspark.sql import SQLContext, ROW
- Functions like filter, select, groupby can be performed on DataFrame objects.
- As DataFrames are on RDD, they can be got back anytime.

SPARK 2.0

- In Spark 2.0, a DataFrame is really a DataSet of Row objects.
- DataSets can wrap known, typed data too.

- Spark SQL exposes a JDBC/ODBC server.
- You can create new tables, or query existing ones that were cached using hiveCtx.cacheTable("tableName")

UDF's

- It supports user defined functions

```
from pyspark.sql.types import IntegerType
hiveCtx.registerFunction("square", lambda x: x*x, IntegerType())
df = hiveCtx.sql("SELECT square('someNumericFiled') FROM tableName)
```

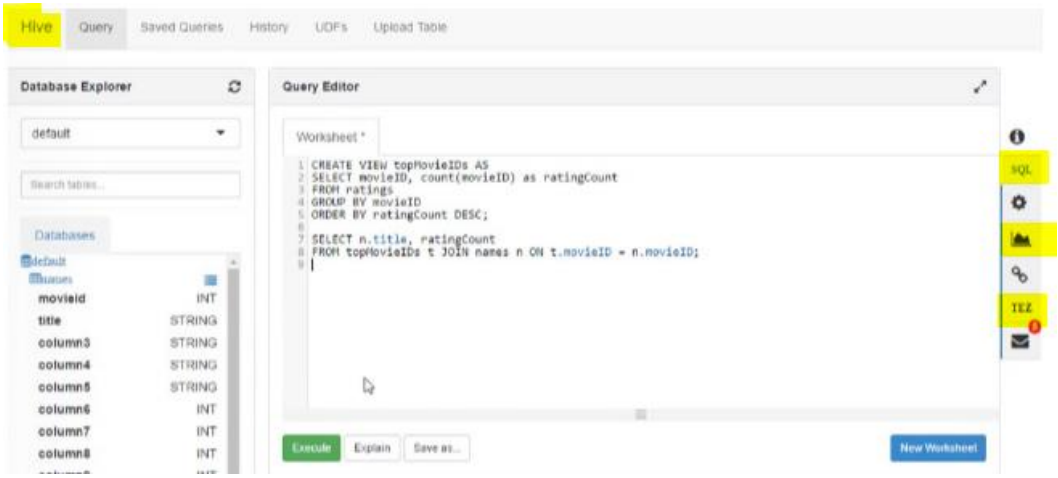
- Spark can also be used to do predictions on bigdata using its machine learning library MLLib.

HIVE

Hive is used to make the haddock cluster look like a RDMS and perform SQL like queries on it.

- Uses familiar SQL syntax (HiveQL)
- Interactive
- Scalable- appropriate for data warehouse applications
- Easy OLAP queries
- Highly optimized
- Highly extensible – UDF's, Thrift Servers, JDBC/ODBC driver

- Not appropriate for OLTP
- Pig, Spark allows more complex stuff
- No transactions
- No record level updates, inserts, deletes



- Again, running on TEZ makes it way faster.
- Visualization can also be done in this interface itself.
- **RDMS is Schema on Write**
- **Hive is Schema on Read**

```
CREATE TABLE ratings (
    userID INT,
    movieID    INT,
    rating INT,
    time    INT)
ROW FORMAT DELIMTED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '${env:HOME}/ml-100k/u.data'
OVERWRITE INTO TABLE ratings;
```

Hive can be used via

- hive> prompt
- saved query files
  - hive -f /somepath/queries.hql
- Ambari/hue
- JDBC/ODBC server
- Thrift service
- Oozie

SQOOP

A Data Ingestion tool.

- Import data from MySQL to HDFS

sqoop import --connect jdbc:mysql://localhost/database --drivercom.mysql.jdbc.Driver --table name

- Import data from MySQL directly into Hive

sqoop import --connect jdbc:mysql://localhost/database --drivercom.mysql.jdbc.Driver --table name

--hive-import

- Export data from Hive to MYSQL, target table must already exist in MySQL

sqoop export --connect jdbc:mysql://localhost/database -m 1 --drivercom.mysql.jdbc.Driver

--table name –export-dir/../../../ --input-fields-terminated-by '\0001'

- Hortonworks Sandbox will have MySQL pre-installed.
- for admin **permissions we use mysql -u root -p**

- **GRANT ALL PERMISSIONS ON database.* to ''@'localhost';**

NOSQL

Non-Relational Database, Not only SQL, No Schema

HBASE

- HBase is a non-relational, scalable database built on Hadoop.
- It has no language, but has an API for operations
- **CRUD operations**
    - **Create**
    - **Read**
    - **Update**
    - **Delete**

- Its is helpful for fast access to any given ROW
- A row is referenced by a unique KEY
- Each row has some small number of **COLUMN FAMILIES**
- A **COLUMN FAMILY** may contain arbitrary **COLUMNS**
- Sparse data is ok – missing columns in a row consume no storage

Some ways to access HBase

- HBase shell
- Java API
- Spark, Hive, Pig
- REST service
- Thrift service

Open a port on Virtual Box for HBase and connect it on terminal

for rest service access

to start and stop the HBase service following commands are used.

- **……..**
- HBase and Pig can be integrated for faster performances
- **create table 'name' 'column family'**
- scan, disable, drop commands can be performed on the table

- When there is already a HDFS using HBase is suggested because its built-on hadoop.

CASSANDRA

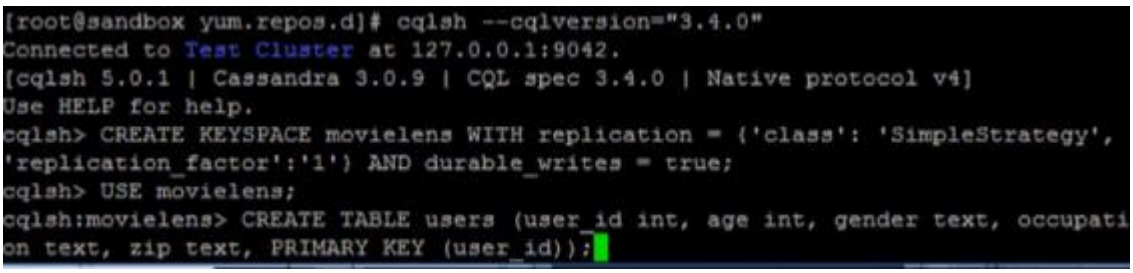
- Unlike HBase, there is no master node at all, every node runs the same software and performs the same function.
- The client can talk to any node.
- Thus, has a ring architecture, where each node talks to the others among themselves and keep backup copies.
- Cassandra prefers Availability over Consistency.
- Data model is like HBase/BigTable
- It's non-relational but has a limited CQL query language as its interface.

- CQL have no joins
- Queries should be on some primary key
- Its command line CQLSH
- Here a database is termed as keyspace

Cassandra and Spark

- DataStax offers a Spark-Cassandra connector
- Allows you to read and write Cassandra tables as DataFrames
- Use Spark for analytics on data stored in Cassandra
- Use Spark to transform data and store it into Cassandra for transactional use.

- Install Cassandra
- service cassandra start
- cqlsh –cqlversion="x.x.x"



spark-submit –packages datastax:spark-cassandra-connector:x.x.x-M2-s x.x CassandraSpark.py

x - version

- service cassandra stop

MONGO DB

managing hu**MONGO**s data

- It is a document-based model.

- can have different fields in every document
- no single “key” as in other databases
- can create indices on any field or combination of fields
- if you want to “shard”, then you must do so on some index
- gives lot of flexibility
- MONGO DB doesn’t give index like Cassandra, if need we have to create one
- No real schema is enforced
- contains **Databases**, **Collections**, **Documents**
- contains a Single-master
- maintains backup copies of your database instance
- secondaries can elect new primary within seconds if primary goes down
- but should make sure the operation log is long enough to give time to recover the primary when it comes back

- It’s not just NoSQL Database – a very flexible document model
- shell is a full JavaScript interpreter
- supports many indices
- built-in aggregation capabilities, MapReduce, GridFS(local HDFS)
- SQL connector is available

If Mongo DB is not on Ambari should be installed inside

/var/lib/ambari-server/resources/stacks/HDP/version/services

git clone httpd://github.com/nikunjness/mongo-ambari.git

spark -submit –packages org.mongodb.spark:mongo-spark-connector_x.x:x.x.x MongoSpark.py

x – version

can also be performed on mongo shell using command

mongo

```
switched to db movielens
> db.users.find( {user_id: 100 } )
{ "_id" : ObjectId("588a1c2046e0fb17ab62db7d"), "age" : NumberLong(36), "gender"
 : "M", "occupation" : "executive", "user_id" : NumberLong(100), "zip" : "90254"
 }
```

```
> db.users.aggregate( [
... { $group: { _id: { occupation: "$occupation"}, avgAge: { $avg: "$age" } } }
... ])
```

factors to think while choosing a Data Base

- Support
- Budget
- CAP theorem
- Simplicity