[Introduction 2](#_Toc45706622)

[Storing And Processing Data In Hadoop 2](#_Toc45706623)

[BigData And Hadoop Testing 2](#_Toc45706624)

[Big Data Formats 3](#_Toc45706625)

[What Is The Strategy Or Plan For Testing BigData? 4](#_Toc45706626)

[Strategies behind Testing Big Data 4](#_Toc45706627)

[Big Data Testing Approach 5](#_Toc45706628)

[Testing Types For BigData Testing 6](#_Toc45706629)

[#1) Architectural Testing 6](#_Toc45706630)

[#2) Database Testing 6](#_Toc45706631)

[#3) Performance Testing 7](#_Toc45706632)

[#4) Functional Testing 8](#_Toc45706633)

[#5) Non-Functional Testing 9](#_Toc45706634)

[Roles And Responsibilities Of Hadoop Testing 10](#_Toc45706635)

[Step 1: Data Staging Validation 12](#_Toc45706636)

[Step 2: "MapReduce" Validation 12](#_Toc45706637)

[Step 3: Output Validation Phase 12](#_Toc45706638)

[Testing Approach For Hadoop Testing / BigData Testing 15](#_Toc45706639)

[Test Environment Needs 16](#_Toc45706640)

[Challenges in Big Data Testing 17](#_Toc45706641)

[Traditional Testing and Big Data Testing 18](#_Toc45706642)

[Big Data Components 19](#_Toc45706643)

[Hadoop 20](#_Toc45706644)

[Hive 21](#_Toc45706645)

[What is Hive 22](#_Toc45706646)

[Advantages of Hive 22](#_Toc45706647)

[Features of Hive 23](#_Toc45706648)

[Architecture of Hive 23](#_Toc45706649)

[1. Hive Clients: 24](#_Toc45706650)

[2. Hive Services: 24](#_Toc45706651)

[3.Hive Storage and Computing: 24](#_Toc45706652)

[Different modes of Hive 26](#_Toc45706653)

[Metastore: 27](#_Toc45706654)

[HIVE INSTALLATION 28](#_Toc45706655)

[Data Model 28](#_Toc45706656)

[Tables: 29](#_Toc45706657)

[1. Managed Table: 29](#_Toc45706658)

# Introduction

BigData testing is defined as testing of Bigdata applications. Big data is a collection of large datasets that cannot be processed using traditional computing techniques.[Testing](https://www.guru99.com/software-testing.html)of these datasets involves various tools, techniques, and frameworks to process. Big data relates to data creation, storage, retrieval and analysis that is remarkable in terms of volume, variety, and velocity.

# Storing And Processing Data In Hadoop

To perform these processes on the Hadoop system, we have the manpower that is categorized into four sections.

* **Hadoop Administrators** are responsible for setting up the environment and have the Administration Rights to access the Hadoop Systems.
* **Hadoop Developers** develop the programs regarding pulling, storing and processing the data from different locations to centralized locations.
* **Hadoop Testers** for validating and verifying the data before pulling from different locations and after pulling at the centralized location as well as validating & verification is done while loading the data to the client environment.
* **Hadoop Analysts** operate when data loading is done and when the data reaches the warehouse at the client location. They use this data for report and dashboard generation. The analysts perform the data analysis for growth and business development.

We know that Hadoop is not a single system; it contains multiple systems and machines. The data is split and stored into multiple machines and if we want to access it again we need to combine and pull the data into reports and so on.

The developer is responsible for writing programs in JAVA and Python to extract the data and store it.

The other job of a developer is to process the data. There are two layers of Hadoop, one is for storing i.e. Hadoop HDFS and another for Processing i.e. Hadoop MapReduce

Storing means whatever data we have in the source just gets stored/inserted in the system. Processing means we need to split it into multiple machines and again combine and send it to the client.

Thus, Storing and Processing are done by programming scripts, and the developer is responsible for writing the scripts.

Apart from programming, the other method to store and process the data in Hadoop is using database applications like Hive, Impala, HBase, etc. These tools don't need any programming knowledge.

# BigData And Hadoop Testing

Once storing and processing are done by the developer the data goes for report generation. Before that, we need to verify the processed data for accuracy and check if the data is accurately loaded and processed correctly or not.

So the program and/or scripts created by a developer need to be verified by the Hadoop or BigData Tester. The tester needs to know basic programming like Mapper, Hive, Pig Scripts, etc. to verify the scripts and to execute the commands.

So, before testing, the testers need to know what all programs and scripts are working, how to write the code and then think about how to test them. Testing can be done either manually or by using automation tools.

Hadoop has various kinds of testing like Unit Testing, Regression Testing, System Testing, and Performance Testing, etc. So these are the common testing types that we use in our normal testing as well as Hadoop and BigData testing.

We have the same kind of testing terminologies like test strategy, test scenarios, and test cases, etc. in Hadoop and BigData Testing. Only the environment is different and there are different kinds of techniques that we use to test the BigData and Hadoop System because here we need to test the data and not the application.

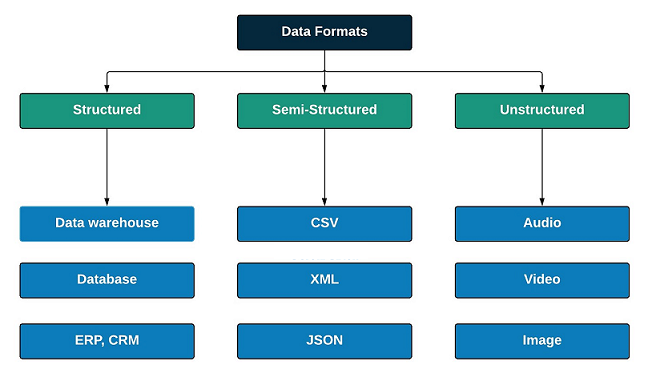
***How to test the BigData and what all things require testing in BigData?***

For BigData testing, we need to have some plans and strategies.

**Thus we need to consider the following points:**

* What is the strategy or plan of testing for the BigData?
* What kind of testing approaches are applied to BigData?
* What is the environment required?
* How to validate and verify the BigData?
* What are the tools used in BigData Testing?

## Big Data Formats



Big-Data gains its fame through its superiority in handling multiple formats of data which the traditional data processing units fail to handle. The data formats which the big data can handle are as follows.

* Structured Data
* Semi-Structured Data
* Unstructured Data

**Structured Data**

Any tabular data which is meaningfully organised under rows and columns with easy accessibility is known as Structured Data. It can be organised under named columns in different storage units such as an RDBMS.

Example: Tabular Data

**Semi-Structured Data**

Semi-Structured Data lies perfectly in between the Structured and Unstructured Data. It cannot be directly ingested into an RDBMS as it includes metadata, tags, and sometimes duplicate values. Data needs some operations to be applied to it before the data is ready to be ingested.

Example: .CSV, .JSON

**Unstructured Data**

Data that does not obey any kind structure is known as Unstructured data. Unlike the Structured Data, The unstructured Data is difficult to store and retrieve. Most of the data generated by the organisations are Unstructured type of data.

Example: Images, Videos, Audio

# What Is The Strategy Or Plan For Testing BigData?

## ****Strategies behind Testing Big Data****

Testing an Application that handles terabytes of data would take the skill from a whole new level and out of the box thinking. The core and important tests that the Quality Assurance Team concentrates is based on three Scenarios. Namely,

* Batch Data Processing Test
* Real-Time Data Processing Test
* Interactive Data Processing Test

**Batch Data Processing Test**

The Batch Data Processing Test involves test procedures that run the data when the applications in Batch Processing mode where the application is processed using Batch Processing Storage units like **HDFS**. The Batch Process Testing mainly involves running the application against faulty inputs varying the volume of the data

**Real-Time Data Processing Test**

The Real-Time Data Processing Test deals with the data when the application is in Real-Time Data Processing mode. The application is run using Real-Time Processing tools like **Spark**. Real-Time testing involves the application to be tested in the real-time environment and it is checked for its stability.

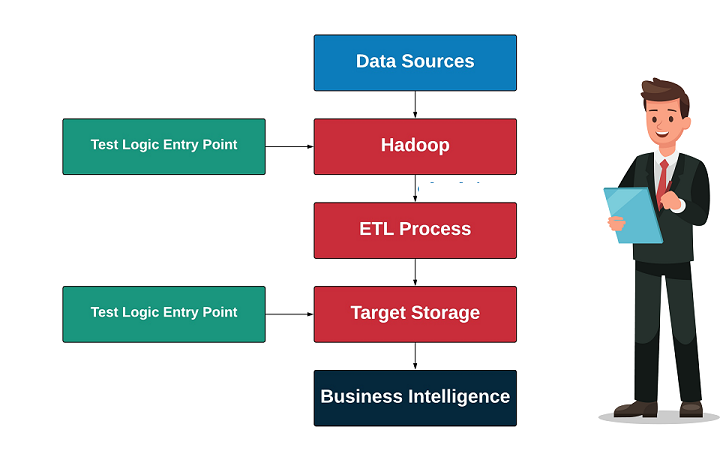
**Interactive Data Processing Test**

The Interactive Data Processing Test integrates the real-life test protocols that interact with the application as in the view of the real-life user. Interactive Data Processing mode uses Interactive Processing tools like **HiveSQL**.

## Big Data Testing Approach

The General approach to test a Big Data Application involves the following stages.

* Data Ingestion
* Data Processing
* Validation of the Output



**Data Ingestion**

Data is first loaded from source to Big Data System using extracting tools. The Storage might be HDFS, MongoDB or any similar storage. Then, the loaded data is cross-checked for errors and missing values.

Example: Talend

**Data Processing**

In this stage, the key-value pairs for the data get generated. Later, the MapReduce logic is applied to all the nodes and checked if the algorithm works fine or not. A data validati

**Validation of the Output**

At this stage, the output generated is ready to be migrated to the data warehouse. Here, The transformation logic is checked, the data integrity is verified and the key-value pairs at the location are validated for accuracy.on process takes place here to make sure the output is generated as expected.

# Testing Types For BigData Testing

These testing types are also as important as Functional and Performance Testing.

## #1) Architectural Testing

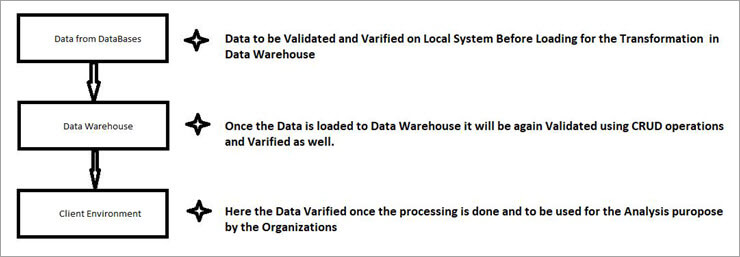
Architecture testing concentrates on establishing a stable Hadoop Architecture. The architecture of Big Data Processing Application plays a key role in achieving smooth operations. Poorly designed architecture leads to chaos like,

1. Performance Degradation
2. Node Failure
3. High Data Latency
4. May require high Maintenance

## #2) Database Testing

Here, the process validation comes into the picture and we need to validate the data from various Databases i.e. we need to ensure that the data fetched from the source databases or local databases must be correct and proper.

Also, we need to check that the data available in the Source Databases are matched with the data entered in Hadoop System.



Similarly, we need to validate if the data in Hadoop System is correct and proper after processing or say after transformation and to be loaded to the client’s environment with proper validation and verification.

As a part of Database Testing, we need to go through the **CRUD** operations i.e. **Create** the data in Local Databases then **Retrieve** the data and we need to search it and it should be available in Database before and after loading into Data Warehouse and from Data Warehouse to the Client’s Environment.

Verification of any **Updated** Data on every stage of Storing or Loading and Processing the data. Deletion of any corrupted data or any duplicate and null data.

## #3) Performance Testing

Performance testing highly concentrates on the performance delivered by all the components of the big data system. Performance testing includes the following Categories.

* Data Collecting Phase
* Data Ingesting Phase
* Data Processing
* Component Peripheral testing

**Data Collecting Phase**

In this Stage, Big Data System is validated based on its speed and capacity to grasp the data within a given timeframe from the different sources like RDBMS, Databases, Data-ware houses and many more.

**Data Ingesting Phase**

The next phase after Data Collection is the Data Ingestion. Here the application is tested and validated based on its pace and capacity to load the collected data from the source to the destination which might be HDFS, MongoDB, Cassandra or any similar Data Storage unit.

**Data Processing**

Here, the Application is tested based on the Map-Reduce logic written. The logic is run against every single node in the cluster and the processing speeds are validated. The Queries to be executed are expected to perform with high speeds with low latency.

**Component Peripheral testing**

This stage is related to component performance. Each component in the system should be highly available and connected. The component backup should be online when any node faces failure. High Capacity Data Exchange should smoothly be supported.

**Parameters involved in Performance testing**

* **Data Storage:** Take note of the orientation in which the data gets stored in the system
* **Commit Logs:** Mark the limits for committing logs
* **Concurrency:** Check the number of threads allocated for the read-write process
* **Cashing:** Dedicate the **row cache** and **key cache**
* Timeout: Set timeouts for application, connection and queries etc.

## #4) Functional Testing

Functional Testing can be otherwise called as the different phases in testing the big data application. The Big Data Application is designed to deal with huge blocks of data. Such a huge volume and variety of data is often prone to bring data issues, such as bad data, duplicate values, metadata, missing values and whatnot.

This is exactly why the pioneers in testing the big data, designed the procedure for functional testing of big data. The different phases in which the big data is tested are as follows.

* Data Validation Phase
* Data Integrity Phase
* Data Ingestion Phase
* Data Processing Phase
* Data Storage Phase
* Report Generation Phase

**Data Validation Phase**

* Data validation phase deals with the Business logic and the layers in the Big Data Application
* The Data is collected from the source and it is run against the business use case
* Data collected is checked for accuracy and movement through the Layers of the application
* At this stage, the big data is tested with aggregation and filtering mechanisms
* The data undergoes End-to-End Validation and Transformation logic, based on Business rules

**Data Integrity Phase**

* Data is checked for completeness with referential integrity validation
* Data constraints and duplication is verified against error conditions
* Boundary testing which recognizes schema limits of each layer

**Data Ingestion Phase**

* The ability of the application to connect with different data modules is checked
* The data is replayed with messaging systems and any loss of data is monitored
* The main motto of this phase is to achieve the following qualities
  1. Fault Tolerance
  2. Continuous Data Availability
  3. Stable connection with a variety of Data Streams

**Data Processing Phase**

* Data processing Phase carefully examines and executes the business logic
* The business rules are cross-validated
* The Map-Reduce logic is validated at every stage
* Data is processed from End-to-End
* The application is checked for exceptions and they get perfectly handled

**Data Storage Phase**

* Data Storage Phase concentrates on the following parameters
  1. Read and Write Timeouts
  2. Continuous Availability
  3. Load Balancing
  4. Query Performance Analysis

**Report Generation Phase**

* It is the final stage in Functional testing. it deals with the following.
  1. Data Validation for measures and Dimensions
  2. Real-time reporting
  3. Data Drill up and Drill down mechanisms
  4. Business Reports and Charts

## #5) Non-Functional Testing

The Non-Functional Testing phase takes care of the three major dimensions and characteristics of Big Data. The Volume, Velocity, and finally the Variety of the Big Data. There are five stages involved in Non-Functional Testing.

* Data Quality Monitoring
* Infrastructure
* Data Security
* Data Performance
* Fail-over Test Mechanism

**Data Quality Monitoring**

* Data Quality Monitoring checks for erroneous data, records, and messages.
* Data Quality Monitoring makes sure the following parameters about data.
  1. Data Accuracy
  2. Data Precision
  3. Data Timeliness
  4. Data Consistency
  5. Data Profiling

**Infrastructure**

* Infrastructure makes sure there is a continuous service availability in both
  1. Internal Big Data Application Systems
  2. External Big Data Application Systems
* Infrastructure also takes care of the following
  1. Data Replication
  2. Data Backup
  3. Data Restore

**Data Security**

* Data Security is the most important aspect of any Big Data Application.
* Data Securit stage protects the sensitive data.
* It manages User Authentication checks and User Role-Based Authorization.
* Data Encryption and Masking of Personal Information

**Data Performance**

* Data Performance evaluates every single component.
* Evaluates maximum Data processing speed
* Evaluates maximum data capacity size
* Checks the Message transfer speed and response time
* Calculates the number of operations performed per unit time
* Engages parallel job monitoring
* Performs Read, Write and Update operations on Real-time Databases

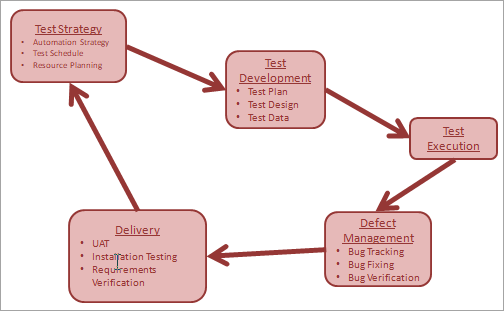
**Fail-over Test Mechanism**

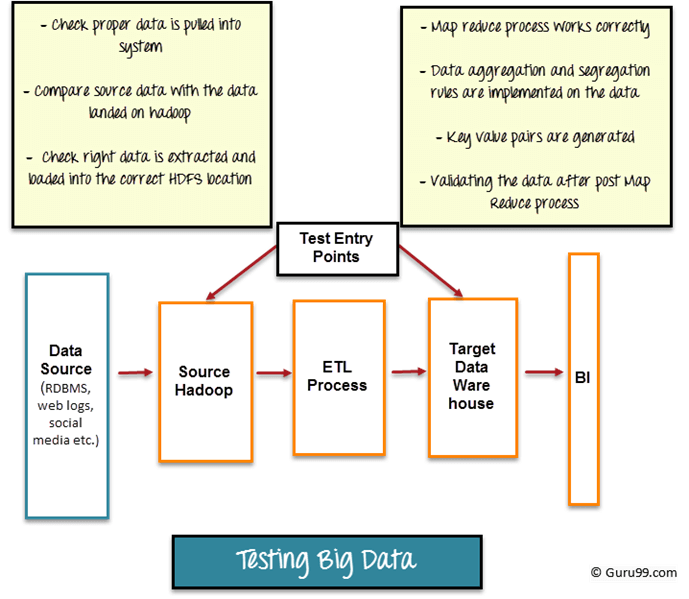
* Ensure seamless data processing while switching to neighbouring data nodes
* Create data recovery points parallelly and be ready for calamities.
* Be ready to replay the data using multiple offsets
* Enable Dynamic Clustering

# Roles And Responsibilities Of Hadoop Testing

As a Hadoop Tester, we are responsible for understanding the requirements, prepare the Testing Estimations, Planning of the Testcases, Get some Test Data to test some Testcases, be involved with Test Bed creation, Executing the Test Plans, Reporting & Retesting of defects.

Also, we need to be responsible for Daily Status Reporting and Test Completion.





Big Data Testing can be broadly divided into three steps

## Step 1: Data Staging Validation

The first step of big data testing also referred as pre-Hadoop stage involves process validation.

* Data from various source like RDBMS, weblogs, social media, etc. should be validated to make sure that correct data is pulled into the system
* Comparing source data with the data pushed into the Hadoop system to make sure they match
* Verify the right data is extracted and loaded into the correct HDFS location

Tools like [**Talend**](https://www.talend.com/)**,**[**Datameer**](http://www.datameer.com/)**,** can be used for data staging validation

## Step 2: "MapReduce" Validation

The second step is a validation of "MapReduce". In this stage, the tester verifies the business logic validation on every node and then validating them after running against multiple nodes, ensuring that the

* Map Reduce process works correctly
* Data aggregation or segregation rules are implemented on the data
* Key value pairs are generated
* Validating the data after the Map-Reduce process

## Step 3: Output Validation Phase

The final or third stage of Big Data testing is the output validation process. The output data files are generated and ready to be moved to an EDW (Enterprise Data Warehouse) or any other system based on the requirement.

Activities in the third stage include

* To check the transformation rules are correctly applied
* To check the data integrity and successful data load into the target system
* To check that there is no data corruption by comparing the target data with the HDFS file system data

We might be using Sqoop or Flume or whatever particular product to get the Data, records or whatever as my Data Sources. We may use these tools to get the data from the Data Sources into my staging directory which is the first phase of our process called **Extraction.**

Once the Data therein Staging Directory which actually happens to be HDFS (Hadoop Distribution File System), we will particularly use the scripting Language such as PIG to **Transform**that Data. That **Transformation** will be according to the Data that we have.

Once the Data is transformed accordingly using whatever scripting technology that we have, we will be **Loading** that Data into the Data Warehouse. From the Data Warehouse, that data will be used for OLAP Analysis, Reporting and Data Mining or for Analytics.

***Let’s go ahead and discuss which all phases we can use for Hadoop Testing.***

The first phase will be the Extraction phase. Here, we are going to get the data from our Source DataBases or from Flat files, and in that case, what we can do is, we can verify that all the Data has been copied successfully and correctly from source to the Staging Directory.

It may include, verifying the number of Records, the type of the Records and the type of the Fields, etc.

Once this data is copied to the Staging Directory, we will go ahead and trigger the second phase which is Transformation. Here, we will have some business logic that will act on the copied data from the Source Systems and will actually create or transform the data into the required business logic.

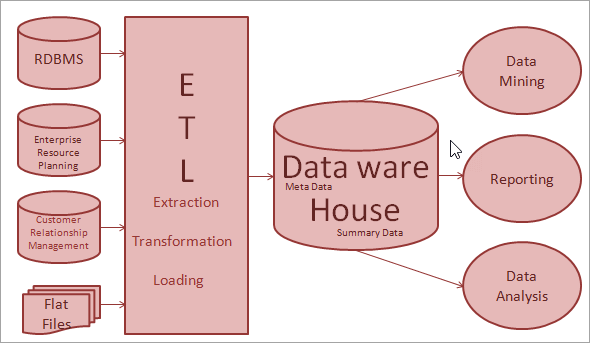
Transformation may include Sorting the Data, Filtering the Data, Joining the Data from two different Data Sources and certain other operations.

Once the Data is transformed, we will go ahead and have test plans ready and we will check if we are getting the output as expected, and all the output that we are getting are meeting the expected result and the Data Types, Field Values, and the ranges, etc. are something which is falling in place.

Once it is correct, we can go ahead and load the data into Data Warehouse.

In the loading phase, we are actually checking if the number of records from the Stage and the number of records in Data Warehouse is in sync, they might not be similar, but they are supposed to be in sync. We also see if the type of Data that has been transformed is in sync.

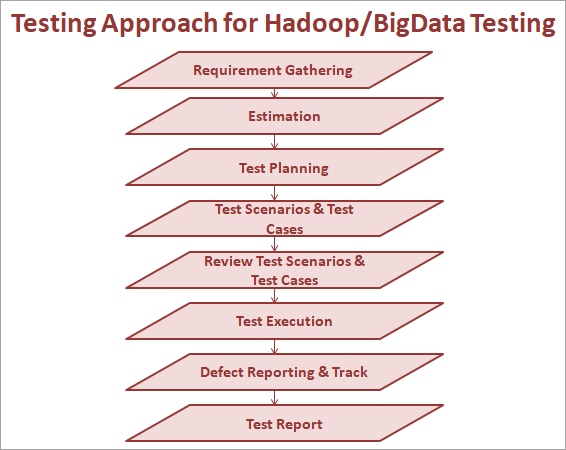
Post that we will use this Data for OLAP Analysis, Reporting and Data Mining which is the last layer of our product and in that case, we can have subsequent or we can say that the Test Plans available for all these layers.

Let’s discuss authentication in a particular manner, there are certain businesses which have Data that is restricted by its nature, this type of Data is called as PII Data as per the United States standards.

***PII*** stands for ***Personal Identifiable Information,***any information such as the Date of Birth, SSN, Mobile Number, Email Address and Address of House, etc. all fall under PII. This is restricted and cannot be shared with everyone.

The Data should be shared only with the persons who needed it the most and those who need the Data for actual processing. Having this check and the first line of defense in place is called Authentication.

# Testing Approach For Hadoop Testing / BigData Testing



**Test Scenarios And Test Cases**

Once we are done with the Test Planning, we need to prepare **Test Scenarios and Test Cases**, especially for Big Data Testing, we require a few documents along with the requirement document. Along with this requirement document what all do we need?

We need the **Requirement Document** that contains the needs of the Client, along with this we need the **Input Document**i.e. **Data Models.**Data Model in the sense what is the DataBase Schemas, what are the tables and what are the relationships all this Data will be available in the Data Models.

Also, we have the **Mapping Documents**, Mapping Documents for **E.g.** in Relational DataBases we have some Tables and after loading the Data through ETL in Data Warehouse in HDFS, what are all mapping we need to do? i.e. Mapping Data Type.

Also, we have required a **Design Document**, Design Document required for both the Development Team as well as the QA Team, because in the Design Document the Customer will provide, what kind of Map Reduce Jobs they are going to implement and what type of MapReduce Jobs takes Inputs and what type of MapReduce Jobs gives Outputs.

Similarly, if we have HIVE or PIG, what are all UDF’s the Customer has created as well as what are all the input they will take and what kind of output they will produce, etc.

**To prepare Test Scenarios and Test Cases, we need to have all these Documents by hand:**

* Requirement Document
* Data Model
* Mapping Document
* Design Document

These can vary from one Organization to another Organization, and there is no mandatory rule that we must have all these documents. Sometimes we have all documents and sometimes we have only two or three documents or sometimes we need to rely on one document also, that is up to project complexity, company schedules, and everything.

**Test Execution**

If we want to execute our Test Cases during execution, we need to check that the Developer has to send the information, if it is normal Functional Testing or some other testing or Automation Testing we require a Build. But, here from the Hadoop or BigData Testing point of view, the Developer will provide MapReduce Jobs.

HDFS Files – whatever files which are copied in HDFS those files information is required to check the privileges, HIVE Scripts which were created by the Developers to verify the Data in HIVE Table and also we need the HIVE UDF’s which were developed by the Developers, PIG Scripts and PIG UDF’s.

These are all the things we need to get from Developers. Before going for the execution we should have all these things.

For MapReduce Jobs, they will provide some JAR Files and as a part of the HDFS they have already loaded the data in HDFS and the files should be ready and HIVE Scripts to validate the Data in HIVE Tables. Whatever the UDF’s they have implemented will be available in the HIVE UDF’s. We require the same thing for PIG Scripts and UDF’s as well.

**Defect Reporting & Tracking**

Suppose if we find some Defect in the MapReduce Job, then we will report it to the Developer and they will again recreate the MapReduce Job and they do some code level modifications and then again they will provide the latest MapReduce Job, which we need to test.

This is an ongoing process, once the job is tested and passed, we again need to retest it and report it to the Developer and then get the next one for testing. This is how the Defect Reporting and Tracking activity is accomplished.

# Test Environment Needs

Test Environment needs to depend on the type of application you are testing. For Big data testing, the test environment should encompass

* Space for Storing, Processing and Validating Terra bytes of data should be available.
* It should have a cluster with distributed nodes and data
* It should have minimum CPU and memory utilization to keep performance high

**Tools used in Big Data Scenarios**

Various tools used in testing the Big Data are mentioned as shown below:

|  |  |
| --- | --- |
| **Process** | **Tools Description** |
| **Data Ingestion** | **Zookeeper, Kafka, Sqoop** |
| **Data Processing** | **MapR, Hive, Pig** |
| **Data Storage** | **Amazon S3, HDFS** |
| **Data Migration** | **Talend, Kettle, CloverDX** |

* HDFS Hadoop Distribution File System for Storing the BigData.
* HDFS Map Reduce for Processing the BigData.
* For NoSQL or HQL Cassandra DB, ZooKeeper and HBase, etc.
* Cloud-Based server tools like EC2.

|  |  |
| --- | --- |
| **Big Data Cluster** | **Big Data Tools** |
| **NoSQL:** | * CouchDB, DatabasesMongoDB, Cassandra, Redis, ZooKeeper, HBase |
| **MapReduce:** | * Hadoop, Hive, Pig, Cascading, Oozie, Kafka, S4, MapR, Flume |
| **Storage:** | * S3, HDFS ( Hadoop Distributed File System) |
| **Servers:** | * Elastic, Heroku, Elastic, Google App Engine, EC2 |
| **Processing** | * R, Yahoo! Pipes, Mechanical Turk, BigSheets, Datameer |

# Challenges in Big Data Testing

* **Automation**

Automation testing for Big data requires someone with technical expertise. Also, automated tools are not equipped to handle unexpected problems that arise during testing. Big Data Testing is highly complicated and the process requires a highly skilled official.

* **Virtualization**

It is one of the integral phases of testing. Virtual machine latency creates timing problems in real time big data testing. Also managing images in Big data is a hassle.

* **Large Dataset**
  + Need to verify more data and need to do it faster
  + Need to automate the testing effort
  + Need to be able to test across different platform
* **Environment**

Test environment and automation should be developed for different platforms

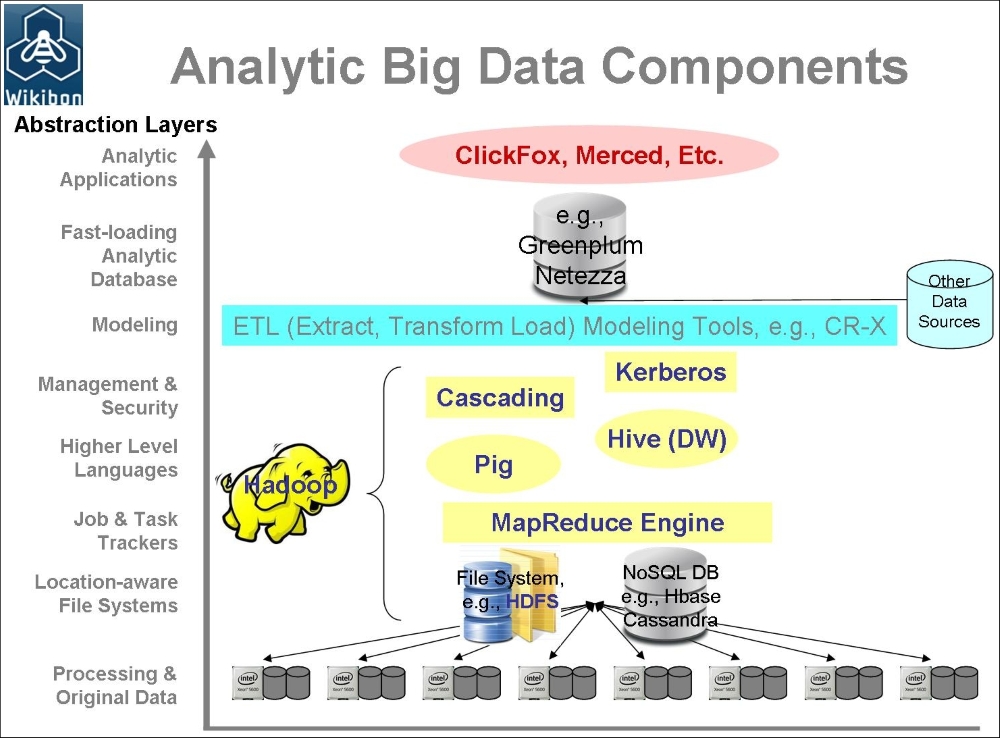
**Misc**

* Each component is from a different technology hence, requires isolated testing
* No single tool can perform end to end testing
* High Degree of scripting is required for designing test cases
* Customised Solutions are required to increase performance and test critical areas

## ****Traditional Testing and Big Data Testing****

|  |  |
| --- | --- |
| **Traditional Testing** | **Big Data Testing** |
| **Supports Structured Data** | **Supports all types of Data** |
| **Testing Does not R&D** | **R&D is Testing needed** |
| **limited Data size does not require special Environment** | **Special Environment is needed because of the huge data** |
| **Uses EXCEL based Macros or UI based automation tools** | **Has a vast range of programmable tools for testing** |
| **Basic Operations knowledge is enough to run tests** | **The highly qualified skill set is necessary** |

# Big Data Components



* A storage system can be one of the following:
  + **HDFS** (short for **Hadoop Distributed File System**) is the storage layer that handles the storing of data, as well as the metadata that is required to complete the computation
  + NoSQL stores that can be tabular stores such as HBase or key-value based columnar Cassandra
* A computation or logic layer can be one of the following:
  + **MapReduce**: This is a combination of two separate processes, the mapper and the reducer. The mapper executes first and takes up the raw dataset and transforms it to another key-value data structure. Then, the reducer kicks in, which takes up the map created by the mapper job as an input, and collates and converges it into a smaller dataset.
  + **Pig**: This is another platform that's put on top of Hadoop for processing, and it can be used in conjunction with or as a substitute for MapReduce. It is a high-level language and is widely used for creating processing components to analyze very large datasets. One of the key aspects is that its structure is amendable to various degrees of parallelism. At its core, it has a compiler that translates Pig scripts to MapReduce jobs.

It is used very widely because:

* + Programming in Pig Latin is easy
  + Optimizing the jobs is efficient and easy
  + It is extendible
* Application logic or interaction can be one of the following:
  + **Hive**: This is a data warehousing layer that's built on top of the Hadoop platform. In simple terms, Hive provides a facility to interact with, process, and analyze HDFS data with Hive queries, which are very much like SQL. This makes the transition from the RDBMS world to Hadoop easier.
  + **Cascading**: This is a framework that exposes a set of data processing APIs and other components that define, share, and execute the data processing over the Hadoop/Big Data stack. It's basically an abstracted API layer over Hadoop. It's widely used for application development because of its ease of development, creation of jobs, and job scheduling.
* Specialized analytics databases, such as:
  + Databases such as Netezza or Greenplum have the capability for scaling out and are known for a very fast data ingestion and refresh, which is a mandatory requirement for analytics models.

## Hadoop

Hadoop is an open-source framework to store and process Big Data in a distributed environment. It contains two modules, one is MapReduce and another is Hadoop Distributed File System (HDFS).

* **MapReduce:** It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware.
* **HDFS:**Hadoop Distributed File System is a part of Hadoop framework, used to store and process the datasets. It provides a fault-tolerant file system to run on commodity hardware.

The Hadoop ecosystem contains different sub-projects (tools) such as Sqoop, Pig, and Hive that are used to help Hadoop modules.

* **Sqoop:** It is used to import and export data to and from between HDFS and RDBMS.
* **Flume:** Import unstructured data to HDFS
* **Pig:** It is a procedural language platform used to develop a script for MapReduce operations.
* **Hive:** It is a platform used to develop SQL type scripts to do MapReduce operations.
* **MongoDB –** No Sql(data stored is not in tables like RDBMS)

**Note:** There are various ways to execute MapReduce operations:

* The traditional approach using Java MapReduce program for structured, semi-structured, and unstructured data.
* The scripting approach for MapReduce to process structured and semi structured data using Pig.
* The Hive Query Language (HiveQL or HQL) for MapReduce to process structured data using Hive.

# Hive

Hive is developed on top of Hadoop. Hive is rigorously industry-wide used tool for Big Data Analytics.It is a data warehouse framework for querying and analysis of data that is stored in HDFS. Hive is an open source-software that lets programmers analyze large data sets on Hadoop.

The size of data sets being collected and analyzed in the industry for business intelligence is growing and in a way, it is making traditional data warehousing solutions more expensive.[Hadoop](https://www.guru99.com/bigdata-tutorials.html)with MapReduce framework, is being used as an alternative solution for analyzing data sets with huge size. Though, Hadoop has proved useful for working on huge data sets, its MapReduce framework is very low level and it requires programmers to write custom programs which are hard to maintain and reuse. Hive comes here for rescue of programmers.

Hive provides SQL-like declarative language, called **HiveQL**, which is used for expressing queries. Using Hive-QL users associated with[SQL](https://www.guru99.com/sql.html)are able to perform data analysis very easily.

**Hive engine** compiles these queries into Map-Reduce jobs to be executed on Hadoop. In addition, custom Map-Reduce scripts can also be plugged into queries. Hive operates on data stored in tables which consists of primitive data types and collection data types like arrays and maps.

Hive comes with a command-line shell interface which can be used to create tables and execute queries.

## What is Hive

Apache Hive is a data warehouse system built on top of Hadoop and is used for analyzing structured and semi-structured data. Hive abstracts the complexity of Hadoop MapReduce. Basically, it provides a mechanism to project structure onto the data and perform queries written in HQL (Hive Query Language) that are similar to SQL statements. Internally, these queries or HQL gets converted to map reduce jobs by the Hive compiler. Therefore, you don’t need to worry about writing complex MapReduce programs to process your data using Hadoop. It is targeted towards users who are comfortable with SQL. Apache Hive supports Data Definition Language (DDL), Data Manipulation Language (DML) and User Defined Functions (UDF).

**SQL + Hadoop MapReduce = HiveQL**

Hive is not

A relational database

A design for OnLine Transaction Processing (OLTP)

A language for real-time queries and row-level updates

## Advantages of Hive

* Useful for people who aren’t from a programming background as it eliminates the need to write complex MapReduce program.
* **Extensible**and**scalable** to cope up with the growing volume and variety of data, without affecting performance of the system.
* It is as an efficient ETL (Extract, Transform, Load) tool.
* Hive supports any client application written in Java, PHP, Python, C++ or Ruby by exposing its **Thrift server**. (You can use these client – side languages embedded with SQL for accessing a database such as DB2, etc.).
* As the metadata information of Hive is stored in an RDBMS, it significantly reduces the time to perform semantic checks during query execution.

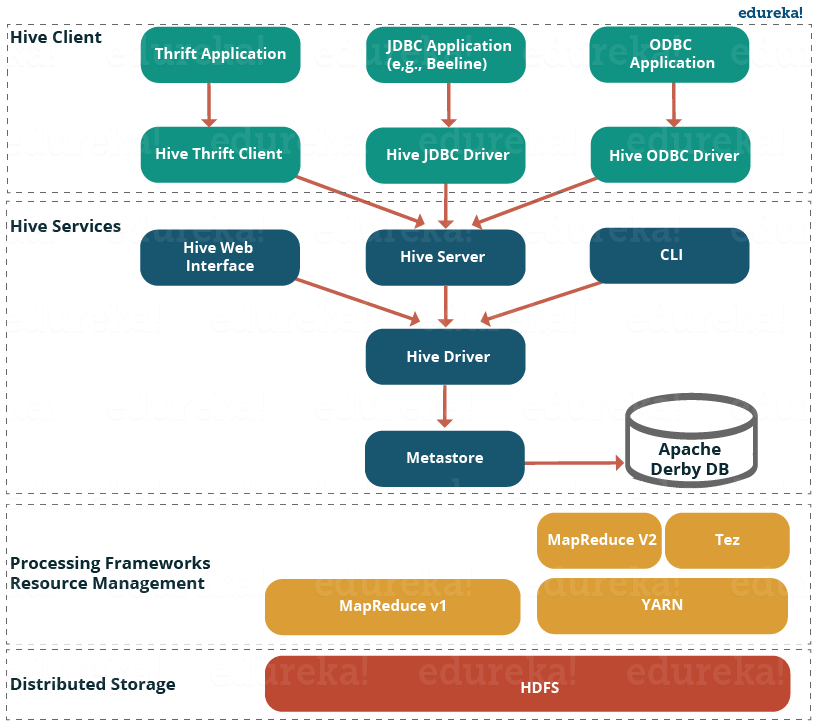
Apache Hive takes advantage of both the worlds i.e. SQL Database System and [***Hadoop – MapReduce***](https://www.edureka.co/blog/mapreduce-tutorial/)framework. Therefore, it is used by a vast multitude of companies. It is mostly used for data warehousing where you can perform analytics and data mining that does not require real time processing. Some of the fields where you can use Apache Hive are as follows:

* Data Warehousing
* Ad-hoc Analysis

## Features of Hive

* It stores schema in a database and processed data into HDFS.
* It is designed for OLAP.
* It provides SQL type language for querying called HiveQL or HQL.
* It is familiar, fast, scalable, and extensible.

## Architecture of Hive

The following image describes the Hive Architecture and the flow in which a query is submitted into Hive and finally processed using the MapReduce framework:

As shown in the above image, the Hive Architecture can be categorized into the following components:

* **Hive Clients:**Hive supports application written in many languages like Java, C++, Python etc. using JDBC, Thrift and ODBC drivers.  Hence one can always write hive client application written in a language of their choice.
* **Hive Services:** Apache Hive provides various services like CLI, Web Interface etc. to perform queries. We will explore each one of them shortly in this Hive tutorial blog.
* **Processing framework and Resource Management:**Internally, Hive uses Hadoop MapReduce framework as de facto engine to execute the queries. [***Hadoop MapReduce framework***](https://www.edureka.co/blog/mapreduce-tutorial/) is a separate topic in itself and therefore, is not discussed here.
* **Distributed Storage:** As Hive is installed on top of Hadoop, it uses the underlying HDFS for the distributed storage. You can refer to the [***HDFS blog***](https://www.edureka.co/blog/hdfs-tutorial) to learn more about it.

Now, let us explore the first two major components in the Hive Architecture:

### ****1. Hive Clients:****

Apache Hive supports different types of client applications for performing queries on the Hive. These clients can be categorized into three types:

* Thrift Clients: As Hive server is based on Apache Thrift, it can serve the request from all those programming language that supports Thrift.
* JDBC Clients: Hive allows Java applications to connect to it using the JDBC driver which is defined in the class org.apache.hadoop.hive.jdbc.HiveDriver.
* ODBC Clients: The Hive ODBC Driver allows applications that support the ODBC protocol to connect to Hive. (Like the JDBC driver, the ODBC driver uses Thrift to communicate with the Hive server.)

### ****2. Hive Services:****

Client interactions with Hive can be performed through Hive Services. If the client wants to perform any query related operations in Hive, it has to communicate through Hive Services.

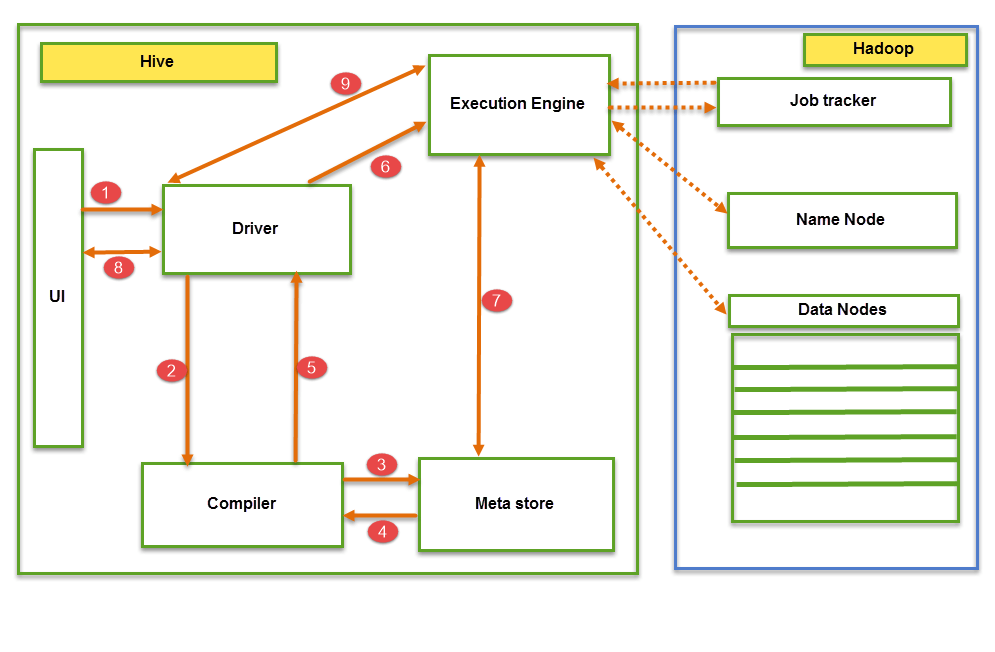
Hive provides many services as shown in the image above. Let us have a look at each of them:

* **Hive CLI (Command Line Interface):**This is the default shell provided by the Hive where you can execute your Hive queries and commands directly. CLI is the command line interface acts as Hive service for DDL (Data definition Language) operations
* **Apache Hive Web Interfaces:**Apart from the command line interface, Hive also provides a web based GUI for executing Hive queries and commands.
* **Hive Server:**Hive server is built on Apache Thrift and therefore, is also referred as Thrift Server that allows different clients to submit requests to Hive and retrieve the final result.
* **Apache Hive Driver:** All drivers (the queries submitted through the CLI, the web UI, Thrift, ODBC or JDBC interfaces by a client) communicate with Hive server and to the main driver in Hive services. Driver present in the Hive services represents the main driver. Driver will process those requests from different applications to meta store and field systems for further processing.

### ****3.Hive Storage and Computing:****

Hive services such as Meta store, File system, and Job Client in turn communicates with Hive storage and performs the following actions

* Metadata information of tables created in Hive is stored in Hive "Meta storage database".
* Query results and data loaded in the tables are going to be stored in Hadoop cluster on HDFS.



The data flow in Hive behaves in the following pattern;

1. Executing Query from the UI( User Interface)
2. The driver is interacting with Compiler for getting the plan. (Here plan refers to query execution) process and its related metadata information gathering
3. The compiler creates the plan for a job to be executed. Compiler communicating with Meta store for getting metadata request
4. Meta store sends metadata information back to compiler
5. Compiler communicating with Driver with the proposed plan to execute the query
6. Driver Sending execution plans to Execution engine
7. Execution Engine (EE) acts as a bridge between Hive and Hadoop to process the query. For DFS operations.

* EE should first contacts Name Node and then to Data nodes to get the values stored in tables.
* EE is going to fetch desired records from Data Nodes. The actual data of tables resides in data node only. While from Name Node it only fetches the metadata information for the query.
* It collects actual data from data nodes related to mentioned query
* Execution Engine (EE) communicates bi-directionally with Meta store present in Hive to perform DDL (Data Definition Language) operations. Here DDL operations like CREATE, DROP and ALTERING tables and databases are done. Meta store will store information about database name, table names and column names only. It will fetch data related to query mentioned.
* Execution Engine (EE) in turn communicates with Hadoop daemons such as Name node, Data nodes, and job tracker to execute the query on top of Hadoop file system

1. Fetching results from driver
2. Sending results to Execution engine. Once the results fetched from data nodes to the EE, it will send results back to driver and to UI ( front end)

Hive Continuously in contact with Hadoop file system and its daemons via Execution engine. The dotted arrow in the Job flow diagram shows the Execution engine communication with Hadoop daemons.

### Different modes of Hive

Hive can operate in two modes depending on the size of data nodes in Hadoop.

These modes are,

* **Local mode**
* **Map reduce mode**

**When to use Local mode:**

* If the Hadoop installed under pseudo mode with having one data node we use Hive in this mode
* If the data size is smaller in term of limited to single local machine, we can use this mode
* Processing will be very fast on smaller data sets present in the local machine

**When to use Map reduce mode:**

* If Hadoop is having multiple data nodes and data is distributed across different node we use Hive in this mode
* It will perform on large amount of data sets and query going to execute in parallel way
* Processing of large data sets with better performance can be achieved through this mode

In Hive, we can set this property to mention which mode Hive can work? By default, it works on Map Reduce mode and for local mode you can have the following setting.

Hive to work in local mode set

**SET mapred.job.tracker=local;**

Metastore:

You can think metastore as a central repository for storing all the Hive metadata information. Hive metadata includes various types of information like structure of tables and the partitions along with the column, column type, serializer and deserializer which is required for Read/Write operation on the data present in HDFS. The metastore comprises of two fundamental units:

* + A service that provides metastore access to other Hive services.
  + Disk storage for the metadata which is separate from HDFS storage.

Now, let us understand the different ways of implementing Hive metastore in the next section of this Hive Tutorial.

**Metastore Configuration**

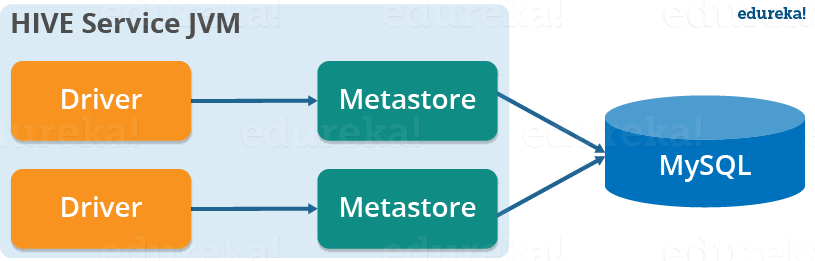
Metastore stores the meta data information using RDBMS and an open source ORM (Object Relational Model) layer called Data Nucleus which converts the object representation into relational schema and vice versa. The reason for choosing RDBMS instead of HDFS is to achieve low latency. We can implement metastore in following three configurations:

**1**. Embedded Metastore:



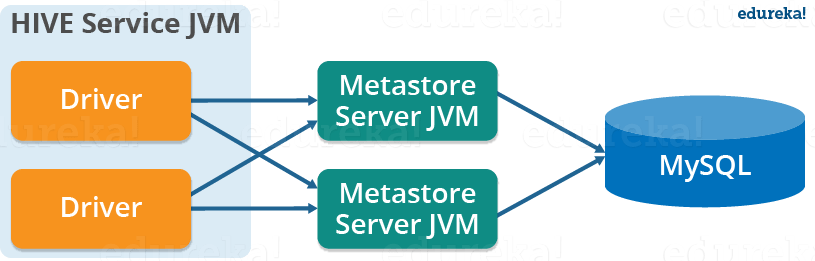
Both the metastore service and the Hive service runs in the same JVM by default using an embedded Derby Database instance where metadata is stored in the local disk. This is called embedded metastore configuration. In this case, only one user can connect to metastore database at a time. If you start a second instance of Hive driver, you will get an error. This is good for unit testing, but not for the practical solutions.

**2. Local Metastore:**



This configuration allows us to have multiple Hive sessions i.e. Multiple users can use the metastore database at the same time. This is achieved by using any JDBC compliant database like MySQL which runs in a separate JVM or a different machine than that of the Hive service and metastore service which are running in the same JVM as shown above. In general, the most popular choice is to implement a MySQL server as the metastore database.

**3. Remote Metastore:**



In the remote metastore configuration, the metastore service runs on its own separate JVM and not in the Hive service JVM. Other processes communicate with the metastore server using Thrift Network APIs. You can have one or more metastore servers in this case to provide more availability. The main advantage of using remote metastore is you do not need to share JDBC login credential with each Hive user to access the metastore database.

## HIVE INSTALLATION

## ****Data Model****

Data in Hive can be categorized into three types on the granular level:

* Table
* Partition
* Bucket

### ****Tables:****

Tables in Hive are the same as the tables present in a Relational Database. You can perform filter, project, join and union operations on them. There are two types of tables in Hive:

**1. Managed Table:**

|  |
| --- |
| CREATE TABLE <table\_name> (column1 data\_type, column2 data\_type);  LOAD DATA INPATH <HDFS\_file\_location> INTO table managed\_table; |

As the name suggests (managed table), Hive is responsible for managing the data of a managed table. In other words, what I meant by saying, “Hive manages the data”, is that if you load the data from a file present in HDFS into a Hive Managed Table and issue a DROP command on it, the table along with its metadata will be deleted. So, the data belonging to the dropped managed\_table no longer exist anywhere in HDFS and you can’t retrieve it by any means. Basically, you are moving the data when you issue the LOAD command from the HDFS file location to the Hive warehouse directory.

**Note:** The default path of the warehouse directory is set to/user/hive/warehouse. The data of a Hive table resides in warehouse\_directory**/**table\_name (HDFS). You can also specify the path of the warehouse directory in the hive.metastore.warehouse.dir configuration parameter present in the hive-site.xml.

**2. External Table:**

|  |
| --- |
| CREATE EXTERNAL TABLE <table\_name> (column1 data\_type, column2 data\_type) LOCATION ‘<table\_hive\_location>’;  LOAD DATA INPATH ‘<HDFS\_file\_location>’ INTO TABLE <table\_name>; |

For external table, Hive is not responsible for managing the data. In this case, when you issue the LOAD command, Hive moves the data into its warehouse directory. Then, Hive creates the metadata information for the external table. Now, if you issue a DROP command on the external table, only metadata information regarding the external table will be deleted. Therefore, you can still retrive the data of that very external table from the warehouse directory using HDFS commands.

### ****Partitions:****

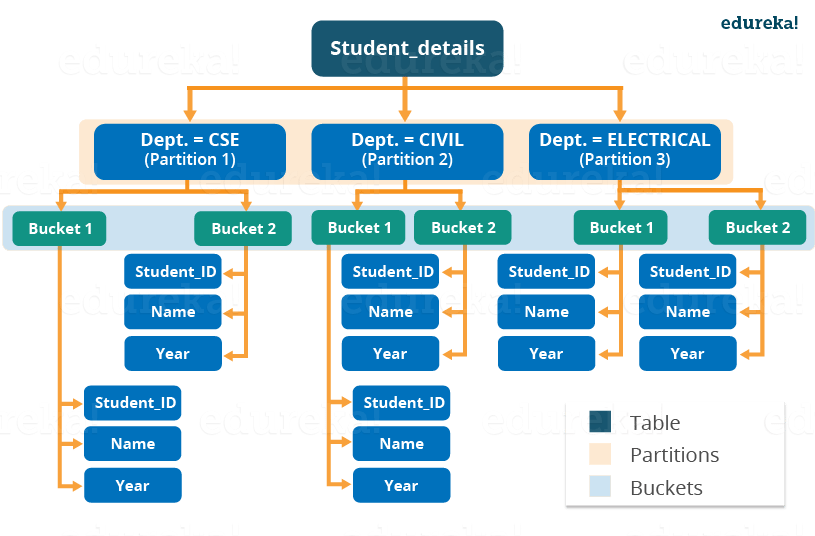
|  |
| --- |
| CREATE TABLE table\_name (column1 data\_type, column2 data\_type) PARTITIONED BY (partition1 data\_type, partition2 data\_type,….); |

Hive organizes tables into partitions for grouping similar type of data together based on a column or partition key. Each Table can have one or more partition keys to identify a particular partition. This allows us to have a faster query on slices of the data.

***Note:***Remember, the most common mistake made while creating partitions is to specify an existing column name as a partition column. While doing so, you will receive an error – “Error in semantic analysis: Column repeated in partitioning columns”.

Let us understand partition by taking an example where I have a table student\_details containing the student information of some engineering college like student\_id, name, department, year, etc. Now, if I perform partitioning based on department column, the information of all the students belonging to a particular department will be stored together in that very partition. Physically, a partition is nothing but a sub-directory in the table directory.

Let’s say we have data for three departments in our student\_details table – CSE, ECE and Civil. Therefore, we will have three partitions in total for each of the departments as shown in the image below. And, for each department we will have all the data regarding that very department residing in a separate sub – directory under the Hive table directory. For example, all the student data regarding CSE departments will be stored in user/hive/warehouse/student\_details/dept.=CSE. So, the queries regarding CSE students would only have to look through the data present in the CSE partition. This makes partitioning very useful as it reduces the query latency by scanning only **relevant** partitioned data instead of the whole data set. In fact, in real world implementations, you will be dealing with hundreds of TBs of data. So, imagine scanning this huge amount of data for some query where **95%** data scanned by you was un-relevant to your query.  



### ****Buckets:****

|  |
| --- |
| CREATE TABLE table\_name PARTITIONED BY (partition1 data\_type, partition2 data\_type,….) CLUSTERED BY (column\_name1, column\_name2, …) SORTED BY (column\_name [ASC|DESC], …)] INTO num\_buckets BUCKETS; |

Now, you may divide each partition or the unpartitioned table into Buckets based on the hash function of a column in the table. Actually, each bucket is just a file in the partition directory or the table directory (unpartitioned table). Therefore, if you have chosen to divide the partitions into n buckets, you will have n files in each of your partition directory. For example, you can see the above image where we have bucketed each partition into 2 buckets. So, each partition, say CSE, will have two files where each of them will be storing the CSE student’s data.