**BIG DATA AND HADOOP**

**J COMPONENET**

**GROUP MEMBERS:-**

**Rutanshu Jhaveri 15BCE2016**

**Paul Mathai 15BCE0981**

**Rahul Raghunadhan 15BCE0987**

**Venika Anand 15BCE0998**

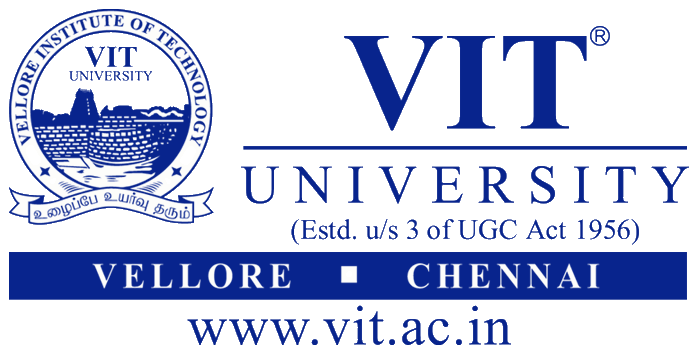
**Ujjwal Khanna 15BCE0658**

Course Code : CSE4001

Course Title : Parallel And Distributed Computing

Under the guidance of

**Prof.Manoov R**

****

**Department of**

**School of Computer Science**

**And Engineering**

**HADOOP ECOSYSTEM**

The Hadoop ecosystem includes both official Apache open source projects and a wide range of commercial tools and solutions. Some of the best-known open source examples include Spark, Hive, Pig, Oozie and Sqoop. Commercial Hadoop offerings are even more diverse and include platforms and packaged distributions from vendors such as Cloudera, Hortonworks, and MapR, plus a variety of tools for specific Hadoop development, production, and maintenance tasks.

Most of the solutions available in the Hadoop ecosystem are intended to supplement one or two of Hadoop’s four core elements (HDFS, MapReduce, YARN, and Common). However, the commercially available framework solutions provide more comprehensive functionality. The sections below provide a closer look at some of the more prominent components of the Hadoop ecosystem, starting with the Apache projects.



## **Apache open source Hadoop ecosystem elements**

The Apache Hadoop project actively supports multiple projects intended to extend Hadoop’s capabilities and make it easier to use. There are several top-level projects to create development tools as well as for managing Hadoop data flow and processing. Many commercial third-party solutions build on the technologies developed within the Apache Hadoop ecosystem.

Spark, Pig, and Hive are three of the best-known Apache Hadoop projects. Each is used to create applications to process Hadoop data. While there are a lot of articles and discussions about whether Spark, Hive or Pig is better, in practice many organizations do not only use a single one because each is optimized for specific functions.

#### [**Spark**](http://www.bmcsoftware.in/guides/hadoop-apache-spark.html)

Spark is both a programming model and a computing model. It provides a gateway to in-memory computing for Hadoop, which is a big reason for its popularity and wide adoption. Spark provides an alternative to MapReduce that enables workloads to execute in memory, instead of on disk. Spark accesses data from HDFS but bypasses the MapReduce processing framework, and thus eliminates the resource-intensive disk operations that MapReduce requires. By using in-memory computing, Spark workloads typically run between 10 and 100 times faster compared to disk execution.

Spark can be used independently of Hadoop. However, it is used most commonly with Hadoop as an alternative to MapReduce for data processing. Spark can easily coexist with MapReduce and with other ecosystem components that perform other tasks.

Spark is also popular because it supports SQL, which helps overcome a shortcoming in core Hadoop technology. The Spark programming environment works interactively with Scala, Python, and R shells. It has been used for data extract/transform/load (ETL) operations, stream processing, machine learning development and with the Apache GraphX API for graph computation and display. Spark can run on a variety of Hadoop and non-Hadoop clusters, including Amazon S3.

#### **Hive**

Hive is data warehousing software that addresses how data is structured and queried in distributed Hadoop clusters. Hive is also a popular development environment that is used to write queries for data in the Hadoop environment. It provides tools for ETL operations and brings some SQL-like capabilities to the environment. Hive is a declarative language that is used to develop applications for the Hadoop environment, however it does not support real-time queries.

Hive has several components, including:

* HCatalog – Helps data processing tools read and write data on the grid. It supports MapReduce and Pig.
* WebHCat – Lets you use an HTTP/REST interface to run MapReduce, Yarn, Pig, and Hive jobs.
* HiveQL – Hive’s query language intended as a way for SQL developers to easily work in Hadoop. It is similar to SQL and helps both structure and query data in distributed Hadoop clusters.

Hive queries can run from the Hive shell, JDBC, or ODBC. MapReduce (or an alternative) breaks down HiveQL statements for execution across the cluster.

Hive also allows MapReduce-compatible mapping and reduction software to perform more sophisticated functions. However, Hive does not allow row-level updates or support for real-time queries, and it is not intended for OLTP workloads. Many consider Hive to be much more effective for processing structured data than unstructured data, for which Pig is considered advantageous.

#### [**Pig**](http://www.bmcsoftware.in/guides/hadoop-apache-pig.html)

Pig is a procedural language for developing parallel processing applications for large data sets in the Hadoop environment. Pig is an alternative to Java programming for MapReduce, and automatically generates MapReduce functions. Pig includes Pig Latin, which is a scripting language. Pig translates Pig Latin scripts into MapReduce, which can then run on YARN and process data in the HDFS cluster. Pig is popular because it automates some of the complexity in MapReduce development.

Pig is commonly used for complex use cases that require multiple data operations. It is more of a processing language than a query language. Pig helps develop applications that aggregate and sort data and supports multiple inputs and exports. It is highly customizable, because users can write their own functions using their preferred scripting language. Ruby, Python and even Java are all supported. Thus, Pig has been a popular option for developers that are familiar with those languages but not with MapReduce. However, SQL developers may find Hive easier to learn.

#### [**HBase**](http://www.bmcsoftware.in/guides/hadoop-hbase.html)

HBase is a scalable, distributed, NoSQL database that sits atop the HFDS. It was designed to store structured data in tables that could have billions of rows and millions of columns. It has been deployed to power historical searches through large data sets, especially when the desired data is contained within a large amount of unimportant or irrelevant data (also known as sparse data sets). It is also an underlying technology behind several large messaging applications, including Facebook’s.

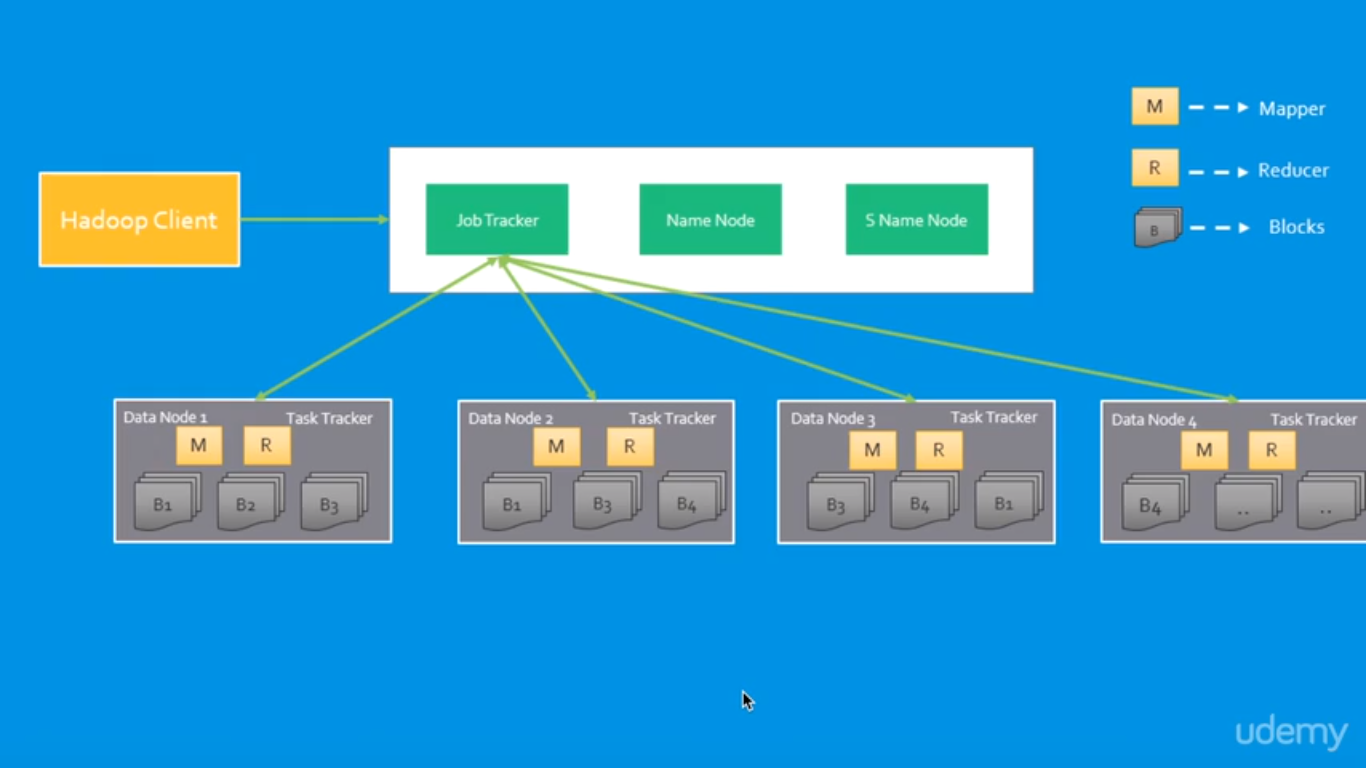
HBase is not a relational database and wasn’t designed to support transactional and other real-time applications. It is accessible through a Java API and has ODBC and JDBC drivers. HBase does not support SQL queries, however there are several SQL support tools available from the Apache project and from software vendors. For example, Hive can be used to run SQL-like queries in HBase.

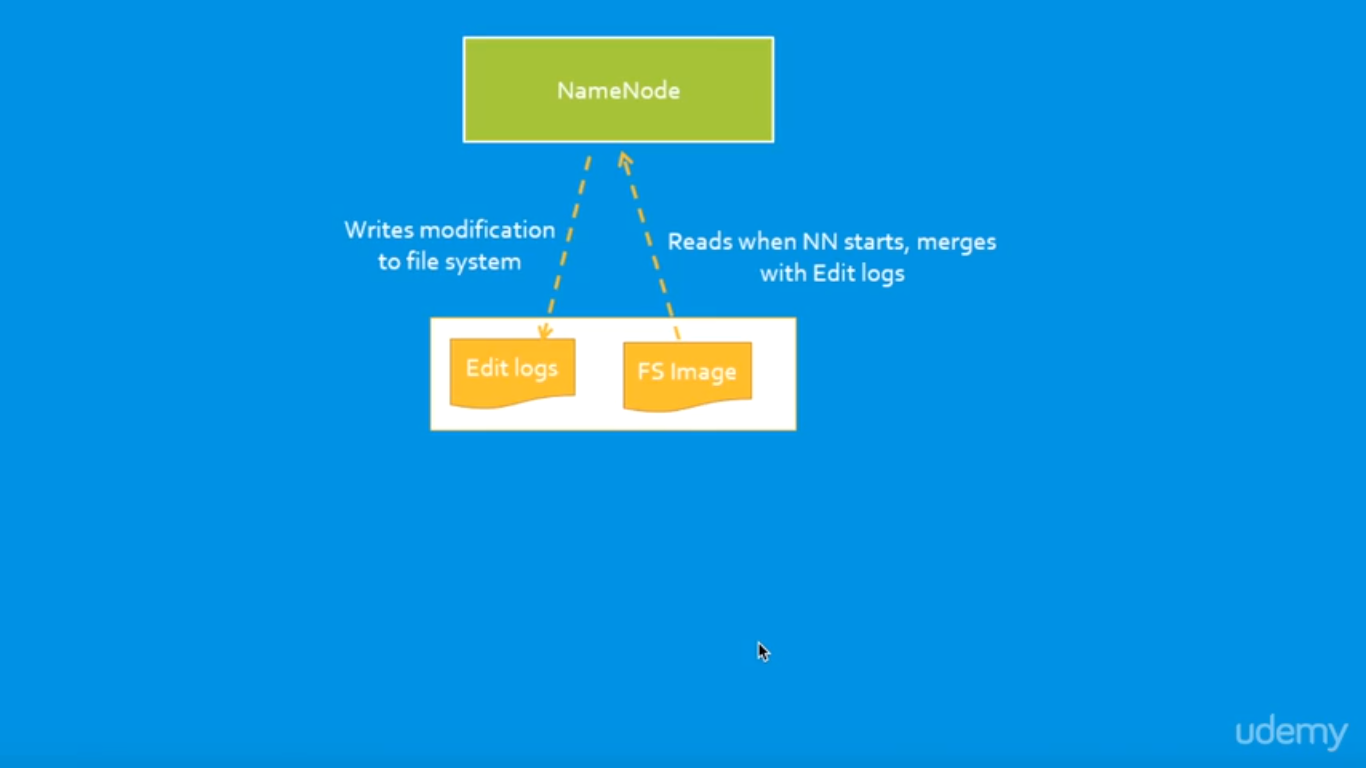
#### **Oozie**

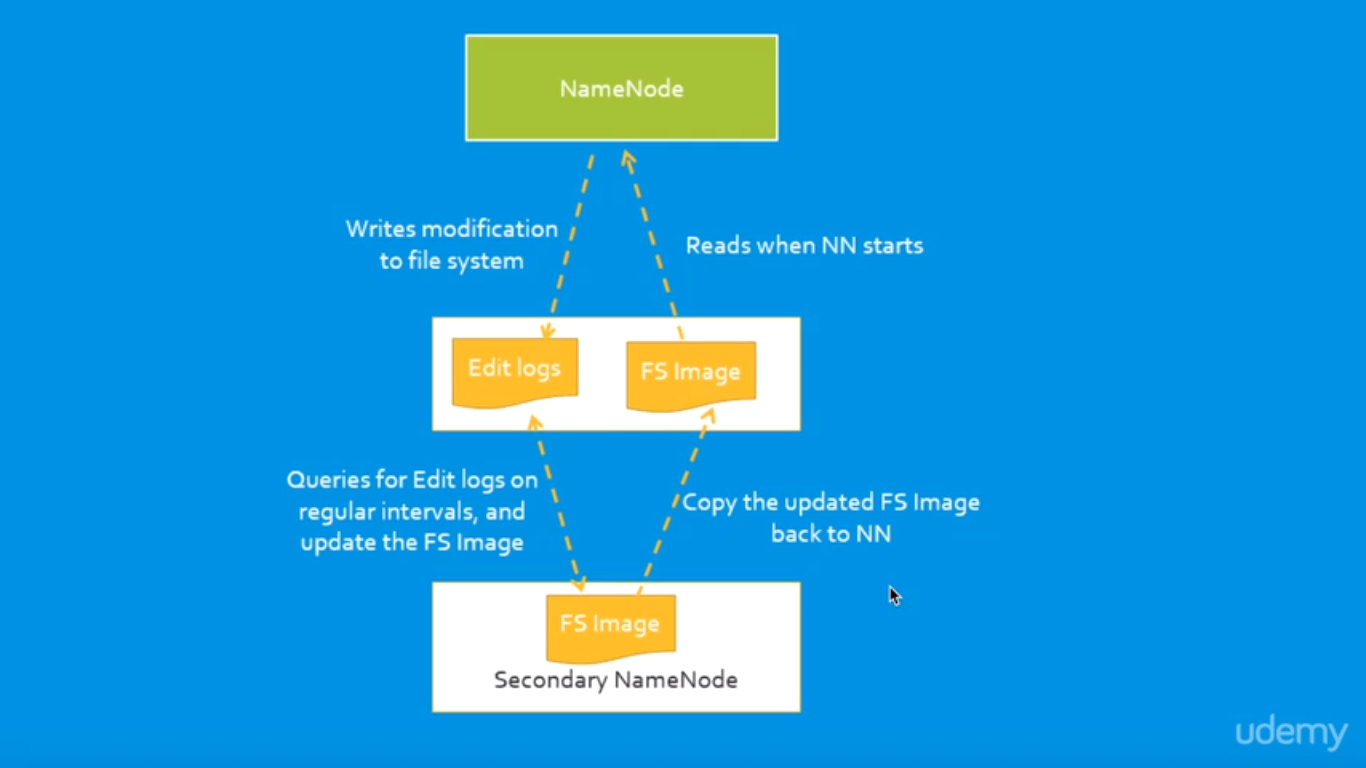
Oozie is the workflow scheduler that was developed as part of the Apache Hadoop project. It manages how workflows start and execute, and also controls the execution path. Oozie is a server-based Java web application that uses workflow definitions written in hPDL, which is an XML Process Definition Language similar to [JBOSS JBPM](http://www.jboss.org/jbossjbpm/) jPDL. Oozie only supports specific workflow types, so other workload schedulers are commonly used instead of or in addition to Oozie in Hadoop environments.

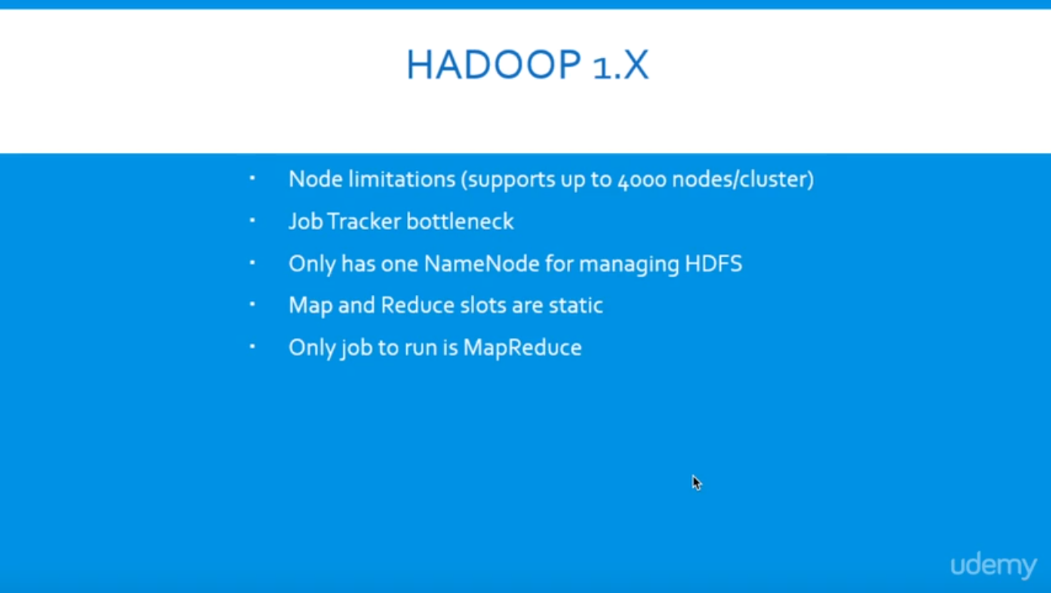
#### **Sqoop**

Think of Sqoop as a front-end loader for big data. Sqoop is a command-line interface that facilitates moving bulk data from Hadoop into relational databases and other structured data stores. Using Sqoop replaces the need to develop scripts to export and import data. One common use case is to move data from an enterprise data warehouse to a Hadoop cluster for ETL processing. Performing ETL on the commodity Hadoop cluster is resource efficient, while Sqoop provides a practical transfer method.



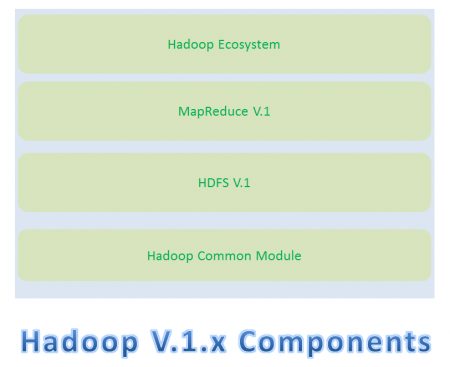




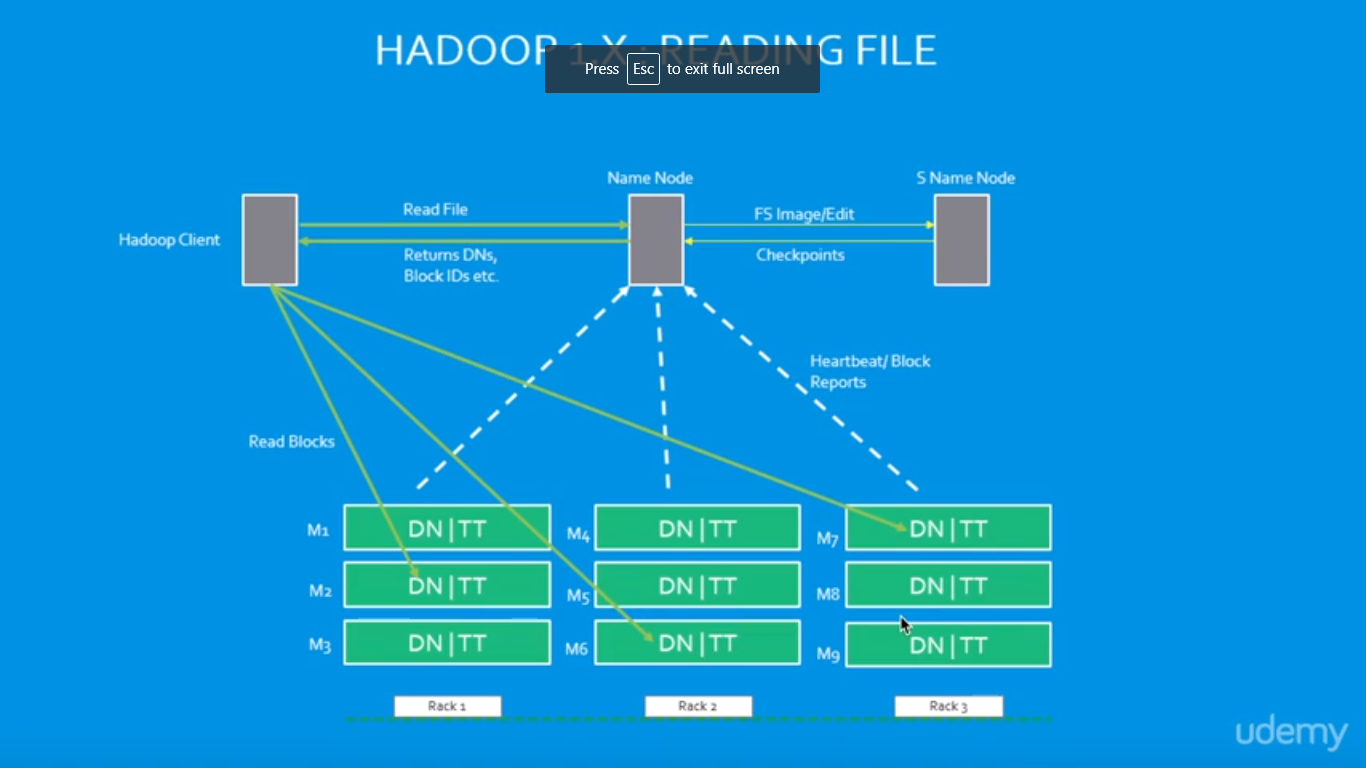
**HADOOP 1.X**

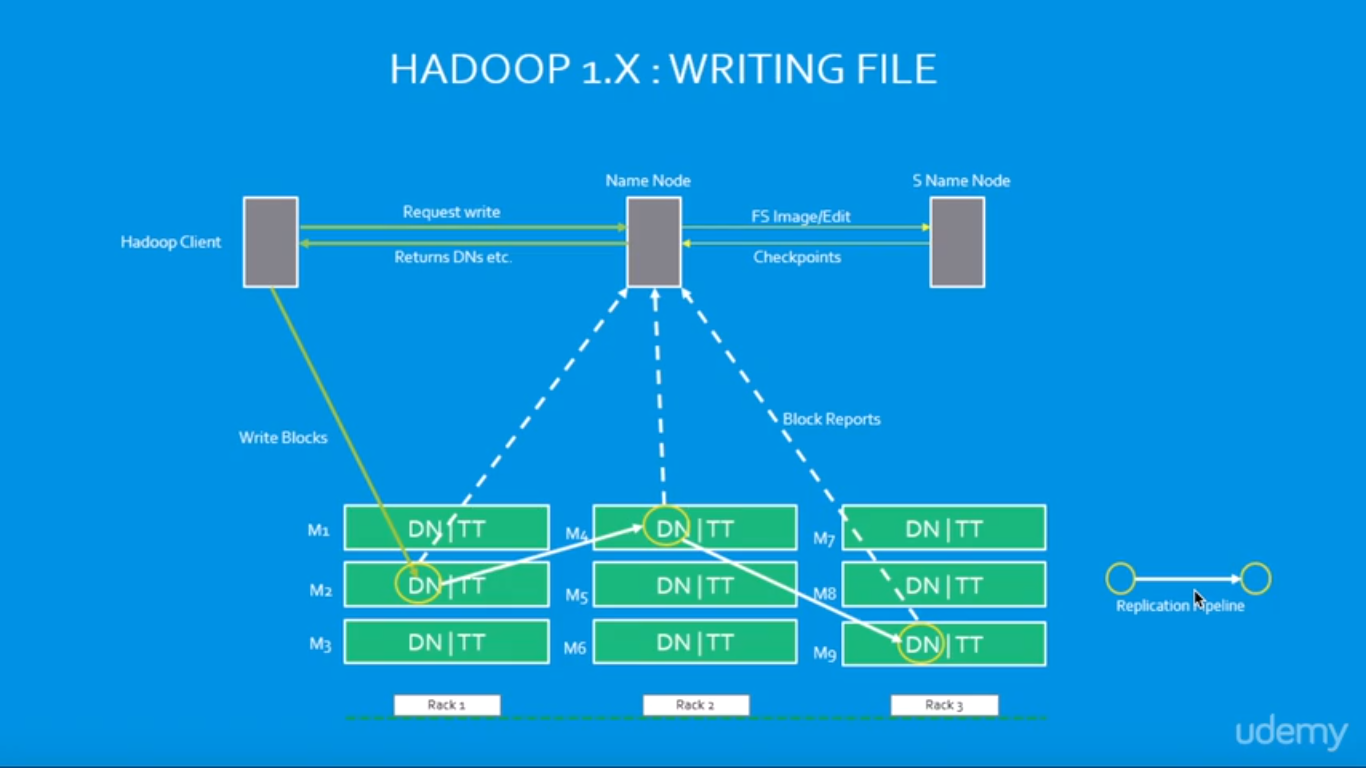
### **Hadoop 1.x Architecture**

Apache Hadoop 1.x or earlier versions are using the following Hadoop Architecture. It is a Hadoop 1.x High-level Architecture. We will discuss in-detailed Low-level Architecture in coming sections.

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop1.x-components.png)If you don’t understand this Architecture at this stage, no need to worry. Read next sections in this post and also coming posts to understand it very well.

* Hadoop Common Module is a Hadoop Base API (A Jar file) for all Hadoop Components. All other components works on top of this module.
* HDFS stands for Hadoop Distributed File System. It is also know as HDFS V1 as it is part of Hadoop 1.x. It is used as a Distributed Storage System in Hadoop Architecture.
* MapReduce is a Batch Processing or Distributed Data Processing Module. It is built by following Google’s MapReduce Algorithm. It is also know as “MR V1” or “Classic MapReduce” as it is part of Hadoop 1.x.
* Remaining all Hadoop Ecosystem components work on top of these two major components: HDFS and MapReduce. We will discuss all Hadoop Ecosystem components in-detail in my coming posts.





### **Hadoop 1.x Major Components**

* Hadoop 1.x Major Components components are: HDFS and MapReduce. They are also know as “Two Pillars” of Hadoop 1.x.

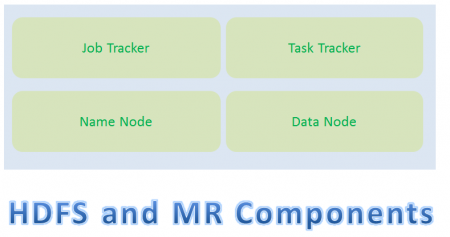
**HDFS:**  
HDFS is a Hadoop Distributed FileSystem, where our BigData is stored using Commodity Hardware. It is designed to work with Large DataSets with default block size is 64MB (We can change it as per our Project requirements).

* HDFS component is again divided into two sub-components:
* Name Node
* Name Node is placed in Master Node. It used to store Meta Data about Data Nodes like “How many blocks are stored in Data Nodes, Which Data Nodes have data, Slave Node Details, Data Nodes locations, timestamps etc” .
* Data Node
* Data Nodes are places in Slave Nodes. It is used to store our Application Actual Data. It stores data in Data Slots of size 64MB by default.

**MapReduce:**  
MapReduce is a Distributed Data Processing or Batch Processing Programming Model. Like HDFS, MapReduce component also uses

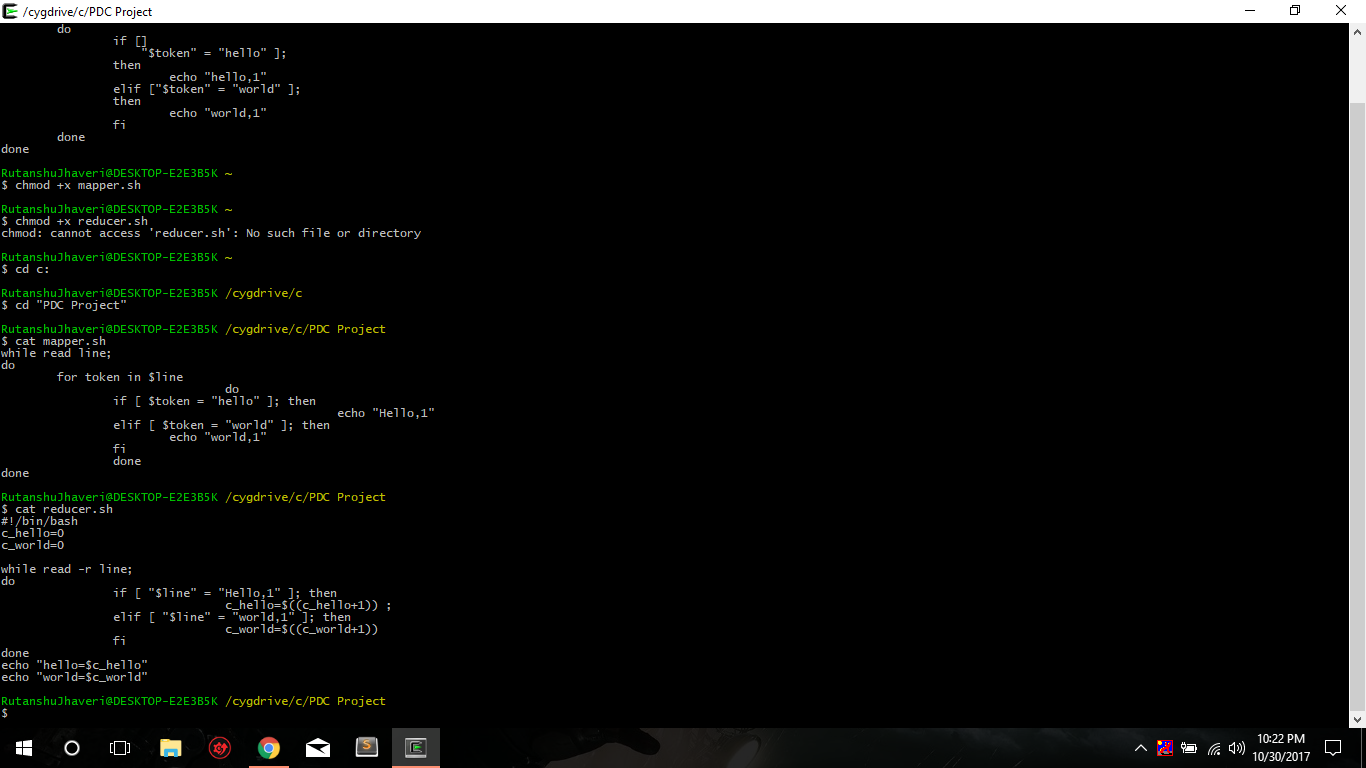
Commodity Hardware to process “High Volume of Variety of Data at High Velocity Rate” in a reliable and fault-tolerant manner.

* MapReduce component is again divided into two sub-components:
* Job Tracker
* Job Tracker is used to assign MapReduce Tasks to Task Trackers in the Cluster of Nodes. Sometimes, it reassigns same tasks to other Task Trackers as previous Task Trackers are failed or shutdown scenarios.
* Job Tracker maintains all the Task Trackers status like Up/running, Failed, Recovered etc.
* Task Tracker
* Task Tracker executes the Tasks which are assigned by Job Tracker and sends the status of those tasks to Job Tracker.

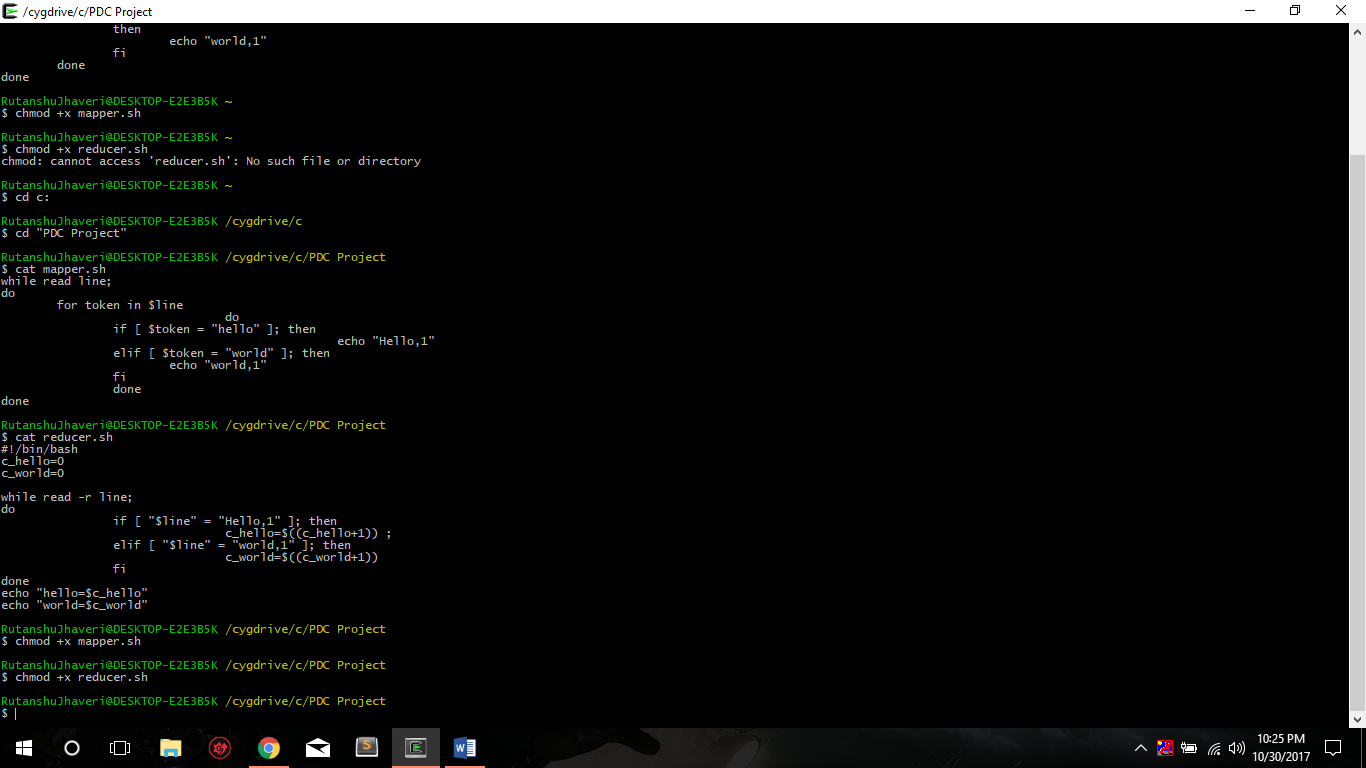
[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop1.x-hdfs-mr-components.png)

* We will discuss these four sub-component’s responsibilities and how they interact each other to perform a “Client Application Tasks” in detail in next section.
* How Hadoop 1.x Major Components Works
* Hadoop 1.x components follow this architecture to interact each other and to work parallel in a reliable and fault-tolerant manner.

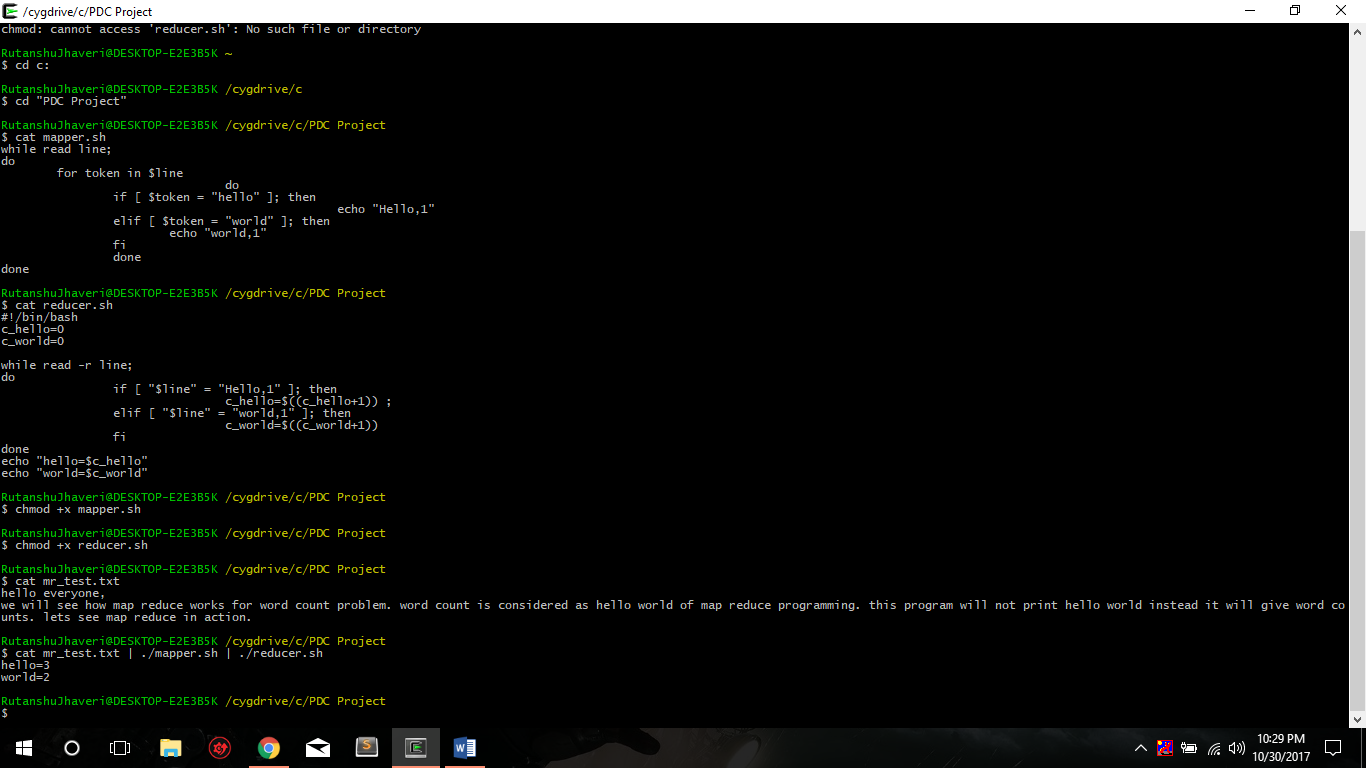
**Implementation Example:-**



Mapper and reducer shell scripts to count the words “hello” and “world”



**Giving permission to file of shell scripts**



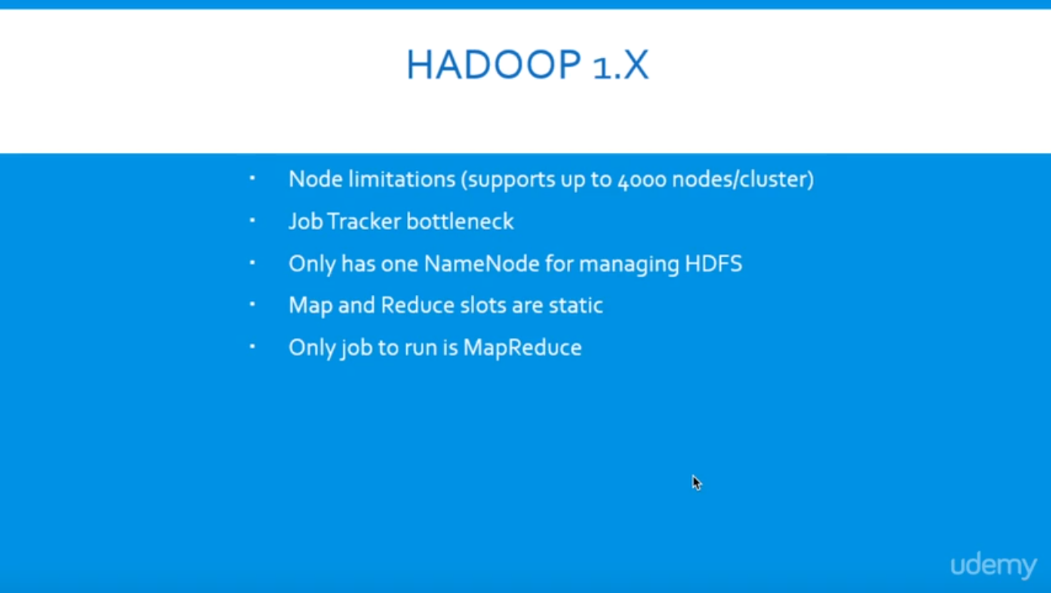
**Project is on Virtual Box**

We can show the code pig and hive files.

Therefore, there is no directory to show, The final project would be shown in video of Review 3.

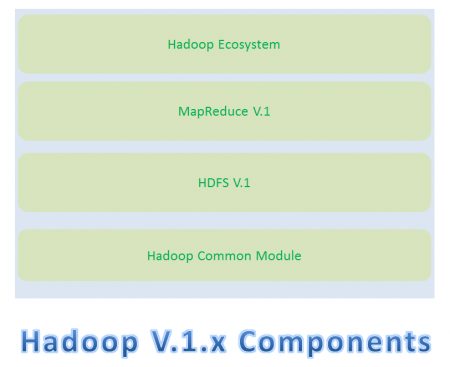
https://github.com/rutanshuj/PDCProject

**HADOOP 1.X**

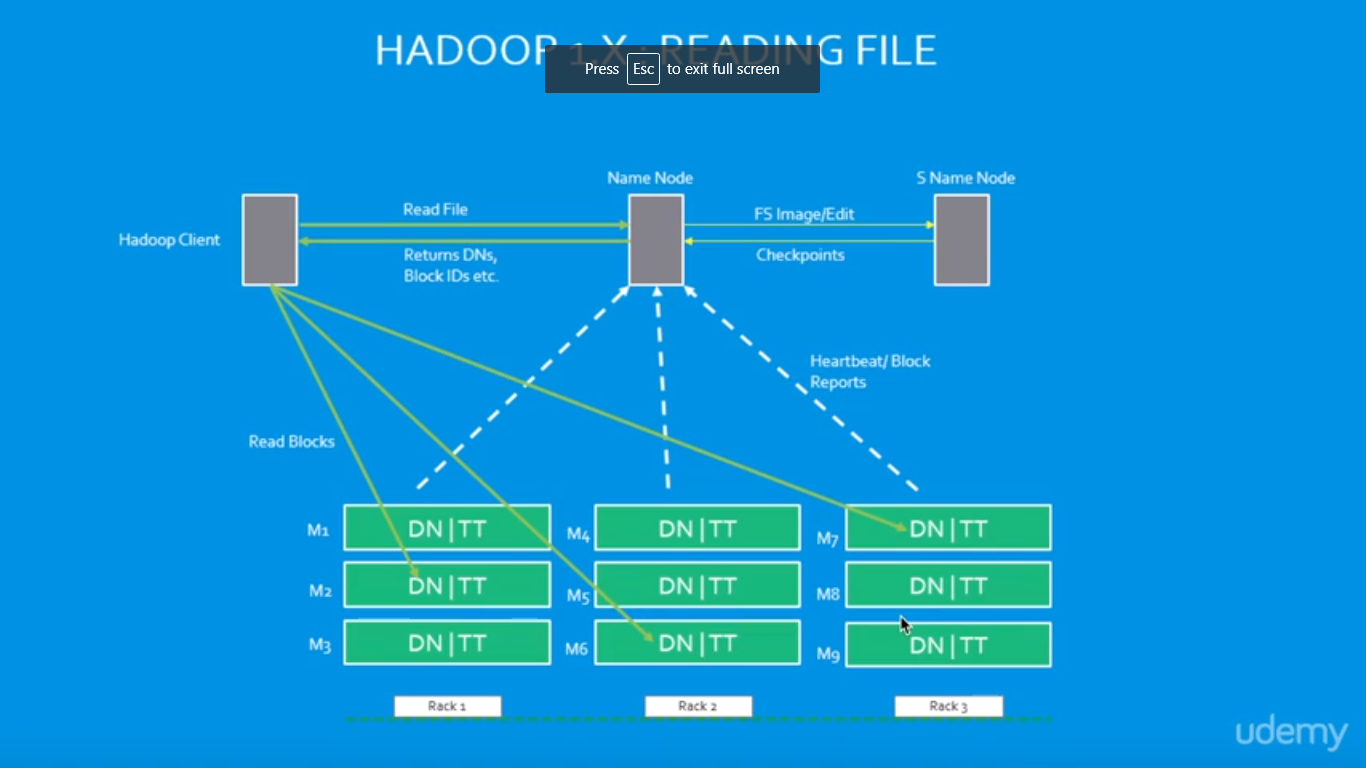


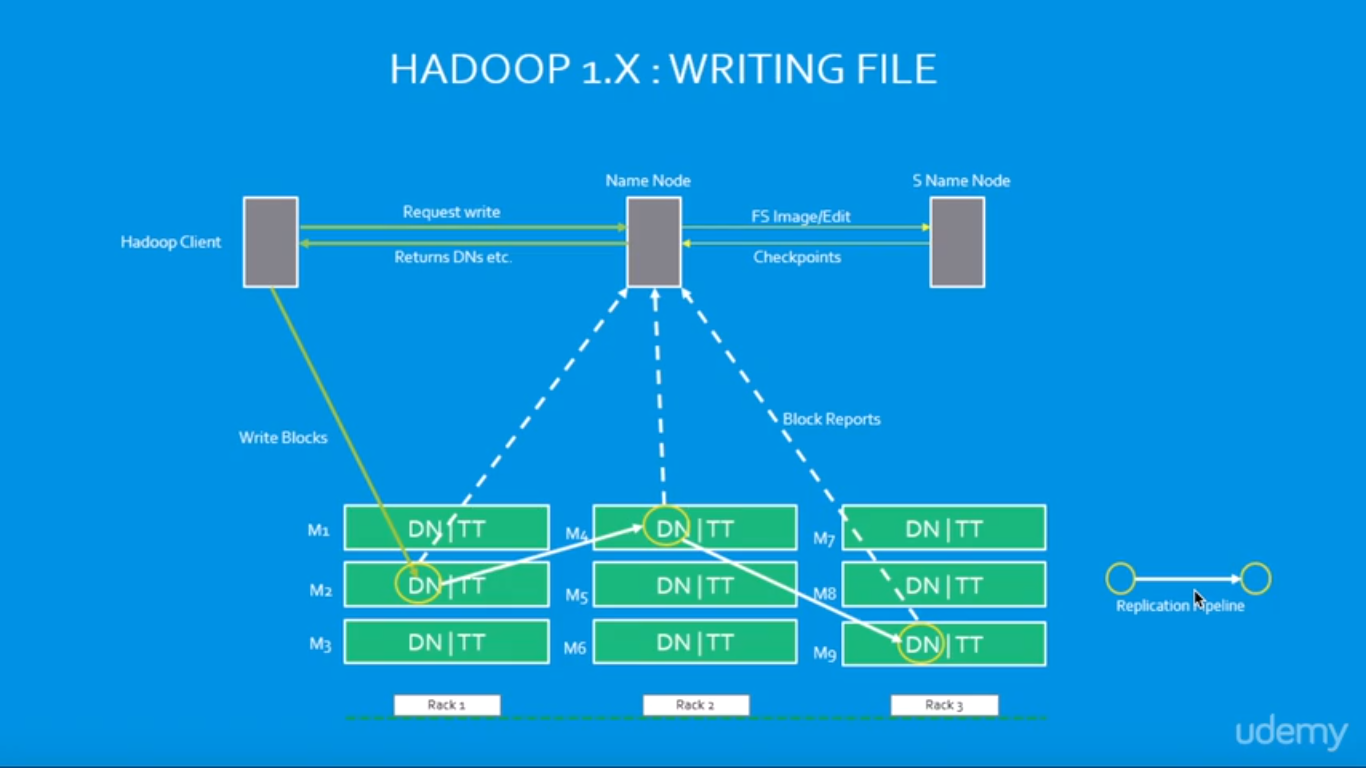
### **Hadoop 1.x Architecture**

Apache Hadoop 1.x or earlier versions are using the following Hadoop Architecture. It is a Hadoop 1.x High-level Architecture. We will discuss in-detailed Low-level Architecture in coming sections.

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop1.x-components.png)If you don’t understand this Architecture at this stage, no need to worry. Read next sections in this post and also coming posts to understand it very well.

* Hadoop Common Module is a Hadoop Base API (A Jar file) for all Hadoop Components. All other components works on top of this module.
* HDFS stands for Hadoop Distributed File System. It is also know as HDFS V1 as it is part of Hadoop 1.x. It is used as a Distributed Storage System in Hadoop Architecture.
* MapReduce is a Batch Processing or Distributed Data Processing Module. It is built by following Google’s MapReduce Algorithm. It is also know as “MR V1” or “Classic MapReduce” as it is part of Hadoop 1.x.
* Remaining all Hadoop Ecosystem components work on top of these two major components: HDFS and MapReduce. We will discuss all Hadoop Ecosystem components in-detail in my coming posts.





### Hadoop 1.x Major Components

* Hadoop 1.x Major Components components are: HDFS and MapReduce. They are also know as “Two Pillars” of Hadoop 1.x.

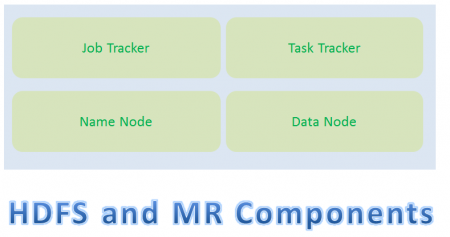
**HDFS:**  
HDFS is a Hadoop Distributed FileSystem, where our BigData is stored using Commodity Hardware. It is designed to work with Large DataSets with default block size is 64MB (We can change it as per our Project requirements).

* HDFS component is again divided into two sub-components:
* Name Node
* Name Node is placed in Master Node. It used to store Meta Data about Data Nodes like “How many blocks are stored in Data Nodes, Which Data Nodes have data, Slave Node Details, Data Nodes locations, timestamps etc” .
* Data Node
* Data Nodes are places in Slave Nodes. It is used to store our Application Actual Data. It stores data in Data Slots of size 64MB by default.

**MapReduce:**  
MapReduce is a Distributed Data Processing or Batch Processing Programming Model. Like HDFS, MapReduce component also uses

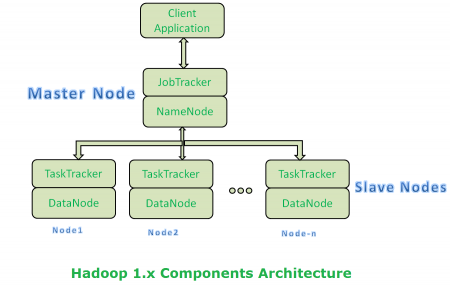
Commodity Hardware to process “High Volume of Variety of Data at High Velocity Rate” in a reliable and fault-tolerant manner.

* MapReduce component is again divided into two sub-components:
* Job Tracker
* Job Tracker is used to assign MapReduce Tasks to Task Trackers in the Cluster of Nodes. Sometimes, it reassigns same tasks to other Task Trackers as previous Task Trackers are failed or shutdown scenarios.
* Job Tracker maintains all the Task Trackers status like Up/running, Failed, Recovered etc.
* Task Tracker
* Task Tracker executes the Tasks which are assigned by Job Tracker and sends the status of those tasks to Job Tracker.

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop1.x-hdfs-mr-components.png)

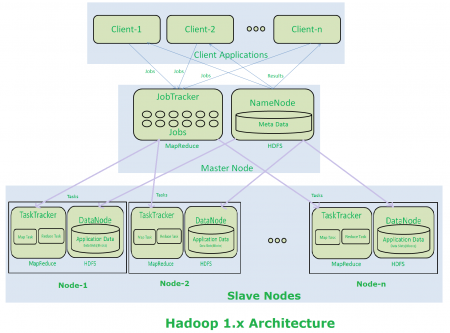
* We will discuss these four sub-component’s responsibilities and how they interact each other to perform a “Client Application Tasks” in detail in next section.
* How Hadoop 1.x Major Components Works
* Hadoop 1.x components follow this architecture to interact each other and to work parallel in a reliable and fault-tolerant manner.

**Hadoop 1.x Components High-Level Architecture**

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop1.x-components-architecture.png)

* Both Master Node and Slave Nodes contain two Hadoop Components:
* HDFS Component
* MapReduce Component
* Master Node’s HDFS component is also known as “Name Node”.
* Slave Node’s HDFS component is also known as “Data Node”.
* Master Node’s “Name Node” component is used to store Meta Data.
* Slave Node’s “Data Node” component is used to store actual our application Big Data.
* HDFS stores data by using 64MB size of “Data Slots” or “Data Blocks”.
* Master Node’s MapReduce component is also known as “Job Tracker”.
* Slave Node’s MapReduce component is also known as “Task Tracker”.
* Master Node’s “Job Tracker” will take care assigning tasks to “Task Tracker” and receiving results from them.
* Slave Node’s MapReduce component “Task Tracker” contains two MapReduce Tasks:
* Map Task
* Reduce Task
* We will discuss in-detail about MapReduce tasks (Mapper and Reducer) in my coming post with some simple End-to-End Examples.
* Slave Node’s “Task Tracker” actually performs Client’s tasks by using MapReduce Batch Processing model.
* Master Node is a Primary Node to take care of all remaining Slave Nodes (Secondary Nodes)

**Hadoop 1.x Components In-detail Architecture**

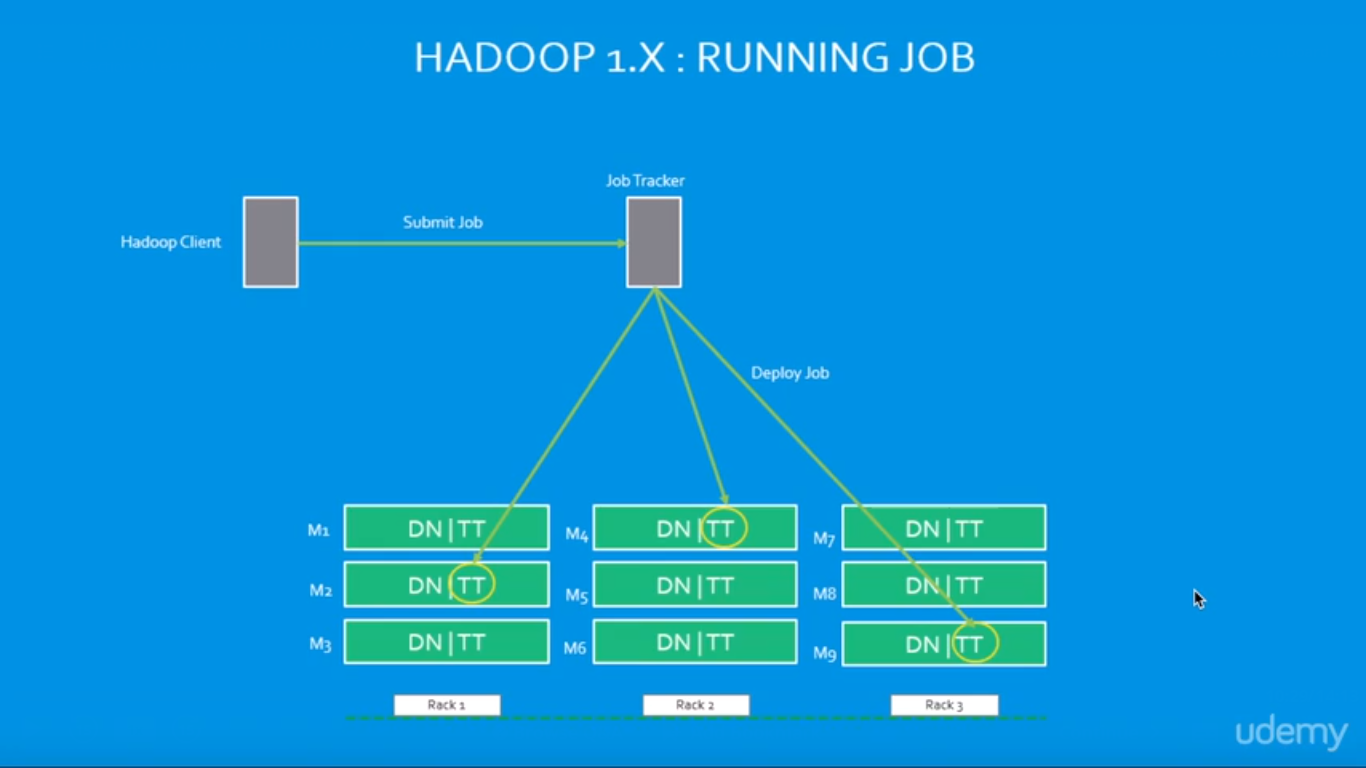
[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop2.x-components-architecture.png)

**Hadoop 1.x Architecture Description**

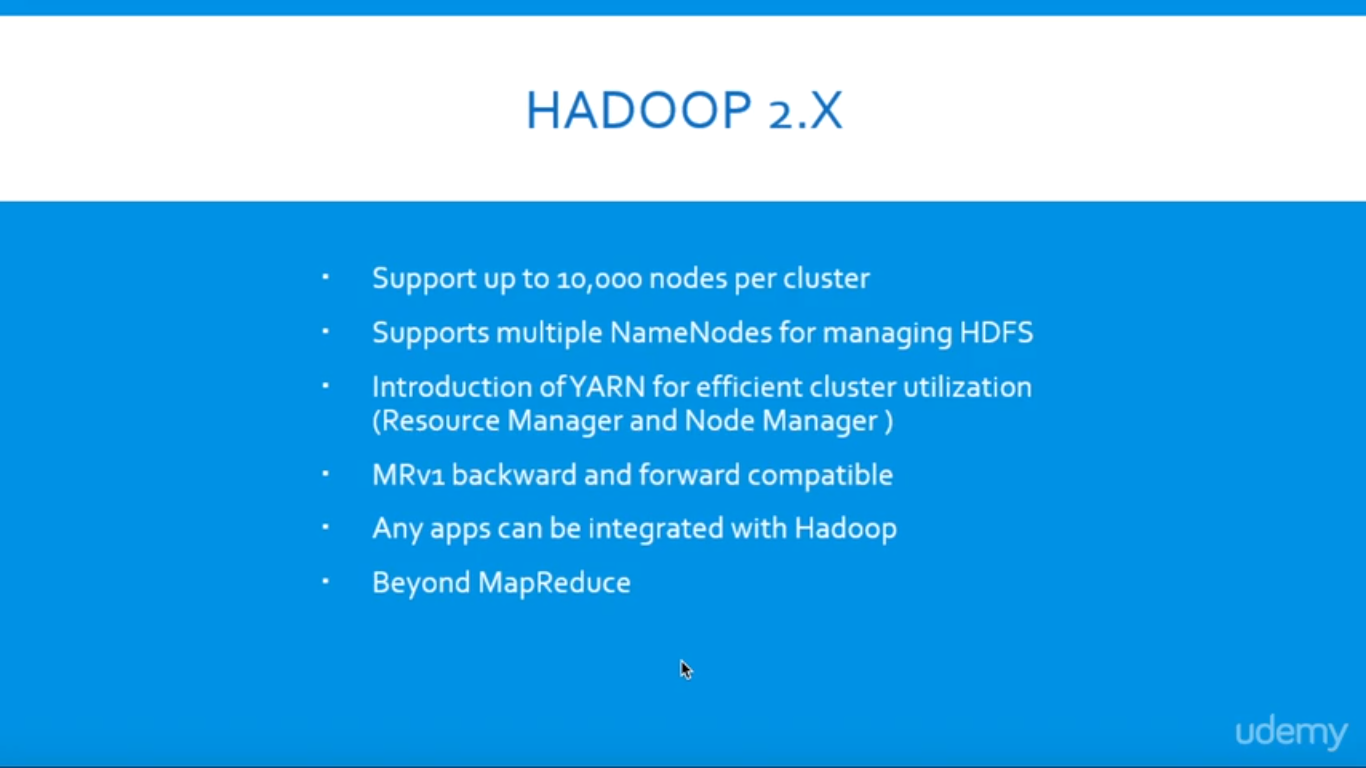
* Clients (one or more) submit their work to Hadoop System.
* When Hadoop System receives a Client Request, first it is received by a Master Node.
* Master Node’s MapReduce component “Job Tracker” is responsible for receiving Client Work and divides into manageable independent Tasks and assign them to Task Trackers.
* Slave Node’s MapReduce component “Task Tracker” receives those Tasks from “Job Tracker” and perform those tasks by using MapReduce components.
* Once all Task Trackers finished their job, Job Tracker takes those results and combines them into final result.
* Finally Hadoop System will send that final result to the Client.
* How Store and Compute Operations Work in Hadoop
* All these Master Node and Slave Nodes are organized into a Network of clusters. Each Cluster is again divided into Racks. Each rack contains a set of Nodes (Commodity Computer).
* When Hadoop system receives “Store” operation like storing Large DataSets into HDFS, it stores that data into 3 different Nodes (As we configure Replication Factor = 3 by default). This complete data is not stored in one single node. Large Data File is divided into manageable and meaningful Blocks and distributed into different nodes with 3 copies.
* If Hadoop system receives any “Compute” operation, it will talk to near-by nodes to retrieve those blocks of Data. While Reading Data or Computing if one or more nodes get failed, then it will automatically pick-up performing those tasks by approaching any near-by and available node.
* That’s why Hadoop system provides highly available and fault tolerant BigData Solutions.

**NOTE:-**

* Hadoop 1.x Architecture has lot of limitations and drawbacks. So that Hadoop Community has evaluated and redesigned this Architecture into Hadoop 2.x Architecture.
* Hadoop 2.x Architecture is completely different and resolved all Hadoop 1.x Architecture’s limitations and drawbacks.

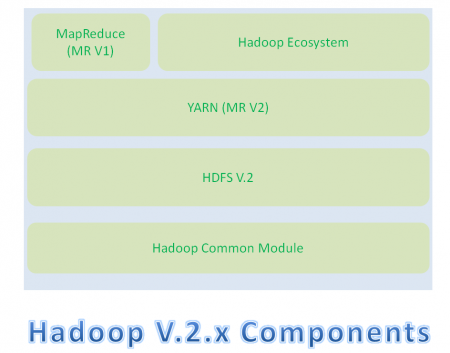


**HADOOP 2.X**



**Hadoop 2.x Architecture**

* Apache Hadoop 2.x or later versions are using the following Hadoop Architecture. It is a Hadoop 2.x High-level Architecture. We will discuss in-detailed Low-level Architecture in coming sections.

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop2.x-components.png)

* Hadoop Common Module is a Hadoop Base API (A Jar file) for all Hadoop Components. All other components works on top of this module.
* HDFS stands for Hadoop Distributed File System. It is also know as HDFS V2 as it is part of Hadoop 2.x with some enhanced features. It is used as a Distributed Storage System in Hadoop Architecture.
* YARN stands for Yet Another Resource Negotiator. It is new Component in Hadoop 2.x Architecture. It is also know as “MR V2”.
* MapReduce is a Batch Processing or Distributed Data Processing Module. It is also know as “MR V1” as it is part of Hadoop 1.x with some updated features.
* Remaining all Hadoop Ecosystem components work on top of these three major components: HDFS, YARN and MapReduce. We will discuss all Hadoop Ecosystem components in-detail in my coming posts.

When compared to Hadoop 1.x, Hadoop 2.x Architecture is designed completely different. It has added one new component : YARN and also updated HDFS and MapReduce component’s Responsibilities.

### **Hadoop 2.x Major Components**

Hadoop 2.x has the following three Major Components:

* HDFS
* YARN
* MapReduce

These three are also known as Three Pillars of Hadoop 2. Here major key component change is YARN. It is really game changing component in BigData Hadoop System.

### How Hadoop 2.x Major Components Works

Hadoop 2.x components follow this architecture to interact each other and to work parallel in a reliable, highly available and fault-tolerant manner.

**Hadoop 2.x Components High-Level Architecture**

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop2.x-highlevel-architecture.png)

* All Master Nodes and Slave Nodes contains both MapReduce and HDFS Components.
* One Master Node has two components:
  1. Resource Manager(YARN or MapReduce v2)
  2. HDFS

It’s HDFS component is also knows as NameNode. It’s NameNode is used to store Meta Data.In Hadoop 2.x, some more Nodes acts as Master Nodes as shown in the above diagram. Each this 2nd level Master Node has 3 components:

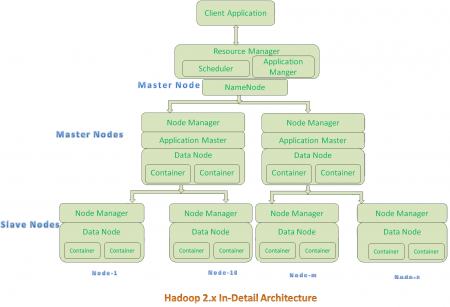
* 1. Node Manager
  2. Application Master
  3. Data Node

Each this 2nd level Master Node again contains one or more Slave Nodes as shown in the above diagram.

* These Slave Nodes have two components:
  1. Node Manager
  2. HDFS

It’s HDFS component is also knows as Data Node. It’s Data Node component is used to store actual our application Big Data. These nodes does not contain Application Master component.

**Hadoop 2.x Components In-detail Architecture**

[](https://cdn.journaldev.com/wp-content/uploads/2015/08/hadoop2.x-indetail-architecture.png)

**Hadoop 2.x Architecture Description**

**Resource Manager:**

* Resource Manager is a Per-Cluster Level Component.
* Resource Manager is again divided into two components:
  1. Scheduler
  2. Application Manager
* Resource Manager’s Scheduler is :
  1. Responsible to schedule required resources to Applications (that is Per-Application Master).
  2. It does only scheduling.
  3. It does care about monitoring or tracking of those Applications.

**Application Master:**

* Application Master is a per-application level component. It is responsible for:
  1. Managing assigned Application Life cycle.
  2. It interacts with both Resource Manager’s Scheduler and Node Manager
  3. It interacts with Scheduler to acquire required resources.
  4. It interacts with Node Manager to execute assigned tasks and monitor those task’s status.

**Node Manager:**

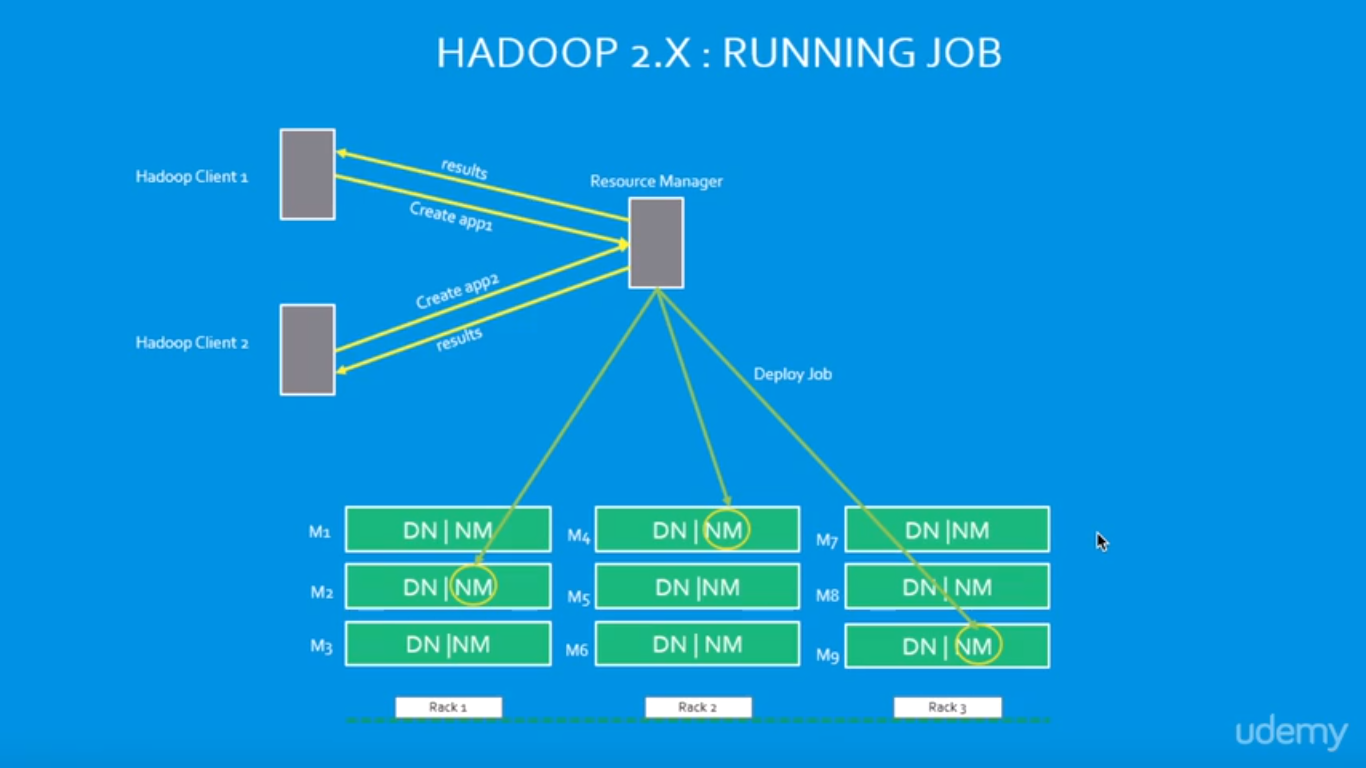
* Node Manager is a Per-Node Level component.
* It is responsible for:
  1. Managing the life-cycle of the Container.
  2. Monitoring each Container’s Resources utilization.

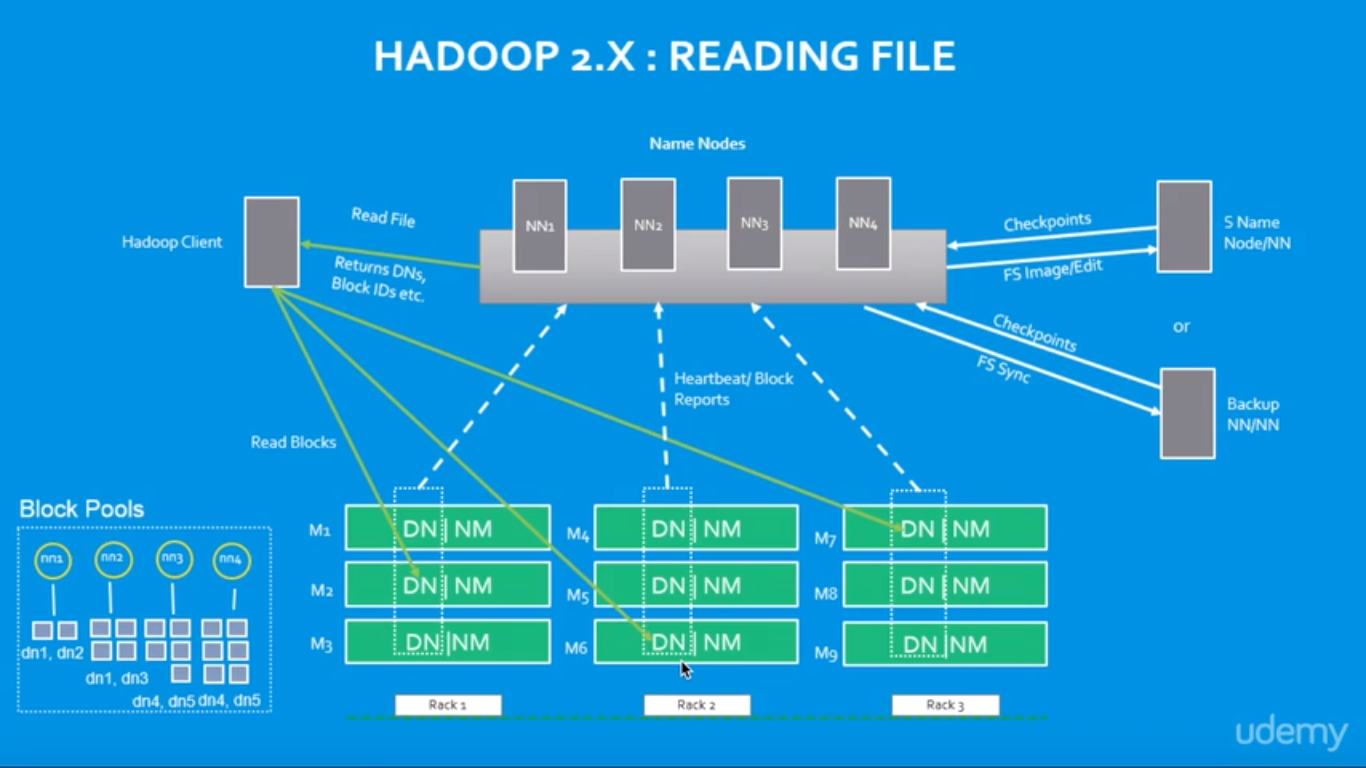
**Container:**

* Each Master Node or Slave Node contains set of Containers. In this diagram, Main Node’s Name Node is not showing the Containers. However, it also contains a set of Containers.
* Container is a portion of Memory in HDFS (Either Name Node or Data Node).
* In Hadoop 2.x, Container is similar to Data Slots in Hadoop 1.x. We will see the major differences between these two Components: Slots Vs Containers in my coming posts.

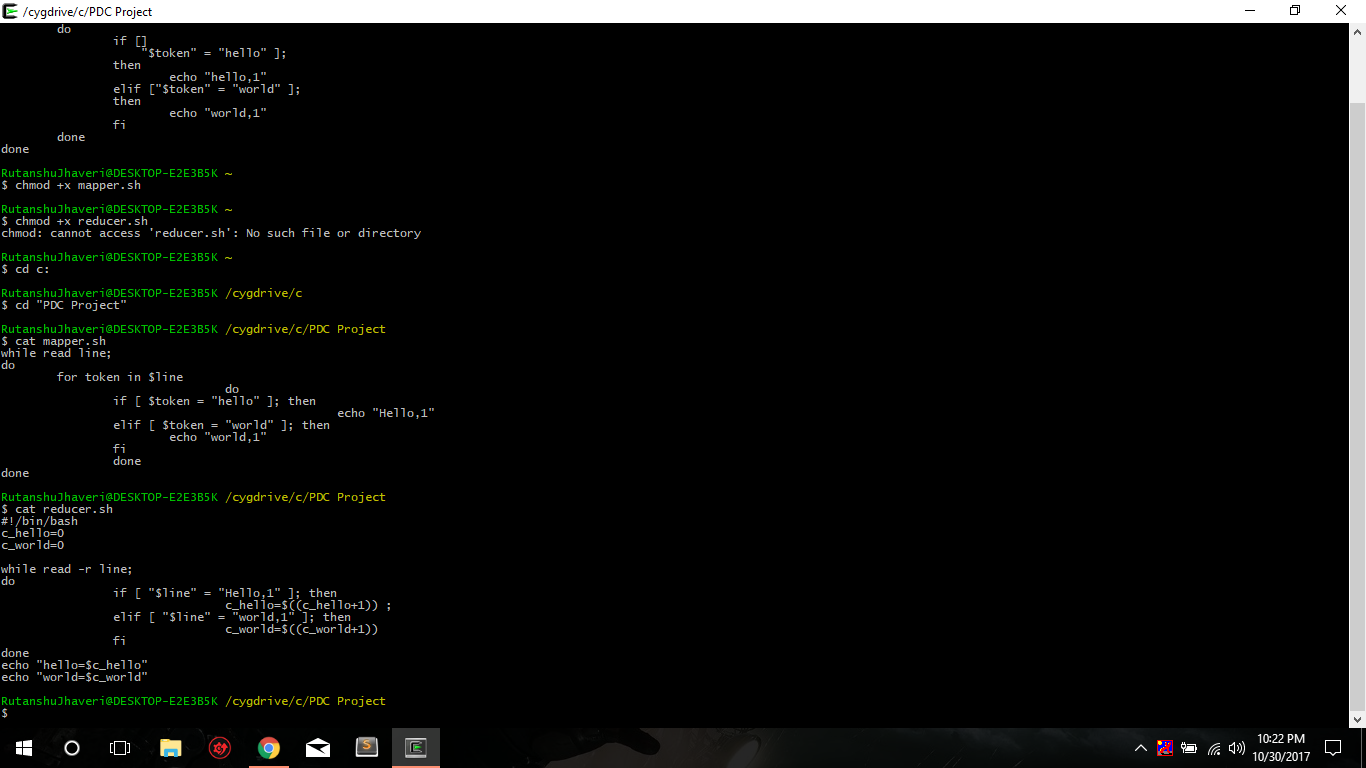
**NOTE:-**

* Resource Manager is Per-Cluster component where as Application Master is per-application component.
* Both Hadoop 1.x and Hadoop 2.x Architectures follow Master-Slave Architecture Model.

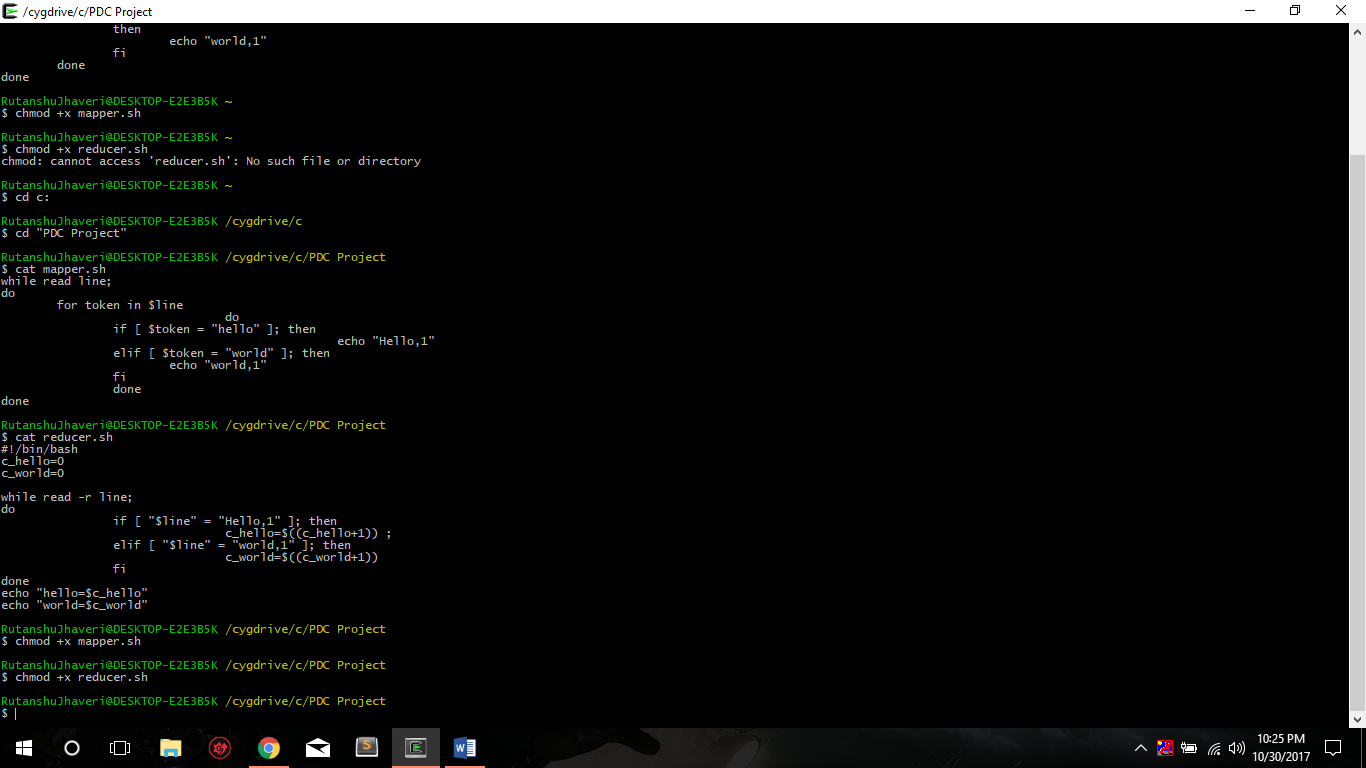




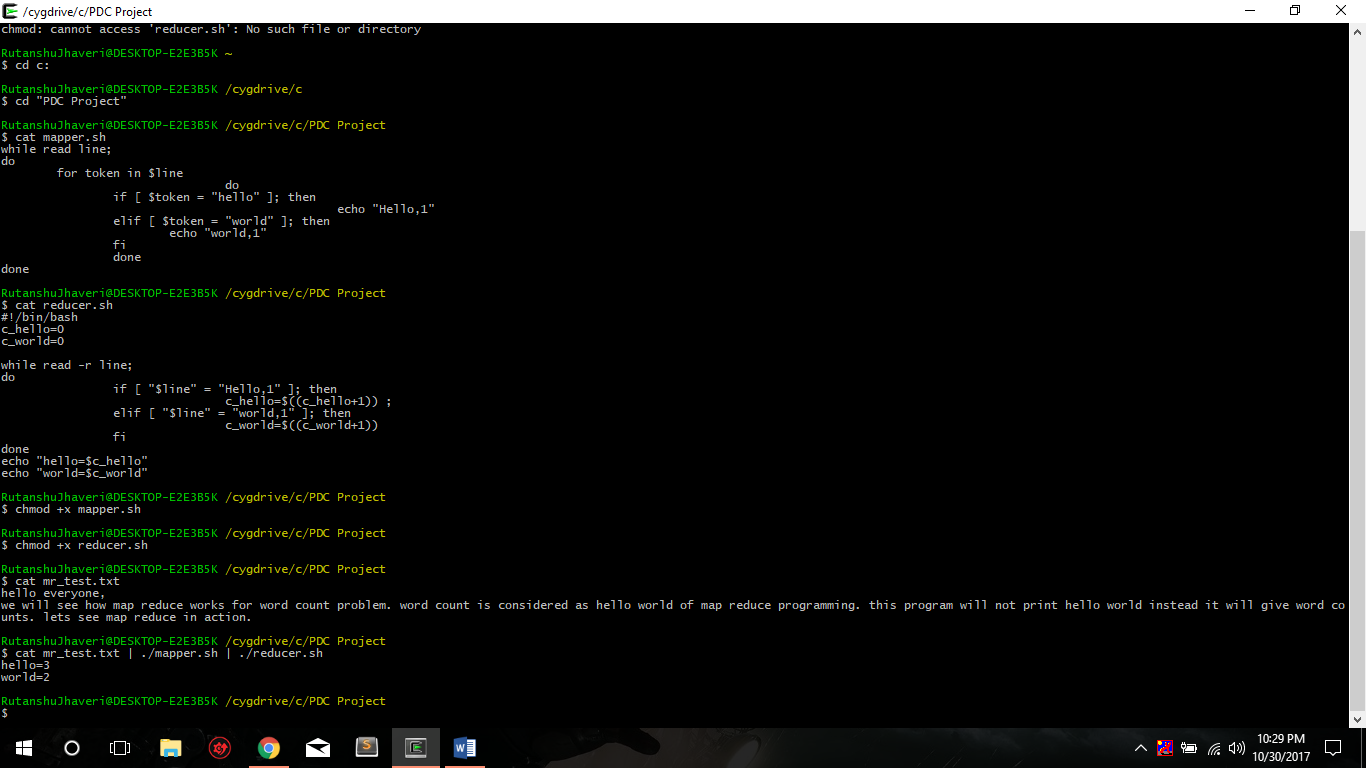
**Implementation Example:-**

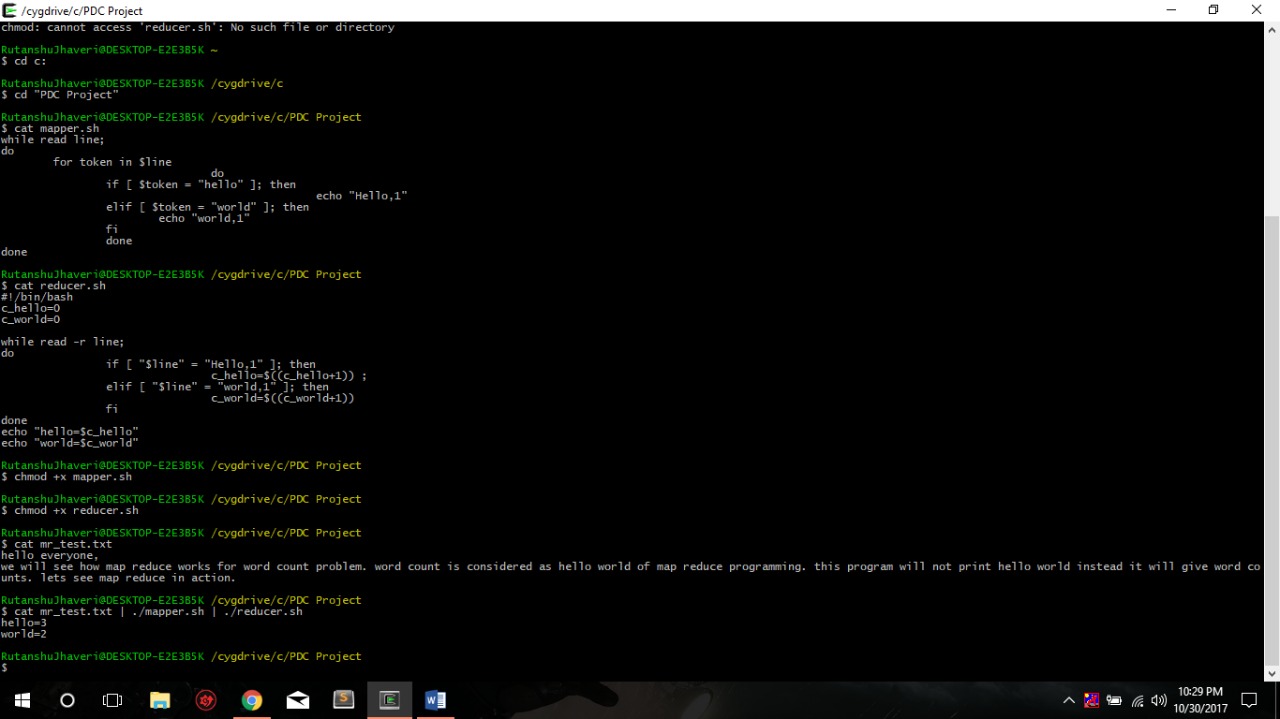


Mapper and reducer shell scripts to count the words “hello” and “world”



Giving permission to file of shell scripts





**Project is on Virtual Box**

We can show the code pig and hive files.

Therefore, there is no directory to show, The final project would be shown in video of Review 3.

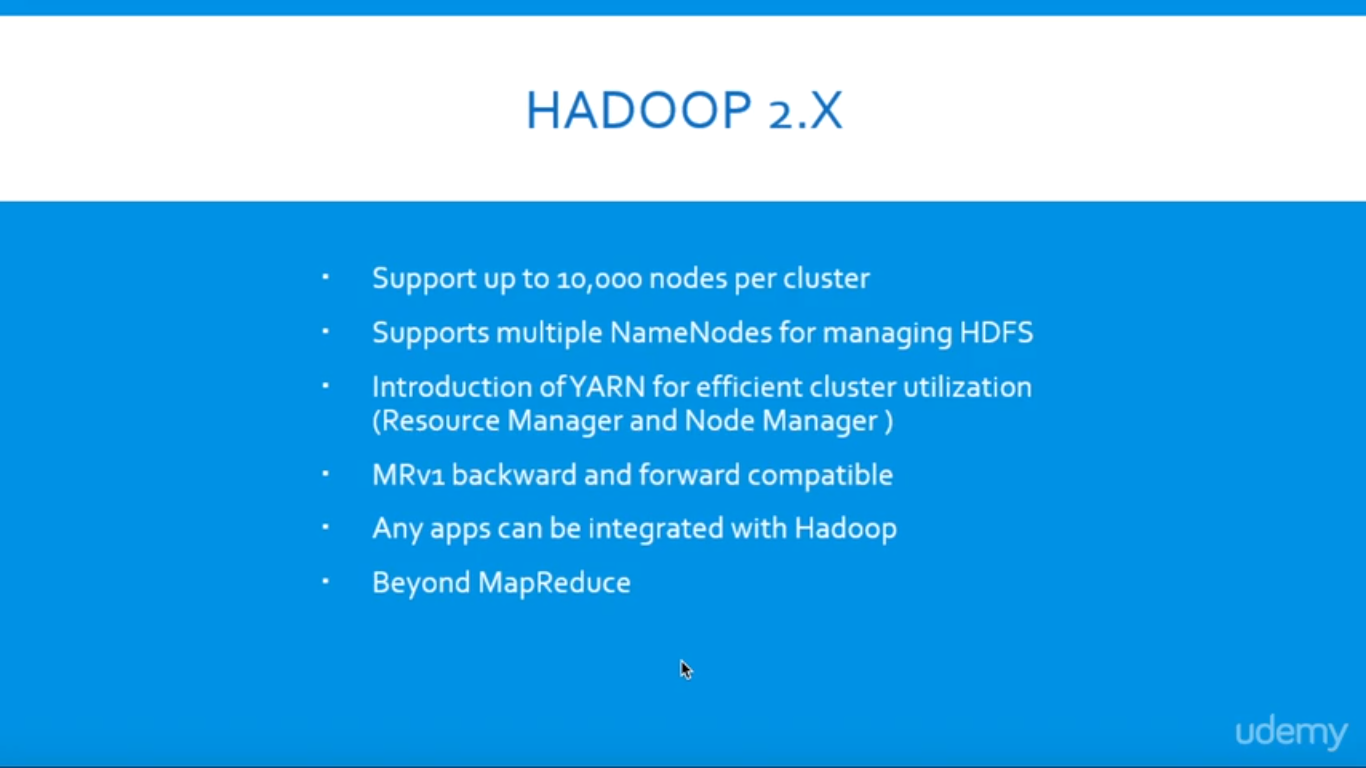
https://github.com/rutanshuj/PDCProject

Hadoop 2.x

# Introduction

Apache Hadoop 2.0 represents a generational shift in the architecture of Apache Hadoop. With YARN, Apache Hadoop is recast as a significantly more powerful platform – one that takes Hadoop beyond merely batch applications to taking its position as a ‘data operating system’ where HDFS is the file system and YARN is the operating system.

YARN is a re-architecture of Hadoop that allows multiple applications to run on the same platform. With YARN, applications run “in” Hadoop, instead of “on” Hadoop:



The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker and TaskTracker into separate entities. In Hadoop 2.0, the JobTracker and TaskTracker no longer exist and have been replaced by three components:

* ResourceManager: a scheduler that allocates available resources in the cluster amongst the competing applications.
* NodeManager: runs on each node in the cluster and takes direction from the ResourceManager. It is responsible for managing resources available on a single node.
* ApplicationMaster: an instance of a framework-specific library, an ApplicationMaster runs a specific YARN job and is responsible for negotiating resources from the ResourceManager and also working with the NodeManager to execute and monitor Containers.

The actual data processing occurs within the Containers executed by the ApplicationMaster. A Container grants rights to an application to use a specific amount of resources (memory, cpu etc.) on a specific host.

YARN is not the only new major feature of Hadoop 2.0. HDFS has undergone a major transformation with a collection of new features that include:

* NameNode HA: automated failover with a hot standby and resiliency for the NameNode master service.
* Snapshots: point-in-time recovery for backup, disaster recovery and protection against use errors.
* Federation: a clear separation of namespace and storage by enabling generic block storage layer.

NameNode HA is achieved using existing components like ZooKeeper along with new components like a quorum of JournalNodes and the ZooKeeper Failover Controller (ZKFC) processes:

Federation enables support for multiple namespaces in the cluster to improve scalability and isolation. Federation also opens up the architecture, expanding the applicability of HDFS cluster to new implementations and use cases.

# HDFS File Read Workflow

Now let’s understand complete end to end HDFS data read operation. As shown in the above figure the data read operation in HDFS is distributed, the client reads the data parallelly from datanodes, the steps by step explanation of data read cycle is:

i) Client opens the file it wishes to read by calling open() on the*FileSystem* object, which for HDFS is an instance of *DistributedFileSystem*.

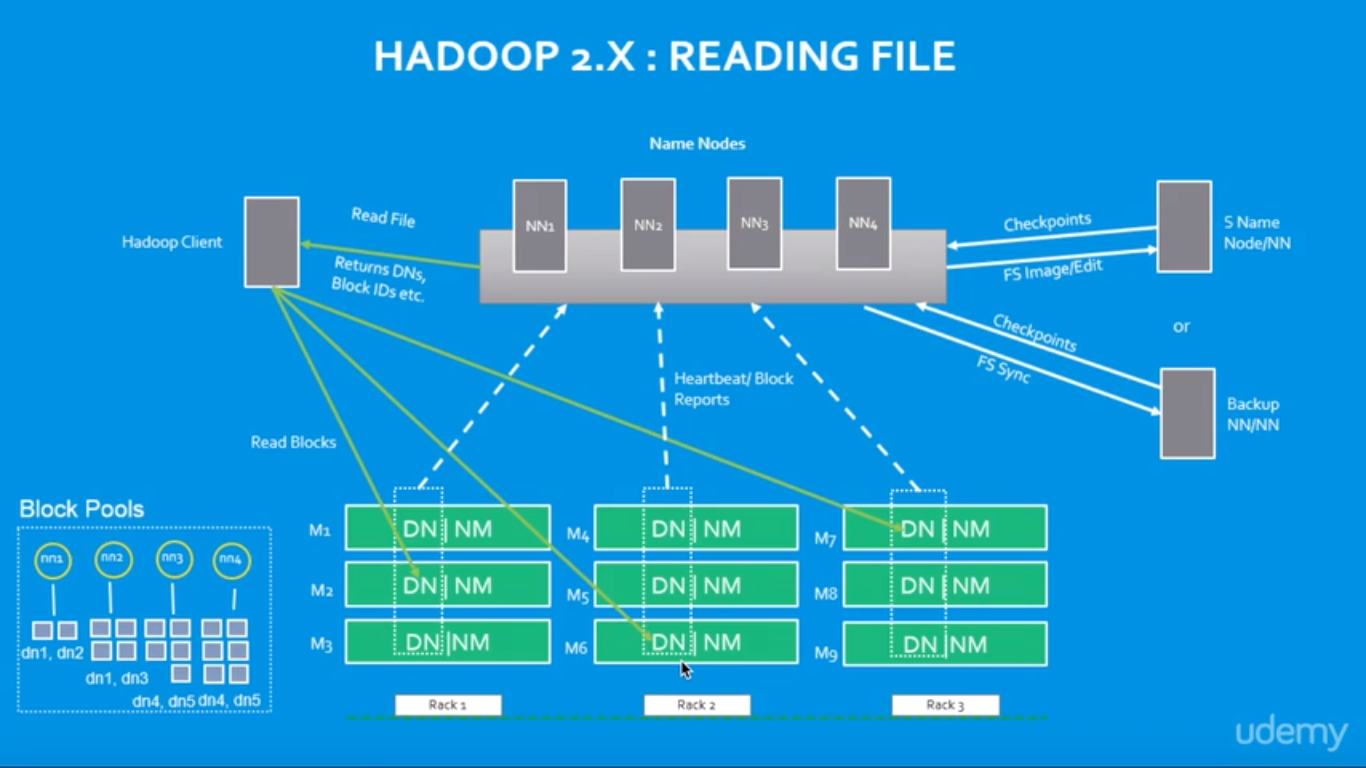
ii) *DistributedFileSystem* calls the namenode using RPC to determine the locations of the blocks for the first few blocks in the file. For each block, the namenode returns the addresses of the datanodes that have a copy of that block and datanode are sorted according to their proximity to the client.

iii)*DistributedFileSystem* returns a *FSDataInputStream* to the client for it to read data from. *FSDataInputStream*, thus, wraps the *DFSInputStream* which manages the datanode and namenode I/O. Client calls read() on the stream. DFSInputStream which has stored the datanode addresses then connects to the closest datanode for the first block in the file.

iv) Data is streamed from the datanode back to the client, as a result client can call read() repeatedly on the stream. When the block ends, DFSInputStream will close the connection to the datanode and then finds the best datanode for the next block.

v) If the *DFSInputStream* encounters an error while communicating with a datanode, it will try the next closest one for that block. It will also remember datanodes that have failed so that it doesn’t needlessly retry them for later blocks. The *DFSInputStream* also verifies checksums for the data transferred to it from the datanode. If it finds a corrupt block, it reports this to the namenode before the*DFSInputStream* attempts to read a replica of the block from another datanode.

vi) When the client has finished reading the data, it calls close() on the stream.



# Running a job on Hadoop

Before we jump into the programming our MapReduce, we may need to talk about the preparation steps that are commonly taken. Because MapReduce is usually operating on a huge data, we need to consider those steps before we actually do the MapReduce.

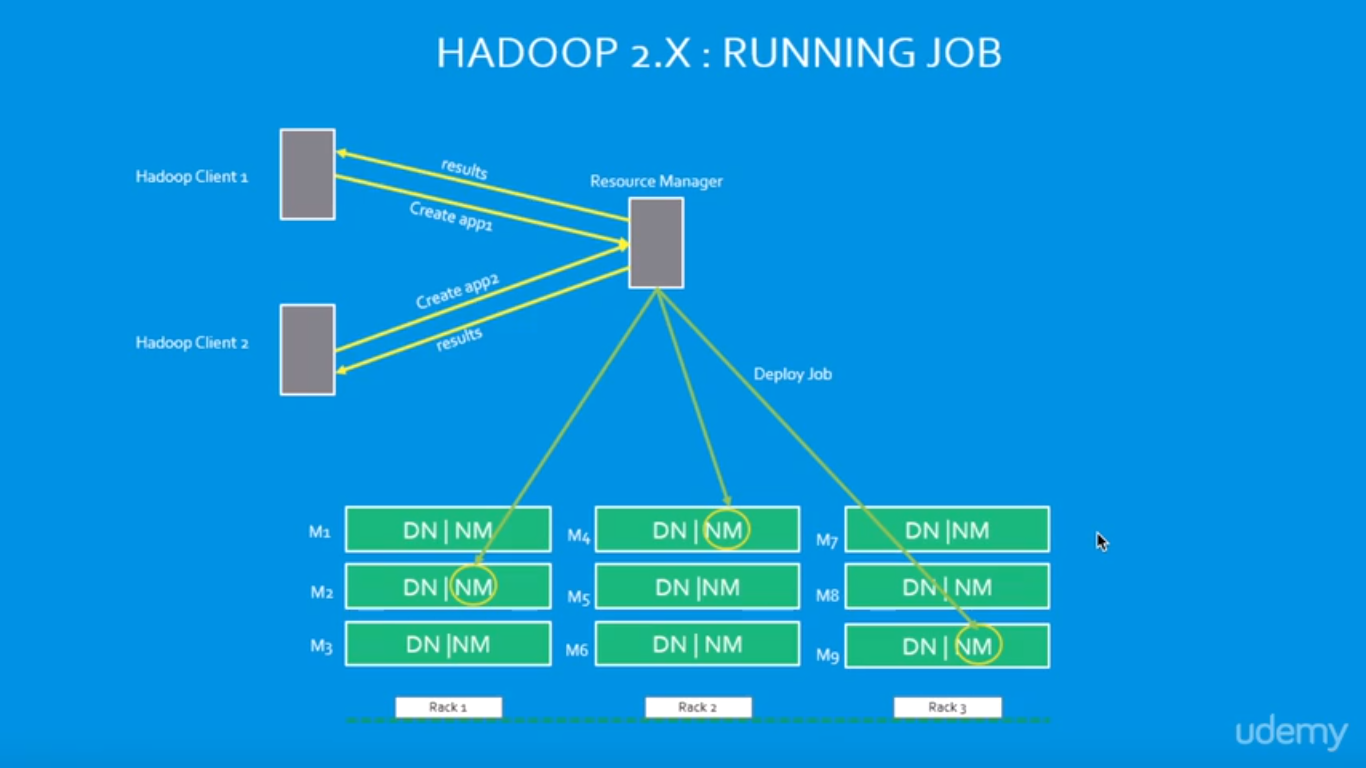
The underlying structure of the HDFS filesystem is very different from our normal file systems. The block sizes are quite a bit larger, and the actual block size for our clusters dependent on the cluster configuration as shown in the picture below: 64, 128, or 256 MB. So, we may need to have blocks with customized partitioned.

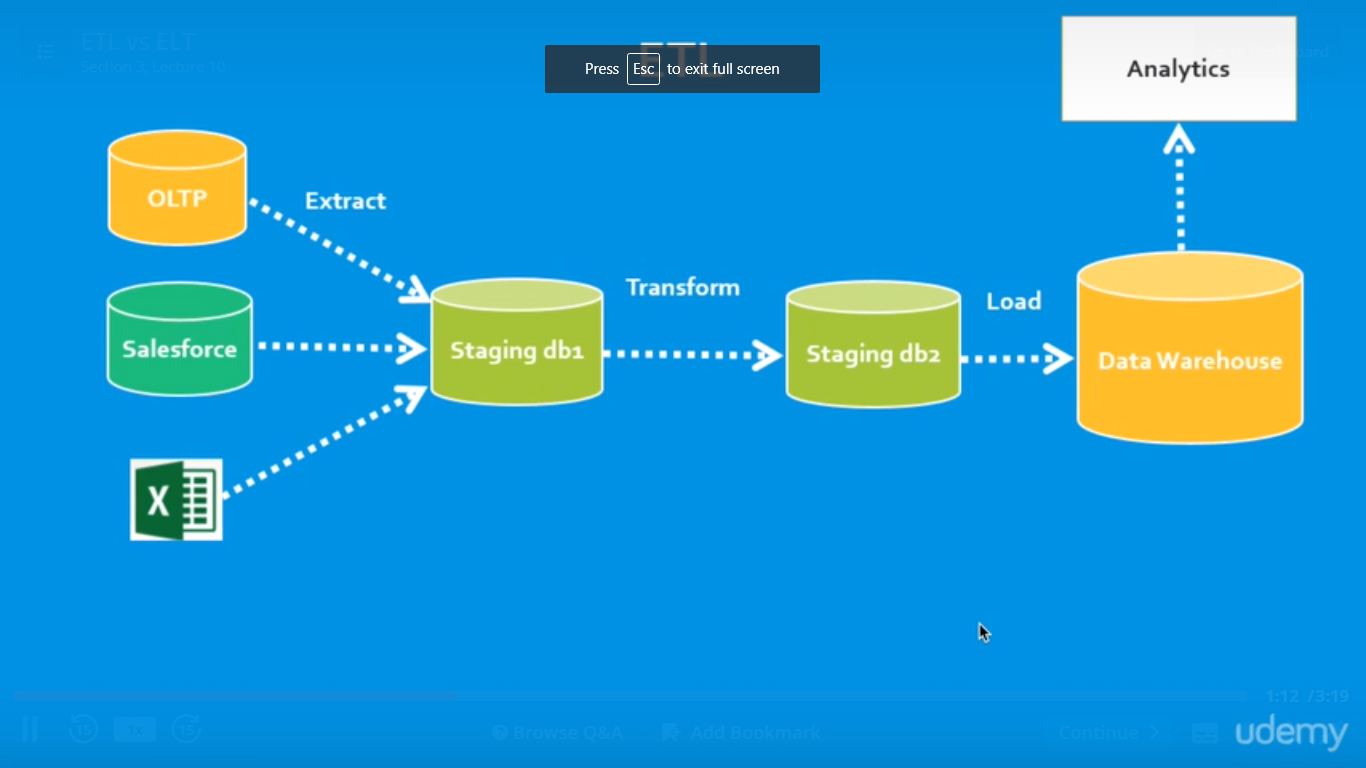
Another consideration is where we're going to retrieve our data from in order to perform the MapReduce operations or the parallel processing on it. Though we'll work with the core Hadoop filesystem, we may execute MapReduce algorithms against information stored on different locations such as native filesystem, cloud storage such as Amazon S3 buckets, or Windows Azure blobs.

Another considration is the output of the MapReduce job results are immutable. So, our output is a one-time output, and when a new output is generated, we have a new file name for it.

The last consideration in preparing for MapReduce is about the logic that we'll be writing, and it should fit our situation that we're trying to address. We'll be writing logic in some programming language, library, or tools to map our data to, and then reduce it, and then we have some output.

Note also that we'll be working with key-value pairs, so regardless of the format of the data coming in, we want to output key-value pairs.





# ELT and ETL

These two definitions of ETL are what make ELT a bit confusing. ELT is a different way of looking at the tool approach to data movement. Instead of transforming the data before it’s written, ELT leverages the target system to do the transformation. The data is copied to the target and then transformed in place.

ELT makes sense when the target is a high-end data engine, such as a data appliance, Hadoop cluster, or cloud installation to name three examples.  If this power is there, why not use it?

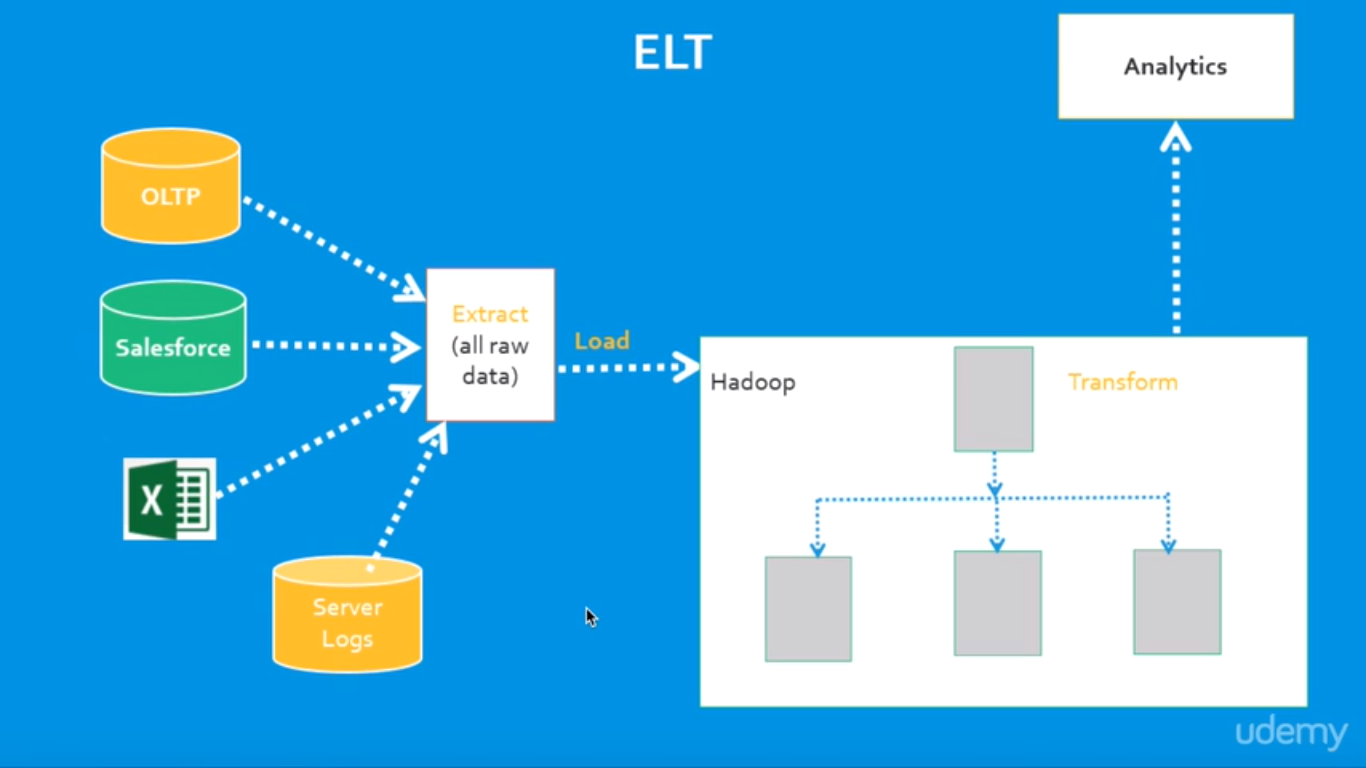
ETL, on the other hand, is designed using a pipeline approach. While data is flowing from the source to the target, a transformation engine (something unique to the tool) takes care of any data changes.

Which is better depends on priorities. All things being equal, it’s better to have fewer moving parts. ELT has no transformation engine – the work is done by the target system, which is already there and probably being used for other development work. On the other hand, the ETL approach can provide drastically better performance in certain scenarios. The training and development costs of ETL need to be weighed against the need for better performance. (Additionally, if you don’t have a target system powerful enough for ELT, ETL may be more economical.

The specifics of ELT development will vary depending on the platform. For example, Hadoop clusters work by breaking a problem into smaller chunks, then distributing those chunks across a large number of machines for processing. The problem is solved faster because it’s being done in parallel. This requires careful design to make sure that the act of splitting the problem can be done without affecting the answer. Some problems can be easily split, others will be much harder.

In all cases, developers need to be aware of the nature of the system they’re using to perform transformations. Some systems (such as hardware appliances) have enough resources to handle nearly any transformation, but others require careful planning and design.

ELT is an excellent tactical tool for loading a data warehouse. It requires a powerful system in place as the target, but more and more warehouses are being built with such systems in mind to meet ever-growing analytic needs. As with any tool, knowing when to use it is at least as important as knowing how to use it. Ironsides can provide strategic direction and/or technical support in data integration and management. Contact us today to discuss which options fit your environment best.



Different Vendors:

1.Amazon Elastic MapReduce (Amazon EMR)

Amazon Elastic MapReduce (EMR) is an Amazon Web Services ([AWS](http://whatis.techtarget.com/definition/Amazon-Web-Services-AWS)) tool for big data processing and analysis. Amazon EMR offers the expandable low-configuration service as an easier alternative to running in-house [cluster computing](http://searchdatacenter.techtarget.com/definition/cluster-computing).

Amazon EMR is based on Apache [Hadoop](http://searchcloudcomputing.techtarget.com/definition/Hadoop), a Java-based programming framework that supports the processing of large data sets in a [distributed computing](http://whatis.techtarget.com/definition/distributed-computing) environment. [MapReduce](http://searchcloudcomputing.techtarget.com/definition/MapReduce) is a software framework that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of [processors](http://searchcio-midmarket.techtarget.com/definition/processor) or stand-alone computers. It was developed at Google for indexing web pages and replaced their original indexing algorithms and [heuristics](http://whatis.techtarget.com/definition/heuristic) in 2004.

Amazon EMR processes big data across a [Hadoop cluster](http://searchbusinessanalytics.techtarget.com/definition/Hadoop-cluster) of virtual servers on Amazon Elastic Compute Cloud ([EC2](http://searchcloudcomputing.techtarget.com/definition/Amazon-Elastic-Compute-Cloud)) and Amazon Simple Storage Service (S3). The elastic in EMR's name refers to its dynamic resizing ability, which allows it to ramp up or reduce resource use depending on the demand at any given time.

2.Cloud Era

Cloudera Inc. is a [United States](https://en.wikipedia.org/wiki/United_States)-based software company that provides [Apache Hadoop](https://en.wikipedia.org/wiki/Apache_Hadoop)-based software, support and services, and training to business customers.

Cloudera's open-source Apache Hadoop distribution, CDH (Cloudera Distribution Including Apache Hadoop), targets enterprise-class deployments of that technology. Cloudera says that more than 50% of its engineering output is donated upstream to the various Apache-licensed open source projects (Apache Hive, Apache Avro, [Apache HBase](https://en.wikipedia.org/wiki/Apache_HBase), and so on) that combine to form the Hadoop platform.

Cloudera is also a sponsor of the [Apache Software Foundation](https://en.wikipedia.org/wiki/Apache_Software_Foundation).

3.IBM InfoSphere

InfoSphere DataStage is a powerful data integration tool.

It was acquired by IBM in 2005 and has become a part of IBM Information Server Platform. It uses a client/server design where jobs are created and administered via a Windows client against central repository on a server.

The IBM InfoSphere DataStage is capable of integrating data on demand across multiple and high volumes of data sources and target applications using a high performance parallel framework.

InfoSphere DataStage also facilitates extended metadata management and enterprise connectivity

It has three levels of Parallelism which are:

Pipeline Parallelism, Data Parallelism, Component Parallelism

Teradata Logo.png4.Teradata

Teradata Corporation is a provider of [database](https://en.wikipedia.org/wiki/Database)-related products and services. The company was formed in 1979 in [Brentwood, California](https://en.wikipedia.org/wiki/Brentwood,_California), as a collaboration between researchers at [Caltech](https://en.wikipedia.org/wiki/Caltech) and [Citibank](https://en.wikipedia.org/wiki/Citibank)'s advanced technology group.[[2]](https://en.wikipedia.org/wiki/Teradata#cite_note-2) The company was acquired by [NCR Corporation](https://en.wikipedia.org/wiki/NCR_Corporation) in 1991, and subsequently spun-off again as an independent public company on October 1, 2007.

The company produces a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) of the same name, which it markets as a [data warehouse](https://en.wikipedia.org/wiki/Data_warehouse)

Teradata offers three main services to its customers: cloud and hardware-based data warehousing, business analytics, and consulting services.

the company launched Teradata Everywhere, which allows users to submit queries against public and private databases.

The service uses [massively parallel](https://en.wikipedia.org/wiki/Massively_parallel) processing across both its physical data warehouse and cloud storage, including managed environments such as [Amazon Web Services](https://en.wikipedia.org/wiki/Amazon_Web_Services), [Microsoft Azure](https://en.wikipedia.org/wiki/Microsoft_Azure), [VMware](https://en.wikipedia.org/wiki/VMware), and Teradata's Managed Cloud and IntelliFlex

 Teradata offers customers both [hybrid cloud](https://en.wikipedia.org/wiki/Hybrid_cloud) and [multi-cloud](https://en.wikipedia.org/wiki/Multicloud) storage.