Semester: 6 Year: 2019-20

|  |  |
| --- | --- |
| *Department: Information Science and Engineering* | *Course Type:  Core* |
| *Course Title: Big Data Lab* | *Course Code:17ISL68* |
| *L-T-P:0-2-2* | *Credits:2* |
| *Total Contact Hours:52 Hrs* | *Duration of SEE:3 Hrs* |
| *SEE Marks: 50* | *CIE Marks: 50* |

**Course Outcomes:**

**Students will be able to:**

|  |  |  |
| --- | --- | --- |
| **Cos** | **Course Outcome Description** | **Blooms Level** |
| 1 | Use Mongo DB commands to implement given application. | L3 |
| 2 | Develop map/reduce programs to perform basic operations on the given data set | L3 |
| 3 | Use HiveQL to filter and aggregate the given data | L3 |

**Teaching Methodology:**

Black Board Teaching / Power Point Presentation

Laboratory experiments

**Assessment Methods:**

Rubrics for evaluating laboratory experiments for 10 marks

Two internals, 20 Marks each will be conducted and average of two internals will be taken.

Final examination of 50 Marks will be conducted.

**Course Outcome to Programme Outcome Mapping:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cos | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 |
| CO1 | 3 | 2 | 3 | 2 |  |  |  |  | 2 | 2 |  |  | 3 |  |
| CO2 | 3 | 2 | 3 | 2 |  |  |  |  | 2 | 2 |  |  | 3 |  |
| CO3 | 3 | 2 | 3 | 2 |  |  |  |  | 2 | 2 |  |  | 3 |  |
| CO4 | 3 | 2 | 3 | 2 |  |  |  |  | 2 | 2 |  |  | 3 |  |
| 17ISL68 | 3 | 2 | 3 | 2 |  |  |  |  | 2 | 2 |  |  | 3 |  |

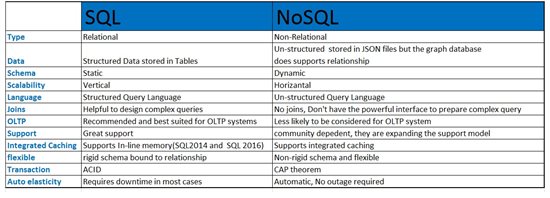
**Course content**

|  |  |
| --- | --- |
| **Exercise-1: Mongo DB**  Create a collection by the name “students” and insert documents.  Insert the document for” Arya” in to the student’s collection only if it does not exist else update the document with the new values.  Demonstrate save method to insert a document for student “Vamshi” in the student collection  Add a new field location with value “newyork”to the document (\_id:4) of student collection.  Remove a new field location with value “newyork”to the document (\_id:4) of student collection. | CO1 |
| **Exercise-2: Map/Reduce Job Submission**  Start by reviewing HDFS. It found that the composition of HDFS is similar to your local Linux file system. Use the *hadoopfs* command while interacting with HDFS. | CO3 |
| 1. Review the commands available for the Hadoop Distributed File System:  2. Copy file foo.txt from local disk to the user’s directory in HDFS  3. Get a directory listing of the user’s home directory in HDFS  4. Get a directory listing of the HDFS root directory  5. Display the contents of the HDFS file user/fred/bar.txt  6. Move that file to the local disk, named as baz.txt  7. Create a directory called input under the user’s home directory  8. Delete the directory input old and all its contents  9. Verify the copy by listing the directory contents in HDFS: | CO2 |
| **Exercise-3: Map Reduce (Programs)Using the Movie Lens data at the** URL<http://grouplens.org/datasets/movielens/>  Develop map/reduce program to perform basic operations on the data set. | CO3 |
| List all the movies and the number of ratings  List all the users and the number of ratings they have done for a movie  List all the Movie IDs which have been rated (Movie Id with at least one user rating it)  List all the Users who have rated the movies (Users who have rated at least one movie)  List of all the User with the max, min, average ratings they have given against any movie  List all the Movies with the max, min, average ratings given by any user | CO3 |
| **Exercise-4: Extract facts using Hive :**  Use HiveQL to filter and aggregate “click data” to build facts about user’s movie preferences. The query results should be saved in a staging table and must be used to populate the database. | CO4 |
| Write a query to select only those clicks which correspond to starting, browsing, completing, or purchasing movies. Use a CASE statement to transform the RECOMMENDED column into integers where ‘Y’ is 1 and ‘N’ is 0. Also, ensure GENREID is not null. Only include the first 25 rows.  Write a query to select the customer ID, movie ID, recommended state and most recent rating for each movie.  Load the results of the previous two queries into a staging table.  Load the results of the queries into the staging table. | CO4 |

**Experiment 1: Explore Mongo DB**

**Introduction**

## Differences between SQL and NOSQL



## The list

**sql**: mssql, oracle, mysql,maria, ibm db2, sybase, microsoft azure, and postgres

**nosql**: mongodb, bigtable, redis, ravendb, cassandra, hbase, neo4j, oracle nosql, and couchdb.

## Types of NOSQL databases

### Key-value model

The least complex nosql option, which stores data in a schema-less way that consists of indexed keys and values.

**Examples:** cassandra, azure, leveldb, and riak.

### Column store — or wide-column store

It stores data tables as columns rather than rows. It’s more than just an inverted table — sectioning out columns allows for excellent scalability and high performance.

**Examples:** hbase, bigtable, and hypertable.

### Document database

Taking the key-value concept and adding more complexity, each document in this type of database has its own data and its own unique key, which is used to retrieve it. It’s a great option for storing, retrieving, and managing data that’s document-oriented but still somewhat structured.

**Examples:** mongodb and couchdb.

### ****Graph database****

Here, the data is interconnected and best represented as a graph. This method is capable of lots of complexity.

**Examples:** polyglot andneo4j.

## Summary

The choice of the database between sql and nosql cannot be concluded on the differences between them but the project requirements. If your application has a fixed structure and doesn’t need frequent modifications, sql is a preferable database. Conversely, if you have applications where data is changing frequently and growing rapidly, like in big data analytics, nosql is the best option for you. And remember, sql is not deceased and can never be superseded by nosql, or any other database technology.

## Conclusion

All the technologies are best in what they do. It is up to the architect/developer/dba to make a better use of them depending on the situations and needs.

**Installation steps to Install Mongo DB**

## Steps to Install Mongo DB Community Edition

Follow these steps to install Mongo DB Community Edition using the apt package manager.

### 1. Import the public key used by the package management system.

From a terminal, issue the following command to import the Mongo DB public GPG Key from <https://www.mongodb.org/static/pgp/server-4.4.asc>

$ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -

The operation should respond with an OK.

However, if you receive an error indicating that gnupg is not installed, you can:

1. Install gnupg and its required libraries using the following command:

$ sudo apt-get install gnupg

* Once installed, retry importing the key:

$ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -

### 2. Create a list file for MongoDB.

Create the list file /etc/apt/sources.list.d/mongodb-org-4.4.list for your version of Ubuntu.

The following instruction is for Ubuntu 20.04 (Focal).

Create the /etc/apt/sources.list.d/mongodb-org-4.4.list file for Ubuntu 20.04 (Focal):

$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

The following instruction is for Ubuntu 18.04 (Bionic).

Create the /etc/apt/sources.list.d/mongodb-org-4.4.list file for Ubuntu 18.04 (Bionic):

$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

### 3. Reload local package database.

Issue the following command to reload the local package database:

$ sudo apt-get update

### 4. Install the MongoDB packages.

To install either the latest stable version of MongoDB

$ sudo apt-get install -y mongodb-org

## Steps to Run MongoDB Community Edition

#### **1. Start MongoDB.**

You can start the [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) process by issuing the following command:

**$ sudo systemctl start mongod**

If you receive an error similar to the following when starting [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod):

Failed to start mongod.service: Unit mongod.service not found.

Run the following command first:

**$ sudo systemctl daemon-reload**

Then run the start command above again.

#### **2. Verify that MongoDB has started successfully.**

**$ sudo systemctl status mongod**

You can optionally ensure that MongoDB will start following a system reboot by issuing the following command:

$ sudo systemctl enable mongod

#### **3. Stop MongoDB.**

As needed, you can stop the [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) process by issuing the following command:

$ sudo systemctl stop mongod

#### **4. Restart MongoDB.**

You can restart the [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) process by issuing the following command:

$ sudo systemctl restart mongod

You can follow the state of the process for errors or important messages by watching the output in the /var/log/mongodb/mongod.log file.

#### **5. Begin using MongoDB.**

Start a [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell on the same host machine as the [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod). You can run the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell without any command-line options to connect to a [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) that is running on your localhost with default port 27017:

**$ mongo**

Source : <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>

* **The use Command**

MongoDB use DATABASE\_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of use DATABASE statement is as follows −

***use DATABASE\_NAME***

* **To check your currently selected database**, use the command db

*>****db***nmit

* If you want to check your databases list, use the command show dbs.

>***show dbs***

local 0.78125GB

test 0.23012GB

* The **dropDatabase()** Method

MongoDB db.dropDatabase() command is used to drop a existing database.

**Syntax**

Basic syntax of dropDatabase() command is as follows −

***db.dropDatabase()***

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

* The **createCollection()** Method

MongoDB db.createCollection(name, options) is used to create collection.

**Syntax**

Basic syntax of createCollection() command is as follows −

***db.createCollection(name, options)***

***db.createCollection("student", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )***

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

>db.student.insert({"name" : “john"}),

WriteResult({ "nInserted" : 1 })

* **The drop() Method**

MongoDB's db.collection.drop() is used to drop a collection from the database.

Syntax

Basic syntax of drop() command is as follows −

db.COLLECTION\_NAME.drop()

* **The insert() Method**

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

**Syntax**

The basic syntax of insert() command is as follows −

>db.COLLECTION\_NAME.insert(document)

**Example**

> db.users.insert({

... \_id : ObjectId("507f191e810c19729de860ea"),

... title: "MongoDB",

... description: "MongoDB is no sql database",

... by: "nmit",

... url: "http://www.nmit.ac.in",

... tags: ['mongodb', 'database', 'NoSQL'],

... likes: 100

... })

WriteResult({ "nInserted" : 1 })

>

***You can also pass an array of documents into the insert() method as shown below:***

> db.createCollection("post")

> db.post.insert([

{

title: "MongoDB",

description: "MongoDB is no SQL database",

by: " nmit",

url: " http://www.nmit.ac.in ",

tags: ["mongodb", "database", "NoSQL"],

likes: 100

},

{

title: "NoSQL Database",

description: "NoSQL database doesn't have tables",

by: "nmit",

url: " http://www.nmit.ac.in ",

tags: ["mongodb", "database", "NoSQL"],

likes: 20,

comments: [

{

user:"user1",

message: "My first comment",

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

* **The insertOne() method**

If you need to insert only one document into a collection you can use this method.

**Syntax**

The basic syntax of insert() command is as follows −

>db.COLLECTION\_NAME.insertOne(document)

* **The insertMany() method**

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

**Example**

Following example inserts three different documents into the empDetails collection using the insertMany() method.

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

* **The find() Method**

To query data from MongoDB collection, you need to use MongoDB's find() method.

**Syntax**

The basic syntax of find() method is as follows −

>db.COLLECTION\_NAME.find()

* **The pretty() Method**

To display the results in a formatted way, you can use pretty() method.

**Syntax**

>db.COLLECTION\_NAME.find().pretty()

* **MongoDB updateOne() method**

This methods updates a single document which matches the given filter.

**Syntax**

The basic syntax of updateOne() method is as follows −

>db.COLLECTION\_NAME.updateOne(<filter>, <update>)

**Example**

> db.empDetails.updateOne(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

* **MongoDB updateMany() method**

The updateMany() method updates all the documents that matches the given filter.

**Syntax**

The basic syntax of updateMany() method is as follows −

>db.COLLECTION\_NAME.update(<filter>, <update>)

**Exampl**e

> db.empDetails.updateMany(

{Age:{ $gt: "25" }},

{ $set: { Age: '00'}}

)

* **Remove Only One**

If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

* **Remove All Documents**

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.

> db.mycol.remove({})

* The basic syntax of **find() method with projection** is as follows −

>db.COLLECTION\_NAME.find({},{KEY:1})

Please note \_id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

* **The Limit() Method**

To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

**Syntax**

The basic syntax of limit() method is as follows −

>db.COLLECTION\_NAME.find().limit(NUMBER)

* **The aggregate() Method**

**Syntax**

Basic syntax of aggregate() method is as follows −

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

|  |  |  |
| --- | --- | --- |
| **Expression** | **Description** | **Example** |
| $sum | Sums up the defined value from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | Gets the minimum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | Gets the maximum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | Inserts the value to an array in the resulting document. | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

**RDBMS Where Clause Equivalents in MongoDB**

To query the document on the basis of some condition, you can use following operations.

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Syntax | Example | RDBMS Equivalent |
| Equality | {<key>:{$eg;<value>}} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |
| Values in an array | {<key>:{$in:[<value1>, <value2>,……<valueN>]}} | db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty() | Where name matches any of the value in :["Raj", "Ram", "Raghu"] |
| Values not in an array | {<key>:{$nin:<value>}} | db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty() | Where name values is not in the array :["Ramu", "Raghav"] or, doesn’t exist at all |

* **AND in MongoDB**

**Syntax**

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND −

>db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })

* **OR in MongoDB**

**Syntax**

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

# **Aggregation Pipeline**

The aggregation pipeline is a framework for data aggregation modelled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results.

For example:

db.orders.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } }

])

**First Stage**: The [$match](https://docs.mongodb.com/manual/reference/operator/aggregation/match/#mongodb-pipeline-pipe.-match) stage filters the documents by the status field and passes to the next stage those documents that have status equal to "A".

**Second Stage**: The [$group](https://docs.mongodb.com/manual/reference/operator/aggregation/group/#mongodb-pipeline-pipe.-group) stage groups the documents by the cust\_id field to calculate the sum of the amount for each unique cust\_id.

The MongoDB aggregation pipeline consists various stages. Each stage transforms the documents passes through the pipeline. In this stage it is not needed to produce one output document for every input document. The pipeline stages can appear multiple times in the pipeline.

Here is a list of MongoDB Aggregation Pipeline Operators in the table below.

|  |  |
| --- | --- |
| **Name** | **Description** |
| [$project](https://www.w3resource.com/mongodb/aggregation/mongodb-aggregatrion-project-operator.php) | The $project function in MongoDB passes along the documents with only the specified fields to the next stage in the pipeline. This may be the existing fields from the input documents or newly computed fields. |
| [$match](https://www.w3resource.com/mongodb/aggregation/mongodb-aggregatrion-match-operator.php) | The MongoDB $match operator filters the documents to pass only those documents that match the specified condition(s) to the next pipeline stage. |
| [$unwind](https://www.w3resource.com/mongodb/aggregation/mongodb-aggregatrion-unwind-operator.php) | The MongoDB $unwind stages operator is used to deconstructs an array field from the input documents to output a document for each element. Every output document is the input document with the value of the array field replaced by the element. |
| [$group](https://www.w3resource.com/mongodb/aggregation/mongodb-aggregatrion-group-operator.php) | The MongoDB $group stages operator groups the documents by some specified expression and groups the document for each distinct grouping. An \_id field in the output documents contains the distinct group by key. The output documents can also contain computed fields that hold the values of some accumulator expression grouped by the $group‘s \_id field. This operator does not order its output documents. |
| [$out](https://www.w3resource.com/mongodb/aggregation/mongodb-aggregatrion-out-operator.php) | The MongoDB $out write the resulting document of the aggregation pipeline to a specified collection. The $out operator must be the last stage in the pipeline. The $out operator lets the aggregation framework return result sets of any size. |

**MapReduce**

* As per the MongoDB documentation, Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command

Following is the syntax of the basic mapReduce command −

>db.collection.mapReduce(

function() {emit(key,value);}, //map function

function(key,values) {return reduceFunction}, { //reduce function

out: collection,

query: document,

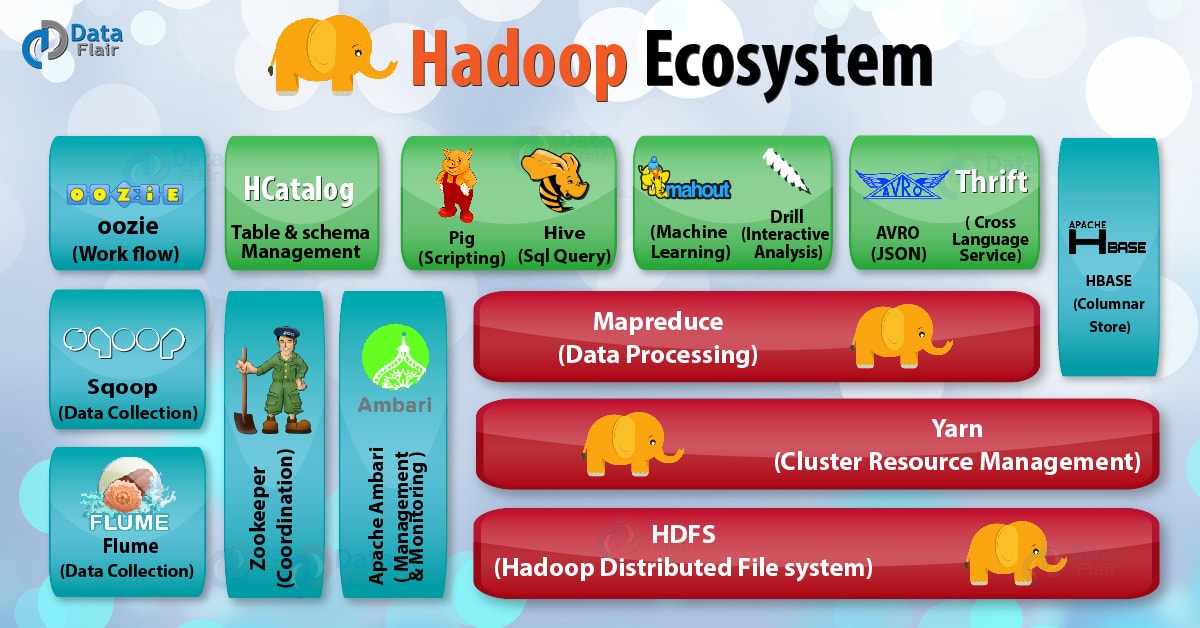
sort: document,

limit: number

}

)

**Experiment 2: HDFS Commands**



## 1. Hadoop Ecosystem Components

# The objective of this **Apache Hadoop** ecosystem components tutorial is to have an overview of what are the different components of Hadoop ecosystem that make Hadoop so powerful and due to which several **Hadoop job roles** are available now. We will also learn about Hadoop ecosystem components like **HDFS** and HDFS components, **MapReduce**, **YARN**, **Hive**,**Apache Pig**, **Apache HBase** and HBase components, **HCatalog**, **Avro**,**Thrift**, **Drill**, **Apache mahout**, **Sqoop**, **Apache Flume**, **Ambari**, **Zookeeper** and **Apache OOzie** to deep dive into Big Data Hadoop and to acquire master level knowledge of the Hadoop Ecosystem.

## 2. Introduction to Hadoop Ecosystem

As we can see the different Hadoop ecosystem explained in the above figure of Hadoop Ecosystem. Now we are going to discuss the list of Hadoop Components in this section one by one in detail.

### 2.1. Hadoop Distributed File System

It is the most important component of Hadoop Ecosystem.**HDFS** is the primary storage system of Hadoop. Hadoop distributed file system (HDFS) is a java based file system that provides scalable, fault tolerance, reliable and cost efficient data storage for **Big data**. HDFS is a distributed filesystem that runs on commodity hardware. HDFS is already configured with default configuration for many installations. Most of the time for large clusters configuration is needed. Hadoop interact directly with HDFS by shell-like commands.

**HDFS  Components:**

There are two major components of Hadoop HDFS- NameNode and DataNode. Let’s now discuss these Hadoop HDFS Components-

**i. NameNode**

It is also known as Master node. NameNode does not store actual data or dataset. NameNode stores Metadata i.e. number of **blocks**, their location, on which Rack, which Datanode the data is stored and other details. It consists of files and directories.

**Tasks of HDFS NameNode**

* Manage file system namespace.
* Regulates client’s access to files.
* Executes file system execution such as naming, closing, opening files and directories.

**ii. DataNode**

It is also known as Slave. HDFS Datanode is responsible for storing actual data in HDFS. Datanode performs **read and write operation** as per the request of the clients. Replica block of Datanode consists of 2 files on the file system. The first file is for data and second file is for recording the block’s metadata. HDFS Metadata includes checksums for data. At startup, each Datanode connects to its corresponding Namenode and does handshaking. Verification of namespace ID and software version of DataNode take place by handshaking. At the time of mismatch found, DataNode goes down automatically.

**Tasks of HDFS DataNode**

* DataNode performs operations like block replica creation, deletion, and replication according to the instruction of NameNode.
* DataNode manages data storage of the system.

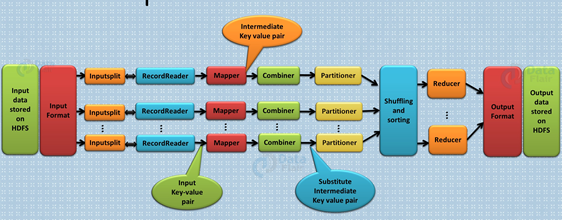
This was all about HDFS as a Hadoop Ecosystem component.

Refer **HDFS Comprehensive Guide** to read Hadoop HDFS in detail and then proceed with the Hadoop Ecosystem tutorial.

### 2.2. MapReduce

**Hadoop MapReduce** is the core Hadoop ecosystem component which provides data processing. MapReduce is a software framework for easily writing applications that process the vast amount of structured and unstructured data stored in the Hadoop Distributed File system.  
MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster. Thus, it improves the speed and reliability of cluster this parallel processing.

**MAP REDUCE JOB EXECUTION FLOW**



**Working of MapReduce**

Hadoop Ecosystem component ‘MapReduce’ works by breaking the processing into two phases:

* Map phase
* Reduce phase

Each phase has [**key-value pairs**](http://data-flair.training/blogs/key-value-pairs-hadoop-mapreduce/) as input and output. In addition, programmer also specifies two functions:**map function**and**reduce function**

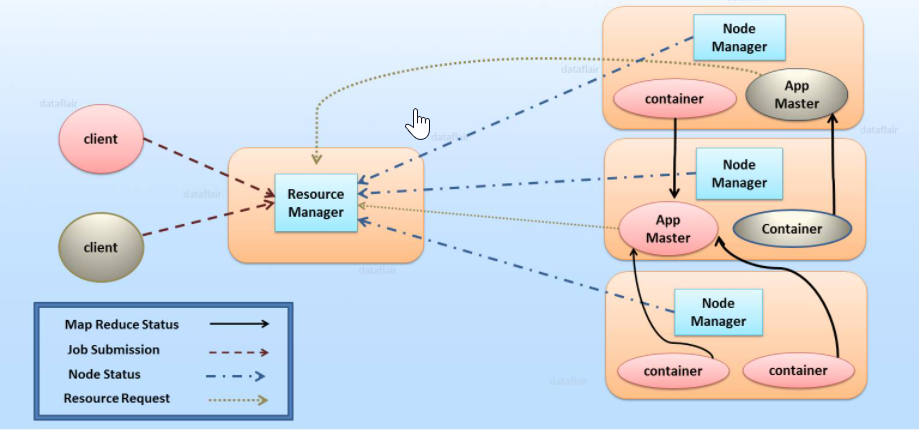
**Map** **function** takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). [**Read Mapper in detail**.](http://data-flair.training/blogs/mapper-in-hadoop-mapreduce/)  
**Reduce** **function**takes the output from the Map as an input and combines those data tuples based on the key and accordingly modifies the value of the key. [**Read Reducer in detail**.](http://data-flair.training/blogs/reducer-in-hadoop-mapreduce/)

**Features of MapReduce**

* **Simplicity –**MapReduce jobs are easy to run. Applications can be written in any language such as **java, C++**, and **python**.
* **Scalability –**MapReduce can process petabytes of data.
* **Speed –**By means of parallel processing problems that take days to solve, it is solved in hours and minutes by MapReduce.
* **Fault Tolerance –**MapReduce takes care of failures. If one copy of data is unavailable, another machine has a copy of the same key pair which can be used for solving the same subtask.

### 2.3. YARN

**Hadoop YARN** (**Y**et **A**nother **R**esource **N**egotiator) is a Hadoop ecosystem component that provides the resource management. Yarn is also one the most important component of Hadoop Ecosystem.  YARN is called as the operating system of Hadoop as it is responsible for managing and monitoring workloads. It allows multiple data processing engines such as real-time streaming and batch processing to handle data stored on a single platform.



YARN has been projected as a data operating system for [**Hadoop2**](http://data-flair.training/blogs/setup-hadoop-2-yarn-psedo-distributed-mode/). Main features of YARN are:

* **Flexibility –** Enables other purpose-built data processing models beyond MapReduce (batch), such as interactive and streaming. Due to this feature of YARN, other applications can also be run along with Map Reduce programs in Hadoop2.
* **Efficiency –** As many applications run on the same cluster, Hence, efficiency of Hadoop increases without much effect on quality of service.
* **Shared –** Provides a stable, reliable, secure foundation and shared operational services across multiple workloads. Additional programming models such as graph processing and iterative modelling are now possible for data processing.

# **Installation guide for installing Hadoop on Ubuntu 18.04 or 20.04**

**Prerequisite:**

## Install OpenJDK on Ubuntu

The Hadoop framework is written in Java, and its services require a compatible Java Runtime Environment (JRE) and Java Development Kit (JDK). Use the following command to update your system before initiating a new installation:

$ sudo apt update

At the moment, Apache Hadoop 3.x fully supports Java 8. The OpenJDK 8 package in Ubuntu contains both the runtime environment and development kit.

Type the following command in your terminal to install OpenJDK 8:

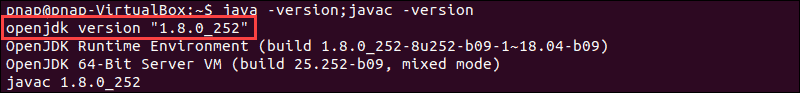
$ sudo apt install openjdk-8-jdk -y

The OpenJDK or Oracle Java version can affect how elements of a Hadoop ecosystem interact.

Once the installation process is complete, verify the current Java version:

$ java -version; javac -version

The output informs you which Java edition is in use.



## Set Up a Non-Root User for Hadoop Environment

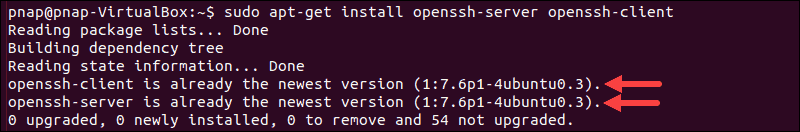
It is advisable to create a non-root user, specifically for the Hadoop environment. A distinct user improves security and helps you manage your cluster more efficiently. To ensure the smooth functioning of Hadoop services, the user should have the ability to establish a passwordless SSH connection with the localhost.

### Install OpenSSH on Ubuntu

Install the OpenSSH server and client using the following command:

$ sudo apt install openssh-server openssh-client -y

In the example below, the output confirms that the latest version is already installed.



### Create Hadoop User

Utilize the adduser command to create a new Hadoop user:

$ sudo adduser hdoop

The username, in this example, is hdoop. You are free to use any username and password you see fit.

$ sudo usermod -aG sudo hdoop

Switch to the newly created user and enter the corresponding password:

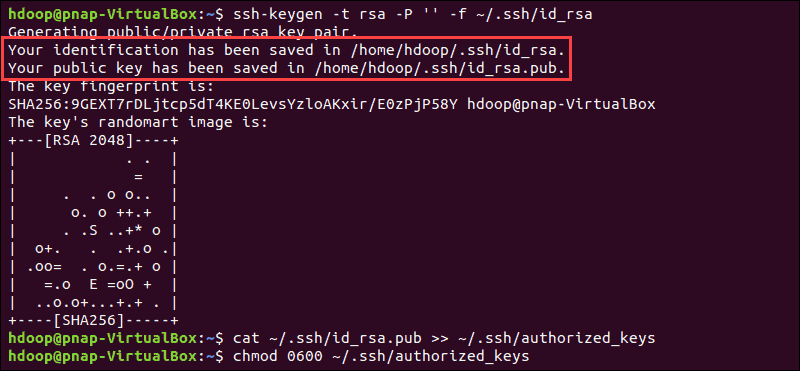
$su - hdoop

The user now needs to be able to SSH to the localhost without being prompted for a password.

Generate an SSH key pair and define the location is is to be stored in:

$ ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa

The system proceeds to generate and save the SSH key pair.



Use the cat command to store the public key as authorized\_keys in the ssh directory:

$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

Set the permissions for your user with the chmod command:

$ chmod 0600 ~/.ssh/authorized\_keys

The new user is now able to SSH without needing to enter a password every time. Verify everything is set up correctly by using the hdoop user to SSH to localhost:

$ ssh localhost

After an initial prompt, the Hadoop user is now able to establish an SSH connection to the localhost seamlessly.

## Download and Install Hadoop on Ubuntu

Use the provided mirror link and download the Hadoop package with the wget command:

$ wget https://downloads.apache.org/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz



Once the download is complete, extract the files to initiate the Hadoop installation:

$ tar xzf hadoop-3.2.1.tar.gz

The Hadoop binary files are now located within the hadoop-3.2.1 directory.

## Single Node Hadoop Deployment (Pseudo-Distributed Mode)

Hadoop excels when deployed in a fully distributed mode on a large cluster of networked servers. However, if you are new to Hadoop and want to explore basic commands or test applications, you can configure Hadoop on a single node.

This setup, also called pseudo-distributed mode, allows each Hadoop daemon to run as a single Java process. A Hadoop environment is configured by editing a set of configuration files:

* + - bashrc
    - hadoop-env.sh
    - core-site.xml
    - hdfs-site.xml
    - mapred-site-xml
    - yarn-site.xml

### Configure Hadoop Environment Variables (bashrc)

Edit the .bashrc shell configuration file using a text editor of your choice (we will be using nano):

$ sudo nano .bashrc

Define the Hadoop environment variables by adding the following content to the end of the file:

#Hadoop Related Options

export HADOOP\_HOME=/home/hdoop/hadoop-3.2.1

export HADOOP\_INSTALL=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

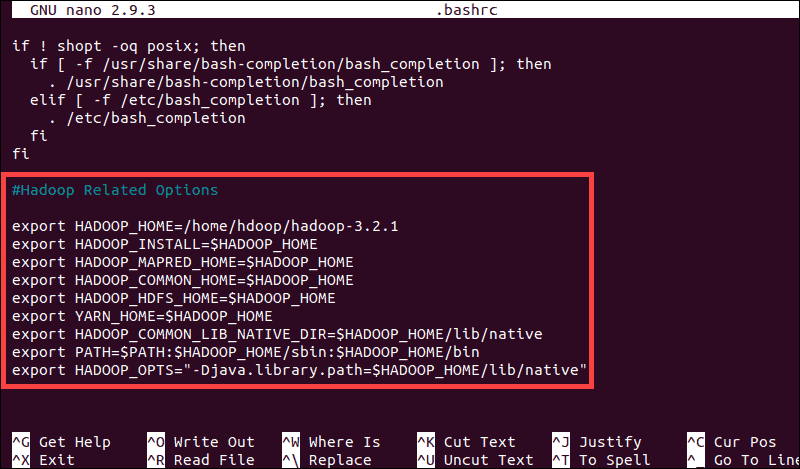
export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin

export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_HOME/lib/nativ"

Once you add the variables, save and exit the .bashrc file.



It is vital to apply the changes to the current running environment by using the following command:

$ source ~/.bashrc

### Edit hadoop-env.sh File

The hadoop-env.sh file serves as a master file to configure YARN, HDFS, MapReduce, and Hadoop-related project settings.

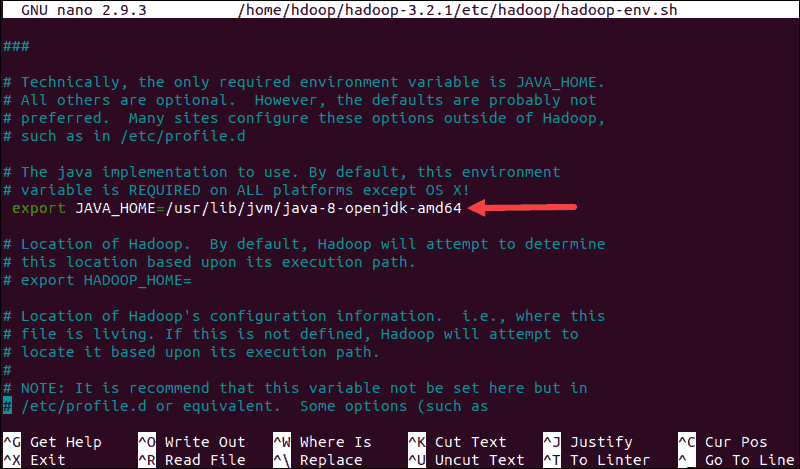
When setting up a single node Hadoop cluster, you need to define which Java implementation is to be utilized. Use the previously created $HADOOP\_HOME variable to access the hadoop-env.sh file:

$ sudo nano $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

Uncomment the $JAVA\_HOME variable (i.e., remove the # sign) and add the full path to the OpenJDK installation on your system. If you have installed the same version as presented in the first part of this tutorial, add the following line:

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

The path needs to match the location of the Java installation on your system.



If you need help to locate the correct Java path, run the following command in your terminal window:

which javac

The resulting output provides the path to the Java binary directory.



Use the provided path to find the OpenJDK directory with the following command:

$ readlink -f /usr/bin/javac

The section of the path just before the /bin/javac directory needs to be assigned to the $JAVA\_HOME variable.



### Edit core-site.xml File

The core-site.xml file defines HDFS and Hadoop core properties.

To set up Hadoop in a pseudo-distributed mode, you need to specify the URL for your NameNode, and the temporary directory Hadoop uses for the map and reduce process.

Open the core-site.xml file in a text editor:

$ sudo nano $HADOOP\_HOME/etc/hadoop/core-site.xml

Add the following configuration to override the default values for the temporary directory and add your HDFS URL to replace the default local file system setting:

<configuration>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hdoop/tmpdata</value>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://127.0.0.1:9000</value>

</property>

</configuration>

This example uses values specific to the local system. You should use values that match your systems requirements. The data needs to be consistent throughout the configuration process.



Do not forget to create a Linux directory in the location you specified for your temporary data.

### Edit hdfs-site.xml File

The properties in the hdfs-site.xml file govern the location for storing node metadata, fsimage file, and edit log file. Configure the file by defining the NameNode and DataNode storage directories.

Additionally, the default dfs.replication value of 3 needs to be changed to 1 to match the single node setup.

Use the following command to open the hdfs-site.xml file for editing:

$ sudo nano $HADOOP\_HOME/etc/hadoop/hdfs-site.xml

Add the following configuration to the file and, if needed, adjust the NameNode and DataNode directories to your custom locations:

<configuration>

<property>

<name>dfs.data.dir</name>

<value>/home/hdoop/dfsdata/namenode</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/home/hdoop/dfsdata/datanode</value>

</property>

<property>

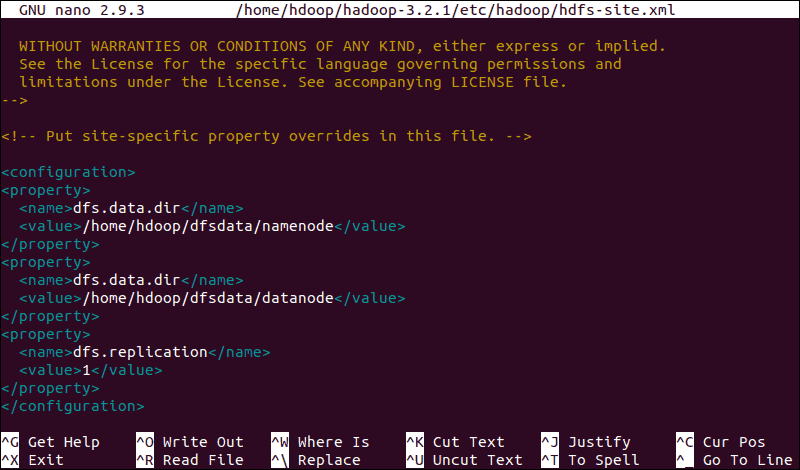
<name>dfs.replication</name>

<value>1</value>

</property>

</configuration>

If necessary, create the specific directories you defined for the dfs.data.dir value.



Edit mapred-site.xml File

Use the following command to access the mapred-site.xml file and define MapReduce values:

$ sudo nano $HADOOP\_HOME/etc/hadoop/mapred-site.xml

Add the following configuration to change the default MapReduce framework name value to yarn:

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>



### Edit yarn-site.xml File

The yarn-site.xml file is used to define settings relevant to YARN. It contains configurations for the Node Manager, Resource Manager, Containers, and Application Master.

Open the yarn-site.xml file in a text editor:

$ sudo nano $HADOOP\_HOME/etc/hadoop/yarn-site.xml

Append the following configuration to the file:

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.hostname</name>

<value>127.0.0.1</value>

</property>

<property>

<name>yarn.acl.enable</name>

<value>0</value>

</property>

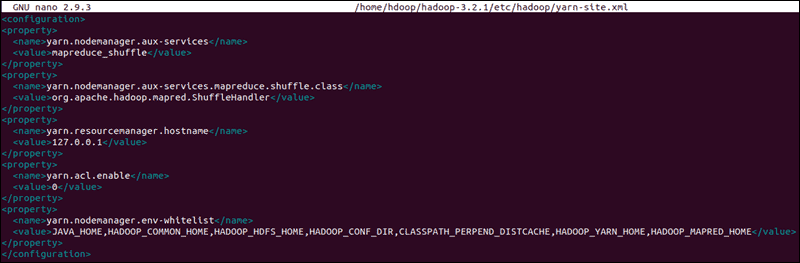
<property>

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PERPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>

</property>

</configuration>

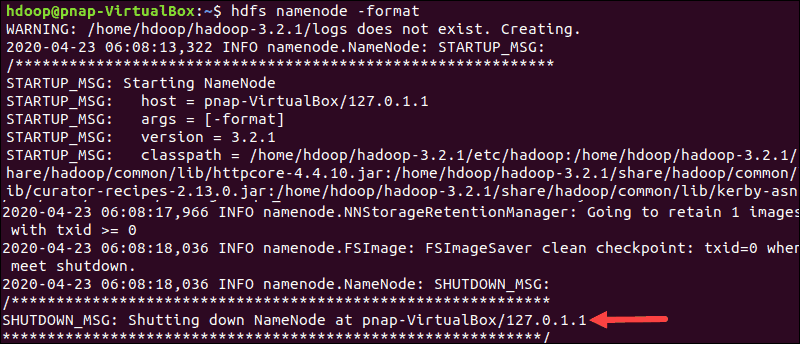


## Format HDFS NameNode

It is important to format the NameNode before starting Hadoop services for the first time:

$ hdfs namenode -format

The shutdown notification signifies the end of the NameNode format process.

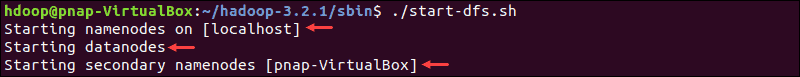


## Start Hadoop Cluster

Navigate to the hadoop-3.2.1/sbin directory and execute the following commands to start the NameNode and DataNode:

./start-dfs.sh

The system takes a few moments to initiate the necessary nodes.



Once the namenode, datanodes, and secondary namenode are up and running, start the YARN resource and nodemanagers by typing:

$ ./start-yarn.sh

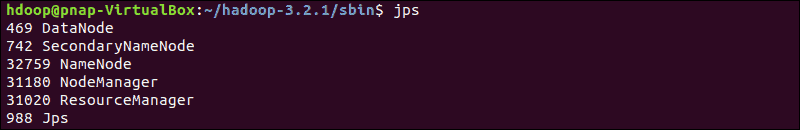
As with the previous command, the output informs you that the processes are starting.



Type this simple command to check if all the daemons are active and running as Java processes:

$ jps

If everything is working as intended, the resulting list of running Java processes contains all the HDFS and YARN daemons.

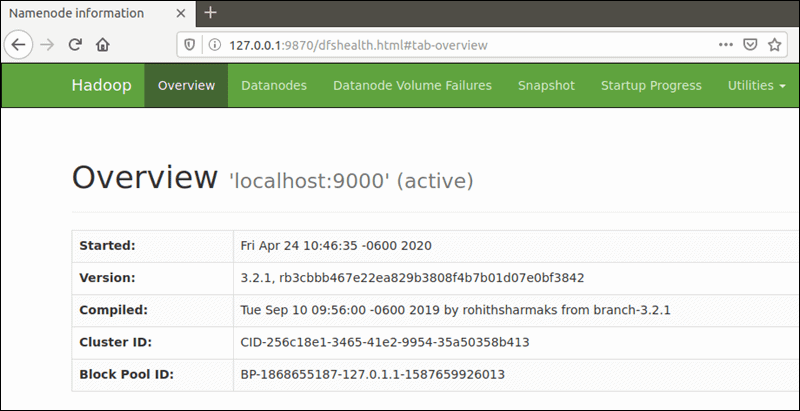


## Access Hadoop UI from Browser

Use your preferred browser and navigate to your localhost URL or IP. The default port number 9870 gives you access to the Hadoop NameNode UI:

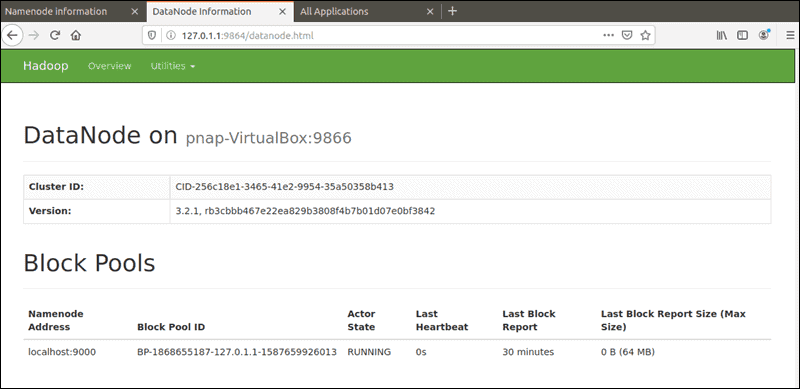
http://localhost:9870

The NameNode user interface provides a comprehensive overview of the entire cluster.



The default port 9864 is used to access individual DataNodes directly from your browser:

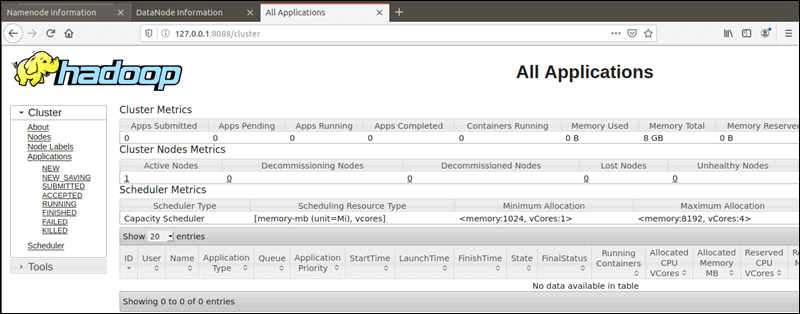
http://localhost:9864



The YARN Resource Manager is accessible on port 8088:

http://localhost:8088

The Resource Manager is an invaluable tool that allows you to monitor all running processes in your Hadoop cluster.



To stop services Navigate to the hadoop-3.2.1/sbin directory and execute the following commands to stop the NameNode and DataNode:

$ ./stop-yarn.sh

$ ./stop-dfs.sh

**Most commonly used commands in HDFS**

To print the Hadoop running processes

**$ jps**

To check the health of the HDFS file system - fsck

**$  hdfs fsck /**

To list all the files in HDFS directory (/)

**$ hdfs dfs -ls /**

To Make a directory in HDFS

**$ hdfs dfs -mkdir /directory\_name**

To create an empty file in HDFS directory

**$ hdfs dfs -touchz filename**

To check the size of the file - du

**$ hdfs dfs -du -s file\_name**

To add the content to the existing file - appendToFile

**$ hdfs dfs -appendToFile - filename**

## press ctrl+D to exit

To display contents of the file present in the hdfs - cat

**$ hdfs dfs -cat file\_name**

To copy a file from the local filesystem to HDFS- copyFromLocal/(-put)

**$hdfs dfs -copyFromLocal local\_file hdfs\_directory**

**$hdfs dfs -put local\_file hdfs\_directory**

To copy a file from HDFS to local filesystem - copyToLocal/(-get)

**$ hdfs dfs -copyToLocal hdfs\_local\_file local\_path\_file**

**$ hdfs dfs -get hdfs\_local\_file local\_path\_file**

Copying within the same HDFS file system

**$hdfs dfs -cp hdfs\_local\_file\_path hdfs\_second\_local\_path**

To move a file within the HDFS file system/ used for renaming too

**$hdfs dfs -mv local\_filesystem\_file new\_local\_file\_path**

rm - remove command - remove the files

**$ hdfs dfs -rm hdfs\_file\_name**

rm -rf - remove forcefully - remove directories

**$ hdfs dfs -rm -r hdfs\_directory**

To get help - usage command - provides help for commands

**$ hdfs dfs -usage command\_name**

**$ hdfs dfs -help - shows all the commands in hdfs**

**Experiment 3: Map reduce program for word count**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

StringTokenizer itr = new StringTokenizer(value.toString());

while (itr.hasMoreTokens()) {

word.set(itr.nextToken());

context.write(word, one);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Experiment 4:**

package my.mapred.pack;

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class TransactionCount {

//MAPPER CODE

public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

//private Text word = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

String myString = value.toString();

String[] userCount = myString.split(",");

output.collect(new Text(userCount[3]), one);

}

}

//REDUCER CODE

public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException { //{little: {1,1}}

int finaluserCount = 0 ;

Text mykey = key ;

while(values.hasNext()) {

IntWritable value = values.next();

finaluserCount += value.get();

}

output.collect(mykey, new IntWritable(finaluserCount));

}

}

//DRIVER CODE

public static void main(String[] args) throws Exception {

JobConf conf = new JobConf(TransactionCount.class);

conf.setJobName("wordcount");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);

conf.setCombinerClass(Reduce.class);

conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class); // hadoop jar jarname classpath inputfolder outputfolder

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

**Experiment 5: HIVE**

Apache Hive Introduction

## What is Hive?

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

**Hive is not**

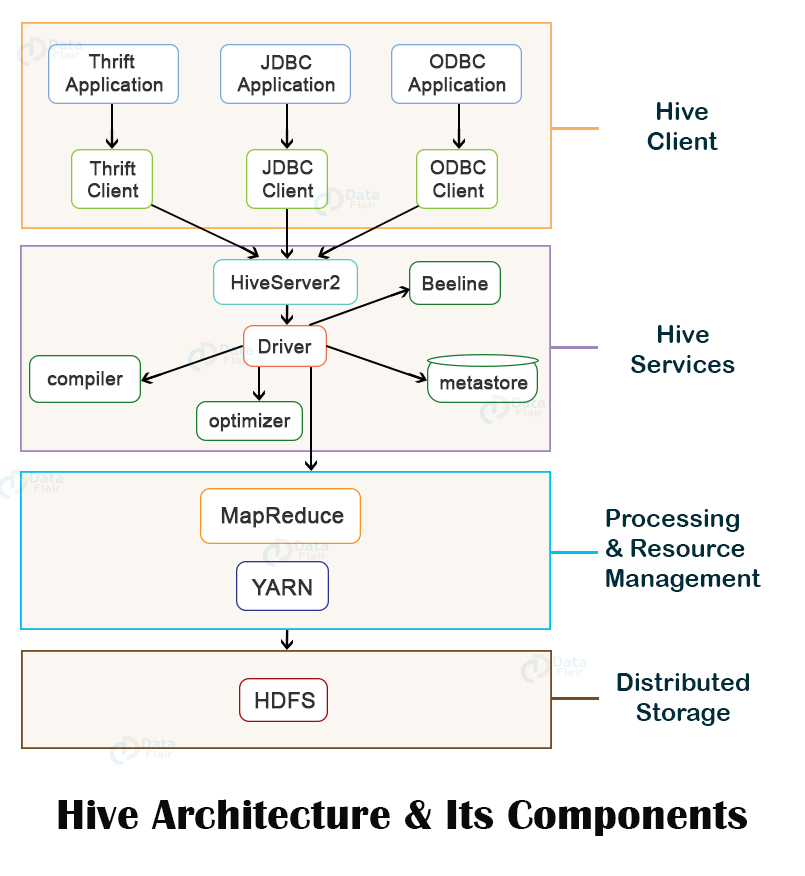
* A relational database
* A design for OnLine Transaction Processing (OLTP)
* A language for real-time queries and row-level updates

**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.

Apache Hive is an enterprise [data warehouse](https://phoenixnap.com/kb/data-warehouse-architecture-explained) system used to query, manage, and analyze data stored in the [Hadoop Distributed File System](https://phoenixnap.com/kb/what-is-hdfs).

* The Hive Query Language (HiveQL) facilitates queries in a Hive command-line interface shell. Hadoop can use HiveQL as a bridge to communicate with relational database management systems and perform tasks based on SQL-like commands.



**Hive Client**

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

* Thrift Server - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
* JDBC Driver - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
* ODBC Driver - It allows the applications that support the ODBC protocol to connect to Hive.

## Hive Services

The following are the services provided by Hive:-

* Hive CLI - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
* Hive Web User Interface - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
* Hive MetaStore - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
* Hive Server - It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
* Hive Driver - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
* Hive Compiler - The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
* Hive Execution Engine - Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Install Apache Hive on Ubuntu 20.04.

Prerequisites

Apache Hive is based on Hadoop and requires a fully functional Hadoop framework.

## Install Apache Hive on Ubuntu

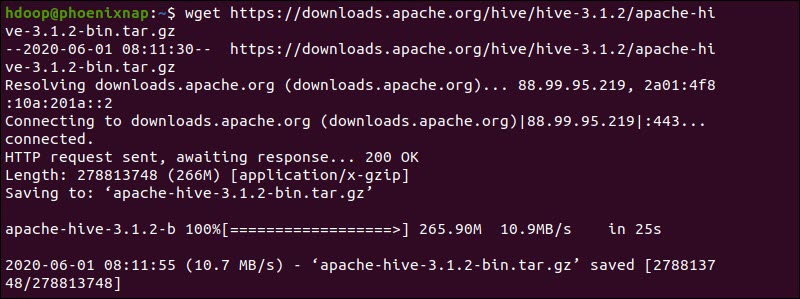
To configure Apache Hive, first you need to download and unzip Hive. Then you need to customize the following files and settings:

* + - Edit .bashrc file
    - Edit hive-config.sh file
    - Create Hive directories in HDFS
    - Configure hive-site.xml file
    - Initiate Derby database

### Step 1: Download and Untar Hive

Access your Ubuntu command line and download the compressed Hive files using and the wget command followed by the download path:

$ wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz



Once the download process is complete, untar the compressed Hive package:

Step 2: Configure Hive Environment Variables (bashrc)

The $HIVE\_HOME environment variable needs to direct the client shell to the apache-hive-3.1.2-bin directory. Edit the .bashrc shell configuration file using a text editor of your choice (we will be using nano):

$ sudo nano .bashrc

Append the following Hive environment variables to the .bashrc file:

export HIVE\_HOME= “home/hdoop/apache-hive-3.1.2-bin”

export PATH=$PATH:$HIVE\_HOME/bin

The Hadoop environment variables are located within the same file.

$ tar xzf apache-hive-3.1.2-bin.tar.gz

The Hive binary files are now located in the apache-hive-3.1.2-bin directory.



Save and exit the .bashrc file once you add the Hive variables. Apply the changes to the current environment with the following command:

$ source ~/.bashrc

### Step 3: Edit hive-config.sh file

Apache Hive needs to be able to interact with the Hadoop Distributed File System. Access the hive-config.sh file using the previously created $HIVE\_HOME variable:

$ sudo nano $HIVE\_HOME/bin/hive-config.sh

Add the HADOOP\_HOME variable and the full path to your Hadoop directory:

export HADOOP\_HOME=/home/hdoop/hadoop-3.2.1



Save the edits and exit the hive-config.sh file.

### Step 4: Create Hive Directories in HDFS

Create two separate directories to store data in the HDFS layer:

* + - The temporary, tmp directory is going to store the intermediate results of Hive processes.
    - The warehouse directory is going to store the [Hive related tables](https://phoenixnap.com/kb/hive-create-table).

#### Create tmp Directory

Create a tmp directory within the HDFS storage layer. This directory is going to store the intermediary data Hive sends to the HDFS:

$ hdfs dfs -mkdir /tmp

Add write and execute permissions to tmp group members:

$ hdfs dfs -chmod g+w /tmp

Check if the permissions were added correctly:

$ hdfs dfs -ls /

The output confirms that users now have write and execute permissions.



#### Create warehouse Directory

Create the warehouse directory within the /user/hive/ parent directory:

$ hdfs dfs -mkdir -p /user/hive/warehouse

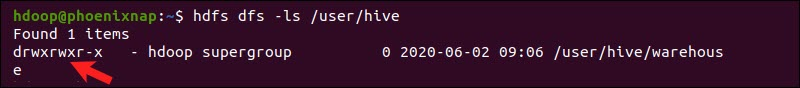
Add write and execute permissions to warehouse group members:

$ hdfs dfs -chmod g+w /user/hive/warehouse

Check if the permissions were added correctly:

$ hdfs dfs -ls /user/hive

The output confirms that users now have write and execute permissions.



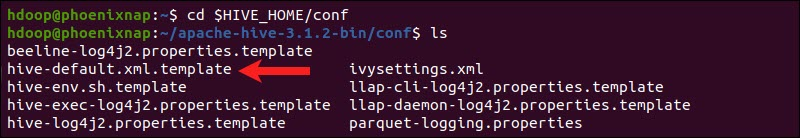
### Step 5: Configure hive-site.xml File (Optional)

Apache Hive distributions contain template configuration files by default. The template files are located within the Hive conf directory and outline default Hive settings.

Use the following command to locate the correct file:

$ cd $HIVE\_HOME/conf

List the files contained in the folder using the ls command.



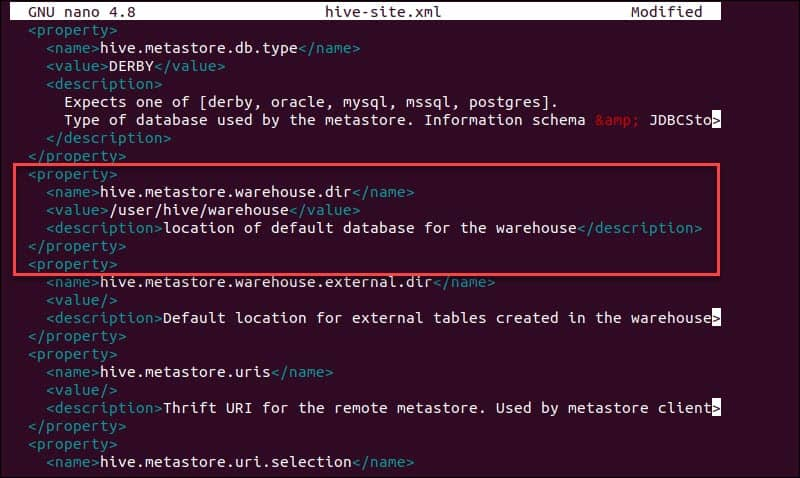
Use the hive-default.xml.template to create the hive-site.xml file:

$ cp hive-default.xml.template hive-site.xml

Access the hive-site.xml file using the nano text editor:

$ sudo nano hive-site.xml

Using Hive in a stand-alone mode rather than in a real-life Apache Hadoop cluster is a safe option for newcomers. You can configure the system to use your local storage rather than the HDFS layer by setting the hive.metastore.warehouse.dir parameter value to the location of your Hive warehouse directory.

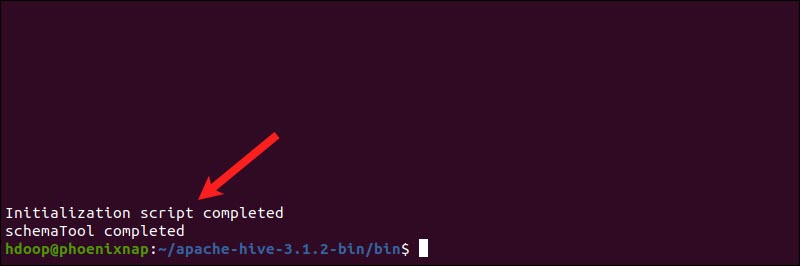


### Step 6: Initiate Derby Database

Apache Hive uses the Derby database to store metadata. Initiate the Derby database, from the Hive bin directory using the schematool command:

$ HIVE\_HOME/bin/schematool –initSchema –dbType derby

The process can take a few moments to complete.



Derby is the default metadata store for Hive. If you plan to use a different database solution, such as [MySQL or PostgreSQL](https://phoenixnap.com/kb/postgres-vs-mysql), you can specify a database type in the hive-site.xml file.

#### How to Fix guava Incompatibility Error in Hive

If the Derby database does not successfully initiate,  you might receive an error with the following content:

“Exception in thread “main” java.lang.NoSuchMethodError: com.google.common.base.Preconditions.checkArgument(ZLjava/lang/String;Ljava/lang/Object;)V”

This error indicates that there is most likely an incompatibility issue between Hadoop and Hive guava versions.

Locate the guava jar file in the Hive lib directory:

$ ls $HIVE\_HOME/lib



Locate the guava jar file in the Hadoop lib directory as well:

$ ls $HADOOP\_HOME/share/hadoop/hdfs/lib



The two listed versions are not compatible and are causing the error. Remove the existing guava file from the Hive lib directory:

$ rm $HIVE\_HOME/lib/guava-19.0.jar

Copy the guava file from the Hadoop lib directory to the Hive lib directory:

$ cp $HADOOP\_HOME/share/hadoop/hdfs/lib/guava-27.0-jre.jar $HIVE\_HOME/lib/

Use the schematool command once again to initiate the Derby database:

$ HIVE\_HOME/bin/schematool –initSchema –dbType derby

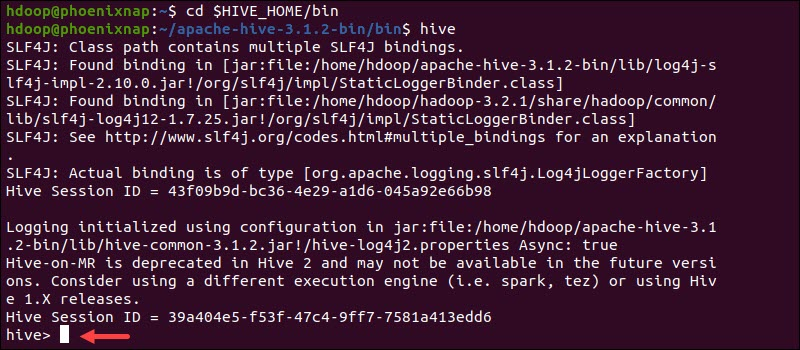
## Launch Hive Client Shell on Ubuntu

Start the Hive command-line interface using the following commands:

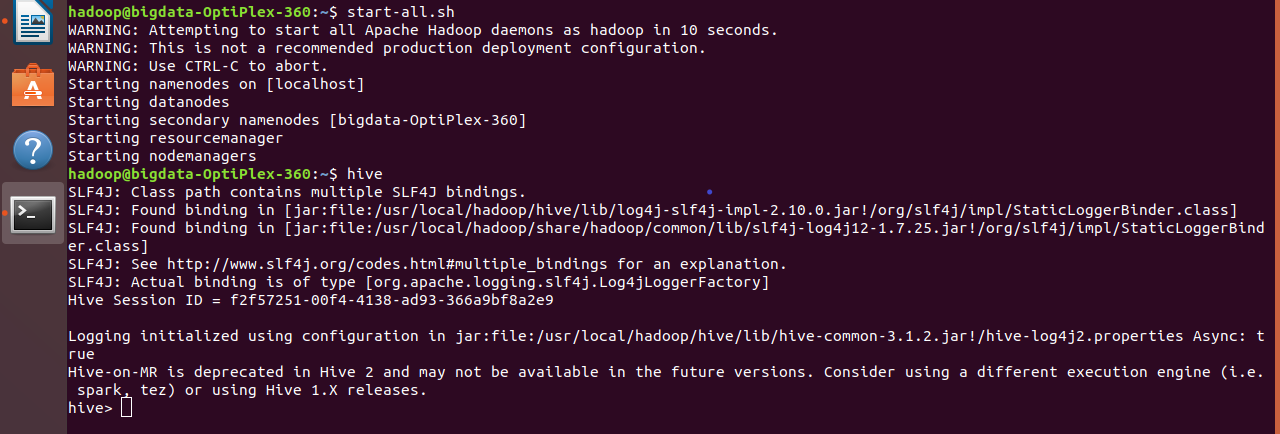
$ cd $HIVE\_HOME/bin

hive

You are now able to issue SQL-like commands and directly interact with HDFS.



Start-all.sh



**Create Database Statement**

Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows:

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

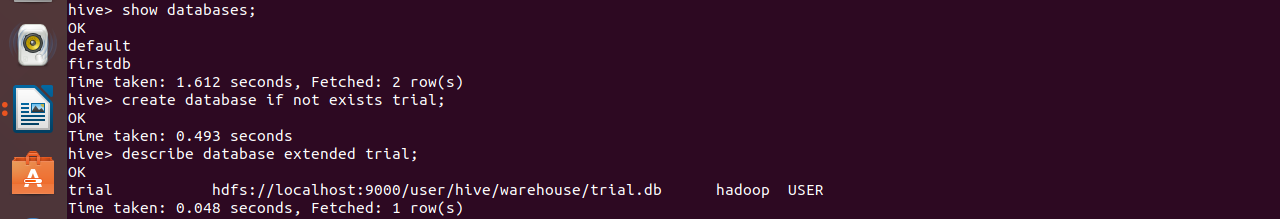
Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named userdb:

hive> CREATE DATABASE [IF NOT EXISTS] userdb;

or

hive> CREATE SCHEMA userdb;

create database if not exists firstdb ;



**show databases;**



**Drop Database Statement**

Drop Database is a statement that drops all the tables and deletes the database.

Its syntax is as follows:

DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database\_name

[RESTRICT|CASCADE];

The following queries are used to drop a database. Let us assume that the database name is userdb.

hive> DROP DATABASE IF EXISTS userdb;

The following query drops the database using CASCADE. It means dropping respective tables before dropping the database.

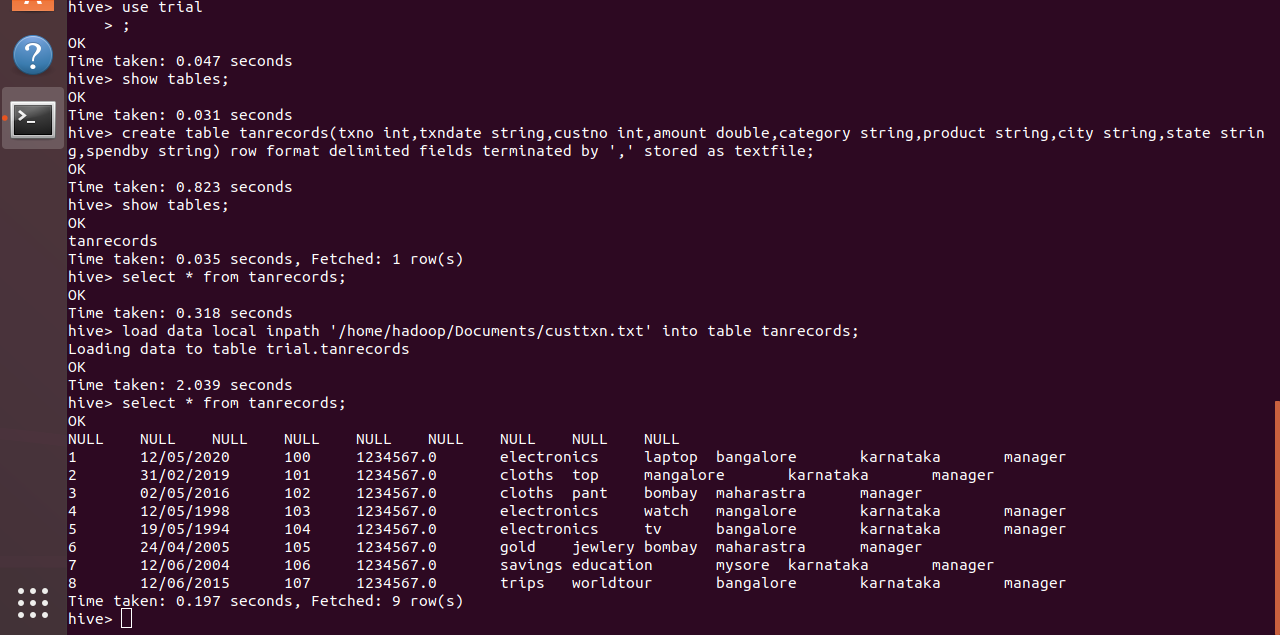
hive> DROP DATABASE IF EXISTS userdb CASCADE;

The following query drops the database using SCHEMA.

hive> DROP SCHEMA userdb;

This clause was added in Hive 0.6.

Use database\_name; (example  -> use trial;)



**Show tables;**

**Create Table Statement**

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

**Syntax**

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.] table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[ROW FORMAT row\_format]

[STORED AS file\_format]

**Example**

Let us assume you need to create a table named employee using CREATE TABLE statement. The following table lists the fields and their data types in employee table:

|  |  |  |
| --- | --- | --- |
| Sr.No | Field Name | Data Type |
| 1 | Eid | int |
| 2 | Name | String |
| 3 | Salary | Float |
| 4 | Designation | string |

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

COMMENT ‘Employee details’

FIELDS TERMINATED BY ‘\t’

LINES TERMINATED BY ‘\n’

STORED IN TEXT FILE

The following query creates a table named employee using the above data.

hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,

salary String, destination String)

COMMENT ‘Employee details’

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ‘\t’

LINES TERMINATED BY ‘\n’

STORED AS TEXTFILE;

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

On successful creation of table, you get to see the following response:

OK

Time taken: 5.905 seconds

**hive>** create external table geography (anonid int,eprofileclass int,fueltypes string,acorn\_category int,acorn\_group string,acorn\_type int,nuts4 string,lacode string,nutsl string,gspgroup string,ldz string,gas\_elec string,gas\_tout string)row format delimited fields terminated by ',' stored as textfile ;

**desc allgas;**

create external table allgas ( anon\_id int,advancedatetime string,hh int ,gaskwh double)row format delimited fields terminated by ',' stored as textfile;

create external table geography (anonid int,eprofileclass int,fueltypes string,acorn\_category int,acorn\_group string,acorn\_type int,nuts4 string,lacode string,nutsl string,gspgroup string,ldz string,gas\_elec string,gas\_tout string)row format delimited fields terminated by ',' stored as textfile ;

**Simple Selects – Selecting columns**

In Hive, querying data is performed by a SELECT statement. A select statement has 6 key components;

* SELECT column names
* FROM table-name
* GROUP BY column names
* WHERE conditions
* HAVING conditions
* ORDER by column names

Amongst all the hive queries, the simplest query is effectively one which returns the contents of the whole table

Select \* from geog\_all;

## Simple selects – selecting rows

In addition to limiting the columns returned by a query, you can also limit the rows returned. The simplest case is to say how many rows are wanted using the Limit clause.

SELECT anonid, fueltypes, acorn\_type

FROM geog\_all

LIMIT 10;

This is useful if you just want to get a feel for what the data looks like. Usually, you will want to restrict the rows returned based on some criteria. i.e. certain values or ranges within one or more columns.

SELECT anonid, fueltypes, acorn\_type

FROM geog\_all

WHERE fueltypes = "ElecOnly";

The Expression in the where clause can be more complex and involve more than one column.

SELECT anonid, fueltypes, acorn\_type

FROM geog\_all

WHERE fueltypes = "ElecOnly" AND acorn\_type > 42;

SELECT anonid, fueltypes, acorn\_type

FROM geog\_all

WHERE fueltypes = "ElecOnly" AND acorn\_type > 42 AND nuts1 <> "--";

## Creating new columns

It is possible to create new columns in the output of the query. These columns can be from combinations from the other columns using operators and/or built-in Hive functions.

SELECT anonid, eprofileclass, acorn\_type, (eprofileclass \* acorn\_type) AS multiply, (eprofileclass + acorn\_type) AS added

FROM edrp\_geography\_data b;

## Hive Functions

### Simple functions

**Concat** can be used to add strings together

SELECT anonid, acorn\_category,

acorn\_group,

acorn\_type,

concat (acorn\_category, ",", acorn\_group, ",", acorn\_type) AS acorn\_code

FROM geog\_all;

**substr** can be used to extract a part of a string

SELECT anon\_id,

advancedatetime,

substr (advancedatetime, 1, 2) AS day,

substr (advancedatetime, 3, 3) AS month,

substr (advancedatetime, 6, 2) AS year

FROM elec\_c;

**Examples of length, instr, and reverse**

SELECT anonid,

acorn\_code,

length (acorn\_code),

instr (acorn\_code, ',') AS a\_catpos,

instr (reverse (acorn\_code), "," ) AS reverse\_a\_typepo

Where needed functions can be nested within each other cast and type conversions

SELECT anonid,

substr (acorn\_code, 7, 2) AS ac\_type\_string,

cast (substr (acorn\_code, 7, 2) AS INT) AS ac\_type\_int,

substr (acorn\_code, 7, 2) +1 AS ac\_type\_not\_sure

FROM geog\_all;

### Aggregations

### Aggregate functions are used to perform some kind of mathematical or statistical calculation across a group of rows. The rows in each group are determined by the different values in a specified column or columns. A list of all of the available functions is available in the apache documentation.

### SELECT anon\_id,

### count (eleckwh) AS total\_row\_count,

### sum (eleckwh) AS total\_period\_usage,

### min (eleckwh) AS min\_period\_usage,

### avg (eleckwh) AS avg\_period\_usage,

### max (eleckwh) AS max\_period\_usage

### FROM elec\_c

### GROUP BY anon\_id;

### In the above example, five aggregations were performed over the single column anon\_id. It is possible to aggregate over multiple columns by specifying them in both the select and the group by clause. The grouping will take place based on the order of the columns listed in the group by clause. What is not allowed is specifying a non‐aggregated column in the select clause which is not mentioned in the group by clause.

### SELECT anon\_id,

### substr (advancedatetime, 6, 2) AS reading\_year,

### count (eleckwh) AS total\_row\_count,

### sum (eleckwh) AS total\_period\_usage,

### min (eleckwh) AS min\_period\_usage,

### avg (eleckwh) AS avg\_period\_usage,

### max (eleckwh) AS max\_period\_usage

### FROM elec\_c

### GROUP BY anon\_id, substr (advancedatetime, 6, 2);

