Unit 2: Big Data Tools I

UNIT 2 BIG DATA TOOLS I **9 Hrs.**

Big Data Applications using Pig and Hive – Fundamentals of HBase and ZooKeeper – IBM Infosphere Big Insights – Introduction to FLUME – KAFKA.

ECO System Tools:

Apache Hadoop Ecosystem

Zookeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization and providing group services which are very useful for a variety of distributed systems. HBase is not operational without ZooKeeper.

Oozie

- Oozie is a workflow scheduler system to manage Apache Hadoop jobs.
- Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions.
- Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by **time** (frequency) and **data availability**.
- Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java map-reduce, Streaming map-reduce, Pig, Hive, Sqoop and Distcp) as well as system specific jobs (such as Java programs and shell scripts).
- Oozie is a scalable, reliable and extensible system.

Mahout

Mahout is a scalable machine learning library that implements various different approaches machine learning. At present Mahout contains four **main groups of algorithms:**

- Recommendations, also known as collective filtering
- Classifications, also known as categorization
- Clustering
- Frequent itemset mining, also known as parallel frequent pattern mining

Algorithms in the Mahout library belong to the subset that can be executed in a distributed fashion and have been written to be executable in MapReduce. Mahout is scalable along three dimensions: It scales to reasonably large data sets by leveraging algorithm properties or implementing versions based on Apache Hadoop.

Oracle R connector

- The Oracle R Connector for Hadoop (ORCH) provides access to a Hadoop cluster from R, enabling manipulation of HDFS-resident data and the execution of MapReduce jobs.
- MapReduce is similar to combination of apply operations in R or GROUP BY in Oracle Database
- ORCH can be used on the Oracle Big Data Appliance or on non-Oracle Hadoop clusters

Big Data applications using Pig and Hive

Pig

Pig is an open-source high-level dataflow system.

It provides a simple language for queries and data manipulation Pig Latin, that is compiled into map-reduce jobs that are run on Hadoop.

Why is it Important?

→ Companies like Yahoo, Google and Microsoft are collecting enormous data sets in the form of click streams, search logs, and web crawls.

→ Some form of ad-hoc processing and analysis of all of this information is required.

Pig applications

→ Processing of Web Logs.

→ Data processing for search platforms.

→ Support for Ad Hoc queries across large datasets.

→ Quick Prototyping of algorithms for processing large datasets.

Examples of Data Analysis task

Find users who tend to visit "good" pages:

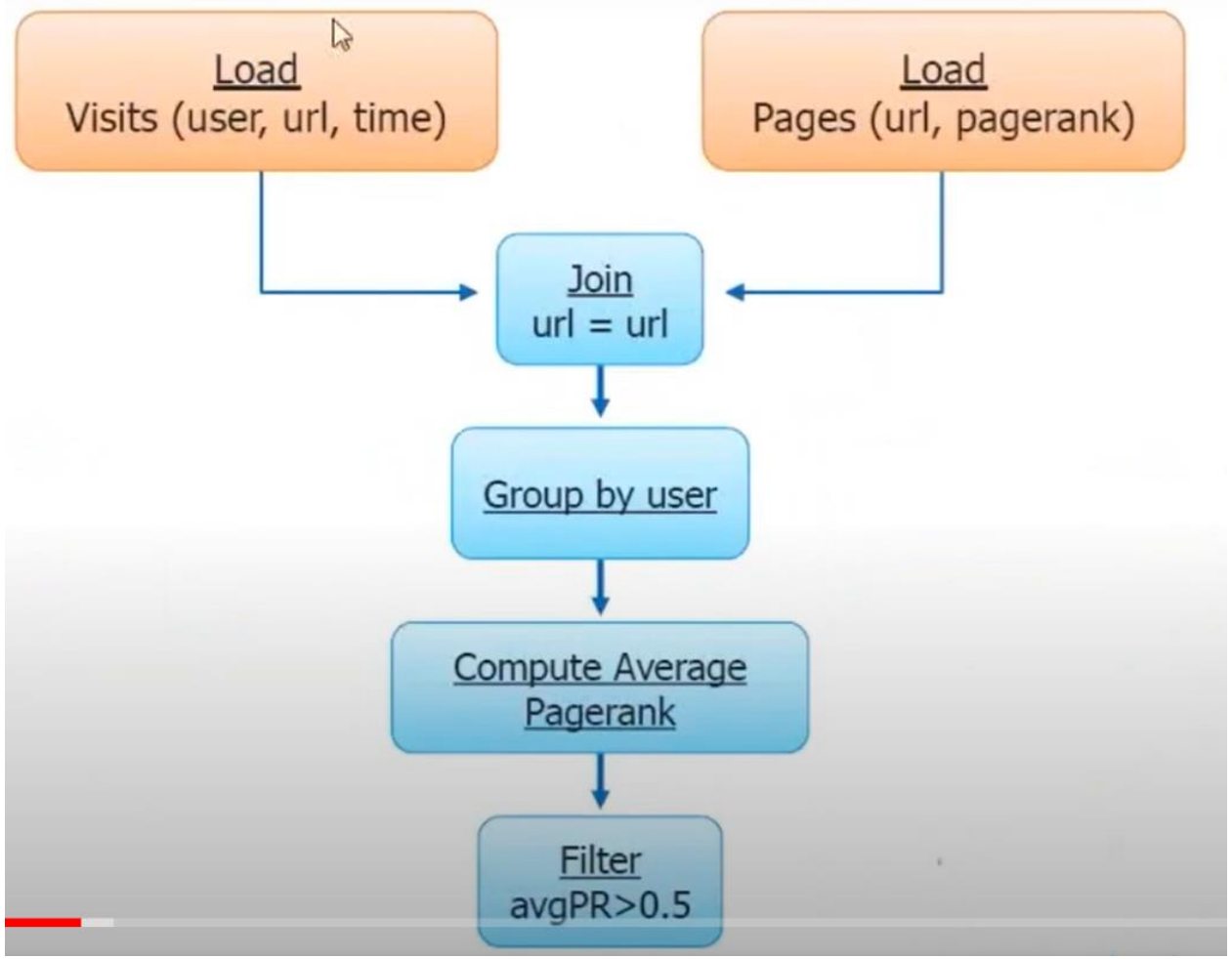
VISITS

User	URL	Time
Amy	www.cnn.com	8:00
Amy	www.crap.com	8:05
Amy	www.myblog.com	10:00
Amy	www.flickr.com	10:05
Fred	cnn.com/index.htm	12:00

PAGES

URL	Page Rank
www.cnn.com	0.9
www.flickr.com	0.9
www.myblog.com	0.7
www.crap.com	0.2

Conceptual data flow



How Yahoo uses Pig

Pig is best suited for the data factory.

Data Factory contains:

Pipelines:

→Pipelines bring logs from Yahoo!'s web servers.
→These logs undergo a cleaning step where bots, company internal views, and clicks are removed.

Research:

→Researchers want to quickly write a script to test a theory.
→Pig integration with streaming makes it easy for researchers to take a Perl or Python script and run it against a huge data set.

Pig

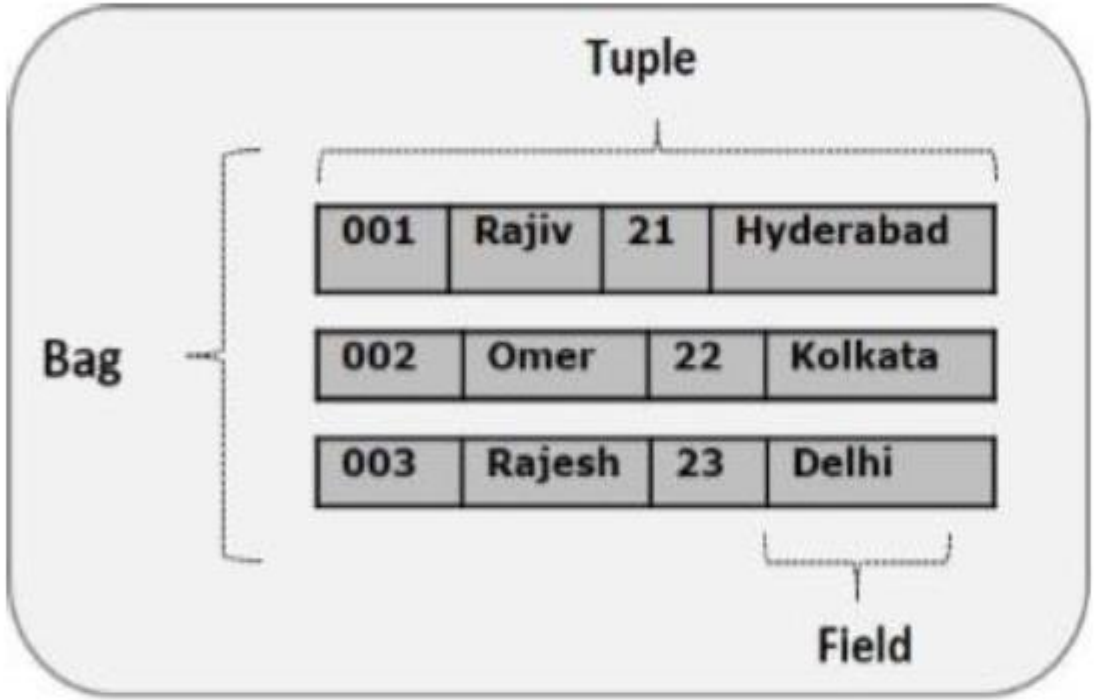
- Pig is a platform for analyzing and querying huge data sets that consist of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.
- Pig's built-in operations can make sense of semi-structured data, such as log files, and the language is extensible
- using Java to add support for custom data types and transformations.

Pig has three main key properties:

- Extensibility
- Optimization opportunities
- Ease of programming

Pig Data Model.

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**.



Types

- Pig's data types can be divided into two categories: *scalar types, which contain a single value, and complex types, which contain other types.*
  Scalar Types
- Pig's scalar types are simple types that appear in most programming languages.
- With the exception of bytearray, they are all represented in Pig interfaces byjava.lang classes, making them easy to work with in UDFs
- *Int,Long ,Float Double, CharArray and byteArray.*

Complex Types

- Pig has three complex data types: maps, tuples, and bags.
- All of these types can contain data of any type, including other complex types.

Map

A *map* in Pig is a chararray to data element mapping, where that element can be any Pig type, including a complex type. The chararray is called a key and is used as an index to find the element, referred to as the value.

Map constants are formed using brackets to delimit the map, a hash between keys and values, and a comma between key-value pairs. For example,['name'#'bob', 'age'#55] will create a map with two keys, "name" and"age". The first value is a chararray, and the second is an integer.

Tuple

A *tuple* is a fixed-length, ordered collection of Pig data elements. Tuples are divided into *fields*, with each field containing one data element. These elements can be of any type—they do not all need to be the same type. A tuple is analogous to a row in SQL, with the fields being SQL columns.

Bag

- A bag is an unordered collection of tuples. Because it has no order, it is not possible to reference tuples in a bag by position.
- Like tuples, a bag can, but is not required to, have a schema associated with it.
- In the case of a bag, the schema describes all tuples within the bag.
- Bag constants are constructed using braces, with tuples in the bag separated by commas.
- For example, {('bob', 55), ('sally', 52), ('john', 25)}constructs a bag with three tuples, each with two fields.

Pig Basic Program structure

Script:

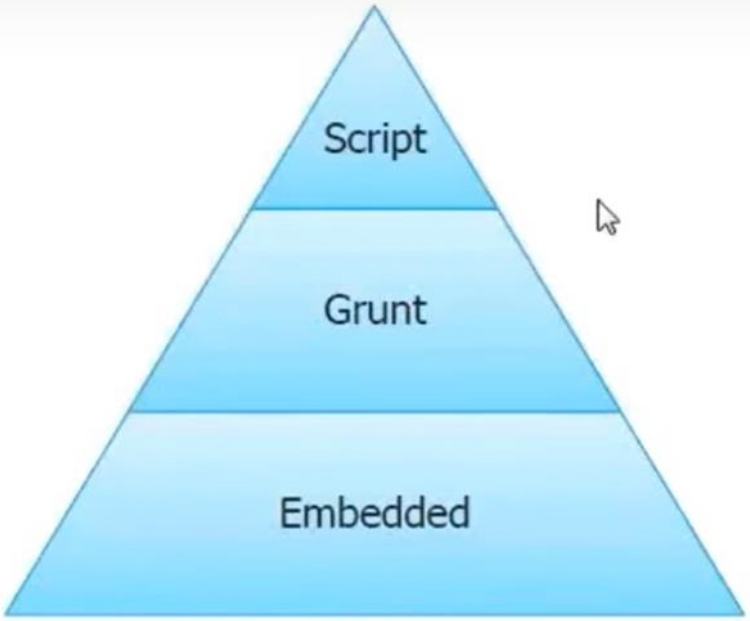
Pig can run a script file that contains Pig commands.

Example: pig script.pig runs the commands in the local file script.pig.

Grunt:
Grunt is an interactive shell for running Pig commands. It is also possible to run Pig scripts from within Grunt using run and exec (execute).

Embedded:

Embedded can run Pig programs from Java, much like you can use JDBC to run SQL programs from Java.



Pig running modes

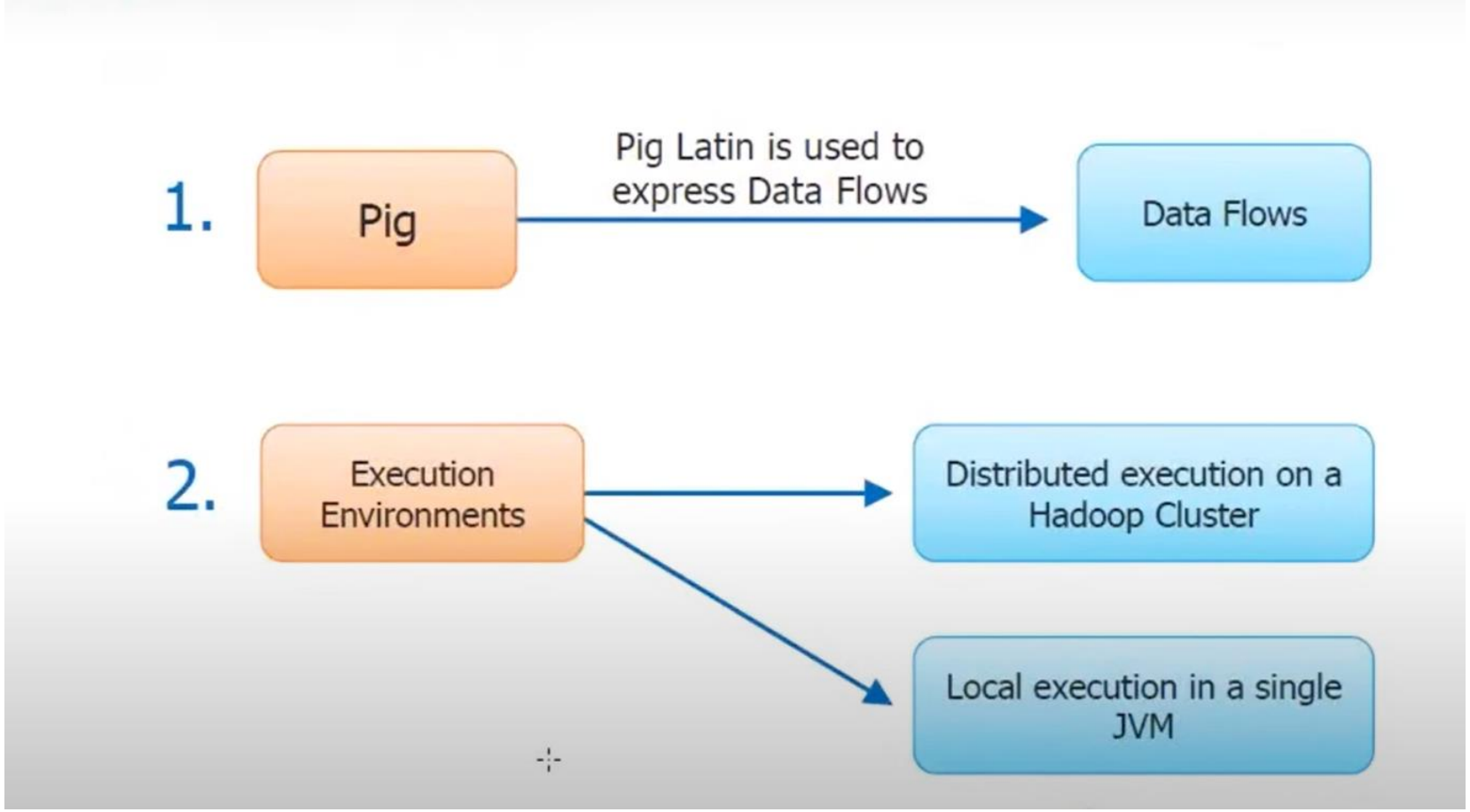
→ Local Mode → pig -x local

```
user@ubuntu:~$ pig -x local
2014-07-25 06:37:43,351 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.1-SNAPSHOT (rexported) compiled Ma
r 22 2014, 07:41:29
2014-07-25 06:37:43,351 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/user/pig_1406284663350.lo
g
2014-07-25 06:37:43,371 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/user/.pigbootup not f
ound
2014-07-25 06:37:43,560 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated
. Instead, use fs.defaultFS
2014-07-25 06:37:43,560 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depreca
ted. Instead, use mapreduce.jobtracker.address
2014-07-25 06:37:43,561 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to h
adoop file system at: file:///
```

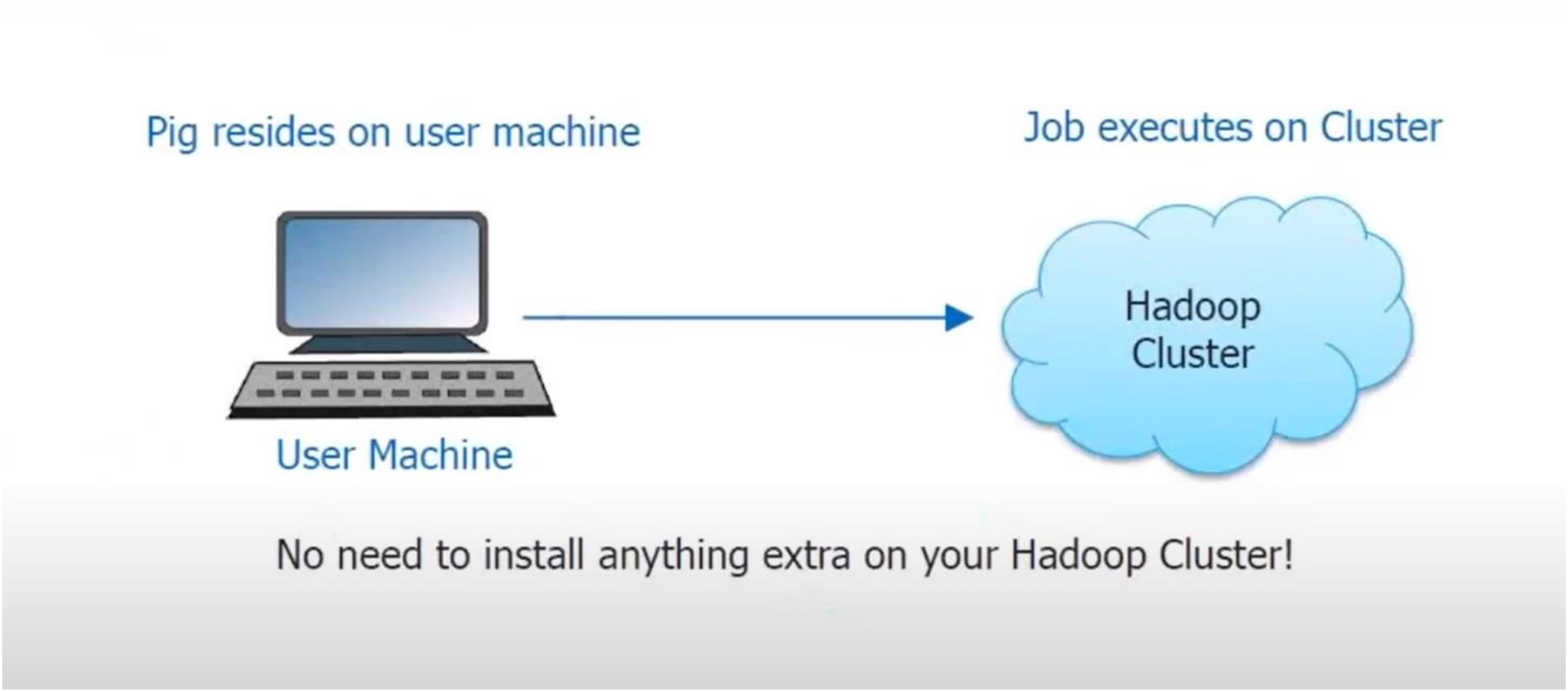
→ MapReduce or HDFS Mode → pig

```
user@ubuntu:~$ pig
2014-07-25 06:34:42,564 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.1-SNAPSHOT (rexported) compiled Ma
r 22 2014, 07:41:29
2014-07-25 06:34:42,565 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/user/pig_1406284482563.lo
g
2014-07-25 06:34:42,583 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/user/.pigbootup not f
ound
2014-07-25 06:34:42,819 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is depreca
ted. Instead, use mapreduce.jobtracker.address
2014-07-25 06:34:42,819 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated
. Instead, use fs.defaultFS
```

Pig Programming components

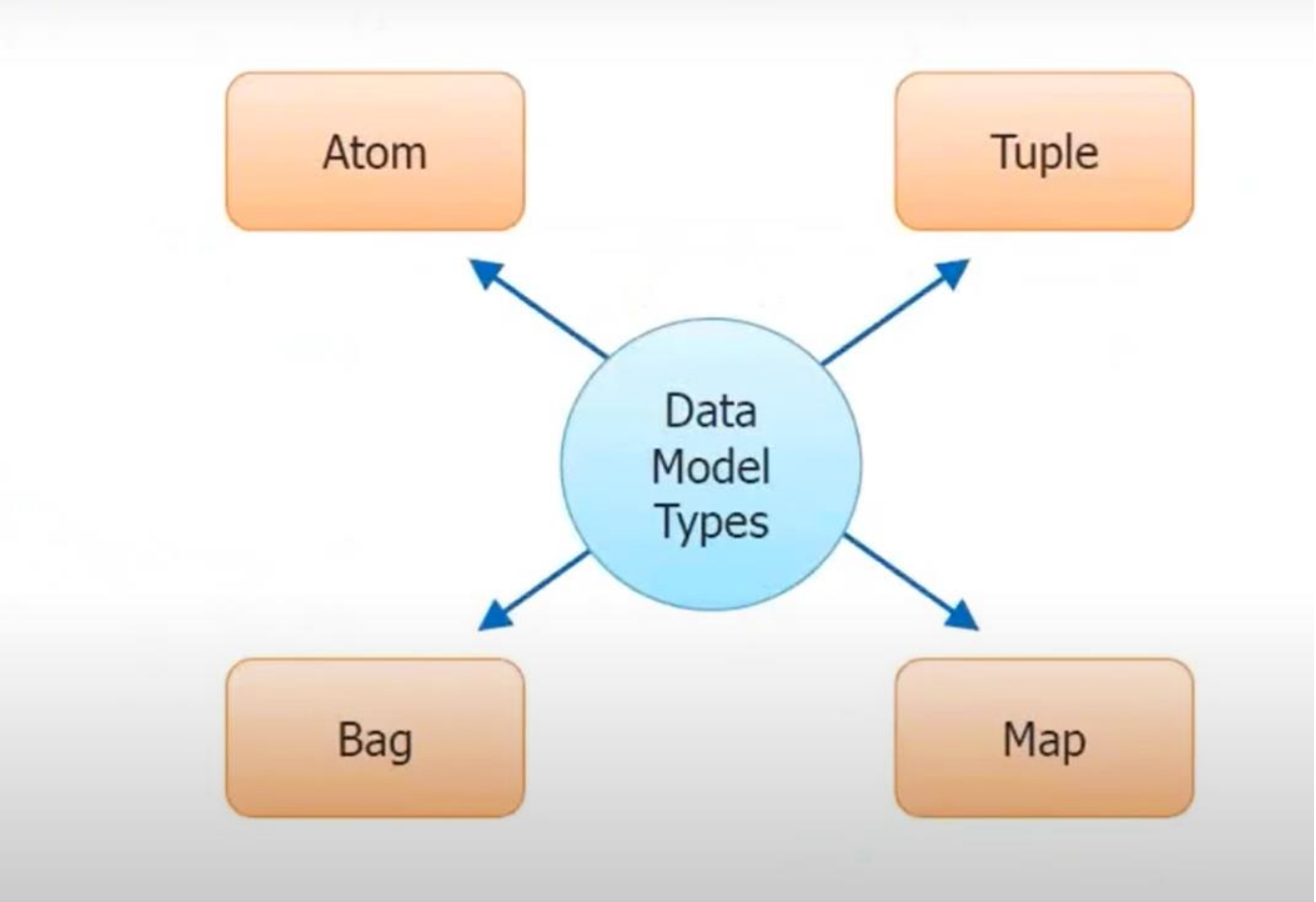


Pig execution



Pig Latin Program

Basic types of Data models



- Atom: Primitive Data type
- Tuple: An ordered list of continuous data items

Data Models

Data Models can be defined as follows:

→A bag is a collection of tuples.

→A tuple is an ordered set of fields.

→A field is a piece of data.

→A Data Map is a map from keys that are string literals to values that can be any data type.

Example

```
t = ( 1, {(2,3),(4,6),(5,7)}, ['apache':'search'] )
```

Pig Latin.

The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a high level data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Pig Latin – Type Construction Operators-

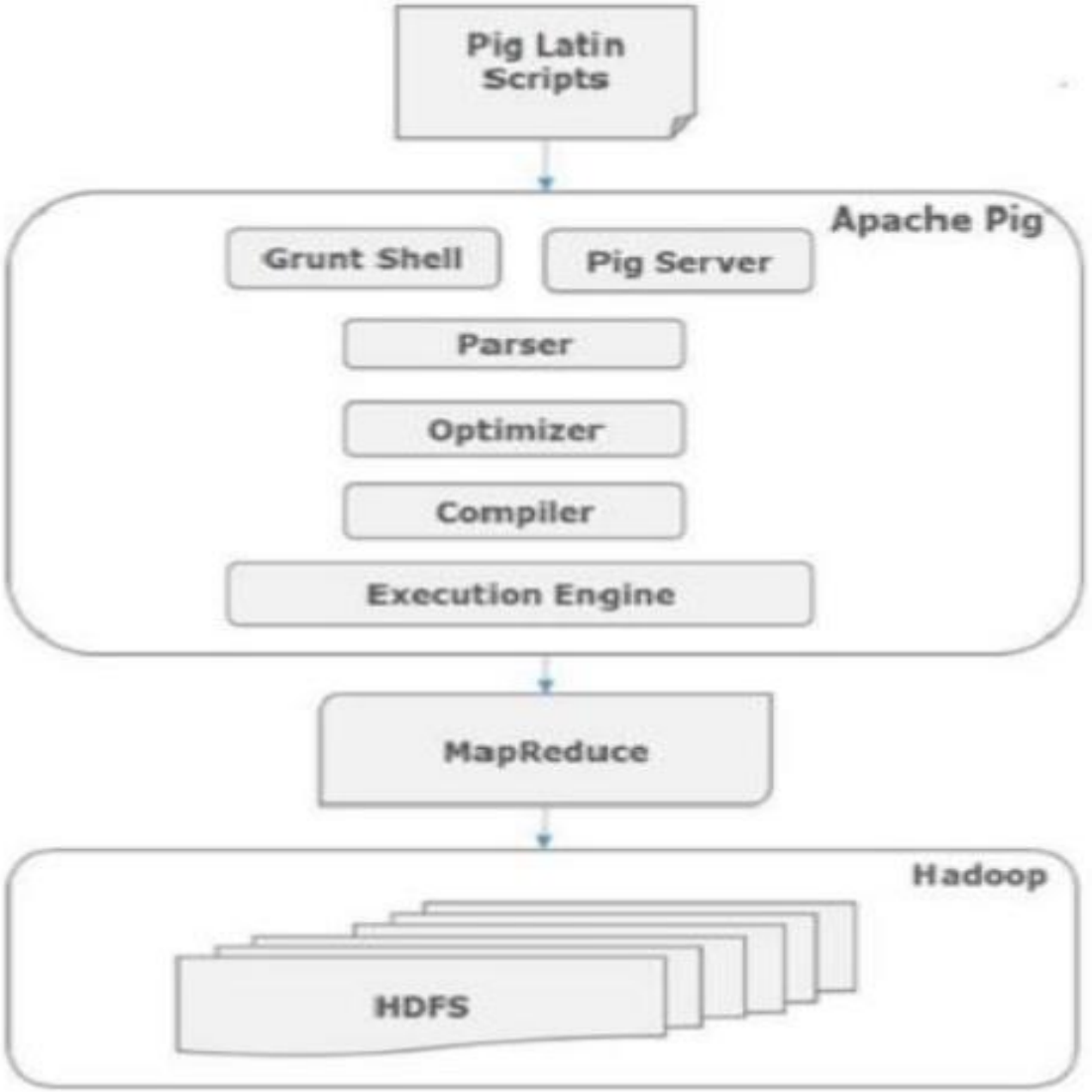
()	**Tuple constructor operator** − This operator is used to construct a tuple.
{}	**Bag constructor operator** − This operator is used to construct a bag.
[]	**Map constructor operator** − This operator is used to construct a tuple.

Pig Latin – Relational Operators

Operator	Description
Loading and Storing	
LOAD	To Load the data from the file system (local/HDFS) into a relation.
STORE	To save a relation to the file system (local/HDFS).
Filtering	
FILTER	To remove unwanted rows from a relation.
Grouping and Joining	
JOIN	To join two or more relations.
COGROUP	To group the data in two or more relations.

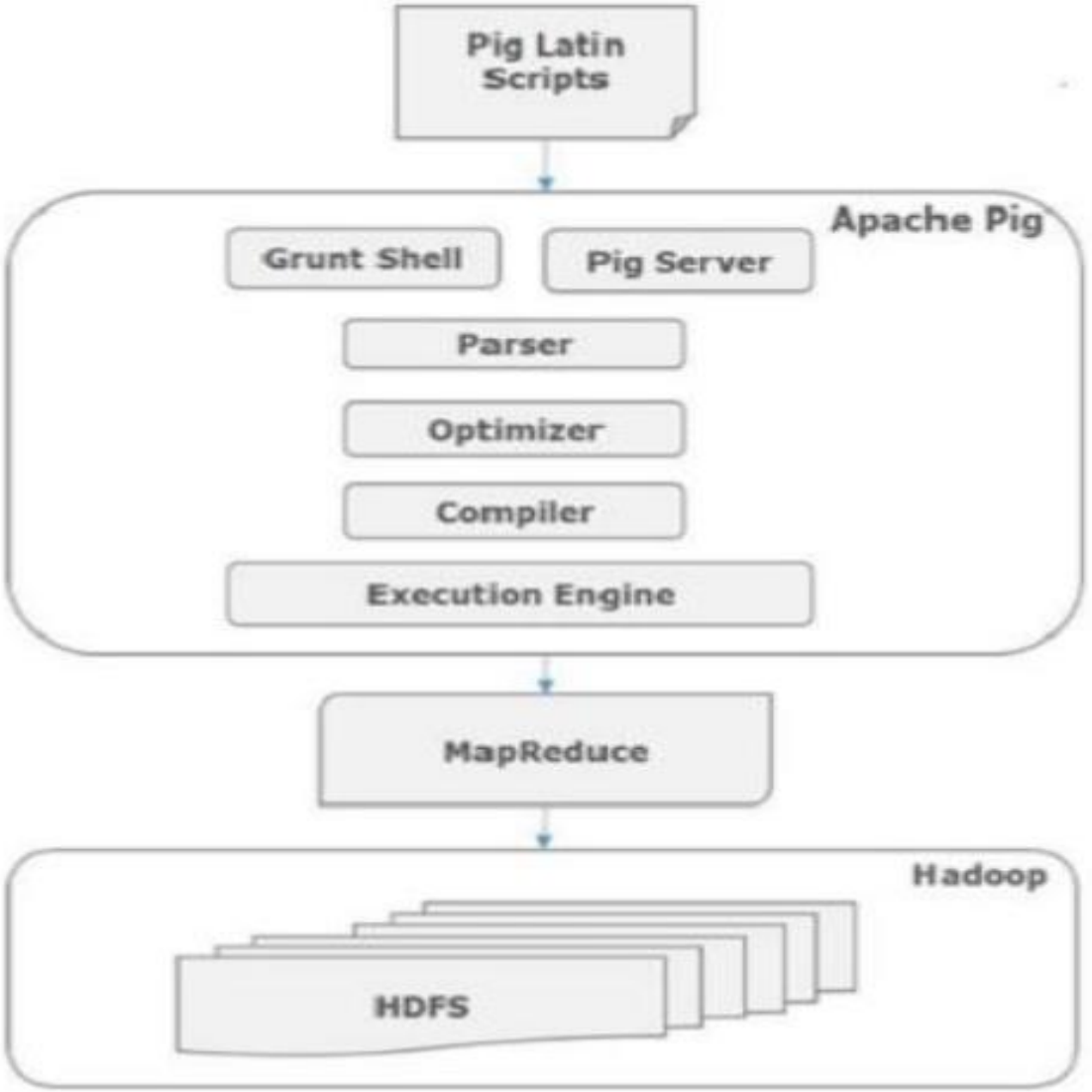
GROUP	To group the data in a single relation.
CROSS	To create the cross product of two or more relations.
Sorting	
ORDER	To arrange a relation in a sorted order based on one or more fields (ascending or descending).
LIMIT	To get a limited number of tuples from a relation.
Combining and Splitting	
UNION	To combine two or more relations into a single relation.
SPLIT	To split a single relation into two or more relations.

Diagnostic Operators	
DUMP	To print the contents of a relation on the console.
DESCRIBE	To describe the schema of a relation.
EXPLAIN	To view the logical, physical, or MapReduce execution plans to compute a relation.
ILLUSTRATE	To view the step-by-step execution of a series of statements.


Pig Latin Scripts
Apache Pig
Grunt Shell
Pig Server
Parser
Optimizer
Compiler
Execution Engine
MapReduce
Hadoop
HDFS

GROUP	To group the data in a single relation.
CROSS	To create the cross product of two or more relations.
Sorting	
ORDER	To arrange a relation in a sorted order based on one or more fields (ascending or descending).
LIMIT	To get a limited number of tuples from a relation.
Combining and Splitting	
UNION	To combine two or more relations into a single relation.
SPLIT	To split a single relation into two or more relations.

Diagnostic Operators	
DUMP	To print the contents of a relation on the console.
DESCRIBE	To describe the schema of a relation.
EXPLAIN	To view the logical, physical, or MapReduce execution plans to compute a relation.
ILLUSTRATE	To view the step-by-step execution of a series of statements.


Pig Latin Scripts
Apache Pig
Grunt Shell
Pig Server
Parser
Optimizer
Compiler
Execution Engine
MapReduce
Hadoop
HDFS

Pig Components

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

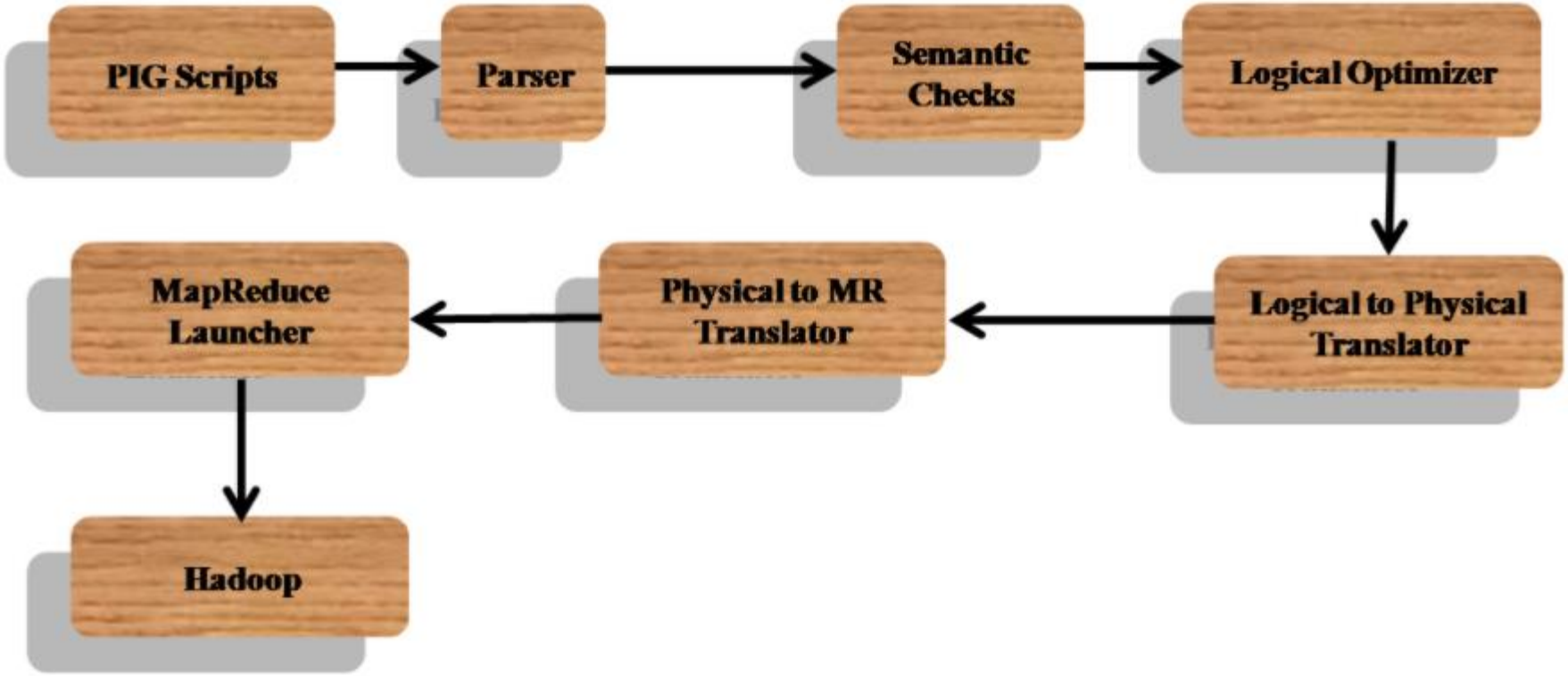
Compiler

The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

Pig Program Flow



Case study:Encoding a word using Pig Latin

- The Problem statement:

Design a program to take a word as an input, and then encode it into Pig Latin. A Pig Latin is an encrypted word in English, which is generated by doing the following alterations:
The first vowel occurring in the input word is placed at the start of the new word along with the remaining alphabet of it. The alphabet is present before the first vowel is shifted, at the end of the new word it is followed by “ay”.

```
Input: s = "paris"
Output: arispay

Input: s = "amazon"
Output: amazonay
```

Algorithm:

1) Find index of first vowel.
2) Create pig latin by appending following three.
    a) Substring after starting with the first vowel .....till end.
    b) Substring before first vowel.
    c) "ay".

Code for the problem: ‘Encoding a word in Pig Latin’

```
// Java program to encode a word to a Pig Latin.
class GFG {
static boolean isVowel(char c) {
    return (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U' ||
            c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}

static String pigLatin(String s) {

    // the index of the first vowel is stored.
    int len = s.length();
    int index = -1;
    for (int i = 0; i < len; i++)
    {
        if (isVowel(s.charAt(i))) {
        index = i;
        break;
    }
    }

    // Pig Latin is possible only if vowels
    // is present
    if (index == -1)
        return "-1";

    // Take all characters after index (including
    // index). Append all characters which are before
    // index. Finally append "ay"
    return s.substring(index) +
           s.substring(0, index) + "ay";
}
```

```
// Driver code
public static void main(String[] args) {
    String str = pigLatin("graphic");
    if (str == "-1")
        System.out.print("No vowels found." +
                         "Pig Latin not possible");

    else
        System.out.print(str);
}
}
```

Output:
aphicgray

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
- 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

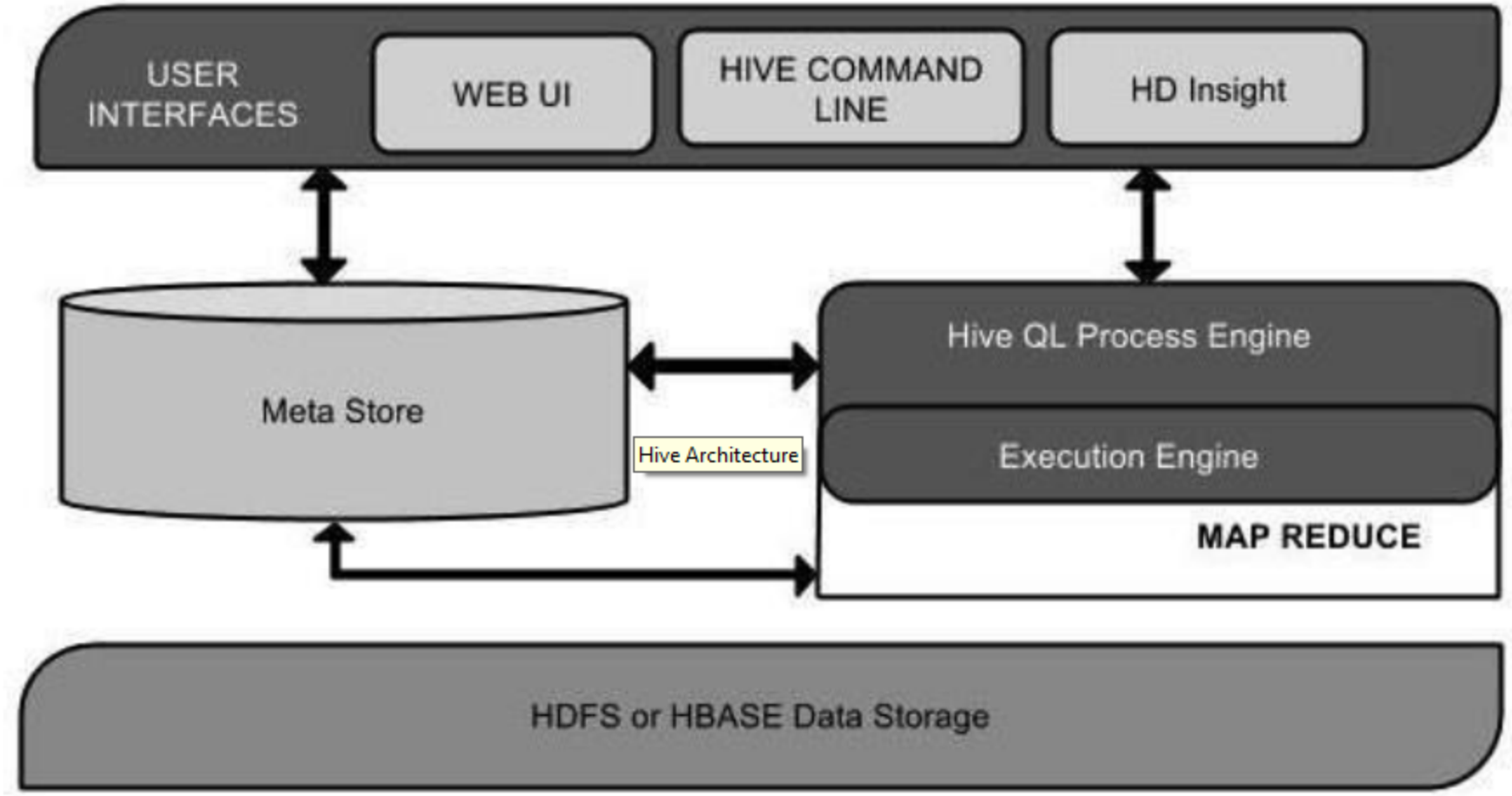
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

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.

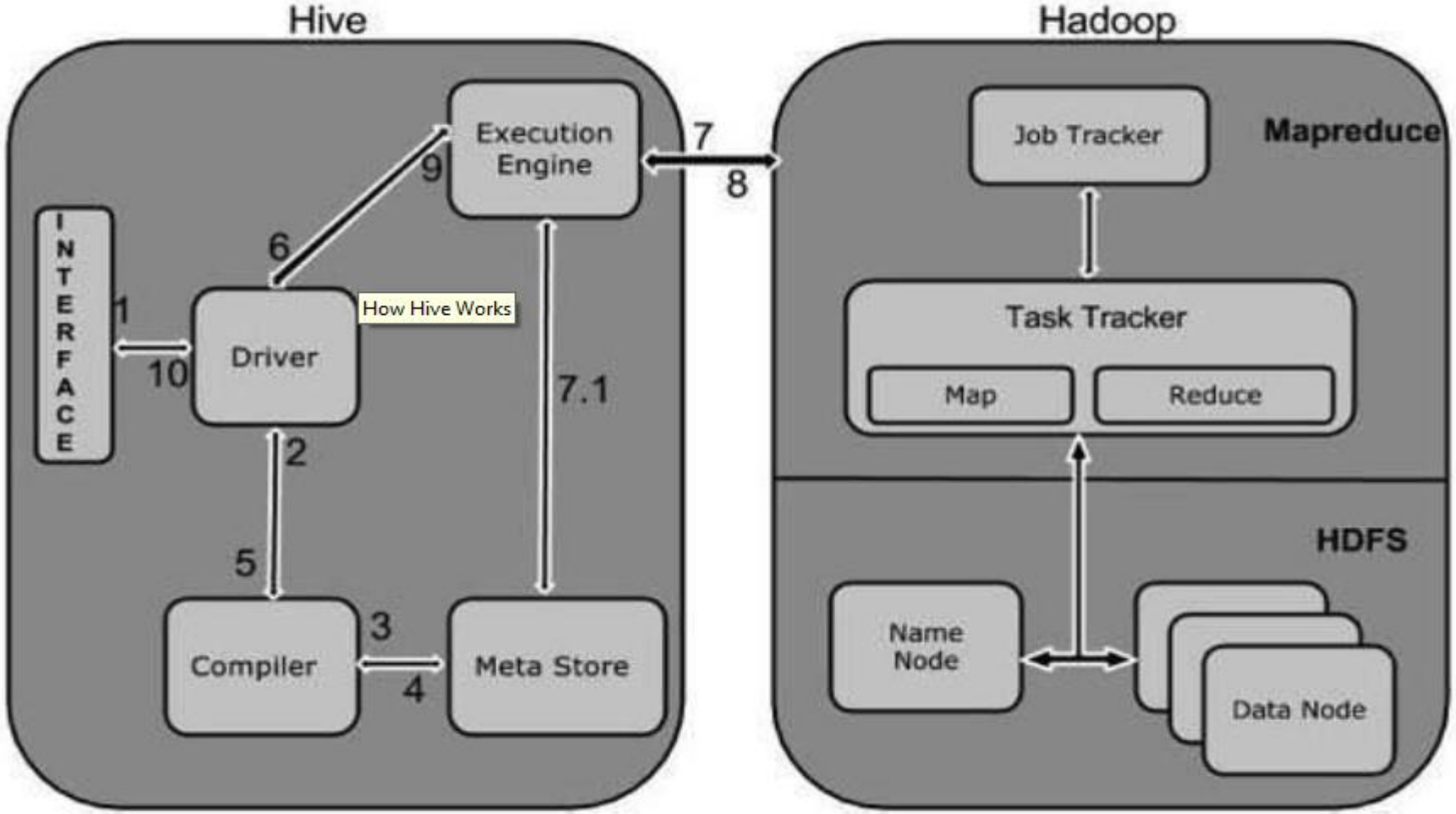
Architecture of Hive

The following component diagram depicts the architecture of Hive:



Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

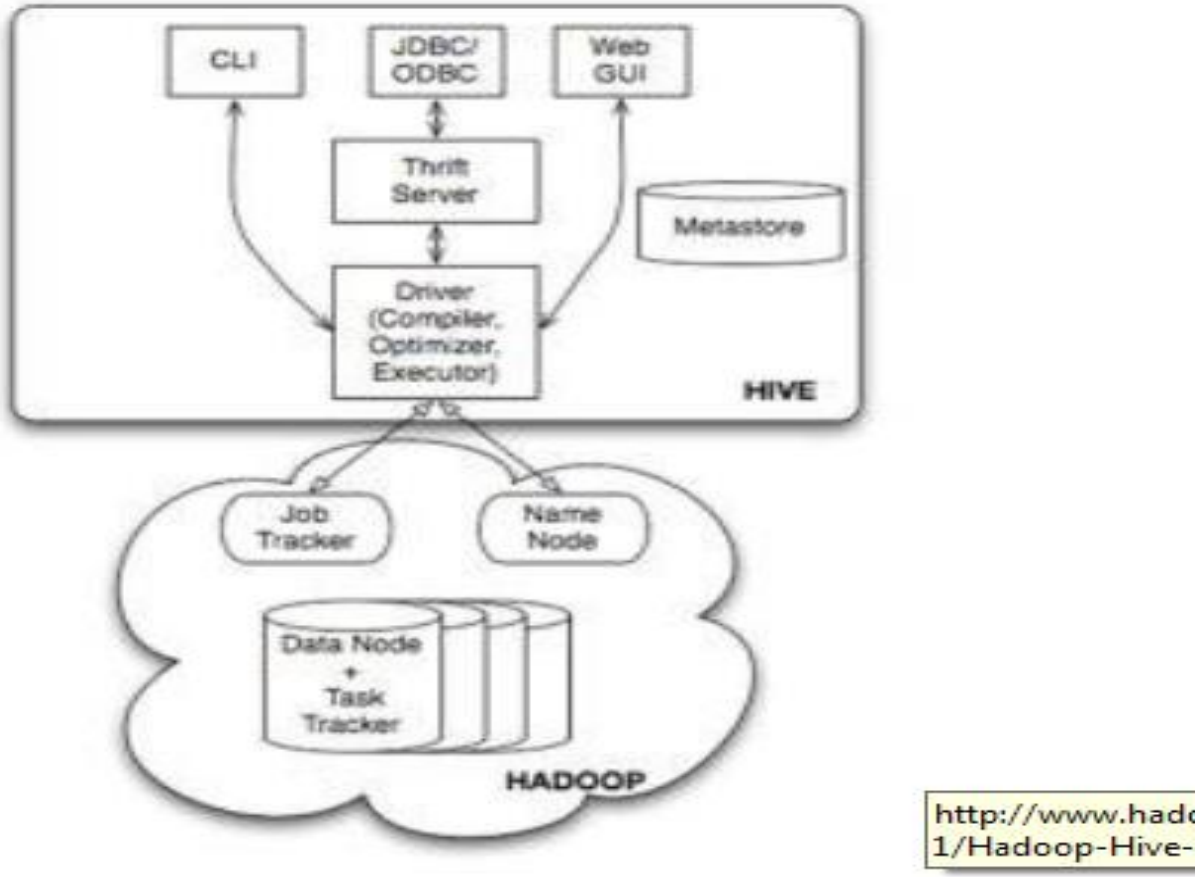
Working of Hive



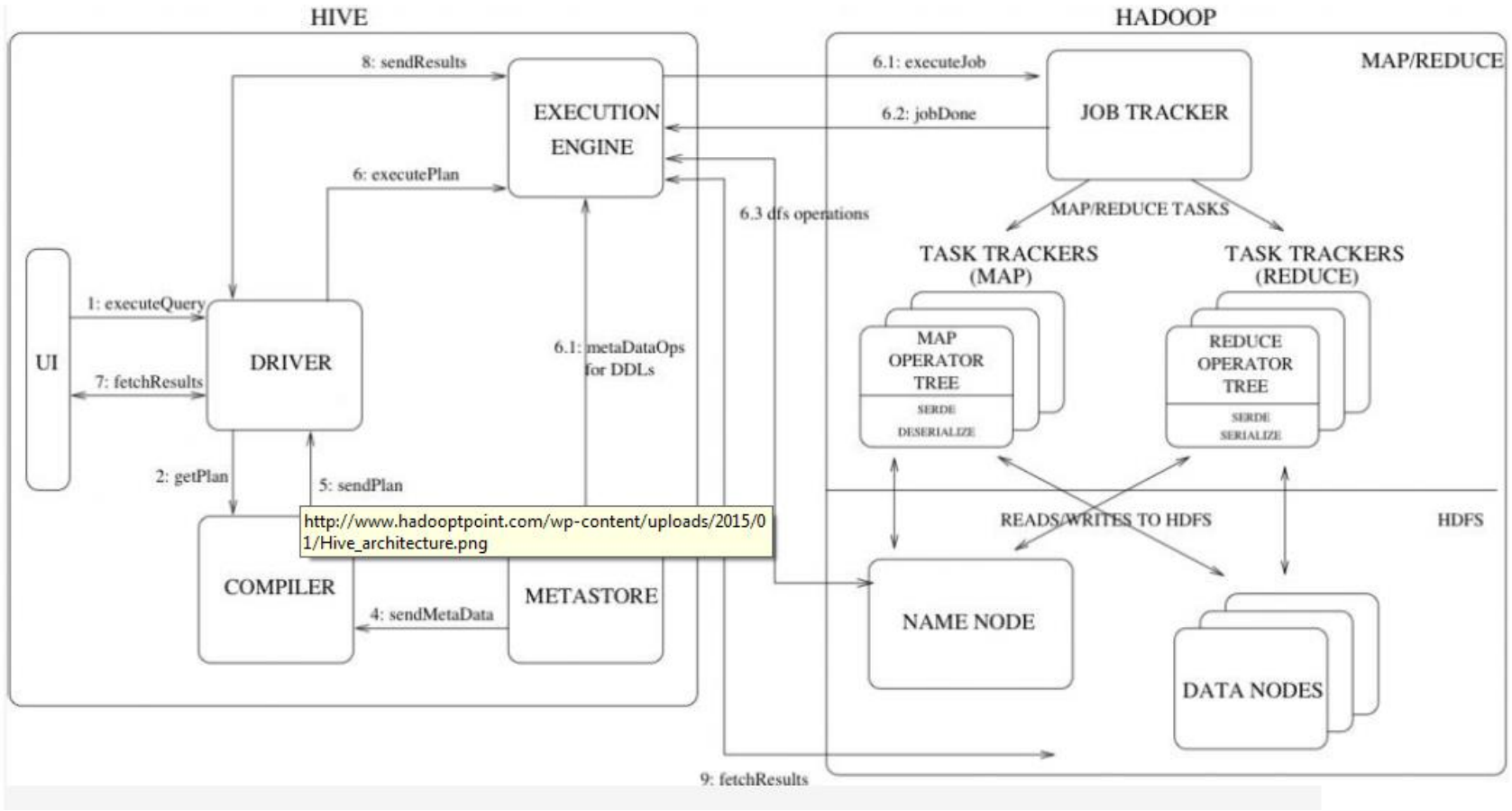
Step No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata**
	The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.
5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	**Execute Plan** The driver sends the execute plan to the execution engine.

7	**Execute Job** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results**
	The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

Hadoop hive architecture



Detailed architecture



Different modes of Hive

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

These modes are,

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

When to use Local mode:

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

When to use Map reduce mode:

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

HIVE VS TRADITIONAL DATABASE

- Hive resembles a traditional database by supporting SQL interface but it is not a full database. Hive can be better called as data warehouse instead of database.

- Hive enforces schema on read time whereas RDBMS enforces schema on write time.

In RDBMS, a table's schema is enforced at data load time, If the data being loaded doesn't conform to the schema, then it is rejected. This design is called schema on write.

But Hive doesn't verify the data when it is loaded, but rather when a it is retrieved. This is called schema on read.

Schema on read makes for a very fast initial load, since the data does not have to be read, parsed, and serialized to disk in the database's internal format. The load operation is just a file copy or move.

Schema on write makes query time performance faster, since the database can index columns and perform compression on the data but it takes longer to load data into the database.

- Hive is based on the notion of Write once, Read many times but RDBMS is designed for Read and Write many times.
- In RDBMS, record level updates, insertions and deletes, transactions and indexes are possible. Whereas these are not allowed in Hive because Hive was built to operate over HDFS data using MapReduce, where full-table scans are the norm and a table update is achieved by transforming the data into a new table.
- In RDBMS, maximum data size allowed will be in 10's of Terabytes but whereas Hive can 100's Petabytes very easily.
- As Hadoop is a batch-oriented system, Hive doesn't support OLTP (Online Transaction Processing) but it is closer to OLAP (Online Analytical Processing) but not ideal since there is significant latency between issuing a query and receiving a reply, due to the overhead of Mapreduce jobs and due to the size of the data sets Hadoop was designed to serve.
- RDBMS is best suited for dynamic data analysis and where fast responses are expected but Hive is suited for data warehouse applications, where relatively static data is analyzed, fast response times are not required, and when the data is not changing rapidly.
- To overcome the limitations of Hive, HBase is being integrated with Hive to support record level operations and OLAP.
- Hive is very easily scalable at low cost but RDBMS is not that much scalable that too it is very costly scale up.

Hive QL Data Types

- Column Types
- Literals
- Null Values
- Complex Types

Column Data Types

- Integral Types
- String Types
- Timestamp
- Date
- Decimal
- Union Types

Integer Types

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types

- String type data types can be specified using single quotes (' ') or double quotes (" ").
- It contains two data types: VARCHAR and CHAR.
- Hive follows C-types escape characters.

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Timestamp

- It supports traditional UNIX timestamp with optional nanosecond precision.
- It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.fffffffff” and format “yyyy-mm-dd hh:mm:ss.fffffffff”.

Date

- DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimal

- The DECIMAL type in Hive is as same as Big Decimal format of Java.
- It is used for representing immutable arbitrary precision.
- The syntax and example is as follows:

```
DECIMAL(precision, scale)
decimal(10,0)
```

Union Types

- Union is a collection of heterogeneous data types.
- We can create an instance using **create union.**
- **The syntax and example is as follows:**

```
UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>


{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}
```

Literals

The following literals are used in Hive:

Floating Point Types

Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type

Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10^{-308} to 10^{308}.

Null Value

Missing values are represented by the special value NULL.

Complex Types

The Hive complex data types are as follows:

Arrays

Arrays in Hive are used the same way they are used in Java.

```
Syntax: ARRAY<data_type>
```

Maps

Maps in Hive are similar to Java Maps.

```
Syntax: MAP<primitive_type, data_type>
```

Structs

Structs in Hive is similar to using complex data with comment.

```
Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ...>
```

Hive -Built-in Operators

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Complex Operators

Relational Operators

```
| Id   | Name         | Salary | Designation                | Dept |
+-----+--------------+--------+----------------------------+------+
|1201 | Gopal        | 45000  | Technical manager          | TP   |
|1202 | Manisha      | 45000  | Proofreader                | PR   |
|1203 | Masthanvali  | 40000  | Technical writer           | TP   |
|1204 | Krian        | 40000  | Hr Admin                   | HR   |
|1205 | Kranthi      | 30000  | Op Admin                   | Admin|
+-----+--------------+--------+----------------------------+------+
```

```
hive> SELECT * FROM employee WHERE Id=1205;
```

```
+-----+-----------+-----------+----------------------------------+
| ID   | Name      | Salary    | Designation                | Dept  |
+-----+---------------+-------+----------------------------------+
|1205 | Kranthi    | 30000     | Op Admin                   | Admin |
+-----+-----------+-----------+----------------------------------+
```

Arithmetic Operators

Example

The following query adds two numbers, 20 and 30.

```
hive> SELECT 20+30 ADD FROM temp;
```

On successful execution of the query, you get to see the following response:

```
+--------+
|  ADD   |
+--------+
|  50    |
+--------+
```

Logical Operators

Example

The following query is used to retrieve employee details whose Department is TP and Salary is more than Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>40000 && Dept=TP;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
+------+--------------+-------------+-------------------+--------+
```

Complex Operators

These operators provide an expression to access the elements of Complex Types.

Operator	Operand	Description
A[n]	A is an Array and n is an int	It returns the nth element in the array A. The first element has index 0.
M[key]	M is a Map<K, V> and key has type K	It returns the value corresponding to the key in the map.
S.x	S is a struct	It returns the x field of S.

JDBC code to create Table

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
public class HiveCreateTable {
private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
public static void main(String[] args) throws SQLException {
// Register driver and create driver instance
Class.forName(driverName);
```

```
// get connection
Connection con =
    DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "", "");
// create statement
Statement stmt = con.createStatement();
// execute statement
stmt.executeQuery("CREATE TABLE IF NOT EXISTS "
+" employee ( eid int, name String, "
+" salary String, destignation String)"
+" COMMENT ‘Employee details’"
+" ROW FORMAT DELIMITED"
+" FIELDS TERMINATED BY ‘\t’"
+" LINES TERMINATED BY ‘\n’"
+" STORED AS TEXTFILE;");
System.out.println(“ Table employee created.”);
con.close();
}
}
```

```
$ javac HiveCreateDb.java
$ java HiveCreateDb
```

Output

```
Table employee created.
```

Load data

Syntax

The syntax for load data is as follows:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

```
1201  Gopal       45000    Technical manager
1202  Manisha     45000    Proof reader
1203  Masthanvali 40000    Technical writer
1204  Kiran       40000    Hr Admin
1205  Kranthi     30000    Op Admin
```

The following query loads the given text into the table.

```
hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'
OVERWRITE INTO TABLE employee;
```

On successful download, you get to see the following response:

```
OK
Time taken: 15.905 seconds
hive>
```

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveLoadData {

   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

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

      // Register driver and create driver instance
      Class.forName(driverName);
```

```
      // get connection
      Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "",
"");


      // create statement
      Statement stmt = con.createStatement();


      // execute statement
      stmt.executeQuery("LOAD DATA LOCAL INPATH '/home/user/sample.txt'" + "OVERWRITE INTO
TABLE employee;");
      System.out.println("Load Data into employee successful");


      con.close();
   }
}
```

```
$ javac HiveLoadData.java
$ java HiveLoadData
```

Output:

```
Load Data into employee successful
```

Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

Rename To... Statement

The following query renames the table from **employee** to **emp**.

```
hive> ALTER TABLE employee RENAME TO emp;
```

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveAlterRenameTo {
   private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

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

```
      // Register driver and create driver instance
      Class.forName(driverName);

      // get connection
      Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "",
"");

      // create statement
      Statement stmt = con.createStatement();

      // execute statement
      stmt.executeQuery("ALTER TABLE employee RENAME TO emp;");
      System.out.println("Table Renamed Successfully");
      con.close();
   }
```

```
$ javac HiveAlterRenameTo.java
$ java HiveAlterRenameTo
```

Output:

```
Table renamed successfully.
```

Hive QL- Select Order by

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]
[LIMIT number];
```

Code Snippet

```
// execute statement
Resultset res = stmt.executeQuery("SELECT * FROM employee WHERE salary>30000;");


System.out.println("Result:");
System.out.println(" ID \t Name \t Salary \t Designation \t Dept ");


while (res.next()) {
   System.out.println(res.getInt(1) + " " + res.getString(2) + " " + res.getDouble(3) +
+ res.getString(4) + " " + res.getString(5));
}
```

```
$ javac HiveQLWhere.java
$ java HiveQLWhere
```

Output:

```
ID      Name            Salary      Designation          Dept
1201    Gopal           45000       Technical manager    TP
1202    Manisha         45000       Proofreader          PR
1203    Masthanvali     40000       Technical writer     TP
1204    Krian           40000       Hr Admin             HR
```

Select Group by

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

```
+------+--------------+------------+-------------------+--------+
| ID   | Name         | Salary     | Designation       | Dept   |
+------+--------------+------------+-------------------+--------+
|1201  | Gopal        | 45000      | Technical manager | TP     |
|1202  | Manisha      | 45000      | Proofreader       | PR     |
|1203  | Masthanvali  | 40000      | Technical writer  | TP     |
|1204  | Krian        | 45000      | Proofreader       | PR     |
|1205  | Kranthi      | 30000      | Op Admin          | Admin  |
+------+--------------+------------+-------------------+--------+
```

The following query retrieves the employee details using the above scenario.

```
hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

On successful execution of the query, you get to see the following response:

```
+------+--------------+
| Dept | Count(*)     |
+------+--------------+
|Admin |    1         |
|PR    |    2         |
|TP    |    3         |
+------+--------------+
```

Code

```
      // create statement
      Statement stmt = con.createStatement();

      // execute statement
      Resultset res = stmt.executeQuery("SELECT Dept,count(*) " + "FROM employee GROUP BY
DEPT; ");
      System.out.println(" Dept \t count(*)");

      while (res.next()) {
         System.out.println(res.getString(1) + " " + res.getInt(2));
      }
```

```
$ javac HiveQLGroupBy.java
$ java HiveQLGroupBy
```

Output:

```
Dept      Count(*)
Admin       1
PR          2
TP          3
```

HiveQL-Select Joins

```
join_table:

   table_reference JOIN table_factor [join_condition]
   | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference
   join_condition
   | table_reference LEFT SEMI JOIN table_reference join_condition
   | table_reference CROSS JOIN table_reference [join_condition]
```

table named CUSTOMERS..

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Consider another table ORDERS as follows:

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

There are different types of joins given as follows:

- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT OUTER JOIN

The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.

A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c
LEFT OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

RIGHT OUTER JOIN

The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

The following query demonstrates RIGHT OUTER JOIN between the CUSTOMER and ORDER tables.

```
notranslate"> hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c
RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

FULL OUTER JOIN

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c
FULL OUTER JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 1    | Ramesh   | NULL   | NULL                |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5    | Hardik   | NULL   | NULL                |
| 6    | Komal    | NULL   | NULL                |
| 7    | Muffy    | NULL   | NULL                |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

JOIN

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

The following query executes JOIN on the CUSTOMER and ORDER tables, and retrieves the records:

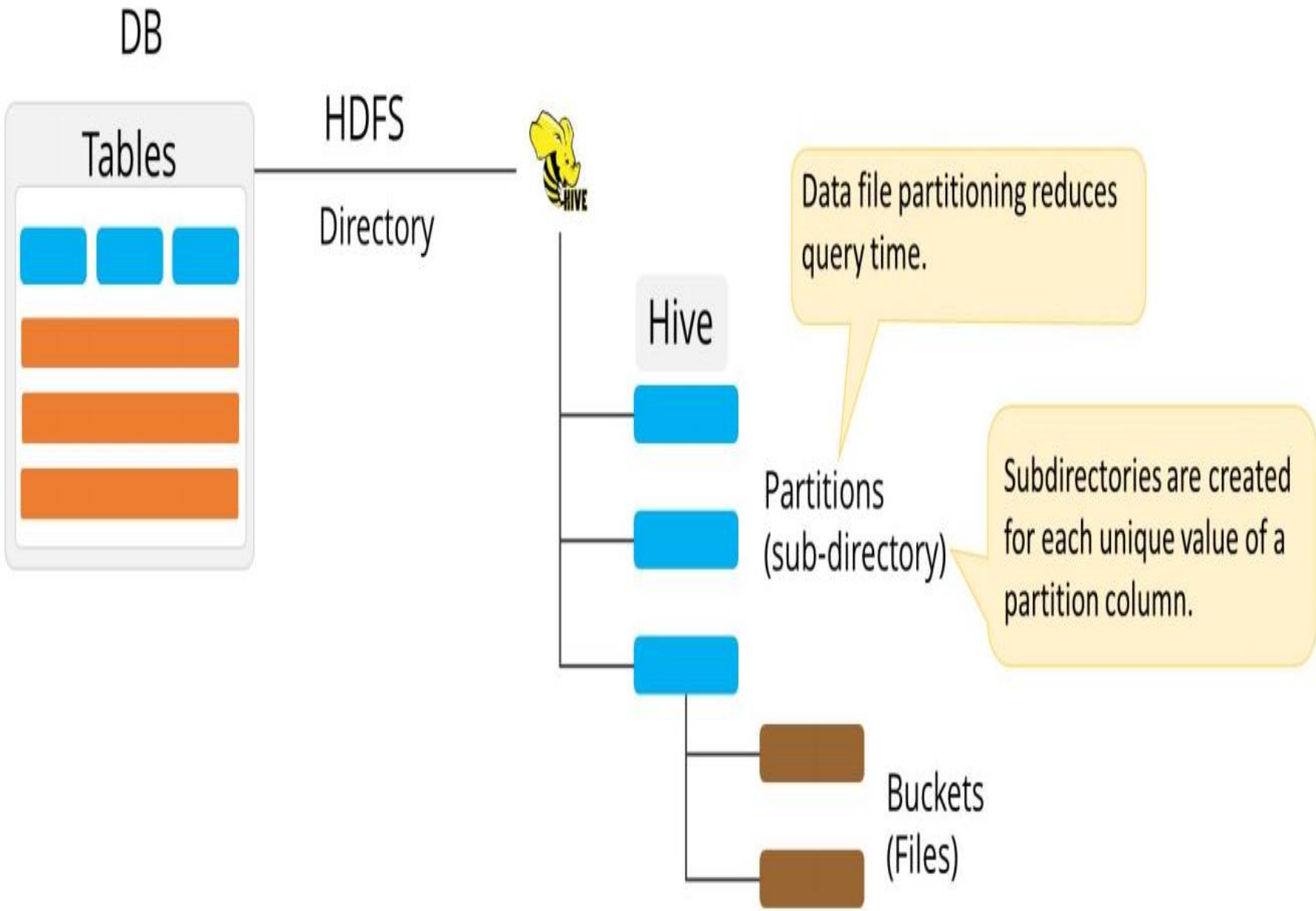
```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

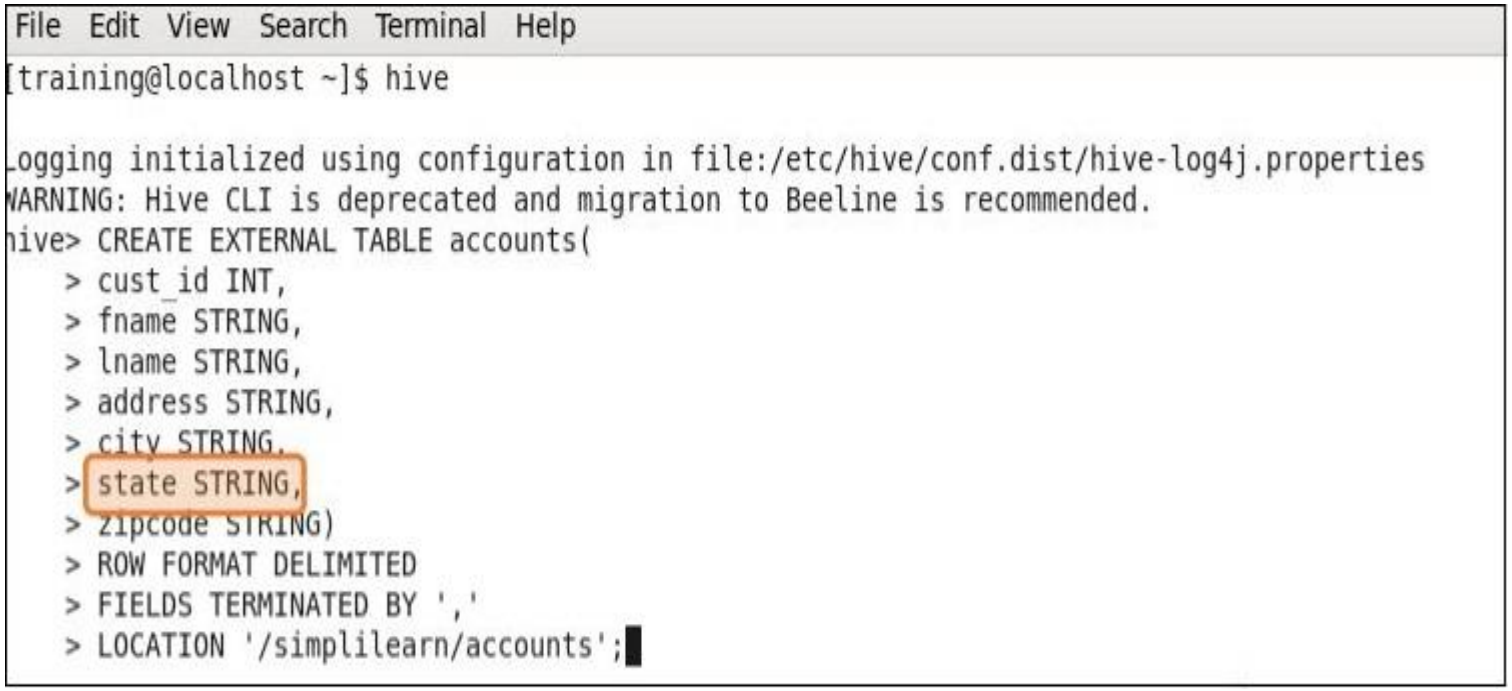
Advanced Hive

- HIVE is considered a tool of choice for performing queries on large datasets, especially those that require full table scans.
- HIVE has advanced partitioning features.
- Data file partitioning in hive is very useful to prune data during the query, in order to reduce query times.
- There are many instances where users need to filter the data on specific column values.
- Using the partitioning feature of HIVE that subdivides the data, HIVE users can identify the columns, which can be used to organize the data.
- Using partitioning, the analysis can be done only on the relevant subset of data, resulting in a highly improved performance of HIVE queries.


DB
Tables
HDFS
Directory
Hive
Data file partitioning reduces query time.
Partitions (sub-directory)
Subdirectories are created for each unique value of a partition column.
Buckets (Files)

Example of a Non-Partitioned Table

- In non-partitioned tables, by default, all queries have to scan all files in the directory.
- This means that HIVE will need to read all the files in a table's data directory.
- This can be a very slow and expensive process, especially when the tables are large.



- There is a State column created in HIVE
- State-wise partition so that separate tables are created for separate states.

Example of a Partitioned Table

```
File  Edit  View  Search  Terminal  Help
[training@localhost ~]$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> CREATE EXTERNAL TABLE accounts_by_state(
    > cust_id INT,
    > fname STRING,
    > lname STRING,
    > address STRING,
    > city STRING,
    > zipcode STRING)
    > PARTITIONED BY (state STRING)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > LOCATION '/simplilearn/accounts_by_state';
```

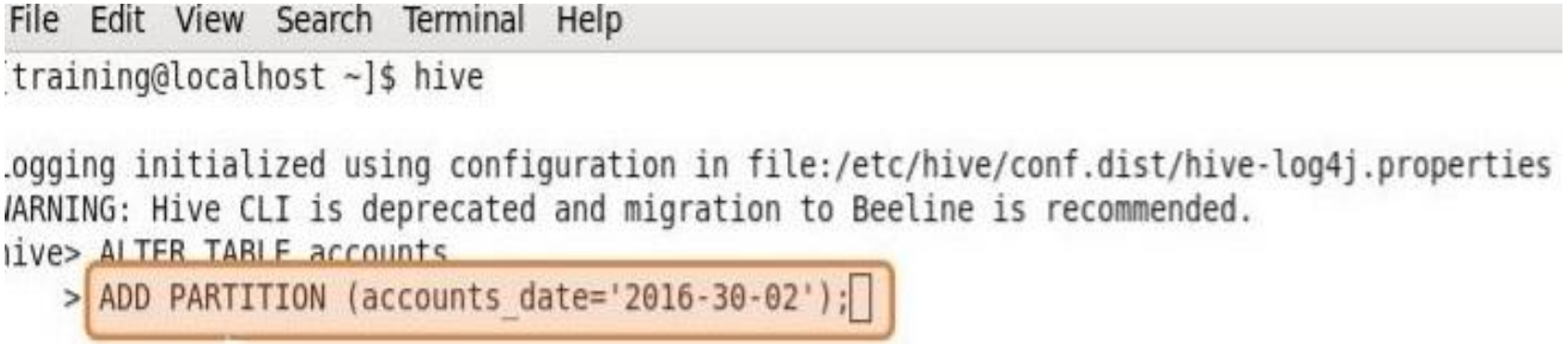
- Partitions are actually horizontal slices of data that allow larger sets of data to be separated into more manageable chunks.
- This essentially means that you can use partitioning in hive to store data in separate files by state
- At the time of table creation, partitions are defined using the PARTITIONED BY clause, with a list of column definitions for partitioning.
- A partition column is a “virtual column, where data is not actually stored in the file.

Dynamic and Static Partitioning in hive

- Data insertion into partitioned tables can be done in two ways or modes: Static partitioning Dynamic partitioning

Static Partitioning in Hive

•In the static partitioning mode, we can insert or input the data files individually into a partition table.
•We can create new partitions as needed, and define the new partitions using the ADD PARTITION clause.
•While loading data, we need to specify which partition to store the data in.
•This means that with each load, we need to specify the partition column value.
•We can add a partition in the table and move the data file into the partition of the table.



Dynamic Partitioning in Hive

- Dynamic partitioning in hive, partitions get created automatically at load times.
- New partitions can be created dynamically from existing data.
- Partitions are automatically created based on the value of the last column. If the partition does not already exist, it will be created.
- In case the partition does exist, it will be overwritten by the OVERWRITE keyword as shown in the below example.

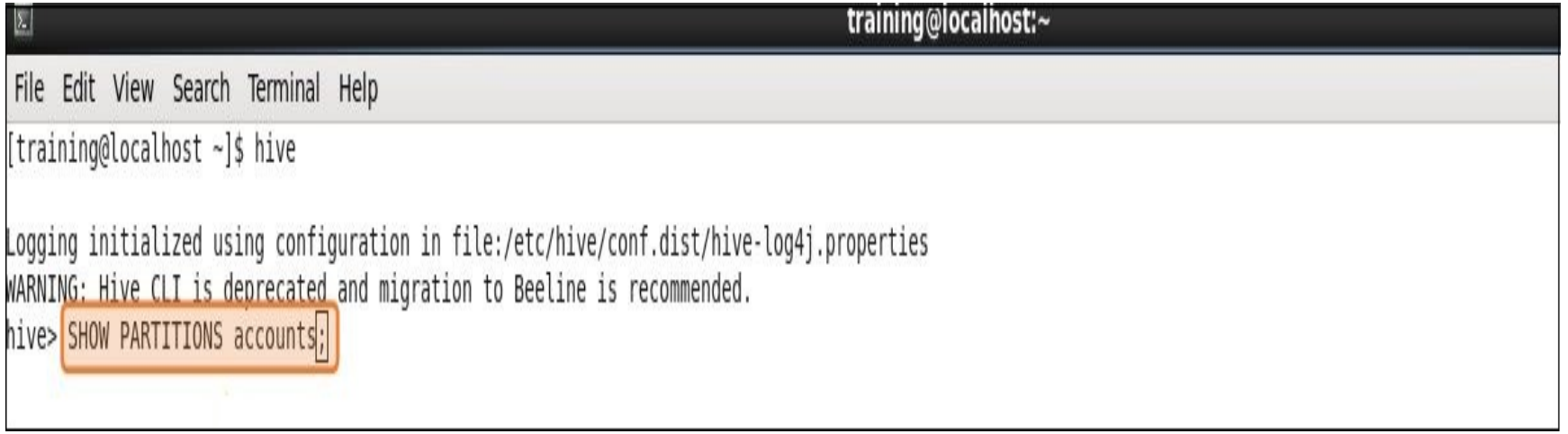
- When you have a large amount of data stored in a table, then the dynamic partition is suitable.
- By default, dynamic partitioning is disabled in HIVE to prevent accidental partition creation.
- Enable the following settings to use dynamic partitioning:

```
SET hive.exec.dynamic.partition=true;

SET hive.exec.dynamic.partition.mode=nonstrict;.
```

Viewing and Deleting Partitions

- View the partitions of a partitioned table using the SHOW command



To delete drop the partitions, use the ALTER command

By using the ALTER command, you can also add or change partitions.

When to use partitioning?

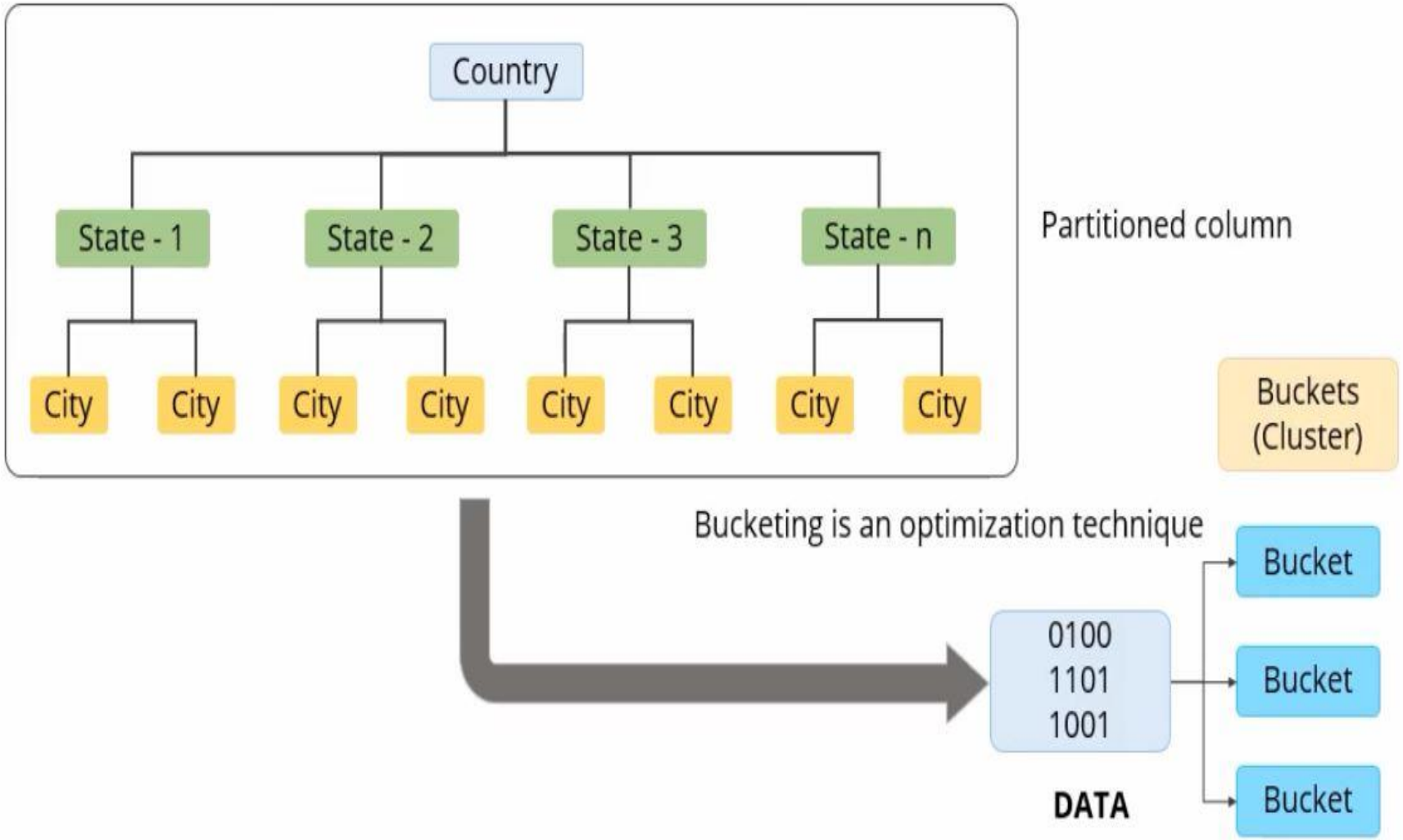
- Reading the entire data set takes too long.
- Queries almost always filter on the partition columns.
- There are a reasonable number of different values for partition columns.

when you should avoid using a partitioning:

- Avoid partition on columns that have too many unique rows.
- Be cautious while creating a dynamic partition as it can lead to a high number of partitions.
- Try to limit partition to less than 20k.

Bucketing in Hive

- There may be instances where partitioning the tables results in a large number of partitions.
- This is where the concept of bucketing comes in.
- Bucketing is an optimization technique similar to partitioning.
- We can use bucketing if you need to run queries on columns that have huge data, which makes it difficult to create partitions.


Country
State - 1
State - 2
State - 3
State - n
City
City
City
City
City
City
City
City
Partitioned column
Buckets (Cluster)
Bucketing is an optimization technique
0100
1101
1001
DATA
Bucket
Bucket
Bucket

What Do Buckets Do?

- They distribute the data load into a user-defined set of clusters by calculating the hash code of the key mentioned in the query.

```
CREATE TABLE page_views( user_id INT, session_id BIGINT, url

STRING)

PARTITIONED BY (day INT)

CLUSTERED BY (user_id) INTO 100;
```

•As per the syntax, the data would be classified depending on the hash number of user underscore id into 100 buckets.

•The processor will first calculate the hash number of the user underscore id in the query and will look for only that bucket.

Hive Query Language - Introduction

- It’s the SQL-like query language for HIVE to process and analyze structured data in a Metastore.

```
SELECT
dt,
COUNT (DISTINCT (user_id))
FROM events
GROUP BY dt;
```

- An important principle of HIVEQL is extensibility.

HIVEQL can be extended in multiple ways:

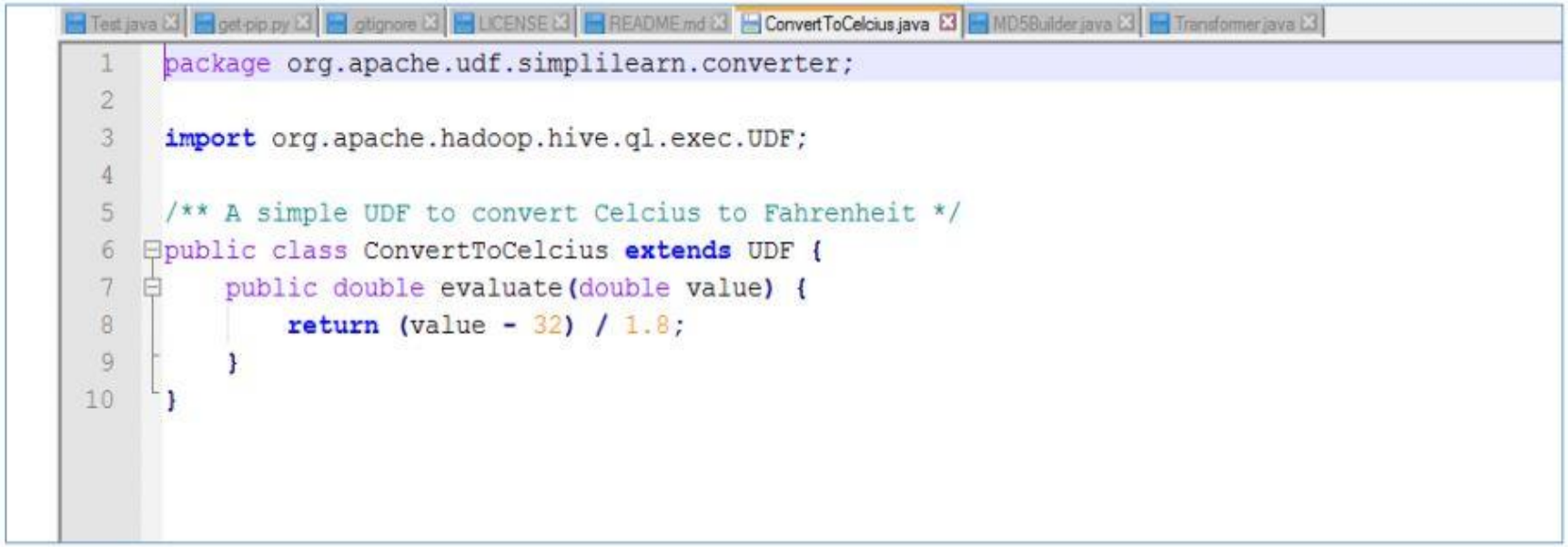
- Pluggable user-defined functions
- Pluggable MapReduce scripts
- Pluggable user-defined types
- Pluggable data formats

User-defined function(UDF)

- HIVE has the ability to define a function.
- UDFs provide a way of extending the functionality of HIVE with a function, written in Java that can be evaluated in HIVEQL statements.
- All UFDs extend the HIVE UDF class.
- A UDF subclass needs to implement one or more methods named evaluate, which will be called by HIVE.
- Evaluate should never be a void method. However, it can return null, if required.

Hive UDF Example

- To convert any value to Celsius:



Code for Extending UDF

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;

import org.apache.hadoop.io.Text;

public final class Lower extends UDF {

public Text evaluate(final Text s) {

if (s == null) { return null; }

return new Text(s.toString().toLowerCase());

}

}
```

User-Defined Function-codes

- After compiling the UDF, you must include it in the HIVE classpath.
- Here is a code that you can use to register the class.

```
CREATE FUNCTION my_lower AS 'com.example.hive.udf.Lower';
```

- Once HIVE gets started, you can use the newly defined function in a query statement after registering them.
- This is a code to use the function in a HIVE query statement.

```
SELECT my_lower(title), sum(freq) FROM titles GROUP BY my_lower(title);
```

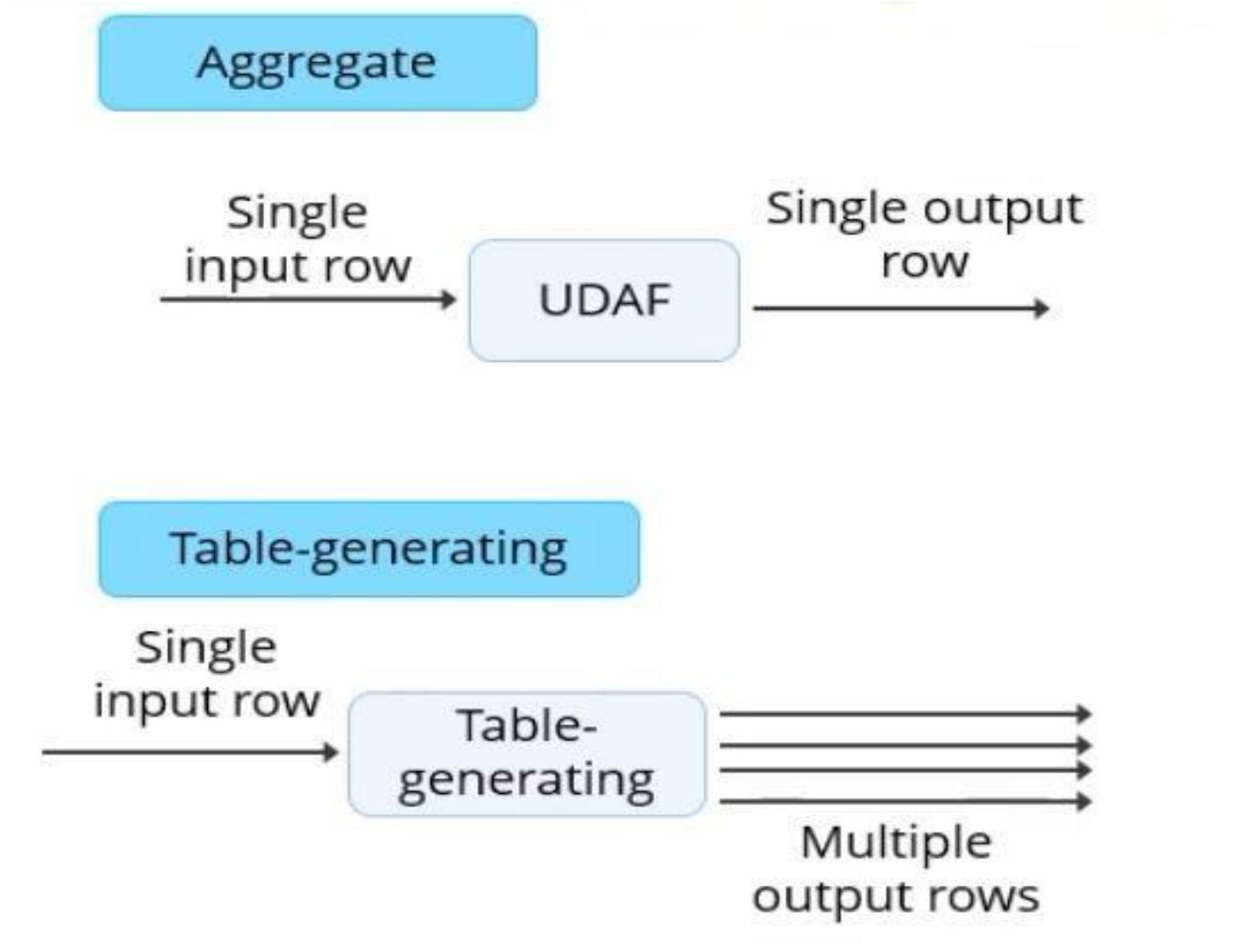
- Writing the functions in JavaScript creates its own UDF.

Built-in functions of Hive

- Mathematical: For mathematical operations, you can use the examples of the round, floor, and so on.
- Collection: For collections, you can use size, map keys, and so on.
- Type conversion: For data type conversions, you can use a cast.
- Date:For dates, use the following APIs like a year, datediff, and so on.
- Conditional: For conditional functions, use if, case, and coalesce.
- String: For string files, use length, reverse, and so on.

Aggregate functions

- Aggregate functions create the output if the full set of data is given.
- The implementation of these functions is complex compared with that of the UDF.
- The user should implement a few more methods, however, the format is similar to UDF.
- Therefore, HIVE provides many built-in User-Defined Aggregate Functions or UDAF.


Aggregate
Single input row
UDAF
Single output row
Table-generating
Single input row
Table-generating
Multiple output rows

HBase –Introduction

HBase

- Since 1970, RDBMS is the solution for data storage and maintenance related problems.
- After the advent of big data, companies realized the benefit of processing big data and started opting for solutions like Hadoop.
- Hadoop uses distributed file system for storing big data, and MapReduce to process it.
- Hadoop excels in storing and processing of huge data of various formats such as arbitrary, semi-, or even unstructured.

Limitations of Hadoop

- Hadoop can perform only batch processing, and data will be accessed only in a sequential manner.
- That means one has to search the entire dataset even for the simplest of jobs.
- A huge dataset when processed results in another huge data set, which should also be processed sequentially.
- At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

- HBase,
- Cassandra,
- couchDB,
- Dynamo, and
- MongoDB

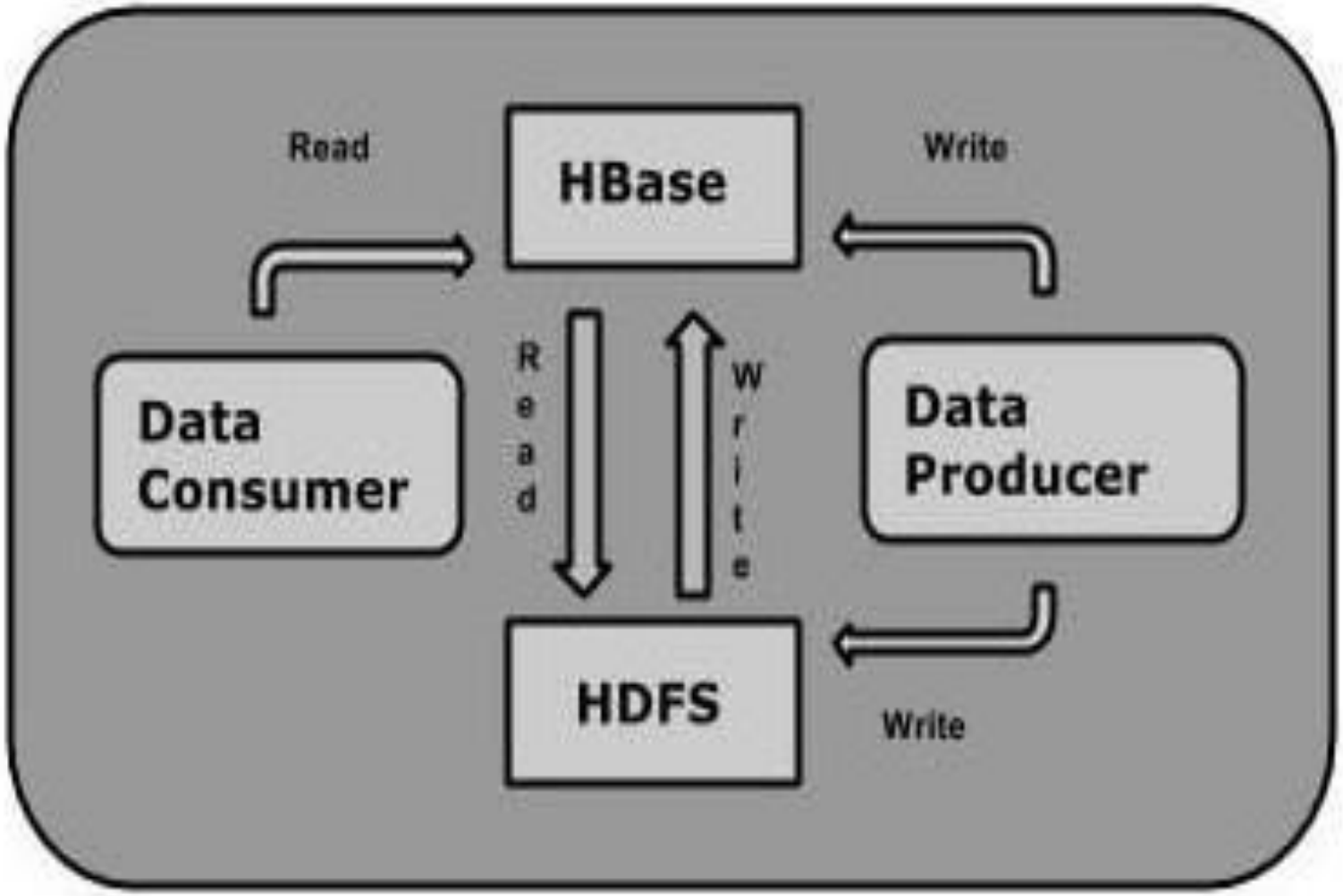
These are some of the databases that store huge amounts of data and access the data in a random manner.

What is HBase?

- HBase is a distributed column-oriented database
- Built on top of the Hadoop file system.
- It is an open-source project and is horizontally scalable.
- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data.
- It leverages the fault tolerance provided by the Hadoop File System (HDFS).

What is Hbase?

- It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.
- One can store the data in HDFS either directly or through HBase.
- Data consumer reads/accesses the data in HDFS randomly using HBase.
- HBase sits on top of the Hadoop File System and provides read and write access


Read
HBase
Write
Data Consumer
Read
Write
Data Producer
HDFS
Write

HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing;	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

- HBase is a column-oriented database and the tables in it are sorted by row.
- The table schema defines only column families, which are the key value pairs.
- A table have multiple column families and each column family can have any number of columns.
- Subsequent column values are stored contiguously on the disk.
- Each cell value of the table has a timestamp

Hbase Description

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Sample Schema of a table in HBase

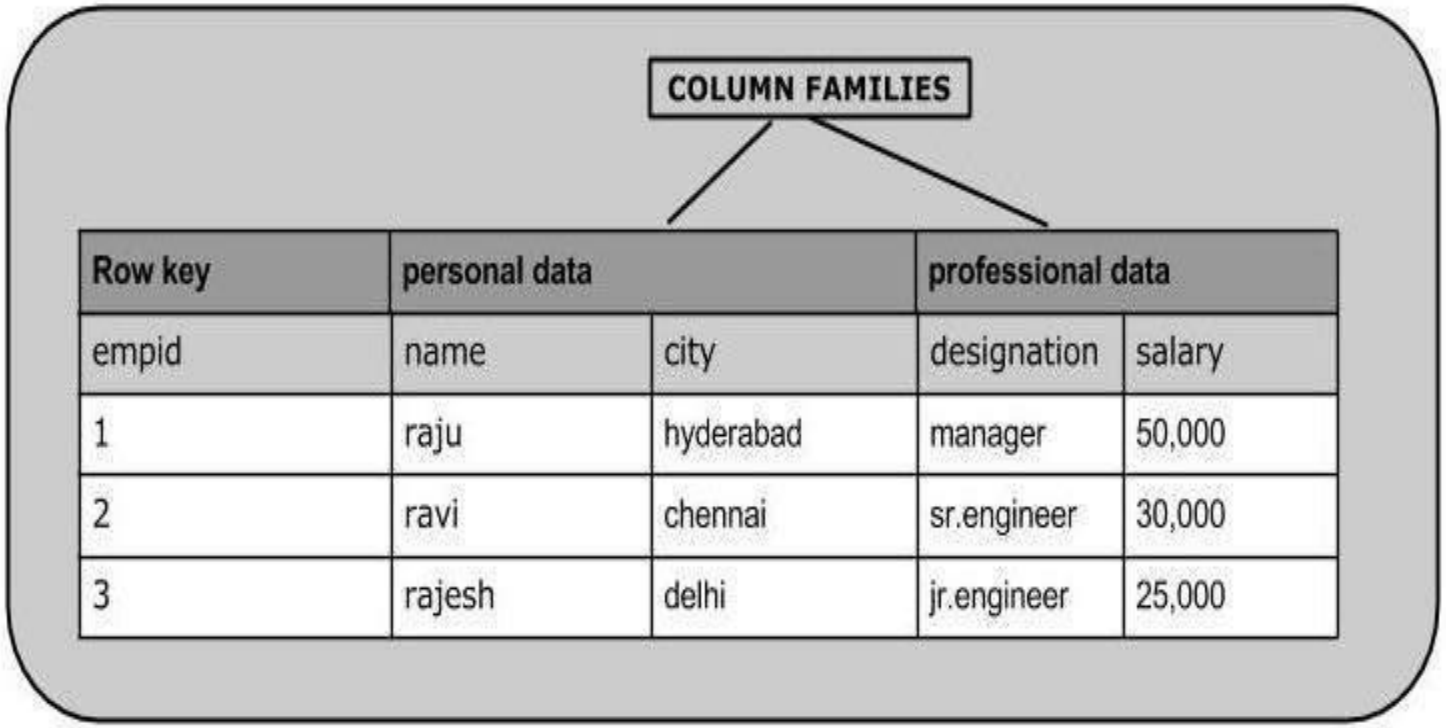
Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Column Oriented and Row Oriented

- Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data.
- Shortly, they will have column families

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

column families in a column-oriented database:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

HBase and RDBMS

HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Where to Use HBase

- Apache HBase is used to have random, real-time read/write access to Big Data.
- It hosts very large tables on top of clusters of commodity hardware.
- Apache HBase is a non-relational database modeled after Google's Bigtable.
- Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

Applications of HBase

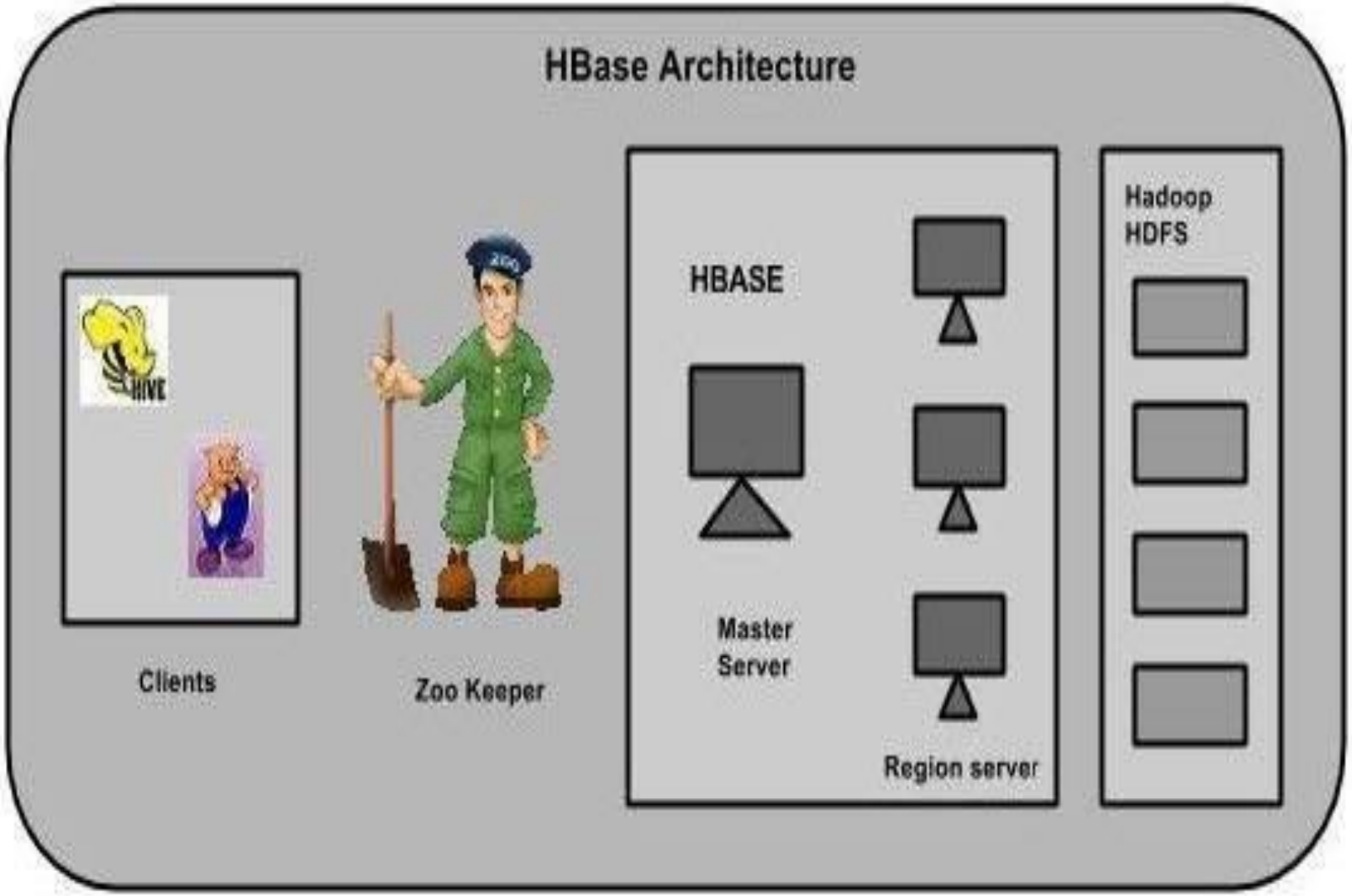
- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase History

Year	Event
Nov 2006	Google released the paper on BigTable.
Feb 2007	Initial HBase prototype was created as a Hadoop contribution.
Oct 2007	The first usable HBase along with Hadoop 0.15.0 was released.
Jan 2008	HBase became the sub project of Hadoop.
Oct 2008	HBase 0.18.1 was released.
Jan 2009	HBase 0.19.0 was released.
Sept 2009	HBase 0.20.0 was released.
May 2010	HBase became Apache top-level project.

- In HBase, tables are split into regions and are served by the region servers.
- Regions are vertically divided by column families into "Stores".
- Stores are saved as files in HDFS.
- The term 'store' is used for regions to explain the storage structure.

- HBase has three major components: the client library, a master server, and region servers.
- Region servers can be added or removed as per requirement.


HBase Architecture
HIVE
Clients
Zoo Keeper
HBASE
Master
Server
Region server
Hadoop
HDFS

MasterServer

- Assigns regions to the region servers
- Takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers.
- It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

- Regions are nothing but tables that are split up

Region server

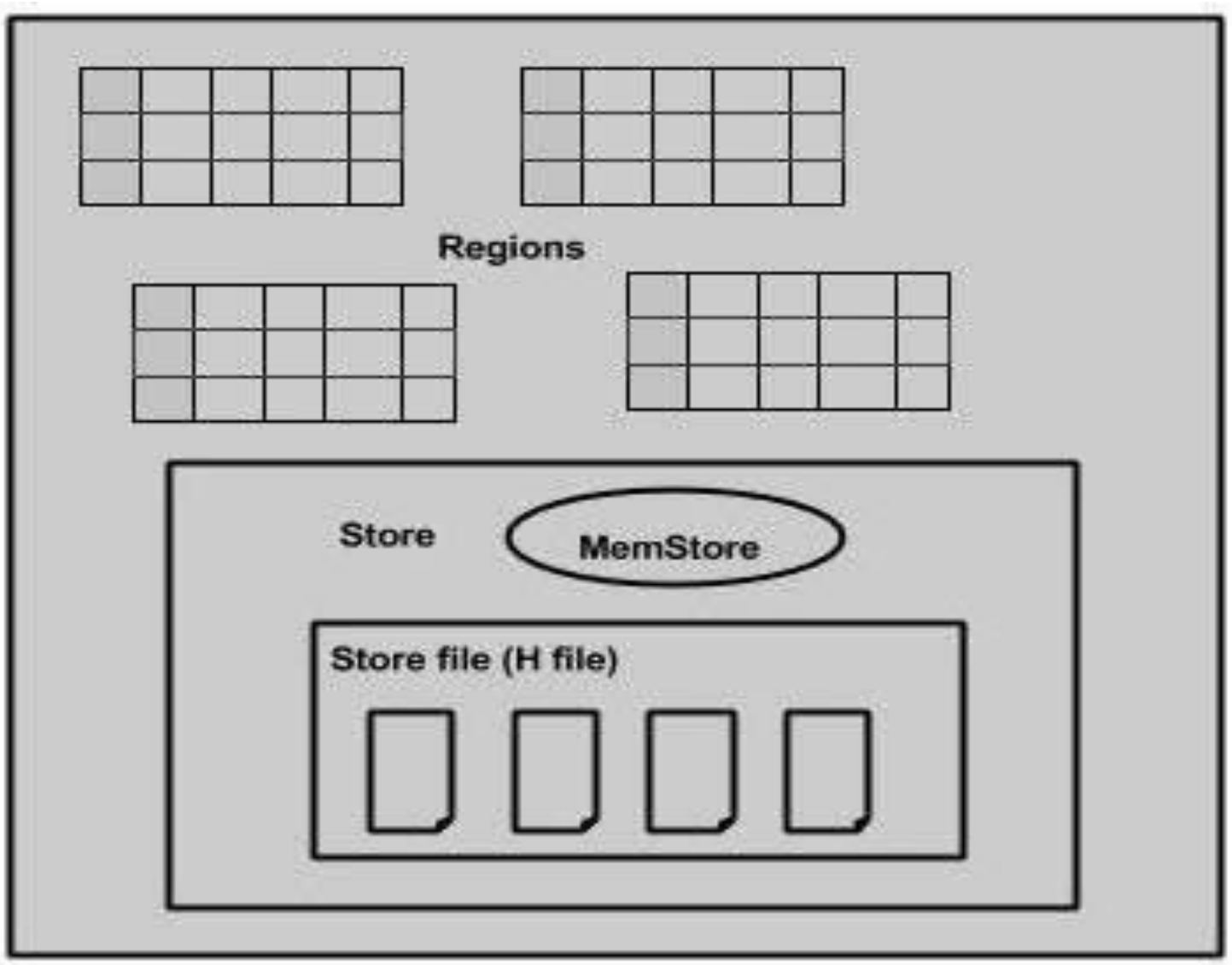
- The region servers have regions that -

➢ Communicate with the client and handle data-related operations.

➢ Handle read and write requests for all the regions under it.

➢ Decide the size of the region by following the region size thresholds.

➢ Spread across the region servers.


Regions
Store
MemStore
Store file (H file)

Storing the Data

- The store contains memory store and HFiles.
- Memstore is just like a cache memory.
- Anything that is entered into the HBase is stored in Memstore initially.
- Later, the data is transferred and saved in Hfiles as blocks and the
- Finally the memstore is flushed.

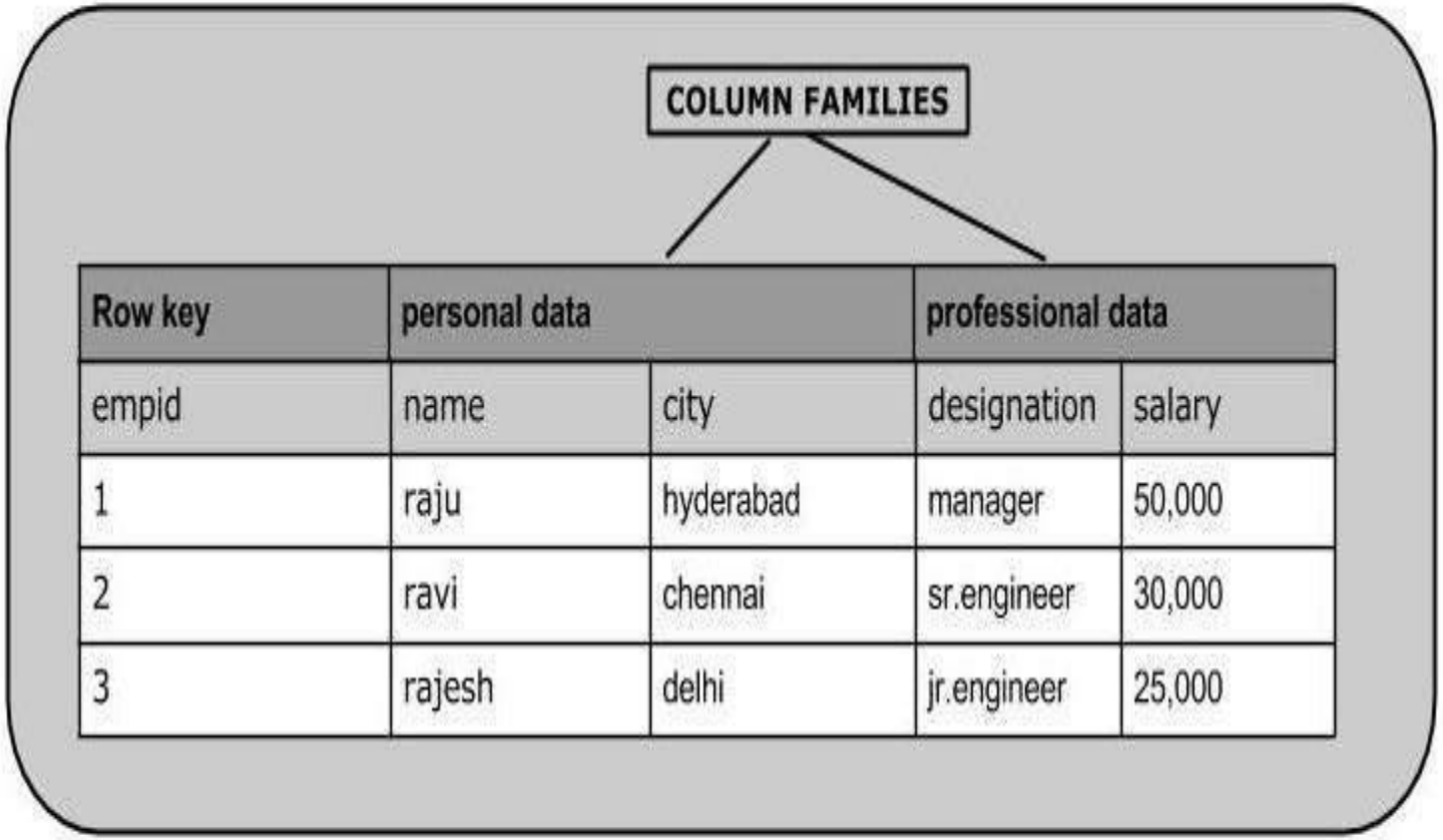
Zookeeper

- Zookeeper is an open-source project that provides services like :
  - maintaining configuration information,
  - naming,
  - providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes(Temporary) representing different region servers.
- Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

Loading Data in HBase & Querying Data in Hbase

- Inserting Data using HBase Shell
- To create data in an HBase table, the following commands and methods are used:
- **put** command,
- **add()** method of **Put** class, and
- **put()** method of **HTable** class.

```
put '<table name>','row1','<colfamily:colname>','<value>'
```



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

```
hbase(main):005:0> put 'emp','1','personal data:name','raju'
0 row(s) in 0.6600 seconds
hbase(main):006:0> put 'emp','1','personal data:city','hyderabad'
0 row(s) in 0.0410 seconds
hbase(main):007:0> put 'emp','1','professional
data:designation','manager'
0 row(s) in 0.0240 seconds
hbase(main):007:0> put 'emp','1','professional data:salary','50000'
0 row(s) in 0.0240 seconds
```

```
hbase(main):022:0> scan 'emp'

   ROW                              COLUMN+CELL
1 column=personal data:city, timestamp=1417524216501, value=hyderabad

1 column=personal data:name, timestamp=1417524185058, value=ramu

1 column=professional data:designation, timestamp=1417524232601,

 value=manager

1 column=professional data:salary, timestamp=1417524244109, value=50000

2 column=personal data:city, timestamp=1417524574905, value=chennai

2 column=personal data:name, timestamp=1417524556125, value=ravi

2 column=professional data:designation, timestamp=1417524592204,

 value=sr:engg

2 column=professional data:salary, timestamp=1417524604221, value=30000

3 column=personal data:city, timestamp=1417524681780, value=delhi

3 column=personal data:name, timestamp=1417524672067, value=rajesh

3 column=professional data:designation, timestamp=1417524693187,

value=jr:engg
3 column=professional data:salary, timestamp=1417524702514,

value=25000
```

Inserting Data Using Java API

You can insert data into Hbase using the **add()** method of the **Put** class. You can save it using the **put()** method of the **HTable** class. These classes belong to the **org.apache.hadoop.hbase.client** package. Below given are the steps to create data in a Table of HBase.

Step 1:Instantiate the Configuration Class

The **Configuration** class adds HBase configuration files to its object. You can create a configuration object using the **create()** method of the**HbaseConfiguration** class as shown below.

```
Configuration conf = HbaseConfiguration.create();
```

Step 2:Instantiate the HTable Class

You have a class called **HTable**, an implementation of Table in HBase. This class is used to communicate with a single HBase table. While instantiating this class, it accepts configuration object and table name as parameters. You can instantiate HTable class as shown below.

```
HTable hTable = new HTable(conf, tableName);
```

Step 3: Instantiate the PutClass

To insert data into an HBase table, the **add()** method and its variants are used. This method belongs to **Put**, therefore instantiate the put class. This class requires the row name you want to insert the data into, in string format. You can instantiate the **Put** class as shown below.

```
Put p = new Put(Bytes.toBytes("row1"));
```

Step 4: Insert Data

The **add()** method of **Put** class is used to insert data. It requires 3 byte arrays representing column family, column qualifier (column name), and

the value to be inserted, respectively. Insert data into the HBase table using the add() method as shown below.

```
p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes("column
name"),Bytes.toBytes("value"));
```

Step 5: Save the Data in Table

After inserting the required rows, save the changes by adding the put instance to the **put()** method of HTable class as shown below.

```
hTable.put(p);
```

Step 6: Close the HTable Instance

After creating data in the HBase Table, close the **HTable** instance using the**close()** method as shown below.

```
hTable.close();
```

Given below is the complete program to create data in HBase Table.

Create Data in Hbase Table

```
import java.io.IOException;


import org.apache.hadoop.conf.Configuration;


import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;


public class InsertData{


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

```
// Instantiating Configuration class
Configuration config = HBaseConfiguration.create();

// Instantiating HTable class
HTable hTable = new HTable(config, "emp");

// Instantiating Put class
// accepts a row name.
Put p = new Put(Bytes.toBytes("row1"));

// adding values using add() method
// accepts column family name, qualifier/row name ,value
p.add(Bytes.toBytes("personal"),
Bytes.toBytes("name"),Bytes.toBytes("raju"));

p.add(Bytes.toBytes("personal"),
Bytes.toBytes("city"),Bytes.toBytes("hyderabad"));

p.add(Bytes.toBytes("professional"),Bytes.toBytes("designation"),
Bytes.toBytes("manager"));

p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),
Bytes.toBytes("50000"));

// Saving the put Instance to the HTable.
hTable.put(p);
System.out.println("data inserted");

// closing HTable
hTable.close();
```

```
   }
}
```

Compile and execute the above program as shown below.

```
$javac InsertData.java
$java InsertData
```

The following should be the output:

```
data inserted
```

HBase -Update Data

Example

Suppose there is a table in HBase called **emp** with the following data.

```
hbase(main):003:0> scan 'emp'
 ROW                COLUMN + CELL
row1 column = personal:name, timestamp = 1418051555, value = raju
row1 column = personal:city, timestamp = 1418275907, value = Hyderabad
row1 column = professional:designation, timestamp = 14180555,value = manager
row1 column = professional:salary, timestamp = 1418035791555,value = 50000
1 row(s) in 0.0100 seconds
```

The following command will update the city value of the employee named 'Raju' to Delhi.

```
hbase(main):002:0> put 'emp','row1','personal:city','Delhi'
0 row(s) in 0.0400 seconds
```

The updated table looks as follows where you can observe the city of Raju has been changed to 'Delhi'.

```
hbase(main):003:0> scan 'emp'
  ROW          COLUMN + CELL
row1 column = personal:name, timestamp = 1418035791555, value = raju
row1 column = personal:city, timestamp = 1418274645907, value = Delhi
row1 column = professional:designation, timestamp = 141857555,value = manager
row1 column = professional:salary, timestamp = 1418039555, value = 50000
1 row(s) in 0.0100 seconds
```

Updating Data Using Java API

You can update the data in a particular cell using the **put()** method. Follow the steps given below to update an existing cell value of a table.

Step 1: Instantiate the Configuration Class

Configuration class adds HBase configuration files to its object. You can create a configuration object using the **create()** method of the**HbaseConfiguration** class as shown below.

```
Configuration conf = HbaseConfiguration.create();
```

Step 2: Instantiate the HTable Class

You have a class called **HTable,** an implementation of Table in HBase. This class is used to communicate with a single HBase table. While instantiating this class, it accepts the configuration object and the table name as parameters. You can instantiate the HTable class as shown below.

```
HTable hTable = new HTable(conf, tableName);
```

Step 3: Instantiate the Put Class

To insert data into HBase Table, the **add()** method and its variants are used. This method belongs to **Put**, therefore instantiate the **put** class. This class requires the row name you want to insert the data into, in string format. You can instantiate the **Put** class as shown below.

```
Put p = new Put(Bytes.toBytes("row1"));
```

Step 4: Update an Existing Cell

The **add()** method of **Put** class is used to insert data. It requires 3 byte arrays representing column family, column qualifier (column name), and the value to be inserted, respectively. Insert data into HBase table using the**add()** method as shown below.

```
p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes("column
name"),Bytes.toBytes("value"));
p.add(Bytes.toBytes("personal"),
Bytes.toBytes("city"),Bytes.toBytes("Delih"));
```

Step 5: Save the Data in Table

After inserting the required rows, save the changes by adding the put instance to the **put()** method of the HTable class as shown below.

```
hTable.put(p);
```

Step 6: Close HTable Instance

After creating data in HBase Table, close the **HTable** instance using the close() method as shown below.

```
hTable.close();
```

Given below is the complete program to update data in a particular table.

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;

public class UpdateData{

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

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();

      // Instantiating HTable class
      HTable hTable = new HTable(config, "emp");

      // Instantiating Put class
      //accepts a row name
      Put p = new Put(Bytes.toBytes("row1"));

      // Updating a cell value
      p.add(Bytes.toBytes("personal"),
      Bytes.toBytes("city"),Bytes.toBytes("Delih"));

      // Saving the put Instance to the HTable.
      hTable.put(p);
      System.out.println("data Updated");
```

```
      // closing HTable
      hTable.close();
   }
}
```

Compile and execute the above program as shown below.

```
$javac UpdateData.java
$java UpdateData
```

The following should be the output:

```
data Updated
```

HBase - Read Data

Reading Data using HBase Shell

The **get** command and the **get()** method of **HTable** class are used to read data from a table in HBase. Using **get** command, you can get a single row of data at a time. Its syntax is as follows:

```
get '<table name>','row1'
```

Example

The following example shows how to use the get command. Let us scan the first row of the **emp** table.

```
hbase(main):012:0> get 'emp', '1'

   COLUMN                        CELL

personal : city timestamp = 1417521848375, value = hyderabad
```

```
personal : name timestamp = 1417521785385, value = ramu

professional: designation timestamp = 1417521885277, value = manager

professional: salary timestamp = 1417521903862, value = 50000

4 row(s) in 0.0270 seconds
```

Reading a Specific Column

Given below is the syntax to read a specific column using the **get** method.

```
hbase> get 'table name', 'rowid', {COLUMN ⇒ 'column family:column name '}
```

Example

Given below is the example to read a specific column in HBase table.

```
hbase(main):015:0> get 'emp', 'row1', {COLUMN ⇒ 'personal:name'}
  COLUMN                CELL
personal:name timestamp = 1418035791555, value = raju
1 row(s) in 0.0080 seconds
```

Reading Data Using Java API

To read data from an HBase table, use the **get()** method of the HTable class. This method requires an instance of the **Get** class. Follow the steps given below to retrieve data from the HBase table.

Step 1: Instantiate the Configuration Class

Configuration class adds HBase configuration files to its object. You can create a configuration object using the **create()** method of the**HbaseConfiguration** class as shown below.

```
Configuration conf = HbaseConfiguration.create();
```

Step 2: Instantiate the HTable Class

You have a class called **HTable**, an implementation of Table in HBase. This class is used to communicate with a single HBase table. While instantiating this class, it accepts the configuration object and the table name as parameters. You can instantiate the HTable class as shown below.

```
HTable hTable = new HTable(conf, tableName);
```

Step 3: Instantiate the Get Class

You can retrieve data from the HBase table using the **get()** method of the**HTable** class. This method extracts a cell from a given row. It requires a **Get**class object as parameter. Create it as shown below.

```
Get get = new Get(toBytes("row1"));
```

Step 4: Read the Data

While retrieving data, you can get a single row by id, or get a set of rows by a set of row ids, or scan an entire table or a subset of rows.

You can retrieve an HBase table data using the add method variants in **Get**class.

To get a specific column from a specific column family, use the following method.

```
get.addFamily(personal)
```

To get all the columns from a specific column family, use the following method.

```
get.addColumn(personal, name)
```

Step 5: Get the Result

Get the result by passing your **Get** class instance to the get method of the**HTable** class. This method returns the **Result** class object, which holds the requested result. Given below is the usage of **get()** method.

```
Result result = table.get(g);
```

Step 6: Reading Values from the Result Instance

The **Result** class provides the **getValue()** method to read the values from its instance. Use it as shown below to read the values from the **Result** instance.

```
byte [] value = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("name"));
byte [] value1 = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("city"));
```

Read values from HBase

```
import java.io.IOException;


import org.apache.hadoop.conf.Configuration;


import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;


public class RetriveData{


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


      // Instantiating Configuration class
```

```
      Configuration config = HBaseConfiguration.create();

      // Instantiating HTable class
      HTable table = new HTable(config, "emp");

      // Instantiating Get class
      Get g = new Get(Bytes.toBytes("row1"));

      // Reading the data
      Result result = table.get(g);

      // Reading values from Result class object
      byte [] value = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("name"));

      byte [] value1 = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("city"));

      // Printing the values
      String name = Bytes.toString(value);
      String city = Bytes.toString(value1);

      System.out.println("name: " + name + " city: " + city);
   }
}
```

Compile and execute the above program as shown below.

```
$javac RetriveData.java
$java RetriveData
```

The following should be the output:

```
name: Raju city: Delhi
```

Data Loading Techniques

Flume

- Apache Flume is a tool for data ingestion in HDFS.
- It collects, aggregates and transports large amount of streaming data such as log files, events from various sources like network traffic, social media, email messages etc. to HDFS.
- Flume is a highly reliable & distributed.

Basic idea behind Flume

- The main idea behind the Flume's design is to capture streaming data from various web servers to HDFS.
- It has simple and flexible architecture based on streaming data flows.
- It is fault-tolerant and provides reliability mechanism for Fault tolerance & failure recovery.

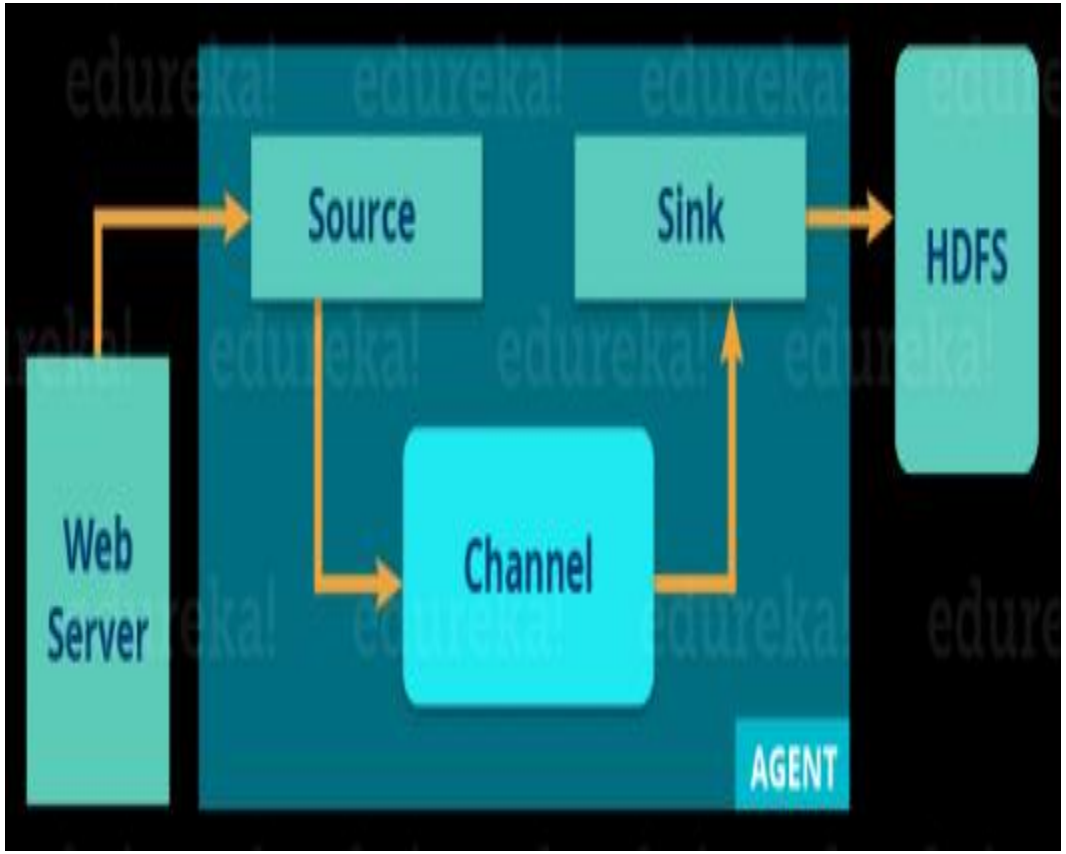
Advantages of Apache Flume

- Flume is scalable, reliable, fault tolerant and customizable for different sources and sinks.
- Apache Flume can store data in centralized stores (i.e data is supplied from a single store) like HBase & HDFS.
- Flume is horizontally scalable.
- If the read rate exceeds the write rate, Flume provides a steady flow of data between read and write operations.
- Flume provides reliable message delivery. The transactions in Flume are channel-based where two transactions (one sender & one receiver) are maintained for each message.

Advantages of Apache Flume

- Using Flume, we can ingest data from multiple servers into Hadoop.
- It gives us a solution which is reliable and distributed and helps us in collecting, aggregating and moving large amount of data sets like Facebook, Twitter and e-commerce websites.
- It helps us to ingest online streaming data from various sources like network traffic, social media, email messages, log files etc. in HDFS.
- It supports a large set of sources and destinations types.

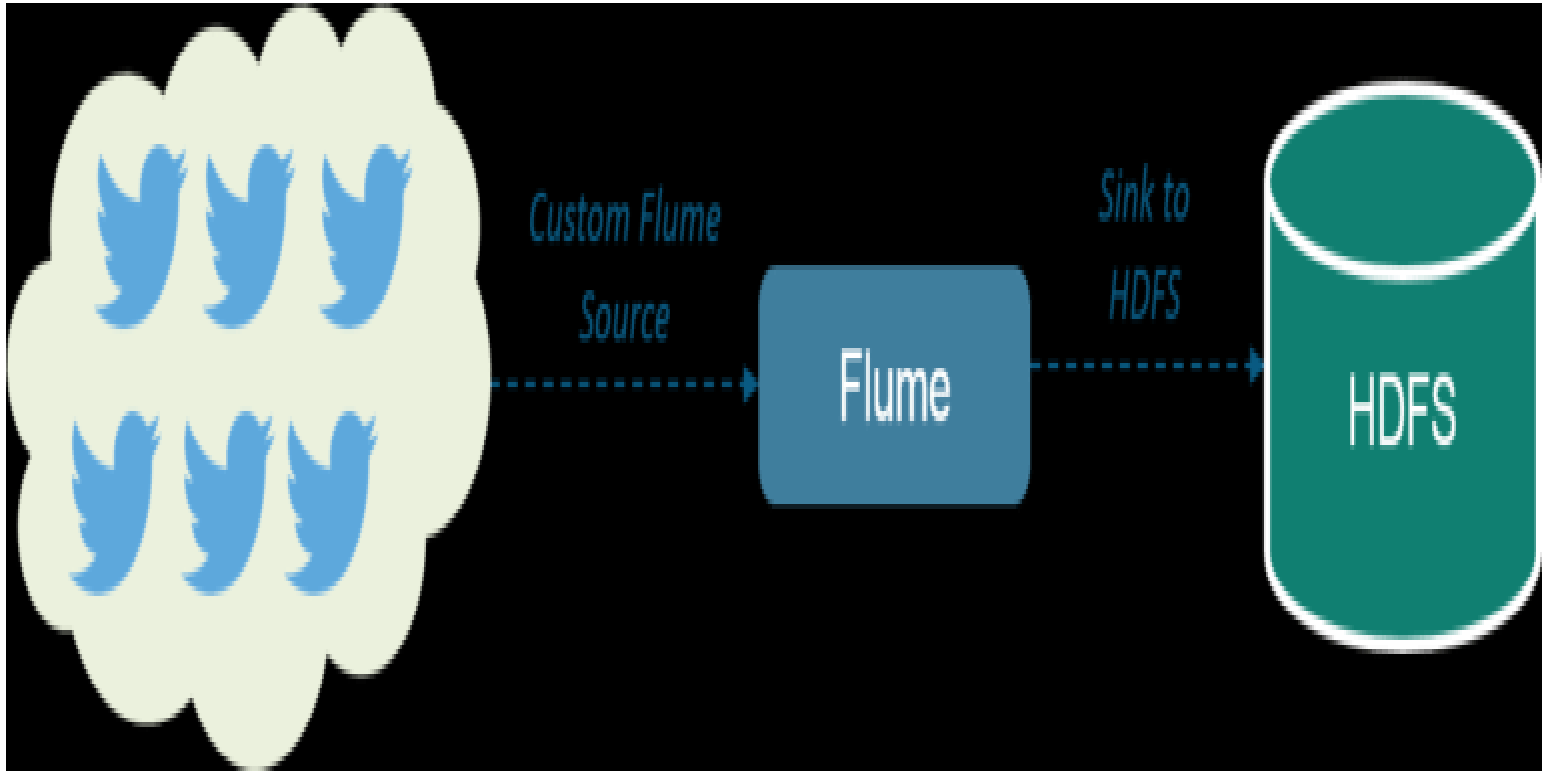
Flume Architecture



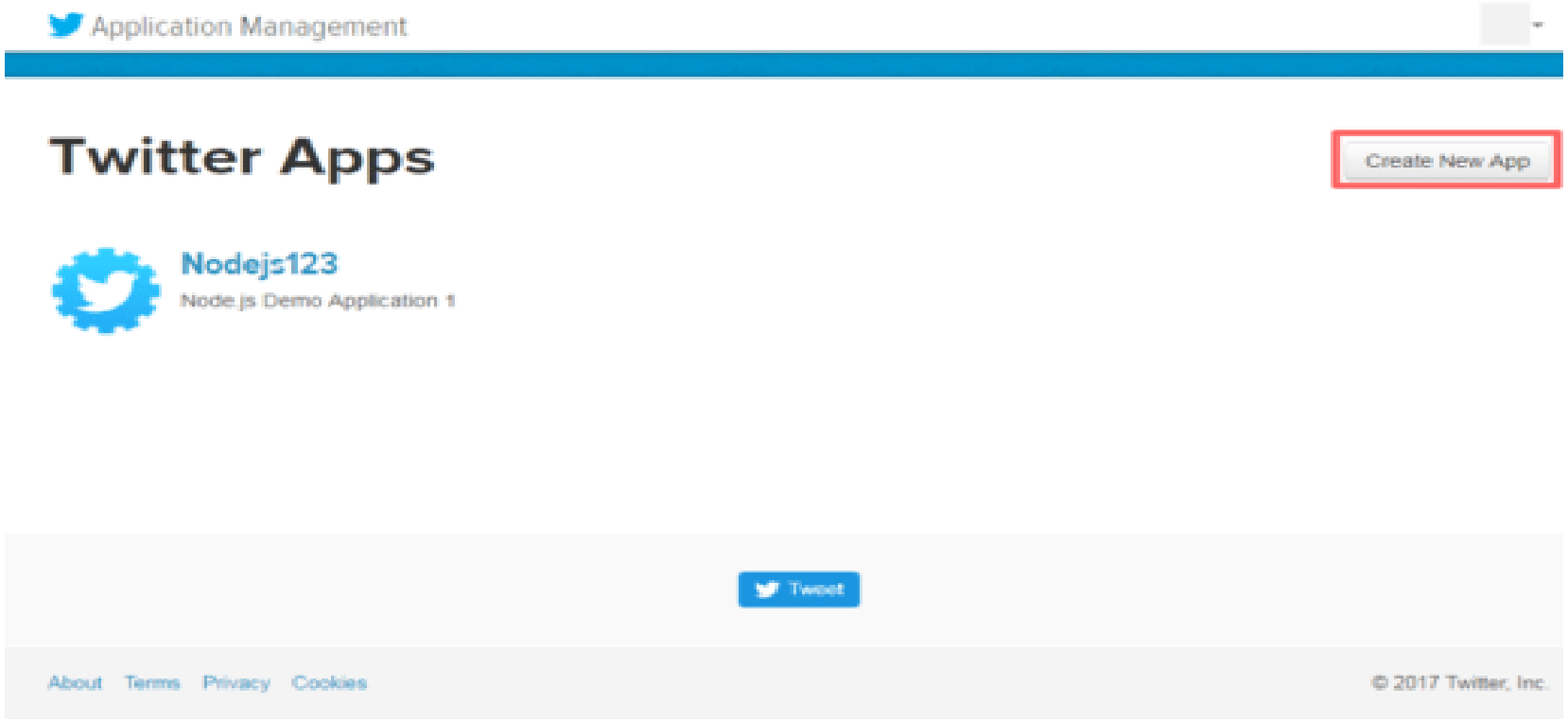
- There is a Flume agent which ingests the streaming data from various data sources to HDFS.
- From the diagram, we can easily understand that the web server indicates the data source.
- Twitter is among one of the famous sources for streaming data.

- The flume agent has 3 components: source, sink and channel.
  - **Source**: It accepts the data from the incoming streamline and stores the data in the channel.
  - **Channel**: In general, the reading speed is faster than the writing speed. Thus, we need some buffer to match the read & write speed difference. Basically, the buffer acts as a intermediary storage that stores the data being transferred temporarily and therefore prevents data loss. Similarly, channel acts as the local storage or a temporary storage between the source of data and persistent data in the HDFS.
  - **Sink**: Then, our last component i.e. Sink, collects the data from the channel and commits or writes the data in the HDFS permanently.

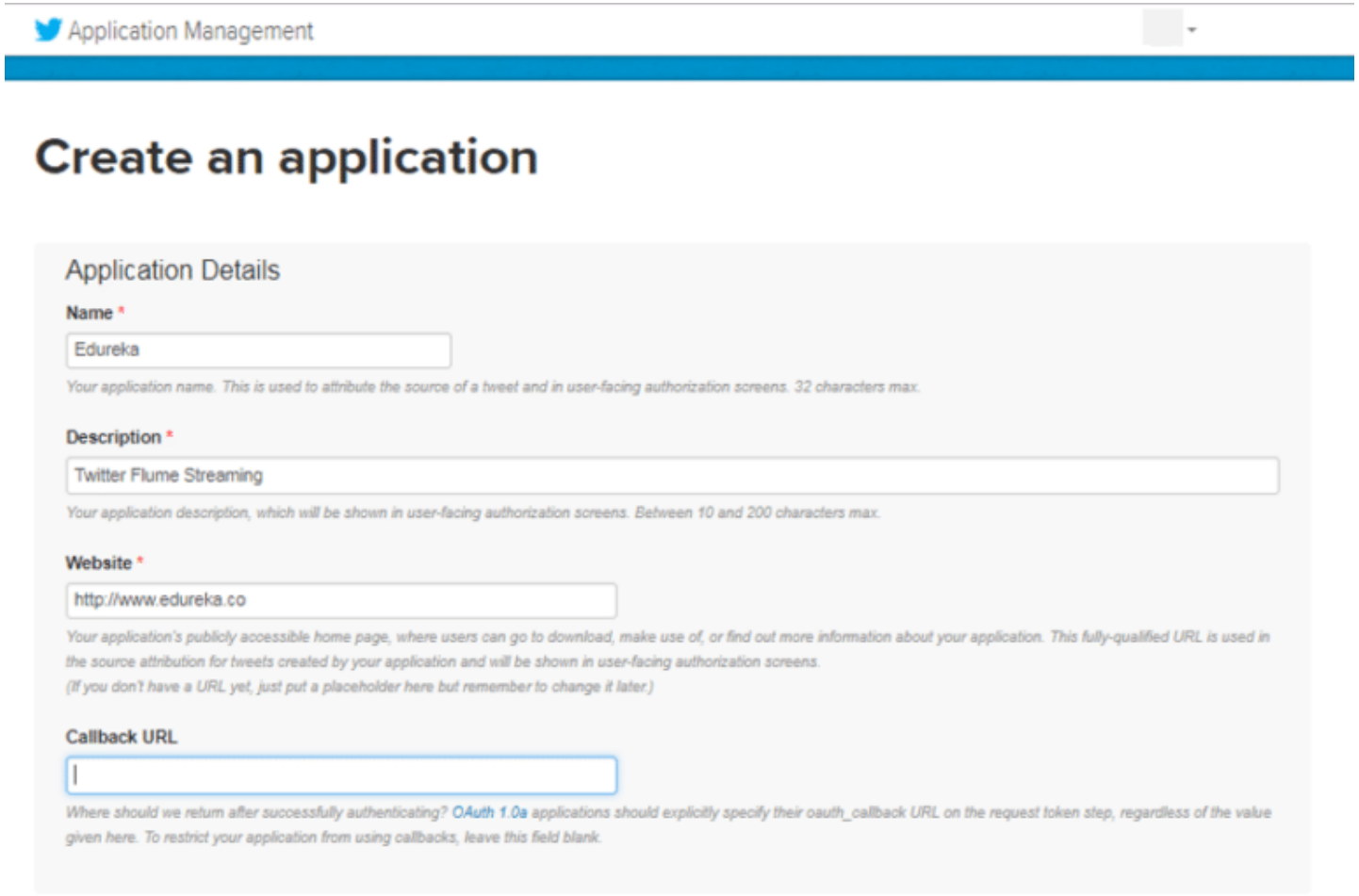
Streaming Twitter Data



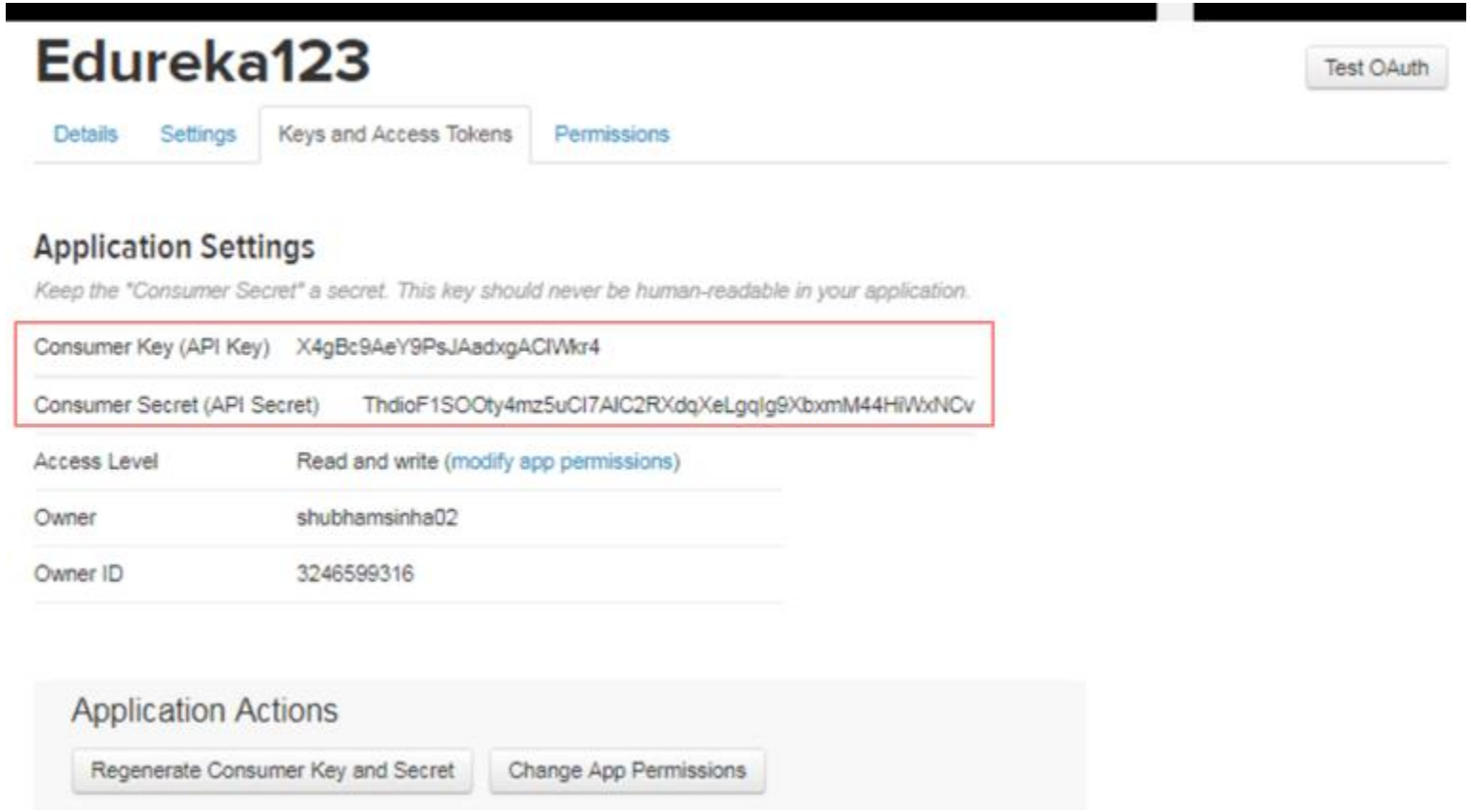
The first step is to create a Twitter application. For this, you first have to go to this url: https://apps.twitter.com/ and sign in to your Twitter account. Go to create application tab as shown in the below image.



Then, create an application as shown in the below image.



After creating this application, you will find Key & Access token. Copy the key and the access token. We will pass these tokens in our Flume configuration file to connect to this application.



Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	3246599316-gmNbAmeaOZa0t5jchAJqADF0VB5zdY4bQe5DYOT
Access Token Secret	TjDkFPpvSojsmGKEaBjsUOeCnSsVblxi18gHFDwVM6Dn
Access Level	Read and write
Owner	shubhamsinha02
Owner ID	3246599316

Now create a flume.conf file in the flume's root directory as shown in the below image. As we discussed, in the Flume's Architecture, we will configure our Source, Sink and Channel. Our Source is Twitter, from where we are streaming the data and our Sink is HDFS, where we are writing the data.

```
[edureka@localhost ~]$ cd $FLUME_HOME
[edureka@localhost apache-flume-1.7.0-bin]$ sudo gedit flume.conf
```

In source configuration we are passing the Twitter source type as *org.apache.flume.source.twitter.TwitterSource*. Then, we are passing all the four tokens which we received from Twitter. At last in source configuration we are passing the keywords on which we are going to fetch the tweets.

In the Sink configuration we are going to configure HDFS properties. We will set HDFS path, write format, file type, batch size etc. At last we are going to set memory channel as shown in the below image.

In the Sink configuration we are going to configure HDFS properties. We will set HDFS path, write format, file type, batch size etc. At last we are going to set memory channel as shown in the below image.

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = X4gBc9AeY9PsJAadxgACWk=4
TwitterAgent.sources.Twitter.consumerSecret =  ThdioF1SOOty4wz5uCI7AlC2RXzqXsLgqlg9XtxxM44HiWxNCv
TwitterAgent.sources.Twitter.accessToken = 324659931 6-gnNb4nae0Ze0t5jch4Jq4DF8V95zoY4bQe5DYOT
TwitterAgent.sources.Twitter.accessTokenSecret = TijDkF2pv5ojsn3KEsBjstUDeC=Ss7bIx18g+FOwVY6Dn
TwitterAgent.sources.Twitter.keywords = hadoop, big data, analytics, bigdata, cloudera, data science, data scientist, business intelligence,
mapreduce, data warehouse, data warehousing, mahout, hbase, nosql, newsql, businessintelligence, cloudcomputing

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://localhost:9000/user/flume/tweets/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000
TwitterAgent.sinks.HDFS.hdfs.rollInterval = 600

TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100
```

Source Properties

Sink Properties

Channel Properties

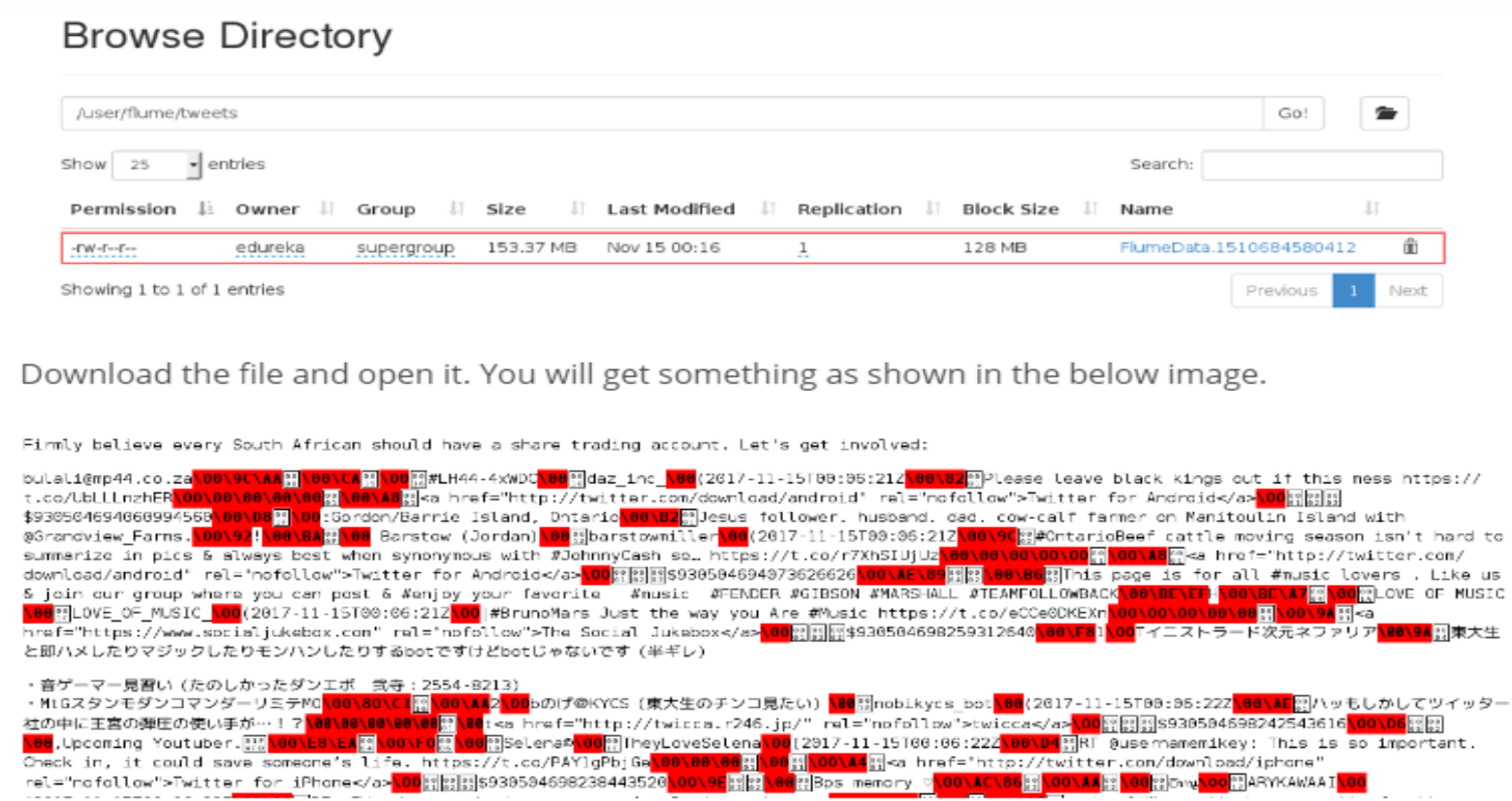
Now we are all set for execution. Let us go ahead and execute this command:

```
$FLUME_HOME/bin/flume-ng agent --conf ./conf/ -f $FLUME_HOME/flume.conf
```

```
$FLUME_HOME/bin/flume-ng agent --conf ./conf/ -f $FLUME_HOME/flume.conf
```

```
[edureka@localhost apache-flume-1.7.0-bin]$ bin/flume-ng agent --conf ./conf/ -f
 flume.conf Dflume.root.logger=DEBUG,console -n TwitterAgent
Info: Sourcing environment configuration script /usr/lib/apache-flume-1.7.0-bin/
conf/flume-env.sh
Info: Including Hadoop libraries found via (/usr/lib/hadoop-2.8.1/bin/hadoop) fo
r HDFS access
Info: Including Hive libraries found via () for Hive access
+ exec /usr/lib/jvm/jdk1.8.0_144/bin/java -Xmx20m -cp '/usr/lib/apache-flume-1.7
.0-bin/conf:/usr/lib/apache-flume-1.7.0-bin/lib/*:/usr/lib/hadoop-2.8.1/etc/hado
op:/usr/lib/hadoop-2.8.1/share/hadoop/common/lib/*:/usr/lib/hadoop-2.8.1/share/h
adoop/common/*:/usr/lib/hadoop-2.8.1/share/hadoop/hdfs:/usr/lib/hadoop-2.8.1/sha
re/hadoop/hdfs/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/hdfs/*:/usr/lib/hadoop-2
.8.1/share/hadoop/yarn/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/yarn/*:/usr/lib/
hadoop-2.8.1/share/hadoop/mapreduce/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/map
reduce/*:/usr/lib/hadoop-2.8.1/contrib/capacity-scheduler/*.jar:/lib/*' -Djava.l
ibrary.path=:/usr/lib/hadoop-2.8.1/lib/native org.apache.flume.node.Application
-f flume.conf Dflume.root.logger=DEBUG,console -n TwitterAgent
```

After executing this command for a while, and then you can exit the terminal using CTRL+C. Then you can go ahead in your Hadoop directory and check the mentioned path, whether the file is created or not.

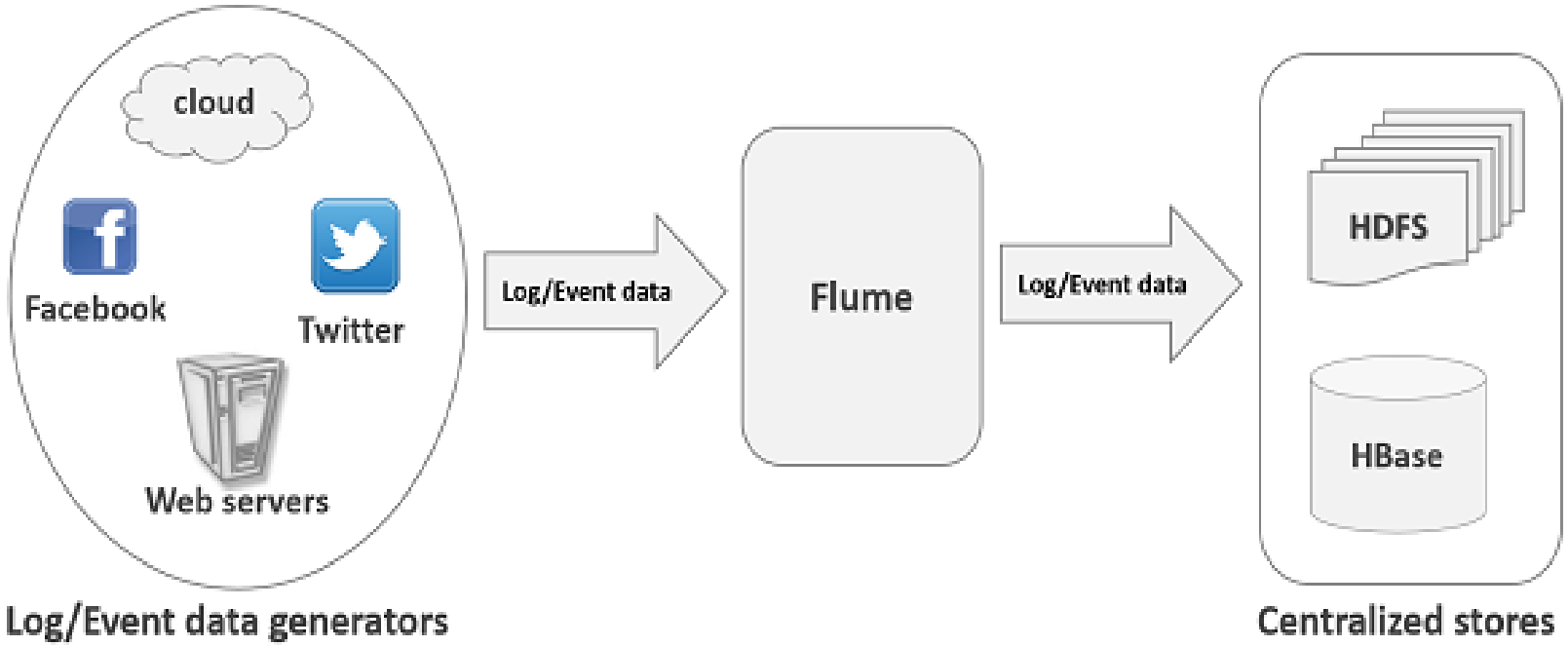


References:
https://www.edureka.co/blog/apache-flume-tutorial/

What is Flume?

Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store.

Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS


cloud
Facebook
Twitter
Web servers
Log/Event data generators
Log/Event data
Flume
Log/Event data
HDFS
HBase
Centralized stores

Applications of Flume

Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data in to Hadoop for analysis. Here, Apache Flume comes to our rescue.

Flume is used to move the log data generated by application servers into HDFS at a higher speed.

Advantages of Flume

Here are the advantages of using Flume −

- Using Apache Flume we can store the data in to any of the centralized stores (HBase, HDFS).
- When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
- Flume provides the feature of contextual routing.
- The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.
- Flume is reliable, fault tolerant, scalable, manageable, and customizab

Features of Flume

Some of the notable features of Flume are as follows −

- Flume ingests log data from multiple web servers into a centralized store (HDFS, HBase) efficiently.
- Using Flume, we can get the data from multiple servers immediately into Hadoop.
- Along with the log files, Flume is also used to import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart.
- Flume supports a large set of sources and destinations types.
- Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.
- Flume can be scaled horizontally.

What is Kafka?

Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption. Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss. Kafka is built on top of the ZooKeeper synchronization service. It integrates very well with Apache Storm and Spark for real-time streaming data analysis.

Kafka

Kafka tends to work very well as a replacement for a more traditional message broker. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications.

Point to Point Messaging System

In a point-to-point system, messages are persisted in a queue. One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, it disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor, but Multiple Order Processors can work as well at the same time. The following diagram depicts the structure.

KAFKA Message queueing

Publish-Subscribe Messaging System

In the publish-subscribe system, messages are persisted in a topic. Unlike point-to-point system, consumers can subscribe to one or more topic and consume all the messages in that topic. In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers. A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

Benefits

Following are a few benefits of Kafka −

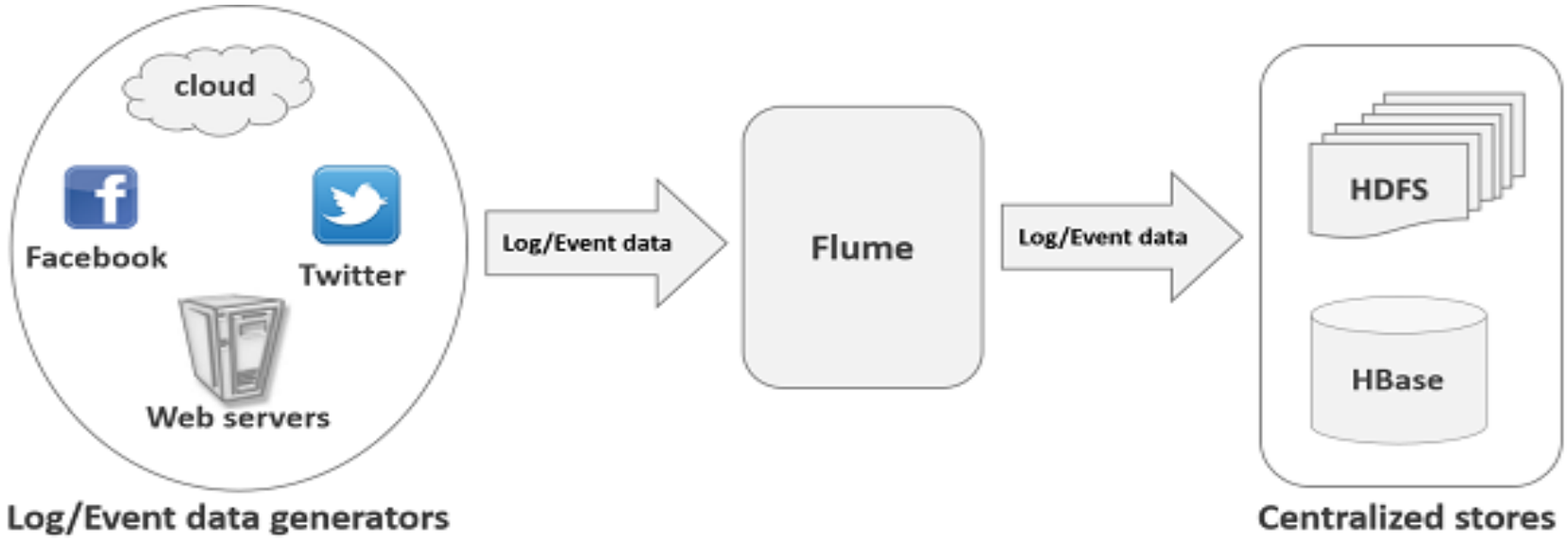
- Reliability − Kafka is distributed, partitioned, replicated and fault tolerance.
- Scalability − Kafka messaging system scales easily without down time..
- Durability − Kafka uses Distributed commit log which means messages persists on disk as fast as possible, hence it is durable..
- Performance − Kafka has high throughput for both publishing and subscribing messages. It maintains stable performance even many TB of messages are stored.

Kafka is very fast and guarantees zero downtime and zero data loss.

Need for Kafka

Kafka is a unified platform for handling all the real-time data feeds. Kafka supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures. It has the ability to handle a large number of diverse consumers. Kafka is very fast, performs 2 million writes/sec. Kafka persists all data to the disk, which essentially means that all the writes go to the page cache of the OS (RAM). This makes it very efficient to transfer data from page cache to a network socket.

KAFKA and FLUME



IBM Infosphere Big Insights

- InfoSphere® BigInsights™ is **a software platform for discovering, analyzing, and visualizing data from disparate sources**. You use this software to help process and analyze the volume, variety, and velocity of data that continually enters your organization every day.

Highlights of InfoSphere BigInsights

- BigInsights allows organizations to cost-effectively analyze a wide variety and large volume of data to gain insights that were not previously possible.
- BigInsights is focused on providing enterprises with the capabilities they need to meet critical business requirements while maintaining compatibility with the Hadoop project.
- BigInsights includes a variety of IBM technologies that enhance and extend the value of open-source Hadoop software to facilitate faster time-to-value, including application accelerators, analytical facilities, development tools, platform improvements and enterprise software integration.
- While BigInsights offers a wide range of capabilities that extend beyond the Hadoop functionality, IBM has taken an optin approach: you can use the IBM extensions to Hadoop based on your needs rather than being forced to use the extensions that come with InfoSphere BigInsights.

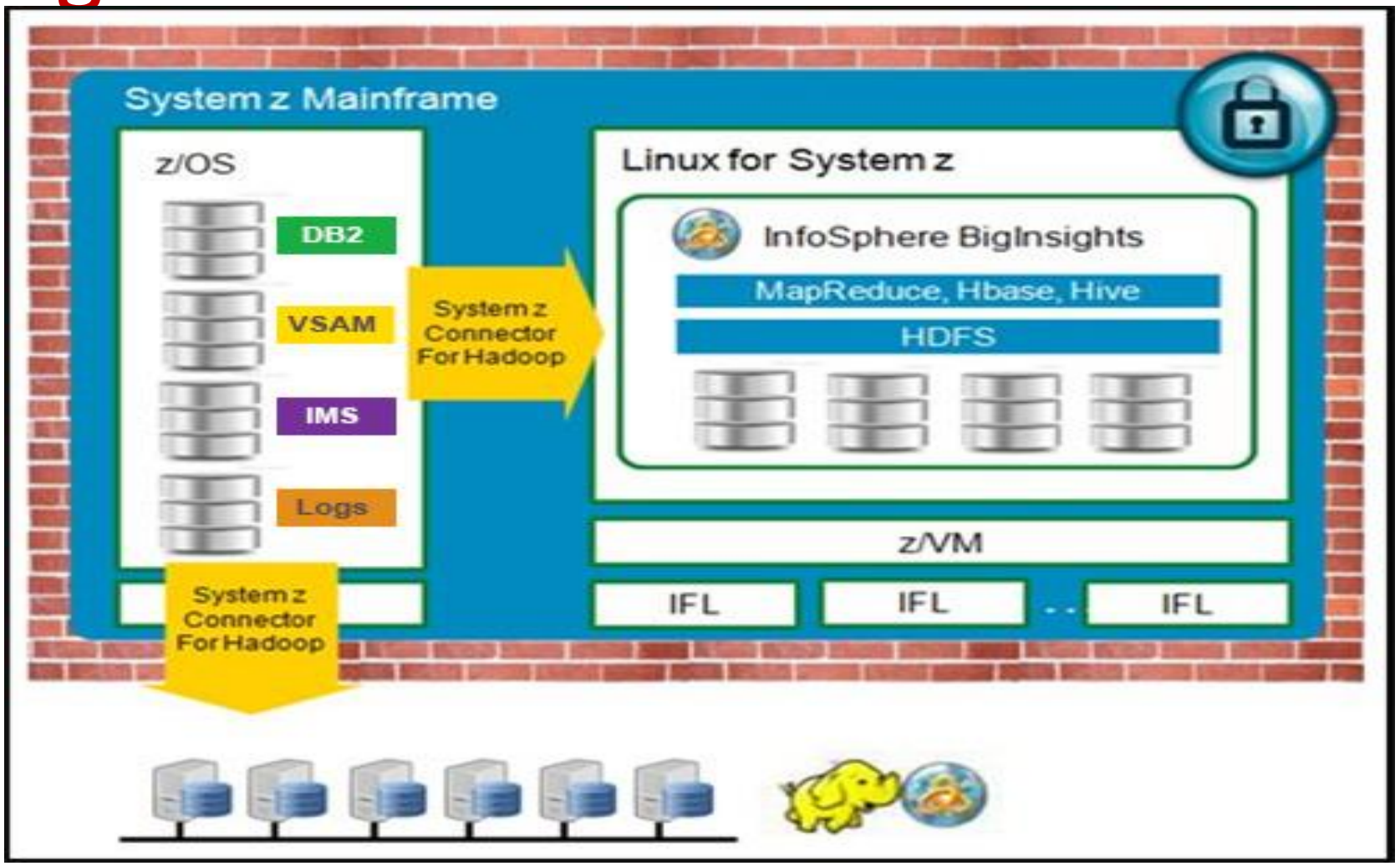
Highlights of InfoSphere BigInsights

- In addition to core capabilities for installation, configuration and management, InfoSphere BigInsights includes advanced analytics and user interfaces for the non-developer business analyst.
- It is flexible to be used for unstructured or semi-structured information; the solution does not require schema definitions or data preprocessing and allows for structure and associations to be added on the fly across information types.
- The platform runs on commonly available, low-cost hardware in parallel, supporting linear scalability; as information grows, we simply add more commodity hardware.

Highlights of InfoSphere BigInsights….

- InfoSphere BigInsights provides a unique set of capabilities that combine the innovation from the Apache Hadoop ecosystem with robust support for traditional skill sets and already installed tools. The ability to leverage existing skills and tools through open-source capabilities helps drive lower total cost of ownership and faster time-to-value. Thus InfoSphere BigInsights enables new solutions for problems that were previously too large and complex to solve cost-effectively.

Extend mainframe analytic capabilities with BigInsights



VSAM- Virtual Storage Access Method ; IMS-Information Management System

IFL processor

- The Integrated Facility for Linux is an IBM mainframe and Power Systems processor dedicated to running the Linux operating system. On IBM Z and IBM LinuxONE machines, IFLs can be used with or without hypervisors such as z/VM and KVM.

Advantages of IBM infosphere Big insights

- Hadoop applications can exist within the System z security perimeter.
- Clients can use mainframe technologies, including IBM HiperSockets™, to securely access production data, and move that data to and from Hadoop for processing.
- Clients can realize the management advantages of running Hadoop on a private cloud infrastructure, providing configuration flexibility and virtualized storage, and avoiding need to deploy and manage discrete cluster nodes and a separate network infrastructure.
- Clients can extend System z governance to hybrid Hadoop implementations.

Open source utilities in InfoSphere BigInsights:

- InfoSphere BigInsights is 100% compatible with open source Hadoop.
- open source utilities in InfoSphere BigInsights:
  - PIG
  - Hive / HCatalog
  - Oozie
  - HBASE
  - Zookeeper
  - Flume
  - Avro
  - Chukwa

Advanced software capabilities of IBM Infosphere:

- Big SQL: Big SQL is a rich, ANSI-compliant SQL implementation.
- SQL language compatibility
- Support for native data sources
- Performance
- Federation
- Security

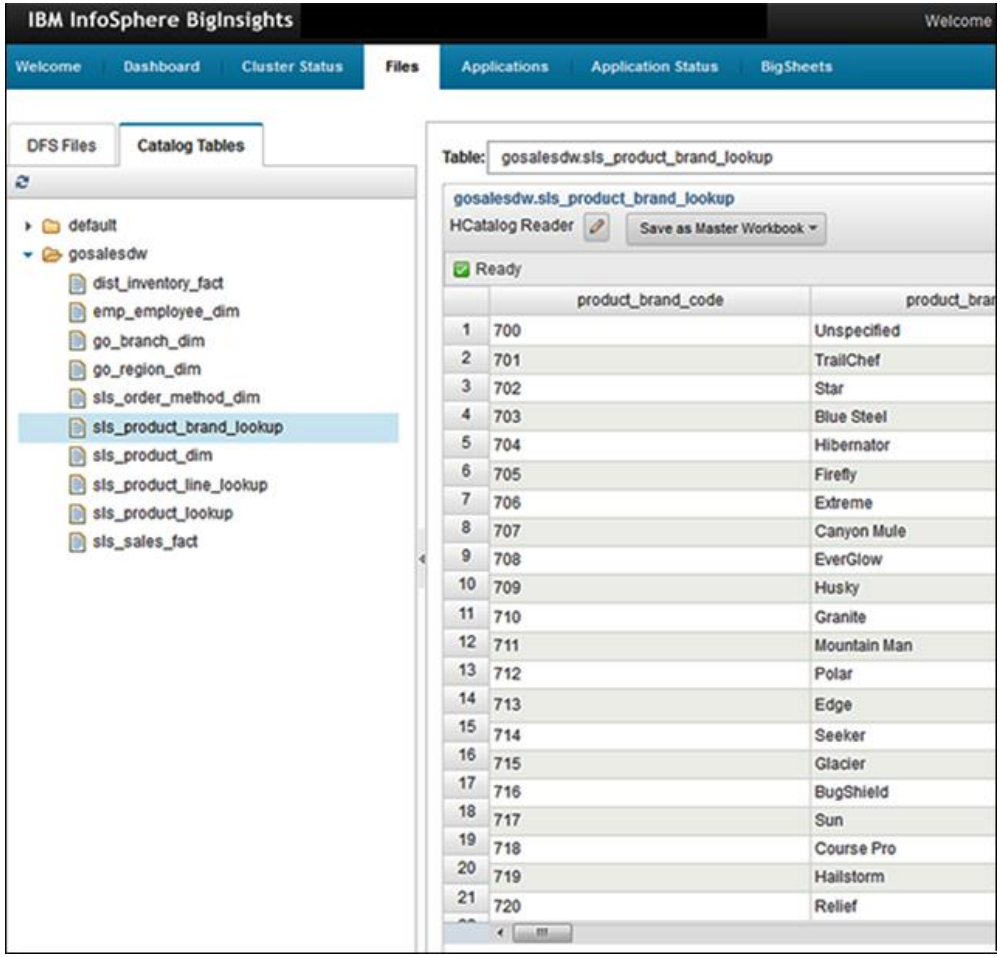
IBM infosphere interfaces

- Big R: Big R is a set of libraries that provide end-to-end integration with the popular R programming language that is included in InfoSphere BigInsights. Big R provides a familiar environment for developers and data scientists proficient with the R language.

Big Sheets

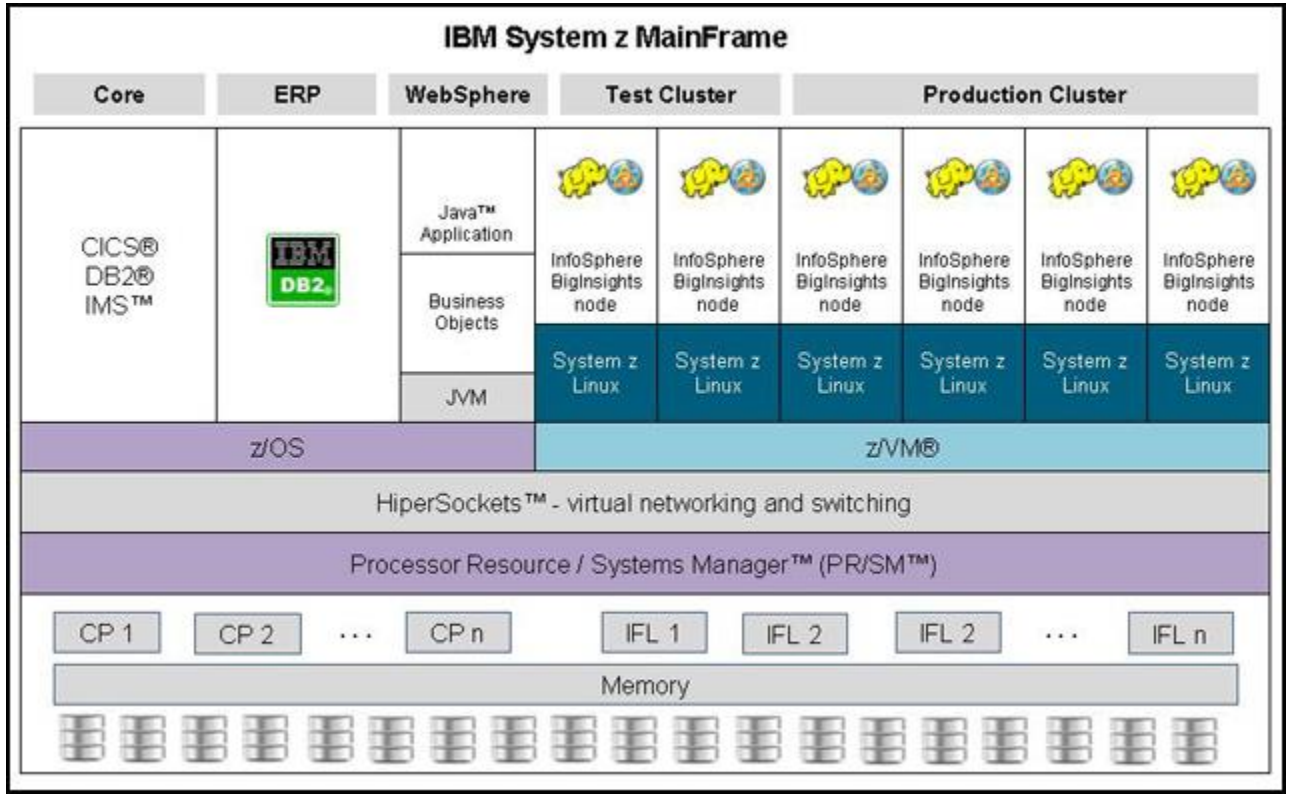
- Big Sheets: Big Sheets is a spreadsheet style data manipulation and visualization tool that allows business users to access and analyze data in Hadoop without the need to be knowledgeable in Hadoop scripting languages or MapReduce programming. The BigSheets interface is shown in Figure 3. Using built-in line readers, BigSheets can import data in multiple formats. In this example, it is importing data that is stored in Hive.

InfoSphere BigSheets interface



- **Application Accelerators**: IBM InfoSphere BigInsights extends the capabilities of open source Hadoop with accelerators that use pre-written capabilities for common big data use cases to build quickly high-quality applications. Here are some of the accelerators that are included in InfoSphere BigInsights:
- **Text Analytics Accelerators:** A set of facilities for developing applications that analyze text across multiple spoken languages
- **Machine Data Accelerators**: Tools that are aimed at developers that make it easy to develop applications that process log files, including web logs, mail logs, and various specialized file formats
- **Social Data Accelerators:** Tools to easily import and analyze social data at scale from multiple online sources, including tweets, boards, and blogs

Deploying Big insights on System Z



When to use Biginsights on a cloud based service:

- Clients do not want to be bothered with acquiring and maintain a Hadoop on cluster on their own premises.
- Data that is stored and processed on Hadoop comes largely from external sources as opposed to local data sources (such as cloud-based social media aggregators).
- Clients want to retain the flexibility to alter the size of clusters up and down based on changing requirements.

References

- IBM Infosphere:

https://namitkabra.wordpress.com/2015/01/08/what-is-infosphere-biginsights/

http://www.redbooks.ibm.com/abstracts/tips1215.html#contents

References

- NoSQL:https://www.xenonstack.com/blog/nosql-databases

References

- https://techvidvan.com/tutorials/mapreduce-job-execution-flow/
- Job configuration: https://www.tutorialspoint.com/map_reduce/map_reduce_api.htm#:~:text=the%20JobContext%20interface.-,Job%20Class,they%20will%20throw%20an%20IllegalStateException.