What is Hive?

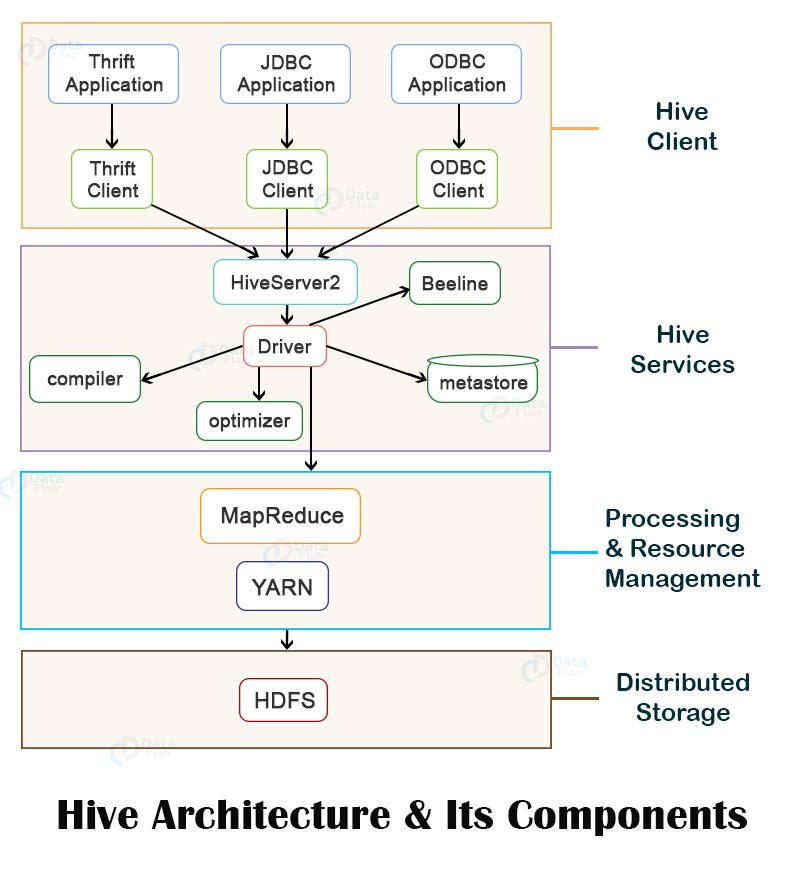
Apache Hive is an **open-source data warehousing tool** for performing distributed processing and data analysis. It was developed by **Facebook** to reduce the work of writing the Java MapReduce program.

Apache Hive uses a **Hive Query language**, which is a declarative language similar to SQL. Hive translates the hive queries into MapReduce programs.

It supports developers to perform processing and analyses on structured and semi-structured data by replacing complex java MapReduce programs with hive queries.

Hive makes the job easy for performing operations like

* Analysis of huge datasets
* Ad-hoc queries
* Data encapsulation

Hive Architecture

1. **Hive Client**
2. **Hive Services**
3. **Processing and Resource Management**
4. **Distributed Storage**

Hive supports applications written in any language like Python, Java, C++, Ruby, etc. using JDBC, ODBC, and Thrift drivers, for performing queries on the Hive. Hence, one can easily write a hive client application in any language of its own choice.

Hive clients are categorized into three types:

1. Thrift Clients

The Hive server is based on Apache Thrift so that it can serve the request from a thrift client.

2. JDBC client

Hive allows for the Java applications to connect to it using the JDBC driver. JDBC driver uses Thrift to communicate with the Hive Server.

3. ODBC client

Hive ODBC driver allows applications based on the ODBC protocol to connect to Hive. Similar to the JDBC driver, the ODBC driver uses Thrift to communicate with the Hive Server.

To perform all queries, Hive provides various services like the Hive server2, Beeline, etc. The various services offered by Hive are:

1. Beeline

The Beeline is a command shell supported by HiveServer2, where the user can submit its queries and command to the system. It is a **JDBC** client that is based on **SQLLINE CLI** (pure Java-console-based utility for connecting with relational databases and executing SQL queries).

#### 2. Hive Server 2

HiveServer2 is the successor of HiveServer1. HiveServer2 enables clients to execute queries against the Hive. It allows multiple clients to submit requests to Hive and retrieve the final results. It is basically designed to provide the best support for open API clients like JDBC and ODBC.

3. Hive Driver

The Hive driver receives the **HiveQL** statements submitted by the user through the command shell. It creates the session handles for the query and sends the query to the compiler.

4. Hive Compiler

Hive compiler parses the query. It performs semantic analysis and type-checking on the different query blocks and query expressions by using the metadata stored in metastore and generates an execution plan.

The execution plan created by the compiler is the **DAG(Directed Acyclic Graph)**, where each stage is a map/reduce job, operation on HDFS, a metadata operation.

5. Optimizer

Optimizer performs the transformation operations on the execution plan and splits the task to improve efficiency and scalability.

6. Execution Engine

Execution engine, after the compilation and optimization steps, executes the execution plan created by the compiler in order of their dependencies using Hadoop.

7. Metastore

Metastore is a central repository that stores the metadata information about the structure of tables and partitions, including column and column type information.

It also stores information of serializer and deserializer, required for the read/write operation, and HDFS files where data is stored. This metastore is generally a relational database.

Metastore provides a Thrift interface for querying and manipulating Hive metadata.

We can configure metastore in any of the two modes:

* **Remote:** In remote mode, metastore is a Thrift service and is useful for non-Java applications.
* **Embedded:** In embedded mode, the client can directly interact with the metastore using JDBC.

#### 8. HCatalog

HCatalog is the table and storage management layer for Hadoop. It enables users with different data processing tools such as Pig, MapReduce, etc. to easily read and write data on the grid.

It is built on the top of Hive metastore and exposes the tabular data of Hive metastore to other data processing tools.

#### 9. WebHCat

WebHCat is the REST API for HCatalog. It is an HTTP interface to perform Hive metadata operations. It provides a service to the user for running Hadoop MapReduce (or YARN), Pig, Hive jobs.

Hive internally uses a **MapReduce** framework as a defacto engine for executing the queries.

MapReduce is a software framework for writing those applications that process a massive amount of data in parallel on the large clusters of commodity hardware. MapReduce job works by splitting data into chunks, which are processed by map-reduce tasks.

### Working of Hive

**Step 1: executeQuery:** The user interface calls the execute interface to the driver.

**Step 2: getPlan:** The driver accepts the query, creates a session handle for the query, and passes the query to the compiler for generating the execution plan.

**Step 3: getMetaData:** The compiler sends the metadata request to the metastore.

**Step 4: sendMetaData:** The metastore sends the metadata to the compiler.

The compiler uses this metadata for performing type-checking and semantic analysis on the expressions in the query tree. The compiler then generates the execution plan (**Directed acyclic Graph**). For Map Reduce jobs, the plan contains **map operator trees** (operator trees which are executed on mapper) and **reduce operator tree** (operator trees which are executed on reducer).

**Step 5: sendPlan:** The compiler then sends the generated execution plan to the driver.

**Step 6: executePlan:** After receiving the execution plan from compiler, driver sends the execution plan to the execution engine for executing the plan.

**Step 7: submit job to MapReduce:**The execution engine then sends these stages of DAG to appropriate components.

For each task, either mapper or reducer, the deserializer associated with a table or intermediate output is used in order to read the rows from HDFS files. These are then passed through the associated operator tree.

Once the output gets generated, it is then written to the HDFS temporary file through the serializer. These temporary HDFS files are then used to provide data to the subsequent map/reduce stages of the plan.

For DML operations, the final temporary file is then moved to the table’s location.

**Step 8,9,10: sendResult:** Now for queries, the execution engine reads the contents of the temporary files directly from HDFS as part of a fetch call from the driver. The driver then sends results to the Hive interface

# **What is HBase**

Hbase is an open source and sorted map data built on Hadoop. It is column oriented and horizontally scalable.

It is based on Google's Big Table. It has set of tables which keep data in key value format. Hbase is well suited for sparse data sets which are very common in big data use cases. Hbase provides APIs enabling development in practically any programming language. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

## **Why HBase**

* RDBMS get exponentially slow as the data becomes large
* Expects data to be highly structured, i.e. ability to fit in a well-defined schema
* Any change in schema might require a downtime
* For sparse datasets, too much of overhead of maintaining NULL values

## **Features of Hbase**

* Horizontally scalable: You can add any number of columns anytime.
* Automatic Failover: Automatic failover is a resource that allows a system administrator to automatically switch data handling to a standby system in the event of system compromise
* Integrations with Map/Reduce framework: Al the commands and java codes internally implement Map/ Reduce to do the task and it is built over Hadoop Distributed File System.
* sparse, distributed, persistent, multidimensional sorted map, which is indexed by rowkey, column key,and timestamp.
* Often referred as a key value store or column family-oriented database, or storing versioned maps of maps.
* fundamentally, it's a platform for storing and retrieving data with random access.
* It doesn't care about datatypes(storing an integer in one row and a string in another for the same column).
* It doesn't enforce relationships within your data.
* It is designed to run on a cluster of computers, built using commodity hardware.

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.

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

## **MasterServer**

The master server -

* Assigns regions to the region servers and 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 and spread across the region servers.

### 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.

## **Zookeeper**

* Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
* Zookeeper has ephemeral nodes 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.

## **HBase Shell**

HBase contains a shell using which you can communicate with HBase. HBase uses the Hadoop File System to store its data. It will have a master server and region servers. The data storage will be in the form of regions (tables). These regions will be split up and stored in region servers.

The master server manages these region servers and all these tasks take place on HDFS. Given below are some of the commands supported by HBase Shell.

## **General Commands**

* **status** - Provides the status of HBase, for example, the number of servers.
* **version** - Provides the version of HBase being used.
* **table\_help** - Provides help for table-reference commands.
* **whoami** - Provides information about the user.

## **Data Definition Language**

These are the commands that operate on the tables in HBase.

* **create** - Creates a table.
* **list** - Lists all the tables in HBase.
* **disable** - Disables a table.
* **is\_disabled** - Verifies whether a table is disabled.
* **enable** - Enables a table.
* **is\_enabled** - Verifies whether a table is enabled.
* **describe** - Provides the description of a table.
* **alter** - Alters a table.
* **exists** - Verifies whether a table exists.
* **drop** - Drops a table from HBase.
* **drop\_all** - Drops the tables matching the ‘regex’ given in the command.
* **Java Admin API** - Prior to all the above commands, Java provides an Admin API to achieve DDL functionalities through programming. Under **org.apache.hadoop.hbase.client** package, HBaseAdmin and HTableDescriptor are the two important classes in this package that provide DDL functionalities.

## **Data Manipulation Language**

* **put** - Puts a cell value at a specified column in a specified row in a particular table.
* **get** - Fetches the contents of row or a cell.
* **delete** - Deletes a cell value in a table.
* **deleteall** - Deletes all the cells in a given row.
* **scan** - Scans and returns the table data.
* **count** - Counts and returns the number of rows in a table.
* **truncate** - Disables, drops, and recreates a specified table.
* **Java client API** - Prior to all the above commands, Java provides a client API to achieve DML functionalities, **CRUD** (Create Retrieve Update Delete) operations and more through programming, under org.apache.hadoop.hbase.client package. **HTable Put** and **Get** are the important classes in this package.

HBase is written in java, therefore it provides java API to communicate with HBase. Java API is the fastest way to communicate with HBase.

## **Class HBaseAdmin**

**HBaseAdmin** is a class representing the Admin. This class belongs to the **org.apache.hadoop.hbase.client** package. Using this class, you can perform the tasks of an administrator. You can get the instance of Admin using **Connection.getAdmin()** method.

### Methods and Description

|  |  |
| --- | --- |
| **S.No.** | **Methods and Description** |
| 1 | **void createTable(HTableDescriptor desc)**  Creates a new table. |
| 2 | **void createTable(HTableDescriptor desc, byte[][] splitKeys)**  Creates a new table with an initial set of empty regions defined by the specified split keys. |
| 3 | **void deleteColumn(byte[] tableName, String columnName)**  Deletes a column from a table. |
| 4 | **void deleteColumn(String tableName, String columnName)**  Delete a column from a table. |
| 5 | **void deleteTable(String tableName)**  Deletes a table. |

## **Class Descriptor**

This class contains the details about an HBase table such as:

* the descriptors of all the column families,
* if the table is a catalog table,
* if the table is read only,
* the maximum size of the mem store,
* when the region split should occur,
* co-processors associated with it, etc.

### Constructors

|  |  |
| --- | --- |
| **S.No.** | **Constructor and summary** |
| 1 | **HTableDescriptor(TableName name)**  Constructs a table descriptor specifying a TableName object. |

### Methods and Description

|  |  |
| --- | --- |
| **S.No.** | **Methods and Description** |
| 1 | **HTableDescriptor addFamily(HColumnDescriptor family)**  Adds a column family to the given descriptor |

What is JSON?

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

* A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
* An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with **{***left brace* and ends with **}***right brace*. Each name is followed by **:***colon* and the name/value pairs are separated by **,***comma*.

## **JSON Example**

'{"name":"John", "age":30, "car":null}'

It defines an object with 3 properties:

* name
* age
* car

Each property has a value.

If you parse the JSON string with a JavaScript program, you can access the data as an object:

* JSON is a lightweight data-interchange format
* JSON is plain text written in JavaScript object notation
* JSON is used to send data between computers
* JSON is language independent **\***

\*The JSON syntax is derived from JavaScript object notation, but the JSON format is text only. Code for reading and generating JSON exists in many programming languages.

## **Why Use JSON?**

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

JSON.parse()

JavaScript also has a built in function for converting an object into a JSON string:

JSON.stringify()