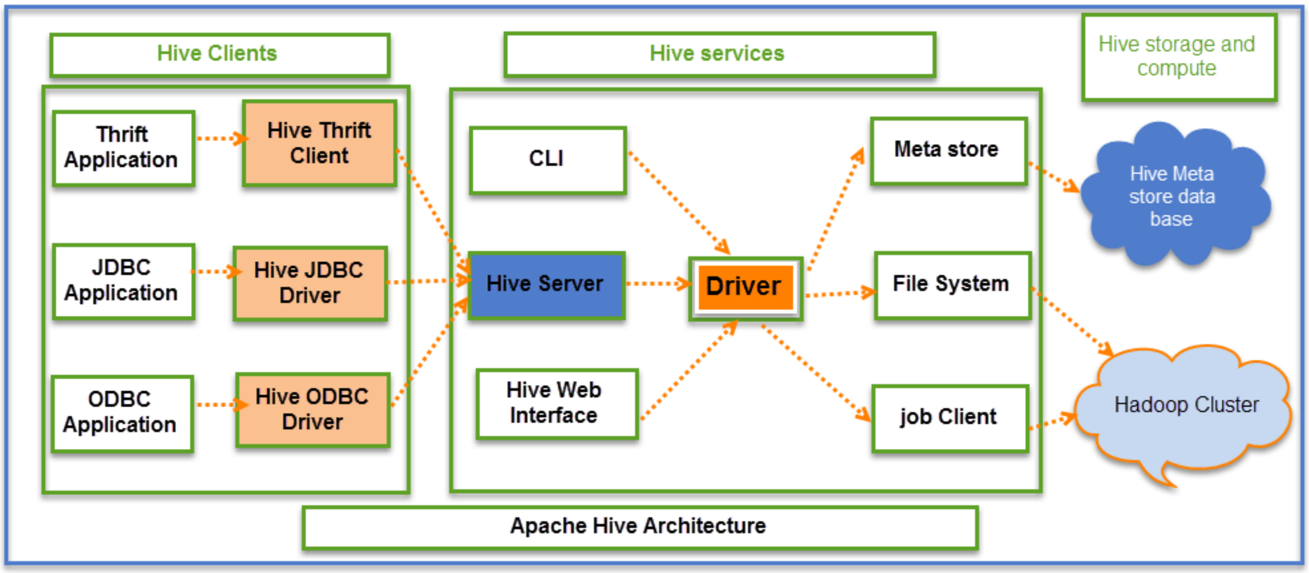
Hive Architecture

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



Hive Components

Hive Consists of Mainly 3 core parts

1. **Hive Clients**
2. **Hive Services**
3. **Hive Storage and Computing**

Hive Clients:

The Hive supports different types of client applications for performing queries. These clients are categorized into 3 types:

- **Thrift Clients –** As Apache Hive server is based on Thrift, so it can serve the request from all those languages that support Thrift.
- **JDBC Clients –** Apache Hive allows Java applications to connect to it using JDBC driver. It is defined in the class *apache.hadoop.hive.jdbc.HiveDriver.*

- **ODBC Clients –** ODBC Driver allows applications that support ODBC protocol to connect to Hive. For example JDBC driver, ODBC uses Thrift to communicate with the Hive server.

Hive Services:

Apache Hive provides various services as shown in above diagram. Now, let us look at each in detail:

a) CLI(Command Line Interface) – This is the default shell that Hive provides, in which you can execute your Hive queries and command directly.

b) Web Interface – Hive also provides web based GUI for executing Hive queries and commands.

c) Hive Server – It is built on Apache Thrift and thus is also called as Thrift server. It allows different clients to submit requests to Hive and retrieve the final result.

d) Hive Deriver – Driver is responsible for receiving the queries submitted Thrift, JDBC, ODBC, CLI, Web UL interface by a Hive client.

- **Complier –**After that hive driver passes the query to the compiler. Where parsing, type checking, and semantic analysis takes place with the help of schema present in the metastore.
- **Optimizer –** It generates the optimized logical plan in the form of a DAG (Directed Acyclic Graph) of MapReduce and HDFS tasks.
- **Executor –** Once compilation and optimization complete, execution engine executes these tasks in the order of their dependencies using Hadoop.

e) Metastore – Metastore is the central repository of Apache Hive metadata in the Hive Architecture. It stores metadata for Hive tables (like their schema and location) and partitions in a relational database. It provides client access to this information by using metastore service API. Hive metastore consists of two fundamental units:

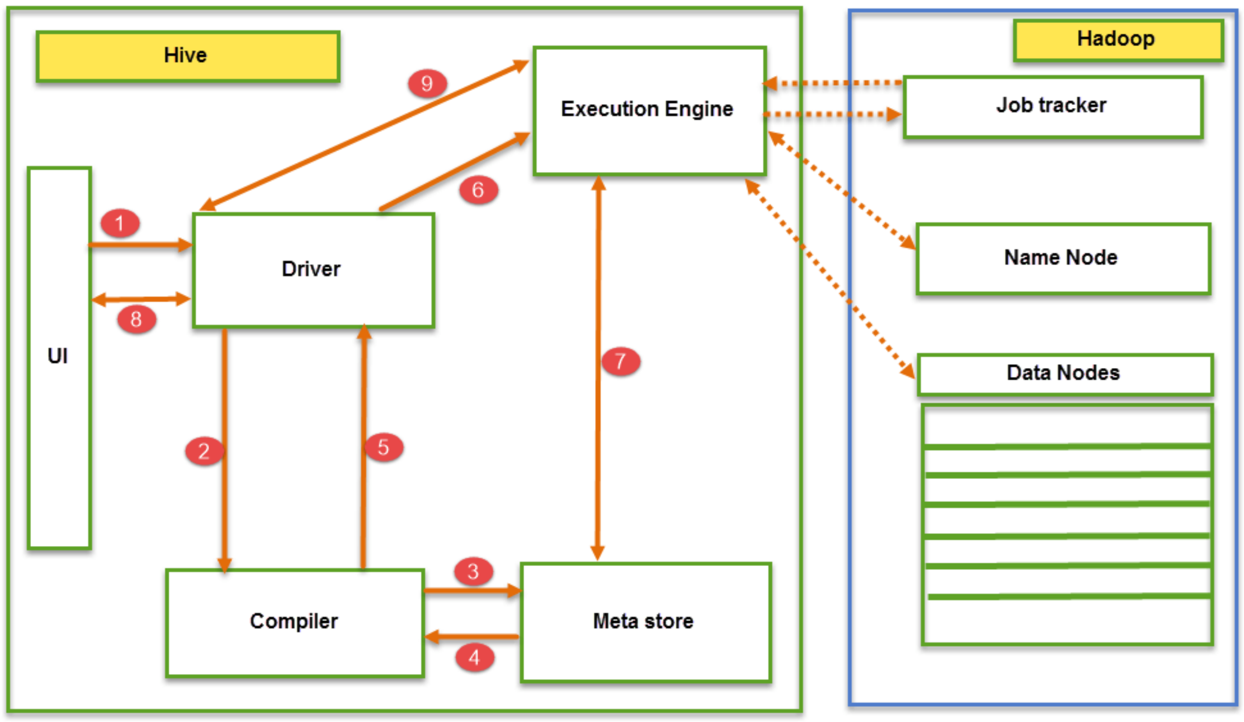
- A service that provides metastore access to other Apache Hive services.
- Disk storage for the Hive metadata which is separate from

Hive Storage and Computing:

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

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

Job exectution flow:



From the above screenshot we can understand the Job execution flow in Hive with Hadoop

The data flow in Hive behaves in the following pattern;

- User Interface (UI) calls the execute interface to the Driver.
- The driver creates a session handle for the query. Then it sends the query to the compiler to generate an execution plan.
- The compiler needs the metadata. So it sends a request for *getMetaData*. Thus receives the *sendMetaData* request from Metastore.
- Now compiler uses this metadata to type check the expressions in the query. The compiler generates the plan which is **DAG** of stages with each stage being either a map/reduce job, a metadata operation or an operation

on HDFS. The plan contains map operator trees and a reduce operator tree for map/reduce stages.

- Now execution engine submits these stages to appropriate components. After in each task the deserializer associated with the table or intermediate outputs is used to read the rows from HDFS files. Then pass them through the associated operator tree. Once it generates the output, write it to a temporary HDFS file through the serializer. Now temporary file provides the subsequent map/reduce stages of the plan. Then move the final temporary file to the table's location for DML operations.
- Now for queries, execution engine directly read the contents of the temporary file from HDFS as part of the fetch call from the Driver.

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