**HIVE ARCHITECTURE:**

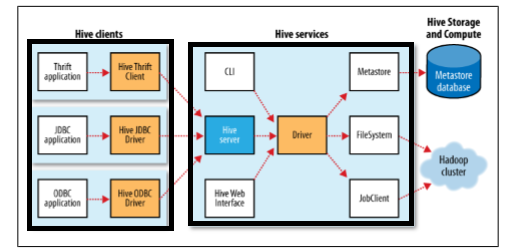


Fig 1: Hadoop Hive Architecture

The above diagram shows the basic **Hadoop Hive architecture**.

Primarily The diagram represents **CLI (Command Line Interface), JDBC/ODBC and Hive Web Interface.** This represents when user comes with CLI (Hive Terminal) it directly connected to **Hive Drivers**.

There are 3 major components in Hive as shown in the diagram. They are **Hive Clients**, **Hive Services** and **Meta Store**. Under hive client, we can have different ways to connect to HIVE SERVER in hive services.

These are **Thrift client**, **ODBC driver** and **JDBC driver**. The thrift client provides an easy environment to execute the hive commands from a vast range of programming languages. Thrift client bindings for Hive are available for C++, Java, PHP scripts, python scripts and Ruby. Similarly, JDBC and ODBC drivers can be used for communication between hive client and hive servers for compatible options.

The hive driver receives the tasks (Queries) from user and send to Hadoop architecture. The Hadoop architecture uses name node.data node.job tracker and task tracker for receiving and dividing the work what Hive sends to Hadoop.

## Job Execution inside Hive:

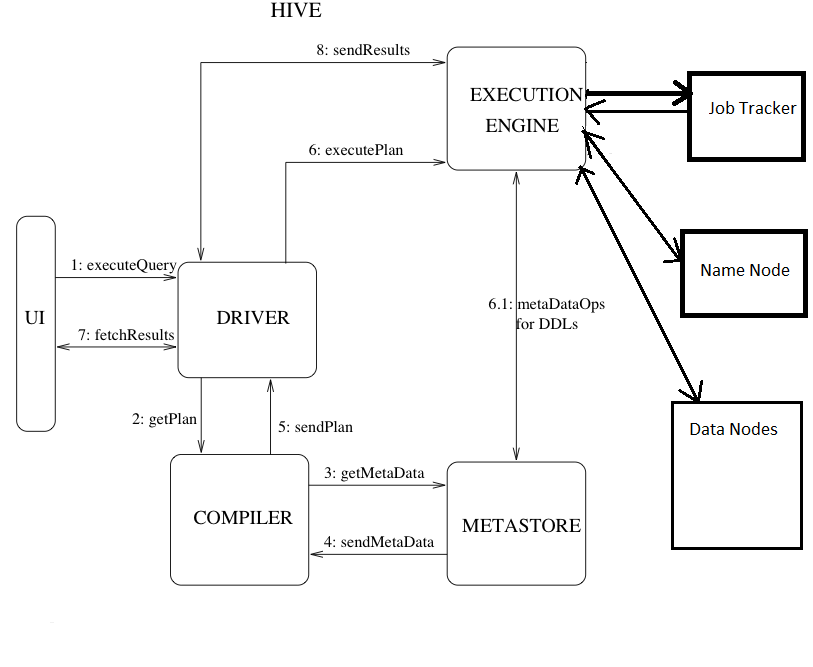


Fig 2: Internal Hive Architecture

HIVESERVER is an API that allows the clients (JDBC) to execute the queries on hive data warehouse and get the desired results. Under hive services driver, compiler and execution engine interact with each other and process the query.

The client submits the query via a GUI. The driver receives the queries in the first instance from GUI and it will define session handlers, which will fetch required APIs that is designed with different interfaces like JDBC or ODBC. The compiler creates the plan for the job to be executed. Compiler in turn is in contact with matter and its gets metadata from Meta Store.

Execution Engine (EE) is the key component here to execute a query by directly communicating with Job Tracker, Name Node and Data nodes. As discussed earlier, by running hive query at the backend, it will generate a series of MR (Map Reduce) Jobs. In this scenario, the execution engine plays like a bridge between hive and Hadoop to process the query. For DFS operations, EE contacts Name Node.

At the end, EE is going to fetch desired results from Data Nodes. EE will be having bi-directional communication with Metastore. In hive, side is a framework to serialize and de-serialize input and output data from HDFS to local or vice versa.

Metastore is used for collection of all the Hive metadata and it’s having back up services to backup meta store info. The service runs on the same JVM as the services of hive running on. The structural information of tables, their columns, column types and similarly the partition structure information will also be stored in this.

**Query Flow through the System: (Step by Step Guide)**

**Step 1:**The UI calls the execute interface to the Driver

**Step 2:** The Driver creates a session handle for the query and sends the query to the compiler to generate an execution plan

**Step 3&4:**The compiler needs the metadata so send a request for getMetaData and receives the sendMetaData request from MetaStore.

**Step 5:**This metadata is used to typecheck the expressions in the query tree as well as to prune partitions based on query predicates. The plan generated by the compiler is a DAG of stages with each stage being either a map/reduce job, a metadata operation or an operation on HDFS. For map/reduce stages, the plan contains map operator trees (operator trees that are executed on the mappers) and a reduce operator tree (for operations that need reducers).

**Step 6:**  The execution engine submits these stages to appropriate components (steps 6, 6.1, 6.2 and 6.3). In each task (mapper/reducer) the deserializer associated with the table or intermediate outputs is used to read the rows from HDFS files and these are passed through the associated operator tree. Once the output generate it is written to a temporary HDFS file though the serializer. The temporary files are used to provide the subsequent map/reduce stages of the plan. For DML operations the final temporary file is moved to the table’s location

**Step 7&8&9:**For queries, the contents of the temporary file are read by the execution engine directly from HDFS as part of the fetch call from the Driver

**HIVE COMPONENTS:**

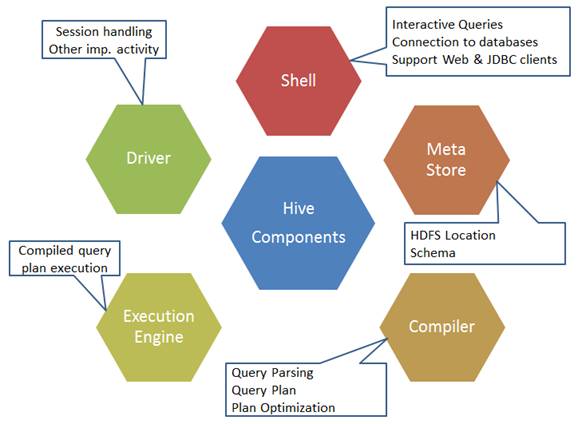


Fig 3: Hive Components

**UI/Shell:**

Hive 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).The user interface allows for users to submit queries and other operations to the system.

**Driver:**

The Driver manages the life cycle of a **HiveQL** statement during compilation, optimization and execution.

The Driver is used for receives the quires from UI .This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces.

**Compiler:**

The Compiler is invoked by the driver upon receiving a HiveQL statement. The compiler translates this statement into a plan which consists of a DAG of map/reduce jobs.

The component that parses the query, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.

**MetaStore:**

The **Metastore** is the system catalog. All other components of Hive interact with the Metastore.

Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.

It stores all the structure information of the various tables and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding HDFS files where the data is stored.

**Execution Engine:**

The component which executes the execution plan created by the compiler. The plan is a DAG of stages. The execution engine manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components.