

[Apache Hive](http://hive.apache.org/) helps to analyze large data by using the query language called **HiveQL** for data source, such as HDFS or [HBase](http://www.cubrid.org/blog/tags/HBase/). Architecture is divided into Map-Reduce-oriented execution, meta data information for a data storage, and an execution part that receives a query from user or applications for execution.

## Hive Architecture

* Web interface - The user interface for users to submit queries and other operations to the system. Currently the system has a command line interface and a web based GUI is being developed.
* Thrift HiveServer is an optional service that allows a remote client to submit requests to Hive, using a variety of programming languages, and retrieve results. HiveServer is built on Apache ThriftTM (<http://thrift.apache.org/>), therefore it is sometimes called the *Thrift server.*

HiveServer cannot handle concurrent requests from more than one client. This is actually a limitation imposed by the Thrift interface that HiveServer exports, and can't be resolved by modifying the HiveServer code.

[HiveServer2](https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients) is a rewrite of HiveServer that addresses these problems, starting with Hive 0.11.0.

* Driver - The component which receives the queries. This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces.
* Compiler - The component that parses the query, does semantic analysis on the different qurey 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 component that stores all the structure information of the various table 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.

The UI calls the execute interface to the Driver. The Driver creates a session handle for the query and sends the query to the compiler to generate an execution plan. The compiler gets the necessary metadata from the metastore. 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 operations 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). The execution engines submits these stages to appropriate components. 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 is generated, it is written to a temporary hdfs file though the serializer(this happens in the mapper in case the operation does not need a reduce). The temporary files are used to provide data to subsequent map/reduce stages of the plan. For DML operations the final temporary file is moved to the tables location. This scheme is used to ensure that dirty data is not read(file rename being an atomic operation in hdfs). 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.