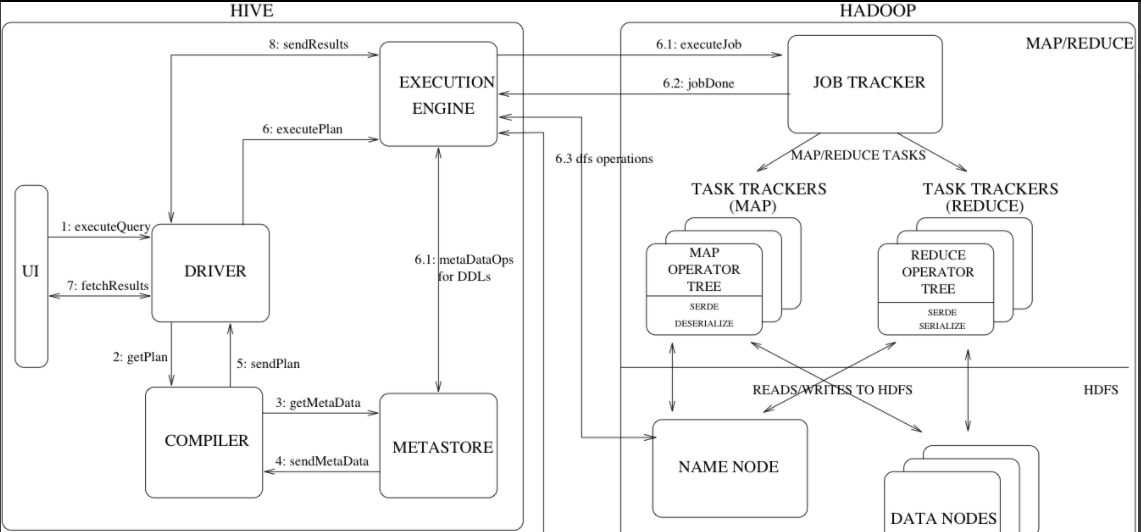
**Explain Hive architecture and components**



High level components of hive are:

* **UI**: The user interface for users to submit queries and other operations to the system.
* **Driver** – The component which receives the queries.
* **Compiler**:
  + Parser: Transform a query string to a parse tree representation.
  + Semantic Analyzer – Transform the parse tree to an internal query representation, which is still block based and not an operator tree. The column names are verified and expansions like \* are performed.
  + Logical Plan Generator – Convert the internal query representation to a logical plan, which consists of a tree of operators. Some of the operators are relational algebra operators like 'filter', 'join' etc. But some of the operators are Hive specific and are used later on to convert this plan into a series of map-reduce jobs. One such operator is a reduceSink operator which occurs at the map-reduce boundary.
  + Query Plan Generator – Convert the logical plan to a series of map-reduce tasks. The operator tree is recursively traversed, to be broken up into a series of map-reduce serializable tasks which can be submitted later on to the map-reduce framework for the Hadoop distributed file system.
* **Optimizer**:

More plan transformations are performed by the optimizer. The optimizer is an evolving component

* **Metastore**:

It offers two features.

* 1. Data abstraction: information about data format, extractors, and loaders are given at the time of table creation and reused every time the table is referenced. So we don’t have to give them at the time of query.
  2. Data discovery: enables users to discover and explore relevant and specific data in the warehouse.

Metadata objects:

* + 1. Database – is a namespace for tables. The database 'default' is used for tables with no user-supplied database name.
    2. Table – Metadata for a table contains list of columns, owner, storage and SerDe information.
    3. Partition – Each partition can have its own columns and SerDe and storage information. This facilitates schema changes without affecting older partitions.

**Metastore Architecture:**

Metastore is an object store with a database or file backed store. The database backed store is implemented using an object-relational mapping (ORM) solution called the [DataNucleus](http://www.datanucleus.org/).

Advantage:  relational database is queriability of metadata.

Disadvantage: because of using a separate data store for metadata instead of using HDFS are synchronization and scalability issues

The metastore can be configured to be used in a couple of ways: remote and embedded. In remote mode, the metastore is a [Thrift](https://thrift.apache.org/) service. This mode is useful for non-Java clients. In embedded mode, the Hive client directly connects to an underlying metastore using JDBC. This mode is useful because it avoids another system that needs to be maintained and monitored.

Hive Query language:

HiveQL is an SQL-like query language for Hive. It mostly mimics SQL syntax for creation of tables, loading data into tables and querying the tables. HiveQL also allows users to embed their custom map-reduce scripts. Another feature unique to HiveQL is multi-table insert. In this construct, users can perform multiple queries on the same input data using a single HiveQL query. Hive optimizes these queries to share the scan of the input data, thus increasing the throughput of these queries several orders of magnitude.

**Flow of the diagram:**

1. The UI calls the execute interface to the Driver.
2. The Driver creates a session handle for the query and sends the query to the compiler to generate an execution plan.
3. The compiler needs the metadata. So it sends a request for getMetaData. Thus receives the sendMetaData request from Metastore.
4. 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
5. 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 table's 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.

Hive Data Model

Data in Hive is organized into:

* **Tables** – These are analogous to Tables in Relational Databases. Tables can be filtered, projected, joined and union. Additionally all the data of a table is stored in a directory in HDFS. Hive also supports the notion of external tables wherein a table can be created on prexisting files or directories in HDFS by providing the appropriate location to the table creation DDL. The rows in a table are organized into typed columns similar to Relational Databases.
* **Partitions** – Each Table can have one or more partition keys which determine how the data is stored, for example query results for a particular attribute will be stored in the subfolder inside the table with the name attribute value.
* **Buckets** – Data in each partition may in turn be divided into Buckets based on the hash of a column in the table. Each bucket is stored as a file in the partition directory. Bucketing allows the system to efficiently evaluate queries that depend on a sample of data
* Apart from primitive column types (integers, floating point numbers, generic strings, dates and booleans), Hive also supports arrays and maps. Additionally, users can compose their own types programmatically from any of the primitives, collections or other user-defined types. The typing system is closely tied to the SerDe (Serailization/Deserialization) and object inspector interfaces. User can create their own types by implementing their own object inspectors, and using these object inspectors they can create their own SerDes to serialize and deserialize their data into HDFS files).