**Hive**

Apache Hive is a data warehouse system built on top of Apache Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in various databases and file systems that integrate with Hadoop, including the MapR Data Platform with MapR XD and MapR Database.

Hive is a Hadoop component that is normally deployed by data analysts. Hive is used more by researchers and programmers. It is an open-source data warehousing system, which is exclusively used to query and analyze huge datasets stored in Hadoop. Hive easily integrates with traditional data center technologies using the familiar JDBC/ODBC interface.

Even though Apache Pig can also be deployed for the same purpose, The three important functionalities for which Hive is deployed are data summarization, data analysis, and data query.

The query language, exclusively supported by Hive, is HiveQL. This language translates SQL-like queries into MapReduce jobs for deploying them on Hadoop. HiveQL also supports MapReduce scripts.

**File systems supported by Hive are:**

* Flat files or text files
* Sequence files consisting of binary key–value pairs
* RCFiles that store columns of a table in a columnar database

**Architecture of Apache Hive**

## Major Components of Hive Architecture

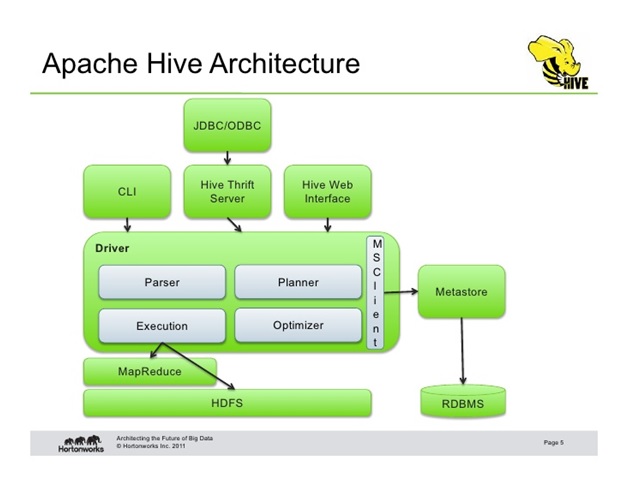
**Metastore:**It is the repository of metadata. This metadata consists of data for each table like its location and schema. It also holds the information for partition metadata which lets you monitor various distributed data progresses in the cluster. This data is generally present in the relational databases. The metadata keeps track of the data, replicates it, and provides a backup in the case of data loss.

**Driver:** The driver receives HiveQL statements and works like a controller. It monitors the progress and life cycle of various executions by creating sessions. The driver stores the metadata that is generated while executing the HiveQL statement. When the reducing operation is completed by the Map Reduce job, the driver collects the data points and query results.

**Compiler:** The compiler is assigned with the task of converting a HiveQL query into a MapReduce input. It includes a method to execute the steps and tasks needed to let the HiveQL output as needed by MapReduce.

**Optimizer:** This performs various transformation steps for aggregation and pipeline conversion by a single join for multiple joins. It also is assigned to split a task while transforming data, before the reduce operations, for improved efficiency and scalability.

**Executor:** The executor executes tasks after the compilation and optimization steps. It directly interacts with the Hadoop Job Tracker for scheduling the tasks to be run.

[](https://cdn.intellipaat.com/blog/wp-content/uploads/2016/12/Architecture-of-Apache-Hive.jpg)

**CLI, UI, and Thrift Server:** The command-line interface (CLI) and the user interface (UI) submit queries and process monitoring and instructions so that the external users can interact with Hive. Thrift Server lets other clients interact with Hive.

**Job execution flow in Hive with Hadoop- Flow Chart:**

Job execution flow in Hive with Hadoop is demonstrated step by step.

Step-1: Execute Query –

Interface of the Hive such as Command Line or Web user interface delivers query to the driver to execute. In this, UI calls the execute interface to the driver such as ODBC or JDBC.

Step-2: Get Plan –

Driver designs a session handle for the query and transfer the query to the compiler to make execution plan. In other words, driver interacts with the compiler.

Step-3: Get Metadata –

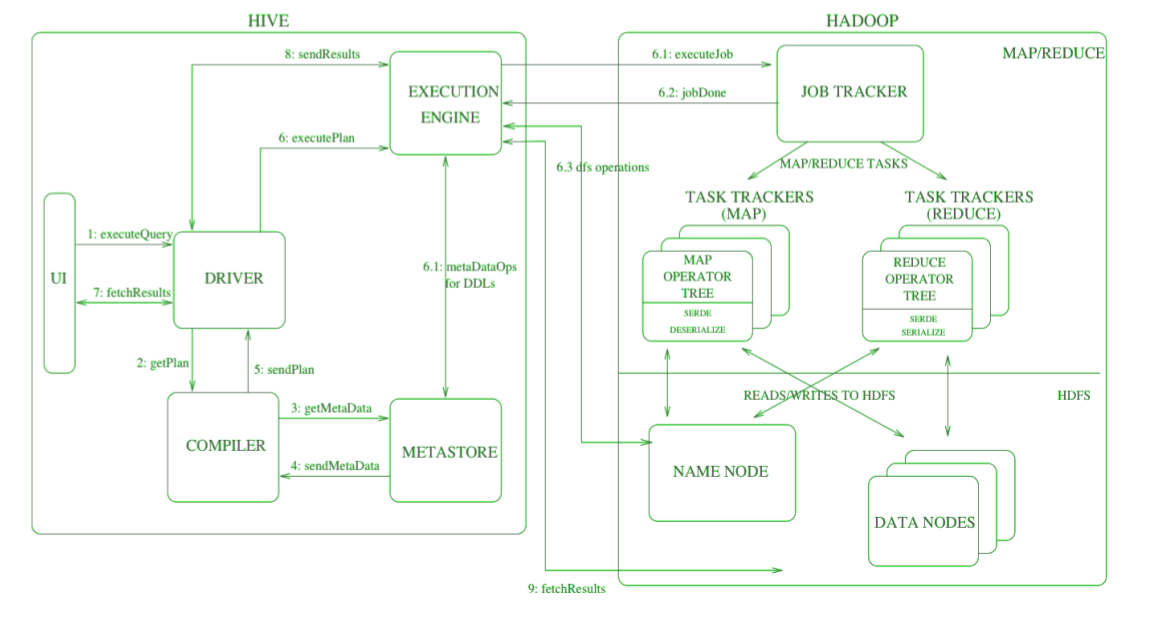
In this, the compiler transfers the metadata request to any database and the compiler gets the necessary metadata from the metastore.

Step-4: Send Metadata –

Metastore transfers metadata as an acknowledgement to the compiler.

Step-5: Send Plan –

Compiler communicating with driver with the execution plan made by the compiler to execute the query.



Step-6: Execute Plan –

Execute plan is sent to the execution engine by the driver.

Execute Job

Job Done

Dfs operation (Metadata Operation)

Step-7: Fetch Results –

Fetching results from the driver to the user interface (UI).

Step-8: Send Results –

Result is transferred to the execution engine from the driver. Sending results to Execution engine. When the result is retrieved from data nodes to the execution engine, it returns the result to the driver and to user interface (UI).

**Need for HBase**

Apache Hadoop has gained popularity in the big data space for storing, managing and processing big data as it can handle high volume of multi-structured data. However, Hadoop cannot handle high velocity of random writes and reads and also cannot change a file without completely rewriting it. HBase is a NoSQL, column oriented database built on top of hadoop to overcome the drawbacks of HDFS as it allows fast random writes and reads in an optimized way. Also, with exponentially growing data, relational databases cannot handle the variety of data to render better performance. HBase provides scalability and partitioning for efficient storage and retrieval.

**HBase –Understanding the Basics**

HBase is a data model similar to Google’s big table that is designed to provide random access to high volume of structured or unstructured data. HBase is an important component of the Hadoop ecosystem that leverages the fault tolerance feature of HDFS. HBase provides real-time read or write access to data in HDFS. HBase can be referred to as a data store instead of a database as it misses out on some important features of traditional RDBMs like typed columns, triggers, advanced query languages and secondary indexes.

**HBase Data Model**

HBase data model stores semi-structured data having different data types, varying column size and field size. The layout of HBase data model eases data partitioning and distribution across the cluster. HBase data model consists of several logical components- row key, column family, table name, timestamp, etc. Row Key is used to uniquely identify the rows in HBase tables. Column families in HBase are static whereas the columns, by themselves, are dynamic.

HBase Tables – Logical collection of rows stored in individual partitions known as Regions.

HBase Row – Instance of data in a table.

RowKey -Every entry in an HBase table is identified and indexed by a RowKey.

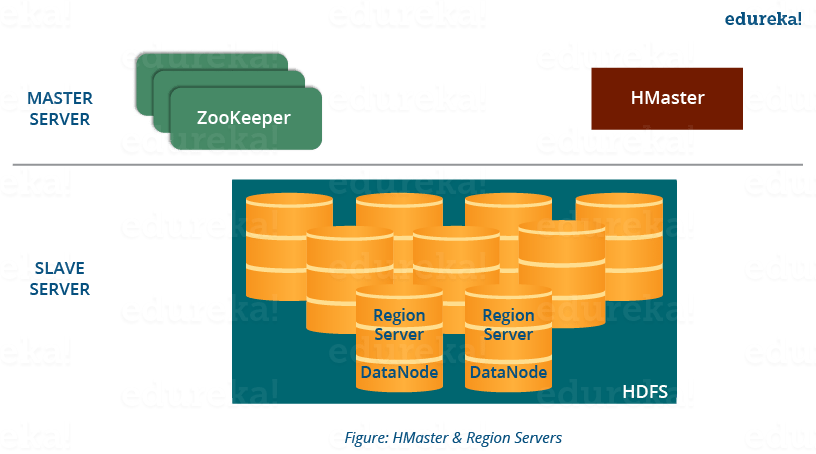
Columns - For every RowKey an unlimited number of attributes can be stored.

Column Family – Data in rows is grouped together as column families and all columns are stored together in a low level storage file known as HFile.

## ****HBase Architecture : Components of HBase Architecture****

HBase has three major components i.e., **HMaster Server**, **HBase Region Server, Regions** and **Zookeeper**.

The below figure explains the hierarchy of the HBase Architecture.



### ****Components of Apache HBase Architecture****

HBase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

**HBase Architecture: Region**

A region contains all the rows between the start key and the end key assigned to that region. HBase tables can be divided into a number of regions in such a way that all the columns of a column family is stored in one region. Each region contains the rows in a sorted order.

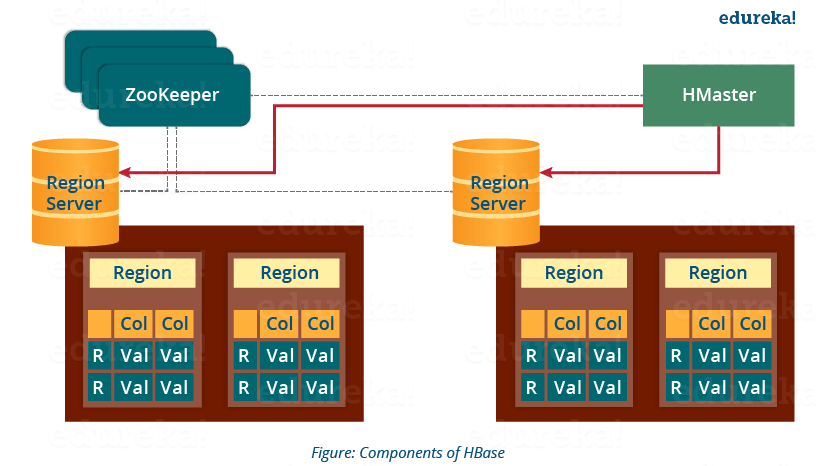
Many regions are assigned to a **Region Server**, which is responsible for handling, managing, executing reads and writes operations on that set of regions.

Main Functionality:

* A table can be divided into a number of regions. A Region is a sorted range of rows storing data between a start key and an end key.
* A Region has a default size of 256MB which can be configured according to the need.
* A Group of regions is served to the clients by a Region Server.
* A Region Server can serve approximately 1000 regions to the client.

## ****HBase Architecture: HMaster****

* HMaster handles a collection of Region Server which resides on DataNode. HBase HMaster performs DDL operations (create and delete tables) and assigns regions to the Region servers as you can see in the above image.
* It coordinates and manages the Region Server (similar as NameNode manages DataNode in HDFS).
* It assigns regions to the Region Servers on startup and re-assigns regions to Region Servers during recovery and load balancing.
* It monitors all the Region Server’s instances in the cluster (with the help of Zookeeper) and performs recovery activities whenever any Region Server is down.
* It provides an interface for creating, deleting and updating tables.



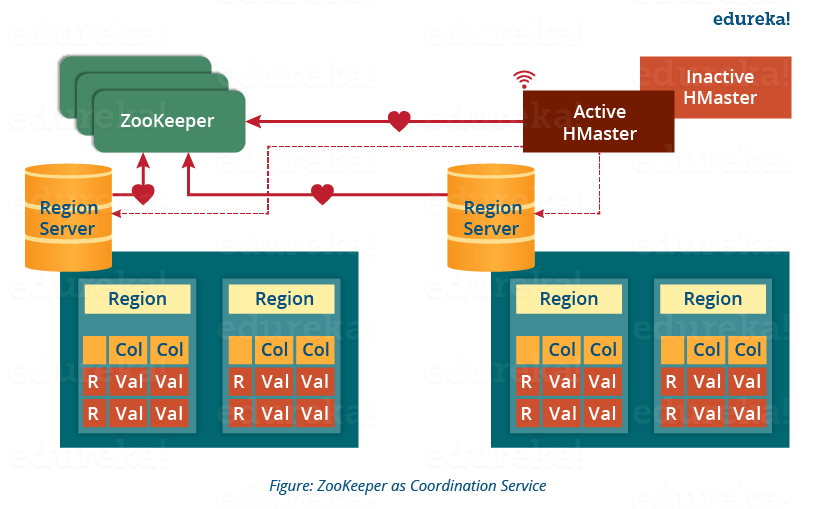
## ****HBase Architecture: ZooKeeper – The Coordinator****

HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning. ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization.

Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.

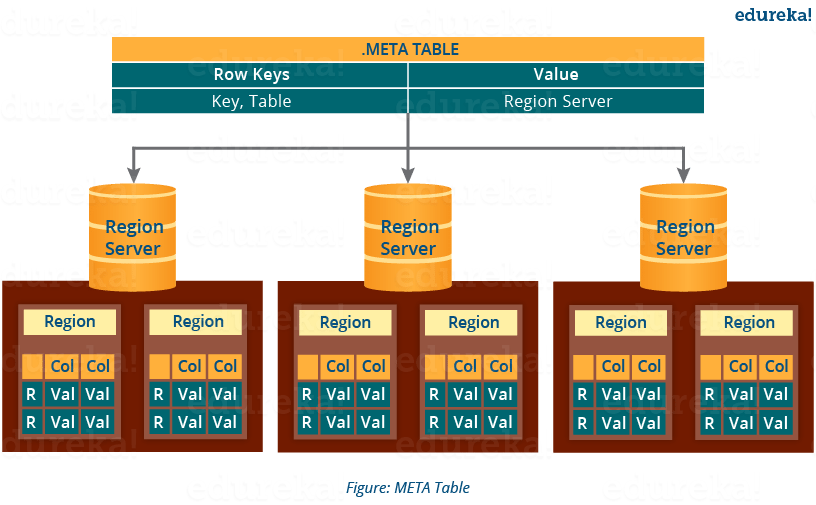
ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

* Establishing client communication with region servers.
* Tracking server failure and network partitions.
* Maintain Configuration Information
* Provides ephemeral nodes, which represent different region servers.



* Zookeeper acts like a coordinator inside HBase distributed environment. It helps in maintaining server state inside the cluster by communicating through sessions.
* Every Region Server along with HMaster Server sends continuous heartbeat at regular interval to Zookeeper and it checks which server is alive and available as mentioned in above image. It also provides server failure notifications so that, recovery measures can be executed.
* Referring from the above image you can see, there is an inactive server, which acts as a backup for active server. If the active server fails, it comes for the rescue.
* The active HMaster sends heartbeats to the Zookeeper while the inactive HMaster listens for the notification send by active HMaster. If the active HMaster fails to send a heartbeat the session is deleted and the inactive HMaster becomes active.
* While if a Region Server fails to send a heartbeat, the session is expired and all listeners are notified about it. Then HMaster performs suitable recovery actions which we will discuss later in this blog.
* Zookeeper also maintains the .META Server’s path, which helps any client in searching for any region. The Client first has to check with .META Server in which Region Server a region belongs, and it gets the path of that Region Server.

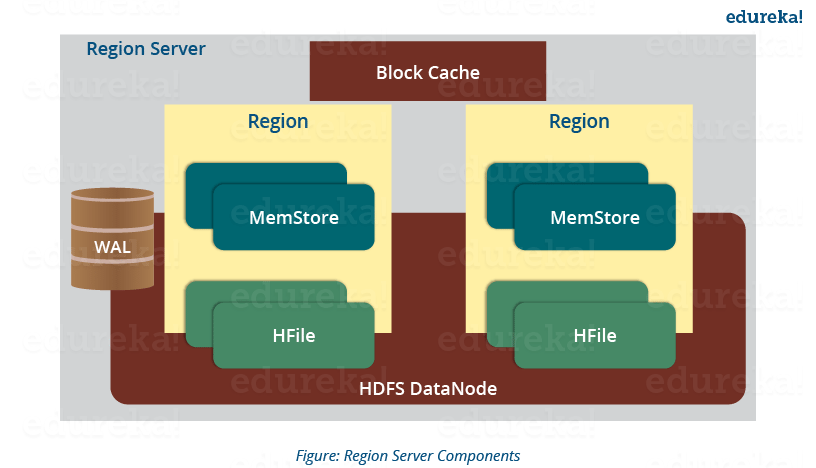
## ****HBase Architecture: Meta Table****



* The META table is a special HBase catalog table. It maintains a list of all the Regions Servers in the HBase storage system, as you can see in the above image.
* Looking at the figure you can see,**.META** file maintains the table in form of keys and values. Key represents the start key of the region and its id whereas the value contains the path of the Region Server.

## ****HBase Architecture: Components of Region Server****

This below image shows the components of a Region Server



A Region Server maintains various regions running on the top of [***HDFS***](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/). Components of a Region Server are:

* **WAL:** Write Ahead Log (WAL) is a file attached to every Region Server inside the distributed environment. The WAL stores the new data that hasn’t been persisted or committed to the permanent storage. It is used in case of failure to recover the data sets.
* **Block Cache:** Block Cache resides in the top of Region Server. It stores the frequently read data in the memory. If the data in BlockCache is least recently used, then that data is removed from BlockCache.
* **MemStore:**It is the write cache. It stores all the incoming data before committing it to the disk or permanent memory. There is one MemStore for each column family in a region. As you can see in the image, there are multiple MemStores for a region because each region contains multiple column families. The data is sorted in lexicographical order before committing it to the disk.
* **HFile:** HFile is stored on HDFS. Thus it stores the actual cells on the disk. MemStore commits the data to HFile when the size of MemStore exceeds.

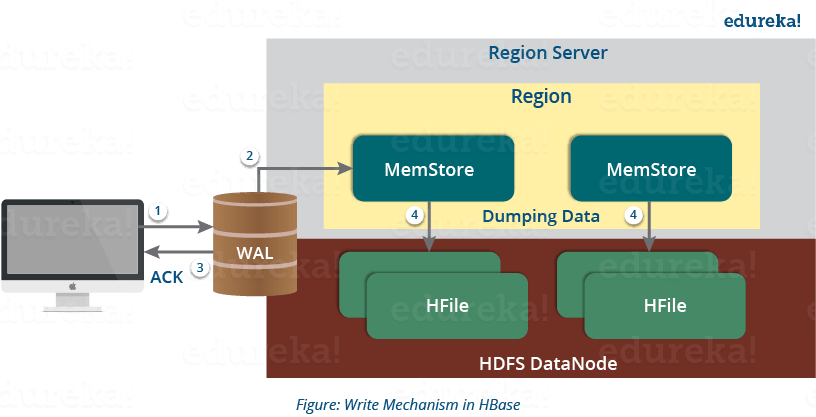
## ****HBase Architecture: How Search Initializes in HBase?****

As you know, Zookeeper stores the META table location. Whenever a client approaches with a read or writes requests to HBase following operation occurs:

1. The client retrieves the location of the META table from the ZooKeeper.
2. The client then requests for the location of the Region Server of corresponding row key from the META table to access it. The client caches this information with the location of the META Table.
3. Then it will get the row location by requesting from the corresponding Region Server.

## ****HBase Architecture: HBase Write Mechanism****

This below image explains the write mechanism in HBase.



The write mechanism goes through the following process sequentially (refer to the above image):

Step 1: Whenever the client has a write request, the client writes the data to the WAL (Write Ahead Log).

* The edits are then appended at the end of the WAL file.
* This WAL file is maintained in every Region Server and Region Server uses it to recover data which is not committed to the disk.

Step 2: Once data is written to the WAL, then it is copied to the MemStore.

Step 3: Once the data is placed in MemStore, then the client receives the acknowledgment.

Step 4: When the MemStore reaches the threshold, it dumps or commits the data into a HFile.

## ****HBase Write Mechanism- MemStore****

* The MemStore always updates the data stored in it, in a lexicographical order (sequentially in a dictionary manner) as sorted KeyValues. There is one MemStore for each column family, and thus the updates are stored in a sorted manner for each column family.
* When the MemStore reaches the threshold, it dumps all the data into a new HFile in a sorted manner. This HFile is stored in HDFS. HBase contains multiple HFiles for each Column Family.
* Over time, the number of HFile grows as MemStore dumps the data.
* MemStore also saves the last written sequence number, so Master Server and MemStore both knows, that what is committed so far and where to start from. When region starts up, the last sequence number is read, and from that number, new edits start.

As I discussed several times, that HFile is the main persistent storage in an HBase architecture. At last, all the data is committed to HFile which is the permanent storage of HBase. Hence, let us look at the properties of HFile which makes it faster for search while reading and writing.

## ****HBase Architecture: HBase Write Mechanism- HFile****

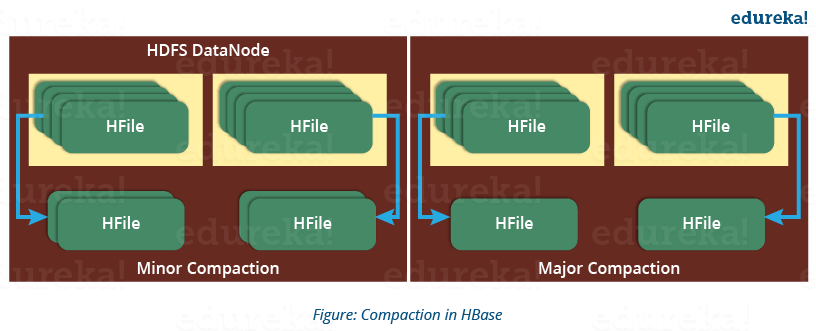
* The writes are placed sequentially on the disk. Therefore, the movement of the disk’s read-write head is very less. This makes write and search mechanism very fast.
* The HFile indexes are loaded in memory whenever an HFile is opened. This helps in finding a record in a single seek.
* The trailer is a pointer which points to the HFile’s meta block . It is written at the end of the committed file. It contains information about timestamp and bloom filters.
* Bloom Filter helps in searching key value pairs, it skips the file which does not contain the required rowkey. Timestamp also helps in searching a version of the file, it helps in skipping the data.

## ****HBase Architecture: Read Mechanism****

As discussed in our search mechanism, first the client retrieves the location of the Region Server from .META Server if the client does not have it in its cache memory. Then it goes through the sequential steps as follows:

* For reading the data, the scanner first looks for the Row cell in Block cache. Here all the recently read key value pairs are stored.
* If Scanner fails to find the required result, it moves to the MemStore, as we know this is the write cache memory. There, it searches for the most recently written files, which has not been dumped yet in HFile.
* At last, it will use bloom filters and block cache to load the data from HFile..

## ****HBase Architecture: Compaction****



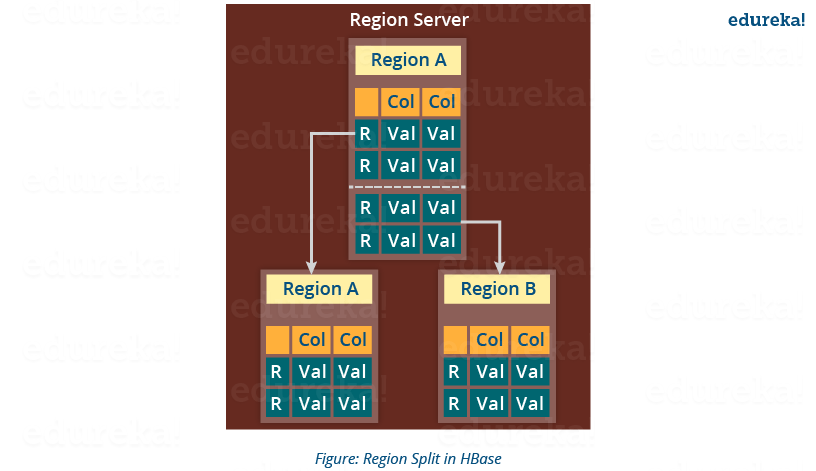
**HBase**combines HFiles to reduce the storage and reduce the number of disk seeks needed for a read. This process is called **compaction**. Compaction chooses some HFiles from a region and combines them. There are two types of compaction as you can see in the above image.

1. **Minor Compaction**: HBase automatically picks smaller HFiles and recommits them to bigger HFiles as shown in the above image. This is called Minor Compaction. It performs merge sort for committing smaller HFiles to bigger HFiles. This helps in storage space optimization.
2. **Major Compaction:** As illustrated in the above image, in Major compaction, HBase merges and recommits the smaller HFiles of a region to a new HFile. In this process, the same column families are placed together in the new HFile. It drops deleted and expired cell in this process. It increases read performance.

But during this process, input-output disks and network traffic might get congested. This is known as **write amplification**. So, it is generally scheduled during low peak load timings.

## ****HBase Architecture: Region Split****

The below figure illustrates the Region Split mechanism.



Whenever a region becomes large, it is divided into two child regions, as shown in the above figure. Each region represents exactly a half of the parent region. Then this split is reported to the HMaster. This is handled by the same Region Server until the HMaster allocates them to a new Region Server for load balancing.

Moving down the line, last but the not least, I will explain you how does HBase recover data after a failure. As we know that **Failure Recovery** is a very important feature of HBase, thus let us know how HBase recovers data after a failure.

## ****HBase Architecture: HBase Crash and Data Recovery****

* Whenever a Region Server fails, ZooKeeper notifies to the HMaster about the failure.
* Then HMaster distributes and allocates the regions of crashed Region Server to many active Region Servers. To recover the data of the MemStore of the failed Region Server, the HMaster distributes the WAL to all the Region Servers.
* Each Region Server re-executes the WAL to build the MemStore for that failed region’s column family.
* The data is written in chronological order (in a timely order) in WAL. Therefore, Re-executing that WAL means making all the change that were made and stored in the MemStore file.
* So, after all the Region Servers executes the WAL, the MemStore data for all column family is recovered.

**Hbase commands**

In Hbase, general commands are categorized into following commands

* **Status**
* **Version**
* **Table\_help ( scan, drop, get, put, disable, etc.)**
* **Whoami**

**Tables Managements commands**

These commands will allow programmers to create tables and table schemas with rows and column families.

The following are Table Management commands

* Create
* List
* Describe
* Disable
* Disable\_all
* Enable
* Enable\_all
* Drop
* Drop\_all
* Show\_filters
* Alter
* Alter\_status

**Data manipulation commands**

These commands will work on the table related to data manipulations such as putting data into a table, retrieving data from a table and deleting schema, etc.

The commands come under these are

* Count
* Put
* Get
* Delete
* Delete all
* Truncate
* Scan

**Cluster Replication Commands**

* These commands work on cluster set up mode of HBase.
* For adding and removing peers to cluster and to start and stop replication these commands are used in general.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Command** | **Functionality** |
| **1** | add\_peer | Add peers to cluster to replicate  hbase> add\_peer '3', zk1,zk2,zk3:2182:/hbase-prod |
| **2** | remove\_peer | Stops the defined replication stream.  Deletes all the metadata information about the peer  hbase> remove\_peer '1' |
| **3** | start\_replication | Restarts all the replication features  hbase> start\_replication |
| **4** | stop\_replication | Stops all the replication features  hbase>stop\_replication |

**Sqoop**

The traditional application management system, that is, the interaction of applications with relational database using RDBMS, is one of the sources that generate Big Data. Such Big Data, generated by RDBMS, is stored in Relational **Database Servers** in the relational database structure. Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. It is provided by the Apache Software Foundation. Apache Sqoop is an effective hadoop tool used for importing data from RDBMS’s like MySQL, Oracle, etc. into HBase, Hive or HDFS. Sqoop hadoop can also be used for exporting data from HDFS into RDBMS. Apache Sqoop is a command line interpreter i.e. the Sqoop commands are executed one at a time by the interpreter.

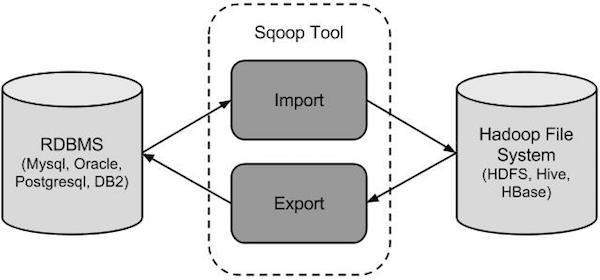
### ****Features of Apache Sqoop****

* Apache Sqoop supports bulk import i.e. it can import the complete database or individual tables into HDFS. The files will be stored in the HDFS file system and the data in built-in directories.
* Sqoop parallelizes data transfer for optimal system utilization and fast performance.
* Apache Sqoop provides direct input i.e. it can map relational databases and import directly into HBase and Hive.
* Sqoop makes data analysis efficient.
* Sqoop helps in mitigating the excessive loads to external systems.
* Sqoop provides data interaction programmatically by generating Java classes.

### ****Working Apache Sqoop****

Sqoop is an effective hadoop tool for non-programmers which functions by looking at the databases that need to be imported and choosing a relevant import function for the source data. Once the input is recognized by Sqoop hadoop, the metadata for the table is read and a class definition is created for the input requirements. Hadoop Sqoop can be forced to function selectively by just getting the columns needed before input instead of importing the entire input and looking for the data in it. This saves considerable amount of time. In reality, the import from the database to HDFS is accomplished by a MapReduce job that is created in the background by Apache Sqoop.

The following image describes the workflow of Sqoop.



Sqoop Import

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

## ****Flume in Hadoop****

Apache Flume is service designed for streaming logs into Hadoop environment. Flume is a distributed and reliable service for collecting and aggregating huge amounts of log data. With a simple and easy to use architecture based on streaming data flows, it also has tunable reliability mechanisms and several recovery and failover mechanisms.

### ****Need for Flume****

Logs are usually a source of stress and argument in most of the big data companies. Logs are one of the most painful resources to manage for the operations team as they take up huge amount of space. Logs are rarely present at places on the disk where someone in the company can make effective use of them or hadoop developers can access them. Many big data companies wind up building tools and processes to collect logs from application servers, transfer them to some repository so that they can control the lifecycle without consuming unnecessary disk space.

This frustrates developers as the logs are often not present at the location where they can view them easily, they have limited number of tools available for processing logs and have confined capabilities in intelligently managing the lifecycle. Apache Flume is designed to address the difficulties of both operations group and developers by providing them an easy to use tool that can push logs from bunch of applications servers to various repositories via a highly configurable agent.

### ****Flume working****

Flume has a simple event driven pipeline architecture with 3 important roles-Source, Channel and Sink.

* Source defines where the data is coming from, for instance a message queue or a file.
* Sinks defined the destination of the data pipelined from various sources.
* Channels are pipes which establish connect between sources and sinks.

Apache flume works on two important concepts-

1. The master acts like a reliable configuration service which is used by nodes for retrieving their configuration.
2. If the configuration for a particular node changes on the master then it will dynamically be updated by the master.

Node is generally an event pipe in Hadoop Flume which reads from the source and writes to the Sink. The characteristics and role of a flume node is determine by the behaviour of source and sinks. Apache Flume is built with several source and sink options but if none of them fits in your requirements then developers can write their own. A flume node can also be configured with the help of a sink decorator which can interpret the event and transforms it as it passes through. With all these basic primitives, developers can create different topologies to collect data on any application server and direct it to any log repository.

### ****Features of Apache Flume****

* Flume is a flexible tool as it allows to scale in environments with as low as five machines to as high as several thousands of machines.
* Apache Flume provides high throughput and low latency.
* Apache Flume has a declarative configuration but provides ease of extensibility.
* Flume in Hadoop is fault tolerant, linearly scalable and stream oriented.

## ****Difference between Sqoop and Flume****

* Apache Sqoop and Apache Flume work with various kinds of data sources. Flume functions well in streaming data sources which are generated continuously in hadoop environment such as log files from multiple servers whereas Apache Sqoop is designed to work well with any kind of relational database system that has JDBC connectivity. Sqoop can also import data from NoSQL databases like MongoDB or Cassandra and also allows direct data transfer or Hive or HDFS. For transferring data to Hive using Apache Sqoop tool, a table has to be created for which the schema is taken from the database itself.
* In Apache Flume data loading is event driven whereas in Apache Sqoop data load is not driven by events.
* Flume is a better choice when moving bulk streaming data from various sources like JMS or Spooling directory whereas Sqoop is an ideal fit if the data is sitting in databases like Teradata, Oracle, MySQL Server, Postgres or any other JDBC compatible database then it is best to use Apache Sqoop.
* In Apache Flume, data flows to HDFS through multiple channels whereas in Apache Sqoop HDFS is the destination for importing data.
* Apache Flume has agent based architecture i.e. the code written in flume is known as agent which is responsible for fetching data whereas in Apache Sqoop the architecture is based on connectors. The connectors in Sqoop know how to connect with the various data sources and fetch data accordingly.
* Lastly, Sqoop and Flume cannot be used achieve the same tasks as they are developed specifically to serve different purposes. Apache Flume agents are designed to fetch streaming data like tweets from Twitter or log file from the web server whereas Sqoop connectors are designed to work only with structured data sources and fetch data from them.
* Apache Sqoop is mainly used for parallel data transfers, for data imports as it copies data quickly where Apache Flume is used for collecting and aggregating data because of its distributed, reliable nature and highly available backup routes.