HBase

In this session, you will learn about:

- **Introduction to HBase**
- **Architecture**
- **Installation**
- **HBase shell**
- **Common operations**
- **Java API**
- **Security**

HBase

- HBase is an open source, multidimensional, distributed, scalable and a NoSQL database written in Java.
- HBase is not a relational database.
- HBase runs on top of HDFS (Hadoop Distributed File System) and provides BigTable like capabilities to Hadoop.
- It is designed to provide a fault tolerant way of storing large collection of sparse data sets.
- HBase achieves high throughput and low latency by providing faster Read/Write Access on huge data sets.
- HBase is suitable for random access of data.
- It is column oriented.
- Provides compression, in-memory operations and Bloom filters (data structure which tells whether a value is present in a set or not).
- Apache HBase is modelled after Google's BigTable, which is used to collect data and serve request for various Google services like Maps, Finance, Earth etc.

Row-oriented vs column-oriented Databases:

Row-oriented databases store table records in a sequence of rows. Whereas column-oriented databases store table records in a sequence of columns, i.e. the entries in a column are stored in contiguous locations on disks.

To better understand it, let us take an example and consider the table below.

Customer ID	Name	Address	Product ID	Product Name
1	Paul Walker	US	231	Gallardo
2	Vin Diesel	Brazil	520	Mustang

In row-oriented databases data is stored on the basis of rows or tuples.

While the column-oriented databases store this data as:

1,2, Paul Walker, Vin Diesel, US, Brazil, 231, 520, Gallardo, Mustang

In a column-oriented databases, all the column values are stored together like first column values will be stored together, then the second column values will be stored together and data in other columns are stored in a similar manner. Columns are logically grouped into column families which can be either created during schema definition or at runtime.

Column based storage provides faster access of data.

Suitable for OLTP workloads.

Architecture

HBase data model

Architecture components

Write mechanism

Read mechanism

Performance optimization

Data model

- Tables: Data is stored in a table format in HBase. But here tables are in column-oriented format.
- Row Key: Row keys are used to search records which make searches fast.
- Column Families: Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.

- Column Qualifiers: Each column's name is known as its column qualifier.
- Cell: Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
- Timestamp: Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

Image courtesy: Edureka

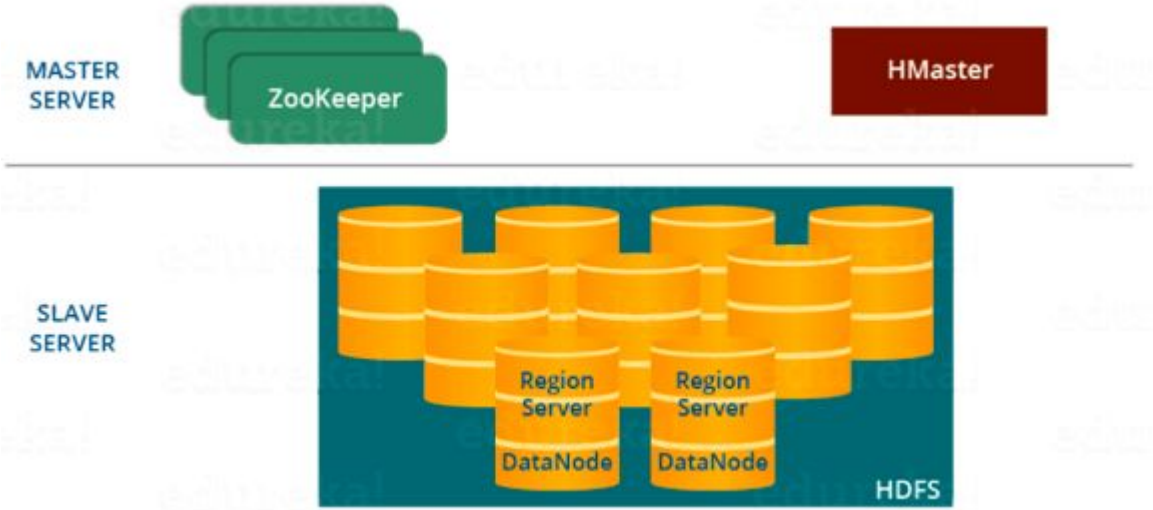
Architecture components

HMaster Server

HBase Region Server

Regions

Zookeeper

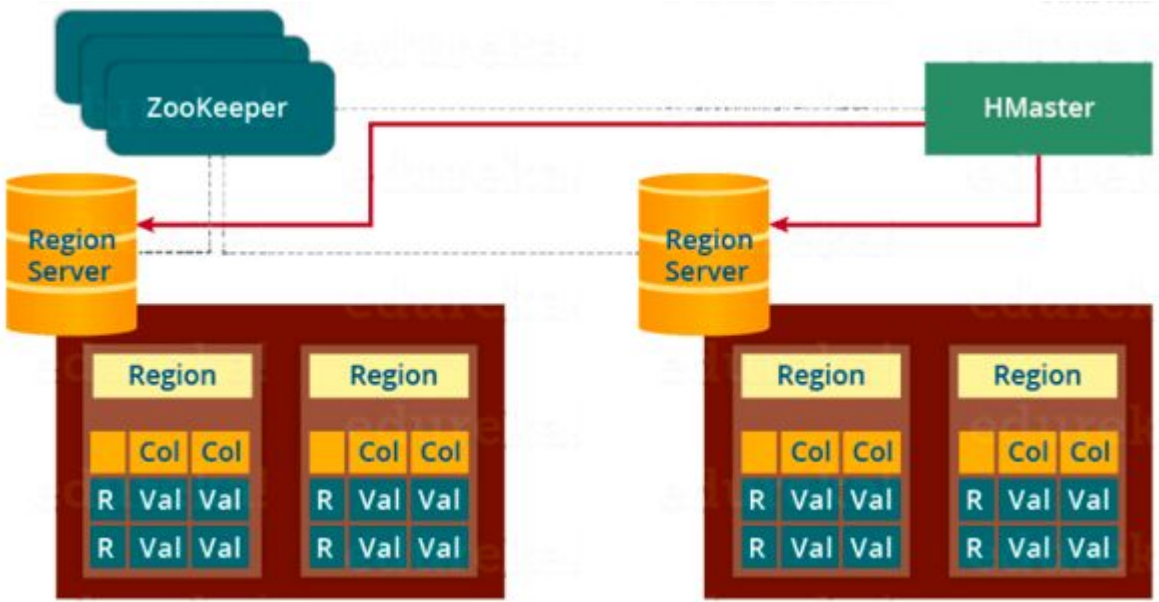


Architecture components - Region

- 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.
- A region contains all the rows between the start key and the end key assigned to that 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 write operations on that set of regions.
- 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.

Architecture components - Master

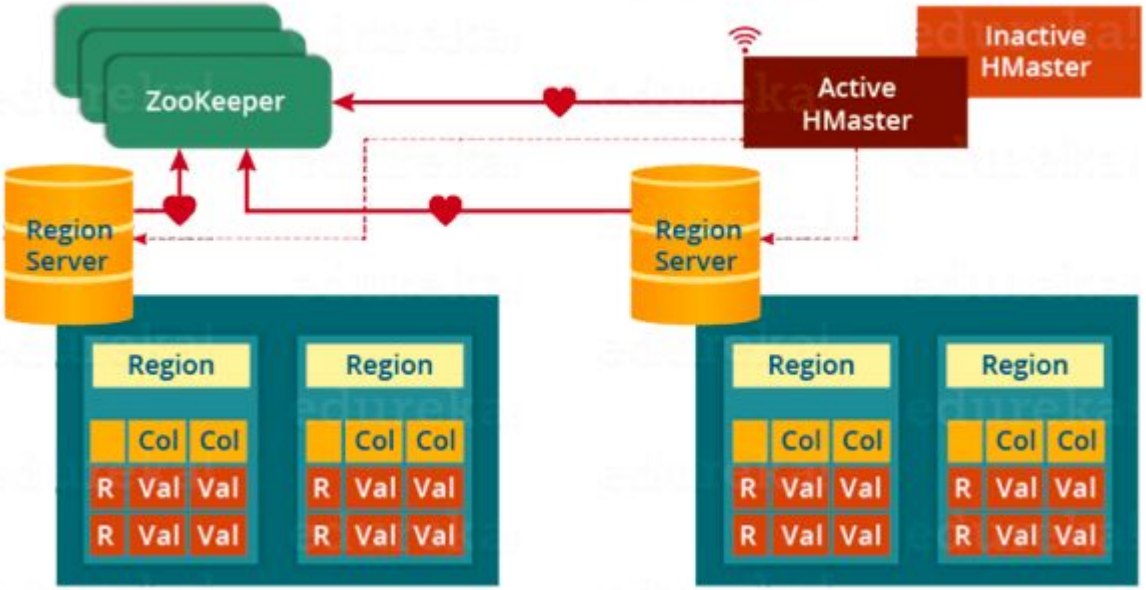
- HBase HMaster performs DDL operations (create and delete tables) and assigns regions to the Region servers
- It coordinates and manages the Region Server (similar as NameNode manages DataNode in HDFS



- 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.

Architecture components - Zookeeper

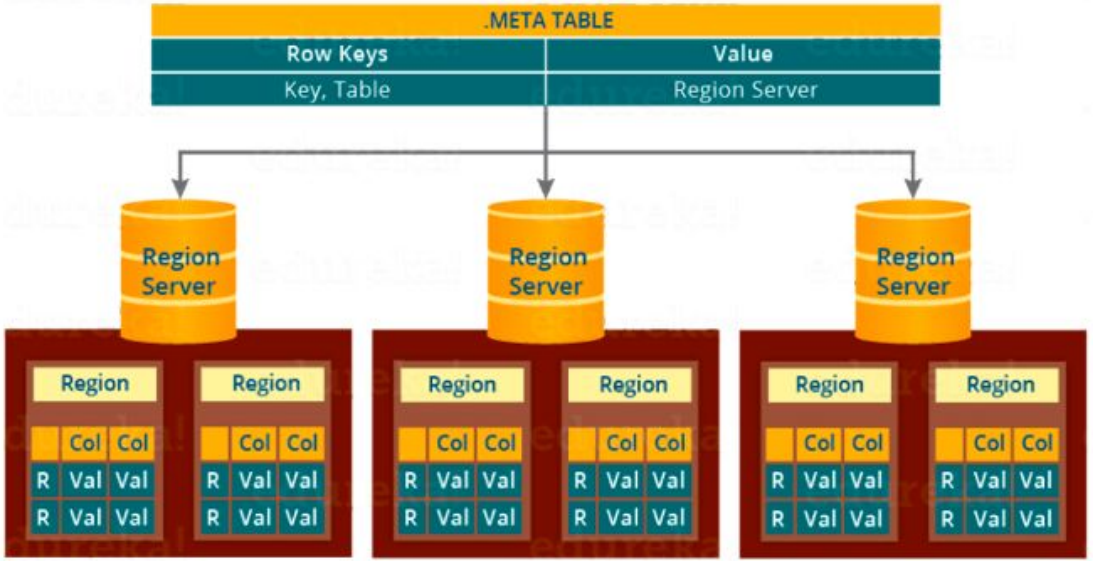
- Zookeeper acts like a coordinator inside HBase distributed environment. It helps in maintaining server state inside the cluster by communicating through sessions.
- Changes standby to active server after failure.



- Every Region Server along with HMaster Server sends continuous heartbeat at regular interval to Zookeeper and it checks which server is alive. It also provides server failure notifications so that, recovery measures can be executed
- 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.
- 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.

Architecture components - .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.
- .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.



Architecture components - Region server components

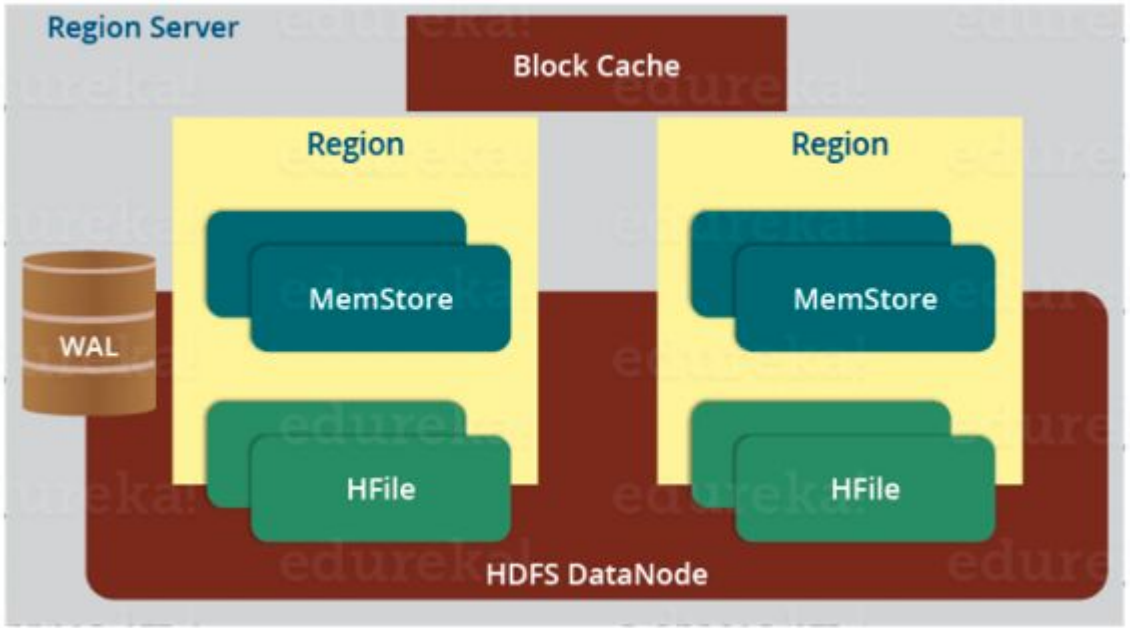


Image courtesy: Edureka

Architecture components - Region server components

A Region Server maintains various regions running on the top of HDFS. 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 Block Cache 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. 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. It stores the actual cells on the disk. MemStore commits the data to HFile when the size of MemStore exceeds.

Architecture components - Search in HBase

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.
4. For future references, the client uses its cache to retrieve the location of .META table and previously read row key's Region Server.
5. Then the client will not refer to the META table, until and unless there is a miss because the region is shifted or moved. Then it will again request to the META server and update the cache.

Architecture components - Writes in HBase

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.

Architecture components - Writes - 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.

Architecture components - Writes - 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.

Architecture - Reads

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.

Compaction & Region split

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:

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.

During this process, input-output disks and network traffic might get congested. This is known as write amplification.

Whenever a region becomes large, it is divided into two child regions. 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.

Crash and 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 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 shell commands

hbase shell

Information about logged in user:

whoami

Information about hbase version:

version

system status variants:

status 'simple'

status 'detailed'

status 'summary'

Hbase shell commands

Create table:

Syntax: create <tablename>, <columnfamilyname>

create 'testtab', 'cf1'

In order to check whether the table 'education' is created or not, we have to use the "list" command as mentioned below.

list

Syntax:list

Describe

Syntax:describe <table name>

describe 'testtab'

Hbase shell commands

Before a table can be dropped, it needs to be disabled:

Syntax: disable <tablename>

Enable a table

Syntax: enable <tablename>

Show filters available

Syntax: show_filters

This command displays all the filters present in HBase like ColumnPrefix Filter, TimestampsFilter, PageFilter, FamilyFilter, etc.

Hbase shell commands

drop a table

Syntax:drop <table name>

alter table

Syntax: alter <tablename>, NAME=><column familyname>, ...

Syntax: alter <tablename>, {NAME=><column familyname>, ...}, {NAME=><column familyname>, ...}

Hbase shell commands

show count

Syntax: count <'tablename'>, CACHE =>1000

The command will retrieve the count of a number of rows in a table. The value returned by this one is the number of rows.

Current count is shown per every 1000 rows by default. Count interval may be optionally specified.

Hbase shell commands

Put data in a table

Syntax: put <'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>

This command is used for following things:

1. It will put a cell 'value' at defined or specified table or row or column.

2. It will optionally coordinate time stamp.

Hbase shell commands

Get data from table

Syntax: get <'tablename'>, <'rowname'>, {< Additional parameters>}

Here <Additional Parameters> include TIMERANGE, TIMESTAMP, VERSIONS and FILTERS.

By using this command, you will get a row or cell contents present in the table. In addition to that you can also add additional parameters to it like TIMESTAMP, TIMERANGE,VERSIONS, FILTERS, etc. to get a particular row or cell content.

Delete from a table

Syntax:delete <'tablename'>,<'row name'>,<'column name'>

This command will delete cell value at defined table of row or column. Delete must and should match the deleted cells coordinates exactly. When scanning, delete cell suppresses older versions of values.

Hbase shell commands

Truncate

Syntax: truncate <tablename>

After truncate of an hbase table, the schema will present but not the records. This command performs 3 functions; those are listed below:

Disables table if it already presents

Drops table if it already presents

Recreates the mentioned table

Hbase shell commands

Scan a table

Syntax: scan <'tablename'>, {Optional parameters}

This command scans entire table and displays the table contents. We can pass several optional specifications to this scan command to get more information about the tables present in the system.Scanner specifications may include one or more of the following attributes. These are TIMERANGE, FILTER, TIMESTAMP, LIMIT, MAXLENGTH, COLUMNS, CACHE, STARTROW and STOPROW.

Hbase shell commands

Examples:-

scan '.META.', {COLUMNS => 'info:regioninfo'}

It display all the meta data information related to columns that are present in the tables in HBase

scan 'testtab', {COLUMNS => ['cf1', 'cf2'], LIMIT => 10, STARTROW => 'xyz'}

It display contents of table testtab with their column families c1 and c2 limiting the values to 10

scan 'testtab', {COLUMNS => 'c1', TIMERANGE => [1303668804, 1303668904]}

It display contents of testtab with its column name c1 with the values present in between the mentioned time range attribute value.

Hbase Java API access

Sample code discussion

Refer to: More Java API references doc provided

Refer to https://hbase.apache.org/apidocs/index.html

Hbase Delete

When a Delete command is issued through the HBase client, no data is actually deleted. Instead a tombstone marker is set, making the deleted cells effectively invisible. User Scans and Gets automatically filter deleted cells until they get removed. HBase periodically removes deleted cells during compactions.

There are three types of tombstone markers:

1. version delete marker: Marks a single version of a column for deletion
2. column delete marker: Marks all versions of a column for deletion
3. family delete marker: Marks all versions of all columns for a column family for deletion

Hbase security - basics

With the default Apache HBase configuration, everyone is allowed to read from and write to all tables available in the system.

HBase can be configured to provide *User Authentication*, which ensures that only authorized users can communicate with HBase.

The authorization system is implemented at the RPC level, and is based on the Simple Authentication and Security Layer (SASL), which supports (among other authentication mechanisms) Kerberos.

SASL allows authentication, encryption negotiation and/or message integrity verification on a per connection basis (“hbase.rpc.protection” configuration property).

The next step after enabling *User Authentication* is to give an admin the ability to define a series of User Authorization rules that allow or deny particular actions.

The Authorization system, also known as the Access Controller Coprocessor or Access Control List (ACL) gives the ability to define authorization policy (Read/Write/Create/Admin), with table/family/qualifier granularity, for a specified user.

Reference: Official Apache Hbase documentation

Hbase security - kerberos

- Kerberos is a networked authentication protocol.
- It is designed to provide strong authentication for client/server applications by using key cryptography.
- The Kerberos protocol uses strong cryptography (AES, 3DES, …) so that a client can prove its identity to a server (and vice versa) across an insecure network connection.
- After a client and server have used Kerberos to prove their identities, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.
- At a high level, to access a service using Kerberos, each client must follow three steps:
    - Kerberos Authentication: The client authenticates itself to the Kerberos Authentication Server and receive a Ticket Granting Ticket (TGT).
    - Kerberos Authorization: The client request a service ticket from the Ticket Granting Server, which issues a ticket and a session key if the client TGT sent with the request is valid.
    - Service Request: The client uses the service ticket to authenticate itself to the server that is providing the service the client is using (e.g. HDFS, HBase, …)

Hbase security - https

A default HBase install uses insecure HTTP connections for Web UIs for the master and region servers.

To enable secure HTTPS connections, set hbase.ssl.enabled to true in hbase-site.xml.

This does not change the port used by the Web UI. To change the port for the web UI for a given HBase component, configure that port's setting in hbase-site.xml. These settings are:

hbase.master.info.port

hbase.regionserver.info.port

If you enable HTTPS, clients should:

- avoid using the non-secure HTTP connection
- connect to HBase using the https:// URL. Clients using the http:// URL will receive an HTTP response of 200, but will not receive any data

Hbase security - simple mode

Simple user access is not a secure method of operating HBase.

It can be used to mimic the Access Control using on a development system without having to set up Kerberos. This is done using SASL.

This method is not used to prevent malicious or hacking attempts.

Refer document Additional Details.txt for configuration details.

Hbase security - securing ZooKeeper access

- Secure HBase requires secure ZooKeeper and HDFS so that users cannot access and/or modify the metadata and data from under HBase.
- ZooKeeper has a pluggable authentication mechanism to enable access from clients using different methods.
- ZooKeeper allows authenticated and unauthenticated clients at the same time.
- HBase daemons authenticate to ZooKeeper via SASL and kerberos.
- HBase sets up the znode ACLs so that only the HBase user and the configured hbase superuser (hbase.superuser) can access and modify the data.
- In cases where ZooKeeper is used for service discovery or sharing state with the client, the znodes created by HBase will also allow anyone (regardless of authentication) to read these znodes (clusterId, master address, meta location, etc), but only the HBase user can modify them.

Hbase security - securing File system/HDFS access

- All of the data under management is kept under the root directory in the file system (`hbase.rootdir`).
- Access to the data and WAL files in the filesystem should be restricted.
- HBase assumes the filesystem used (HDFS or other) enforces permissions hierarchically. If sufficient protection from the file system (both authorization and authentication) is not provided, HBase level authorization control (ACLs, visibility labels, etc) is meaningless since the user can always access the data from the file system.
- HBase enforces the posix-like permissions 700 (`rwx------`) to its root directory. The default setting can be changed by configuring `hbase.rootdir.perms` in hbase-site.xml. You can, also, manually set the permissions for the root directory if needed. Using HDFS. What would be the command?

Hbase security - Tag and Visibility labels

- A tag is a piece of metadata which is part of a cell, separate from the key, value, and version. Tags are an implementation detail which provides a foundation for other security-related features such as cell-level ACLs and visibility labels.
- Every cell can have zero or more tags. Every tag has a type and the actual tag byte array.
- Visibility labels control can be used to only permit users associated with a given label to access cells with that label.
- For instance, you might label a cell top-secret, and only grant access to that label to the managers group.
- Visibility labels are implemented using Tags.
- If a user’s labels do not match a cell’s label or expression, the user is denied access to the cell.
- A user adds visibility expressions to a cell during a Put operation.

Hbase security - Access Control Labels

HBase has a simpler security model than relational databases, especially in terms of client operations. No distinction is made between an insert (new record) and update (of existing record), for example, as both collapse down into a Put.

HBase access levels are granted independently of each other and allow for different types of operations at a given scope.

- *Read (R)* - can read data at the given scope
- *Write (W)* - can write data at the given scope
- *Execute (X)* - can execute coprocessor endpoints at the given scope
- *Create (C)* - can create tables or drop tables (even those they did not create) at the given scope
- *Admin (A)* - can perform cluster operations such as balancing the cluster or assigning regions at the given scope

Example usage: grant 'user', 'RWXCA', 'TABLE', 'CF', 'CQ'