**Task2**

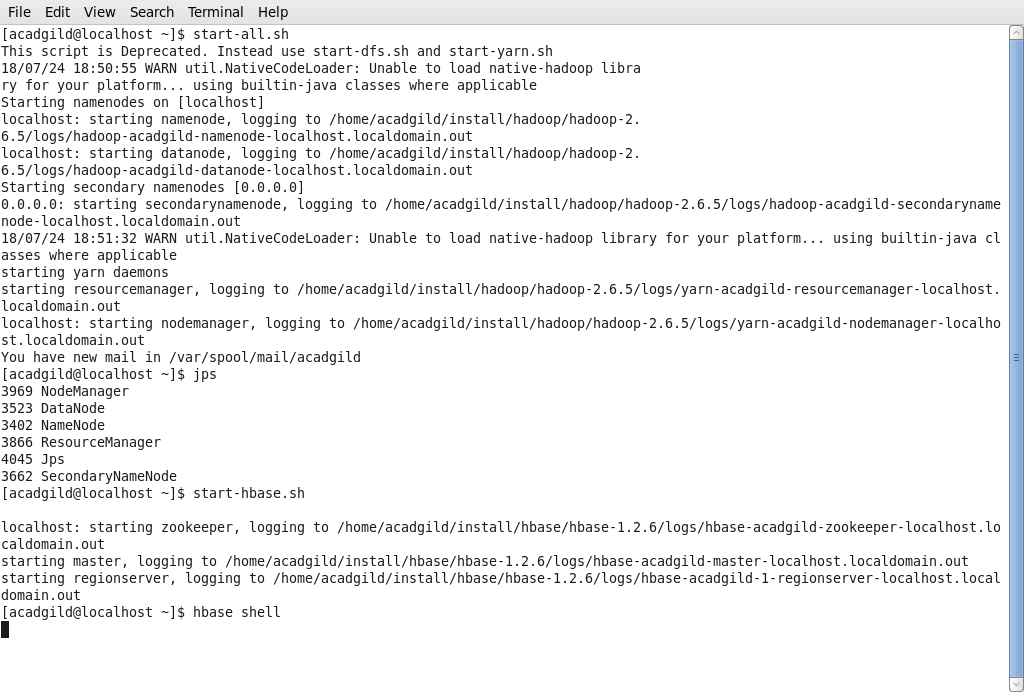
**Execute blog present in below link**

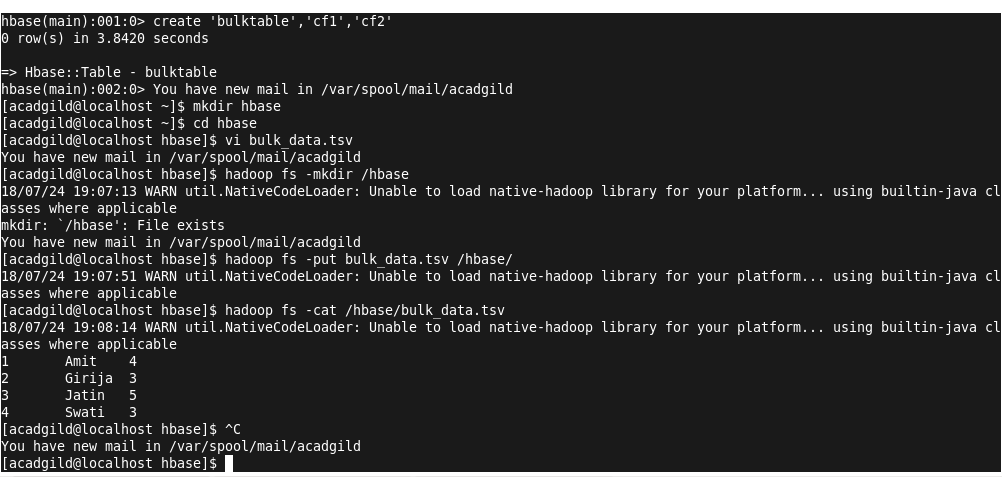
<https://acadgild.com/blog/importtsv-data-from-hdfs-into-hbase/>

Solutions :

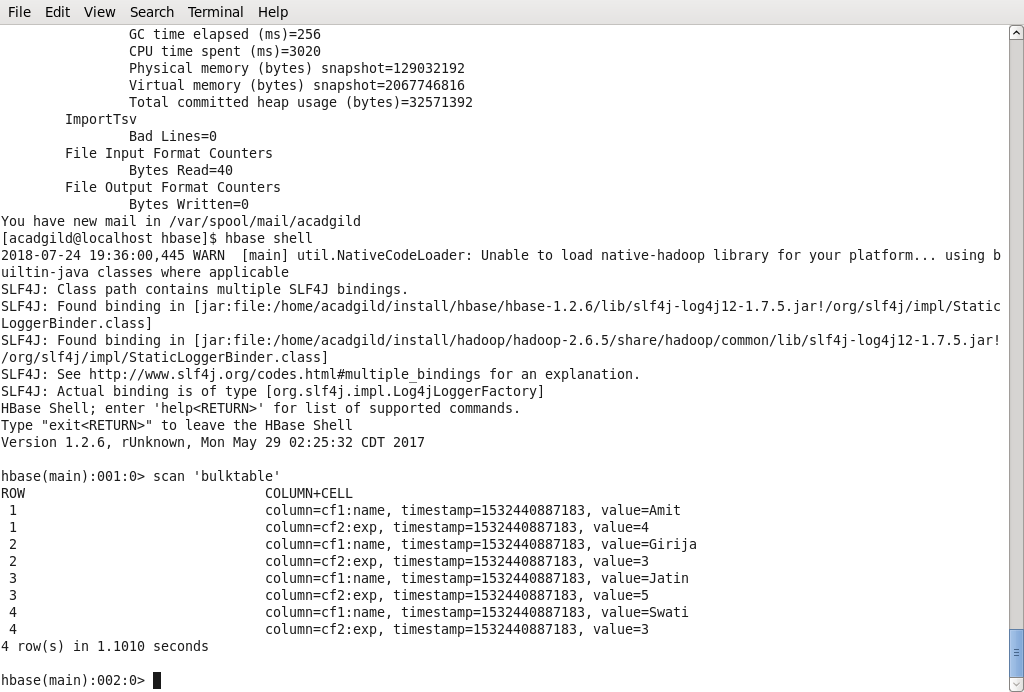
Commands Used :

Start-all.sh  
start-habse.sh  
hbase shell  
create ‘bulktable’, ‘cf1’, ‘cf2’  
  
hbase(main):001:0> create 'bulktable','cf1','cf2'  
0 row(s) in 3.8420 seconds  
  
=> Hbase::Table - bulktable  
hbase(main):002:0> You have new mail in /var/spool/mail/acadgild  
[acadgild@localhost ~]$ mkdir hbase  
[acadgild@localhost ~]$ cd hbase  
[acadgild@localhost hbase]$ vi bulk\_data.tsv  
You have new mail in /var/spool/mail/acadgild  
[acadgild@localhost hbase]$ hadoop fs -mkdir /hbase  
18/07/24 19:07:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
mkdir: `/hbase': File exists  
You have new mail in /var/spool/mail/acadgild  
[acadgild@localhost hbase]$ hadoop fs -put bulk\_data.tsv /hbase/  
18/07/24 19:07:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
[acadgild@localhost hbase]$ hadoop fs -cat /hbase/bulk\_data.tsv  
18/07/24 19:08:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
1    Amit    4  
2    Girija    3  
3    Jatin    5  
4    Swati    3  
[acadgild@localhost hbase]$





[acadgild@localhost hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE\_ROW\_KEY,cf1:name,cf2:exp bulktable /hbase/bulk\_data.tsv

****

**Task 1 :**

**1) NoSQL Databases**

A **NoSQL** or non relational database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relation databases (RDBMS). It encompasses a wide variety of different database technologies that were developed in response to a rise in the volume of data stored about users, objects and products, the frequency in which this data is accessed, and performance and processing needs. Generally, NoSQL databases are structured in a key-value pair, graph database, document-oriented or column-oriented structure.

As an example, consider that you have a blogging application that stores user blogs. Now suppose that you have to incorporate some new features in your application such as users liking these blog posts or commenting on them or liking these comments. With a typical RDBMS implementation, this will need a complete overhaul to your existing database design. However, if you use NoSQL in such scenarios, you can easily modify your data structure to match these agile requirements. With NoSQL you can directly start inserting this new data in your existing structure without creating any new pre-defined columns or pre-defined structure.

Examples for NoSQL DB :

**Document Oriented Databases** – MongoDB, HBase, Cassandra, Amazon SimpleDB, Hypertable, etc.

**Graph Based Databases** – Neo4j, OrientDB, Facebook Open Graph, FlockDB, etc.

**Column Based Databases** – CouchDB, OrientDB, etc.

**Key Value Databases** – Membase, Redis, MemcacheDB, etc.

**2)Types of NoSQL Databases :**

There are 4 basic types of NoSQL databases:

1. **Key-Value Store** – It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
2. **Document-based** **Store- It**stores documents made up of tagged elements. {Example- CouchDB}
3. **Column-based Store-**Each storage block contains data from only one column, {Example- HBase, Cassandra}
4. **Graph-based**-A network database that uses edges and nodes to represent and store data. {Example- Neo4J}

**Key-Value databases**

Key-value stores are the simplest NoSQL data stores to use from an API perspective. The client can either get the value for the key, put a value for a key, or delete a key from the data store. The value is a blob that the data store just stores, without caring or knowing what's inside; it's the responsibility of the application to understand what was stored. Since key-value stores always use primary-key access, they generally have great performance and can be easily scaled.

Some of the popular key-value databases are HBase, Riak, Redis (often referred to as Data Structure server), Memcached and its flavors, Berkeley DB, upscaledb (especially suited for embedded use), Amazon DynamoDB (not open-source), Project Voldemort and Couchbase.

**Column family stores**

 Column-family databases store data in column families as rows that have many columns associated with a row key . Column families are groups of related data that is often accessed together. For a Customer, we would often access their Profile information at the same time, but not their Orders.

Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows.

When a column consists of a map of columns, then we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.

Cassandra is one of the popular column-family databases; there are others, such as HBase, Hypertable, and Amazon DynamoDB. Cassandra can be described as fast and easily scalable with write operations spread across the cluster. The cluster does not have a master node, so any read and write can be handled by any node in the cluster.

**Graph Databases**

Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties. Think of a node as an instance of an object in the application. Relations are known as edges that can have properties. Edges have directional significance; nodes are organized by relationships which allow you to find interesting patterns between the nodes. The organization of the graph lets the data to be stored once and then interpreted in different ways based on relationships.

There are many graph databases available, such as Neo4J, Infinite Graph, OrientDB, or FlockDB (which is a special case: a graph database that only supports single-depth relationships or adjacency lists, where you cannot traverse more than one level deep for relationships).

**Document databases**

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same. Document databases store documents in the value part of the key-value store; think about document databases as key-value stores where the value is examinable. Document databases such as MongoDB provide a rich query language and constructs such as database, indexes etc allowing for easier transition from relational databases.

Some of the popular document databases we have seen are MongoDB, CouchDB , Terrastore, OrientDB, RavenDB, and of course the well-known and often reviled Lotus Notes that uses document storage.

**3) CAP Theorem**

What is CAP Theorem?

CAP describes that before choosing any Database (Including distributed database), Basing on your requirement we have to choose only two properties out of three.

Consistency

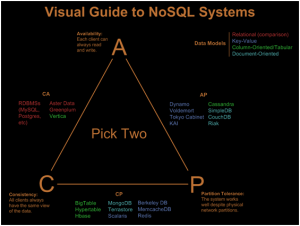
– Whenever you read a record (or data), consistency guaranties that it will give same data how many times you read. Simply we can say that each server returns the right response to each request, thus the system will be always consistent whenever you read or write data into that.

Availability

– Availability simply means that each request eventually receives a response (even if it’s not the latest data or consistent across the system or just a message saying the system isn’t working).

Partition Tolerance

– Partition Tolerance means that the cluster continues to function even if there is a “partition” (communications break) between two nodes (both nodes are up, but can’t communicate).

One property should be scarified among three, so you have to choose combination of CA or CP or AP.

Consistency – Partition Tolerance:

System will give you a consistent data and it will be distributed across the cluster. But it becomes unavailable when a node goes down.

Availability – Partition Tolerance:

System will respond even if nodes are not communicating with each other (nodes are up and running but not communicating). But there is no guaranty that all nodes will have same data.

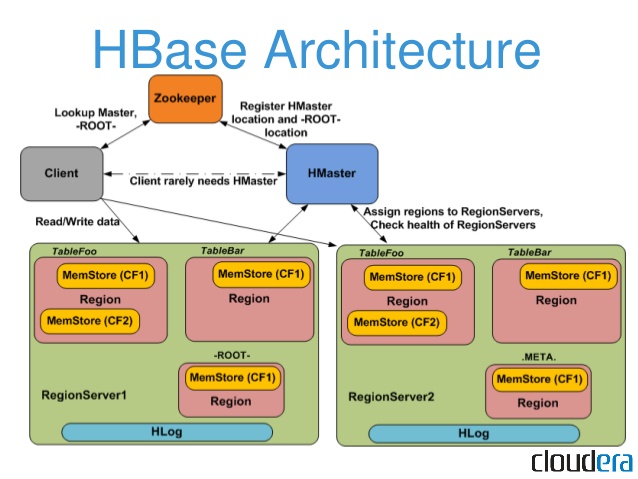
Consistency – Availability:

System will give you a consistent data and you can write/read data to/from any node, but data partitioning will not be sync when develop a partition between nodes (RDBMS systems such as MySQL are of CA combination systems.)

**4)HBASE Architecture**

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.

**Note:** The term ‘store’ is used for regions to explain the storage structure.



HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

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

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

#### ****HMaster****

HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

* Manages and Monitors the Hadoop Cluster
* Performs Administration (Interface for creating, updating and deleting tables.)
* Controlling the failover
* DDL operations are handled by the HMaster
* Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

#### ****Region Server****

These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

* Block Cache – This is the read cache. Most frequently read data is stored in the read cache and whenever the block cache is full, recently used data is evicted.
* MemStore- This is the write cache and stores new data that is not yet written to the disk. Every column family in a region has a MemStore.
* Write Ahead Log (WAL) is a file that stores new data that is not persisted to permanent storage.
* HFile is the actual storage file that stores the rows as sorted key values on a disk.

#### ****Zookeeper****

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.

**5)HBASE vs RDBMS**

1. HBase vs RDBMS

Today, in this article “HBase vs RDBMS: Feature Wise Comparison” we will learn the complete comparison of HBase vs RDBMS, on the basis of several features. Both HDFS and RDBMS are varying concepts of processing, retrieving and storing the data or information. Still, there are some reasons that HBase has lacks comparison to conventional relational databases which are even existed for so long now.

So, let’s begin HBase vs RDBMS

2. Difference Between HBase & RDBMS

i. What is HBase?

An open source and sorted map data built on hadoop is what we call HBase. Basically, it is column-oriented and horizontally scalable. Moreover, it offers APIs enabling development in practically any programming language. Also, it offers random real-time read/write access to data in the Hadoop File System, as it is a part of the Hadoop ecosystem that.

ii. What is RDBMS?

RDBMS refers to Relational Database Management Systems. Basically, systems like SQL, MS SQL Server, IBM DB2, Oracle, MySQL and Microsoft Access are based on RDBMS. Since it is based on the relational model introduced by E.F. Codd so it is called Relational Database Management System (RDBMS).

3. Feature Wise Comparison of HBase vs RDBMS

Below we are discussing the feature wise difference of HBase vs RDBMS, let’s explore this in detail:

i. Database Type

HBase

HBase is the column-oriented database. On defining Column-oriented, each column is a contiguous unit of page.

RDBMS

Whereas, RDBMS is row-oriented that means here each row is a contiguous unit of page.

ii. Schema-type

HBase

Schema of HBase is less restrictive, adding columns on the fly is possible.

RDBMS

However, Schema of RDBMS is more restrictive.

iii. Sparse Tables

HBase

HBase is good with the Sparse table.

RDBMS

Whereas, RDBMS is not optimized for sparse tables.

iv. Scale up/ Scale out

HBase

HBase supports scale out. It means while we need memory processing power and more disk, we need to add new servers to the cluster rather than upgrading the present one.

RDBMS

However, RDBMS supports scale up. That means while we need memory processing power and more disk, we need upgrade same server to a more powerful server, rather than adding new servers.

v. Amount of data

HBase

While here it does not depend on the particular machine but the number of machines.

RDBMS

In RDBMS, on the configuration of the server, amount of data depends.

vi.  Support of

HBase

For HBase, there is no built-in support.

RDBMS

And, RDBMS has ACID support.

vii. Data type

HBase

HBase supports both structured and nonstructural data.

RDBMS

RDBMS is suited for structured data.

viii. Transaction integrity

HBase

In HBase, there is no transaction guaranty.

RDBMS

Whereas, RDBMS mostly guarantees transaction integrity.

ix.  JOINs

HBase

HBase supports JOINs.

RDBMS

RDBMS does not support JOINs.

[adoop Knowledge](https://data-flair.training/blogs/category/quiz/hadoop-quiz/" \t "_blank)

x. Referential integrity

HBase

While it comes to referential integrity, there is no in-built support.

RDBMS

And, RDBMS, supports referential integrity.

4. Features of HBase and RDBMS

Why HBase?

HBase is horizontally scalable.

Integrations with Map/Reduce framework.

Moreover, it is possible to refer HBase as a key-value store or column family-oriented database.

Why RDBMS?

Here, in form of rows and columns, data stores.

By using SQL queries, it also supports virtual tables from where we can retrieve data.

For the purpose of data uniqueness, RDBMS provides a primary key.

Also, it offers referential integrity.