

NoSql Databases

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean “not SQL” or “not only SQL.” As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

Types of Nosql database

There have been various approaches to classify NoSQL databases, each with different categories and subcategories, some of which overlap.

Key-value data stores: Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned *and replicated* across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.

Document stores: Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.

Wide-column stores: Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

Graph stores: A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

Some examples are:

- **Column:** Accumulo, Cassandra, Druid, HBase, Vertica.

- **Document:** Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
- **Key-value:** Aerospike, Apache Ignite, ArangoDB, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, InfinityDB, Memcached, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB, SDBM/Flat File dbm, ZooKeeper
- **Graph:** AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso
- **Multi-model:** Apache Ignite, ArangoDB, Couchbase, FoundationDB, InfinityDB, MarkLogic, OrientDB, Cosmos DB

CAP Theorem

The CAP theorem is a tool used to make system designers aware of the trade-offs while designing networked shared-data systems. CAP has influenced the design of many distributed data systems. It made designers aware of a wide range of tradeoffs to consider while designing distributed data systems. Over the years, the CAP theorem has been a **widely misunderstood tool** used to categorize databases. There is much misinformation floating around about CAP. Most blog posts on CAP are historical and possibly incorrect.

It is important to understand CAP so that you can identify the misinformation around it.

The CAP theorem applies to distributed systems that store state. Eric Brewer, at the 2000 Symposium on Principles of Distributed Computing (PODC), conjectured that in any networked shared-data system there is a fundamental trade-off between consistency, availability, and partition tolerance. In 2002, Seth Gilbert and Nancy Lynch of MIT published a formal proof of Brewer's conjecture. The theorem states that **networked shared-data systems** can only guarantee/strongly support two of the following three properties:

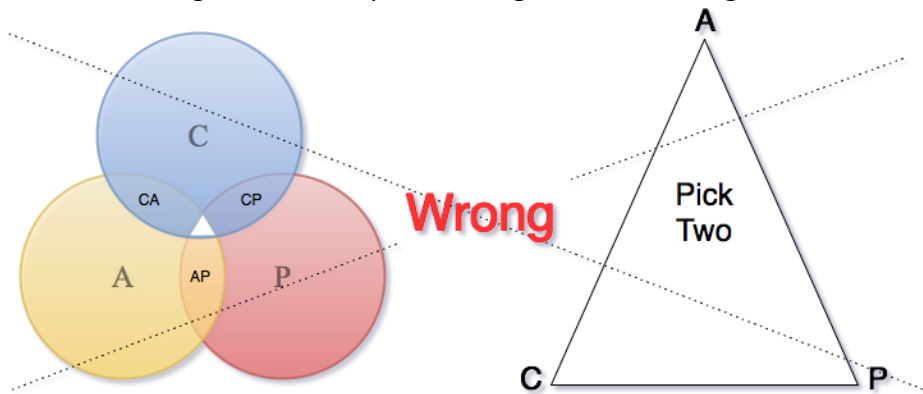
- **Consistency** - A guarantee that every node in a distributed cluster returns the same, most recent, successful write. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.
- **Availability** - Every non-failing node returns a response for all read and write requests in a reasonable amount of time. The key word here is every. To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.
- **Partition Tolerant** - The system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

The C and A in ACID represent different concepts than C and A in the CAP theorem.

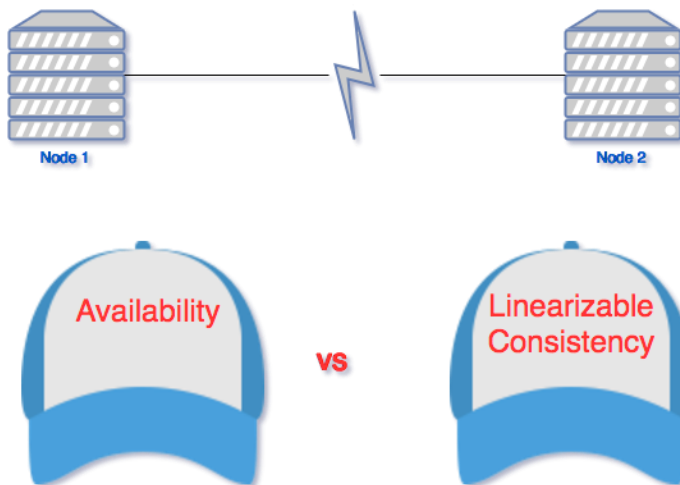
The CAP theorem categorizes systems into three categories:

- CP (Consistent and Partition Tolerant) - At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available. CP is referring to a category of systems where availability is sacrificed only in the case of a network partition.
- CA (Consistent and Available) - CA systems are consistent and available systems in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems. The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the preview of CAP.
- AP (Available and Partition Tolerant) - These are systems that are available and partition tolerant but cannot guarantee consistency.

A Venn diagram or a triangle is frequently used to visualize the CAP theorem. Systems fall into the three categories that depicted using the intersecting circles.



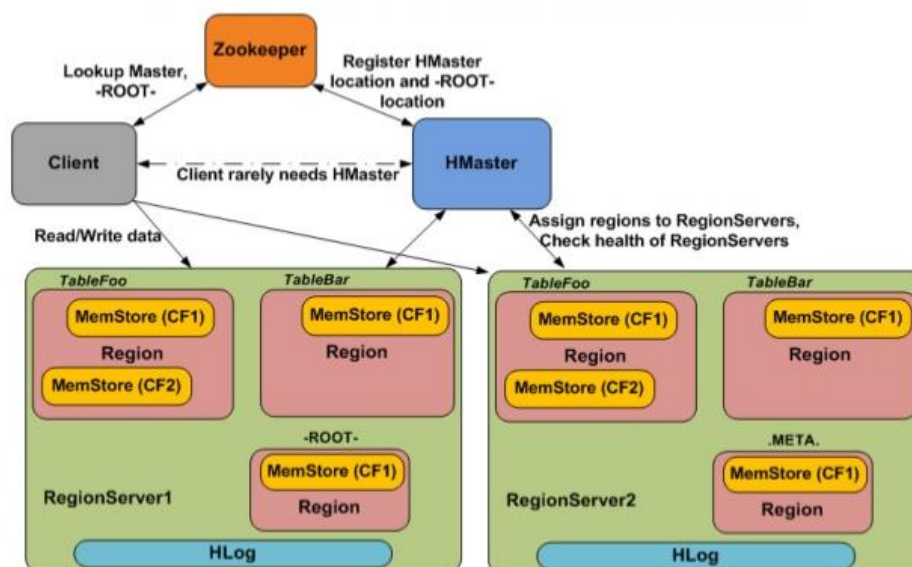
In the event of a partition choose one



The part where all three sections intersect is white because it is impossible to have all three properties in networked shared-data systems. A Venn diagram or a triangle is an **incorrect visualization** of the CAP. Any CAP theorem visualization such as a **triangle or a Venn diagram** is misleading. The correct way to think about CAP is that in case of a network partition (a rare occurrence) one needs to choose between availability and partition tolerance. In any networked shared-data systems partition tolerance is a must. Network partitions and dropped messages are a fact of life and must be handled appropriately. Consequently, system designers must choose between consistency and availability. Simplistically speaking, a network partition forces designers to either choose perfect consistency or perfect availability. Picking consistency means not being able to answer a client's query as the system cannot guarantee to return the most recent write. This sacrifices availability. Network partition forces nonfailing nodes to reject clients' requests as these nodes cannot guarantee consistent data. At the opposite end of the spectrum, being available means being able to respond to a client's request but the system cannot guarantee consistency, i.e., the most recent value written. Available systems provide the best possible answer under the given circumstance

HBase architecture

HBase provides low-latency random reads and writes on top of HDFS. In HBase, tables are dynamically distributed by the system whenever they become too large to handle (Auto Sharding). The simplest and foundational unit of horizontal scalability in HBase is a Region. A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data). HBase architecture has a single HBase master node (HMaster) and several slaves i.e. region servers. Each region server (slave) serves a set of regions, and a region can be served only by a single region server. Whenever a client sends a write request, HMaster receives the request and forwards it to the corresponding region server.



HBase can be run in a multiple master setup, wherein there is only single active master at a time. HBase tables are partitioned into multiple regions with every region storing multiple table's rows.

Components of Apache HBase Architecture

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

i. 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.

ii. 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.

iii. 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, ZKQuorum 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.

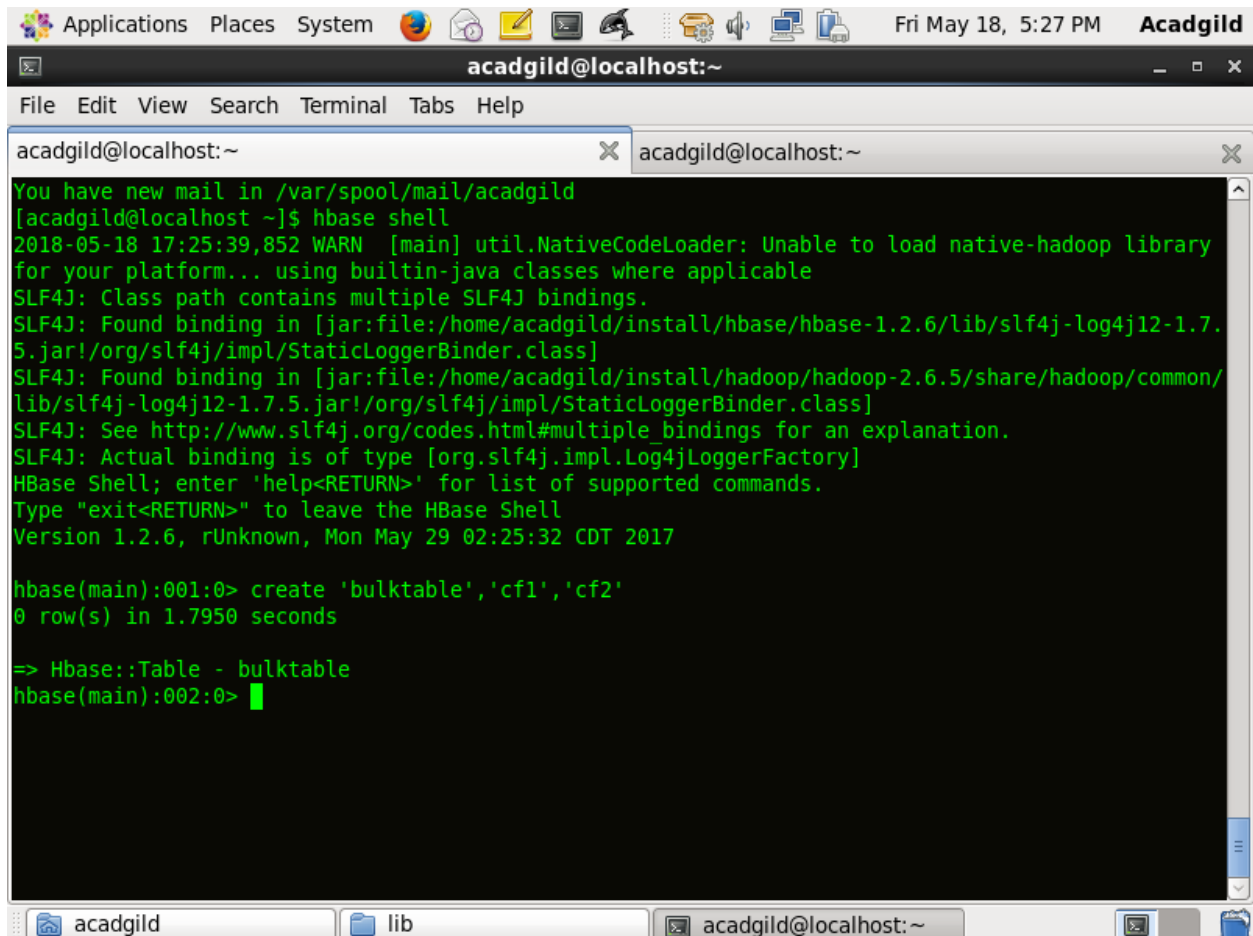
Difference between HBase and RDBMS

The following table differentiates it well:

| Relational database | HBase |
|--|--|
| This supports scale up. In other words, when more disk and memory processing power is needed, we need to upgrade it to a more powerful server. | This supports scale out. In other words, when more disk and memory processing power is needed, we need not upgrade the server. However, we need to add new servers to the cluster. |
| This uses SQL queries for reading records from tables. | This uses APIs and MapReduce for accessing data from HBase tables. |
| This is row oriented, that is, each row is a contiguous unit of page. | This is column oriented, that is, each column is a contiguous unit of page. |
| The amount of data depends on configuration of server. | The amount of data does not depend on the particular machine but the number of machines. |
| It's Schema is more restrictive. | Its schema is flexible and less... |

ImportTSV Data from HDFS into HBase

Creating table



The screenshot shows a terminal window titled 'acadgild@localhost:~'. The window contains the following text:

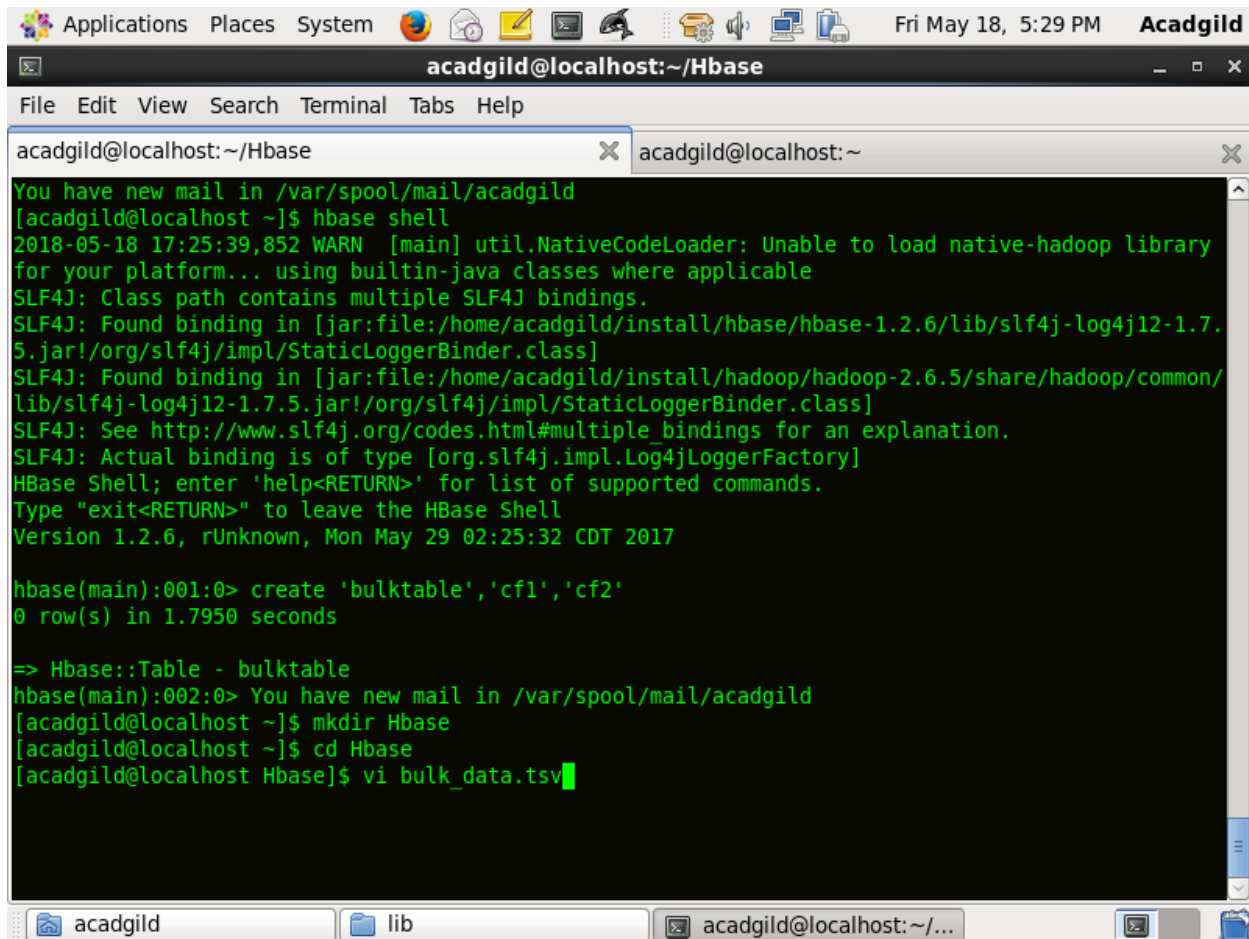
```
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ hbase shell
2018-05-18 17:25:39,852 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.
5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/
lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> create 'bulktable','cf1','cf2'
0 row(s) in 1.7950 seconds

=> Hbase::Table - bulktable
hbase(main):002:0> █
```

The terminal window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', 'Tabs', and 'Help'. The title bar shows 'acadgild@localhost:~'. The bottom of the window shows a taskbar with icons for 'acadgild', 'lib', and 'acadgild@localhost:~'.

Making new local directory Hbase and putting data in bulk_data.tsv in HBase directory



The terminal window shows the user 'acadgild' at 'localhost' in the directory '~/Hbase'. It displays the HBase shell interface where a table named 'bulktable' is created with columns 'cf1' and 'cf2'. After exiting the shell, the user creates a directory 'Hbase', moves into it, and opens a file named 'bulk_data.tsv' in a text editor.

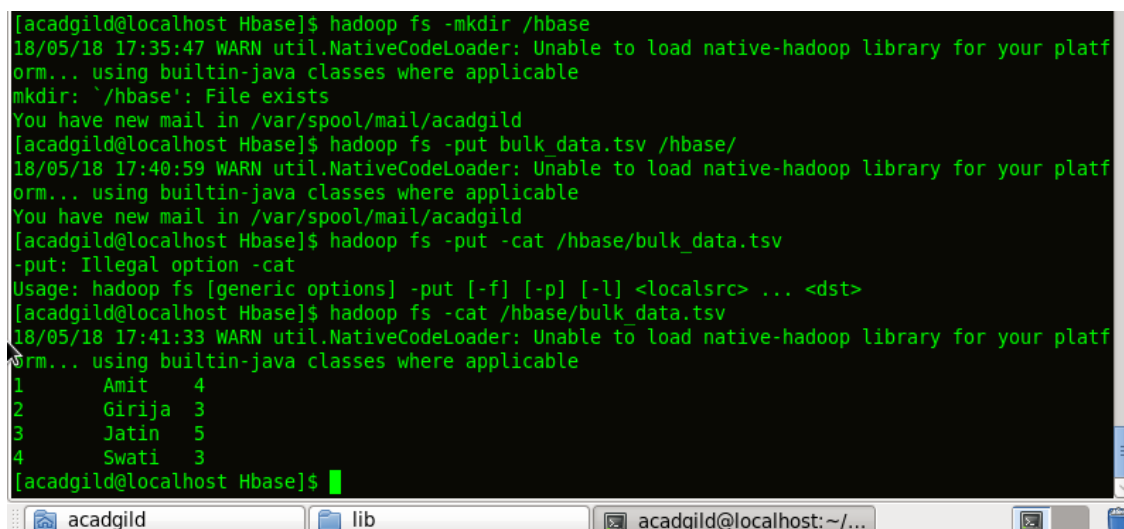
```
acadgild@localhost: ~/Hbase
File Edit View Search Terminal Tabs Help

acadgild@localhost: ~/Hbase
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ hbase shell
2018-05-18 17:25:39,852 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.
5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/
lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> create 'bulktable','cf1','cf2'
0 row(s) in 1.7950 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
```

Lets make the same data available in HDFS in Hbase directory and verify the data using cat command.



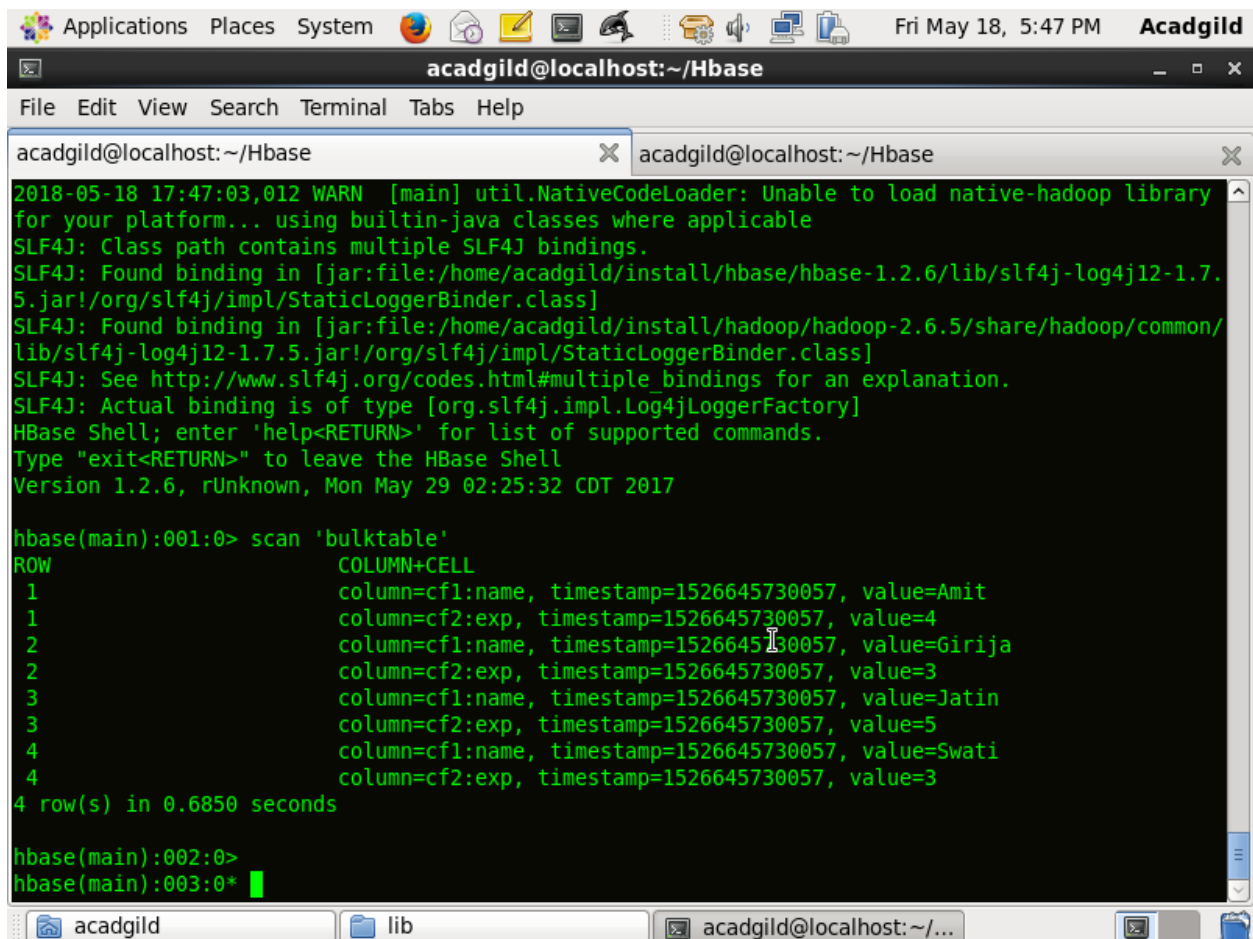
The terminal window shows the user 'acadgild' at 'localhost' in the directory 'Hbase'. It displays Hadoop fs commands to create a directory '/hbase', upload the file 'bulk_data.tsv' to '/hbase/bulk_data.tsv', and then use the 'cat' command to display the contents of the file. The output shows a list of names and counts.

```
[acadgild@localhost Hbase]$ hadoop fs -mkdir /hbase
18/05/18 17:35:47 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platf
orm... 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/
18/05/18 17:40:59 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platf
orm... using builtin-java classes where applicable
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost Hbase]$ hadoop fs -put -cat /hbase/bulk_data.tsv
-put: Illegal option -cat
Usage: hadoop fs [generic options] -put [-f] [-p] [-l] <localsrc> ... <dst>
[acadgild@localhost Hbase]$ hadoop fs -cat /hbase/bulk_data.tsv
18/05/18 17:41:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platf
orm... using builtin-java classes where applicable
1      Amit      4
2      Girija    3
3      Jatin     5
4      Swati     3
[acadgild@localhost Hbase]$
```


Now lets import the data from HDFS to HBase

```
[acadgild@localhost Hbase]$ hbase org.apache.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv
```

Lets check if data is in Hbase or not using scan command



The screenshot shows a terminal window titled 'acadgild@localhost: ~/Hbase'. The window contains the following text:

```
2018-05-18 17:47:03,012 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.
5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/
lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> scan 'bulktable'
ROW          COLUMN+CELL
1            column=cf1:name, timestamp=1526645730057, value=Amit
1            column=cf2:exp, timestamp=1526645730057, value=4
2            column=cf1:name, timestamp=1526645730057, value=Girija
2            column=cf2:exp, timestamp=1526645730057, value=3
3            column=cf1:name, timestamp=1526645730057, value=Jatin
3            column=cf2:exp, timestamp=1526645730057, value=5
4            column=cf1:name, timestamp=1526645730057, value=Swati
4            column=cf2:exp, timestamp=1526645730057, value=3
4 row(s) in 0.6850 seconds

hbase(main):002:0>
hbase(main):003:0*
```

The terminal window also shows a taskbar at the bottom with icons for 'acadgild', 'lib', and 'acadgild@localhost: ~/...'.