HBASE

Presented By
Siva Kumar Bhuchipalli

Your FUTURE is created by what

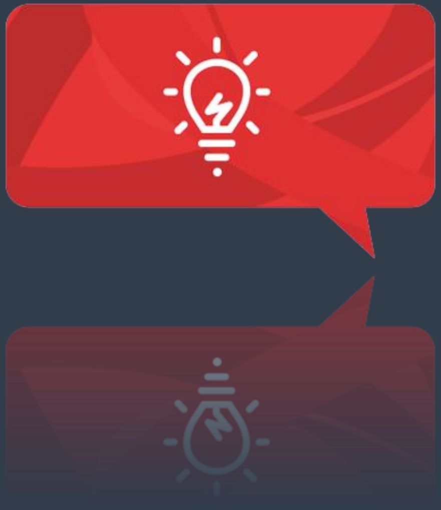
YOU Do TODAY

~~Not~~ ~~TOMORROW~~

- Column-Oriented data store, known as “Hadoop Database”
- Supports random real-time CRUD operations (unlike HDFS)
- Distributed – designed to serve large tables
  - ✓ Billions of rows and millions of columns
- Runs on a cluster of commodity hardware
  - ✓ Server hardware, not laptop/desktops
- Open-source, written in Java
- Type of “NoSQL” DB
  - ✓ Does not provide a SQL based access
  - ✓ Does not adhere to Relational Model for storage
- Horizontally scalable
  - ✓ Automatic sharding*
- Strongly consistent reads and writes
- Automatic fail-over
- Simple Java API
- Integration with Map/Reduce framework
- Thrift, Avro and REST-ful Web-services

When To Use HBASE?

- Not suitable for every problem
  - ✓ Compared to RDBMs has VERY simple and limited API
- Good for large amounts of data
  - ✓ 100s of millions or billions of rows
  - ✓ If data is too small all the records will end up on a single node leaving the rest of the cluster idle



- HBase is memory and CPU intensive

- Two well-known use cases
    - ✓ Lots and lots of data (already mentioned)
    - ✓ Large amount of clients/requests (usually cause a lot of data)
- Great for single random selects and range scans by key
- Great for variable schema
    - ✓ Rows may drastically differ
    - ✓ If your schema has many columns and most of them are null

HBASE Data Model?

- Namespaces are Databases

Data is stored in Tables

- Tables contain rows
  - ✓Rows are referenced by a unique key
- Key is an array of bytes – good news
- Anything can be a key: string, long and your own serialized data structures
- Rows made of columns which are grouped in column families
- Data is stored in cells
  - ✓Identified by row x column-family x column
  - ✓Cell's content is also an array of bytes

HBASE Families?

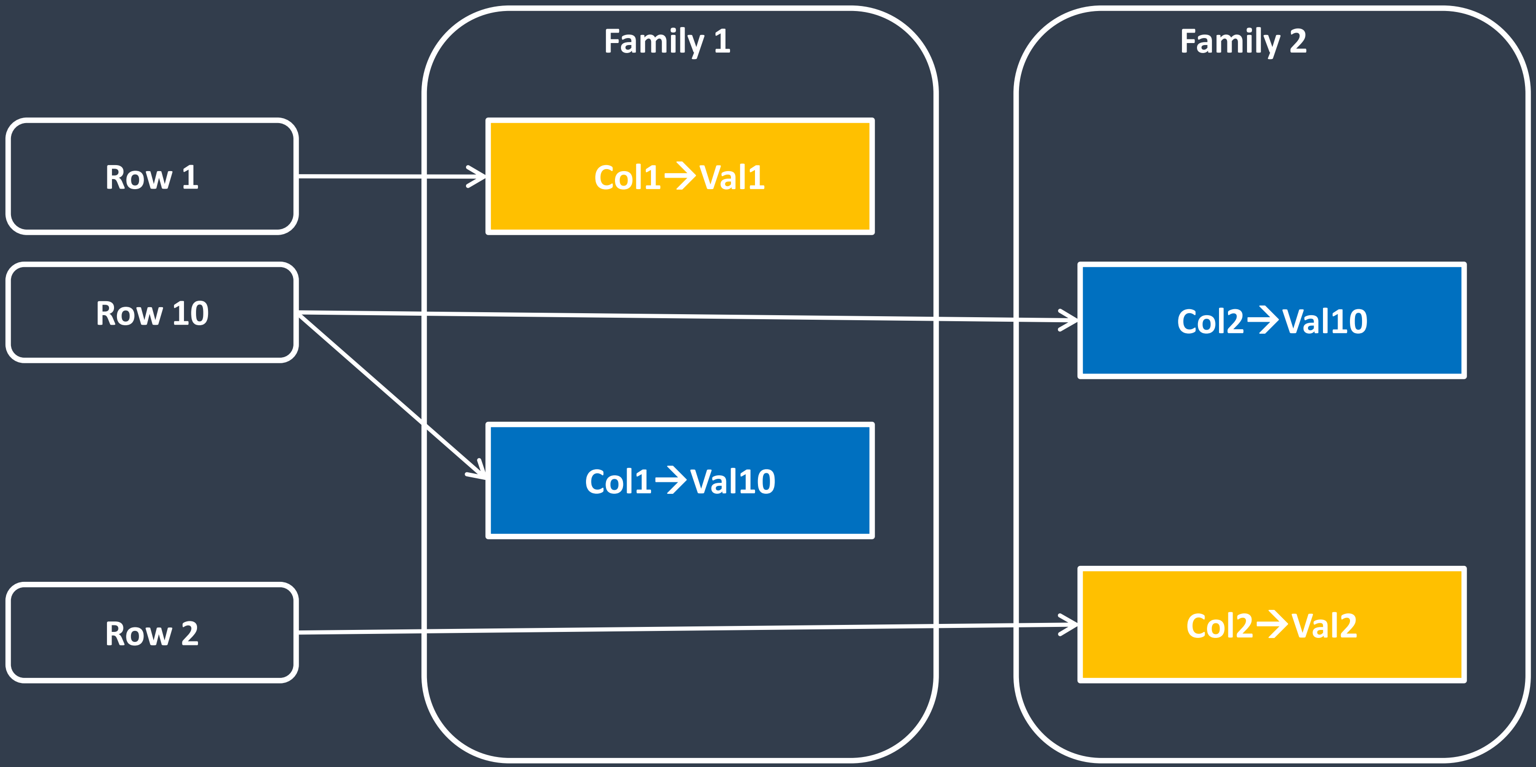
- Columns are grouped into families
  - ✓Labeled as "family:column"
- Example "user:first_name"
  - ✓A way to organize your data
- Compression
- In-memory option
- Stored together - in a file called HFile/StoreFile
- Family definitions are static initially, it is no longer static
  - ✓Created with table, should be rarely added and changed
  - ✓Limited to small number of families
- unlike columns that you can have millions of

- Family name must be composed of printable characters
  - ✓Not bytes, unlike keys and values
- Think of family:column as a tag for a cell value and NOT as a spreadsheet
- Columns on the other hand are NOT static
  - ✓Create new columns at run-time
  - ✓Can scale to millions for a family

HBASE Row Keys

- Rows are sorted lexicographically by key
  - ✓Compared on a binary level from left to right
  - ✓For example keys 1,2,3,10,15 will get sorted as 1, 10, 15, 2, 3
- Somewhat similar to Relational DB primary index
  - ✓Always unique
  - ✓Some but minimal secondary indexes support

Row Composed Of Cells Stored In Families : Columns

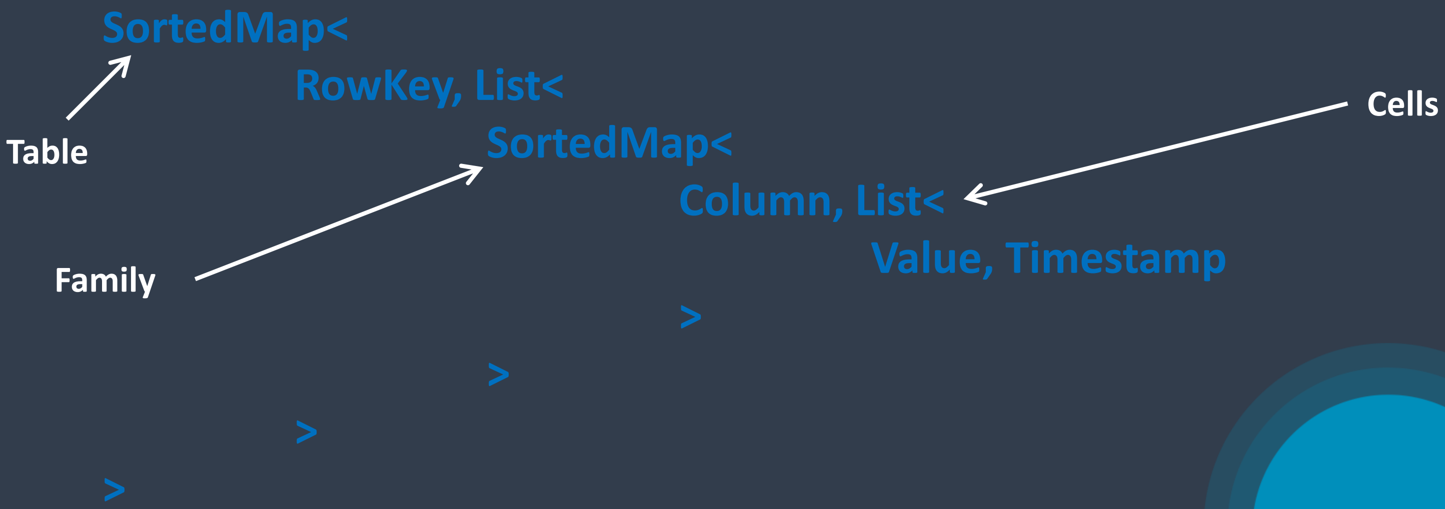


HBASE Timestamp?

- Cells' values are versioned
  - ✓For each cell multiple versions are kept
- 3 by default
  - ✓Another dimension to identify your data
  - ✓Either explicitly timestamped by region server or provided by the client
- Versions are stored in decreasing timestamp order
- Read the latest first – optimization to read the current value
- You can specify how many versions are kept
  - ✓More on this later....

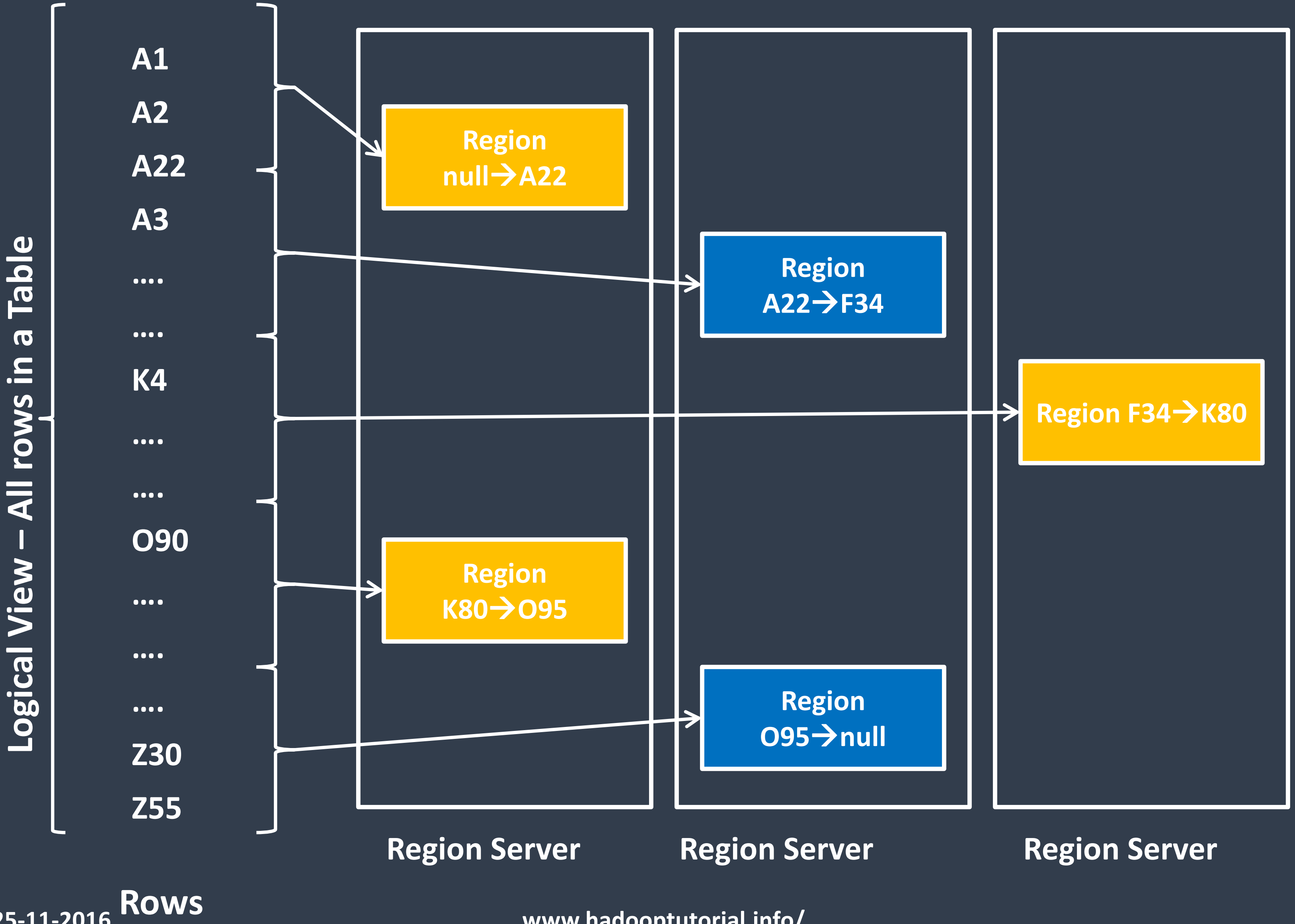
HBASE Cells

- Value = Table+RowKey+Family+Column+Timestamp
- Programming language style:



```
SortedMap<
          RowKey, List<
                    SortedMap<
                              Column, List<
                                        Value, Timestamp
                              >
                    >
          >
>
```

Row Distribution Between Region Server



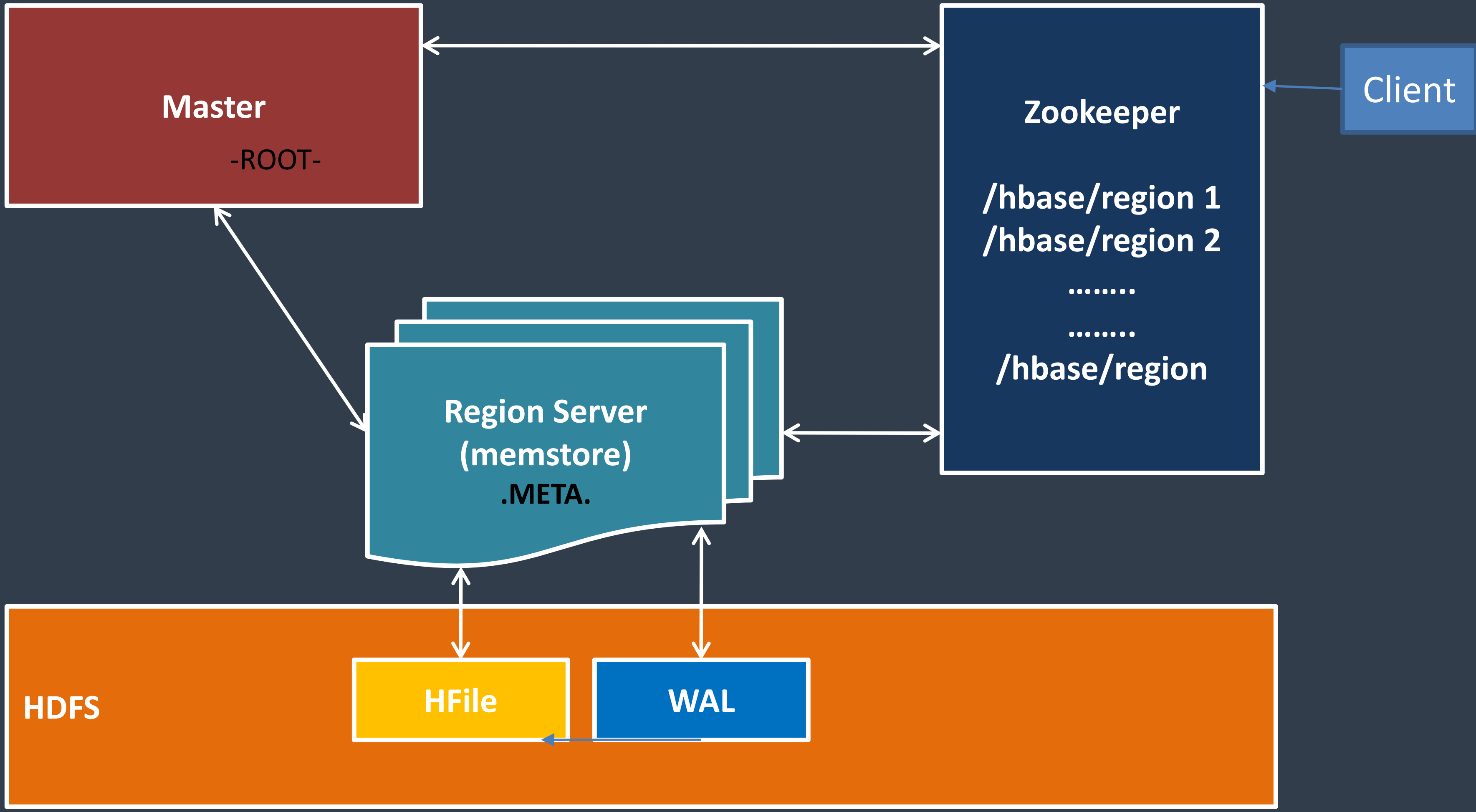
HBASE Cells

Row Key	Time Stamp	Name Family		Address Family	
		First_name	Last_name	Number	Address
row1	t1	Bob	Smith		
	t5			10	First Lane
	t10			30	Other Lane
	t15			7	Last Street
row2	t20	Mary	Tompson		
	t22			77	One Street
	t30		Thompson		

HBASE Architecture

- Table is made of regions
- Region – a range of rows stored together
    - ✓Single shard, used for scaling
    - ✓Dynamically split as they become too big and merged if too small
- Region Server- serves one or more regions
- Master Server – daemon responsible for managing HBase cluster, aka Region Servers
- HBase stores its data into HDFS
    - ✓relies on HDFS's high availability and fault-tolerance features
- HBase utilizes Zookeeper for distributed coordination

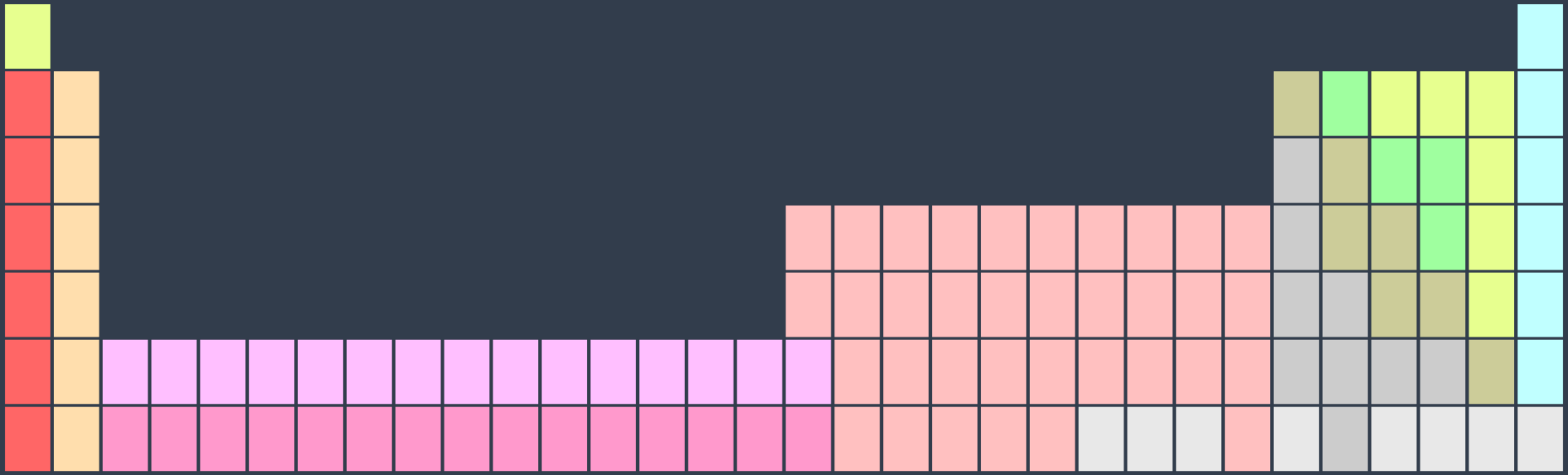
HBASE Components



HBASE Regions

- Region is a range of keys
    - ✓start key → stop key (ex. k3cod → odiekd)
    - ✓start key inclusive and stop key exclusive
- Addition of data
    - ✓At first there is only 1 region
    - ✓Addition of data will eventually exceed the configured maximum then region will be split
        - ❑Default is 256MB
    - ✓The region is split into 2 regions at the middle key
- Regions per server depend on hardware specs, with today's hardware it's common to have:
    - ✓10 to 1000 regions per Region Server
    - ✓Managing as much as 1GB to 2 GB per region

- Splitting data into regions allows
  - ✓Fast recovery when a region fails
  - ✓Load balancing when a server is overloaded
- May be moved between servers
  - ✓Splitting is fast
- Reads from an original file while asynchronous process performs a split
  - ✓All of these happen automatically without user's involvement



HBASE Storage

- Data is stored in files called HFiles/StoreFiles
    - ✓Usually saved in HDFS
- HFile is basically a key-value map
    - ✓Keys are sorted lexicographically
- When data is added it's written to a log called Write Ahead Log (WAL) and is also stored in memory (memstore)
- Flush: when in-memory data exceeds maximum value it is flushed to an HFile
    - ✓Data persisted to HFile can then be removed from WAL
    - ✓Region Server continues serving read-writes during the flush operations, writing values to the WAL and memstore

- Recall that HDFS doesn't support updates to an existing file therefore HFiles are immutable
  - ✓Cannot remove key-values out of HFile(s)
  - ✓Over time more and more HFiles are created
- Delete marker is saved to indicate that a record was removed
  - ✓These markers are used to filter the data - to “hide” the deleted records
  - ✓At runtime, data is merged between the content of the HFile and WAL

- To control the number of HFiles and to keep cluster well balanced HBase periodically performs data compactions
  - ✓Minor Compaction: Smaller HFiles are merged into larger HFiles (n-way merge)
    - ❑Fast - Data is already sorted within files
    - ❑Delete markers are not applied
  - ✓Major Compaction:
    - ❑For each region merges all the files within a column-family into a single file
    - ❑Scan all the entries and apply all the deletes as necessary

HBASE Master

- Responsible for managing regions and their locations
    - ✓Assigns regions to region servers
    - ✓Re-balanced to accommodate workloads
    - ✓Recovers if a region server becomes unavailable
    - ✓Uses Zookeeper – distributed coordination service
- Doesn't actually store or read data
    - ✓Clients communicate directly with Region Servers
    - ✓Usually lightly loaded
- Responsible for schema management and changes
    - ✓Adding/Removing tables and column families

HBASE And Zookeeper

- HBase uses Zookeeper extensively for region assignment

"Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services"
- zookeeper.apache.org

- HBase can manage Zookeeper daemons for you or you can install/manage them separately
- Learn More at http://zookeeper.apache.org

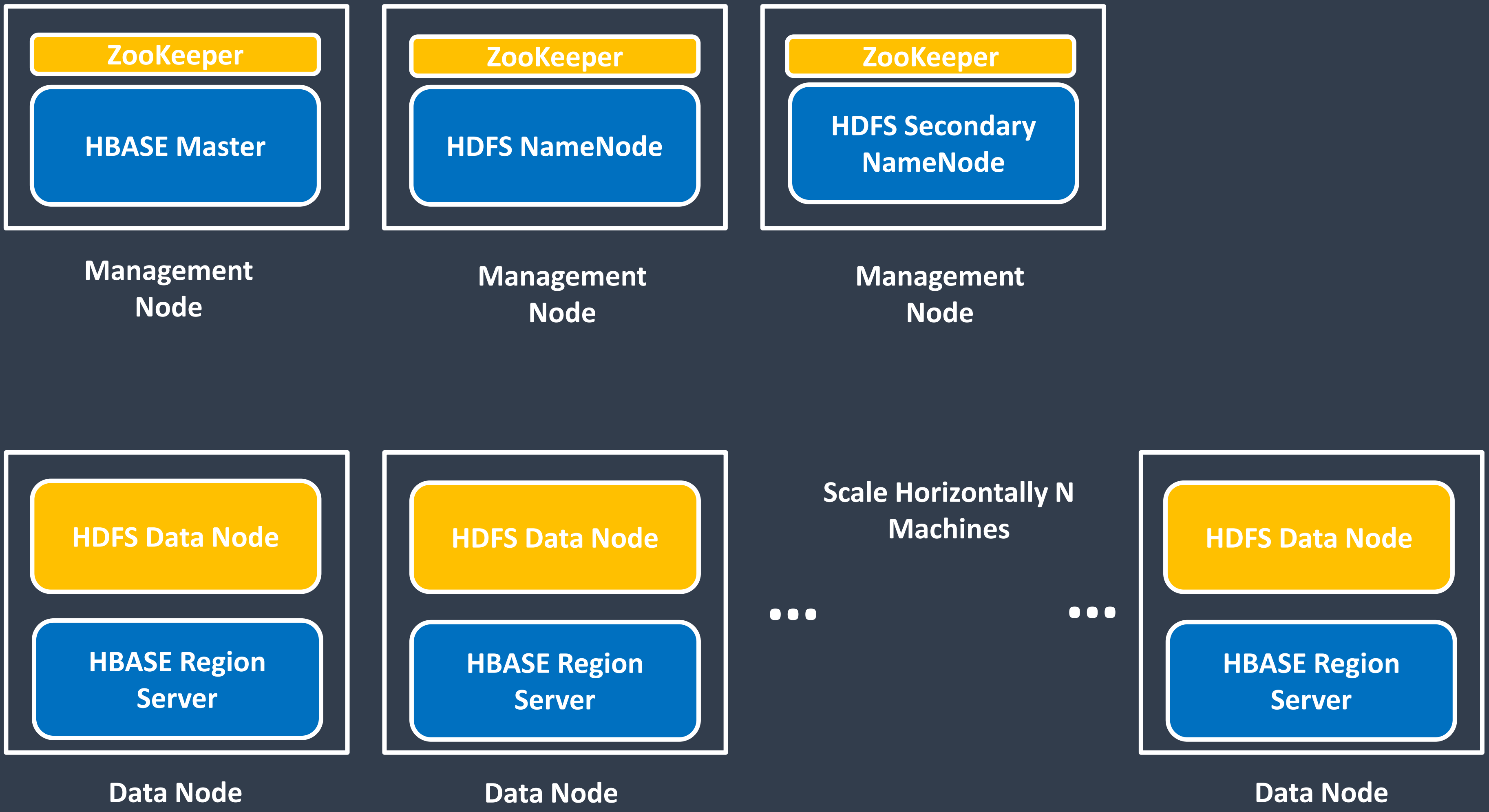
HBASE And Zookeeper

- Zookeeper crash course
  - ✓Very simple file-like API, written in Java
  - ✓Operations on directories and files (called Znodes)
  - ✓CRUD ZNodes and register for updates
    - ❑Supports PERSISTENT and EPHERMAL Znodes
  - ✓Clients connect with a session to Zookeeper
    - ❑Session is maintained via heartbeat, if client fails to report then the session is expired and all the EPHERMAL nodes are deleted
    - ❑Clients listening for updates will be notified of the deleted nodes as well as new nodes

HBASE And Zookeeper

- Each Region Server creates an ephemeral node
  - ✓Master monitors these nodes to discover available region servers
  - ✓Master also tracks these nodes for server failures
- Uses Zookeeper to make sure that only 1 master is registered
- HBase cannot exist without Zookeeper

HBASE Deployment



HBASE

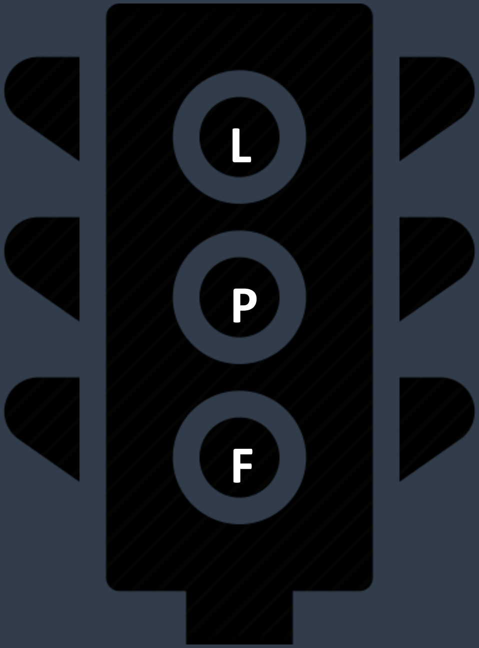
INSTALLATION

HBASE Installation Agenda

- Learn about installation modes
- How to set-up Pseudo-Distributed Mode
- HBase Management Console
- HBase Shell
    - ✓ Define Schema
    - ✓ Create, Read, Update and Delete

Runtime Modes

- Local (Standalone) Mode
  - ✓ Comes Out-of-the-Box, easy to get started
  - ✓ Uses local filesystem (not HDFS), NOT for production
  - ✓ Runs HBase & Zookeeper in the same JVM
- Pseudo-Distributed Mode
  - ✓ Requires HDFS
  - ✓ Mimics Fully-Distributed but runs on just one host
  - ✓ Good for testing, debugging and prototyping
  - ✓ Not for production use or performance benchmarking!
  - ✓ Development mode used in class
- Fully-Distributed Mode
  - ✓ Run HBase on many machines
  - ✓ Great for production and development clusters



Pseudo Distributed Mode

1. Verify Installation Requirements
    - Java, password-less SSH

2. Configure Java

3. Configure the use of HDFS
    - Specify the location of Namenode
    - Configure replication

4. Make sure HDFS is running

5. Start Hbase

6. Verify HBase is running

6.Verify HBASE Running

```
$ hbase shell

HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.4-cdh3u2, r, Thu Oct 13 20:32:26 PDT 2011
hbase(main):001:0> list
TABLE
0 row(s) in 0.4070 seconds
```

Run a command to verify that cluster is actually running

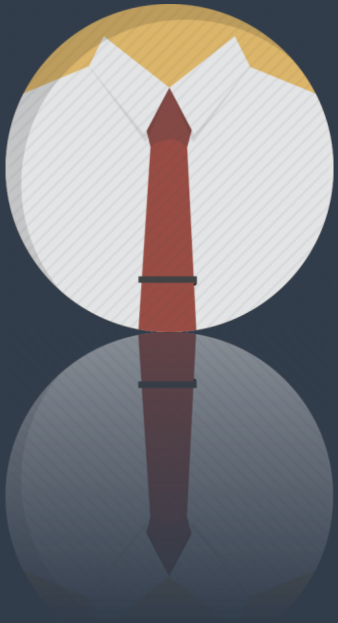
```
$ hadoop fs -ls /hbase

Found 5 items
drwxr-xr-x   - hadoop supergroup    0  2011-12-31 13:18   /hbase/-ROOT-
drwxr-xr-x   - hadoop supergroup    0  2011-12-31 13:18   /hbase/.META.
drwxr-xr-x   - hadoop supergroup    0  2011-12-31 13:18   /hbase/.logs
drwxr-xr-x   - hadoop supergroup    0  2011-12-31 13:18   /hbase/.oldlogs
-rw-r--r--   1 hadoop supergroup    3  2011-12-31 13:18   /hbase/hbase.version
```

HBase data and metadata is stored in HDFS

HBASE Management Console

- HBase comes with web based management
  - ✓ http://localhost:60010
- Both Master and Region servers run web server
  - ✓ Browsing Master will lead you to region servers
    - ❑ Regions run on port 60030
- Firewall considerations
  - ✓ Opening <master_host>:60010 in firewall is not enough
  - ✓ Have to open up <region(s)_host>:60030 on every slave host
  - ✓ An easy option is to open a browser behind the firewall
    - ❑ SSH tunneling and Virtual Network Computing (VNC)



HBASE Shells

- JRuby IRB (Interactive Ruby Shell)
  - ✓ HBase commands added
  - ✓ If you can do it in IRB you can do it in HBase shell
- http://en.wikipedia.org/wiki/Interactive_Ruby_Shell
- To run simply

```
$ <hbase_install>/bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.4-cdh3u2, r, Thu Oct 13 20:32:26 PDT 2011
hbase(main):001:0>
```

  - ✓ Puts you into IRB
  - ✓ Type 'help' to get a listing of commands
    - ❑ $ help "command" (quotes are required)
    - – > help "get"

If Daemons are not running or Hbase is not working

- sudo service hbase-master restart
- sudo service hbase-regionserver restart
- sudo service zookeeper-server restart

- Quote all names
  - ✓ Table and column names
  - ✓ Single quotes for text
  - hbase> get 't1', 'myRowId'
  - ✓ Double quotes for binary
    - ❑ Use hexadecimal representation of that binary value
    - ❑ hbase> get 't1', "key\x03\x3f\xcd"
- Uses ruby hashes to specify parameters
  - ✓ {'key1' => 'value1', 'key2' => 'value2', …}
  - ✓ Example:

    `hbase> get 'UserTable', 'userId1', {COLUMN => 'address:str'}`

- HBase Shell supports various commands
  - ✓ General
    - ❑ status, version
  - ✓ Data Definition Language (DDL)
    - ❑ alter, create, describe, disable, drop, enable, exists, is_disabled, is_enabled, list
  - ✓ Data Manipulation Language (DML)
    - ❑ count, delete, deleteall, get, get_counter, incr, put, scan, truncate
  - ✓ Cluster administration
    - ❑ balancer, close_region, compact, flush, major_compact, move, split, unassign, zk_dump, add_peer, disable_peer, enable_peer,remove_peer, start_replication, stop_replication
- Learn more about each command
  - ✓ `hbase> help "<command>"`

HBASE Shell – Check Status

- Display cluster's status via status command
  - ✓ hbase> status
  - ✓ hbase> status 'detailed'
- Similar information can be found on Hbase Web Management Console
  - ✓ http://localhost:60010

```
hbase> status
1 servers, 0 dead, 3.0000 average load

hbase> status 'detailed'
version 0.90.4-cdh3u2
0 regionsInTransition
1 live servers
        hadoop-laptop:39679 1326056194009
                requests=0, regions=3, usedHeap=30, maxHeap=998
                .META.,,1
                        stores=1, storefiles=0, storefileSizeMB=0, …
                -ROOT-,,0
                        stores=1, storefiles=1, storefileSizeMB=0, …
        Blog,,1326059842133.c1b865dd916b64a6228ecb4f743 …
0 dead servers
```

HBASE Shell DDL and DML

Let's walk through an example

1. Create a table
    - Define column families
2. Populate table with data records
    - Multiple records
3. Access data
    - Count, get and scan
4. Edit data
5. Delete records
6. Drop table

HBASE Shell NameSpace

```
create_namespace 'ns1‘
describe_namespace ‘ns1‘
list_namespace
list_namespace_tables 'default‘

alter_namespace 'ns1', {METHOD => 'set', 'PROERTY_NAME' =>
'PROPERTY_VALUE'}

alter_namespace 'ns1', {METHOD => 'unset',
NAME=>'PROERTY_NAME'}
```

1.Create Table

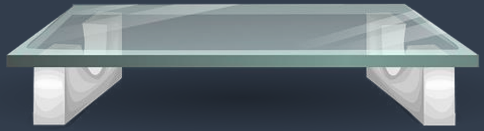
- Create table called 'Blog' with the following schema
    - ✓ 2 families
        - ❑ 'info' with 3 columns: 'title', 'author', and 'date'
        - ❑ 'content' with 1 column family: 'post'

Blog		
Family:	Info:	Columns: title, author, date
	Content:	Columns:Post

1.Create Table

- Various options to create tables and families
  - ✓ hbase> create 't1', {NAME => 'f1', VERSIONS => 5}
  - ✓ hbase> create 't1', {NAME => 'f1', VERSIONS => 1,TTL => 2592000, BLOCKCACHE => true}
  - ✓ hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'},{NAME => 'f3'}
  - ✓ hbase> create 't1', 'f1', 'f2', 'f3'

```
hbase> create 'Blog', {NAME=>'info'}, {NAME=>'content'}
0 row(s) in 1.3580 seconds
```



2.Populate Table With Data Records

- Populate data with multiple records

Row Id	Info:title	Info:author	Info:date	Content:post
Matt-001	Elephant	Matt	2009.05.06	Do Elephants like monkeys?
Matt-002	Monkey	Matt	2011.02.14	Do monkeys like elephants?
Bob-003	Dog	Bob	1995.10.20	People Own Dogs!
Michelle-004	Cat	Michelle	1990.07.06	I have a cat!
John-005	Mouse	John	2012.01.15	Mickey Mouse

- Put command format:

```
hbase> put 'table', 'row_id', 'family:column', 'value'
```

```
# insert row 1
put 'Blog', 'Matt-001', 'info:title', 'Elephant'
put 'Blog', 'Matt-001', 'info:author', 'Matt'
put 'Blog', 'Matt-001', 'info:date', '2009.05.06'
put 'Blog', 'Matt-001', 'content:post', 'Do elephants like monkeys?'
…
…
… # insert rows 2-4
…
…
# row 5
put 'Blog', 'John-005', 'info:title', 'Mouse'
put 'Blog', 'John-005', 'info:author', 'John'
put 'Blog', 'John-005', 'info:date', '1990.07.06'
put 'Blog', 'John-005', 'content:post', 'Mickey mouse.'
```

Put statement per cell

Access Data - Count

- Access Data
  - ✓ count: display the total number of records
  - ✓ get: retrieve a single row
  - ✓ scan: retrieve a range of rows
- Count is simple
  - ✓ hbase> count 'table_name'
  - ✓ Will scan the entire table! May be slow for a large table
    - ❑ Alternatively can run a MapReduce job (more on this later...)
      - ○ $ yarn jar hbase.jar rowcount
  - ✓ Specify count to display every n rows. Default is 1000
    - ❑ hbase> count 't1', INTERVAL => 10

```
hbase> count 'Blog', {INTERVAL=>2}
Current count: 2, row: John-005
Current count: 4, row: Matt-002
5 row(s) in 0.0220 seconds
```

```
hbase> count 'Blog', {INTERVAL=>1}
Current count: 1, row: Bob-003
Current count: 2, row: John-005
Current count: 3, row: Matt-001
Current count: 4, row: Matt-002
Current count: 5, row: Michelle-004
```

Affect how often count is displayed

Access Data - Get

- Select single row with 'get' command
  - ✓ hbase> get 'table', 'row_id'
    - ❑ Returns an entire row
  - ✓ Requires table name and row id
  - ✓ Optional: timestamp or time-range, and versions

- Select specific columns
  - ✓ hbase> get 't1', 'r1', {COLUMN => 'c1'}
  - ✓ hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}

- Select specific timestamp or time-range
  - ✓ hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
  - ✓ hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}

- Select more than one version
  - ✓ hbase> get 't1', 'r1', {VERSIONS => 4}

Access Data - Get

```
hbase> get 'Blog', 'unknownRowId'
COLUMN                                  CELL
0 row(s) in 0.0250 seconds
```

Row Id Doesn’t exist

```
hbase> get 'Blog', 'Michelle-004'
COLUMN                                  CELL
content:post          timestamp=1326061625690, value=I have a cat!
info:author           timestamp=1326061625630, value=Michelle
info:date             timestamp=1326061625653, value=1990.07.06
info:title            timestamp=1326061625608, value=Cat
4 row(s) in 0.0420 seconds
```

Returns ALL Columns, display 1 column per row!!!

Access Data - Get

Narrow down to just two columns

```
hbase> get 'Blog', 'Michelle-004',
                    {COLUMN=>['info:author','content:post']}
COLUMN                                  CELL
content:post          timestamp=1326061625690, value=I have a cat!
info:author           timestamp=1326061625630, value=Michelle
2 row(s) in 0.0100 seconds
```

Narrow down to via columns and timestamp

```
hbase> get 'Blog', 'Michelle-004',
          {COLUMN=>['info:author','content:post'],TIMESTAMP=>1326061625690}
COLUMN                                  CELL
content:post timestamp=1326061625690, value=I have a cat!
1 row(s) in 0.0140 seconds
```

Only One timestamp matches

Access Data - Get

```
hbase> get 'Blog', 'Michelle-004',
                        {COLUMN=>'info:date', VERSIONS=>2}
COLUMN                                    CELL
info:date                                 timestamp=1326071670471, value=1990.07.08
info:date                                 timestamp=1326071670442, value=1990.07.07
2 row(s) in 0.0300 seconds
```

Ask for the latest two versions

```
hbase> get 'Blog', 'Michelle-004',
                        {COLUMN=>'info:date'}
COLUMN                                    CELL
info:date                                 timestamp=1326071670471, value=1990.07.08
1 row(s) in 0.0190 seconds
```

By Default only the latest version is returned

Access Data - Scan

- Scan entire table or a portion of it
- Load entire row or explicitly retrieve column families, columns or specific cells
- To scan an entire table
  - ✓ `hbase> scan 'table_name'`
- Limit the number of results
  - ✓ `hbase> scan 'table_name', {LIMIT=>1}`
- Scan a range
  - ✓ `hbase> scan 'Blog', {STARTROW=>'startRow', STOPROW=>'stopRow'}`
  - ✓ Start row is inclusive, stop row is exclusive
  - ✓ Can provide just start row or just stop row

Access Data - Scan

- Limit what columns are retrieved
  - ✓ hbase> scan 'table', {COLUMNS=>['col1', 'col2']}
- Scan a time range
  - ✓ hbase> scan 'table', {TIMERANGE => [1303, 13036]}
- Limit results with a filter
  - ✓ hbase> scan 'Blog', {FILTER => org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}
  - ✓ More about filters later

  - ✓ scan 'blog2',{FILTER => "ValueFilter(=,'binaryprefix:Mickey')"}
  - ✓ scan 'blog2',{FILTER => "FirstKeyOnlyFilter()"}
  - ✓ scan 'blog2',{FILTER => "KeyOnlyFilter()"}
  - ✓ scan 'blog2',{FILTER => "(PrefixFilter ('john'))"}

http://hadooptutorial.info/hbase-functions-cheat-sheet/

Scan the entire table, grab ALL the columns

```
hbase(main):014:0> scan 'Blog‘

ROW COLUMN+CELL
Bob-003 column=content:post,          timestamp=1326061625569,
                                              value=People own dogs!
Bob-003 column=info:author,           timestamp=1326061625518, value=Bob
Bob-003 column=info:date,             timestamp=1326061625546,
                                              value=1995.10.20
Bob-003 column=info:title,            timestamp=1326061625499, value=Dog
John-005 column=content:post,         timestamp=1326061625820,
                                              value=Mickey mouse.
John-005 column=info:author,          timestamp=1326061625758,
                                              value=John
…
Michelle-004 column=info:author,      timestamp=1326061625630,
                                              value=Michelle
Michelle-004 column=info:date,        timestamp=1326071670471,
                                              value=1990.07.08
Michelle-004 column=info:title,       timestamp=1326061625608,
                                              value=Cat
5 row(s) in 0.0670 seconds
```

Stop row is exclusive, row ids that start with John will not be included

```
hbase> scan 'Blog', {STOPROW=>'John'}
ROW                  COLUMN+CELL
Bob-003              column=content:post,          timestamp=1326061625569,
                                                             value=People own dogs!
Bob-003              column=info:author,           timestamp=1326061625518,
                                                             value=Bob
Bob-003              column=info:date,             timestamp=1326061625546,
                                                             value=1995.10.20
Bob-003              column=info:title,            timestamp=1326061625499,
                                                             value=Dog
1 row(s) in 0.0410 seconds
```

Only retrieve 'info:title' column

```
hbase> scan 'Blog', {COLUMNS=>'info:title',
                     STARTROW=>'John', STOPROW=>'Michelle'}
ROW                  COLUMN+CELL
John-005             column=info:title,             timestamp=1326061625728,
                                                             value=Mouse
Matt-001             column=info:title,             timestamp=1326061625214,
                                                             value=Elephant
Matt-002             column=info:title,             timestamp=1326061625383,
                                                             value=Monkey
3 row(s) in 0.0290 seconds
```

Edit Data

- Put command inserts a new value if row id doesn't exist
- Put updates the value if the row does exist
- But does it really update?
  - ✓ Inserts a new version for the cell
  - ✓ Only the latest version is selected by default
  - ✓ N versions are kept per cell
    - ❑ configured per family at creation:
      - ○
        ```
        hbase> create 'table', {NAME => 'family', VERSIONS => 7}
        ```
    - ❑ 3 versions are kept by default

```
hbase> put 'Blog', 'Michelle-004', 'info:date', '1990.07.06'
0 row(s) in 0.0520 seconds
hbase> put 'Blog', 'Michelle-004', 'info:date', '1990.07.07'
0 row(s) in 0.0080 seconds
hbase> put 'Blog', 'Michelle-004', 'info:date', '1990.07.08'
0 row(s) in 0.0060 seconds
```

Update the same exact row with a different value

```
hbase> get 'Blog', 'Michelle-004',
                    {COLUMN=>'info:date', VERSIONS=>3}
COLUMN              CELL
info:date           timestamp=1326071670471, value=1990.07.08
info:date           timestamp=1326071670442, value=1990.07.07
info:date           timestamp=1326071670382, value=1990.07.06
3 row(s) in 0.0170 seconds
```

Keeps three versions of each cell by default

```
hbase> get 'Blog', 'Michelle-004',
                      {COLUMN=>'info:date', VERSIONS=>2}
COLUMN                CELL
info:date             timestamp=1326071670471, value=1990.07.08
info:date             timestamp=1326071670442, value=1990.07.07
2 row(s) in 0.0300 seconds
```

Asks for the latest two versions

```
hbase> get 'Blog', 'Michelle-004',
                      {COLUMN=>'info:date'}
COLUMN                CELL
info:date             timestamp=1326071670471, value=1990.07.08
1 row(s) in 0.0190 seconds
```

By default only the latest version is returned

Delete Records

- Delete cell by providing table, row id and column coordinates
  - ✓ delete 'table', 'rowId', 'column'
  - ✓ Deletes all the versions of that cell
  - ✓ deleteall command for deleting the entire row
  - ✓ truncate command for deleting all the rows at a time but keeping the schema
- Optionally add timestamp to only delete versions before the provided timestamp
  - ✓ Delete 'table', 'rowId', 'column', timestamp

✓ scan 'blog',{RAW => true}

```
 Matt-001                    column=info:author, timestamp=1467774783983
type=DeleteColumn
 Matt-001                    column=info:author, timestamp=1467772792122
value=Ramesh
```

Delete Records

```
hbase> get 'Blog', 'Bob-003', 'info:date'
COLUMN                CELL
info:date             timestamp=1326061625546, value=1995.10.20
1 row(s) in 0.0200 seconds
```

```
hbase> delete 'Blog', 'Bob-003', 'info:date'
0 row(s) in 0.0180 seconds
```

```
hbase> get 'Blog', 'Bob-003', 'info:date'
COLUMN                CELL
0 row(s) in 0.0170 seconds
```

Delete Records

```
hbase> get 'Blog', 'Michelle-004',
                    {COLUMN=>'info:date', VERSIONS=>3}
COLUMN              CELL
info:date           timestamp=1326254742846, value=1990.07.08
info:date           timestamp=1326254739790, value=1990.07.07
info:date           timestamp=1326254736564, value=1990.07.06
3 row(s) in 0.0120 seconds
```

3 versions

```
hbase> delete 'Blog', 'Michelle-004', 'info:date', 1326254739791
0 row(s) in 0.0150 seconds
```

1 millisecond after the second version

```
hbase> get 'Blog', 'Michelle-004',
                    {COLUMN=>'info:date', VERSIONS=>3}
COLUMN              CELL
info:date           timestamp=1326254742846, value=1990.07.08
1 row(s) in 0.0090 seconds
```

After the timestamp provided at delete statement

Drop Table

- Must disable before dropping
  - ✓ puts the table “offline” so schema based operations can be performed
  - ✓ hbase> disable 'table_name'
  - ✓ hbase> drop 'table_name‘

- For a large table it may take a long time....

```
hbase> list
TABLE
Blog
1 row(s) in 0.0120 seconds

hbase> disable 'Blog'
0 row(s) in 2.0510 seconds

hbase> drop 'Blog'
0 row(s) in 0.0940 seconds

hbase> list
TABLE
0 row(s) in 0.0200 seconds
```

Take the table offline for schema modifications

Bulk Import

```
[cloudera@quickstart ~]$ hadoop fs -appendToFile - /user/cloudera/hbase_bulkimport_tsv
123        emp1      sse
345        emp2      tl
456        emp3      sse
hbase(main):002:0> create 'emp','info','desc'
[cloudera@quickstart ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.columns=HBASE_ROW_KEY,info:name,desc:role emp
/user/cloudera/hbase_bulkimport_tsv
```

Onprem:

```
hbase org.apache.hadoop.hbase.mapreduce.Export ns1:FACT_DETAIL
/user/nreuser/export/fact_detail/fact_detail
```

copy the file to cloud edge node

Cloud

```
hbase org.apache.hadoop.hbase.mapreduce.Import ns1:FACT_DETAIL
/user/selvka12/folder/fact_detail/
```

http://hadooptutorial.info/forums/topic/hbase-bulk-loading-with-importtsv/

Accessing Hbase table in Hive

```
hive> CREATE EXTERNAL TABLE hbase_table_emp(key string, name string, role string)
    > STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES ("hbase.columns.mapping" = "info:name,desc:role")
    > TBLPROPERTIES("hbase.table.name" = "emp");
hive> select * from hbase_table_emp;
123         emp1        sse
345         emp2        tl
456         emp3        sse
```

Load data into hive external table which will be inserted into Hbase tables as well

All these tables will be visible and queried from Impala shell as well

JAVA
API

Agenda

- Create via Put method
- Read via Get method
- Update via Put method
- Delete via Delete method

JAVA Client API Overview

- HBase is written in Java
  - ✓ No surprise that it has a Java Native API
- Supports programmatic access to Data Manipulation Language (DML)
  - ✓ CRUD operations plus more
- Everything that you can do with HBase Shell and more....
- Java Native API is the fastest way to access HBase

Using Client API

1. Create a Configuration object
   - ✓ Recall Configuration from HDFS object
   - ✓ Adds HBase specific props
2. Construct HTable
   - ✓ Provide Configuration object
   - ✓ Provide table name
3. Perform operations
   - ✓ Such as put, get, scan, delete, etc...
4. Close HTable instance
   - ✓ Flushes all the internal buffers
   - ✓ Releases all the resources

1. Create a Configuration object

```
Configuration conf = HbaseConfiguration.create();
```

2. Construct HTable

```
HTable hTable = new HTable(conf, tableName);
```

3. Perform operations

```
hTable.getTableName();
```

4. Close HTable instance

```
hTable.close();
```

ConstructHTable.java

```
public class ConstructHTable
{
public static void main(String[] args) throws IOException
    {
            Configuration conf = HbaseConfiguration.create();


HTable hTable = new HTable(conf, "-ROOT-");

System.out.println("Table is: " +
            Bytes.toString(hTable.getTableName()));

hTable.close();
    }
}
```

Seeds configuration object with required information to establish client connection

Table name

Release all the resource

ConstructHTable.java Output

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.ConstructHTable
12/01/15 13:22:03 INFO zookeeper.ZooKeeper: Client
environment:zookeeper.version=3.3.3-cdh3u2--1, built on
10/14/2011 03:25 GMT
...
...
...
12/01/15 13:22:03 INFO zookeeper.ClientCnxn: Session
establishment complete on server localhost/127.0.0.1:2181,
sessionid = 0x134e27760560013, negotiated timeout = 40000
Table is: -ROOT-
```

1.Create Configuration Object

- Client Code Configuration

- HbaseConfiguration extends Hadoop's Configuration class
    - ✓ Still fully compatible with Configuration

- How did HbaseConfiguration.create() seed Configuration object?
    - ✓ Loads hbase-default.xml and hbase-site.xml from Java CLASSPATH
        - ❑ hbase-default.xml is packaged inside HBase jar
        - ❑ hbase-site.xml will need to be added to the CLASSPATH
        - ❑ hbase-site.xml overrides properties in hbase-default.xml

- How did hbase-site.xml get on CLASSPATH?
  - ✓ Recall that we executed the code via yarn script

```
$ yarn jar HadoopSamples.jar hbase.ConstructHTable
```

  - ✓ Hadoop’s scripts are configured to put hbase’s CLASSPATH onto it’s CLASSPATH
  - ✓ Specified in <hadoop_install>/conf/hadoop-env.sh

```
export HADOOP_CLASSPATH=
    $HBASE_HOME/*:$HBASE_HOME/conf:$HADOOP_CLASSPATH
```

  - ✓ To check what’s on Hadoop's CLASSPATH
    - ❑ $ yarn classpath
    - ❑ $ yarn classpath | grep hbase

- Creating HTable instance is not free
  - ✓ Actually quite costly – scans catalog .META. Table
    - ❑ Checks that table exists and enabled
  - ✓ Create once (per thread) and re-use for as long as possible
  - ✓ If you find yourself constructing many instances consider using HTablePool (utility to re-use multiple Htable instances)

- HTable is NOT thread safe
  - ✓ Create 1 instance per thread

- HTable supports CRUD batch operations
  - ✓ Not atomic
  - ✓ For performance and convenience

Create / Save Data To HBase

1. Construct HTable instance
   - ✓ Create Put instance

2. Add cell values and their coordinates
   - ✓ Specify family:column as a coordinate

3. Call put on HTable instance

4. Close HTable

1.Construct HTable

- Create Configuration
- Construct HTable

```
Configuration conf = HBaseConfiguration.create();
HTable hTable = new HTable(conf, "HBaseSamples");
```

2.Create Put Instance

- Put is a save operation for a single row
- Must provide a row id to the constructor
  - ✓ Row id is raw bytes: can be anything like number or UUID
    - ❑ You are responsible for converting the id to bytes
    - ❑ HBase comes with a helper class Bytes that provides static methods which handles various conversions from and to bytes
      - ○ org.apache.hadoop.hbase.util.Bytes

```
Put put1 = new Put(Bytes.toBytes("row1"));
```

  - ✓ Optionally can provide cell's timestamp and an instance of RowLock

```
Put put2 = new Put(Bytes.toBytes("row2"), timestamp);
Put put3 = new Put(Bytes.toBytes("row3"), rowLock);
Put put4 = new Put(Bytes.toBytes("row4"), timestamp, rowLock);
```

3.Add Cell Value And Their Coordinates

- Add columns to save to Put instance
    - ✓ Provide family:value coordinate and optional timestamp
    - ✓ Few options of the add methods
        - ❑ Put.add(family, column, value)
        - ❑ Put.add(family, column, timestamp, value)
        - ❑ Put.add(KeyValue kv)

    - ✓ Family, column, and value are raw binary
    - ✓ Client’s responsibility to convert to binary format
    - ✓ KeyValue class as its internal cell’s representation
        - ❑ For advanced usage, not usually required

```
put1.add(toBytes("test"), toBytes("col1"), toBytes("val1"));
put1.add(toBytes("test"), toBytes("col2"), toBytes("val2"));
```

4.Call Put On HTable Instance

- Provide initialized Put object to HTable
- The operation is synchronous

```
...
hTable.put(put1);
```

5.Close HTable

- Release resource held by Htable

- Inform HConnectionManager that this instance won't be using connection

```
hTable.close();
```

- Utilize try/finally block

```
HTable hTable = new HTable(conf, "HBaseSamples");
try {
		// to stuff with table
} finally {
		hTable.close();
}
```

- Most examples emit try/finally constructs in favor of readability

PutExample.java

Static import of Bytes class

```
import static org.apache.hadoop.hbase.util.Bytes.*;
public class PutExample
{
        public static void main(String[] args) throws IOException
        {
                Configuration conf = HBaseConfiguration.create();
                HTable hTable = new HTable(conf, "HBaseSamples");

                Put put1 = new Put(toBytes("row1"));

                put1.add(toBytes("test"), toBytes("col1"), toBytes("val1"));
                put1.add(toBytes("test"), toBytes("col2"), toBytes("val2"));

                hTable.put(put1);

                hTable.close();
        }
}
```

Create put with id “row1”

Add “val1” to test:col1 column
Add “val2” to test:col2 column

Save row to HBase

PutExample.java Output

```
$ yarn jar HadoopSamples.jar hbase.PutExample
$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.4-cdh3u2, r, Thu Oct 13 20:32:26 PDT 2011


hbase(main):001:0> get 'HBaseSamples', 'row1'
ROW        COLUMN+CELL
row1       column=test:col1, timestamp=1326663102473, value=val1
row1       column=test:col2, timestamp=1326663102473, value=val2
1 row(s) in 0.3340 seconds
```

Retrieving Data

- API supports
  - ✓ Get a single row by id
  - ✓ Get a set of rows by a set of row ids
    - ❑ Implemented via batching and will be covered later
  - ✓ Scan an entire table or a sub set of rows
    - ❑ To scan a portion of the table provide start and stop row ids
    - ❑ Recall that row-ids are ordered by raw byte comparison
    - ❑ In case of string based ids, the order is alphabetical
- That's it
  - ✓ Very limited simple API

Retrieving Single Row

1. Construct HTable instance

2. Create Get instance

3. Optionally narrow down result
   - ✓ Specify family:column coordinate
   - ✓ Optionally add filters

4. Request and get results
   - ✓ Call get on HTable
   - ✓ Result instance is returned and will contain the data

5. Close HTable

2.Create Get Instance

- Retrieve a single row
- Construct a Get Instance by providing row id
  - ✓ Row id is in raw binary format
- Optional parameter for a row lock

```
Get get = new Get(toBytes("row1"));
```

3.Optionally Narrow Down The Result

- Only retrieve the data that you need
  - ✓ If not specified then an entire row is retrieved
  - ✓ Important, as HBase allows you to scale to millions of rows
  - ✓ Can narrow down by family, column(s), time range and max versions
  - ✓ Can provide more than one narrow down criteria
  - ✓ Family and column name parameters are in raw bytes

- Narrow down by family
  - ✓ get.addFamily(family)

- Narrow down by column
  - ✓ get.addColumn(family, column)

- Narrow down by time range
    - ✓ `get.setTimeRange(minStamp, maxStamp)`

- Specify number of versions returned
    - ✓ `get.setMaxVersions(maxVersions)`
    - ✓ By default set to 1: only returns the latest version

- Can retrieve multiple families and columns
    - ✓ `get.addFamily(family)`
    - ✓ `get.addFamily(family1)`
    - ✓ `get.addColumn(family2, column1)`
    - ✓ `get.addColumn(family2, column2)`
    - ✓ `get.setTimeRange(minStamp, maxStamp)`

4.Request And Get Results

- Utilize get methods on HTable
  - ✓ Provide assembled Get instance
  - ✓ Returns Result object with all the matching cells

```
Result result = hTable.get(get);
byte [] rowId = result.getRow();
byte [] val1 =
        result.getValue(toBytes("test"), toBytes("col1"));
byte [] val2 =
        result.getValue(toBytes("test"), toBytes("col2"));
```

- Result class
  - ✓ Allows you to access everything returned
  - ✓ Result is NOT Thread safe

- Methods of interest
  - ✓ Result.getRow() - get row's id
  - ✓ Result.getValue(family, column) - get a value for a chosen cell
  - ✓ Result.isEmpty() - true if the result is empty false otherwise
  - ✓ Result.size() - returns number of cells
  - ✓ Result.containsColumn(family:column) true if column exists
  - ✓ There are a number of methods that provide access to underlying KeyValue objects
    - ❑ are for advanced usage and usually not required

GetExamlpe.java

```
public static void main(String[] args) throws IOException
 {
        Configuration conf = HBaseConfiguration.create();
        HTable hTable = new HTable(conf, "HBaseSamples");

        Get get = new Get(toBytes("row1"));
        Result result = hTable.get(get);
        print(result);

        get.addColumn(toBytes("test"), toBytes("col2"));
        result = hTable.get(get);
        print(result);

        hTable.close();
}
```

Get the entire row

Select a single column test:col2

GetExamlpe.java

```
private static void print(Result result)
{
        System.out.println("--------------------------------");
        System.out.println("RowId: " + Bytes.toString(result.getRow()));
```

Retrieve row id

```
        byte [] val1 = result.getValue(toBytes("test"), toBytes("col1"));
        System.out.println("test1:col1="+Bytes.toString(val1));
```

Print value test:col1 column

```
        byte [] val2 = result.getValue(toBytes("test"), toBytes("col2"));
        System.out.println("test1:col2="+Bytes.toString(val2));
}
```

Print value test:col2 column

GetExamlpe.java Output

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.GetExample
...
...
--------------------------------
RowId: row1
test1:col1=val1
test1:col2=val2
--------------------------------
RowId: row1
test1:col1=null
test1:col2=val2
```

test1:col1 wasn't selected the second time

Deleting Data

- Deletes are per-row-basis
- Supports batching
  - ✓ Batching is not atomic, for performance and for convenience
  - ✓ More on that later..

1. Construct HTable instance
2. Create and Initialize Delete
3. Call delete on HTable
   - ✓ htable.delete(delete);
4. Close HTable

2.Create And Initialize Delete

- Construct a Delete instance
  - ✓ Similar to Get or Put
  - ✓ Delete(byte[] row)
    - ❑ Provide a row id to delete/modify
  - ✓ Delete(byte[] row, long timestamp, RowLock rowLock)
    - ❑ Optional timestamp and RowLock
- Optionally narrow down the Deletes

```
Delete delete1 = new Delete(toBytes("anotherRow"));
delete1.deleteColumns(toBytes("family"), toBytes("loan"));
delete1.deleteFamily(toBytes("family"));
```

- Narrow down what to delete for a row
  - ✓ If nothing provided then entire row is deleted

  - ✓ Delete a subset of a row by narrowing down
    - ❑ public Delete deleteFamily(byte[] family)
    - ❑ public Delete deleteColumn(byte[] family, byte[] qualifier)
    - ❑ public Delete deleteColumns(byte[] family, byte[] qualifier)

  - ✓ Notice deleteColumn **VS** deleteColumns
    - ❑ deleteColumns deletes ALL the versions of the cell but deleteColumn only deletes the latest

  - ✓ Most of the methods are overloaded to also take timestamp
    - ❑ Deletes everything on or before the provided timestamp
    - ❑ deleteColumn is an exception where only the exact timestamp match is removed

DeleteExample.java

```
public static void main(String[] args) throws IOException
{
        Configuration conf = HBaseConfiguration.create();
        HTable hTable = new HTable(conf, "HBaseSamples");


        Delete delete = new Delete(toBytes("rowToDelete"));
        hTable.delete(delete);

        Delete delete1 = new Delete(toBytes("anotherRow"));
        delete1.deleteColumns(toBytes("metrics"), toBytes("loan"));
        hTable.delete(delete1);


        hTable.close();
}
```

Delete an entire row

Delete one cell rowId “anotherRow” and column metrics:loan

JAVA ADMIN API

Agenda

- Create Table
- Drop Table

JAVA Admin API

- Just like HTable is for client API HBaseAdmin is for administrative tasks
  - ✓ org.apache.hadoop.hbase.client.HBaseAdmin
- Recall that only Table and Family names have to be pre-defined
  - ✓ Columns can be added/deleted dynamically
  - ✓ HBase scheme roughly equals table definitions and their column families

Create Table And Column Families

1. Construct HBaseAdmin instance

2. Create Table's schema
   - ✓ Represented by HTableDescriptor class
   - ✓ Add column families to table descriptor (HColumnDescriptor)

3. Execute create via HBaseAdmin class

1.Construct HBase Admin Instance

- HbaseAdmin's constructor requires an instance of Configuration object
  - ✓ Similar to HTable
  - ✓ We already know how to do that

```
Configuration conf = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
```

2.Create Table Description

- org.apache.hadoop.hbase.HTableDescriptor
    - ✓ Serves as a container for table name and column families
    - ✓ Most importantly, add one or more column families
        - ❑ org.apache.hadoop.hbase.HColumnDescriptor
        - ❑ HColumnDescriptor serves as a container for column family name, compressions settings, number of versions, in-memory setting, and block size

```
HTableDescriptor table = new HTableDescriptor(toBytes("Table"));
HColumnDescriptor family = new HColumnDescriptor(toBytes("f"));
table.addFamily(family);
HColumnDescriptor family1 = new HColumnDescriptor(toBytes("f1"));
table.addFamily(family1);
```

3.Execute Create Via HBase Admin

- HBaseAdmin creates a table via createTable method
    - ✓ Synchronous operation

```
admin.createTable(table);
```

```
public static void main(String[] args) throws IOException
{
        Configuration conf = HBaseConfiguration.create();
        HBaseAdmin admin = new HBaseAdmin(conf);

        String name = "NewTable";
        byte [] tableName = toBytes(name);

        HTableDescriptor table = new HTableDescriptor(tableName);
        HColumnDescriptor family =
                              new HColumnDescriptor(toBytes("new_family"));
        table.addFamily(family);

        System.out.println("Table "+name+" exist: " +
                                                  admin.tableExists(tableName)) ;
        System.out.println("Creating "+name+" table...");
        admin.createTable(table);
        System.out.println("Table "+name+" exist: " +
                                                  admin.tableExists(tableName)) ;
}
```

Descriptor for
NewTable:new_family

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.CreateTableExample
...
Table NewTable exist: false
Creating NewTable table...
Table NewTable exist: true

$ hbase shell
hbase> describe 'NewTable‘

DESCRIPTION                                                        ENABLED
{NAME => 'NewTable', FAMILIES => [{NAME => 'new_family',           true
BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0',
COMPRESSION => 'NONE', VERSIONS => '3', TTL =>
'2147483647', BLOCKSIZE => '65536', IN_MEMORY => 'false',
BLOCKCACHE => 'true'}]}

1 row(s) in 0.0400 seconds
```

Drop Table

1. Construct HBaseAdmin instance
2. Disable table
    - ✓ Table must be taken offline in order to perform any schema modifications
3. Delete table

```
public static void main(String[] args) throws IOException
 {
        Configuration conf = HBaseConfiguration.create();

        HBaseAdmin admin = new HBaseAdmin(conf);
        byte [] tableName = toBytes("NewTable");

        admin.disableTable(tableName);

        admin.deleteTable(tableName);
}
```

Bytes utility class is imported with ‘static’ keyword:
import static org.apache.hadoop.hbase.util.Bytes.toBytes;

JAVA Client Example

```
package com.test.hbase;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;

import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;

public class HBaseClientExample {
```

```
/**
 * @param args
 */
@SuppressWarnings({ "deprecation", "resource" })
public static void main(String[] args) throws IOException {
Configuration config = HBaseConfiguration.create();
// Create table

HBaseAdmin admin = new ~~HBaseAdmin(config);~~
HTableDescriptor htd = new ~~HTableDescriptor("TestHtable");~~
HColumnDescriptor hcd1 = new HColumnDescriptor("CF1");
HColumnDescriptor hcd2 = new HColumnDescriptor("CF2");
htd.addFamily(hcd1);
htd.addFamily(hcd2);
admin.createTable(htd);

//List down the tables in HBase
byte[] tablename = htd.~~getName();~~
HTableDescriptor[] tables = admin.listTables();
if (tables.length != 1 && Bytes.equals(tablename, tables[0].~~getName())) {~~
//throw new IOException("Failed create of table");
System.out.println("table name is " + tables[0].~~getName());~~
}
```

```
// Run some operations -- a put, a get, and a scan -- against the table.

HTable table = new ~~HTable(config, tablename);~~
byte[] row1 = Bytes.toBytes("row1");
Put p1 = new Put(row1);
byte[] databytes = Bytes.toBytes("CF1");
p1.~~add(databytes, Bytes.toBytes("col1"), Bytes.toBytes("value1"));~~
table.put(p1);
databytes = Bytes.toBytes("CF2");
p1.~~add(databytes, Bytes.toBytes("col2"), Bytes.toBytes("value2"));~~
table.put(p1);


Get g = new Get(row1);
Result result = table.get(g);
System.out.println("Get: " + result);

Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
try {
for (Result scannerResult : scanner) {
System.out.println("Scan: " + scannerResult);
}
} finally {
scanner.close();
}
table.close();
admin.close();
// Drop the table
admin.disableTable(tablename);
admin.deleteTable(tablename);
}
}
```

Hbase & Hive Integration

Any Questions in Your Mind?

Thank You

- Siva Kumar Bhuchipalli
  Inventor of www.hadooptutorial.info
  phone no:
  Address: