Copyright © 2012 - 2015 Hortonworks, Inc. All rights reserved.

The contents of this course and all its lessons and related materials, including handouts to audience members, are Copyright © 2012 - 2015 Hortonworks, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Hortonworks, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Hortonworks, Inc. Hortonworks, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

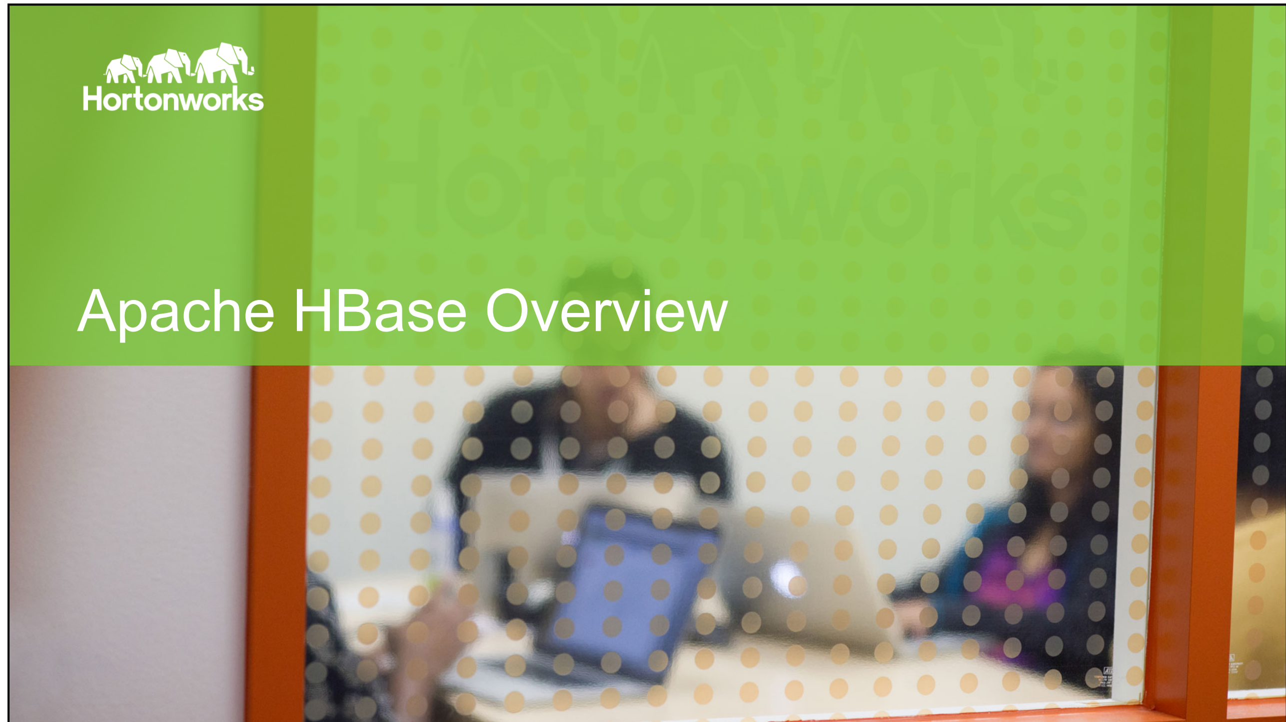
Linux® is the registered trademark of Linus Torvalds in the United States and other Countries.

Java® is a registered trademark of Oracle and/or its affiliates.

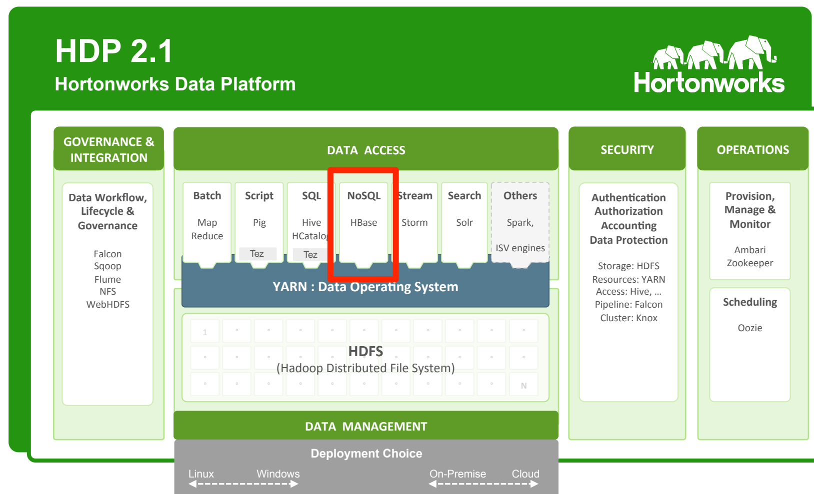
All other trademarks are the property of their respective owners.

 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks



Apache HBase is Part of Data Access in the Hadoop Ecosystem



© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

Learning Objectives

- When you complete this lesson you should be able to:
  - Describe the reason why HBase was created
  - List HBase features
  - List the components in the HBase architecture
  - Describe an HBase table as a set of key-value mappings
  - Identify HBase as either a row- or column-oriented database
  - Describe HBase operation

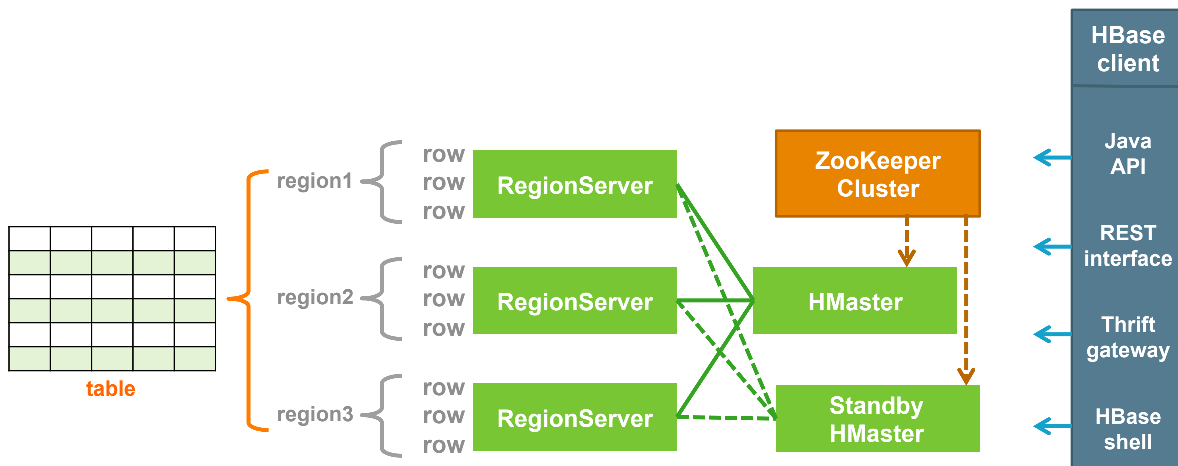
© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Apache HBase

- Apache HBase is a non-relational (NoSQL) database.
  - HBase was created for hosting very large tables with billions of rows and millions of columns.
- Apache HBase:
  - Provides random, real-time data access
  - Allows table inserts, updates, and deletes
  - Runs on top of the Hadoop distributed file system
    - HBase data is automatically replicated by HDFS for higher availability.

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

HBase Architecture



© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Key-Value Mappings

– HBase contains maps of keys and their values.
- key > value
- If you know the key, you can retrieve the value.

– Keys are multi-part.
- (column family name, rowID, column qualifier, timestamp) > value
  – column family name – determines storage properties
    » All data in the same column family is stored together on disk.
  – rowID – used to access data and divide table data into regions
    » Regions are maintained on separate RegionServer nodes.
  – column qualifier – the column name, which is just a label in the multi-part key
    » In any given row, one or more columns might or might not exist.
  – timestamp – used to version the data and support data updates
    » Readers can request any available version of the data.

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Rows and Columns

- Rows and columns are implemented differently than in most relational databases.
  - A multi-part key identifies a *cell* with a value.
  - Because a table is just a set of key>value mappings, a row is nothing more than a logical collection of values that share a rowkey.
  - A column is just an additional label for a value and is included in the multi-part key.
  - Sparse tables are possible because not every cell requires a key>value mapping.

HBase Table Mappings

(key1, Column Family 1, Col 1, timestamp) > column value 1

(key1, Column Family 1, Col 2, timestamp) > column value 2

⋮

HBase Table Conceptual View

	Column Family 1		Column Family 2		
rowkey	Col 1	Col 2	Col 3	Col 4	Col 5
key1	valueA	valueB	valueC	valueD	valueE
key2	valueF	valueG	valueH	valueI	valueJ
key3	valueK	valueL	valueM	valueN	valueO

Page 9 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

HBase is a Column-Oriented Database

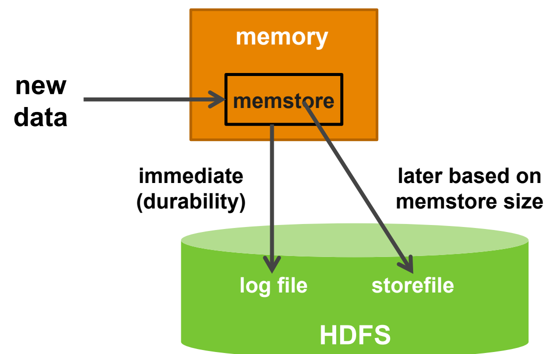
- A column-oriented database stores column items together on disk.
  - A row-oriented relational database stores row items together on disk.
- Column-oriented databases are well suited for:
  - Fast column operations. For example:
    - Calculating the sum or aggregate of an entire column of data
    - Finding the 50 largest items in a column of 2 billion records.
  - Sparse datasets, which are common in big data use cases.

Page 10 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

HBase Operation Overview

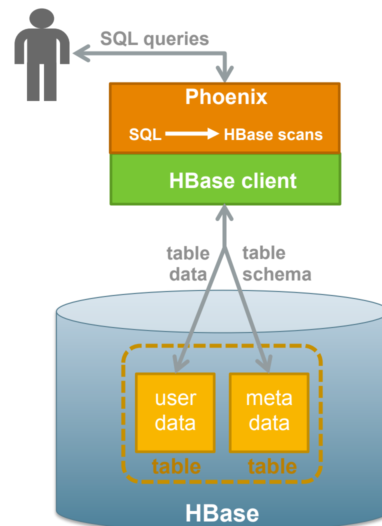
- HBase operations include put, get, delete, and scan.
  - There is no structured query language (SQL).
- Writes initially go to to in-memory memstore.
- Writes are immediately logged to disk for durability.
- Writes are regularly flushed from memstore to a storefile on disk.



Page 11 © Hortonworks Inc. 2011 – 2015. All Rights Reserved Hortonworks

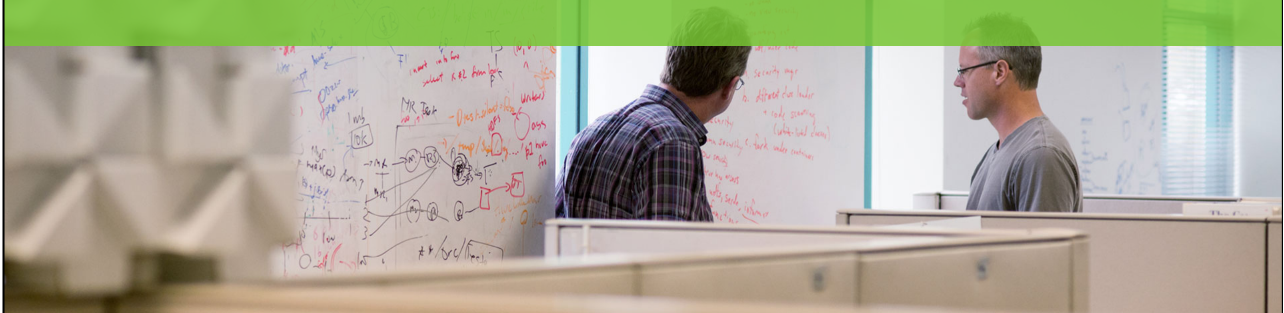
Apache Phoenix

- Apache Phoenix:
  - Is not part of HBase
  - Is a SQL skin over HBase that provides low-latency access to HBase
    - Phoenix enables querying and managing HBase tables using SQL.
    - It compiles your SQL commands into a series of HBase scans.
  - Supports using existing or creating new HBase tables
    - The table metadata that supports SQL-like operation is stored in a companion HBase table.
  - Uses a JDBC driver to provide access to HBase



Page 12 © Hortonworks Inc. 2011 – 2015. All Rights Reserved Hortonworks

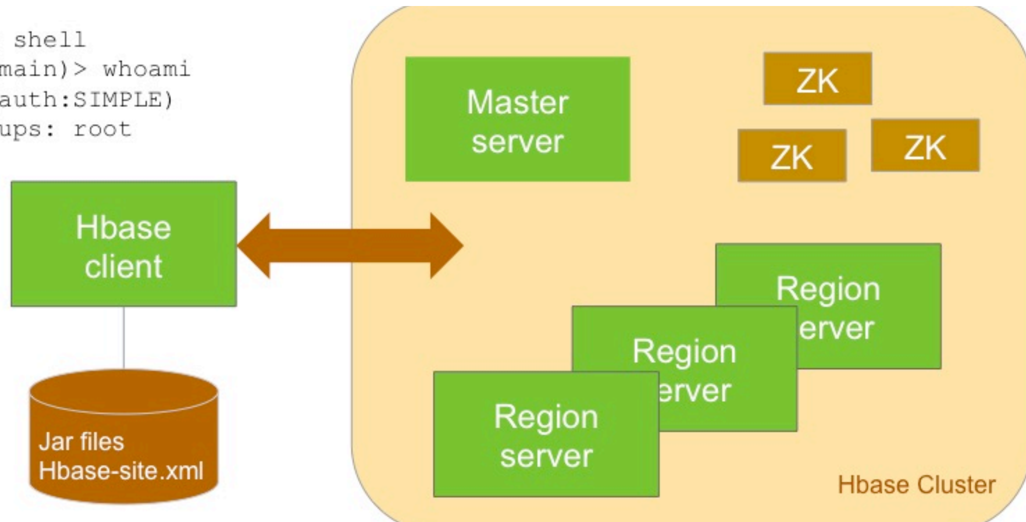
HBase Shell Commands



Invoking the `hbase shell` tool

hbase shell as a client

```
# hbase shell
<hbase(main)> whoami
 root (auth:SIMPLE)
    groups: root
```



Page 15 © Hortonworks Inc. 2011 – 2015. All Rights Reserved Hortonworks

Invoking **hbase shell**

- From within a Linux shell, run **hbase** with '**shell**' as an argument
  - **#hbase shell**
  - must have **hbase** directory in your Linux PATH environment variable
- Opens a subshell with its own command line interpreter
  - Type help to see a detailed list of available commands
  - Take advantage of tab completion

Page 16 © Hortonworks Inc. 2011 – 2015. All Rights Reserved Hortonworks

Command Categories

Hbase shell command categories

- General – high level commands
- DDL – Data Definition Language commands
- DML – Data Manipulation commands
- Tools – Cluster administrator commands
- Replication – Replication administration commands
- Snapshots – Snapshot management
- Security – Authorization Control Lists (ACLs)
- Visibility labels – Manage cell visibility coprocessor configuration

Page 18

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

General Commands

General Commands

- Shows status of the cluster

```
hbase> status
hbase> status 'simple'
Hbase> status 'summary'
hbase> status 'detailed'
```

Output this Hbase version

```
hbase> version
```

Show current Hbase user (taken from the Linux username)

```
hbase> whoami
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Table Management Commands

`create` and `describe`

- Create a table
  – provide at least the table name and column family name

```
hbase> create 't1', {NAME => 'f1', VERSIONS => 5}
hbase> create 't1', {NAME => 'f1'},{NAME => 'f2'},
{NAME => 'f3'}
```

- Describe the named table

```
hbase> describe 't1'
```

Page 22 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

alter

- Alter a table or column family schema
- Provide the table name and a dictionary specifying new columns family schema

```
hbase> alter 't1', NAME => 'f1', VERSIONS => 5
```

- Changes or adds the 'f1' column family in table 't1' from the current value to keep a maximum of 5 cell VERSIONS

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

disable

- Disable the named table

```
hbase> disable 't1'
```

- Disable all tables matching the given regular expression

```
hbase> disable_all 't.*'
```

- Verify the named table is disabled

```
hbase> is_disabled 't1'
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

drop

- Drop named table

```
hbase> drop 't1'
```

- Disable all tables matching the given regular expression

```
hbase> drop_all 't.*'
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

enable

- Enable named table

```
hbase> enable 't1'
```

- Enable all tables matching the given regex

```
hbase> enable_all 't.*'
```

- Verify named table enabled

```
hbase> is_enabled 't1'
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Additional Commands

Query if the named table exists

```
hbase> exists ‘t1’
```

List all tables in HBase (use parameters to filter results)

```
hbase> list ‘t1’
hbase> list ‘abc.*’
```

Show all filters in HBase

```
hbase> show_filters ‘t1’
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

Data Manipulation Commands

Hortonworks

put

- Put a cell value at specified table/row/column and optionally timestamp

```
hbase> put ‘t1’, ‘r1’, ‘c1’, ‘value’, ts1
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

get

- Retrieve row or cell contents
- Can specify table name, row and optionally a dictionary of column(s), timestamp, timerange and versions

```
hbase> get 't1', 'r1'
hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
hbase> get 't1', 'r1', {COLUMN => 'c1'}
hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

scan

- Scan a table
- Pass table name and optionally a dictionary of scanner specs

```
hbase> scan '.META.'
hbase> scan 't1', {COLUMNS => ['c1', 'c2'],
  LIMIT => 10, STARTROW => 'xyz'}
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

delete and deleteall

- Mark a cell for deletion at the specified table/row/column
  - Optionally timestamp coordinates - Must match coordinates exactly

```
hbase> delete 't1', 'r1', 'c1', ts1
```

- Delete all cells in a given row
  - Pass table name, row and optionally column and timestamp

```
hbase> deleteall 't1', 'r1'
hbase> deleteall 't1', 'r1', 'c1'
hbase> deleteall 't1', 'r1', 'c1', ts1
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

count

- Count the number of rows in a table
  - Note: it may take a long time to complete
  - Invokes a MapReduce job
  - Interval parameter specifies how often to display the current count
  - Scan caching is on by default (cache size is in rows)

```
hbase> count ‘t1’
hbase> count ‘t1’, INTERVAL => 100000
hbase> count ‘t1’, CACHE => 1000
hbase> count ‘t1’, INTERVAL => 10, CACHE => 1000
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

get_counter

- Return a counter cell value at the specified table/row/column coordinates

```
hbase> get-counter ‘t1’, ‘r1’, ‘c1’
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

incr

- Increments a cell value at a specified table/row/column coordinates

```
hbase> incr ‘t1’, ‘r1’, ‘c1’
hbase> incr ‘t1’, ‘r1’, ‘c1’, 1
hbase> incr ‘t1’, ‘r1’, ‘c1’, 10
hbase> # increments a cell value in table ‘t1’ at ‘r1’
hbase> # under column ‘c1’ by 1 – or by 10
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

truncate

- Multi-step process to clear all data from a table
- Disables, drops and creates the specified table with the identical schema

```
hbase> truncate ‘t1’
```


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Basic Unit of Storage

- Consider an HBase table as a multi-dimensional map
  - *Tables* contain rows
  - A *row* is a row-key and one or more columns with an associated value
  - A *column* is described as a column family and a column qualifier
    - A *column family* is a group of column qualifiers
    - A *column qualifier* is a unique name for a column within the column family
    - The column is referenced in the format **`<column-family>:<column-qualifier>`**
  - A *cell* is that which is described by a row and column and contains a value and a timestamp
  - A *timestamp* is an identifier of when the value was written and represents a value's *version*
- All of that mapping is contained in a Java object called **`KeyValue`**

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Attributes stored in `KeyValue`

- **`KeyValue`** stores:
  - row – an array of bytes that uniquely identifies a row in an HBase column-family table
  - column family – a preferably short string that identifies a family of column qualifiers
  - column qualifier – an arbitrarily long array of bytes that uniquely identifies a column in a column-family
  - timestamp – An integer that indicates when the key-value was inserted into the table
  - type – indicates the cell type; e.g. **`put`** or **`delete`**
  - sequenceId – a monotonically increasing unique identifier given to each cell in the table to help maintain data consistency in the Memstore.
  - Value – an array of bytes that constitutes the value
- **`KeyValue`** objects are of variable size
  - default allocation of 64kb; however
  - If larger than 64kb will be processed (i.e. read) as a monolithic block.

 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Row keys

- Uniquely identifies a row in a table
- The basis for sorting **`KeyValues`** in Hbase
  - Behaves somewhat like a primary key in RDBMS
- Must be unique
- Row-key design is *extremely* important
  - Impacts performance
  - Impacts region splitting
- An arbitrary array of bytes; not necessarily human-readable
- Examples:
  - **`row-1`**
  - **`12965059333%row-1`**
  - **`9adfeb08cde8418abc0dg8f8ea21de4gf8ab92ecd876efa`**

 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Column Families

- A grouping of columns
- A row spans all column families
- All cells in a column family are stored together in HDFS files
- Number of column families acts as a multiplier on the number of files in HDFS
- Consider 1-3 column families per table
  - Some sources suggest “low 10s” as an upper limit

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Columns grouped in column families

Row key	Column Family 1			Column Family 2		
	Col. 1	Col. 2	Col. 3	Col. A	Col. B	Col. C
Row-1		77		19.99		1000
Row-2	abc		xyz	19.00		2000
Row-3					.2	

Value

table 1, Row-2, Column Camily 2, Col. A => 19.00

Key / Value

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

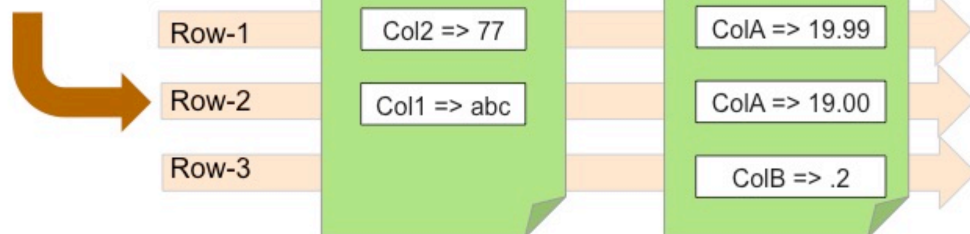
Column Qualifiers

- An arbitrary string of bytes
  - Not necessarily human-readable
- No upper bound to number of column qualifiers in a column family
- Not defined at table creation time
  - Column qualifiers can be added throughout the life of the table
- Possibly difficult to conceptualize boundless column qualifiers when thinking in terms of spreadsheet-style layout
  - Consider all column qualifiers to be stored sequentially in a single column family

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Column qualifiers in column families

Row key	CF1		CF2	
	Col1	Col2	ColA	ColB
Row-1		77	19.99	
Row-2	abc		19.00	
Row-3				.2



© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Timestamps

- By default, generated at server; client can override
- Establishes versioning of cell values

Table 1

Row key	Column Family 1			Column Family 2		
	Col. 1	Col. 2	Col. 3	Col. A	Col. B	Col. C
Row-1		77		19.99		1000
Row-2	abc t=5, def t=17, ghi t=20		xyz	19.00		2000
Row-3					.2 t=1, .1 t=10, .3 t=50	

time

time

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

Lab

Using HBase Shell Commands

Hortonworks

Configuring HBase for High Availability

HBase High Availability

- HBase architecture is designed for *durability* and *availability* of data
  - Durability: Using HDFS as the data store provides multiple replicas of HBase table data
  - Availability: Master server monitors and maintains running Regionservers
- Some vulnerabilities still exist:
  - Failure of the Master server itself
  - Unavailable table data when the failure of a Regionserver requires moving the region to a new Regionserver

Page 48 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Backup Master servers

- A failure of a Master server is survivable for a short period of time
  - Not a component in the client's read or write path
  - Data is still available unless a Regionserver crashes
  - Region/Regionserver assignments not stored in Master server memory
    - Persisted in .META. table
    - Location of .META. table stored in Zookeeper
- A Backup Master server can monitor the health/status of the primary Master and take over Master functions when failure is detected
  - Presence of an ephemeral Zookeeper znode indicates a healthy Master
  - Backup Masters "queue up" for takeover in a Zookeeper znode
- Starting a Backup Master is no different than starting the Primary Master
  - Self-discovery through Zookeeper indicates role of any newly started Master

Page 49 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

Backup Master in Ambari

- Select "Add HBase Master" in Service Actions pulldown menu
- Choose the node for Backup Master
- Start the Backup Master
- View the status of the HBase cluster in the HBase Services screen

Summary | Configs | Quick Links | Service Actions

Summary
Standby HBase Master: Started
Active HBase Master: Started
RegionServers: 3/3 RegionServers Live
Regions In Transition: 0
Master Started: 701.32 secs
Master Activated: 701.32 secs

Service Actions menu: Start, Stop, Restart All, Restart RegionServers, Run Service Check, Turn On Maintenance Mode, Add HBase Master, Download Client Configs

Page 50 © Hortonworks Inc. 2011 – 2015. All Rights Reserved

High Availability reads

- Database theory states that a database can have two of the following three attributes:
  - **C**onsistency – how robust is the database in maintaining data consistency between what is written and what is stored?
  - **A**vailability – how robust is the database in providing access to data stores in the event of failures of components?
  - **P**artition – how survivable is the database when the data store is partitioned – either by design or by failure?

- HBase is inherently **P**, because it's data is distributed across table regions
- HBase writes are highly **C**onsistent within a row, because a single Regionserver manages an entire row
- HBase **A**vailability is problematic when a Regionserver fails

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

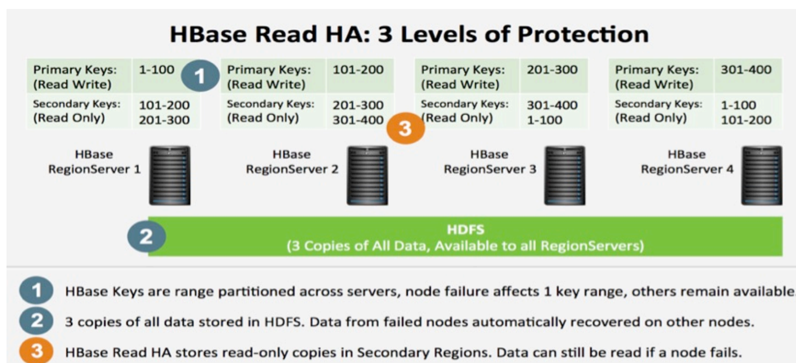
Write consistency

- In reality, HBase is neither fully **C**onsistent nor fully **A**vailable
- Writes are consistent because a client communicates with a single Regionserver to write a row of data
- On the other hand, rows are assigned to regions which are partitioned across Regionservers
  - A client's write of row A may succeed, while row B may fail

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Region replicas

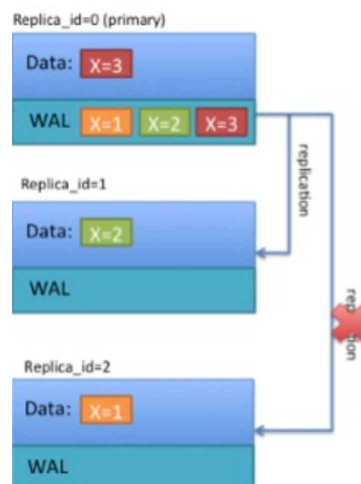
- **A**vailability of a row of table data is directly affected by the health of the Regionserver
- Availability during reads is greatly enhanced by distributed Regionservers, Region Replicas, and HDFS replication



© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Region replication

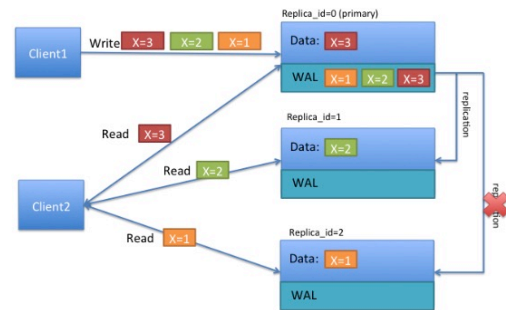
- Region replication is an asynchronous write of new row data to one or more replicas
- At any point in time, the state of the replication is unknown, but in the long term consistency is maintained
- Administrators configure the number of replicas on a per-table basis
- Replication happens out of the primary region's WAL
- Clients opt for:
  - Strong consistency (lower availability)
  - Timeline consistency (higher availabilty)



© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Strong consistency

- When **CONSISTENCY = STRONG**
- Client submits read to Replica ID=0
  - Response only comes from primary
    - Not guaranteed an answer
  - An answer is guaranteed to be consistent
    - Contains a **Stale** flag set to **false**
- If no response comes in 10ms
  - Client submits reads to replica regions
    - An answer is likely
  - **Stale** flag set to **true**

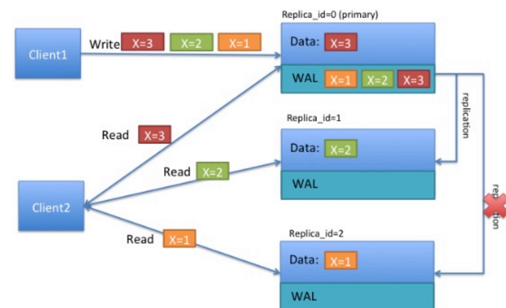


© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

Timeline consistency

- When **CONSISTENCY = TIMELINE**
  - Client submits read to all replica IDs
    - Highly likely to receive an answer
    - Not guaranteed the answer is consistent
- First replica to respond wins
  - If from Primary replica
    - Consistency is assured
    - **Stale** flag set to **false**
  - If from non-primary replica
    - Consistency is unknown
    - **Stale** flag set to **true**



© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks

Configuring region replication

- Region replication is always enabled
  - **REGION_REPLICATION** defaults to 1

- Activated on a per-table basis
  - **REGION_REPLICATION** set to 2 or higher

- Set programmatically in Java clients
- Set in schema in hbase shell
  - `hbase> create 't1', 'cf1', {REGION_REPLICATION => 3}`

© Hortonworks Inc. 2011 – 2015. All Rights Reserved

Hortonworks