HBase Installation & Shell

Originals of slides and source code for examples: http://www.coreservlets.com/hadoop-tutorial/
Also see the customized Hadoop training courses (onsite or at public venues) – http://courses.coreservlets.com/hadoop-training.html

Customized Java EE Training: http://courses.coreservlets.com/
Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live customized Hadoop training (including prep for the Cloudera certification exam), please email info@coreservlets.com

Taught by recognized Hadoop expert who spoke on Hadoop several times at JavaOne, and who uses Hadoop daily in real-world apps. Available at public venues, or customized versions can be held on-site at your organization.

• Courses developed and taught by Marty Hall
  – JSF 2.2, PrimeFaces, Servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  – Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
• Courses developed and taught by coreservlets.com experts (edited by Marty)
  – Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services
Contact info@coreservlets.com for details

© 2012 coreservlets.com and Dima May
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
Set Up 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

1: Verify Installation Requirements

- Latest release of Java 6 from Oracle
- Must have compatible release of Hadoop!
  - runs of top of HDFS
  - Today runs on Hadoop 0.20.x
  - Can run on top of local FS
    - Will lose data when crashes
    - Needs HDFS's durable sync for data fault-tolerance
      - HDFS provides confirmation that the data has been saved
      - Confirmation is provided after all blocks are successfully replicated to all the required nodes
1: Verify Installation Requirements

• SSH installed, sshd must be running
  – Just like Hadoop
  – Need password-less SSH to all the nodes including yourself
  – Required for both pseudo-distributed and fully-distributed modes

• Windows
  – Very little testing – for development only
  – Will need Cygwin

2: Configure Java

• vi <HBASE_HOME>/conf/hbase-env.sh

    export JAVA_HOME=/usr/java/jdk1.6.0
3: Configure the use of HDFS

- **Point to HDFS for its filesystem**
  - Edit `<hbase_home>/conf/hbase-site.xml`
  - `hbase.rootdir` property:
    - Uses HDFS URI
    - Recall URI format: `scheme://namenode/path`
      - Example: `hdfs://localhost:9000/hbase`
    - The location of namenode
directory on HDFS where Region Servers will save it’s data
      - If directory doesn't exist it will be created
  - `dfs.replication` property:
    - The number of times data will be replicated across Region Servers (HLog and HFile)
    - Will set to 1 since there is only 1 host

```
<configuration>
  ...
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:9000/hbase</value>
    <description>The directory shared by RegionServers.</description>
  </property>
  ...
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>The replication count for HLog and HFile storage. Should not be greater than HDFS datanode count.</description>
  </property>
  ...
</configuration>
```

Will this configuration work on a remote client?
3: Configure the use of HDFS

• Since 'localhost' was specified as the location of the namenode remote clients can't use this configuration

...  
  <property>  
    <name>hbase.rootdir</name>  
    <value>hdfs://localhost:9000/hbase</value>  
  </property>
  ...

4: Make sure HDFS is running

• Make sure HDFS is running
  – Easy way is to check web-based management console
    • http://localhost:50070/dfshealth.jsp
  – Or use command line
    • $ hdfs dfs -ls /
5: Start HBase

$ cd <hbase_home>/bin
$ ./start-hbase.sh
starting master, logging to
/home/hadoop/Training/logs/hbase/hbase-hadoop-master-
hadoop-laptop.out
...

6: Verify HBase is 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

$ 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
6: Verify HBase is Running

- By default HBase manages Zookeeper daemon for you
- Logs by default go to `<hbase_home>/logs`
  
  - Change the default by editing `<hbase_home>/conf/hbase-env.sh`
  
    ```
    export HBASE_LOG_DIR=/new/location/logs
    ```

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 Management Console

HBase Shell

- **JRuby IRB (Interactive Ruby Shell)**
  - HBase commands added
  - If you can do it in IRB you can do it in HBase 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"
HBase Shell

• **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

• **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

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'

<table>
<thead>
<tr>
<th>Blog</th>
<th>Family:</th>
<th>info:</th>
<th>Columns: title, author, date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>content:</td>
<td></td>
<td>Columns: post</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Row Id</th>
<th>info:title</th>
<th>info:author</th>
<th>info:date</th>
<th>content:post</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matt-001</td>
<td>Elephant</td>
<td>Matt</td>
<td>2009.05.06</td>
<td>Do elephants like monkeys?</td>
</tr>
<tr>
<td>Matt-002</td>
<td>Monkey</td>
<td>Matt</td>
<td>2011.02.14</td>
<td>Do monkeys like elephants?</td>
</tr>
<tr>
<td>Bob-003</td>
<td>Dog</td>
<td>Bob</td>
<td>1995.10.20</td>
<td>People own dogs!</td>
</tr>
<tr>
<td>Michelle-004</td>
<td>Cat</td>
<td>Michelle</td>
<td>1990.07.06</td>
<td>I have a cat!</td>
</tr>
<tr>
<td>John-005</td>
<td>Mouse</td>
<td>John</td>
<td>2012.01.15</td>
<td>Mickey mouse.</td>
</tr>
</tbody>
</table>

- Put command format:
  `hbase> put 'table', 'row_id', 'family:column', 'value'`
2: Populate Table With Data Records

```java
# 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.'
```

3. 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`
3. Access data - count

```
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
```

Affects how often count is displayed

3. 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}`
3. 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 the columns, displays 1 column per row!!!

3. Access data - get

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 just two columns

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

Narrow down via columns and timestamp

Only one timestamp matches
3. 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
```

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

---

3. 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
3. 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

```bash
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
```
3. Access data - Scan

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

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

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

Only retrieve 'info:title' column
4: 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

```
40 hbase> create 'table', {NAME => 'family', VERSIONS => 7}
```

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

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

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

```
44 hbase> get 'Blog', 'Michelle-004', {COLUMN=>'info:date', VERSIONS=>3}
```

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>CELL</th>
</tr>
</thead>
<tbody>
<tr>
<td>info:date</td>
<td>timestamp=1326071670471, value=1990.07.08</td>
</tr>
<tr>
<td>info:date</td>
<td>timestamp=1326071670442, value=1990.07.07</td>
</tr>
<tr>
<td>info:date</td>
<td>timestamp=1326071670382, value=1990.07.06</td>
</tr>
</tbody>
</table>

3 row(s) in 0.0170 seconds

Update the same exact row with a different value

```
45 hbase> get 'Blog', 'Michelle-004', {COLUMN=>'info:date', VERSIONS=>3}
```

Keeps three versions of each cell by default
4: Edit data

```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

5: Delete records

- **Delete cell by providing table, row id and column coordinates**
  - delete 'table', 'rowId', 'column'
  - Deletes all the versions of that cell

- **Optionally add timestamp to only delete versions before the provided timestamp**
  - delete 'table', 'rowId', 'column', timestamp
5: 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

3 versions

1 millisecond after the second version

After the timestamp provided at delete statement
6: 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....**
Wrap-Up

Summary

• **We learned**
  – How to install HBase in Pseudo-Distributed Mode
  – How to use HBase Shell
  – HBase Shell commands
Questions?

More info:
http://www.coreservlets.com/hadoop-tutorial - Hadoop programming tutorial
http://courses.coreservlets.com/Hadoop-training.html - Customized Hadoop training courses, at public venues or onsite at your organization
http://courses.coreservlets.com/Course-Materials/java.html - General Java programming tutorial
http://www.coreservlets.com/JSF-Tutorial/JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/PrimeFaces/PrimeFaces Tutorial
http://coreservlets.com/ - JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training

Customized Java EE Training: http://courses.coreservlets.com/
Hadoop, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.