# Unleash the Power of HBase Shell

## Big Data Everywhere
## Chicago 2014

Jayesh Thakrar
jthakrar@conversant.com

# HBase = Truly Big Data Store ……..(well, one of many)


- Proven
- Scalable
- Resilient and highly available (HA)
- Low-latency
- OLTP and batch/mapreduce usage
- True big data store
  - billions of rows
  - millions of columns

# However……HBase also makes me……


- No query tools like psql, mysql, etc
- No interactive development tools
- Can't browse Java primitive data in cells, e.g. int, long, double, etc.
- Cell values printed as string if "printable bytes" else bytes printed in "hexadecimal" format

# But I was somewhat wrong……

- HBase shell is a full-fledged jruby shell (jirb)
- It can exploit the strengths of Ruby and Java
- With minimal code can retrieve/view data in any HBase row/cell
- Can possibly use "Ruby-on-Rails" and other frameworks directly interfacing with HBase

# HBase Shell = JRuby Under the Cover

**HBase Shell Ruby Source Code**

$ **cd <HBASE_DIRECTORY>**

$ **find . -name '*.rb' -print**

./bin/get-active-master.rb

./bin/hirb.rb

./bin/region_mover.rb

./bin/region_status.rb

….

./lib/ruby/shell/commands/assign.rb

./lib/ruby/shell/commands/balancer.rb

…

./lib/ruby/shell/commands/create.rb

./lib/ruby/shell/commands/delete.rb

…..

# JRuby under the cover

**HBase Shell Ruby Source Code**

$ **cd <HBASE_DIRECTORY>**

$ **find . -name '*.rb' -print**

./bin/get-active-master.rb

./bin/hirb.rb

./bin/region_mover.rb

./bin/region_status.rb

….

./lib/ruby/shell/commands/assign.rb

./lib/ruby/shell/commands/balancer.rb

…

./lib/ruby/shell/commands/create.rb

./lib/ruby/shell/commands/delete.rb

…..

**Shell Commands = HBase DSL (Domain Specific Language)**

$ **hbase shell**

**get** 'user', 'AB35000000000000350' ← DSL format

**get**('user', 'AB35000000000000350') ← Ruby method format

# JRuby under the cover

**HBase Shell Ruby Source Code**

$ **cd <HBASE_DIRECTORY>**

$ **find . -name '*.rb' -print**

./bin/get-active-master.rb

./bin/hirb.rb

./bin/region_mover.rb

./bin/region_status.rb

….

./lib/ruby/shell/commands/assign.rb

./lib/ruby/shell/commands/balancer.rb

…

./lib/ruby/shell/commands/create.rb

./lib/ruby/shell/commands/delete.rb

…..

**Shell Commands = HBase DSL (Domain Specific Language)**

$ **hbase shell**

**get** 'user', 'AB35000000000000350' ← DSL format

**get**('user', 'AB35000000000000350') ← Ruby method format

table_name = 'user'

rowkey = 'AB35000000000000350'

**get(**table_name, rowkey**)**

# JRuby under the cover

**HBase Shell Ruby Source Code**

$ **cd <HBASE_DIRECTORY>**

$ **find . -name '*.rb' -print**

./bin/get-active-master.rb

./bin/hirb.rb

./bin/region_mover.rb

./bin/region_status.rb

….

./lib/ruby/shell/commands/assign.rb

./lib/ruby/shell/commands/balancer.rb

…

./lib/ruby/shell/commands/create.rb

./lib/ruby/shell/commands/delete.rb

…..

**Shell Commands = HBase DSL (Domain Specific Language)**

$ **hbase shell**

**get** 'user', 'AB35000000000000350' ← DSL format

**get**('user', 'AB35000000000000350') ← Ruby method format

table_name = 'user'

rowkey = 'AB35000000000000350'

**get(**table_name, rowkey**)**

**scan** 'user', {STARTROW => 'AB35000000000000350', LIMIT => 5}

scan_options = {STARTROW => rowkey, LIMIT => 5}

**scan** table_name, scan_options

Defining and using Ruby Hash or Dictionary

# HBase shell JRuby Example - 1

```
include Java
import org.apache.hadoop.hbase. HBaseConfiguration
import org.apache.hadoop.hbase.client.HTable
import org.apache.hadoop.hbase.client.Scan
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.util.Bytes
htable = HTable.new(HBaseConfiguration.new, "sample")
rowkey = Bytes.toBytes("some_rowkey")
get = Get.new(rowkey)
result = htable.get(get)
result.list.collect {|kv|  puts "#{Bytes.toString(kv.getFamily)}:#{Bytes.toString(kv.getQualifier)}"}
```

# HBase shell JRuby Example - 1

```
include Java
import org.apache.hadoop.hbase. HBaseConfiguration
import org.apache.hadoop.hbase.client.HTable
import org.apache.hadoop.hbase.client.Scan
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.util.Bytes
htable = HTable.new(HBaseConfiguration.new, "sample")
rowkey = Bytes.toBytes("some_rowkey")
get = Get.new(rowkey)
result = htable.get(get)
result.list.collect {|kv|  puts "#{Bytes.toString(kv.getFamily)}:#{Bytes.toString(kv.getQualifier)}"}
```

Allow calling Java from within JRuby

# HBase shell JRuby Example - 1

```
include Java
import org.apache.hadoop.hbase. HBaseConfiguration
import org.apache.hadoop.hbase.client.HTable
import org.apache.hadoop.hbase.client.Scan
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.util.Bytes
htable = HTable.new(HBaseConfiguration.new, "sample")
rowkey = Bytes.toBytes("some_rowkey")
get = Get.new(rowkey)
result = htable.get(get)
result.list.collect {|kv|  puts "#{Bytes.toString(kv.getFamily)}:#{Bytes.toString(kv.getQualifier)}"}
```

Allow calling Java from within JRuby

"import" Java classes

# HBase shell JRuby Example - 1

```
include Java
import org.apache.hadoop.hbase. HBaseConfiguration
import org.apache.hadoop.hbase.client.HTable
import org.apache.hadoop.hbase.client.Scan
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.util.Bytes
htable = HTable.new(HBaseConfiguration.new, "sample")
rowkey = Bytes.toBytes("some_rowkey")
get = Get.new(rowkey)
result = htable.get(get)
result.list.collect {|kv|  puts "#{Bytes.toString(kv.getFamily)}:#{Bytes.toString(kv.getQualifier)}"}
```

Allow calling Java from within JRuby

"import" Java classes

Can invoke HBase Java API
Jruby variables for Java class instance and static and instance method output
HTable.new = new Htable() in Java

# HBase shell JRuby Example - 1

```
include Java
import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.client.HTable
import org.apache.hadoop.hbase.client.Scan
import org.apache.hadoop.hbase.client.Get
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.util.Bytes
htable = HTable.new(HBaseConfiguration.new, "sample")
rowkey = Bytes.toBytes("some_rowkey")
get = Get.new(rowkey)
result = htable.get(get)
result.list.collect {|kv|  puts "#{Bytes.toString(kv.getFamily)}:#{Bytes.toString(kv.getQualifier)}"}
```

Allow calling Java from within JRuby

"import" Java classes

Creating Jruby variables for Java class instance and static and instance method output
HTable.new = new Htable() in Java

Ruby expression: "collect" is a Ruby method for list objects. Here it is made available to a Java list thus making available features of both languages transparently and seamlessly.

# HBase shell JRuby Example - 2

```
# Same include and import statements as Example - 1
htable = HTable.new(HBaseConfiguration.new, ".META.")
scanner = htable.getScanner(Scan.new())
tables = {}
scanner.each do |r|
   table_name = Bytes.toString(r.getRow).split(",")[0]
   if not tables.has_key?(table_name)
      tables[table_name] = 0
   end
   tables[table_name] = tables[table_name] + 1
end
tables.keys.each { |t| puts "Table #{t} has #{tables[t]} regions"}
```

This example scans the ".META." to get a count of regions by tables and regionserver

# HBase shell JRuby Example - 2

```
# Same include and import statements as Example - 1
htable = HTable.new(HBaseConfiguration.new, ".META.")
scanner = htable.getScanner(Scan.new())
tables = {}
scanner.each do |r|
   table_name = Bytes.toString(r.getRow).split(",")[0]
   if not tables.has_key?(table_name)
      tables[table_name] = 0
   end
   tables[table_name] = tables[table_name] + 1
end
tables.keys.each { |t| puts "Table #{t} has #{tables[t]} regions"}
```

Empty Ruby hash or dictionary

This example scans the ".META." to get a count of regions by tables and regionserver

# HBase shell JRuby Example - 2

```
# Same include and import statements as Example - 1
htable = HTable.new(HBaseConfiguration.new, ".META.")
scanner = htable.getScanner(Scan.new())
tables = {}
scanner.each do |r|
   table_name = Bytes.toString(r.getRow).split(",")[0]
   if not tables.has_key?(table_name)
      tables[table_name] = 0
   end
   tables[table_name] = tables[table_name] + 1
end
tables.keys.each { |t| puts "Table #{t} has #{tables[t]} regions"}
```

This example scans the ".META.“ to get a count of regions by tables and regionserver

Empty Ruby hash or dictionary

Example of how to iterate through a Java "iterable". Each iteration of scanner gives a "Result" object which is then passed to a code block

# HBase shell JRuby Example - 2

```
# Same include and import statements as Example - 1
htable = HTable.new(HBaseConfiguration.new, ".META.")
scanner = htable.getScanner(Scan.new())
tables = {}
scanner.each do |r|
  table_name = Bytes.toString(r.getRow).split(",")[0]
  if not tables.has_key?(table_name)
    tables[table_name] = 0
  end
  tables[table_name] = tables[table_name] + 1
end
tables.keys.each { |t| puts "Table #{t} has #{tables[t]} regions"}
```

This example scans the ".META." to get a count of regions by tables and regionserver

Empty Ruby hash or dictionary

Example of how to iterate through a Java "iterable". Each iteration of scanner gives a "Result" object which is then passed to a code block

The "code block" can be enclosed by curly braces ({}) or "do" and "end" keywords. Convention is to use {} for single line code blocks and do/end for multi-line code blocks.

# HBase shell JRuby Example - 2

```
# Same include and import statements as Example - 1
htable = HTable.new(HBaseConfiguration.new, ".META.")
scanner = htable.getScanner(Scan.new())
tables = {}
scanner.each do |r|
  table_name = Bytes.toString(r.getRow).split(",")[0]
  if not tables.has_key?(table_name)
    tables[table_name] = 0
  end
  tables[table_name] = tables[table_name] + 1
end
tables.keys.each { |t| puts "Table #{t} has #{tables[t]} regions"}
```

This example scans the ".META." to get a count of regions by tables and regionserver

Empty Ruby hash or dictionary

Example of how to iterate through a Java "iterable".
Each iteration of scanner gives a "Result" object which is then passed to a code block

The "code block" can be enclosed by curly braces ({}) or "do" and "end" keywords. Convention is to use {} for single line code blocks and do/end for multi-line code blocks.

Print region count by table using an iterator that is passed a code block. Compare the code block enclosed in {} v/s “do/end” above

# HBase shell JRuby Example - 3

- See https://github.com/JThakrar/hse
- hbase_shell_extension.rb

```
class HBaseShellExtension
  # Sample class to show the power of jruby under HBase shell.
  # Class methods allow reading a row and making guess about the data type of each column for display.
  #
  # Sample Interactive Usage
  # ------------------------
  # hse = HBaseShellExtension.new
  # hse.read_row?("sample_table", "rowkey")
  # hse.print_row
  # hse.print_column("column_family", "column_name")
  # Bytes.toString(hse.raw_column("column_family", "column_name"))
  #
  # Sample Batch Usage
  # ------------------
  # source 'hbase_shell_extension.rb'
  # hse = HBaseShellExtension.new
  # ARGF.each_line do |line|
  #   parts = line.split()
  #   table_name = parts[0]
  #   rowkey = parts[1]
  #   if hse.read_row?(table_name, rowkey)
  #      hse.print_row
  #   end
  # end
```

# To Conclude…….

- HBase shell
  - is an interactive scripting environment
  - allows mixing of Java and Jruby
  - Is not “recommended” for serious, enterprise/group development that requires automated testing, continuous integration, etc.
- Can use JRuby IDE, provided you add HBase jars using "require"
  e.g. require '<path>/hbase.jar'
- Can also use your custom Java jars in IDE and/or HBase shell
- Can even “compile” your code to generate “jars” from your JRuby scripts for optimal performance and/or to avoid exposing source code