GettingStarted

Table of Contents

Installation and Configuration

You can install a stable release of Hive by downloading a tarball, or you can download the source code and build Hive from that.

Running HiveServer2 and Beeline

Requirements

- Java 1.7
  Note: Hive versions 1.2 onward require Java 1.7 or newer. Hive versions 0.14 to 1.1 work with Java 1.6 as well. Users are strongly advised to start moving to Java 1.8 (see HIVE-8607).
- Hadoop 2.x (preferred), 1.x (not supported by Hive 2.0.0 onward).
  Hive versions up to 0.13 also supported Hadoop 0.20.x, 0.23.x.
- Hive is commonly used in production Linux and Windows environment. Mac is a commonly used development environment. The instructions in this document are applicable to Linux and Mac. Using it on Windows would require slightly different steps.

Installing Hive from a Stable Release

Start by downloading the most recent stable release of Hive from one of the Apache download mirrors (see Hive Releases).

Next you need to unpack the tarball. This will result in the creation of a subdirectory named `hive-x.y.z` (where `x.y.z` is the release number):

```
$ tar -xzvf hive-x.y.z.tar.gz
```

Set the environment variable `HIVE_HOME` to point to the installation directory:

```
$ cd hive-x.y.z
$ export HIVE_HOME={{pwd}}
```

Finally, add `$HIVE_HOME/bin` to your `PATH`:

```
$ export PATH=$HIVE_HOME/bin:$PATH
```

Building Hive from Source

The Hive GIT repository for the most recent Hive code is located here: `git clone https://git-wip-us.apache.org/repos/asf/hive.git` (the master branch).

All release versions are in branches named "branch-0.#" or "branch-1.#" or the upcoming "branch-2.#", with the exception of release 0.8.1 which is in "branch-0.8-r2". Any branches with other names are feature branches for works-in-progress. See Understanding Hive Branches for details.

As of 0.13, Hive is built using Apache Maven.

Compile Hive on master

To build the current Hive code from the master branch:

```
$ git clone https://git-wip-us.apache.org/repos/asf/hive.git
$ cd hive
$ mvn clean package -Pdist
$ cd packaging/target/apache-hive-{version}-SNAPSHOT-bin/apache-hive-{version}-SNAPSHOT-bin
$ ls
LICENSE
NOTICE
README.txt
RELEASE_NOTES.txt
bin/ (all the shell scripts)
lib/ (required jar files)
conf/ (configuration files)
examples/ (sample input and query files)
hcatalog / (hcatalog installation)
scripts / (upgrade scripts for hive-metastore)
```

Here, {version} refers to the current Hive version.

If building Hive source using Maven (mvn), we will refer to the directory "/packaging/target/apache-hive-{version}-SNAPSHOT-bin/apache-hive-{version}-SNAPSHOT-bin" as <install-dir> for the rest of the page.

Compile Hive on branch-1

In branch-1, Hive supports both Hadoop 1.x and 2.x. You will need to specify which version of Hadoop to build against via a Maven profile. To build against Hadoop 1.x use the profile `hadoop-1`; for Hadoop 2.x use `hadoop-2`. For example to build against Hadoop 1.x, the above mvn command becomes:

```
$ mvn clean package -Phadoop-1,dist
```

Compile Hive Prior to 0.13 on Hadoop 0.20

Prior to Hive 0.13, Hive was built using Apache Ant. To build an older version of Hive on Hadoop 0.20:

```
$ svn co http://svn.apache.org/repos/asf/hive/branches/branch-{version} hive
$ cd hive
$ ant clean package
$ cd build/dist
# ls
LICENSE
NOTICE
README.txt
RELEASE_NOTES.txt
bin/ (all the shell scripts)
lib/ (required jar files)
conf/ (configuration files)
examples/ (sample input and query files)
hcatalog / (hcatalog installation)
scripts / (upgrade scripts for hive-metastore)
```

If using Ant, we will refer to the directory "`build/dist`" as `<install-dir>`.

Compile Hive Prior to 0.13 on Hadoop 0.23

To build Hive in Ant against Hadoop 0.23, 2.0.0, or other version, build with the appropriate flag; some examples below:

```
$ ant clean package -Dhadoop.version=0.23.3 -Dhadoop-0.23.version=0.23.3 -Dhadoop.mr.rev=23
$ ant clean package -Dhadoop.version=2.0.0-alpha -Dhadoop-0.23.version=2.0.0-alpha -Dhadoop.mr.rev=23
```

Running Hive

Hive uses Hadoop, so:

- you must have Hadoop in your path OR
- `export HADOOP_HOME=<hadoop-install-dir>`

In addition, you must use below HDFS commands to create `/tmp` and `/user/hive/warehouse` (aka `hive.metastore.warehouse.dir`) and set them `chmod g+w` before you can create a table in Hive.

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir       /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir       /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w   /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w   /user/hive/warehouse
```

You may find it useful, though it's not necessary, to set `HIVE_HOME`:

```
$ export HIVE_HOME=<hive-install-dir>
```

Running Hive CLI

To use the Hive command line interface (CLI) from the shell:

```
$ $HIVE_HOME/bin/hive
```

Running HiveServer2 and Beeline

Starting from Hive 2.1, we need to run the schematool command below as an initialization step. For example, we can use "derby" as db type.

```
$ $HIVE_HOME/bin/schematool -dbType <db type> -initSchema
```

HiveServer2 (introduced in Hive 0.11) has its own CLI called Beeline. HiveCLI is now deprecated in favor of Beeline, as it lacks the multi-user, security, and other capabilities of HiveServer2. To run HiveServer2 and Beeline from shell:

```
$ $HIVE_HOME/bin/hiveserver2

$ $HIVE_HOME/bin/beeline -u jdbc:hive2://$HS2_HOST:$HS2_PORT
```

Beeline is started with the JDBC URL of the HiveServer2, which depends on the address and port where HiveServer2 was started. By default, it will be (localhost:10000), so the address will look like jdbc:hive2://localhost:10000.

Or to start Beeline and HiveServer2 in the same process for testing purpose, for a similar user experience to HiveCLI:

```
$ $HIVE_HOME/bin/beeline -u jdbc:hive2://
```

Running HCatalog

To run the HCatalog server from the shell in Hive release 0.11.0 and later:

```
$ $HIVE_HOME/hcatalog/sbin/hcat_server.sh
```

To use the HCatalog command line interface (CLI) in Hive release 0.11.0 and later:

```
$ $HIVE_HOME/hcatalog/bin/hcat
```

For more information, see HCatalog Installation from Tarball and HCatalog CLI in the HCatalog manual.

Running WebHCat (Templeton)

To run the WebHCat server from the shell in Hive release 0.11.0 and later:

```
$ $HIVE_HOME/hcatalog/sbin/webhcat_server.sh
```

For more information, see WebHCat Installation in the WebHCat manual.

Configuration Management Overview

- Hive by default gets its configuration from `<install-dir>/conf/hive-default.xml`
- The location of the Hive configuration directory can be changed by setting the `HIVE_CONF_DIR` environment variable.
- Configuration variables can be changed by (re-)defining them in `<install-dir>/conf/hive-site.xml`
- Log4j configuration is stored in `<install-dir>/conf/hive-log4j.properties`

- Hive configuration is an overlay on top of Hadoop – it inherits the Hadoop configuration variables by default.

- Hive configuration can be manipulated by:
  - Editing hive-site.xml and defining any desired variables (including Hadoop variables) in it
  - Using the set command (see next section)
  - Invoking Hive (deprecated), Beeline or HiveServer2 using the syntax:
    - `$ bin/hive --hiveconf x1=y1 --hiveconf x2=y2  //this sets the variables x1 and x2 to y1 and y2 respectively`
    - `$ bin/hiveserver2 --hiveconf x1=y1 --hiveconf x2=y2  //this sets server-side variables x1 and x2 to y1 and y2 respectively`
    - `$ bin/beeline --hiveconf x1=y1 --hiveconf x2=y2  //this sets client-side variables x1 and x2 to y1 and y2 respectively.`
  - Setting the `HIVE_OPTS` environment variable to "`--hiveconf x1=y1 --hiveconf x2=y2`" which does the same as above.

Runtime Configuration

- Hive queries are executed using map-reduce queries and, therefore, the behavior of such queries can be controlled by the Hadoop configuration variables.

- The HiveCLI (deprecated) and Beeline command 'SET' can be used to set any Hadoop (or Hive) configuration variable. For example:

  ```
  beeline> SET mapred.job.tracker=myhost.mycompany.com:50030;
  beeline> SET -v;
  ```

  The latter shows all the current settings. Without the `-v` option only the variables that differ from the base Hadoop configuration are displayed.

Hive, Map-Reduce and Local-Mode

Hive compiler generates map-reduce jobs for most queries. These jobs are then submitted to the Map-Reduce cluster indicated by the variable:

```
mapred.job.tracker
```

While this usually points to a map-reduce cluster with multiple nodes, Hadoop also offers a nifty option to run map-reduce jobs locally on the user's workstation. This can be very useful to run queries over small data sets – in such cases local mode execution is usually significantly faster than submitting jobs to a large cluster. Data is accessed transparently from HDFS. Conversely, local mode only runs with one reducer and can be very slow processing larger data sets.

Starting with release 0.7, Hive fully supports local mode execution. To enable this, the user can enable the following option:

```
hive> SET mapreduce.framework.name=local;
```

In addition, `mapred.local.dir` should point to a path that's valid on the local machine (for example `/tmp/<username>/mapred/local`). (Otherwise, the user will get an exception allocating local disk space.)

Starting with release 0.7, Hive also supports a mode to run map-reduce jobs in local-mode automatically. The relevant options are `hive.exec.mode.local.auto`, `hive.exec.mode.local.auto.inputbytes.max`, and `hive.exec.mode.local.auto.tasks.max`:

```
hive> SET hive.exec.mode.local.auto=false;
```

Note that this feature is *disabled* by default. If enabled, Hive analyzes the size of each map-reduce job in a query and may run it locally if the following thresholds are satisfied:

- The total input size of the job is lower than: `hive.exec.mode.local.auto.inputbytes.max` (128MB by default)
- The total number of map-tasks is less than: `hive.exec.mode.local.auto.tasks.max` (4 by default)
- The total number of reduce tasks required is 1 or 0.

So for queries over small data sets, or for queries with multiple map-reduce jobs where the input to subsequent jobs is substantially smaller (because of reduction/filtering in the prior job), jobs may be run locally.

Note that there may be differences in the runtime environment of Hadoop server nodes and the machine running the Hive client (because of different jvm versions or different software libraries). This can cause unexpected behavior/errors while running in local mode. Also note that local mode execution is done in a separate, child jvm (of the Hive client). If the user so wishes, the maximum amount of memory for this child jvm can be controlled via the option h`ive.mapred.local.mem`. By default, it's set to zero, in which case Hive lets Hadoop determine the default memory limits of the child jvm.

Hive Logging

Hive uses log4j for logging. By default logs are not emitted to the console by the CLI. The default logging level is `WARN` for Hive releases prior to 0.13.0. Starting with Hive 0.13.0, the default logging level is `INFO`.

The logs are stored in the directory `/tmp/<user.name>`:

- `/tmp/<user.name>/hive.log`
  Note: In local mode, prior to Hive 0.13.0 the log file name was "`.log`" instead of "`hive.log`". This bug was fixed in release 0.13.0 (see HIVE-5528 and HIVE-5676).

To configure a different log location, set `hive.log.dir` in $HIVE_HOME/conf/hive-log4j.properties. Make sure the directory has the sticky bit set (`chmod 1777 <dir>`).

- `hive.log.dir=<other_location>`

If the user wishes, the logs can be emitted to the console by adding the arguments shown below:

- `bin/hive --hiveconf hive.root.logger=INFO,console  //for HiveCLI (deprecated)`
- `bin/hiveserver2 --hiveconf hive.root.logger=INFO,console`

Alternatively, the user can change the logging level only by using:

- `bin/hive --hiveconf hive.root.logger=INFO,DRFA //for HiveCLI (deprecated)`
- `bin/hiveserver2 --hiveconf hive.root.logger=INFO,DRFA`

Another option for logging is TimeBasedRollingPolicy (applicable for Hive 1.1.0 and above, HIVE-9001) by providing DAILY option as shown below:

- `bin/hive --hiveconf hive.root.logger=INFO,DAILY //for HiveCLI (deprecated)`
- `bin/hiveserver2 --hiveconf hive.root.logger=INFO,DAILY`

Note that setting `hive.root.logger` via the 'set' command does not change logging properties since they are determined at initialization time.

Hive also stores query logs on a per Hive session basis in `/tmp/<user.name>/`, but can be configured in hive-site.xml with the `hive.querylog.location` property. Starting with Hive 1.1.0, EXPLAIN EXTENDED output for queries can be logged at the INFO level by setting the `hive.log.explain.output` property to true.

Logging during Hive execution on a Hadoop cluster is controlled by Hadoop configuration. Usually Hadoop will produce one log file per map and reduce task stored on the cluster machine(s) where the task was executed. The log files can be obtained by clicking through to the Task Details page from the Hadoop JobTracker Web UI.

When using local mode (using `mapreduce.framework.name=local`), Hadoop/Hive execution logs are produced on the client machine itself. Starting with release 0.6 – Hive uses the `hive-exec-log4j.properties` (falling back to `hive-log4j.properties` only if it's missing) to determine where these logs are delivered by default. The default configuration file produces one log file per query executed in local mode and stores it under `/tmp/<user.name>`. The intent of providing a separate configuration file is to enable administrators to centralize execution log capture if desired (on a NFS file server for example). Execution logs are invaluable for debugging run-time errors.

For information about WebHCat errors and logging, see Error Codes and Responses and Log Files in the WebHCat manual.

Error logs are very useful to debug problems. Please send them with any bugs (of which there are many!) to `hive-dev@hadoop.apache.org`.

From Hive 2.1.0 onwards (with HIVE-13027), Hive uses Log4j2's asynchronous logger by default. Setting hive.async.log.enabled to false will disable asynchronous logging and fallback to synchronous logging. Asynchronous logging can give significant performance improvement as logging will be handled in a separate thread that uses the LMAX disruptor queue for buffering log messages. Refer to https://logging.apache.org/log4j/2.x/manual/async.html for benefits and drawbacks.

HiveServer2 Logs

HiveServer2 operation logs are available to clients starting in Hive 0.14. See HiveServer2 Logging for configuration.

Audit Logs

Audit logs are logged from the Hive metastore server for every metastore API invocation.

An audit log has the function and some of the relevant function arguments logged in the metastore log file. It is logged at the INFO level of log4j, so you need to make sure that the logging at the INFO level is enabled (see HIVE-3505). The name of the log entry is "HiveMetaStore.audit".

Audit logs were added in Hive 0.7 for secure client connections (HIVE-1948) and in Hive 0.10 for non-secure connections (HIVE-3277; also see HIVE-2797).

Perf Logger

In order to obtain the performance metrics via the PerfLogger, you need to set DEBUG level logging for the PerfLogger class (HIVE-12675). This can be achieved by setting the following in the log4j properties file.

```
log4j.logger.org.apache.hadoop.hive.ql.log.PerfLogger=DEBUG
```

If the logger level has already been set to DEBUG at root via hive.root.logger, the above setting is not required to see the performance logs.

DDL Operations

The Hive DDL operations are documented in Hive Data Definition Language.

Creating Hive Tables

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
```

creates a table called pokes with two columns, the first being an integer and the other a string.

```
hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

creates a table called invites with two columns and a partition column called ds. The partition column is a virtual column. It is not part of the data itself but is derived from the partition that a particular dataset is loaded into.

By default, tables are assumed to be of text input format and the delimiters are assumed to be ^A(ctrl-a).

Browsing through Tables

```
hive> SHOW TABLES;
```

lists all the tables.

```
hive> SHOW TABLES '.*s';
```

lists all the table that end with 's'. The pattern matching follows Java regular expressions. Check out this link for documentation http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html.

```
hive> DESCRIBE invites;
```

shows the list of columns.

Altering and Dropping Tables

Table names can be changed and columns can be added or replaced:

```
hive> ALTER TABLE events RENAME TO 3koobecaf;
hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');
hive> ALTER TABLE invites REPLACE COLUMNS (foo INT, bar STRING, baz INT COMMENT 'baz replaces new_col2');
```

Note that REPLACE COLUMNS replaces all existing columns and only changes the table's schema, not the data. The table must use a native SerDe. REPLACE COLUMNS can also be used to drop columns from the table's schema:

```
hive> ALTER TABLE invites REPLACE COLUMNS (foo INT COMMENT 'only keep the first column');
```

Dropping tables:

```
hive> DROP TABLE pokes;
```

Metadata Store

Metadata is in an embedded Derby database whose disk storage location is determined by the Hive configuration variable named `javax.jdo.option.ConnectionURL`. By default this location is `./metastore_db` (see `conf/hive-default.xml`).

Right now, in the default configuration, this metadata can only be seen by one user at a time.

Metastore can be stored in any database that is supported by JPOX. The location and the type of the RDBMS can be controlled by the two variables `javax.jdo.option.ConnectionURL` and `javax.jdo.option.ConnectionDriverName`. Refer to JDO (or JPOX) documentation for more details on supported databases. The database schema is defined in JDO metadata annotations file `package.jdo` at `src/contrib/hive/metastore/src/model`.

In the future, the metastore itself can be a standalone server.

If you want to run the metastore as a network server so it can be accessed from multiple nodes, see Hive Using Derby in Server Mode.

DML Operations

The Hive DML operations are documented in Hive Data Manipulation Language.

Loading data from flat files into Hive:

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
```

Loads a file that contains two columns separated by ctrl-a into pokes table. 'LOCAL' signifies that the input file is on the local file system. If 'LOCAL' is omitted then it looks for the file in HDFS.

The keyword 'OVERWRITE' signifies that existing data in the table is deleted. If the 'OVERWRITE' keyword is omitted, data files are appended to existing data sets.

NOTES:

- NO verification of data against the schema is performed by the load command.

- If the file is in hdfs, it is moved into the Hive-controlled file system namespace.
  The root of the Hive directory is specified by the option `hive.metastore.warehouse.dir` in `hive-default.xml`. We advise users to create this directory before trying to create tables via Hive.

```
  hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-
15');
  hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-
08');
```

The two LOAD statements above load data into two different partitions of the table invites. Table invites must be created as partitioned by the key ds for this to succeed.

```
  hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
```

The above command will load data from an HDFS file/directory to the table.
Note that loading data from HDFS will result in moving the file/directory. As a result, the operation is almost instantaneous.

SQL Operations

The Hive query operations are documented in Select.

Example Queries

Some example queries are shown below. They are available in `build/dist/examples/queries`.
More are available in the Hive sources at `ql/src/test/queries/positive`.

SELECTS and FILTERS

```
  hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';
```

selects column 'foo' from all rows of partition `ds=2008-08-15` of the `invites` table. The results are not stored anywhere, but are displayed on the console.

Note that in all the examples that follow, `INSERT` (into a Hive table, local directory or HDFS directory) is optional.

```
  hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='2008-08-15';
```

selects all rows from partition `ds=2008-08-15` of the `invites` table into an HDFS directory. The result data is in files (depending on the number of mappers) in that directory.
NOTE: partition columns if any are selected by the use of *. They can also be specified in the projection clauses.

Partitioned tables must always have a partition selected in the `WHERE` clause of the statement.

```
  hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM pokes a;
```

selects all rows from pokes table into a local directory.

```
  hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a;
  hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a WHERE a.key < 100;
  hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg_3' SELECT a.* FROM events a;
  hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_4' select a.invites, a.pokes FROM profiles a;
  hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT COUNT(*) FROM invites a WHERE a.ds='2008-08-15';
  hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT a.foo, a.bar FROM invites a;
  hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;
```

selects the sum of a column. The avg, min, or max can also be used. Note that for versions of Hive which don't include HIVE-287, you'll need to use `COUNT(1)` in place of `COUNT(*)`.

GROUP BY

```
  hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(*) WHERE a.foo > 0 GROUP BY a.bar;
  hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(*) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;
```

Note that for versions of Hive which don't include HIVE-287, you'll need to use `COUNT(1)` in place of `COUNT(*)`.

JOIN

```
  hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar, t1.foo,
t2.foo;
```

MULTITABLE INSERT

```
  FROM src
  INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100
  INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key < 200
  INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12') SELECT src.key WHERE src.key >= 200 and src.
key < 300
  INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT src.value WHERE src.key >= 300;
```

STREAMING

```
  hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSFORM(a.foo, a.bar) AS (oof, rab) USING '/bin
/cat' WHERE a.ds > '2008-08-09';
```

This streams the data in the map phase through the script `/bin/cat` (like Hadoop streaming).
Similarly – streaming can be used on the reduce side (please see the Hive Tutorial for examples).

Simple Example Use Cases

MovieLens User Ratings

First, create a table with tab-delimited text file format:

```
CREATE TABLE u_data (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

Then, download the data files from **MovieLens 100k** on the GroupLens datasets page (which also has a README.txt file and index of unzipped files):

```
wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
```

or:

```
curl --remote-name http://files.grouplens.org/datasets/movielens/ml-100k.zip
```

Note: If the link to GroupLens datasets does not work, please report it on HIVE-5341 or send a message to the user@hive.apache.org mailing list.

Unzip the data files:

```
unzip ml-100k.zip
```

And load `u.data` into the table that was just created:

```
LOAD DATA LOCAL INPATH '<path>/u.data'
OVERWRITE INTO TABLE u_data;
```

Count the number of rows in table u_data:

```
SELECT COUNT(*) FROM u_data;
```

Note that for older versions of Hive which don't include HIVE-287, you'll need to use COUNT(1) in place of COUNT(*).

Now we can do some complex data analysis on the table `u_data`:

Create `weekday_mapper.py`:

```
import sys
import datetime

for line in sys.stdin:
  line = line.strip()
  userid, movieid, rating, unixtime = line.split('\t')
  weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
  print '\t'.join([userid, movieid, rating, str(weekday)])
```

Use the mapper script:

```
CREATE TABLE u_data_new (
  userid INT,
  movieid INT,
  rating INT,
  weekday INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

add FILE weekday_mapper.py;

INSERT OVERWRITE TABLE u_data_new
SELECT
  TRANSFORM (userid, movieid, rating, unixtime)
  USING 'python weekday_mapper.py'
  AS (userid, movieid, rating, weekday)
FROM u_data;

SELECT weekday, COUNT(*)
FROM u_data_new
GROUP BY weekday;
```

Note that if you're using Hive 0.5.0 or earlier you will need to use `COUNT(1)` in place of `COUNT(*)`.

Apache Weblog Data

The format of Apache weblog is customizable, while most webmasters use the default.
For default Apache weblog, we can create a table with the following command.

More about RegexSerDe can be found here in HIVE-662 and HIVE-1719.

```
CREATE TABLE apachelog (
  host STRING,
  identity STRING,
  user STRING,
  time STRING,
  request STRING,
  status STRING,
  size STRING,
  referer STRING,
  agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "([^]*) ([^]*) ([^]*) (-|\\[^\\]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]
*|\".*\") ([^ \"]*|\".*\"))?"
```

```
)
STORED AS TEXTFILE;
```