Apache Flume™ and Apache Sqoop™ Data Ingestion to Apache™ Hadoop® Clusters on VMware vSphere®

SOLUTION GUIDE

vmware®

Table of Contents

Apache Hadoop Deployment Using VMware vSphere Big Data Extensions 3
Big Data Extensions Overview 3
Data Ingestion with Flume 4
Flume Overview 4
Creation of a Flume Node 5
Configuration of a Flume Node 6
Data Ingestion into HDFS 7
Data Ingestion into HBase. 8
Data Ingestion with Sqoop 10
Sqoop Overview 10
Creation of a Sqoop Node 10
Configuration of a Sqoop Node. 11
Data Ingestion into HDFS 12
Data Ingestion into HBase. 14
Data Ingestion into Hive. 15
Data Export from HDFS 17
Conclusion 19
References 19

Apache Hadoop Deployment Using VMware vSphere Big Data Extensions

The Apache™ Hadoop® software library is a framework that enables the distributed processing of large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines, with each offering local computation and storage. Hadoop is being used by enterprises across verticals for big data analytics, to help make better business decisions based on large data sets.

Through its sponsorship of Project Serengeti, VMware has been investing in making it easier for users to run big data and Hadoop workloads. VMware has introduced Big Data Extensions (BDE) as a commercially supported version of Project Serengeti designed for enterprises seeking VMware support. BDE enables customers to run clustered, scale-out Hadoop applications through VMware vSphere®, delivering all the benefits of virtualization to Hadoop users. BDE provides increased agility through an easy-to-use interface, elastic scaling through the separation of compute and storage resources, and increased reliability and security by leveraging proven vSphere technology. VMware has built BDE to support all major Hadoop distributions and associated Hadoop projects such as Pig, Hive and HBase.

Data ingestion is one of the critical Hadoop workflows. Massive amounts of data must be moved from various sources into Hadoop for analysis. Apache Flume™ and Apache Sqoop™ are two of the data ingestion tools that are commonly used for Hadoop. Flume is a distributed, reliable and available system for efficiently collecting, aggregating and moving data from many different sources to a centralized datastore such as Hadoop Distributed File System (HDFS). Sqoop is a tool designed to transfer data between Hadoop and relational databases. It can import data from a relational database management system (RDBMS) into HDFS, HBase and Hive and then export the data back after transforming it using Hadoop MapReduce.

This solution guide describes how Flume and Sqoop nodes easily can be created and configured on vSphere using the Serengeti™ Hadoop virtual machine template. Detailed procedures are included and simple use cases provided to illustrate data ingestion into the Hadoop cluster deployed by BDE.

Big Data Extensions Overview

Big Data Extensions is a management service that enables users to deploy Hadoop clusters on vSphere. It is a deployment toolkit that leverages the vSphere platform to deploy a highly available Hadoop cluster in minutes—including common Hadoop components such as HDFS, MapReduce, HBase and Hive—on a virtual platform. BDE offers the following key benefits:

- Deploy a Hadoop cluster on vSphere in minutes via the vSphere Web Client GUI
- Employ a fully customizable configuration profile to specify computer, storage and network resources as well as node placement
- Provide better Hadoop manageability and usability, enabling fast and simple cluster scale-out and Hadoop tuning
- Enable separation of data and compute nodes without losing data locality
- Improve Hadoop cluster availability by leveraging VMware vSphere High Availability (vSphere HA), VMware vSphere Fault Tolerance (vSphere FT) and VMware vSphere vMotion®
- Support multiple Hadoop distributions, including Apache Hadoop, Cloudera CDH, Pivotal HD, MapR, Hortonworks Data Platform (HDP) and Intel IDH
- Deploy, manage and use Hadoop remotely using a one-stop–shop command-line interface (CLI) client

The Serengeti virtual appliance runs on top of the vSphere system and includes a Serengeti management server virtual machine and a Hadoop virtual machine template, as shown in Figure 1. The Hadoop virtual machine template includes a base operating system (OS) and an agent. During deployment, the Serengeti management server sends requests to VMware® vCenter™ to clone and reconfigure virtual machines from the template. The agent configures the OS parameters and network configuration, downloads Hadoop software packages from the Serengeti management server, installs Hadoop software and configures Hadoop parameters.

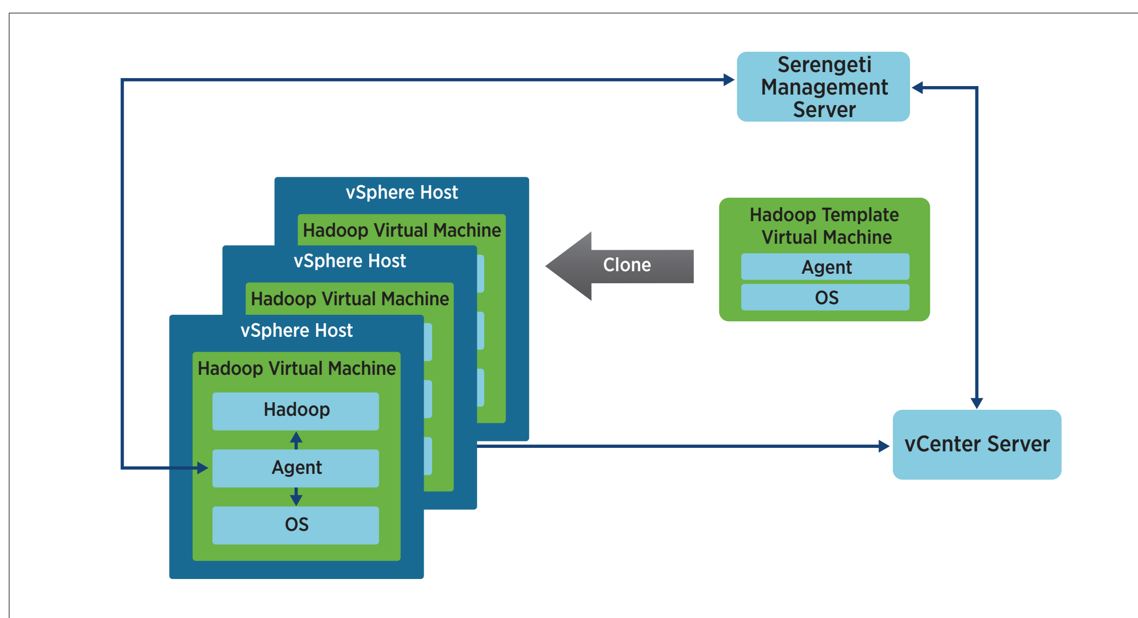

Figure 1. Serengeti Architecture

Data Ingestion with Flume

Flume Overview

A Flume agent is a Java virtual machine (JVM) process that hosts the components through which events flow. Each agent contains at the minimum a source, a channel and a sink. An agent can also run multiple sets of channels and sinks through a flow multiplexer that either replicates or selectively routes an event. Agents can cascade to form a multihop tiered collection topology until the final datastore is reached.

A Flume event is a unit of data flow that contains a payload and an optional set of string attributes. An event is transmitted from its point of origination, normally called a client, to the source of an agent. When the source receives the event, it sends it to a channel that is a transient store for events within the agent. The associated sink can then remove the event from the channel and deliver it to the next agent or the event's final destination. Figure 2 depicts a representative Flume topology.

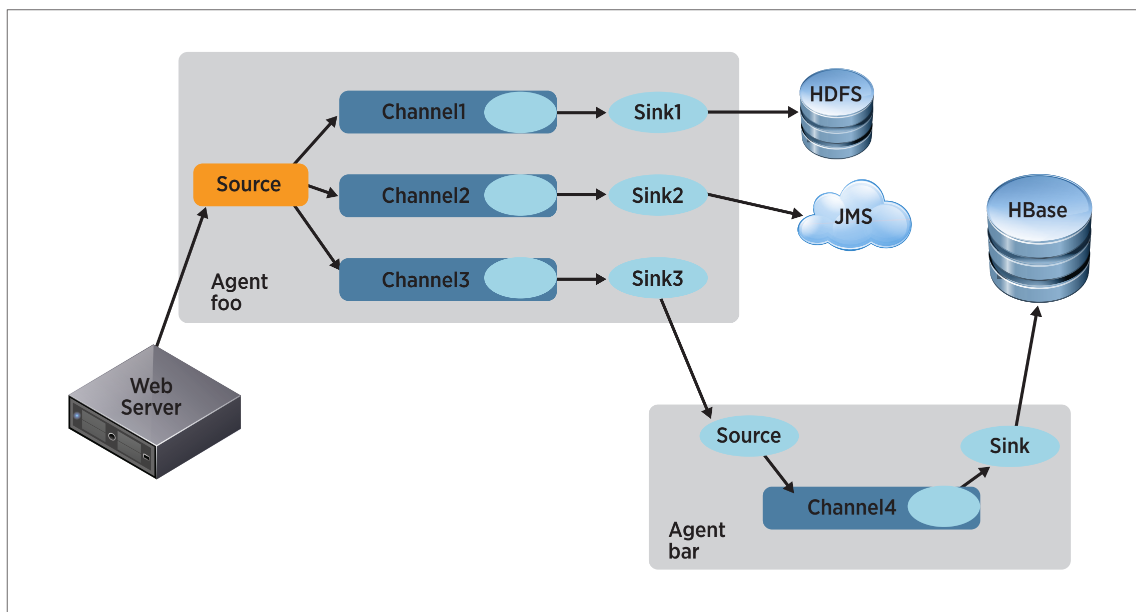

Figure 2. Flume Architecture

Creation of a Flume Node

For a Flume node to be able to sink to HDFS and HBase, the node must have Hadoop client and HBase client installed and configured in addition to Flume. The following procedure easily creates a new Flume node:

1. In vCenter, select the Serengeti virtual appliance.
2. Right-click the **hadoop-template** virtual machine and select **Clone**.
3. Go through the **Clone Virtual Machine** wizard to create a new virtual machine. Details of virtual machine cloning can be found in vSphere product documentation.
4. Power on the newly created virtual machine and enter the virtual machine console.
5. Change to director **/etc/sysconfig/networking/devices** and edit file **ifcfg-eth0** with the following content:

```
ONBOOT=yes
STARTMODE=manual
DEVICE=eth0
BOOTPROTO=dhcp
DHCLIENT_RELEASE_BEFORE_QUIT=yes
DHCLIENT_SET_HOSTNAME=yes
```

If your environment uses fixed IP, edit the file with the following content instead:

```
ONBOOT=yes
DEVICE=eth0
BOOTPROTO=none
IPADDR=<IP address>
NETMASK=<mask address>
GATEWAY=<default gateway address>
USERCTL=no
```

6. Save the file and run the following command to restart the network:

```
# service network restart
```

7. The virtual machine should now have proper network connectivity. Change to the **/opt** directory and download Hadoop and HBase packages from the Serengeti management server:

```
# wget http://<serengeti server>/distros/apache/1.0.1/hadoop-1.0.1.tar.gz
# wget http://<serengeti server>/distros/apache/1.0.1/hbase-0.94.0.tar.gz
```

8. Extract the Hadoop package with the following commands:

```
# tar -zxvf hadoop-1.0.1.tar.gz
# tar –zxvf hbase-0.94.0.tar.gz
```

9. Download the latest Apache Flume package from http://flume.apache.org and extract it under **/opt**:

```
# tar -zxvf apache-flume-1.3.1-bin.tar.gz
```

Instead of cloning from the Serengeti Hadoop template to create a Flume virtual machine node, which provides the best interoperability to the target Hadoop cluster, you can create a new virtual machine that runs a different OS if the OS is supported by both Flume and Hadoop. Due to the large set of possible OS variances and accompanied complexity, corresponding installation procedures are not discussed in this document.

Configuration of a Flume Node

After a Flume node has been created and necessary software packages have been installed, you must perform basic configuration before starting the Flume agent.

1. Create the following environment variables for the diverse software packages:

```
# export HADOOP_COMMON_HOME=/opt/hadoop-1.0.1
# export HADOOP_MAPRED_HOME=/opt/hadoop-1.0.1
# export HBASE_HOME=/opt/hbase-0.94.0
```

Or add the lines to the **~/.bash_profile** file to make the setting permanent for the user.

2. Add the Hadoop, HBase and Flume bin directories to PATH:

```
# export PATH=$PATH:/opt/hadoop-1.0.1/bin:/opt/apache-flume-1.3.1-bin/bin:/opt/hbase-
0.94.0/bin
```

Or add the line to the **~/.bash_profile** file to make the setting permanent for the user.

3. The Flume agent reads the target HDFS information from its configuration file. Therefore, it's optional to set the Hadoop configuration files such as hdfs-site.xml.

4. Copy HBase configuration files from the HBase server to the Flume node:

```
# cd /opt/hbase-0.94.0/conf
# scp root@<HBase Server IP>:/usr/lib/hbase/conf/*
```

Then modify the **hbase-env.sh** file to change the following property so the HBase instance on the Flume node uses the ZooKeeper cluster specified in **hbase-site.xml** instead of managing its own:

```
export HBASE_MANAGES_ZK=false
```

5. Create a Flume configuration file that defines the agent, source, channel and sink that the node will be running. Sample templates can be found under **/opt/apache-flume-1.3.1-bin/conf**.

6. Start the Flume agent:

```
# flume-ng agent -n <agent_name> -c conf -f <configuration_file>
```

<agent_name> is the name of the agent that is defined in the configuration file specified by **<configuration_file>** with full path.

Data Ingestion into HDFS

To illustrate how Flume is used to ingest data into Hadoop HDFS, a simple use case is described in this section. In this use case, the Flume agent receives service requests from a given port on the Flume node and turns each line of text received at the port into an event, which is stored in a memory channel and then retrieved by an HDFS sink to transmit to the target HDFS.

1. Create a configuration file **agent1-conf.properties** with the following sample content and save it under the **/opt/apache-flume-1.3.1-bin/conf** directory. The configuration file defines an agent named **agent1**, a Netcat source named **netcatSrc**, a memory channel named **memoryChannel** and an HDFS sink named **hdfsSink**. Specific parameters for the source, channel and sink can be further tuned:

```
# The configuration file needs to define the sources, the channels and the sinks
# Sources, channels and sinks are defined per agent, in this case called 'agent1'
agent1.sources = netcatSrc
agent1.channels = memoryChannel
agent1.sinks = hdfsSink

# For each one of the sources, the type is defined
agent1.sources.netcatSrc.type = netcat
agent1.sources.netcatSrc.port = 44444
agent1.sources.netcatSrc.bind = <Flume node IP>

# The channel can be defined as follows
agent1.sources.netcatSrc.channels = memoryChannel

# Each sink's type must be defined
agent1.sinks.hdfsSink.type = hdfs
agent1.sinks.hdfsSink.hdfs.path = hdfs://<Name Node IP>/user/root/flume/
agent1.sinks.hdfsSink.hdfs.filePrefix = flume-
agent1.sinks.hdfsSink.hdfs.rollCount = 20
agent1.sinks.hdfsSink.hdfs.batchSize = 20

#Specify the channel the sink should use
agent1.sinks.hdfsSink.channel = memoryChannel

# Each channel's type is defined
agent1.channels.memoryChannel.type = memory
agent1.channels.memoryChannel.capacity = 100
```

2. Start the defined Flume agent:

```
# flume-ng agent -n agent1 -c conf -f /opt/apache-flume-1.3.1-bin/conf/agent1-conf.
properties &
```

3. From another console of the Flume node, or from another host that has the Netcat utility (http://netcat.sourceforge.net/) installed, send text messages to the port from which the Flume agent is receiving service requests:

```
# nc <Flume node IP> 44444
```

Then type in text messages, one line at a time.

4. All event processing messages performed by the source and sink are logged in **/opt/apache-flume-1.3.1-bin/logs/flume.log** on the Flume node.

5. The messages are ultimately transmitted to the target HDFS and written into files in a certain format:

```
# hadoop fs -fs hdfs://<Name Node IP> -ls /user/root/flume

Found 3 items
-rw-r--r--   3 root hadoop        120 2013-05-06 22:58 /user/root/flume/flume-
.1367880981163
-rw-r--r--   3 root hadoop        261 2013-05-07 18:54 /user/root/flume/flume-
.1367952848475
-rw-r--r--   3 root hadoop        311 2013-05-07 18:54 /user/root/flume/flume-
.1367952848476
```

Data Ingestion into HBase

To illustrate how Flume is used to ingest data into HBase, the same Netcat source is used to create events that are transmitted by an HBase sink to the target HBase table.

1. Create an HBase table where Netcat messages will be stored:

```
# hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.0, r1332822, Tue May  1 21:43:54 UTC 2012

hbase(main):002:0> create 'netcat_messages', 'nc'
0 row(s) in 1.3290 seconds
```

2. Create a configuration file **agent2-conf.propertie**s with the following sample content and save it under the **/opt/apache-flume-1.3.1-bin/conf directory**. The configuration file defines an agent named **agent2**, a Netcat source named **netcatSrc**, a memory channel named **memoryChannel** and an HBase sink named **hbaseSink**. Specific parameters for the source, channel and sink can be further tuned:

```
# The configuration file needs to define the sources, the channels and the sinks
# Sources, channels and sinks are defined per agent, in this case called 'agent1'
agent2.sources = netcatSrc
agent2.channels = memoryChannel
agent2.sinks = hbaseSink

# For each one of the sources, the type is defined
agent2.sources.netcatSrc.type = netcat
agent2.sources.netcatSrc.port = 44444
agent2.sources.netcatSrc.bind = <Flume node IP>

# The channel can be defined as follows
agent2.sources.netcatSrc.channels = memoryChannel
```

```
# Each sink's type must be defined
agent2.sinks.hbaseSink.type = org.apache.flume.sink.hbase.HBaseSink
agent2.sinks.hbaseSink.table = netcat_messages
agent2.sinks.hbaseSink.columnFamily = nc
agent2.sinks.hbaseSink.serializer = org.apache.flume.sink.hbase.RegexHbaseEventSerializer

#Specify the channel the sink should use
agent2.sinks.hbaseSink.channel = memoryChannel

# Each channel's type is defined
agent2.channels.memoryChannel.type = memory
agent2.channels.memoryChannel.capacity = 100
```

3. Start the defined Flume agent:

```
# flume-ng agent -n agent2 -c conf -f /opt/apache-flume-1.3.1-bin/conf/agent2-conf.
properties &
```

4. From another console of the Flume node, or from another host that has the Netcat utility (http://netcat.sourceforge.net/) installed, send text messages to the port from which the Flume agent is receiving service requests:

```
# nc <Flume node IP> 44444
```

Then type in text messages, one line at a time.

5. All event processing messages performed by the source and sink are logged in **/opt/apache-flume-1.3.1-bin/logs/flume.log** on the Flume node.
6. The messages are ultimately transmitted to the target HBase table:

hbase(main):003:0> scan 'netcat_messages'	
ROW	COLUMN+CELL
1368047891431-MnxaoIAsRS-0 value=abcde	column=nc:payload, timestamp=1368047894452,
1368047899471-MnxaoIAsRS-1 value=fghij	column=nc:payload, timestamp=1368047902483,
1368047915512-MnxaoIAsRS-2 value=klmno	column=nc:payload, timestamp=1368047918525,
3 row(s) in 0.4140 seconds	

Data Ingestion with Sqoop

Sqoop Overview

Apache Sqoop is a CLI tool designed to transfer data between Hadoop and relational databases. Sqoop can import data from an RDBMS such as MySQL or Oracle Database into HDFS and then export the data back after data has been transformed using MapReduce. Sqoop can also import data into HBase and Hive.

Sqoop connects to an RDBMS through its JDBC connector and relies on the RDBMS to describe the database schema for data to be imported. Both import and export utilize MapReduce, which provides parallel operation as well as fault tolerance. During import, Sqoop reads the table, row by row, into HDFS. Because import is performed in parallel, the output in HDFS is multiple files—delimited text files, binary Avro or SequenceFiles—containing serialized record data.

Creation of a Sqoop Node

For a Sqoop node to be able to import data into HDFS, HBase and Hive, the node must have Hadoop client, HBase client and Hive client, respectively, installed and configured in addition to Sqoop. Follow a procedure similar to that for Flume node creation:

1. In vCenter, select the Serengeti virtual appliance.
2. Right-click the **hadoop-template** virtual machine and select **Clone**.
3. Go through the **Clone Virtual Machine** wizard to create a new virtual machine. Details of virtual machine cloning can be found in vSphere product documentation.
4. Power on the newly created virtual machine and enter the virtual machine console.
5. Change to director **/etc/sysconfig/networking/devices** and edit file **ifcfg-eth0** with the following content:

```
ONBOOT=yes
STARTMODE=manual
DEVICE=eth0
BOOTPROTO=dhcp
DHCLIENT_RELEASE_BEFORE_QUIT=yes
DHCLIENT_SET_HOSTNAME=yes
```

If your environment uses fixed IP, edit the file with the following content instead:

```
ONBOOT=yes
DEVICE=eth0
BOOTPROTO=none
IPADDR=<IP address>
NETMASK=<mask address>
GATEWAY=<default gateway address>
USERCTL=no
```

6. Save the file and run the following command to restart the network:

```
# service network restart
```

7. The virtual machine should now have proper network connectivity. Change to the **/opt** directory and download the following packages from the Serengeti management server:

```
# wget http://<serengeti server>/distros/apache/1.0.1/hadoop-1.0.1.tar.gz
# wget http://<serengeti server>/distros/apache/1.0.1/hbase-0.94.0.tar.gz
# wget http://<serengeti server>/distros/apache/1.0.1/hive-0.8.1.tar.gz
```

8. Extract the packages with the following commands:

```
# tar -zxvf hadoop-1.0.1.tar.gz
# tar –zxvf hbase-0.94.0.tar.gz
# tar –zxvf hive-0.8.1.tar.gz
```

9. Download the latest Apache Sqoop package from http://sqoop.apache.org and extract it under **/opt**.

```
# tar -zxvf sqoop-1.4.3.bin__hadoop-1.0.0.tar.gz
```

Instead of cloning from the Serengeti Hadoop template to create a Sqoop virtual machine node, which provides the best interoperability to the target Hadoop cluster, you can create a new virtual machine that runs a different OS as long as the OS is supported by both Sqoop and Hadoop. Due to the large set of possible OS variances and accompanied complexity, corresponding installation procedures are not discussed in this document.

Configuration of a Sqoop Node

After a Sqoop node has been created and necessary software packages have been installed, you must perform basic configuration before using Sqoop.

1. Create the following environment variables for the various software packages:

```
# export SQOOP_HOME=/opt/sqoop-1.4.3.bin__hadoop-1.0.0
# export HADOOP_COMMON_HOME=/opt/hadoop-1.0.1
# export HADOOP_MAPRED_HOME=/opt/hadoop-1.0.1
# export HBASE_HOME=/opt/hbase-0.94.0
# export HIVE_HOME=/opt/hive-0.8.1
```

Or add the lines to the **~/.bash_profile** file to make the setting permanent for the user.

2. Add the bin directories to PATH:

```
# export PATH=$PATH:/opt/sqoop-1.4.3.bin__hadoop-1.0.0/bin:/opt/hbase-0.94.0/bin:/opt/
hive-0.8.1/bin:/opt/hadoop-1.0.1/bin
```

Or add the line to the **~/.bash_profile** file to make the setting permanent for the user.

3. Copy Hadoop configuration files from NameNode to Sqoop node so Sqoop jobs can be run without specifying the HDFS and JobTracker addresses. This step is optional if you plan to specify HDFS and Job Tracker information directly in Sqoop commands:

```
# cd /opt/hadoop-1.0.1/conf
# scp root@<Name Node IP>:/usr/lib/hadoop/conf/*
```

4. Copy HBase configuration files from the HBase server to the Sqoop node so Sqoop jobs can be run without specifying the HBase Server address:

```
# cd /opt/hbase-0.94.0/conf
# scp root@<HBase Server IP>:/usr/lib/hbase/conf/*
```

Then modify the **hbase-env.sh** file to change the following property so the HBase instance on the Sqoop node uses the ZooKeeper cluster specified in **hbase-site.xml** instead of managing its own:

```
export HBASE_MANAGES_ZK=false
```

Data Ingestion into HDFS

To illustrate how Sqoop is used to ingest data from an RDBMS into HDFS, a simple use case is described in this section. In this use case, a MySQL database named **tpcc** is created with a schema that contains a few tables. One of the tables, **stock**, is used as the source for import; another table, **warehouse**, is used as the target for export. MySQL database installation and configuration is out of the scope of this document.

1. Download the MySQL JDBC driver from http://www.mysql.com/products/connector/ and place the jar file under **/opt/sqoop-1.4.3.bin__hadoop-1.0.0/lib**.

2. Use Sqoop to verify connection to the database:

```
# sqoop list-databases --connect jdbc:mysql://<DB IP>/information_schema --username <DB
User> -P
Enter password:
13/05/07 20:14:54 INFO manager.MySQLManager: Preparing to use a MySQL streaming
resultset
information_schema
mysql
performance_schema
test
tpcc

# sqoop list-tables --connect jdbc:mysql://<DB IP>/tpcc --username <DB User> -P
Enter password:
13/05/07 20:15:28 INFO manager.MySQLManager: Preparing to use a MySQL streaming
resultset
Customer
district
history
item
new_order
order_line
orders
stock
warehouse
```

3. Import data from the **stock** table into the target HDFS. Detailed command output is included here to illustrate usage of MapReduce for the import job. Similar command output is skipped in later sections of this document:

```
# sqoop import --connect jdbc:mysql://<DB IP>/tpcc --table stock --username <DB User> –P
Enter password:
13/05/07 20:23:58 INFO manager.MySQLManager: Preparing to use a MySQL streaming
resultset
13/05/07 20:23:58 INFO tool.CodeGenTool: Beginning code generation
13/05/07 20:23:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM
`stock` AS t LIMIT 1
13/05/07 20:23:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM
`stock` AS t LIMIT 1
13/05/07 20:23:59 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /opt/hadoop-1.0.1
Note: /tmp/sqoop-root/compile/b4666fd4b5896a1103825a18bf26afd6/stock.java uses or
overrides a deprecated API
Note: Recompile with -Xlint:deprecation for details
13/05/07 20:24:03 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/
b4666fd4b5896a1103825a18bf26afd6/stock.jar
```

```
13/05/07 20:24:03 WARN manager.MySQLManager: It looks like you are importing from mysql
13/05/07 20:24:03 WARN manager.MySQLManager: This transfer can be faster! Use the
--direct
13/05/07 20:24:03 WARN manager.MySQLManager: option to exercise a MySQL-specific
fast path
13/05/07 20:24:03 INFO manager.MySQLManager: Setting zero DATETIME behavior to
convertToNull (mysql)
13/05/07 20:24:03 WARN manager.CatalogQueryManager: The table stock contains a multi-
column primary key. Sqoop will default to the column s_i_id only for this job
13/05/07 20:24:03 WARN manager.CatalogQueryManager: The table stock contains a multi-
column primary key. Sqoop will default to the column s_i_id only for this job
13/05/07 20:24:03 INFO mapreduce.ImportJobBase: Beginning import of stock
13/05/07 20:24:05 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`s_i_
id`), MAX(`s_i_id`) FROM `stock`
13/05/07 20:24:07 INFO mapred.JobClient: Running job: job_201304232017_0012
13/05/07 20:24:08 INFO mapred.JobClient:  map 0% reduce 0%
13/05/07 20:24:35 INFO mapred.JobClient:  map 25% reduce 0%
13/05/07 20:25:23 INFO mapred.JobClient:  map 50% reduce 0%
13/05/07 20:27:27 INFO mapred.JobClient:  map 75% reduce 0%
13/05/07 20:28:40 INFO mapred.JobClient:  map 100% reduce 0%
13/05/07 20:28:43 INFO mapred.JobClient: Job complete: job_201304232017_0012
13/05/07 20:28:44 INFO mapred.JobClient: Counters: 18
13/05/07 20:28:44 INFO mapred.JobClient:   Job Counters
13/05/07 20:28:44 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=611001
13/05/07 20:28:44 INFO mapred.JobClient:     Total time spent by all reduces waiting
after reserving slots (ms)=0
13/05/07 20:28:44 INFO mapred.JobClient:     Total time spent by all maps waiting after
reserving slots (ms)=0
13/05/07 20:28:44 INFO mapred.JobClient:     Launched map tasks=4
13/05/07 20:28:44 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/05/07 20:28:44 INFO mapred.JobClient:   File Output Format Counters
13/05/07 20:28:44 INFO mapred.JobClient:     Bytes Written=389680111
13/05/07 20:28:44 INFO mapred.JobClient:   FileSystemCounters
13/05/07 20:28:44 INFO mapred.JobClient:     HDFS_BYTES_READ=454
13/05/07 20:28:44 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=130878
13/05/07 20:28:44 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=389680111
13/05/07 20:28:44 INFO mapred.JobClient:   File Input Format Counters
13/05/07 20:28:44 INFO mapred.JobClient:     Bytes Read=0
13/05/07 20:28:44 INFO mapred.JobClient:   Map-Reduce Framework
13/05/07 20:28:44 INFO mapred.JobClient:     Map input records=1270000
13/05/07 20:28:44 INFO mapred.JobClient:     Physical memory (bytes) snapshot=346566656
13/05/07 20:28:44 INFO mapred.JobClient:     Spilled Records=0
13/05/07 20:28:44 INFO mapred.JobClient:     CPU time spent (ms)=28970
13/05/07 20:28:44 INFO mapred.JobClient:     Total committed heap usage
(bytes)=171180032
13/05/07 20:28:44 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1959284736
13/05/07 20:28:44 INFO mapred.JobClient:     Map output records=1270000
13/05/07 20:28:44 INFO mapred.JobClient:     SPLIT_RAW_BYTES=454
13/05/07 20:28:44 INFO mapreduce.ImportJobBase: Transferred 371.6279 MB in 280.2609
seconds (1.326 MB/sec)
13/05/07 20:28:44 INFO mapreduce.ImportJobBase: Retrieved 1270000 records
```

4. The import is performed leveraging the target Hadoop cluster's MapReduce capability as shown in step 3. After the job finishes, the following directory structure is created in HDFS with files containing the table records:

```
# hadoop fs -ls /user/root/stock
Found 6 items
-rw-r--r--   3 root hadoop          0 2013-05-07 20:28 /user/root/stock/_SUCCESS
drwxr-xr-x   - root hadoop          0 2013-05-07 20:24 /user/root/stock/_logs
-rw-r--r--   3 root hadoop  113372274 2013-05-07 20:24 /user/root/stock/part-m-00000
-rw-r--r--   3 root hadoop   92103524 2013-05-07 20:24 /user/root/stock/part-m-00001
-rw-r--r--   3 root hadoop   92100532 2013-05-07 20:24 /user/root/stock/part-m-00002
-rw-r--r--   3 root hadoop   92103781 2013-05-07 20:24 /user/root/stock/part-m-00003
```

Data Ingestion into HBase

To illustrate how Sqoop ingests data from an RDBMS into HBase, the same MySQL database is used.

1. Verify that the HBase database does not contain a **stock** table:

```
# hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.0, r1332822, Tue May  1 21:43:54 UTC 2012

hbase(main):001:0> list
TABLE
0 row(s) in 1.1080 seconds

# hadoop fs -ls /hadoop/hbase
Found 8 items
drwxrwxr-x   - hbase hadoop         0 2013-04-23 20:18 /hadoop/hbase/-ROOT-
drwxrwxr-x   - hbase hadoop         0 2013-04-23 20:18 /hadoop/hbase/.META.
drwxr-xr-x   - hbase hadoop         0 2013-04-25 17:57 /hadoop/hbase/.corrupt
drwxrwxr-x   - hbase hadoop         0 2013-05-07 20:26 /hadoop/hbase/.logs
drwxr-xr-x   - hbase hadoop         0 2013-05-07 20:27 /hadoop/hbase/.oldlogs
-rw-r--r--   3 hbase hado          38 2013-04-23 20:18 /hadoop/hbase/hbase.id
-rw-r--r--   3 hbase hadoop         3 2013-04-23 20:18 /hadoop/hbase/hbase.version
drwxrwxr-x   - hbase hadoop         0 2013-04-25 18:03 /hadoop/hbase/splitlog
```

2. Import data from the MySQL **stock** table into the target HBase database:

```
# sqoop import --hbase-create-table --hbase-table stock --column-family info --hbase-
row-key s_i_id  --connect jdbc:mysql://<DB IP>/tpcc --table stock --username
<DB User> -P
```

3. The import is performed leveraging the target Hadoop cluster's MapReduce capability. After the job has finished, a table named **stock** is created in HBase and the associated directory structure created in HDFS:

```
# hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.0, r1332822, Tue May 1 21:43:54 UTC 2012

hbase(main):001:0> list
TABLE
stock
1 row(s) in 1.0190 seconds
```

```
hbase(main):002:0> describe 'stock'
DESCRIPTION
ENABLED
 {NAME => 'stock', FAMILIES => [{NAME => 'info', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER true => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION
=> 'NONE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false',
BLOCKSIZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE =>
'true'}]}
1 row(s) in 0.1440 seconds

# hadoop fs -ls /hadoop/hbase/stock
Found 4 items
-rw-r--r--   3 hbase hadoop        691 2013-05-07 20:55 /hadoop/hbase/stock/.
tableinfo.0000000001
drwxrwxr-x   - hbase hadoop          0 2013-05-07 20:55 /hadoop/hbase/stock/.tmp
drwxr-xr-x   - hbase hadoop          0 2013-05-07 20:57 /hadoop/hbase/stock/7513151f7f26
8226280f0ef220f6c372
drwxr-xr-x   - hbase hadoop          0 2013-05-07 20:56 /hadoop/hbase/stock/81d1b1e424cc
d7851a6c77ba75e8d3f6
```

Data Ingestion into Hive

To illustrate how Sqoop ingests data from an RDBMS into Hive, the same MySQL database is used.

1. Verify that the Hive database does not contain a **stock** table:

```
# hive
Logging initialized using configuration in jar:file:/opt/hive-0.8.1/lib/hive-common-
0.8.1.jar!/hive-log4j.properties
Hive history file=/tmp/root/hive_job_log_root_201305072121_164141636.txt

hive> show tables;
OK
Time taken: 12.119 seconds

# hadoop fs -ls /user/hive/warehouse
```

2. Import data from the MySQL **stock** table into the target Hive database:

```
# sqoop import --hive-import --create-hive-table --hive-table stock --connect
jdbc:mysql://<DB IP>/tpcc --table stock --username <DB User> -P
```

3. The import is performed leveraging the target Hadoop cluster's MapReduce capability. After the job has finished, a table named **stock** is created in Hive and the associated directory structure created in HDFS:

```
# hive
Logging initialized using configuration in jar:file:/opt/hive-0.8.1/lib/hive-common-
0.8.1.jar!/hive-log4j.properties
Hive history file=/tmp/root/hive_job_log_root_201305072152_693417704.txt

hive> show tables;
OK
stock
Time taken: 10.695 seconds
```

```
hive> describe stock;
OK
s_i_id  int
s_w_id  int
s_quantity     int
s_dist_01      string
s_dist_02      string
s_dist_03      string
s_dist_04      string
s_dist_05      string
s_dist_06      string
s_dist_07      string
s_dist_08      string
s_dist_09      string
s_dist_10      string
s_ytd  bigint
s_order_cnt    int
s_remote_cnt   int
s_data  string
Time taken: 0.585 seconds

# hadoop fs -ls /user/hive/warehouse/stock
Found 5 items
-rw-r--r--   3 root hadoop          0 2013-05-07 21:47 /user/hive/warehouse/stock/_
SUCCESS
-rw-r--r--   3 root hadoop  113372274 2013-05-07 21:46 /user/hive/warehouse/stock/part-
m-00000
-rw-r--r--   3 root hadoop   92103524 2013-05-07 21:46 /user/hive/warehouse/stock/part-
m-00001
-rw-r--r--   3 root hadoop   92100532 2013-05-07 21:46 /user/hive/warehouse/stock/part-
m-00002
-rw-r--r--   3 root hadoop   92103781 2013-05-07 21:46 /user/hive/warehouse/stock/part-
m-00003
```

4. If the MySQL **stock** table has been imported into HDFS previously to the Hive import, the import will fail because Hive will report that the directory structure for **stock** already exists in HDFS. In this case, you must create the logical table in Hive database and load data into the table from the existing files in HDFS. After loading has been completed, the files will be moved from their original location to the Hive-managed location:

```
# hadoop fs -ls /user/root/stock
Found 6 items
-rw-r--r--   3 root hadoop          0 2013-05-07 20:28 /user/root/stock/_SUCCESS
drwxr-xr-x   - root hadoop          0 2013-05-07 20:24 /user/root/stock/_logs
-rw-r--r--   3 root hadoop  113372274 2013-05-07 20:24 /user/root/stock/part-m-00000
-rw-r--r--   3 root hadoop   92103524 2013-05-07 20:24 /user/root/stock/part-m-00001
-rw-r--r--   3 root hadoop   92100532 2013-05-07 20:24 /user/root/stock/part-m-00002
-rw-r--r--   3 root hadoop   92103781 2013-05-07 20:24 /user/root/stock/part-m-00003

# hive
Logging initialized using configuration in jar:file:/opt/hive-0.8.1/lib/hive-common-
0.8.1.jar!/hive-log4j.properties
Hive history file=/tmp/root/hive_job_log_root_201305072205_1155559963.txt
```

```
hive> create table stock (s_i_id int, s_w_id int, s_quantity int, s_dist_01 string,
s_dist_02 string, s_dist_03 string, s_dist_04 string, s_dist_05 string, s_dist_06
string, s_dist_07 string, s_dist_08 string, s_dist_09 string, s_dist_10 string, s_ytd
int, s_order_cnt int, s_remote_cnt int, s_data int);
OK
Time taken: 8.664 seconds

hive> load data inpath '/user/root/stock/part-m-00000' into table stock;
Loading data to table default.stock
OK
Time taken: 0.526 seconds

hive> load data inpath '/user/root/stock/part-m-00001' into table stock;
Loading data to table default.stock
OK
Time taken: 0.225 seconds

hive> load data inpath '/user/root/stock/part-m-00002' into table stock;
Loading data to table default.stock
OK
Time taken: 0.223 seconds

hive> load data inpath '/user/root/stock/part-m-00003' into table stock;
Loading data to table default.stock
OK
Time taken: 0.273 seconds

# hadoop fs -ls /user/root/stock
Found 2 items
-rw-r--r--   3 root hadoop          0 2013-05-07 22:03 /user/root/stock/_SUCCESS
drwxr-xr-x   - root hadoop          0 2013-05-07 22:02 /user/root/stock/_logs

# hadoop fs -ls /user/hive/warehouse/stock
Found 4 items
-rw-r--r--   3 root hadoop  113372274 2013-05-07 22:02 /user/hive/warehouse/stock/part-
m-00000
-rw-r--r--   3 root hadoop   92103524 2013-05-07 22:02 /user/hive/warehouse/stock/part-
m-00001
-rw-r--r--   3 root hadoop   92100532 2013-05-07 22:02 /user/hive/warehouse/stock/part-
m-00002
-rw-r--r--   3 root hadoop   92103781 2013-05-07 22:02 /user/hive/warehouse/stock/part-
m-00003
```

Data Export from HDFS

Sqoop can also be used to export a set of files from HDFS back to an RDBMS. The target table must already exist in the database. The input files will be read and parsed into a set of database records according to the user-specified delimiters. To illustrate how Sqoop exports data, the same MySQL database is used.

1. Import the **warehouse** table of the **tpcc** database into HDFS:

```
# hadoop fs -ls /user/root/warehouse
Found 6 items
-rw-r--r--   3 root hadoop          0 2013-05-08 15:06 /user/root/warehouse/_SUCCESS
drwxr-xr-x   - root hadoop          0 2013-05-08 15:06 /user/root/warehouse/_logs
-rw-r--r--   3 root hadoop        438 2013-05-08 15:06 /user/root/warehouse/part-m-00000
-rw-r--r--   3 root hadoop        538 2013-05-08 15:06 /user/root/warehouse/part-m-00001
-rw-r--r--   3 root hadoop        567 2013-05-08 15:06 /user/root/warehouse/part-m-00002
-rw-r--r--   3 root hadoop        429 2013-05-08 15:06 /user/root/warehouse/part-m-00003
```

2. Delete all records from the **warehouse** table:

```
mysql> select count(*) from warehouse;
+----------+
| count(*) |
+----------+
|       22 |
+----------+
1 row in set (0.00 sec)

mysql> truncate table warehouse;
Query OK, 0 rows affected (0.01 sec)

mysql> select count(*) from warehouse;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

3. Export data from HDFS back into the **warehouse** table. By default, HDFS stores data imported from an RDBMS using comma as the field delimiter and \n as the row delimiter:

```
# sqoop export --connect jdbc:mysql://<DB IP>/tpcc --table warehouse --export-dir /user/
root/warehouse --username <DB User> -P -input-fields-terminated-by ','
```

There is no special syntax for exporting data from a Hive table back to an RDBMS table. Point the export directory to the Hive-underlying HDFS directory. By default, Hive stores data using ^A (octal representation \0001) as the field delimiter and \n as the row delimiter:

```
# sqoop export --connect jdbc:mysql://<DB IP>/tpcc --table warehouse --export-dir /user/
hive/warehouse/warehouse --username <DB User> -P -input-fields-terminated-by '\001'
```

Currently an HBase table cannot be exported to an RDBMS directly using any tools. Use the HBase native export feature to dump the table into a flat file in HDFS, which can then be exported using Sqoop.

4. Verify that all records have been restored in the table:

```
mysql> select count(*) from warehouse;
+----------+
| count(*) |
+----------+
|       22 |
+----------+
1 row in set (0.00 sec)
```

Conclusion

An Apache Hadoop cluster deployed on VMware vSphere can leverage advanced vSphere HA, vSphere FT and vSphere vMotion features for enhanced availability by using shared storage, while also preserving data locality by using local storage for data nodes. Virtualization enables data and compute separation without compromising data locality. Big Data Extensions simplifies Hadoop deployment on vSphere, accelerates deployment speed, and masks the complexity from the vSphere administrator.

The existing Hadoop ecosystem components and tools seamlessly work with a Hadoop cluster deployed on vSphere, without detecting that the cluster is running on a virtual infrastructure. As demonstrated in this solution guide, two of the most common data ingestion tools, Apache Flume and Apache Sqoop, can be used to ingest data into Hadoop HDFS, HBase and Hive that are deployed by Big Data Extensions. In fact, it is simpler and faster to deploy and scale out Flume and Sqoop nodes alongside the Hadoop nodes on vSphere.

References

1. VMware vSphere Product Documentation Center
   https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-pubs.html
2. Apache Hadoop on VMware vSphere
   http://www.vmware.com/hadoop
3. Serengeti
   http://serengeti.cloudfoundry.com/
4. Apache Hadoop
   http://hadoop.apache.org/
5. Apache Flume
   http://flume.apache.org/
6. Apache Sqoop
   http://sqoop.apache.org/
7. Apache HBase
   http://hbase.apache.org/
8. Apache Hive
   http://hive.apache.org/

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com
Copyright © 2013 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at http://www.vmware.com/go/patents. VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies. Item No: VMW-SB-DATA-INGSTION-vSPHR-USLET-101 Docsouce: OIC-13VM005.02