Implementing Real-Time Analysis with Hadoop in Azure HDInsight

Lab 1 - Getting Started with HBase

Overview

In this lab, you will provision an HDInsight HBase cluster. You will then create an HBase table and use it to store data.

What You'll Need

To complete the labs, you will need the following:

- A web browser
- A Microsoft account
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- The lab files for this course

Note: To set up the required environment for the lab, follow the instructions in the **Setup** document for this course. Specifically, you must have signed up for an Azure subscription.

Provisioning an HDInsight HBase Cluster

The first task you must perform is to provision an HDInsight HBase cluster.

Note: The Microsoft Azure portal is continually improved in response to customer feedback. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing, but may not match the latest design of the portal exactly.

Provision an HDInsight Cluster

1. In a web browser, navigate to http://portal.azure.com, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, in the Hub Menu, click **New**. Then in the **Data + Analytics** menu, click **HDInsight**.
3. Create a new HDInsight cluster with the following settings:
    - **Cluster Name**: *Enter a unique name (and make a note of it!)*
    - **Subscription**: *Select your Azure subscription*
    - **Cluster type**:

        - **Cluster Type**: Hbase
        - **Operating System**: Linux
        - **Version**: *Choose the latest version of HBase available.*
        - **Cluster Tier**: Standard
    - **Cluster Login Username:** *Enter a user name of your choice (and make a note of it!)*
    - **Cluster Login Password:** *Enter a strong password (and make a note of it!)*
    - **SSH Username:** *Enter another user name of your choice (and make a note of it!)*
    - **SSH Password:** *Use the same password as the cluster login password*
    - **Resource Group:**
        - **Create a new resource group**: *Enter a unique name (and make a note of it!)*
    - **Location**: *Choose any available data center location.*
    - **Storage:**
        - **Primary storage type**: Azure Storage
        - **Selection Method**: My Subscriptions
        - **Create a new storage account**: *Enter a unique name consisting of lower-case letters and numbers only (and make a note of it!)*
        - **Default Container**: *Enter the cluster name you specified previously*
    - **Applications**: *None*
    - **Cluster size:**
        - **Number of Region nodes**: 1
        - **Region node size**: *Leave the default size selected*
        - **Head node size**: *Leave the default size selected*
        - **Zookeeper node size**: *Leave the default size selected*
    - **Advanced Settings:** *None*
4. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the cluster to be deployed (this can take a long time – often 30 minutes or more. Now may be a good time to go and have a cup of coffee!)

Note: As soon as an HDInsight cluster is running, the credit in your Azure subscription will start to be charged. Free-trial subscriptions include a limited amount of credit limit that you can spend over a period of 30 days, which should be enough to complete the labs in this course as long as clusters are deleted when not in use. If you decide not to complete this lab, follow the instructions in the *Clean Up* procedure at the end of the lab to delete your cluster to avoid using your Azure credit unnecessarily.

Get Cluster Host Details

1. After the cluster has been provisioned, its blade should be open in the portal (if not, browse to your cluster through the **All resources** menu).
2. In the blade for your cluster, click **Dashboard**, and when prompted, sign in using the HTTP user credentials you specified when provisioning the cluster.
3. In the Ambari dashboard, click **HBase**, and verify that there are no alerts displayed.
4. At the top of the page, click **Hosts** to see the hosts in your cluster. These should include:
    - Two head nodes (prefixed ***hn***)
    - Three zookeeper nodes (prefixed (***zk***)
    - One worker node (prefixed ***wn***)
5. Click any of the hosts to view the fully qualified name, which should be similar to this:

    hn0-<*first 6 characters of cluster name*>.xxxxxxxxxxxxxxxxxxxxxxxx.xx.internal.cloudapp.net

6. The other hosts have names in the same format – only the prefixes should vary to indicate the type and number of each host.

7. Make a note of all of the zookeeper (***zk***) host names in your cluster, being careful to note the correct numbers (they may not be named *zk1*, *zk2*, and *zk3* as you might expect!). You will need these host names for some tasks later in this lab.
8. In the **Services** menu, click **HBase** to return to the HBase page.
9. Click the **Quick Links** drop-down list so observe the status of the zookeeper nodes – two should be *standby* nodes and one should be *active*. Note the active one.

Creating an HBase Table

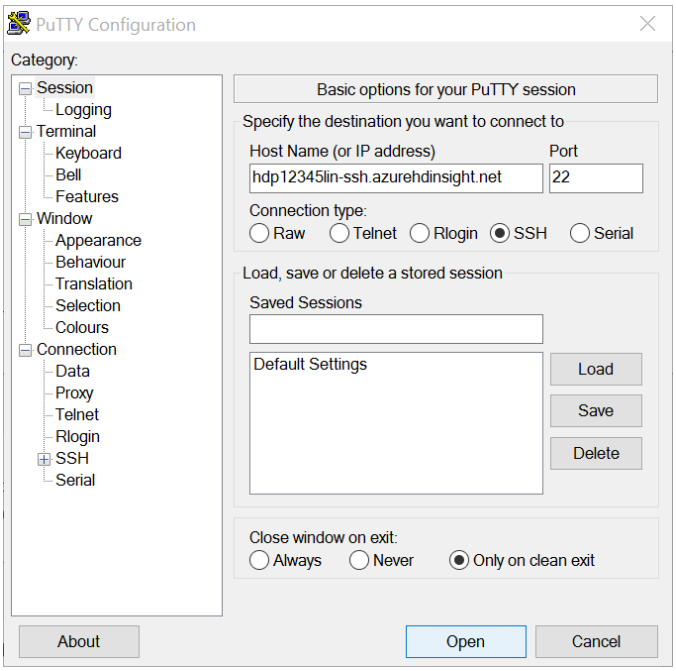
Now that you have provisioned an HDInsight HBase cluster, you can create HBase tables and store data in them.

Open a Secure Shell Connection to the Cluster

To work with HBase in your cluster, you will open a secure shell (SSH) connection.

If you are using a Windows client computer:

1. In the Microsoft Azure portal, on the **HDInsight Cluster** blade for your HDInsight cluster, click **Secure Shell**, and then in the **Secure Shell** blade, in the **hostname** list, note the **Host name** for your cluster (which should be ***your_cluster_name*-ssh.azurehdinsight.net**).
2. Open PuTTY, and in the **Session** page, enter the host name into the **Host Name** box. Then under **Connection type**, select **SSH** and click **Open**.



3. If a security warning that the host certificate cannot be verified is displayed, click **Yes** to continue.
4. When prompted, enter the SSH username and password you specified when provisioning the cluster (not the cluster login username).

If you are using a Mac OS X or Linux client computer:

1. In the Microsoft Azure portal, on the **HDInsight Cluster** blade for your HDInsight cluster, click **Secure Shell**, and then in the **Secure Shell** blade, in the **hostname** list, select the hostname for

your cluster. then copy the **ssh** command that is displayed, which should resemble the following command – you will use this to connect to the head node.

```
ssh sshuser@your_cluster_name-ssh.azurehdinsight.net
```

2. Open a new terminal session, and paste the **ssh** command, specifying your SSH user name (not the cluster login username).
3. If you are prompted to connect even though the certificate can't be verified, enter **yes**.
4. When prompted, enter the password for the SSH username.

Note: If you have previously connected to a cluster with the same name, the certificate for the older cluster will still be stored and a connection may be denied because the new certificate does not match the stored certificate. You can delete the old certificate by using the **ssh-keygen** command, specifying the path of your certificate file (**f**) and the host record to be removed (**R**) - for example:

```
ssh-keygen -f "/home/usr/.ssh/known_hosts" -R clstr-ssh.azurehdinsight.net
```

Create an HBase Table

1. In the console window for your SSH connection, enter the following command to start the HBase shell.

```
hbase shell
```

2. At the hbase prompt, enter the following command to create a table named **Stocks** with two column families named **Current** and **Closing**.

```
create 'Stocks', 'Current', 'Closing'
```

3. Enter the following command to insert a field for a record with the key **ABC** and a value of **97.3** for a column named **Price** in the **Current** column family.

```
put 'Stocks', 'ABC', 'Current:Price', 97.3
```

4. Enter the following command to insert a field for record **ABC** and a value of **95.7** for a column named **Price** in the **Closing** column family.

```
put 'Stocks', 'ABC', 'Closing:Price', 95.7
```

5. Enter the following command to return all rows from the table.

```
scan 'Stocks'
```

6. Verify that the output shows the two values you entered for the row ABC, as shown here:

```
ROW                 COLUMN+CELL
ABC                 column=Closing:Price, timestamp=nnn, value=95.7
ABC                 column=Current:Price, timestamp=nnn, value=97.3
```

7. Enter the following command to insert a field for record **ABC** and a value of **Up** for a column named **Status** in the **Current** column family.

```
put 'Stocks', 'ABC', 'Current:Status', 'Up'
```

8. Enter the following command to return the values for row ABC.

```
get 'Stocks', 'ABC'
```

9. Verify that the output shows the values of all cells for row ABC, as shown here:

```
COLUMN            CELL
Closing:Price     timestamp=nnn, value=95.7
Current:Price     timestamp=nnn, value=97.3
Current:Status    timestamp=nnn, value=Up
```

10. Enter the following command to set the **Price** column in the **Current** column family of row **ABC** to **99.1**.

```
put 'Stocks', 'ABC', 'Current:Price', 99.1
```

11. Enter the following command to return the values for row ABC.

```
get 'Stocks', 'ABC'
```

12. Verify that the output shows the updated values of all cells for row ABC, as shown here:

```
COLUMN            CELL
Closing:Price     timestamp=nnn, value=95.7
Current:Price     timestamp=nnn, value=99.1
Current:Status    timestamp=nnn, value=Up
```

13. Note the **timestamp** value for the **Current:Price** cell. Then enter the following command to retrieve the previous version of the cell value by replacing ***nnn-1*** with the timestamp for **Current:Price** minus 1 (for example, if the timestamp for **Current:Price** in the results above is 144012345678, replace ***nnn-1*** with 144012345677.)

```
get 'Stocks', 'ABC', {TIMERANGE=>[0,nnn-1]}
```

14. Verify that the output shows previous **Current:Price** value, as shown here:

```
COLUMN            CELL
Closing:Price     timestamp=nnn, value=95.7
Current:Price     timestamp=nnn, value=97.3
Current:Status    timestamp=nnn, value=Up
```

15. Enter the following command to delete the **Status** column in the **Current** column family of row **ABC**.

```
delete 'Stocks', 'ABC', 'Current:Status'
```

16. Enter the following command to return the values for row ABC.

```
get 'Stocks', 'ABC'
```

17. Verify that the **Current:Status** cell has been deleted as shown here:

```
COLUMN            CELL
Closing:Price     timestamp=nnn, value=95.7
Current:Price     timestamp=nnn, value=99.1
```

18. Enter the following command to exit the HBase shell and return to the Hadoop command line.

```
quit
```

19. Minimize the remote desktop window (you will return to the Hadoop Command Line later.)

Bulk Load Data into an HBase Table

You can bulk load data from a file into your HBase table. Before you can do this, you must store the source file in the shared storage used by your cluster. The instructions here assume you will use Azure Storage Explorer to do this, but you can use any Azure Storage tool you prefer.

1. In the **HDRTLabs** folder where you extracted the lab files for this course on your local computer, in the **Lab01** folder, open **stocks.txt** in a text editor. Note that this file contains tab-delimited records of closing and current prices for a variety of stocks. Then close the text editor without saving any changes.
2. Start Azure Storage Explorer, and if you are not already signed in, sign into your Azure subscription.
3. Expand your storage account and the **Blob Containers** folder, and then double-click the blob container for your HDInsight cluster.
4. In the **Upload** drop-down list, click **Upload Files**. Then upload **stocks.txt** as a block blob to a new folder named **data** in root of the container.
5. Switch back to the SSH connection console window, and enter the following command to verify that the file is now in the shared storage.

   ```
   hdfs dfs -ls /data
   ```

6. In the SSH connection console, enter the following command (on a single line) to transform the tab-delimited stocks.txt data to the HBase StoreFile format.

   ```
   hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
   Dimporttsv.columns="HBASE_ROW_KEY,Closing:Price,Current:Price" -
   Dimporttsv.bulk.output="/data/storefile" Stocks /data/stocks.txt
   ```

7. Wait for the MapReduce job to complete (this may take several minutes). Then enter the following command (on a single line) to load the transformed data into the **Stocks** table you created previously.

   ```
   hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
   /data/storefile Stocks
   ```

8. Wait for the MapReduce job to complete.

Query the Bulk Loaded Data

1. Enter the following command to start the HBase shell.

   ```
   hbase shell
   ```

2. Enter the following command to return all rows from the table.

   ```
   scan 'Stocks'
   ```

3. Verify that the output includes rows for the ABC stock you entered previously and the stocks in the stocks.txt file you imported, as shown here:

   ```
   ROW                COLUMN+CELL
   AAA                column=Closing:Price, timestamp=nnn, value=12.8
   AAA                column=Current:Price, timestamp=nnn, value=14.2
   ABC                column=Closing:Price, timestamp=nnn, value=95.7
   ABC                column=Current:Price, timestamp=nnn, value=99.1
   BBB                column=Closing:Price, timestamp=nnn, value=30.1
   ```

```
BBB                 column=Current:Price, timestamp=nnn, value=30.1
CBA                 column=Closing:Price, timestamp=nnn, value=120.3
CBA                 column=Current:Price, timestamp=nnn, value=120.3
GDM                 column=Closing:Price, timestamp=nnn, value=126.7
GDM                 column=Current:Price, timestamp=nnn, value=135.2
...
```

4. Enter the following command to return only the **Current:Price** column for each row:

```
scan 'Stocks', {COLUMNS => 'Current:Price'}
```

5. Verify that the output includes a row for each stock with only the Current:Price column, as shown here:

```
ROW                 COLUMN+CELL
AAA                 column=Current:Price, timestamp=nnn, value=14.2
ABC                 column=Current:Price, timestamp=nnn, value=99.1
BBB                 column=Current:Price, timestamp=nnn, value=30.1
CBA                 column=Current:Price, timestamp=nnn, value=120.3
GDM                 column=Current:Price, timestamp=nnn, value=135.2
...
```

6. Enter the following command to return only the first three rows:

```
scan 'Stocks', {LIMIT => 3}
```

7. Verify that the output includes data for only three rows (there are two columns per row), as shown here:

```
ROW                 COLUMN+CELL
AAA                 column=Closing:Price, timestamp=nnn, value=12.8
AAA                 column=Current:Price, timestamp=nnn, value=14.2
ABC                 column=Closing:Price, timestamp=nnn, value=95.7
ABC                 column=Current:Price, timestamp=nnn, value=99.1
BBB                 column=Closing:Price, timestamp=nnn, value=30.1
BBB                 column=Current:Price, timestamp=nnn, value=30.1
```

8. Enter the following command to return only the rows for with key values between C and H:

```
scan 'Stocks', {STARTROW=>'C', STOPROW=>'H'}
```

9. Verify that the output includes only rows for stocks with stock codes between ‘C’ and ‘H’, as shown here:

```
ROW                 COLUMN+CELL
CBA                 column=Closing:Price, timestamp=nnn, value=120.3
CBA                 column=Current:Price, timestamp=nnn, value=120.3
GDM                 column=Closing:Price, timestamp=nnn, value=126.7
GDM                 column=Current:Price, timestamp=nnn, value=135.2
```

10. Enter the following command to exit the HBase shell:

```
quit
```

Querying an HBase Table

In the previous exercise, you queried an HBase table from the HBase shell by using the **scan**, **get**, and **put** commands. While this works well for development and testing, you may want to create a layer of abstraction over HBase tables that enables users to access them through an alternative query interface.

Create a Hive Table on an HBase Table

Hadoop-based big data processing solutions often use Hive to provide a SQL-like query interface over files in HDFS. You can also create Hive tables that are based on HBase tables, which enables you maintain low-latency data in HBase that can be used in a big data processing workflow through Hive.

1. In the SSH console for your cluster, enter the following command to start the Hive command line interface:

   ```
   hive
   ```

2. At the Hive prompt, enter the following code to create a Hive table named **StockPrices** that is based on the **Stocks** HBase table (you can copy and paste this code from the **Create Hive Table.txt** file in the **HDRTLabs\Lab01** folder):

   ```
   CREATE EXTERNAL TABLE StockPrices
   (Stock STRING,
    ClosingPrice FLOAT,
    CurrentPrice FLOAT)
   STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
   WITH SERDEPROPERTIES
    ('hbase.columns.mapping' = ':key,Closing:Price,Current:Price')
   TBLPROPERTIES ('hbase.table.name' = 'Stocks');
   ```

3. Wait for the table to be created.
4. Enter the following code to query the Hive table (you can copy and paste this code from the **Query Hive Table.txt** file in the **HDRTLabs\Lab01** folder):

   ```
   SELECT Stock, CurrentPrice, ClosingPrice,
          IF(CurrentPrice > ClosingPrice, 'Up',IF (CurrentPrice <
   ClosingPrice, 'Down', '-')) AS Status
   FROM StockPrices
   ORDER BY Stock;
   ```

5. Wait for the MapReduce job to complete, and then view the results. Note that the status for stock **ABC** is "Up" (because the current stock price is higher than the previous closing price).
6. Enter the following command to exit the Hive shell.

   ```
   quit;
   ```

7. Enter the following command to start the HBase shell:

   ```
   hbase shell
   ```

8. In the HBase shell, enter the following command to set the **Price** column in the **Closing** column family of row **ABC** to **92.8**.

   ```
   put 'Stocks', 'ABC', 'Current:Price', '92.8'
   ```

9. Enter the following command to exit the HBase shell.

```
quit
```

10. Enter the following command to start the Hive command line interface:

```
hive
```

11. At the Hive prompt, re-enter the following code to query the Hive table again (you can copy and paste this code from the **Query Hive Table.txt** file in the **HDRTLabs\Lab01** folder):

```
SELECT Stock, CurrentPrice, ClosingPrice,
       IF(CurrentPrice > ClosingPrice, 'Up',IF (CurrentPrice <
ClosingPrice, 'Down', '-')) AS Status
FROM StockPrices
ORDER BY Stock;
```

12. Wait for the MapReduce job to complete, and then view the results. Note that the status for stock **ABC** is now "Down" (because the Hive table retrieves the latest data from the underlying HBase table each time it is queried, and the current stock price is now lower than the closing price.)
13. Enter the following command to exit the Hive shell.

```
quit;
```

Create a Phoenix View on an HBase Table

Apache Phoenix is a relational database engine built on HBase. Using Phoenix, you can create relational databases that are queried using Structured Query Language (SQL) while storing data in HBase tables. Creating a table in Phoenix creates an underlying HBase table. You can also create views and tables in Phoenix that are based on existing HBase tables.

Note: In this procedure you will use SQLLine to connect to Phoenix on HBase. SQLLine is a platform-independent JDBC-based SQL client interface. You can use any JDBC Phoenix client tool to work with Phoenix.

1. Review the notes you made about the cluster hosts after provisioning the cluster, and recall the active zookeeper node, which should have a name in the following form:

   zk*N-<first 6 characters of cluster name>*.xxxxxxxxxxxxxxxxxxxxxxxx.xx.internal.cloudapp.net

   Note: In the event that the active zookeeper node has changed since you reviewed the node status earlier, you might encounter errors in the following steps. If this happens, identify the active Zookeeper node by viewing the Ambari dashboard for the cluster again.

2. In the SSH console window, enter the following command to determine the version-specific folder where your Hadoop applications are stored:

   ```
   ls /usr/hdp
   ```

3. Note the version-specific folder (for example, 2.4.2.0-258), and then enter the following command to open SQLLine, replacing ***version*** with the version-specific folder name and ***zookeeper*** with the fully-qualified internal name of your zookeeper node (it may take a minute or so to connect):

   ```
   /usr/hdp/version/phoenix/bin/sqlline.py zookeeper:2181:/hbase-unsecure
   ```

4. When a connection to the Zookeeper node has been established, enter the following command to create a SQL view that is based on the **Stocks** HBase table. You can copy and paste this code from **Create SQL View.txt** in the C:\HDRTLabs\Lab01 folder.

```
CREATE VIEW "Stocks"
(StockCode VARCHAR PRIMARY KEY,
 "Closing"."Price" VARCHAR,
 "Current"."Price" VARCHAR);
```

5. Enter the following command to query the view (and retrieve data from the underlying HBase table). You can copy and paste this code from **Query SQL View.txt** in the **HDRTLabs\Lab01** folder.

```
SELECT StockCode, "Current"."Price"
FROM "Stocks"
WHERE "Current"."Price" > "Closing"."Price";
```

6. View the results returned by the query. Then enter the following command to exit SQLLine:

```
!exit
```

Creating an HBase Client Application

HBase provides an API, which developers can use to implement client applications that query and modify data in HBase tables. In this exercise, you will implement client application code that reads and writes data in the **Stocks** HBase table you created previously.

You can choose to complete this exercise using Java or Microsoft C#.

Note: To complete the lab using C#, you must be using a Windows client computer with Visual Studio 2015 and the latest version of the Microsoft .NET SDK for Azure installed.

Create an HBase Client Application Using Java

The Java Developer Kit (JDK) is installed on HDInsight head nodes, and can be used to develop Java applications. In a real scenario, you would typically install the development tools on your local workstation, develop and build your solutions there, and then deploy them to a production environment to run them. However, for the purposes of this lab, using the JDK on the cluster node simplifies setup.

Install and Configure Maven

Maven is a project build system for Java that simplifies the process of developing and building Java packages with the JDK.

1. In the SSH console window, enter the following command to view the version of the JDK installed on the head node:

```
javac -version
```

2. Enter the following command to verify that the **JAVA_HOME** system variable is set:

```
echo $JAVA_HOME
```

3. Enter the following command to install Maven:

```
sudo apt-get install maven
```

4. If you are prompted to confirm the installation, enter **Y** and wait for the installation to complete (this may take a few minutes).
5. After Maven is installed, enter the following command to view details of the Maven installation:

```
mvn -v
```

6. Note the Maven home path (which should be **/usr/share/maven**), and then enter the following command to add the **bin** subfolder of this path to the system **PATH** variable:

```
export PATH=/usr/share/maven/bin:$PATH
```

Create a Maven Project

1. In the SSH console, enter the following command (on a single line) to create a new Maven project named **Stocks**:

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=lex.microsoft.com -DartifactId=Stocks -DinteractiveMode=false
```

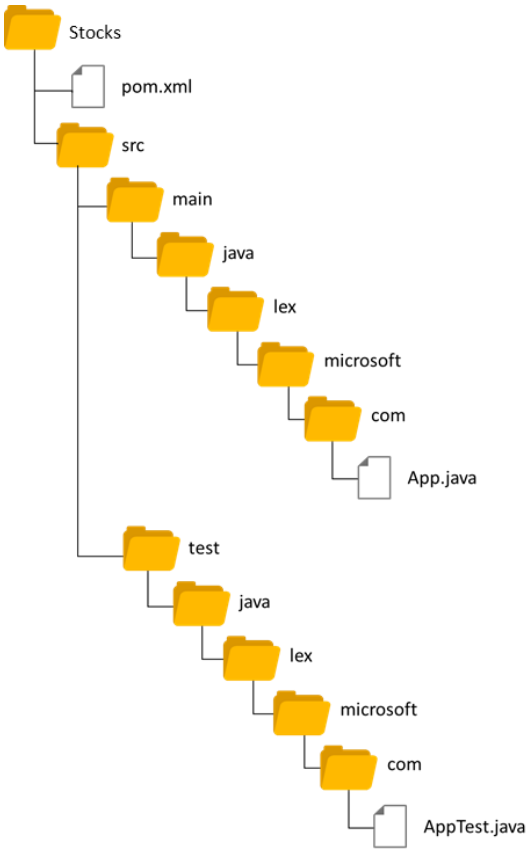
2. After the project is created, enter the following command to change the current directory to the directory for your **Stocks** project:

```
cd Stocks
```

3. Enter the following command to view the directory hierarchy created for the project:

```
ls -R
```

4. View the directory listing, and note that the directory structure for the project matches the following image:



5. You will not require the test harness generated by from the project template, so enter the following command to delete it.

```
rm -r src/test
```

Configure the Project

1. Enter the following command to open the **pom.xml** file in the Nano text editor. This file contains configuration information for the project:

```
nano pom.xml
```

2. Edit the file to remove the dependency on **junit,** which is not required since you won't be using the test harness (to remove the current line in Nano, press **CTRL+K**). Then add the sections indicated in **bold** below (you can copy this from the **pom.xml** file in the **Lab01** folder where you extracted the lab files for this course). This adds a dependency on the **hbase-client** and **phoenix-core** libraries, and adds plug-ins that make it easier to test and compile the project.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>lex.microsoft.com</groupId>
  <artifactId>Stocks</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Stocks</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.apache.hbase</groupId>
      <artifactId>hbase-client</artifactId>
      <version>1.1.2</version>
    </dependency>
    <dependency>
      <groupId>org.apache.phoenix</groupId>
      <artifactId>phoenix-core</artifactId>
      <version>4.4.0-HBase-1.1</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
                  <version>3.3</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
```

```
        <version>2.3</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer
implementation="org.apache.maven.plugins.shade.resource.ApacheLicenseRe
sourceTransformer">
                </transformer>
                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourc
eTransformer">
                  <mainClass>lex.microsoft.com.App</mainClass>
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.4.0</version>
        <executions>
          <execution>
            <goals>
              <goal>java</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <mainClass>lex.microsoft.com.App</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

3. Exit the Nano editor (enter **CTRL+X**), saving the **pom.xml** file (enter **Y** and **ENTER** when prompted).

Implement the HBase Client Application

1. Review your notes about the zookeeper hosts in the cluster – recall that there are three zookeeper nodes with names in the following format:

   zk*N*-<*first 6 characters of cluster name*>.xxxxxxxxxxxxxxxxxxxxxxxx.xx.internal.cloudapp.net

2. Enter the following command to open Nano and edit the **App.java** file in the **src/main/java/lex/microsoft/com** directory:

   ```
   nano src/main/java/lex/microsoft/com/App.java
   ```

Note: You can copy and paste the code for the client application from **App.java** in the **Lab01** folder where you extracted the lab files for this course

3. In Nano, under the existing **package lex.microsoft.com** statement at the top of the code file, replace the **/** Hello World! **/** comment lines with the following import statements:

```
import java.util.Scanner;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.filter.RegexStringComparator;
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter;
import org.apache.hadoop.hbase.filter.CompareFilter.CompareOp;
import org.apache.hadoop.hbase.util.Bytes;
import java.util.Random;
```

4. In the **App** public class, above the **main** function declaration, add the following code to declare a table for the HBase table:

```
static HTable table;
```

5. In the main function, replace the **System.out.println("Hello World!");** line with the following code; substituting the ***zookeeperN*** placeholders with the fully-qualified names of the Zookeeper nodes in your cluster:

```
try
{
 // Configure the HBase connection
 Configuration config = HBaseConfiguration.create();
 config.set("hbase.zookeeper.quorum",
           "zookeeper0,zookeeper1,zookeeper2");
 config.set("hbase.zookeeper.property.clientPort", "2181");
 config.set("hbase.cluster.distributed", "true");
 config.set("zookeeper.znode.parent","/hbase-unsecure");

 // Open the HTable
 table = new HTable(config, "Stocks");

 // Read the command-line input until we quit
 System.out.println("\n");
 Scanner scanner = new Scanner(System.in);
 boolean quit = false;
 do
 {
    System.out.println("Enter a stock code, or enter 'quit' to exit.");
    String input = scanner.next();
    // Should we quit?
    if(input.toLowerCase().equals("quit"))
    {
```

```
        System.out.println("Quitting!");
        quit = true;
    } else {

          // Get the stock data
          getstock(input);

          // Update stock prices
          updatestocks();
    }
  }
  while(!quit);
}
catch (Exception ex)
{
  // Error handling goes here
}
```

This code creates a loop that reads user input until the command "quit" is entered. Each time the user enters a stock ticker symbol, the code calls a function named **getstock** to retrieve the stock price information for the specified symbol, and then calls a function named **updatestocks** to update the stock prices in the HBase table (you will implement these functions in the next procedure).

6. Under the main function, add the following code to define a function named **getstock** that retrieves the current and closing prices of a specified stock from HBase:

```
public static void getstock( String stock ) throws IOException
{
  try
  {
    // Get the stock ticker to search for as a byte array
    byte[] rowId = Bytes.toBytes(stock);
    Get rowData = new Get(rowId);
    // Read the data
    Result result = table.get(rowData);

    // Read the values, converting from byte array to string
    String closingPrice = Bytes.toString(
           result.getValue(Bytes.toBytes("Closing"),
                           Bytes.toBytes("Price")));
    String currentPrice = Bytes.toString(
           result.getValue(Bytes.toBytes("Current"),
                           Bytes.toBytes("Price")));
    // Print out the values
    System.out.println("Closing Price: " + closingPrice);
    System.out.println("Current Price: " + currentPrice);
    System.out.println("\n");
  }
  catch (Exception ex)
  {
    // Error handling goes here
  }
```

```
}
```

7. Under the **getstock** function, add the following code to define a function named **updatestocks** that updates the current price for all stocks by a random amount:

```
public static void updatestocks() throws IOException
{
  try
  {
    // Used to create a new random number
    Random _rand = new Random();
    // Range for random numbers
    Double rangeMin = -1.0;
    Double rangeMax = 1.0;

    // Get all stocks between "AAA" and "ZZZ" in batches of 10
    Scan scan = new Scan(Bytes.toBytes("AAA"), Bytes.toBytes("ZZZ"));
    scan.setBatch(10);
    ResultScanner results = table.getScanner(scan);
    // Iterate over the results
    for(Result result : results)
    {
        String rowId = new String(result.getRow());
        String currentPriceStr = Bytes.toString(result.getValue(
               Bytes.toBytes("Current"),Bytes.toBytes("Price")));
        // Update current price by random amount between -1 and 1
        Double currentPrice = Double.parseDouble(currentPriceStr);
        Double randomValue = rangeMin + (rangeMax - rangeMin) *
               _rand.nextDouble();
        currentPrice = currentPrice + randomValue;
        currentPriceStr = String.valueOf(currentPrice);
        Put stockUpdate = new Put(Bytes.toBytes(rowId));
        stockUpdate.add(Bytes.toBytes("Current"),
             Bytes.toBytes("Price"), Bytes.toBytes(currentPriceStr));
        table.put(stockUpdate);
    }
    // Flush committed updates
    table.flushCommits();
  }
  catch (Exception ex)
  {
     // Error handling goes here
  }
}
```

8. Ensure that your code matches the code in **App.java** in the **Lab01** folder. Then exit the Nano editor (enter **CTRL+X**), saving the **App.java** file (enter **Y** and **ENTER** when prompted).

Compile and Run the Application

1. In the SSH console, ensure that the current directory context is still the **Stocks** directory, and then enter the following command to compile and run the application:

```
mvn compile exec:java
```

2. When prompted, enter the stock code **AAA** and press **ENTER**. Then note the current price of the AAA stock that is retrieved from the HBase table.
3. Enter the stock code **AAA** again, and verify that the current price has been updated.
4. Continue testing the application with other valid stock codes, and finally enter **quit** when you are finished to stop the application.

> **Note**: The code in the application has been kept deliberately simple to make it easy to understand the key methods for working with HBase. The application contains no error handling, and will crash if an invalid stock code is entered. In a production application, you would add error handling code to prevent this.

5. Close the SSH console.

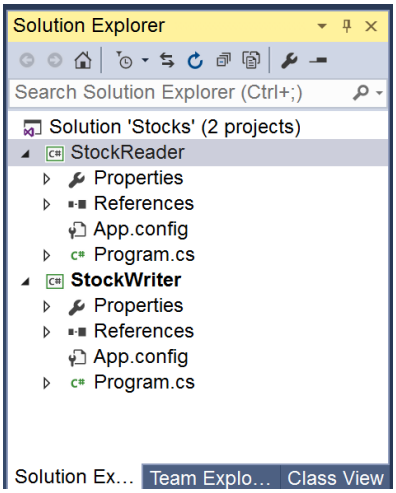
Create an HBase Client Application Using Microsoft C#

The Microsoft .NET HBase REST API is a .NET wrapper around the HTTP REST-based interface for Base. This API makes it easier to create .NET client applications for HBase than programming directly against the REST interface.

Note: You can only complete this exercise on a Windows client computer with Visual Studio 2015 and the latest version of the Microsoft .NET SDK for Azure installed. See the course setup guide for details.

Create Projects for the StockWriter and StockReader Applications

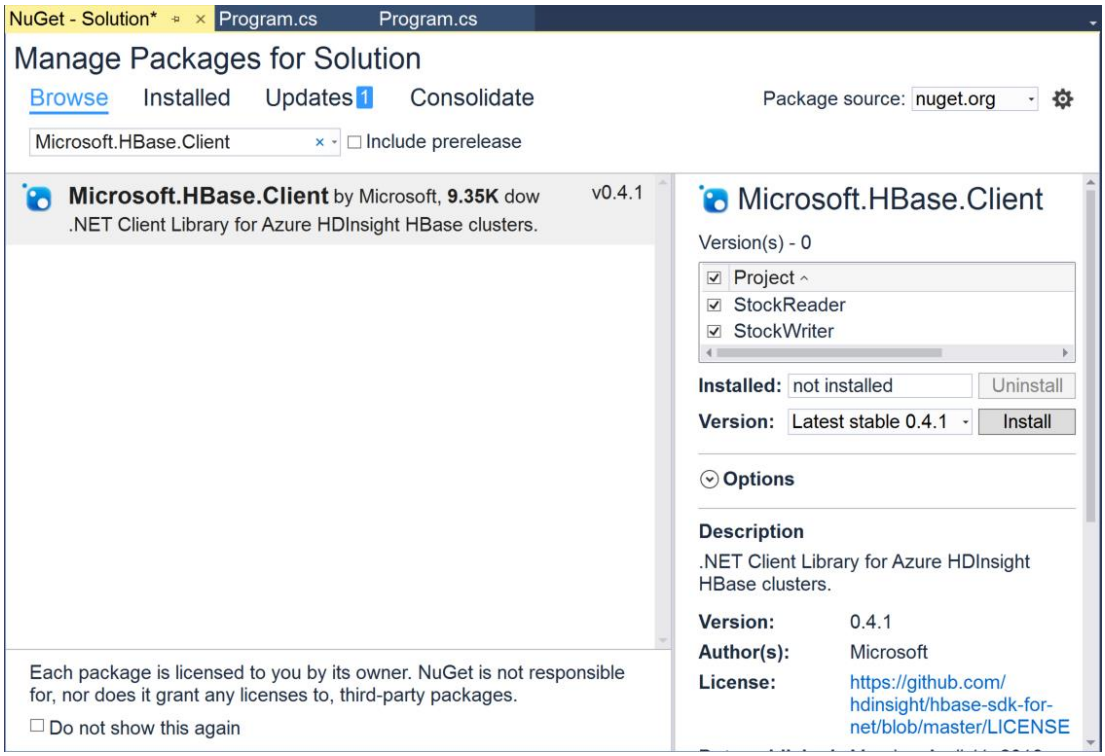
1. Start Visual Studio and on the **File** menu, point to **New**, and click **Project**. Then create a new project based on the Visual C# **Console Application** template. Name the project **StockWriter**, name the solution **Stocks**, and save it in the C:\HDRTLabs\Lab01 folder.
2. On the **File** menu, point to **Add** and click **New Project**. Then add a Visual C# **Console Application** project named **StockReader** to the solution.
3. If the Solution Explorer pane is not visible, on the **View** menu, click **Solution Explorer**; and then verify that your **Stocks** solution contains two projects named **StockReader** and **StockWriter** as shown here:



Add the .NET HBase Client Package

1. On the **Tools** menu, point to **NuGet Package Manager**, and click **Manage NuGet Packages for Solution**.

2. In the NuGet Package Manager window, search nuget.org for *HBase Client*, and in the list of results, select **Microsoft.HBase.Client**. Then configure the following settings and click **Install**, as shown below:
    - **Select which projects to apply changes to**: *Select **StockReader** and **StockWriter**.*
    - **Version**: Latest stable x.x.x



3. If you are prompted to review changes, click **OK**.
4. When the package has been installed, close the NuGet Package Manager window.

Implement the StockWriter Application

1. In Solution Explorer, under **StockWriter**, double-click **Program.cs** to open the main code file for the **StockWriter** project.
2. At the top of the code file, replace all of the existing using statements with the following code. You can copy and paste this code from **StockWriter.txt** in the C:\HDRTLabs\Lab01 folder.

```
using System;
using System.Text;
using Microsoft.HBase.Client;
using org.apache.hadoop.hbase.rest.protobuf.generated;
```

3. In the static void **Main** function, add the following code, replacing the values for the **clusterURL**, **userName**, and **password** variables with the appropriate values for your HDInsight cluster. You can copy and paste this code from **StockWriter.txt** in the C:\HDRTLabs\Lab01 folder.

```
while (true)
{
  Random rnd = new Random();
  Console.Clear();

  string clusterURL = "https://hb12345.azurehdinsight.net";
```

```
string userName = "HDUser";
string password = "HDPa$$w0rd";

string tableName = "Stocks";

// Connect to HBase cluster
ClusterCredentials creds =
          new ClusterCredentials(new Uri(clusterURL),
                                 userName, password);
HBaseClient hbaseClient = new HBaseClient(creds);

// Get all stocks
Scanner scanSettings = new Scanner()
{
    batch = 10,
    startRow = Encoding.UTF8.GetBytes("AAA"),
    endRow = Encoding.UTF8.GetBytes("ZZZ")
};

// Scan APIs are stateful, specify the endpoint
// where the request should be sent to.
// e.g. hbaserest0/ means rest server on workernode0
RequestOptions scanOptions = RequestOptions.GetDefaultOptions();
scanOptions.AlternativeEndpoint = "hbaserest0/";
ScannerInformation stockScanner = null;
try
{
  stockScanner = hbaseClient.CreateScannerAsync(tableName,
                        scanSettings, scanOptions).Result;
  CellSet stockCells = null;

  while ((stockCells = hbaseClient.ScannerGetNextAsync(stockScanner,
                       scanOptions).Result) != null)
  {
    foreach (var row in stockCells.rows)
    {
      string stock = Encoding.UTF8.GetString(row.key);
      Double currentPrice =
         Double.Parse(Encoding.UTF8.GetString(row.values[1].data));
      Double newPrice = currentPrice + (rnd.NextDouble() * (1 - -1) +
         -1);
      Cell c = new Cell
      {
        column = Encoding.UTF8.GetBytes("Current:Price"),
        data = Encoding.UTF8.GetBytes(newPrice.ToString())
      };
      row.values.Insert(2, c);
      Console.WriteLine(stock + ": " +
         currentPrice.ToString() + " := "
         + newPrice.ToString());
    }
    hbaseClient.StoreCellsAsync(tableName, stockCells).Wait();
  }
}
finally
```

```
  {
  // Make sure free up the resources on rest server
  //after finishing the scan.
  if (stockScanner != null)
  {
    hbaseClient.DeleteScannerAsync(tableName, stockScanner,
         scanOptions).Wait();
  }
}
```

Note: This code performs the following actions:

1. Connects to your HBase cluster using the URL and credentials in your code.
2. Uses a **Scanner** object to retrieve a cellset containing all stock records from the **Stocks** HBase table you created earlier in this lab. This is a wrapper around the **scan** HBase command.
3. Loops through each stock record, incrementing the value of the first column (**Current:Price**) by a random value between -1 and 1.
4. Stores the updated cells back to the table.

4. Save **Program.cs** and close it.

Implement the StockReader Application

1. In Solution Explorer, under **StockReader**, double-click **Program.cs** to open the main code file for the **StockReader** project.
2. At the top of the code file, replace all of the existing using statements with the following code. You can copy and paste this code from **StockReader.txt** in the C:\HDRTLabs\Lab01 folder.

```
using System;
using System.Text;
using Microsoft.HBase.Client;
using org.apache.hadoop.hbase.rest.protobuf.generated;
```

3. In the static void **Main** function, add the following code, replacing the values for the **clusterURL**, **userName**, and **password** variables with the appropriate values for your HDInsight cluster. You can copy and paste this code from **StockReader.txt** in the C:\HDRTLabs\Lab01 folder.

```
bool quit = false;
while (!quit)
{
  Console.ResetColor();
  Console.WriteLine("Enter a stock code, or enter 'quit' to exit");

  // Connect to HBase cluster
  string clusterURL = "https://hb12345.azurehdinsight.net";
  string userName = "HDUser";
  string password = "HDPa$$w0rd";
  ClusterCredentials creds = new ClusterCredentials
                                (new Uri(clusterURL),
                                 userName,
                                 password);
  HBaseClient hbaseClient = new HBaseClient(creds);

  string input = Console.ReadLine();
```

```
    if (input.ToLower() == "quit")
    {
      quit = true;
    }
    else
    {
      CellSet cellSet =
                hbaseClient.GetCellsAsync("Stocks", input).Result;
      var row = cellSet.rows[0];
      Double currentPrice = Double.Parse(
                        Encoding.UTF8.GetString(row.values[1].data));
      Double closingPrice = Double.Parse(
                        Encoding.UTF8.GetString(row.values[0].data));
      if (currentPrice > closingPrice)
      {
        Console.ForegroundColor = ConsoleColor.Green;
      }
      else if (currentPrice < closingPrice)
      {
        Console.ForegroundColor = ConsoleColor.Red;
      }
      Console.WriteLine(input + ": " + currentPrice.ToString());
    }
  }
```

Note: This code performs the following actions:

1. Connects to your HBase cluster using the URL and credentials in your code.
2. Reads the input from the command line (which is assumed to be either a stock code or the command "quit").
3. Uses the **GetCells** method to retrieve the record in the HBase **Stocks** table for the specified stock code key value. This is a wrapper around the **get** HBase command.
4. Reads the first (**Current:Price**) and second (**Closing:Price**) cells from the cellset.
5. Displays the current stock price, with color coding to indicate whether it is higher or lower than the closing price.

4. Save **Program.cs** and close it.

Build and Test the Applications

1. On the **Build** menu, click **Build Solution**.
2. When both projects have been built, in Solution Explorer, right-click **StockReader**, point to **Debug**, and click **Start new instance**. This opens a console window for the StockReader application.
3. In the StockReader console window, enter **AAA**, and note that the current stock price for stock **AAA** is displayed.

Note: The code in the application has been kept deliberately simple to make it easy to understand the key methods for working with HBase. The application contains no error handling, and will crash if an invalid stock code is entered. In a production application, you would add error handling code to prevent this.

4. In Solution Explorer, right-click **StockWriter**, point to **Debug**, and click **Start new instance**. This opens a console window for the StockWriter application.

5. Observe the StockWriter console window, noting that it displays a constant sequence of updated stock prices.
6. In the StockReader console window, enter **AAA**, and note that the latest current stock price for stock **AAA** is displayed.
7. In the StockReader console window, enter **BBB**, and note that the latest current stock price for stock **BBB** is displayed.
8. Repeat the previous two steps, noting that the latest price for the specified stock is always retrieved from HBase.
9. In the StockReader console window, enter **quit** to close the console window. Then close the StockWriter console window to end the application.
10. Close Visual Studio, saving your work of prompted.

Clean Up

Now that you have finished using HBase, you can delete your cluster and the associated storage account. This ensures that you avoid being charged for cluster resources when you are not using them. If you are using a trial Azure subscription that includes a limited free credit value, deleting the cluster maximizes your credit and helps to prevent using it all before the free trial period has ended.

Delete the Resource Group

1. Close the browser tab containing the HDInsight Query Console if it is open.
2. In the Azure portal, view your **Resource groups** and select the resource group you created for your cluster. This resource group contains your cluster and the associated storage account.
3. In the blade for your resource group, click **Delete**. When prompted to confirm the deletion, enter the resource group name and click **Delete**.
4. Wait for a notification that your resource group has been deleted.
5. Close the browser.