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 Microsoft Windows computer containing:
  - Microsoft Visual Studio
  - 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, add a new HDInsight cluster with the following settings:
   - **Cluster Name**: *Enter a unique name (and make a note of it!)*
   - **Cluster Type**: HBase
   - **Operating System**: *Choose the latest version of Windows Server*

- **Subscription**: *Your Azure subscription*
- **Resource Group**: *Create a new resource group with a unique name*
- **Credentials**:
  - **Cluster Login Username**: *Enter a user name of your choice (and make a note of it!)*
  - **Cluster Login Password**: *Enter and confirm a strong password (and make a note of it!)*
  - **Enable Remote Desktop:** Yes
  - **Expires on**: *Select the date a week from now*
  - **Remote Desktop Username:** *Enter another user name of your choice (and make a note of it!)*
  - **Remote Desktop Password:** *Enter and confirm a strong password (and make a note of it!)*
- **Data Source**:
  - **Selection Method**: From all subscriptions
  - **Create a new storage account**: *Enter a unique name for your storage account (and make a note of it!)*
  - **Choose Default Container**: *The name of your cluster*
  - **Location**: *Select any available region*.
- **Optional Configuration**:
  - **HDInsight Version**: *Select the most recent version available*
  - **Virtual Network**: Not Configured
  - **External Metastores**: Not Configured
  - **Script Actions**: Not Configured
  - **Azure Storage Keys**: Not Configured
- **Node Pricing Tiers**:
  - **Number of Worker nodes**: 1
  - **Worker Nodes Pricing Tier**: *Leave the default selection*
  - **Head Node Pricing Tier**: *Leave the default selection*
- **Pin to Startboard**: *Not selected*

3. 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. The free-trial subscription includes a credit limit of approximately $100 (or local equivalent) that you can spend over a period of 30 days, which is 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 in order to avoid using your Azure credit unnecessarily.

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 Remote Desktop Connection to the Cluster

1. In the Azure portal, browse to the HBase cluster you just created.
2. Open a remote desktop connection to the cluster using the remote desktop username and password you specified when provisioning the cluster.
3. When the remote desktop window opens, open the **Hadoop Command Line** console and view the syntax documentation for the Hadoop command line tool.

Create an HBase Table

Note: The commands in this procedure are case-sensitive.

1. In the Hadoop Command Line console window, enter the following command to change the current directory to the HBase installation directory:

   ```
   cd %HBASE_HOME%\bin
   ```

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

   ```
   hbase shell
   ```

3. Enter the following command to create a table named **Stocks** with two column families named **Current** and **Closing**.

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

4. 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'
   ```

5. 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'
   ```

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

   ```
   scan 'Stocks'
   ```

7. 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
   ```

8. 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'
   ```

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

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

10. 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
    ```

11. 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'
    ```

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

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

13. 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
```

14. 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]}
```

15. 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
```

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

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

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

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

18. 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
```

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

```
quit
```

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

Bulk Load Data into an HBase Table

1. In the C:\HDRTLabs\Lab01 folder, double-click **stocks.txt** to open the file in Notepad. Note that this file contains tab-delimited records of closing and current prices for a variety of stocks. Then close Notepad without saving any changes.
2. In the C:\HDRTLabs\Lab01 folder, right-click **stocks.txt** and click **Copy** to copy the file to the clipboard.
3. Maximize the remote desktop window, open File Explorer, and view the content of the **C:** folder on your cluster head node. Then paste the copied stocks.txt file into C:\.
4. In the **Hadoop Command Line** window, enter the following commands to create a folder named **/data** in the HDFS file system for the cluster, and copy the stocks.txt file from the local C:\ folder to the new **/data** folder.

```
hadoop fs -mkdir /data
```

```
hadoop fs -copyFromLocal c:\stocks.txt /data/stocks.txt
```

5. In the **Hadoop Command Line** window, 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
```

6. 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
```

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

```
hbase shell
```

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

```
scan 'Stocks'
```

9. 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
...
```

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

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

11. 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
...
```

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

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

13. 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
```

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

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

15. 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
```

16. Minimize the remote desktop window (you will return to the HBase Shell in the next exercise.)

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 your web browser, in the Azure portal, browse all of your Azure resources and select your HDInsight HBase cluster.
2. Open the **Dashboard** for your cluster, and when prompted, sign in using the cluster username and password you specified when creating the cluster.
3. In the **HDInsight Query Console**, open the **Hive Editor**.
4. In the Hive Editor, enter the query name **Create Hive Table on HBase**, and replace the default HiveQL **Select** statement with the following code (you can copy and paste this code from the **Create Hive Table.txt** file in the C:\HDRTLabs\Lab01 folder):

```
SET hive.execution.engine=tez;
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');
```

5. Click **Submit**, and when the **Create Hive Table on HBase** job is listed in the **Job Session** table, wait for its **Status** to change to **Completed**.
6. In the Hive editor, change the query name **Query HBase Table**, and replace the HiveQL **CREATE EXTERNAL TABLE** statement with the following code (you can copy and paste this code from the **Query Hive Table.txt** file in the C:\HDRTLabs\Lab01 folder):

```
SET hive.execution.engine=tez;
SELECT Stock, CurrentPrice, ClosingPrice,
       IF(CurrentPrice > ClosingPrice, 'Up',
            IF (CurrentPrice < ClosingPrice, 'Down', '-')) AS Status
FROM StockPrices
ORDER BY Stock;
```

7. Click **Submit**, and when the **Query HBase Table** job is listed in the **Job Session** table, wait for its **Status** to change to **Completed**.
8. In the **Job Session** table, click **Query HBase Table**. Then in the **Job Details** tab, view the results returned by the query in the **Job Output** section.
9. Note that the status for stock **ABC** is "Up" (because the current stock price is higher than the previous closing price). Then close the **Job Details** tab.
10. Maximize the remote desktop window, and in the Hadoop Command Line window, 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'
```

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

```
quit
```

12. Minimize the remote desktop window (you will return to the Hadoop Command Line later.)
13. In the **Hive Editor** tab in your browser, ensure that the Hive **SELECT** query is still in the editor, change the query name to **Requery HBase Table**, click **Submit**, and wait for the job to complete.
14. In the **Job Session** table, click **Requery HBase Table**. Then in the **Job Details** tab, view the results returned by the query in the **Job Output** section.
15. 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.) Then close the **Job Details** and **Hive Editor** tabs.

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. Maximize the remote desktop window, and in the Hadoop Command Line window, enter the following command:

```
ipconfig
```

2. Make a note of the **Connection-specific DNS Suffix** for the Ethernet adapter on the cluster head node(which should be similar to *cluster.a1.internal.cloudapp.net*.)
3. Enter the following command to change the current directory to the Phoenix installation directory:

```
cd %PHOENIX_HOME%\bin
```

4. Enter the following command to open SQLLine, replacing ***cluster.a1.internal.cloudapp.net*** with the connection-specific DNS suffix you noted in step 2 (it may take a minute or so to connect):

```
sqlline.py zookeeper0.cluster.a1.internal.cloudapp.net
```

5. 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" DECIMAL,
 "Current"."Price" DECIMAL);
```

6. 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 C:\HDRTLabs\Lab01 folder.

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

7. View the results returned by the query (the decimal **Price** values are displayed in scientific notation). Then close the Hadoop Command Line window and sign out of the remote desktop session.

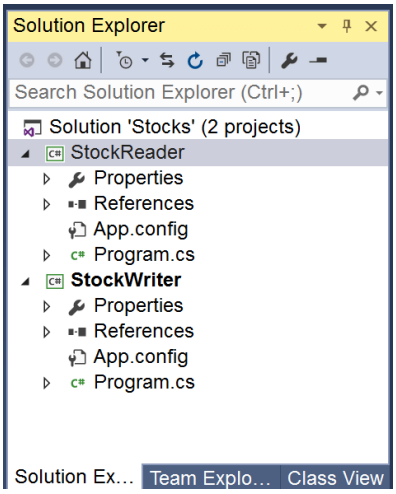
Using the Microsoft .NET HBase REST API Client

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.

In this exercise, you will create two simple .NET console applications; one to constantly update stock data in HBase, and the other to read price data for a specific stock symbol.

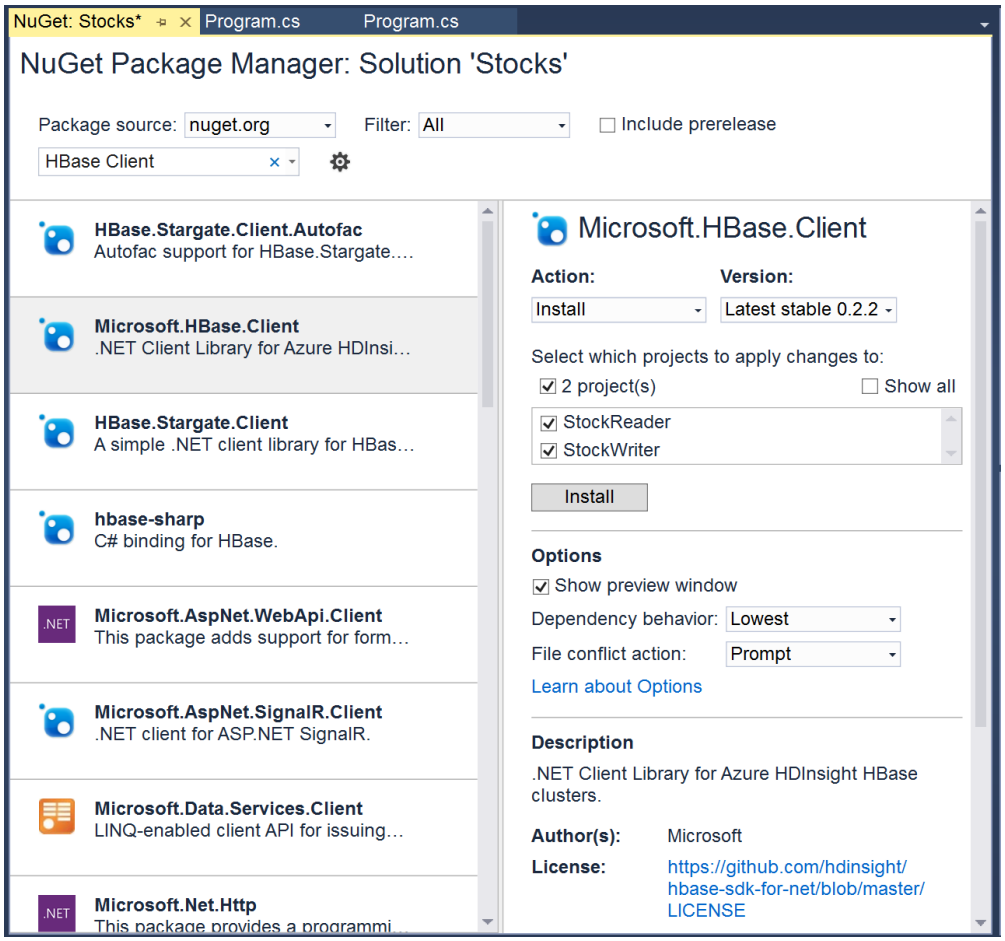
Create Projects for the StockWriter and StockReader Applications

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:
    - **Action**: Install
    - **Version**: Latest stable x.x.x
    - **Select which projects to apply changes to**: *Select **StockReader** and **StockWriter***.



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";

  // 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")
  };

  ScannerInformation stockScanner =
            hbaseClient.CreateScanner("Stocks", scanSettings);
  CellSet stockCells = null;
  while ((stockCells = hbaseClient.ScannerGetNext(stockScanner))
          != 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.StoreCells("Stocks", stockCells);
  }
}
```

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.GetCells("Stocks", input);
    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.