Microsoft Cloud Workshop

Intelligent vending machines

Hands-on lab step-by-step

March 2018

Contents

Intelligent vending machines hands-on lab step-by-step

Abstract and learning objectives

In this workshop, attendees will implement an IoT solution for intelligent vending machines, leveraging facial feature recognition and Azure Machine Learning, to gain a better understanding of building cloud-based machine learning app, and real-time analytics with SQL Database in-memory and columnar indexing.

In addition, attendees will learn to:

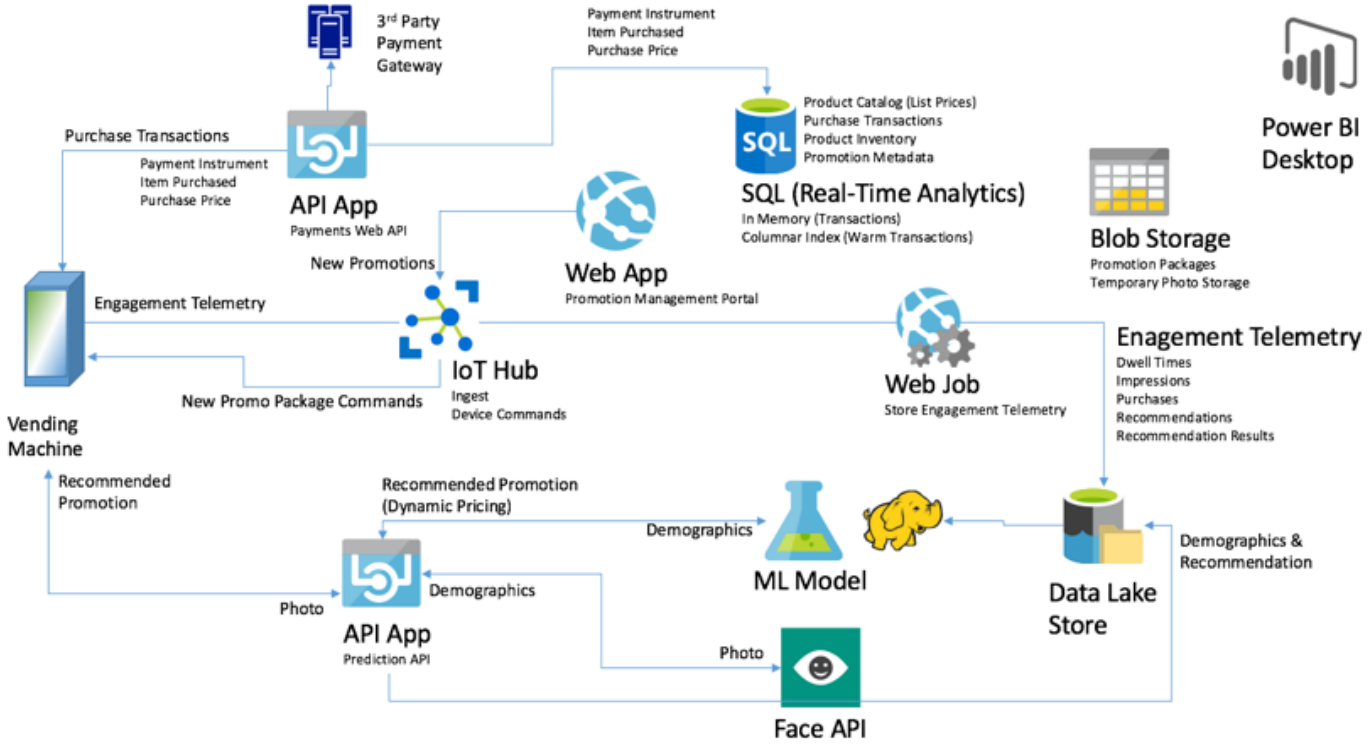
- Enable real-time analytics with SQL Database in-memory and columnar indexing
- Implement distributed machine learning with R Server for HDInsight & Microsoft R Server Operationalization
- Perform facial image processing
- Wrangle data in Power BI Desktop

Overview

Trey Research Inc. looks at the old way of doing things in retail and introduces innovative experiences that delight customers and drive sales. Their latest initiative focuses on intelligent vending machines that support commerce, engagement analytics, and intelligent promotions.

Solution architecture

Below are diagrams of the solution architecture you will build in this lab. Please study this carefully, so you understand the whole of the solution as you are working on the various components.



Requirements

- Microsoft Azure subscription must be pay-as-you-go or MSDN.
  - Trial subscriptions will not work.
- A virtual machine configured with:
  - Visual Studio Community 2017 15.6 or later
  - Azure SDK 2.9 or later (Included with Visual Studio 2017)
  - R Tools for Visual Studio 0.3.2 or later
  - Power BI Desktop (June 2016 build or later)
- A running R Server on HD Insight Spark cluster (see Before the Hands-on Lab).

Before the hands-on lab

Duration: 75 minutes

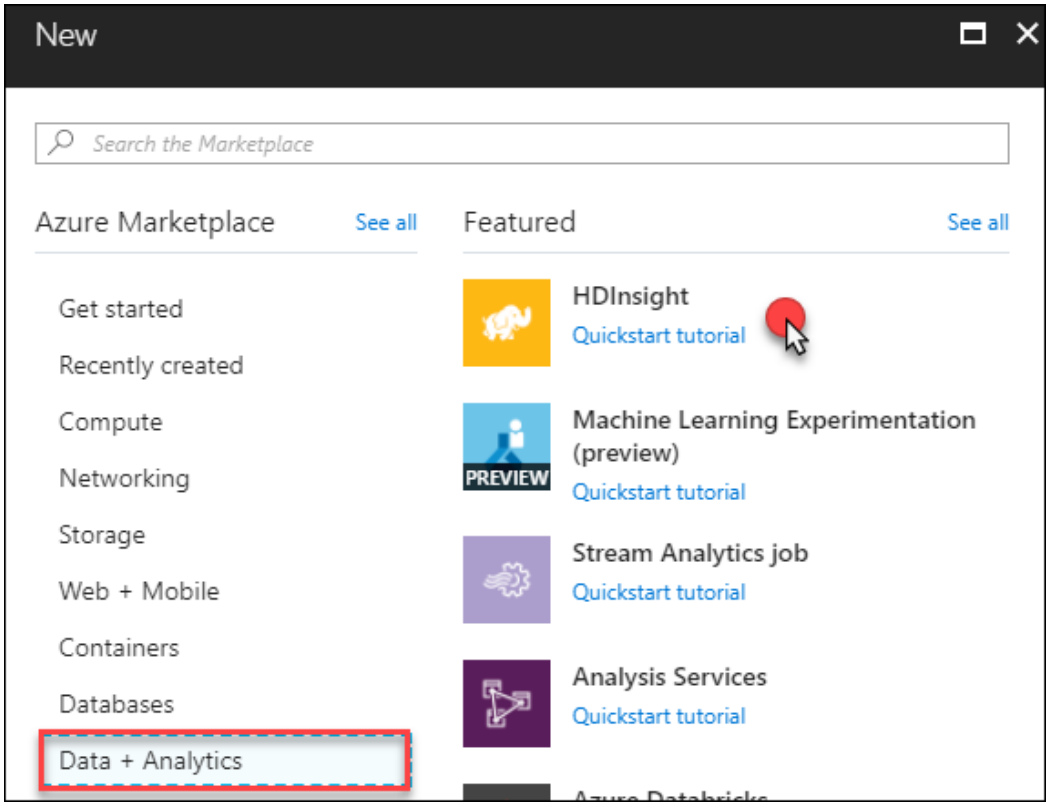
Synopsis: In this exercise, you will set up your environment for use in the rest of the hands-on lab. You should follow all the steps provided in the Before the Hands-on Lab section to prepare your environment before attending the hackathon.

Task 1: Provision an R Server on HDInsight with Spark cluster

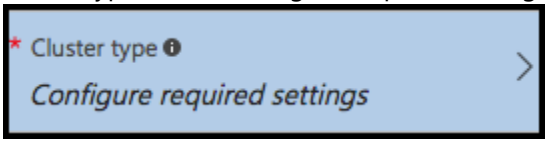
Using the Azure Portal, provision a new HDInsight cluster.

1. Open a browser and go to https://portal.azure.com.

2. Select **+Create a resource,** select **Data + Analytics**, **HDInsight**.

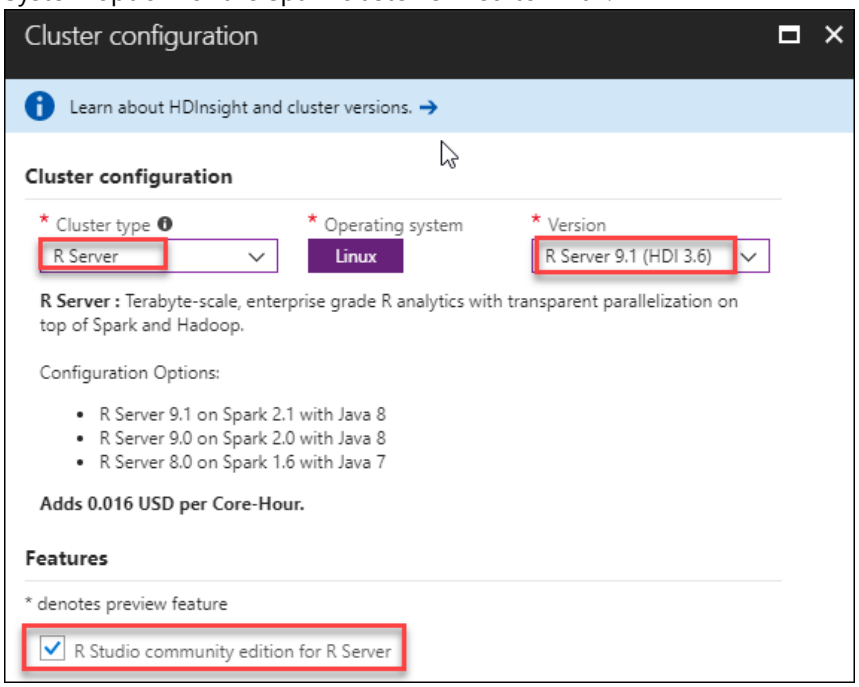


3. On the Basics blade, enter the following settings:
   a. Cluster name: Enter a **unique name** (verified by the green checkmark).
   b. Subscription: Select the Azure subscription into which you want to deploy the cluster.
   c. Custer type: Select Configure required settings.



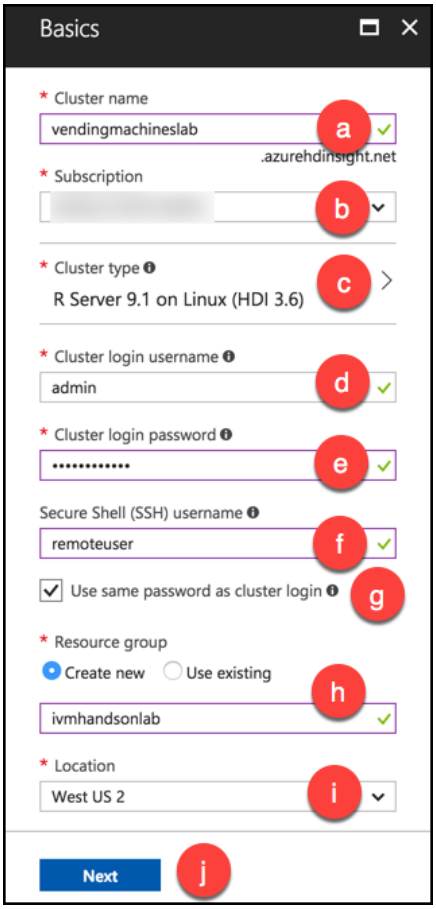
      i. On the Cluster configuration blade, set the Cluster type to **R Server** and the Version to R Server **9.1 (HDI 3.6).**

ii. Check the box next to **R Studio community edition for R Server**. Note that the Operating System option for the Spark cluster is fixed to Linux.

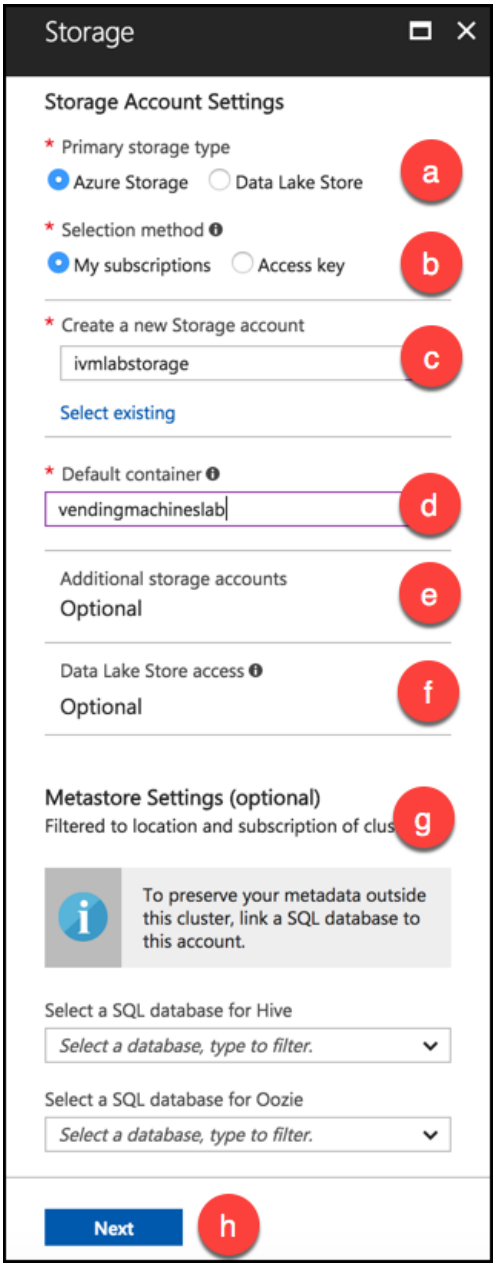


iii. Click **Select** to close the Cluster configuration blade.

d. Cluster login username: Leave as **admin**.
e. Cluster login password: Enter **Password.1!!** for the admin password.
f. Secure Shell (SSH) username: Set to **remoteuser** (this is required).
g. Use same password as cluster login: Ensure the checkbox is checked.
h. Resource group: Select the **Create new** radio button, and enter **ivmhandsonlab** for the resource group name.
i. Select the desired location from the dropdown list.
j. Select **Next** to move on to the storage settings.

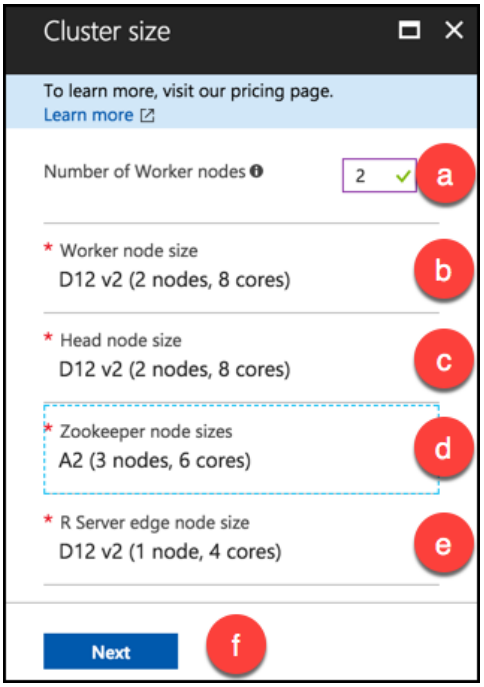


4. On the Storage blade:
    a. Primary storage type: Leave set to **Azure Storage**.
    b. Selection Method: Leave set to **My subscriptions**.
    c. Select a Storage account: Select **Create new**, and enter a name for the storage account, such as ivmlabstorage.
    d. Default container: Set to the **name of your cluster**.
    e. Additional storage accounts: Leave unconfigured.
    f. Data Lake Store access: Leave unconfigured.
    g. Metastore Settings: Leave blank.
    h. Select **Next** to move on to the Cluster summary.



5. On the Cluster Summary blade, select Edit next to Cluster Size.

6. On the Cluster size blade, enter the following:
   a. Number of worker nodes: Enter **2**.
   b. Select **Worker node size**, and select **D12 v2**, then click **Select**.
   c. Select **Head node size**, and select **D12 v2**, then click **Select**.
   d. Leave the Zookeeper node sizes set to A2.
   e. Select **R-Server edge node size**, and select **D12 v2**, then click **Select**.
   f. Select **Next**.



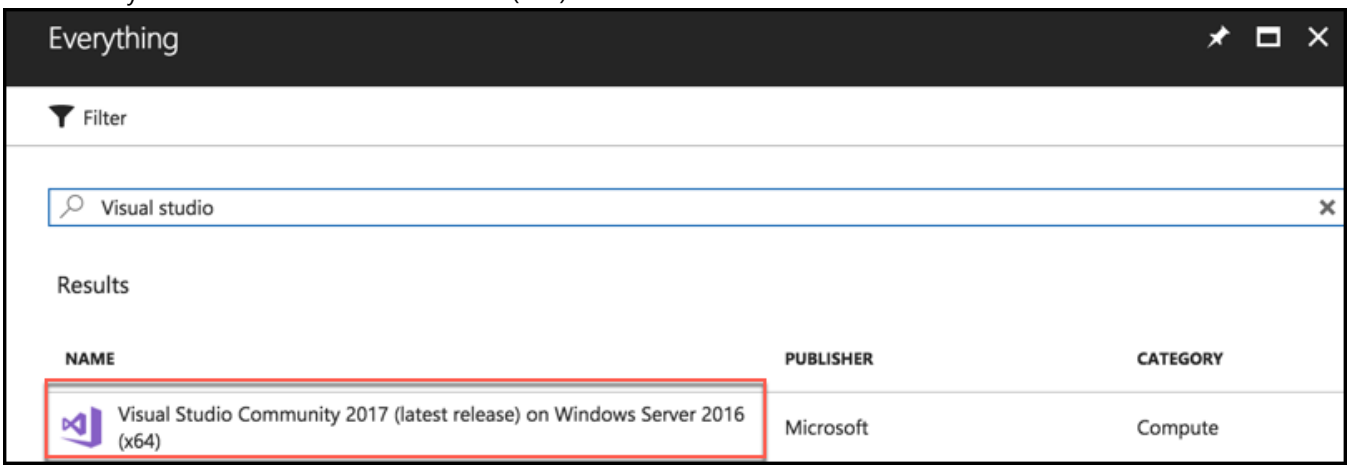
7. Select **Next** on the Advanced settings blade to move to the Cluster summary blade.

8. Select **Create** on the Cluster summary blade to create the cluster.

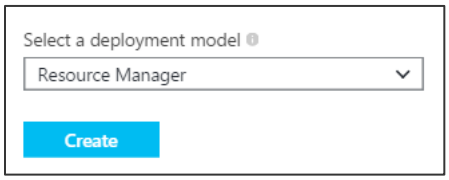
9. It will take approximately 20 minutes to create you cluster. You can move on to the steps below while the cluster is provisioning.

Task 2: Setup a lab virtual machine (VM)

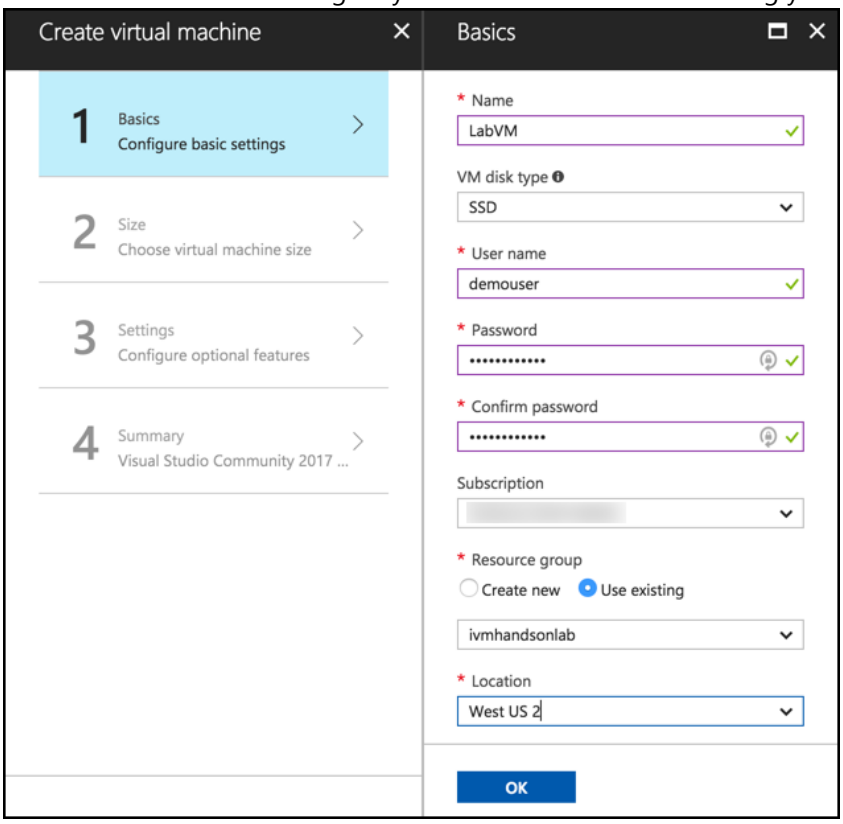
1. In the Azure Portal, select **+Create a resource**, then type "Visual Studio" into the search bar. Select Visual Studio Community 2017 on Windows Server 2016 (x64) from the results.



2. On the blade that comes up, at the bottom, ensure the deployment model is set to Resource Manager and select Create.

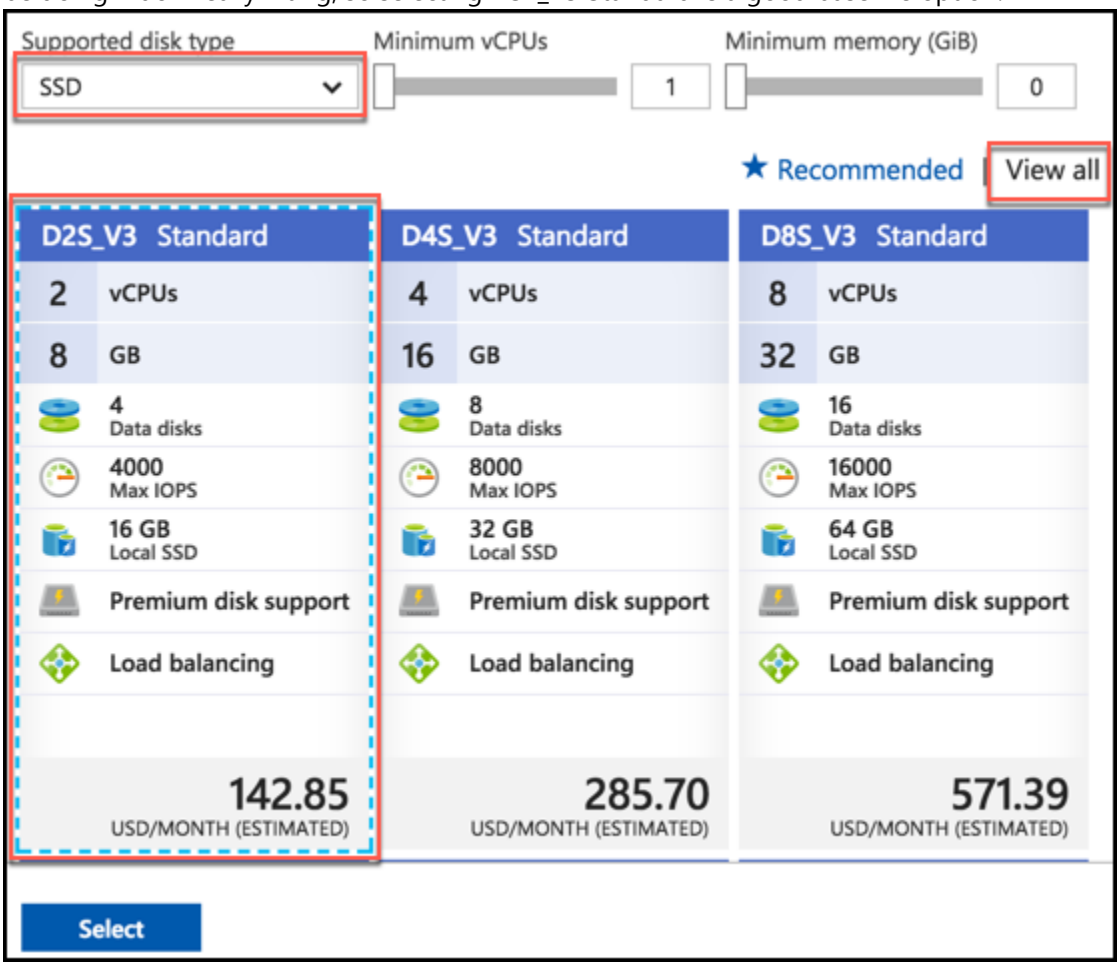


3. Set the following configuration on the Basics tab.
    a. Name: Enter **LabVM**.
    b. VM disk type: Select **SSD**.
    c. User name: Enter **demouser**
    d. Password: Enter **Password.1!!**
    e. Subscription: Select the same subscription you used to create your cluster in Task 1.
    f. Resource Group: Select Use existing, and select the resource group you provisioned while creating your cluster in Task 1.
    g. Location: Select the same region you used in Task 1 while creating your cluster.



4. Select **OK** to move to the next step.

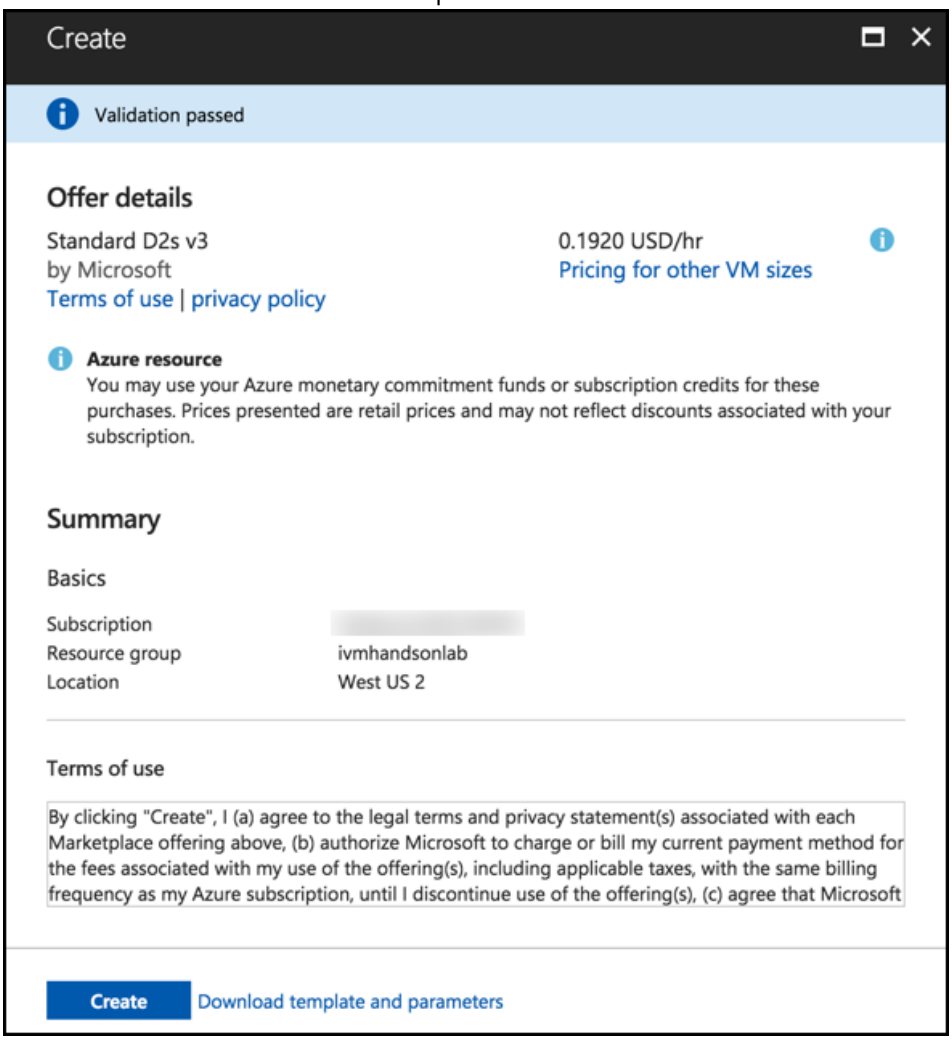
5. On the Choose a size blade, ensure the Supported disk type is set to SSD, and select View all. This machine won't be doing much heavy lifting, so selecting DS2_V3 Standard is a good baseline option.



6. Click **Select** to move on to the Settings blade.

7. Accept all the default values on the Settings blade, and Select **OK**.

8. Select Create on the Create blade to provision the virtual machine.

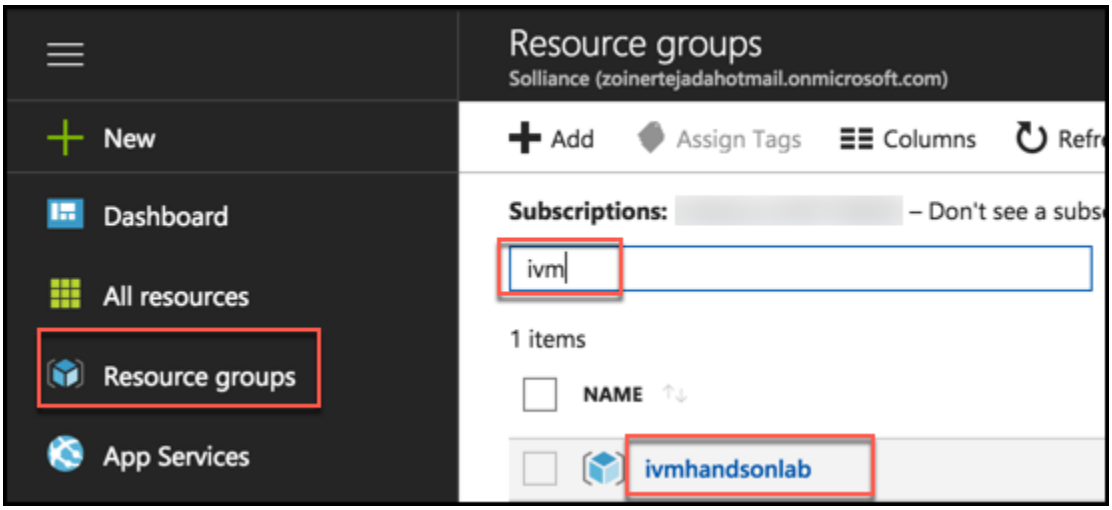


9. It may take 10+ minutes for the virtual machine to complete provisioning.

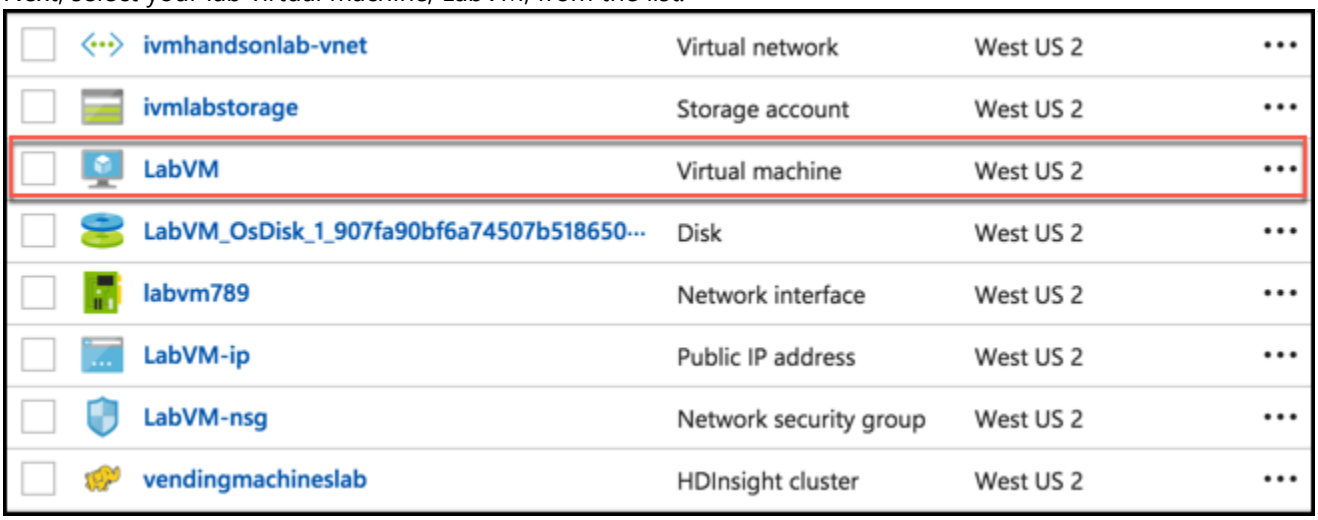
Task 3: Install Power BI Desktop on the lab VM

1. Connect to the **LabVM**. (If you are already connected to your Lab VM, skip to Step 7.)

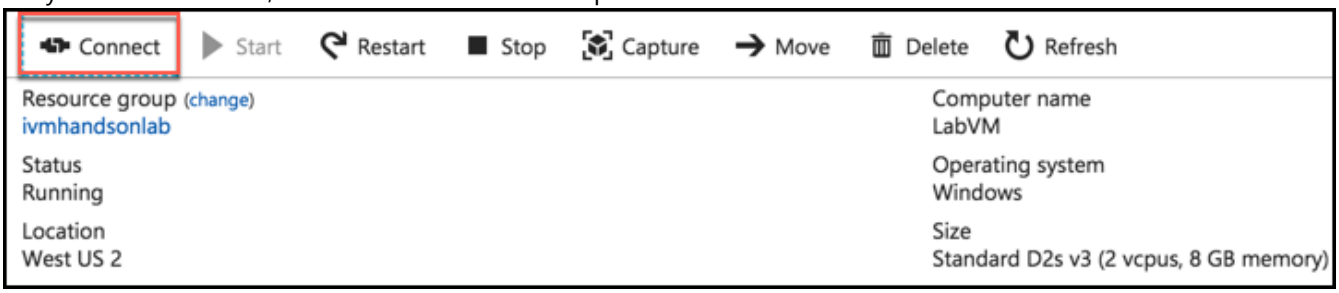
2. From the left side menu in the Azure portal, click on Resource groups, then enter your resource group name into the filter box, and select it from the list.



3. Next, select your lab virtual machine, LabVM, from the list.



4. On your Lab VM blade, select Connect from the top menu.

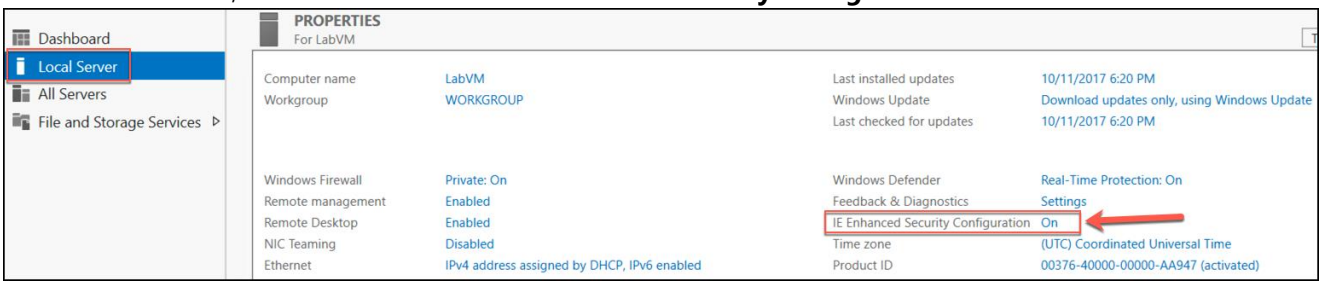


5. Download and open the RDP file.

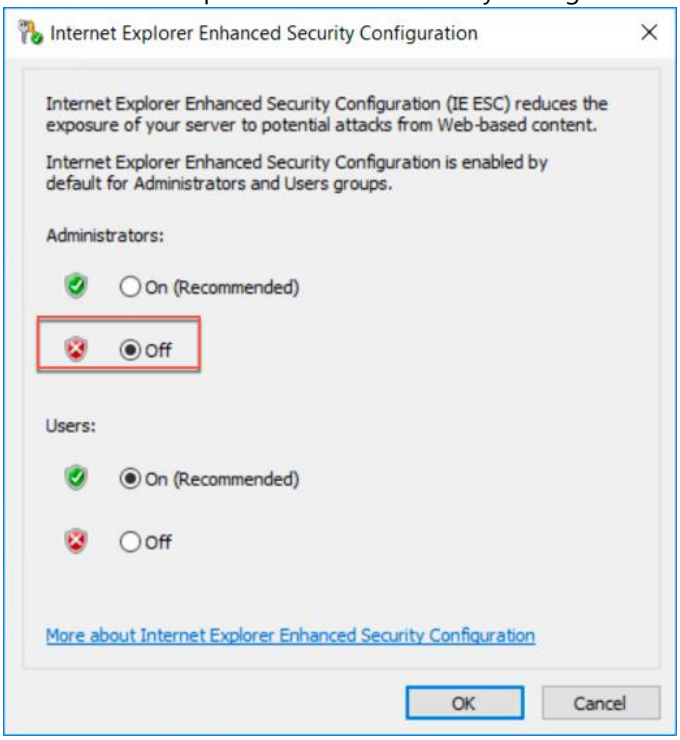
6. Select **Connect**, and enter the following credentials (or the non-default credentials if you changed them):
    a. User name: **demouser**
    b. Password: **Password.1!!**

7. Once logged in, launch the **Server Manager**. This should start automatically, but you can access it via the Start menu if it does not start.

8. Select **Local Server**, then select **On** next to **IE Enhanced Security Configuration**.



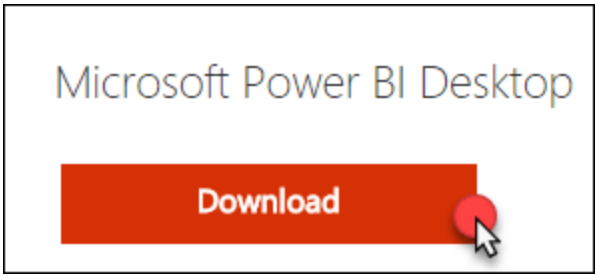
9. In the Internet Explorer Enhanced Security Configuration dialog, select **Off under Administrators**, then select **OK**.



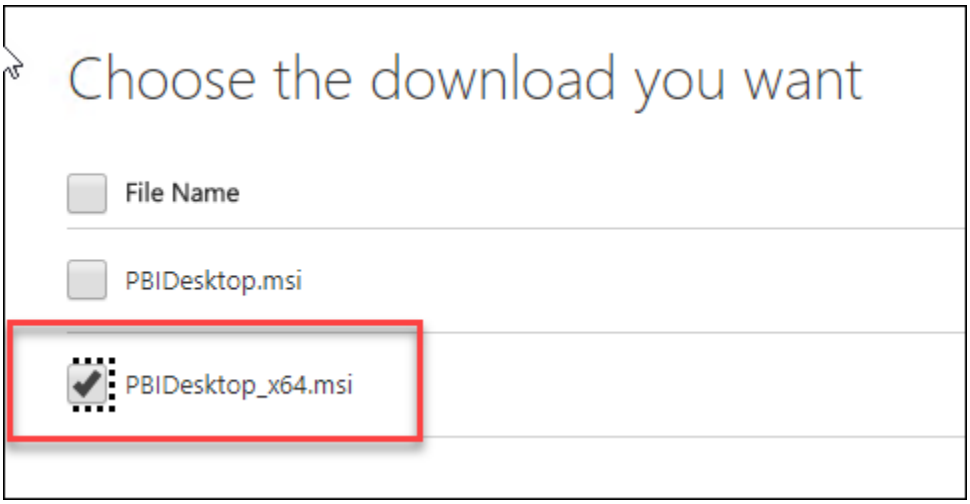
10. Close the Server Manager.

11. In a web browser on the Lab VM navigate to the Power BI Desktop download page https://www.microsoft.com/en-us/download/details.aspx?id=45331

12. Select the **Download Free** link in the middle of the page.

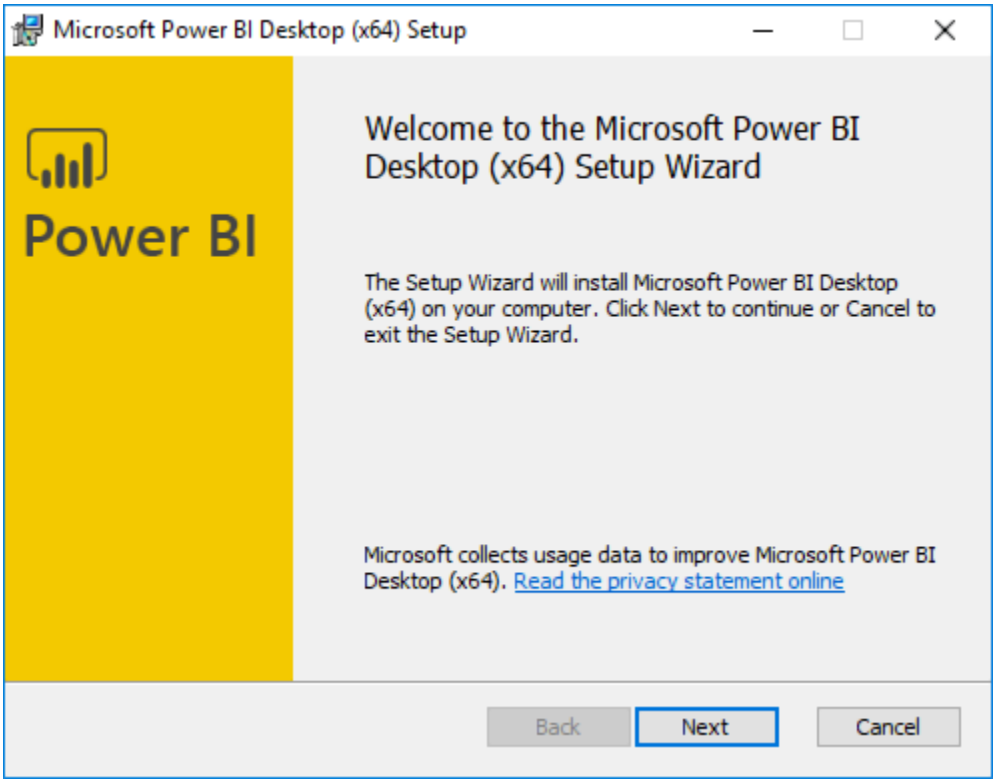


13. Select the **x64 bit version** of the download, then click **Next**.

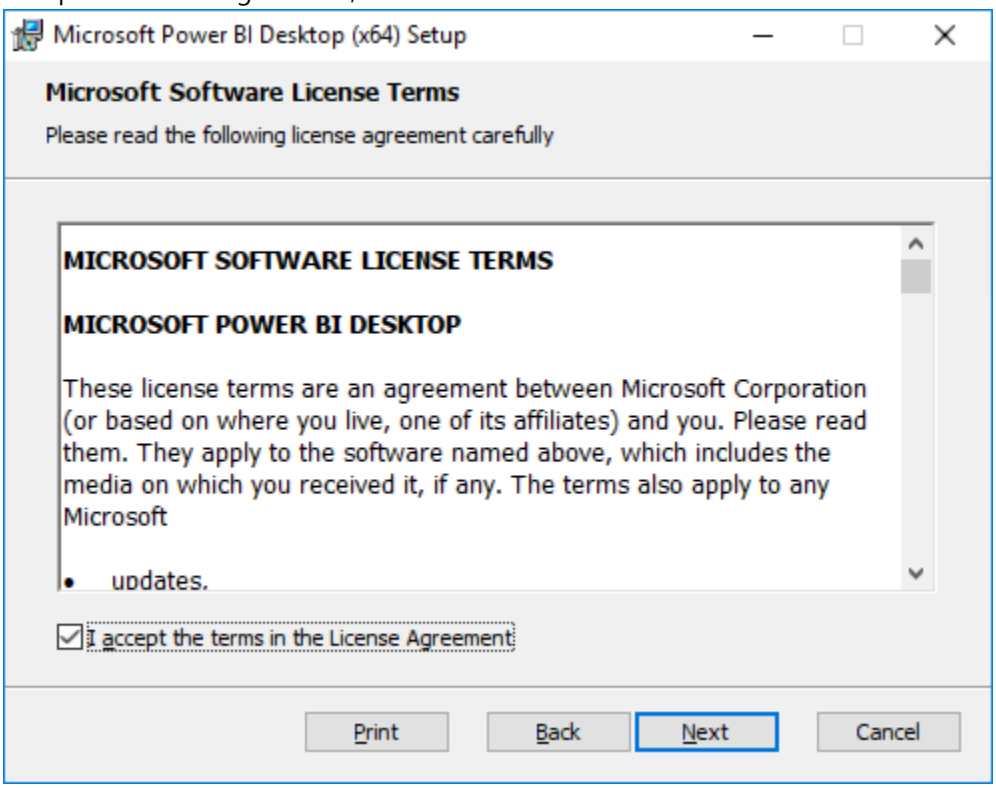


14. Run the installer once it downloads.

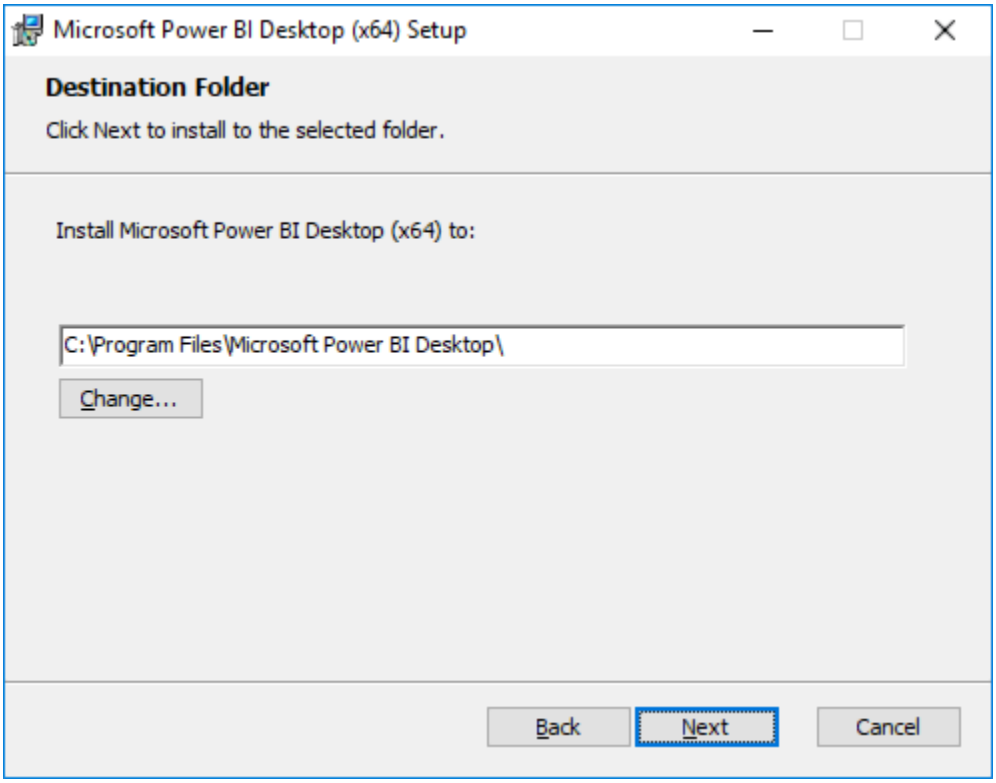
15. Select Next on the welcome screen.



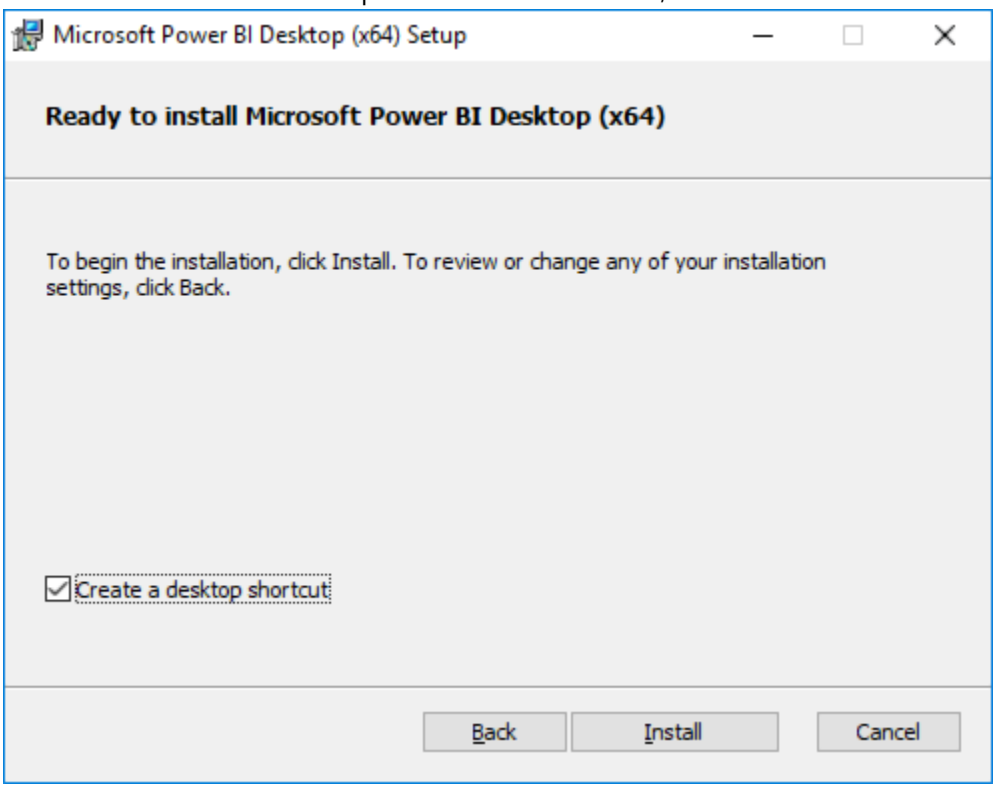
16. Accept the license agreement, and select Next.



17. Leave the default destination folder, and select Next.



18. Make sure the Create a desktop shortcut box is checked, and select Install.



19. Uncheck Launch Microsoft Power BI Desktop, and select Finish.

Task 4: Prepare an SSH client

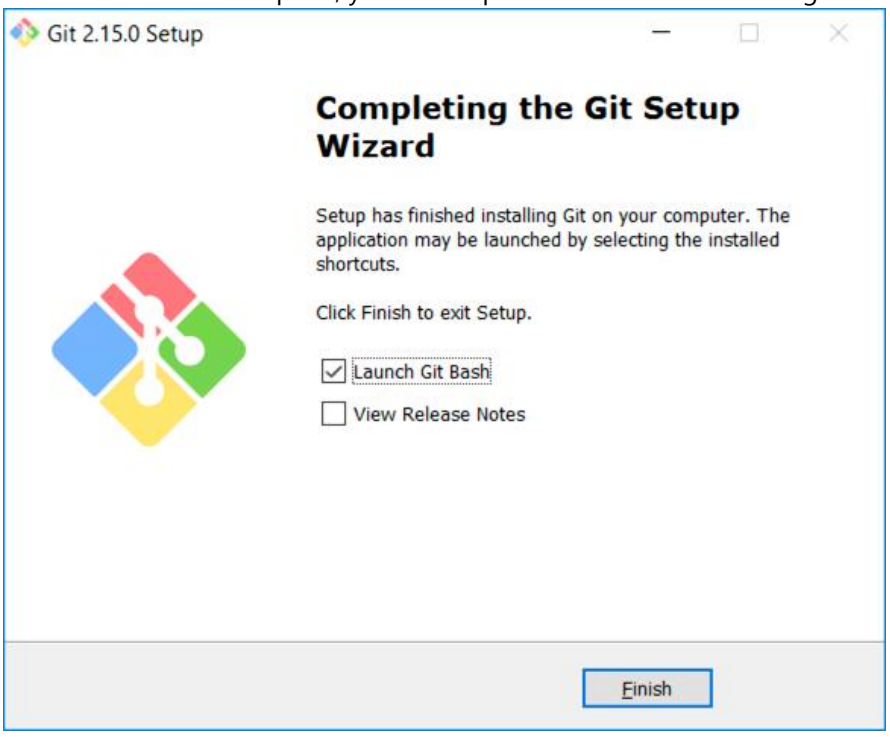
In this task, you will download, install, and prepare the Git Bash SSH client that you will use to access your HDInsight cluster from your Lab VM.

1. On your Lab VM, open a browser, and navigate to https://git-scm.com/downloads to download Git Bash.

2. Select the download for your OS, and then select the Download button.

3. Run the downloaded installer, select Next on each screen to accept the defaults.

4. When the install is complete, you will be presented with the following screen:

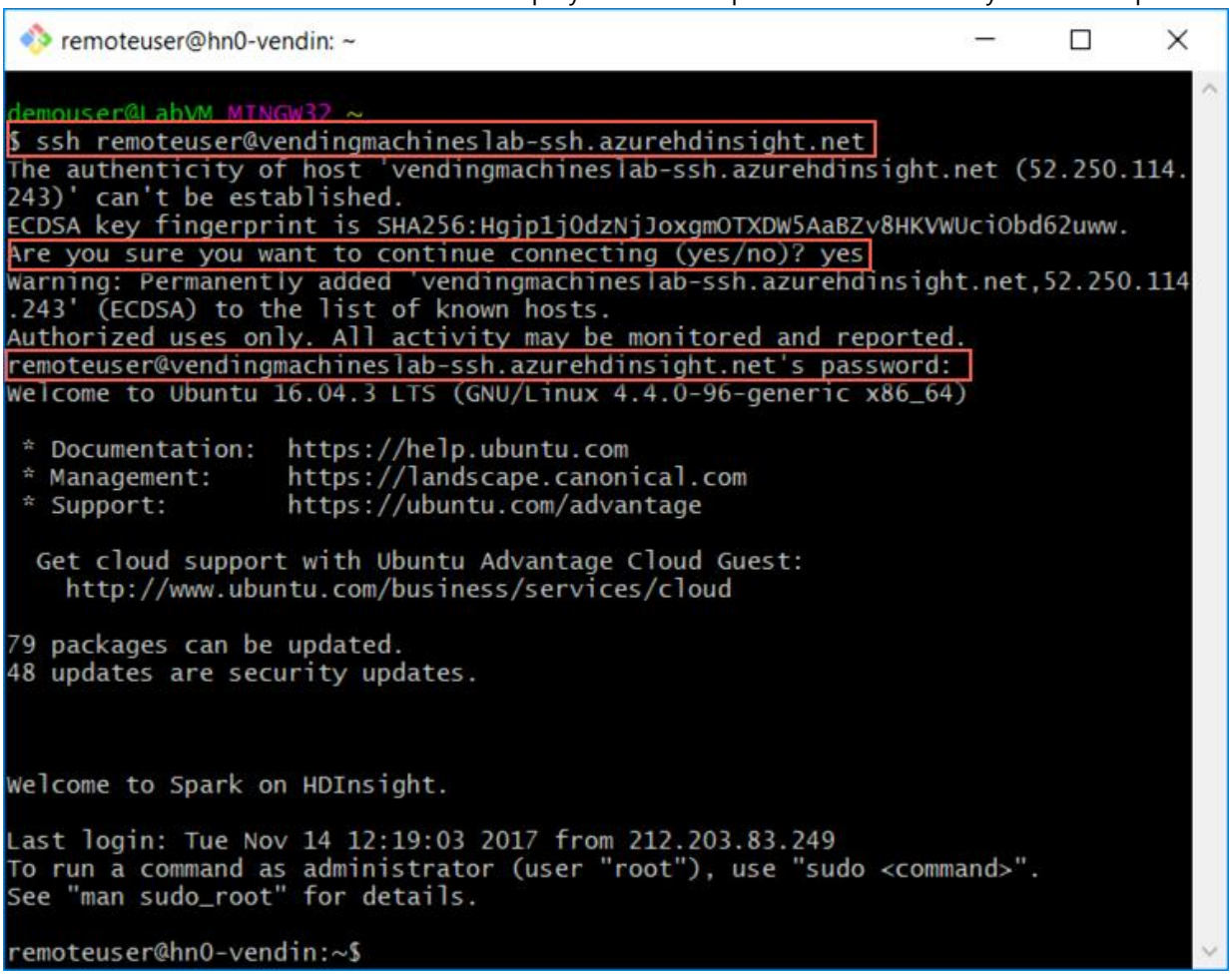


5. Check the Launch Git Bash checkbox, and uncheck View Release Notes. Select **Finish.**

6. The Git Bash client should open in a new window.

7. At the command prompt, enter **ssh remoteuser@<clustername>-ssh.azurehdinsight.net**, replacing <clustername> with the name of the HDInsight cluster created in Task 1 above.

   Note: You will need to wait for the cluster deployment to complete in Azure before you can complete this step.

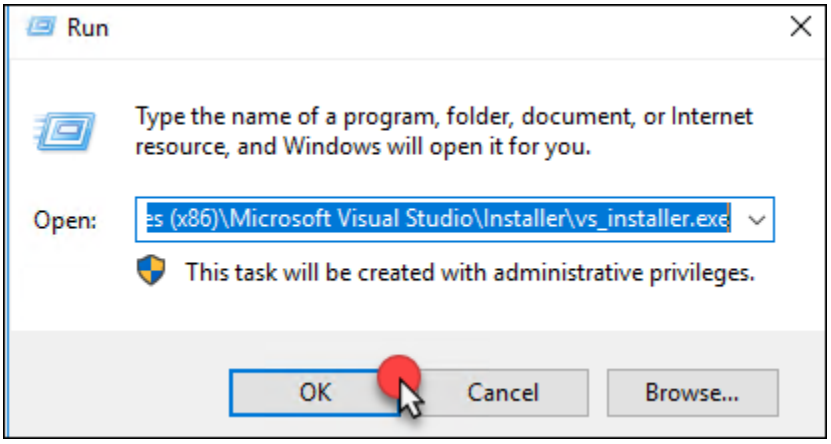


8. Respond to any prompts in the SSH window, and enter the password for **remoteuser** when prompted.

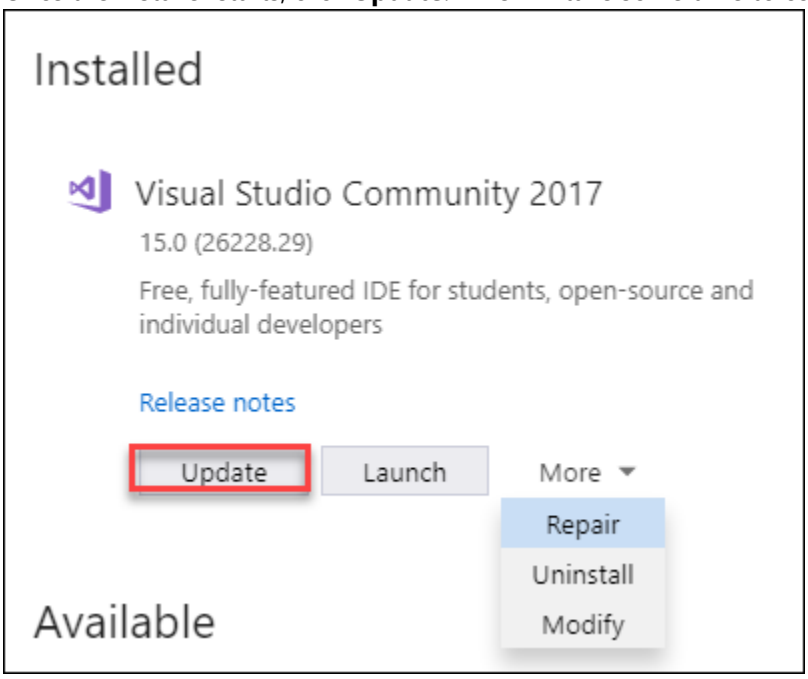
9. Use Git Bash for SSH during the hands-on lab for any instructions requiring an SSH connection. You can repeat these steps any time to re-connect.

Task 5: Install R Tools for Visual Studio 2017

1. Run the Visual Studio installer using Start, Run, C:\Program Files (x86)\Microsoft Visual Studio\Installer\vs_installer.exe.

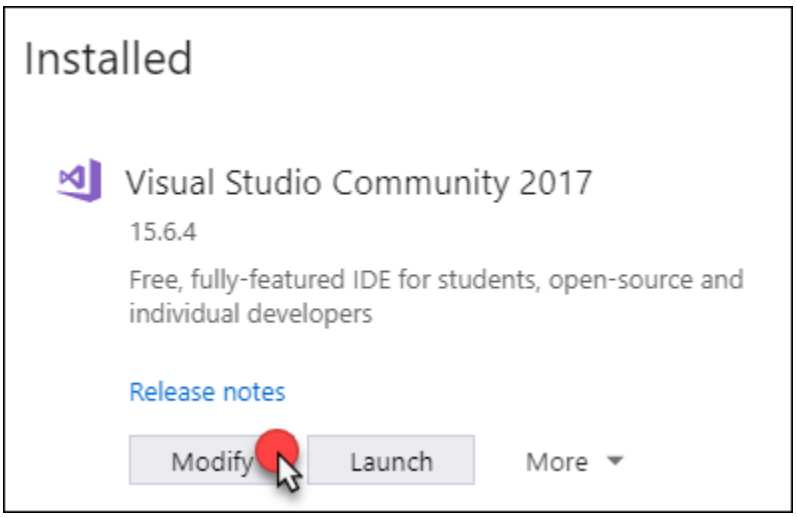


2. Once the Installer starts, click **Update**. This will take some time to complete and require a reboot.

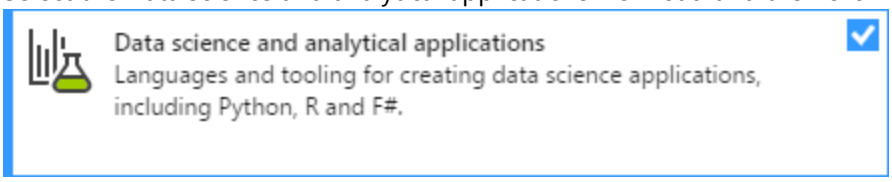


3. Once the update is complete and the VM has restarted, run the installer again. Run the Visual Studio installer using Start, Run, C:\Program Files (x86)\Microsoft Visual Studio\Installer\vs_installer.exe.

4. Next select the **Modify** option.



5. Select the Data science and analytical applications workload and then click Modify.



These steps should be completed prior to starting the rest of the Lab.

Exercise 1: Environment setup

Duration: 45 minutes

Trey Research has provided a starter solution for you. They have asked you to use this as the starting point for creating the Vending Machines solution in Azure.

Task 1: Download and open the vending machines starter project

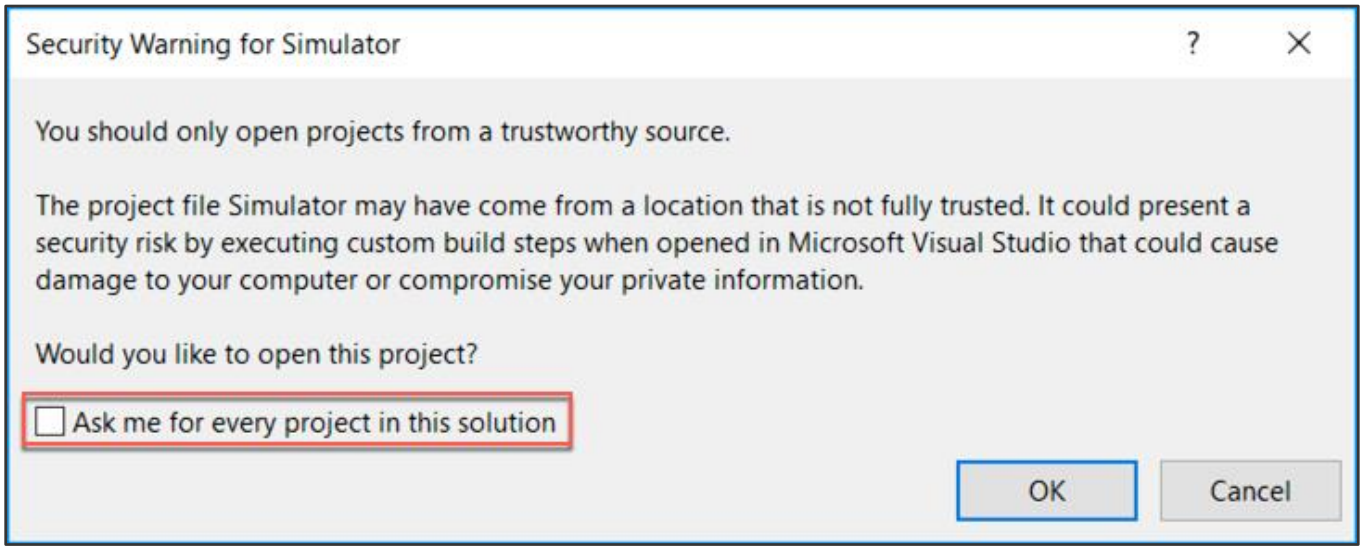
1. From your LabVM, download the starter project from the following URL: http://bit.ly/2w6t2qz

2. Unzip the contents to the folder **C:\VendingMachines\.**

3. Open **VendingMachines.sln** with Visual Studio 2017.

4. Sign in to Visual Studio or create an account, if prompted.

5. If the Security Warning for Simulator window appears, **uncheck Ask me for every project in this solution**, and select **OK**.



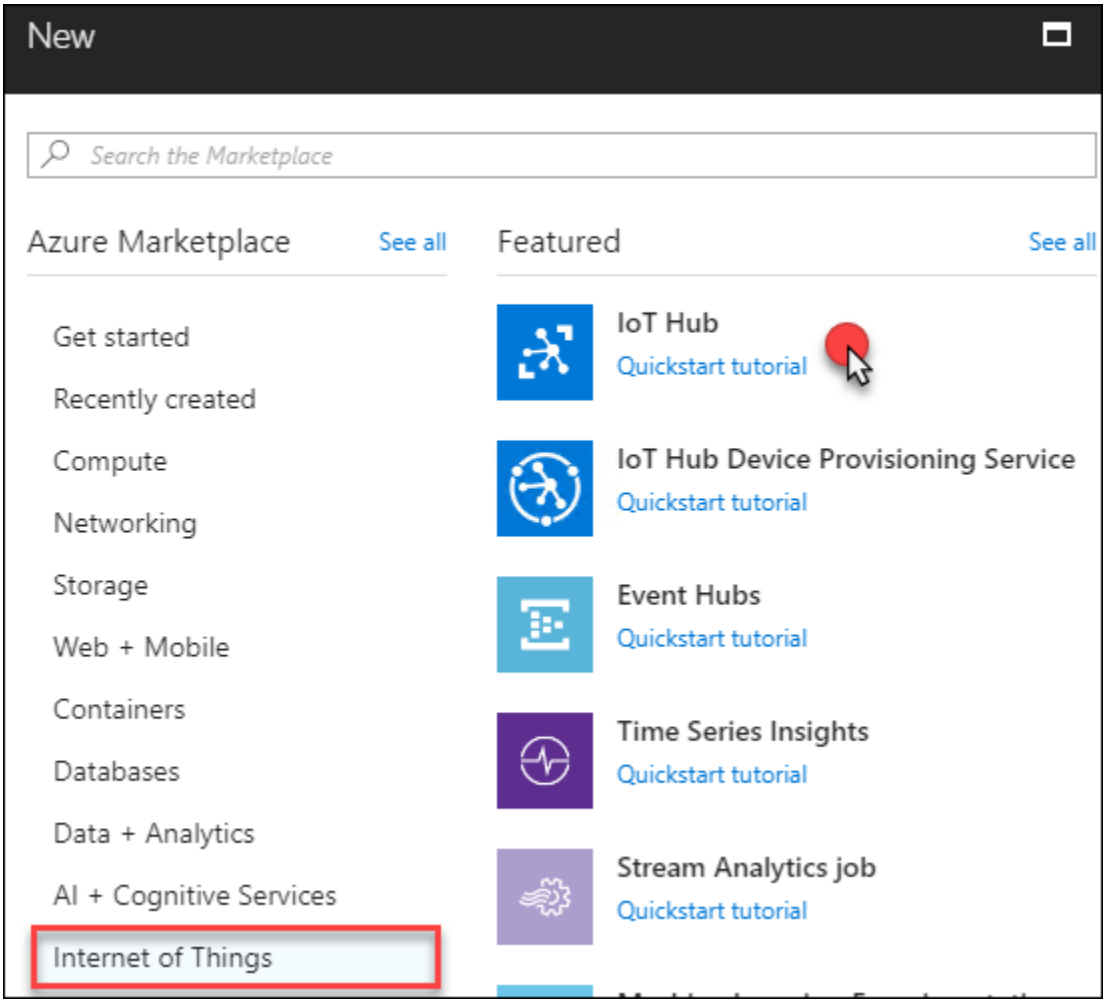
Note: If you attempt to build the solution at this point, you will see many build errors. This is intentional. You will correct these in the exercises that follow.

Task 2: Provision IoT Hub

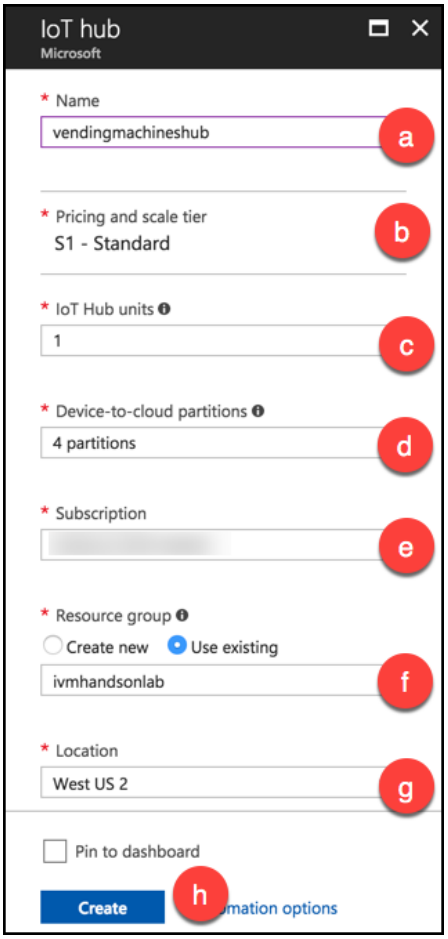
In these steps, you will provision an instance of IoT Hub.

1. In your browser, navigate to the Azure Portal (https://portal.azure.com).

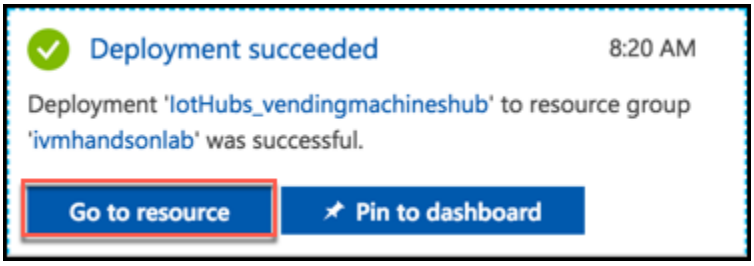
2. Select **+Create a resource**, then select Internet of Things, and select IoT Hub.



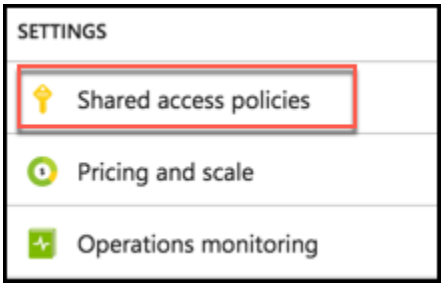
3. In the IoT Hub blade, enter the following:
    a. Name: Provide a name for your new IoT Hub, such as **vendingmachingshub**
    b. Pricing and scale tier: **Select S1 Standard**
    c. IoT Hub units: **Set to 1**
    d. Device-to-cloud partitions: **Select 4 partitions.**
    e. Subscription: Select the same subscription you've been using for previous resources in this lab.
    f. Resource group: Select Use existing, and select the **ivmhandsonlab** resource group you created previously.
    g. Location: Select the location you used previously.
    h. Select **Create**.



4. When the IoT Hub deployment is completed, you will receive a notification in the Azure portal. Select **Go** to resource in the notification.



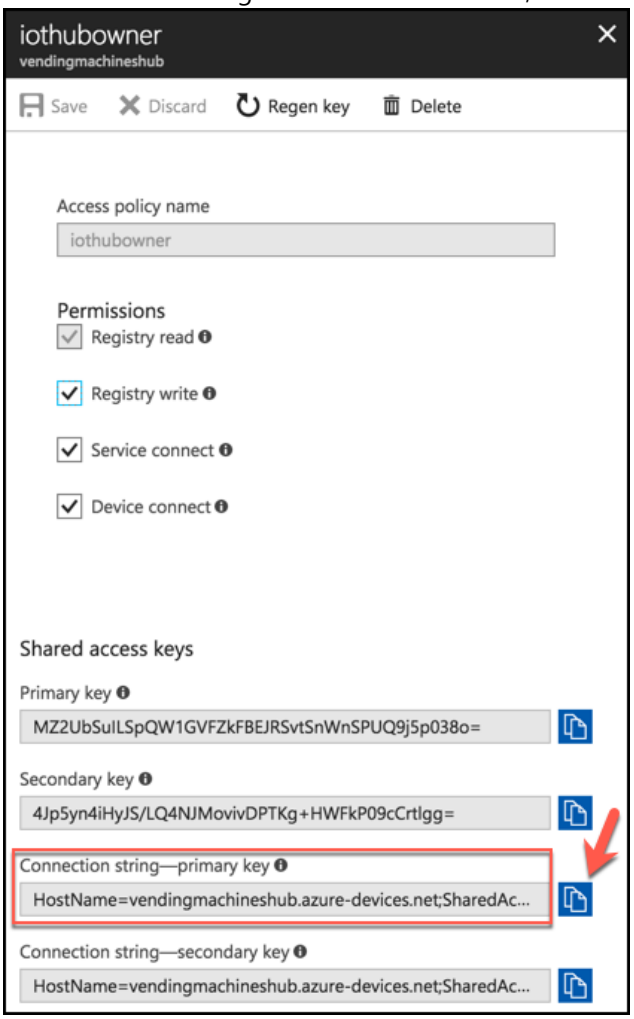
5. From the IoT Hub's Overview blade, select **Shared access policies** under Settings on the left-hand menu.



6. Select **iothubowner** policy.

POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write

7. In the **iothubowner** blade, select the Copy button to the right of the Connection string - primary key field. Paste the connection string value into a text editor, such as Notepad, as this will be needed later in this lab.

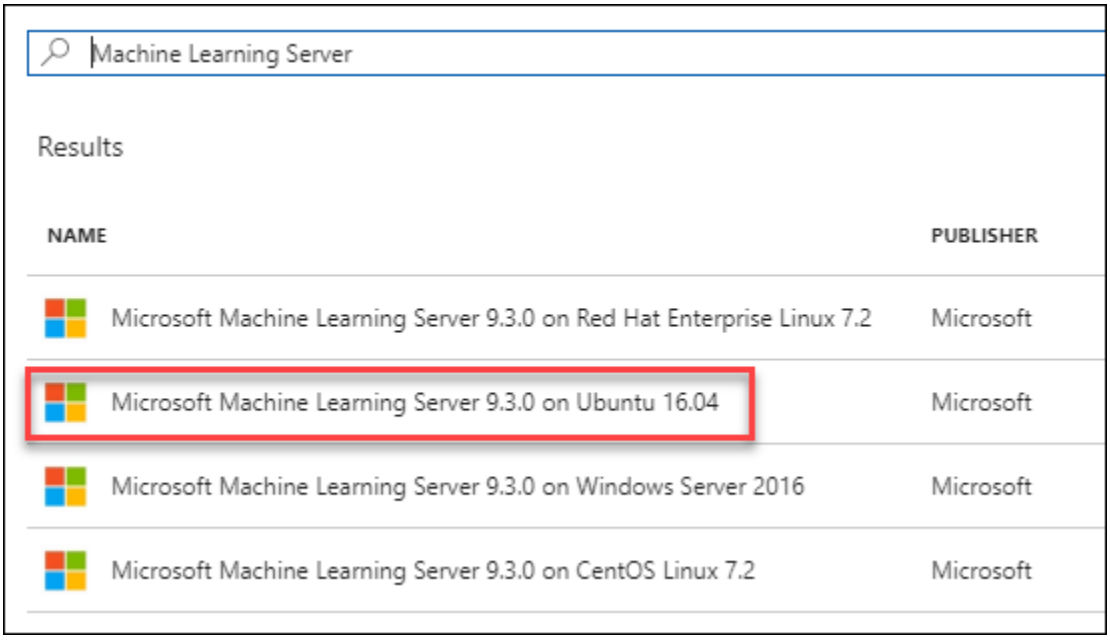


Task 3: Create Microsoft Machine Learning Server on Linux

In these steps, you will provision and configure a Virtual Machine running Microsoft Machine Learning Server. You will use this machine to host the R Server Operationalization service.

1. Within the Azure Portal, select **+Create a resource**, then type **Machine Learning Server** into the Search field.

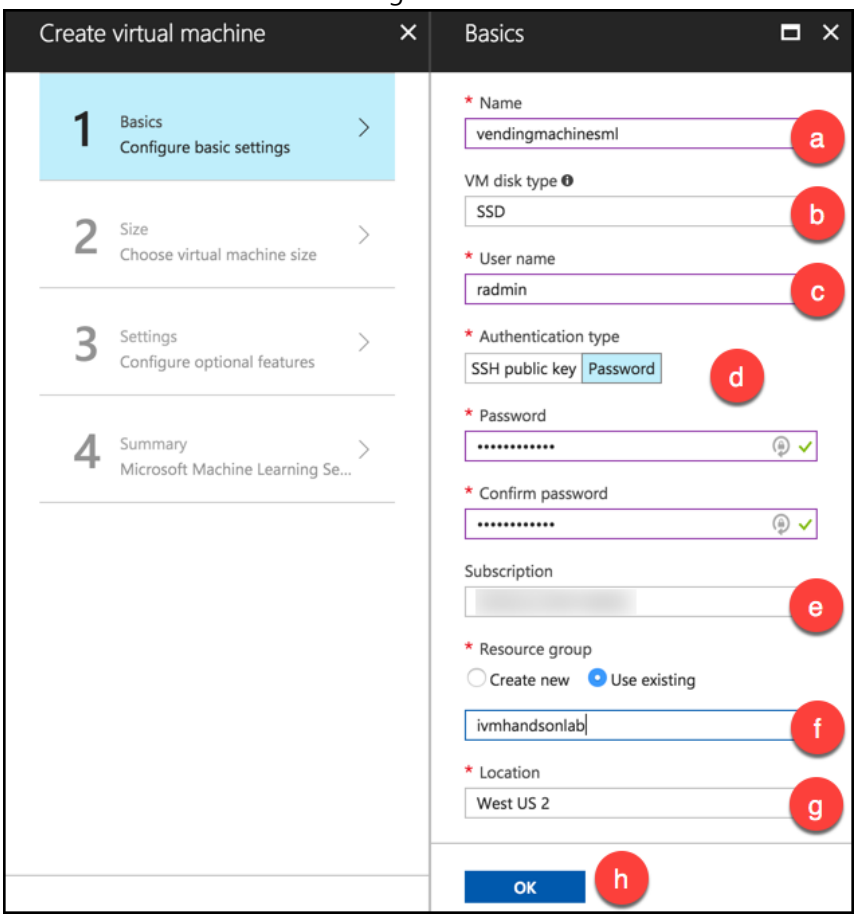
2. In the results list, select **Microsoft Machine Learning Server 9.3.0 on Ubuntu 16.04**.



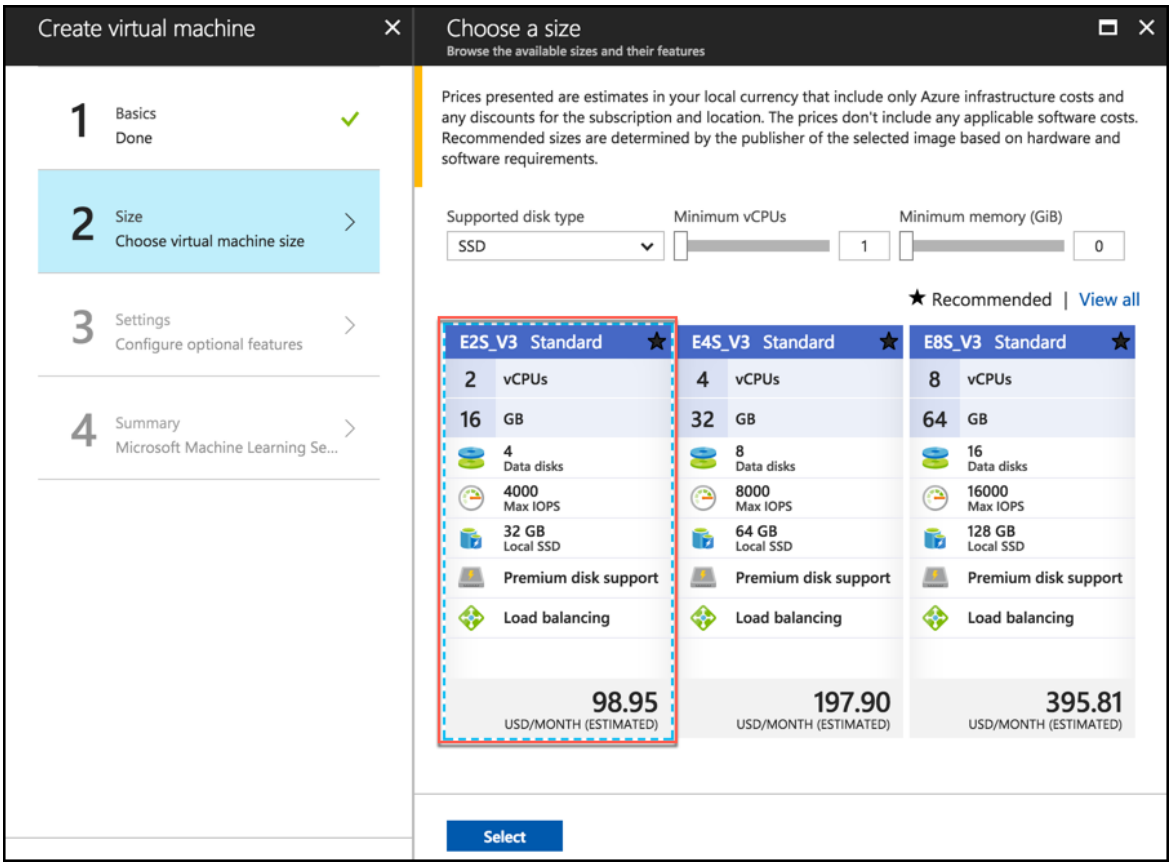
3. On the blade that appears, select **Create.**

4. In the Basics blade, enter:
    a. Name: Enter a **unique name** for the server.
    b. User name: Enter **radmin**.
    c. Authentication type: Select **Password**.
    d. Password: Enter and confirm the password, **Password.1!!**
    e. Subscription: Select the subscription you've been using for this lab.
    f. Resource group: Select **Use existing**, and select the Resource Group you created earlier.
    g. Location: Select the same location used previously.

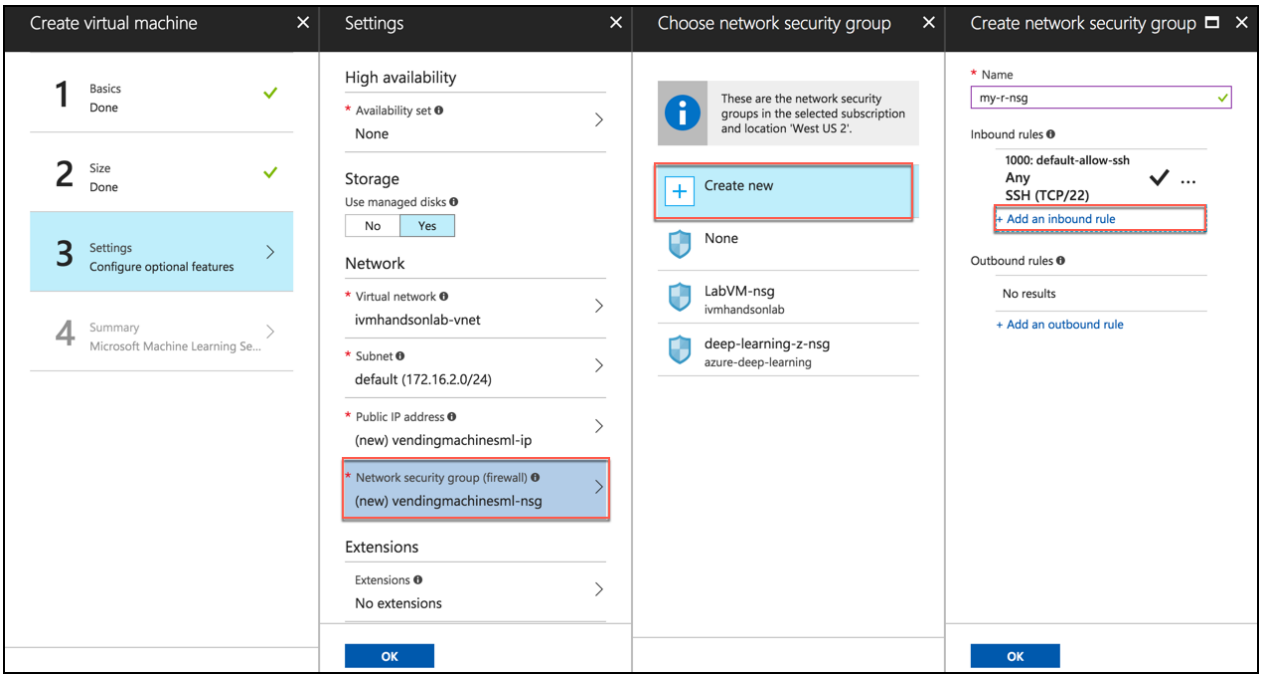
h. Select OK to move on to choosing a VM size.



5. On the Choose a size blade, select **E2S_V3 Standard**, and click **Select**.



6. On the Settings blade, select **Network security group** (firewall).
   a. Click **Create new** under Choose network security group.
   b. Enter a name, such as **my-r-nsg**.
   c. Select **+Add an inbound rule**.



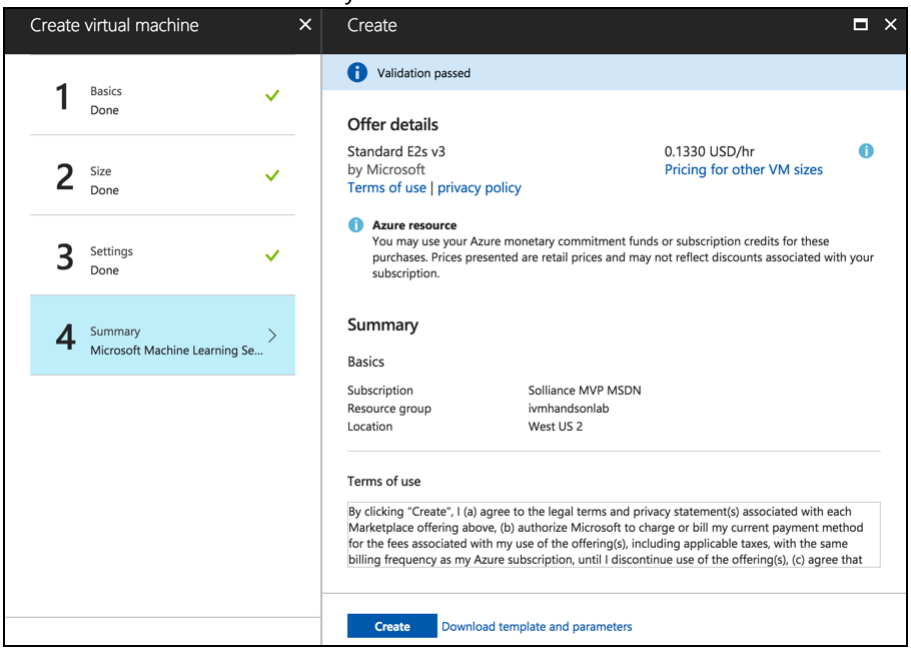
   d. Select **Custom** under Service and enter the following:

i. Port range: **12800**
ii. Priority: **100**
iii. Name: **Port_12800** (should be auto-generated)
iv. Select **OK**

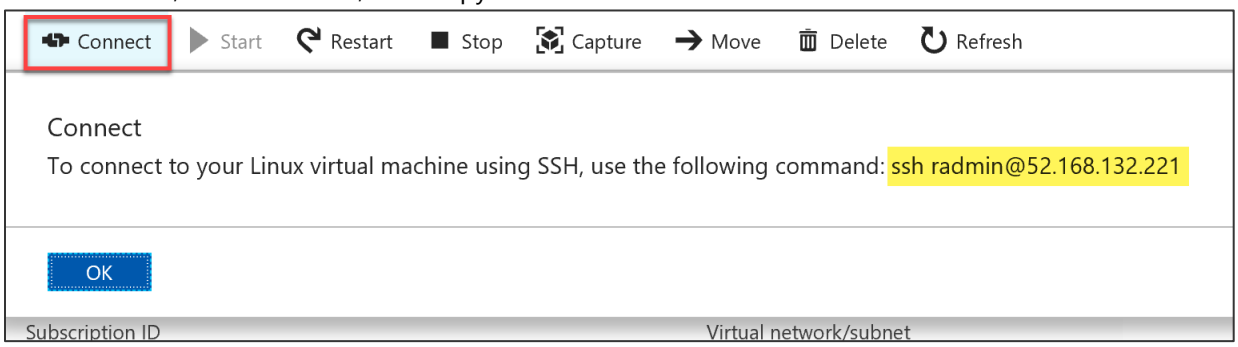
7. Select **OK** on the Create network security group blade.

8. Select **OK** on the Settings blade.

9. Select Create on the Summary blade.



10. Once the machine has provisioned you will need to perform some configurations. On top of the Overview blade of the new server, select Connect, then copy the SSH command.



11. Using a new Git Bash window on your Lab VM, SSH into your Microsoft Machine Learning Server VM by pasting the SSH command you copied above at the command prompt. For example:
ssh radmin@<your-server-ip>.

12. When prompted if you want to continue connecting, enter **yes**.

13. Enter your password, **Password.1!!**

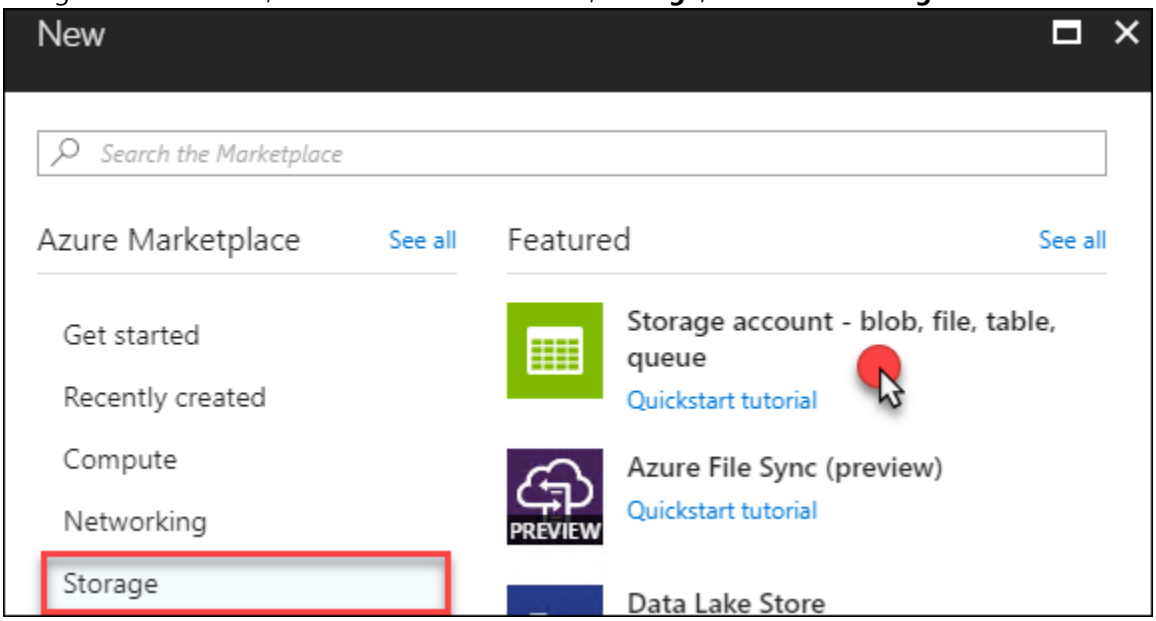
14. At the prompt, after successfully logging in, enter the following command: **sudo apt-get update -y**

15. Type **exit** twice to disconnect from the ssh session.

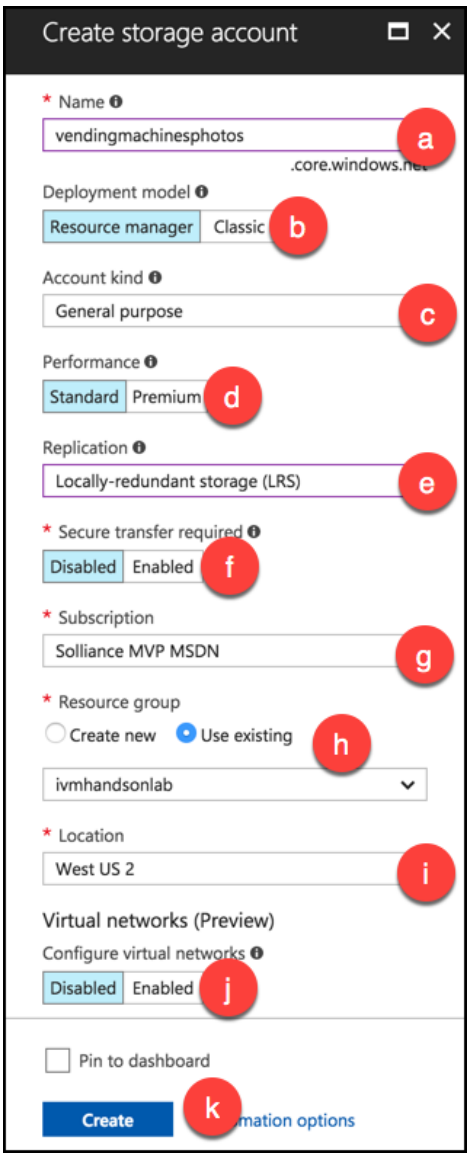
Task 4: Create Storage Account

In these steps, you will provision a storage account that will be used for storing photos sent from the vending machine simulator and for the storage of the promotional package resources.

1. Using the Azure Portal, select **+Create a resource**, **Storage**, then select **Storage account**.

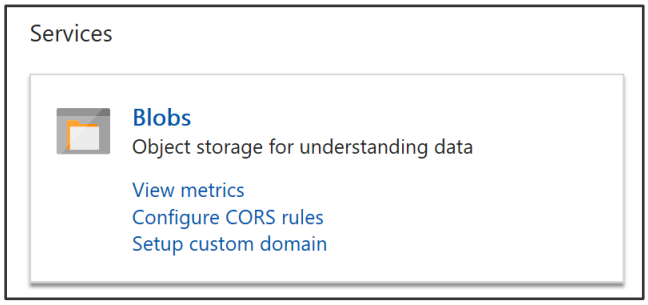


2. On the Create storage account blade, enter the following values:
    a. Name: Enter a **unique name** for the storage account.
    b. Deployment model: Leave **Resource Manager** selected.
    c. Account kind: Leave set at **General purpose**.
    d. Performance: Leave set to **Standard**.
    e. Replication: Set to **Locally-redundant storage (LRS).**
    f. Storage service encryption: Leave as **Disabled**.
    g. Subscription: Select your Subscription.
    h. Resource group: Select the Use existing radio button, and select **ivmhandsonlab** from the resource group list.
    i. Location: Select a Location to be consistent with the other resources you have created.
    j. Virtual networks : Leave set to **Disabled.**
    k. Select Create.

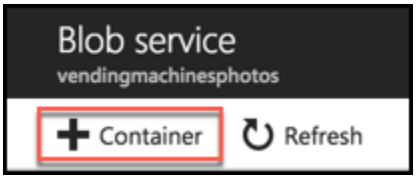


3. Navigate to the newly created storage account in the Azure Portal by clicking on Storage Accounts, and selecting it from the list of available storage accounts.

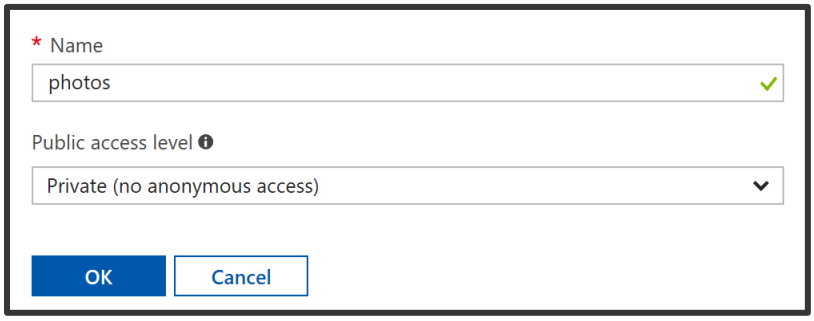
4. On the Storage account blade, select Blobs.



5. In the Blob service blade, select **+Container** from the command bar.



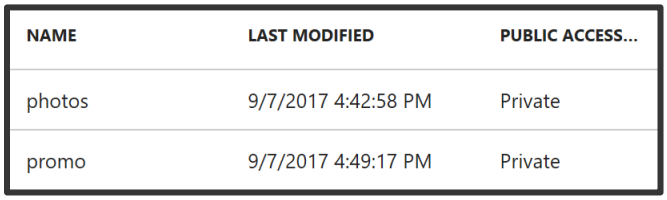
6. On the New container blade, set the name to "**photos**" and select **Private** as the Access type.



7. Click **OK**.

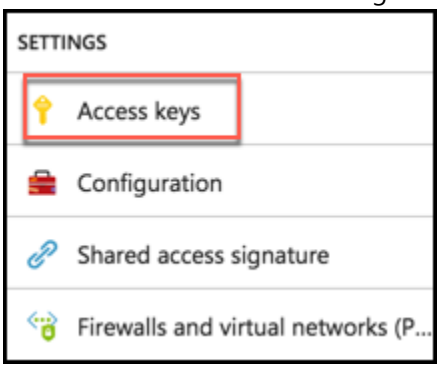
8. Repeat steps 6-8 to create another container named "**promo**".

9. You should now see both containers listed on the Blob service blade.



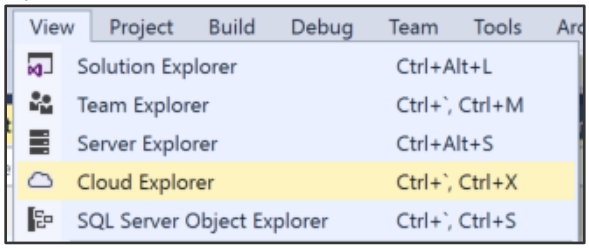
NAME	LAST MODIFIED	PUBLIC ACCESS...
photos	9/7/2017 4:42:58 PM	Private
promo	9/7/2017 4:49:17 PM	Private

10. Close the Blob service blade to go back to the Storage blade. Select **Access Keys** from the left-hand menu.



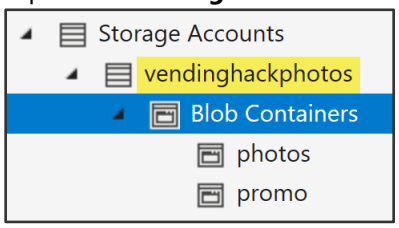
11. Use the copy button to the right of the Connection String for key1 to copy your storage connection string. Save the copied value to a text editor, such as Notepad, as this will be used later on.

12. Open Visual Studio and from the **View Menu** select **Cloud Explorer**.

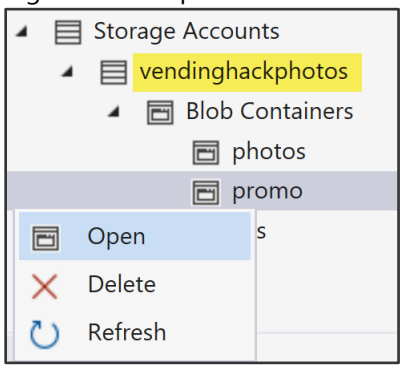


Note: You may need to select your subscription, but clicking the person icon and expanding the subscriptions.

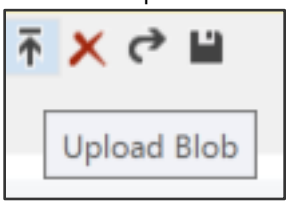
13. Expand the **Storage account** that you just created, and the **Blob Containers** item underneath it.



14. Right-click the promo container and select Open.



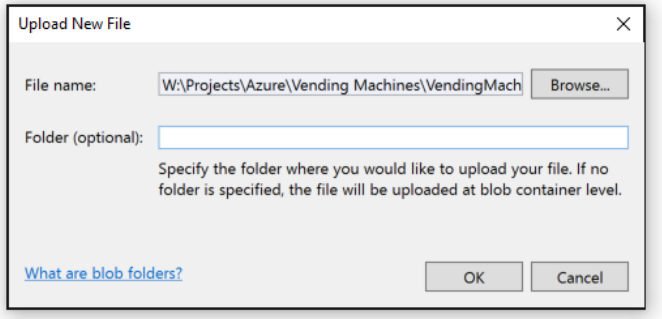
15. Select the Upload blob button.



16. Select **Browse**.

17. In the dialog, select the three images **CoconutWater.png, Water.png, and Soda.png** from the starter solution **Simulator\Images** folder and select **Open**.

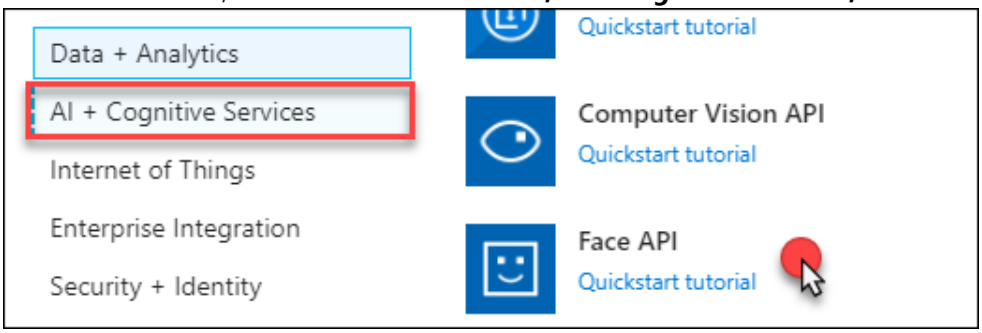
18. Select **OK** on the Upload New File Dialog to upload the images to the container.



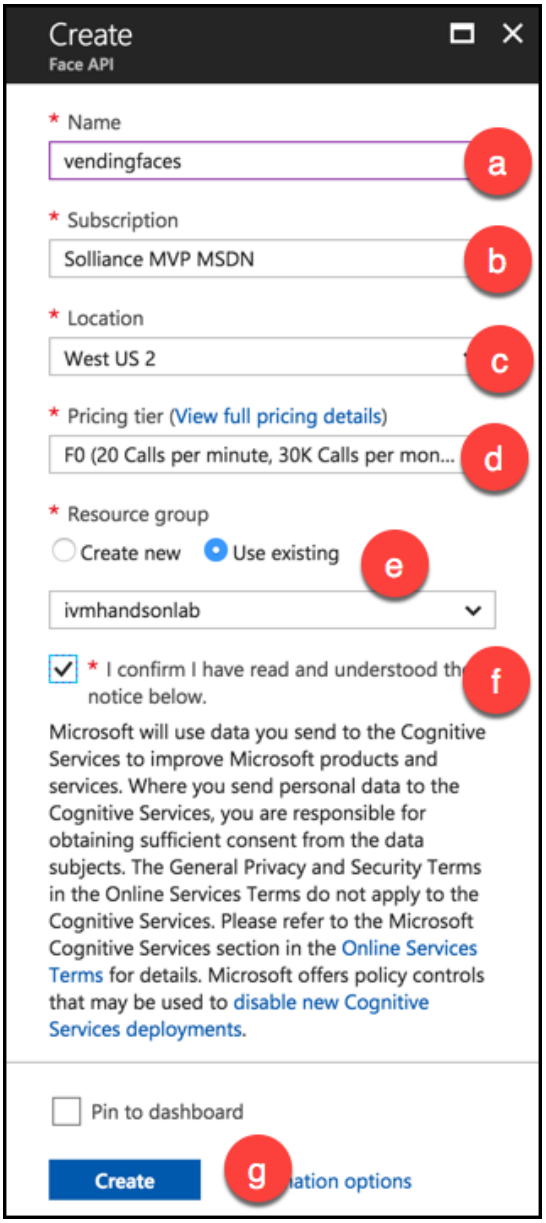
Task 5: Provision Cognitive Services Face API

To provision access to the Face API (which provides demographic information about photos of human subjects), you will need to provision a Cognitive Services account.

1. In the Azure Portal, select **+Create a resource, AI + Cognitive Services,** and select **Face API**.

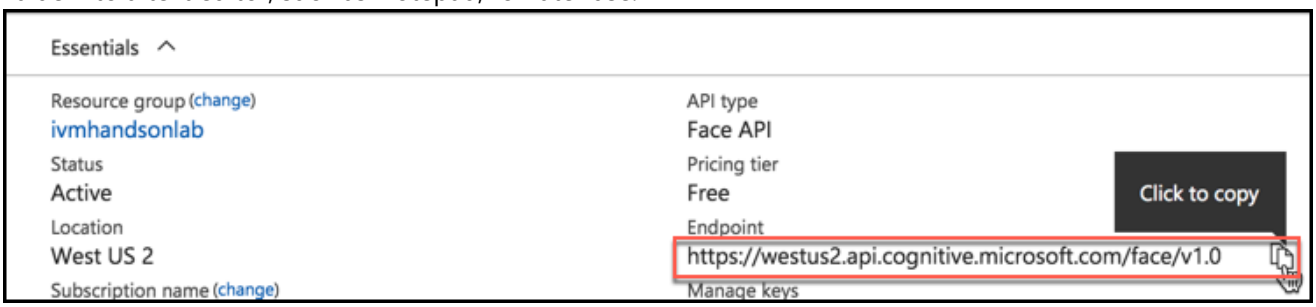


2. On the Create Face API blade:
    a. Name: Enter a name.
    b. Subscription: Choose your Subscription.
    c. Location: Choose the location you've been using for other resources in this lab.
    d. Pricing tier: Select the **Free tier (F0)** from the drop-down list.
    e. Resource group: Select Use existing, and select the **ivmhandsonlab** resource group from the list.
    f. Check the box confirming you have read and understand the legal terms.
    g. Click Create to provision the Cognitive Services account.

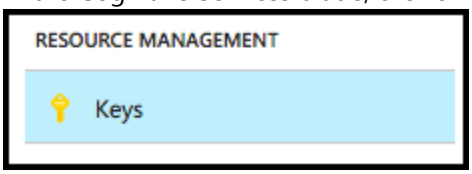


3. When the Face API finishes provisioning, browse to the Cognitive Services Face API by clicking on Go to resource in the Deployment succeeded notification.

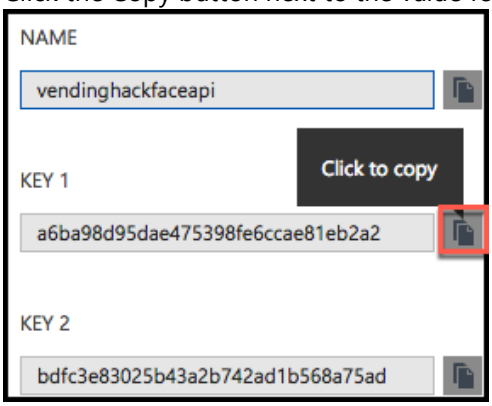
4. On top of the Cognitive Services overview blade, click the Copy button to the right of the Endpoint. Paste this value into a text editor, such as Notepad, for later use.



5. In the Cognitive Services blade, click on Keys under the Resource Management heading.



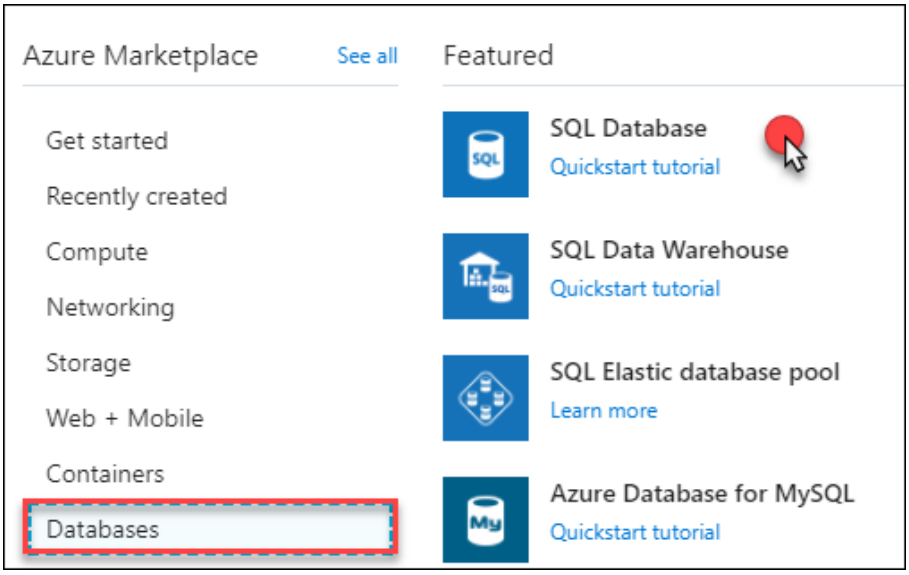
6. Click the Copy button next to the value for Key 1. Paste this value into a text editor, such as Notepad, for later use.



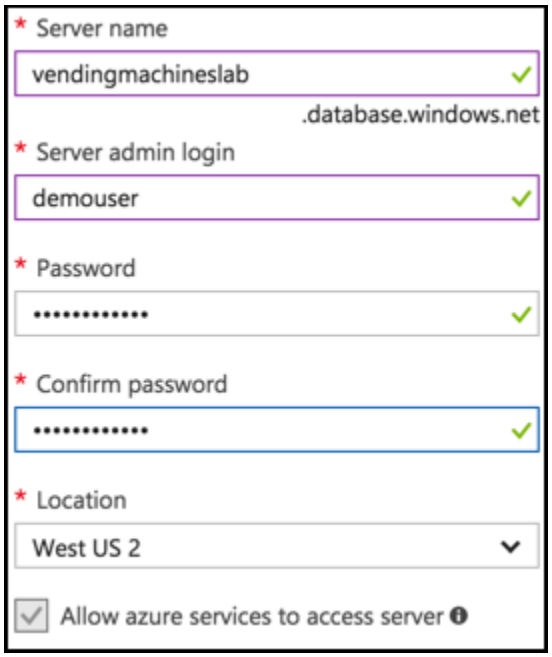
Task 6: Provision SQL Database

In these steps, you will provision a SQL database to support the transactions and real-time analytics.

1. In the Azure Portal, select **+Create a resource**, select **Databases**, then select **SQL Database**.

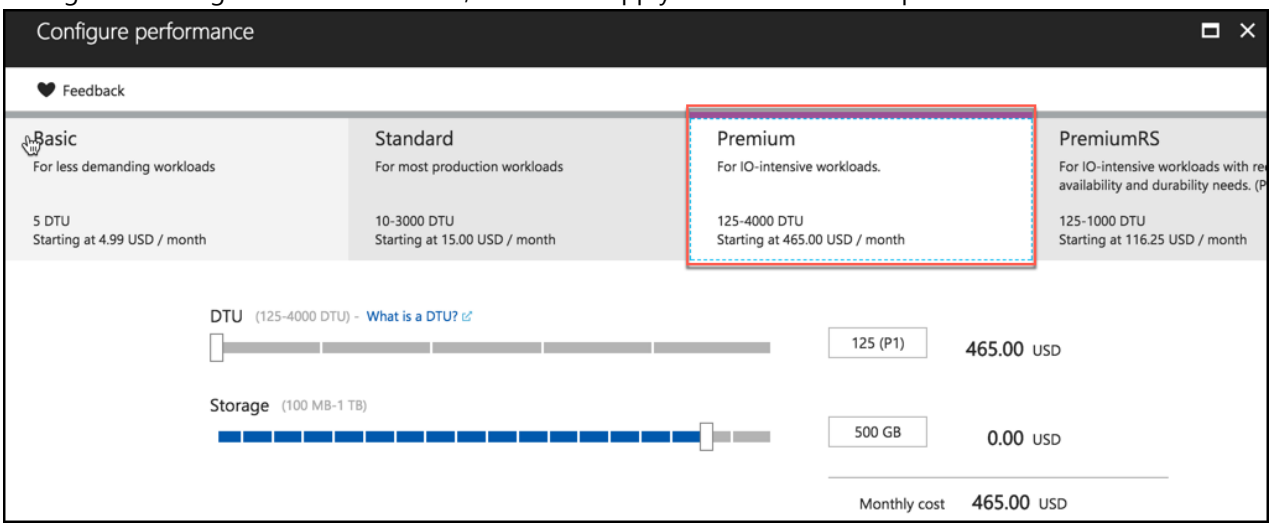


2. In the SQL Database blade, enter the following:
    a. Database name: Enter **vending**.
    b. Subscription: Choose your Subscription.
    c. Resource Group: Select Use existing, and choose **ivmhandsonlab** from the resource group list.
    d. Source: Leave source as **Blank** database.
    e. Server: Select **configure required settings**
        i. Server name: Enter a unique server name
        ii. User name: Enter **demouser**
        iii. Password: Enter **Password.1!!**
        iv. Location: Select the same location you've used for other resources in this lab.
        v. Click Select.



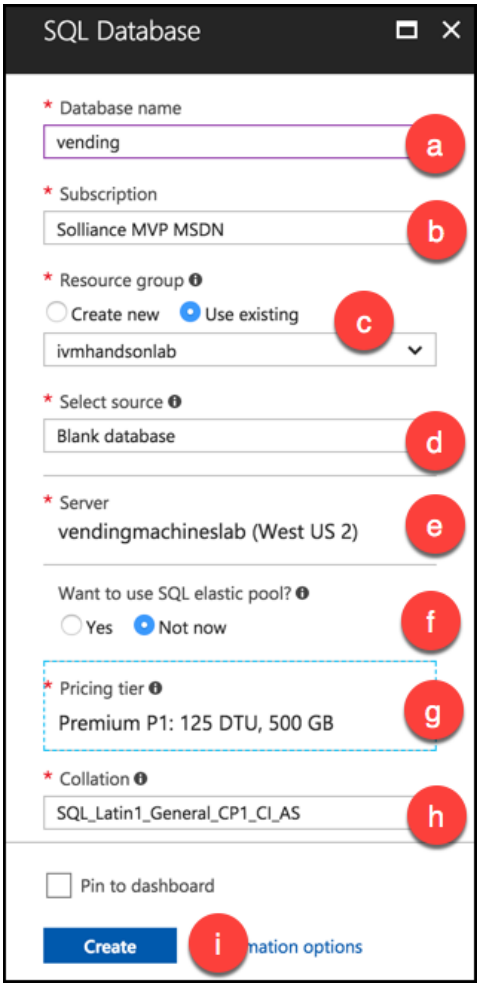
    f. Want to use SQL elastic pool: Leave set to Not now.

g. Change the Pricing tier to **Premium P1**, and select Apply. Premium tier is required for Columnar indexes.

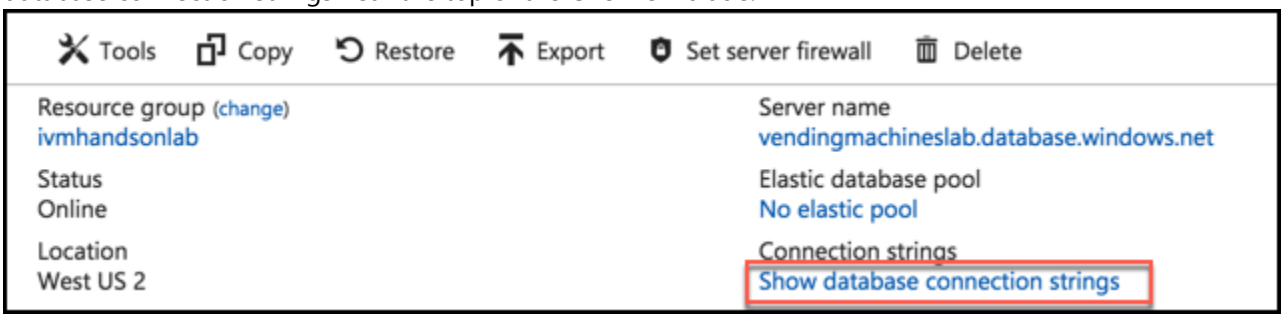


h. Collation: Leave set to the default value (SQL_Latin1_General_CP1_CI_AS).

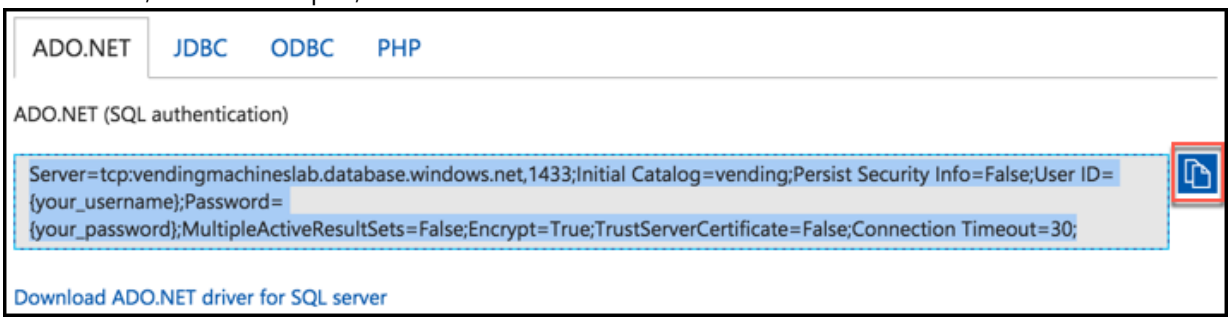
i. Select **Create**.



3. Once the SQL Database finishes provisioning, navigate to the database in the Azure portal and select the Show database connection strings near the top of the Overview blade.



4. Copy the connection string on the ADO.NET tab of the Database connection string blade, and paste the value into a text editor, such as Notepad, for later reference.



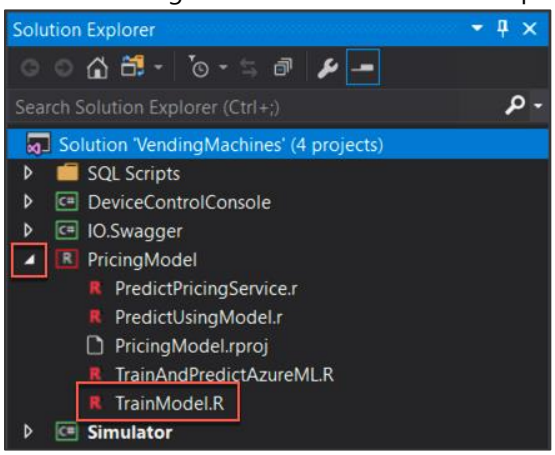
Exercise 2: Create Dynamic Pricing Model

Duration: 45 minutes

In this exercise, you will create a machine learning model that predicts the purchase price for an item sold by the vending machine, provided the demographics of the customer and the item. You will then operationalize this model by exposing it as a web service hosted in Azure Machine Learning, and test it out.

Task 1: Create a model locally

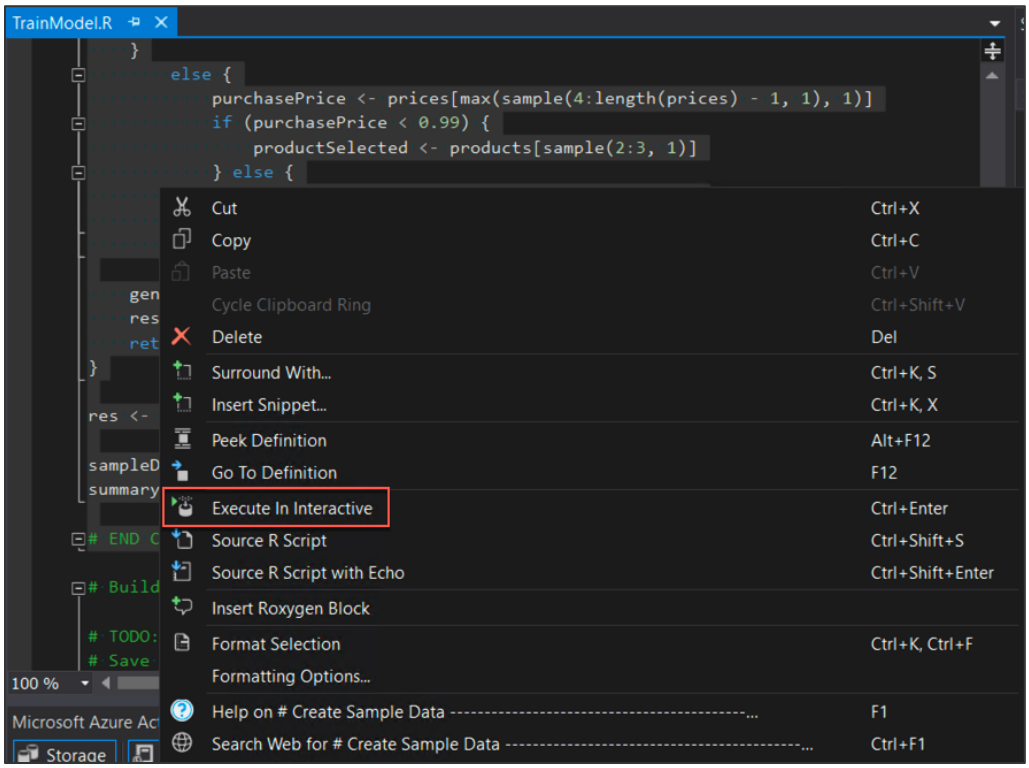
1. Within Visual Studio Solution Explorer, expand the **PricingModel** project and open the file **TrainModel.R** by double-clicking on the file in the Solution Explorer.



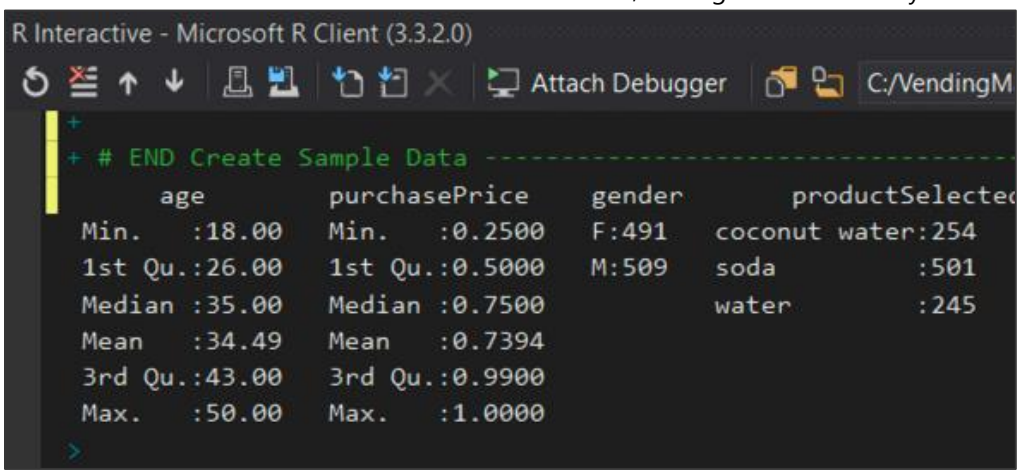
2. Read the script. The top portion, entitled Create Sample Data, has been provided for you and you will generate the sample data you will use to train your model.

3. Highlight all the text between the “Create Sample Data” and “END Create Sample Data” comments.

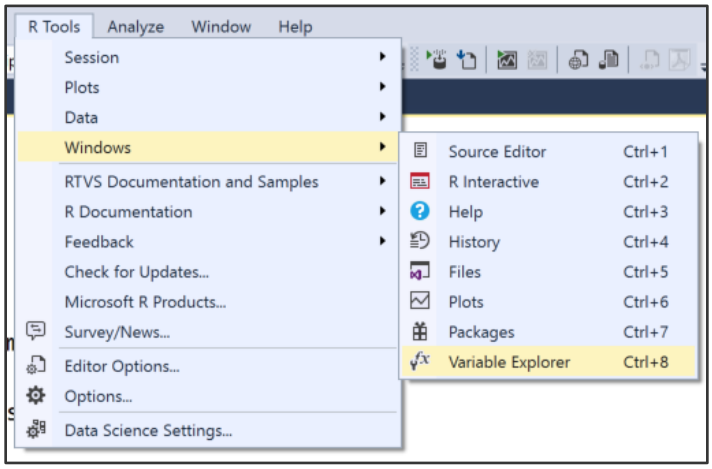
4. Right-click the selected text and select Execute In Interactive.



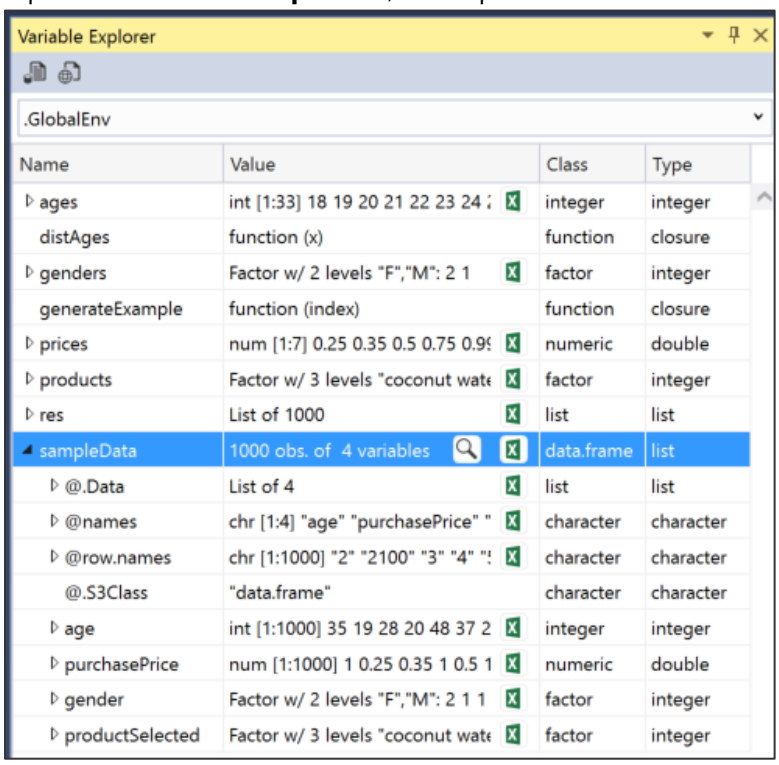
5. You should see it execute in the R Interactive Window, ending with a summary of the created data.



6. From the R Tools menu, select Windows and Variable Explorer.



7. Expand the variable **sampleData**, and explore the structure of the created data.

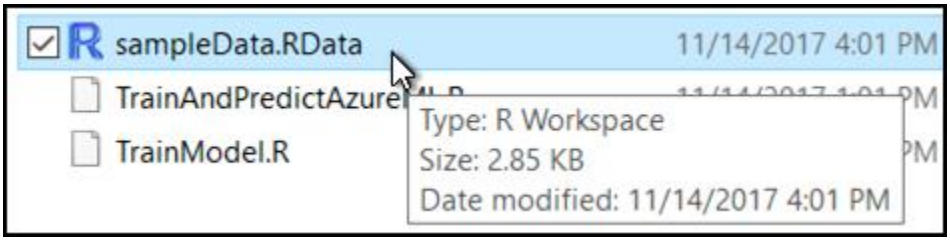


8. Now save this sampleData to a file by replacing TODO 1 in the TrainModel.R script with the following code:

```
# TODO: 1. Export the sample data to a file

save(sampleData, file = "sampleData.RData")
```

9. Highlight the **save line**, and select **Execute In Interactive**.

10. Open File Explorer and navigate to the location of the **PricingModel** (C:\VendingMachines\VendingMachines - Clean\PricingModel) project on disk. **You should see** the file **sampleData.RData** on disk.



11. Back in the **TrainModel.R** file in Visual Studio, replace TODO 2 with the following code that builds the model using a Linear Regression.

```
# TODO: 2. Build a linear regression model to predict purchase price given age, gender and
# productSelect

pricingModel <- rxLinMod(purchasePrice ~ age + gender + productSelected, data = sampleData)
```

12. Save that trained model to disk by replacing TODO 3 with:

```
# TODO: 3. Export the trained model to a file named pricingModel.rda

save(pricingModel, file = "pricingModel.RData")
```

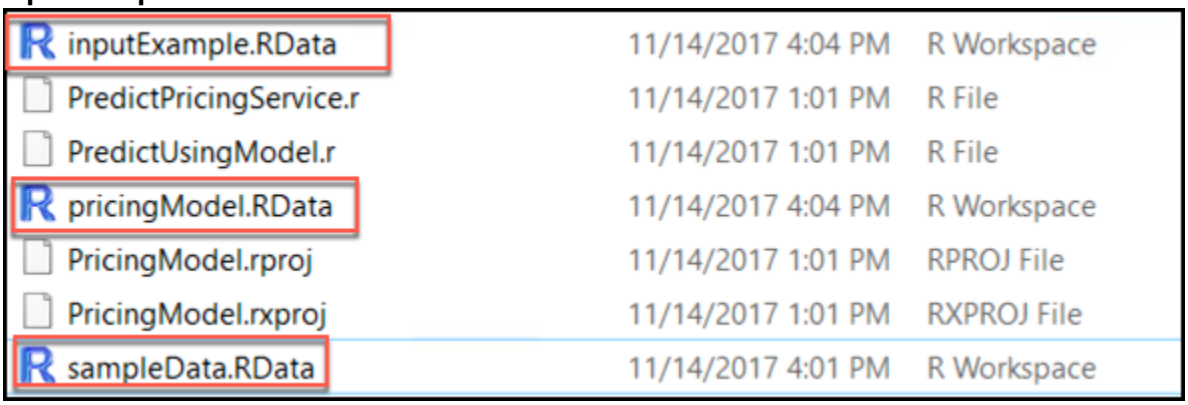
13. Finally, save the first row of the sample data to a file so you can re-use the structure later when operationalizing the model. Replace TODO 4 with:

```
# TODO: 4. Save one example of the sample data to serve as an input template, to a file
# called inputExample.rda

inputExample <- sampleData[1,]

save(inputExample, file = "inputExample.RData")
```

14. Save your changes to **TrainModel.R**.

15. Highlight TODO items 2 through 4 and execute them in interactive.

16. In the same folder as your script, you should now have the files **sampleData.RData**, **pricingModel.RData**, and **inputExample.RData**.



Task 2: Try a prediction locally

1. Within Visual Studio, open **PredictUsingModel.r** under the Pricing Model project in Solution Explorer.

2. Replace TODO 1 with the following:

```
# TODO: 1. Prepare the input to use for prediction

inputExample[1,]$age <- 30

inputExample[1,]$gender <- "F"

inputExample[1,]$productSelected <- "coconut water"
```

3. Replace TODO 2 with the following:

```
# TODO: 2. Execute the prediction

prediction <- rxPredict(pricingModel, data = inputExample)
```

4. Highlight all the script in the file and execute it in interactive.

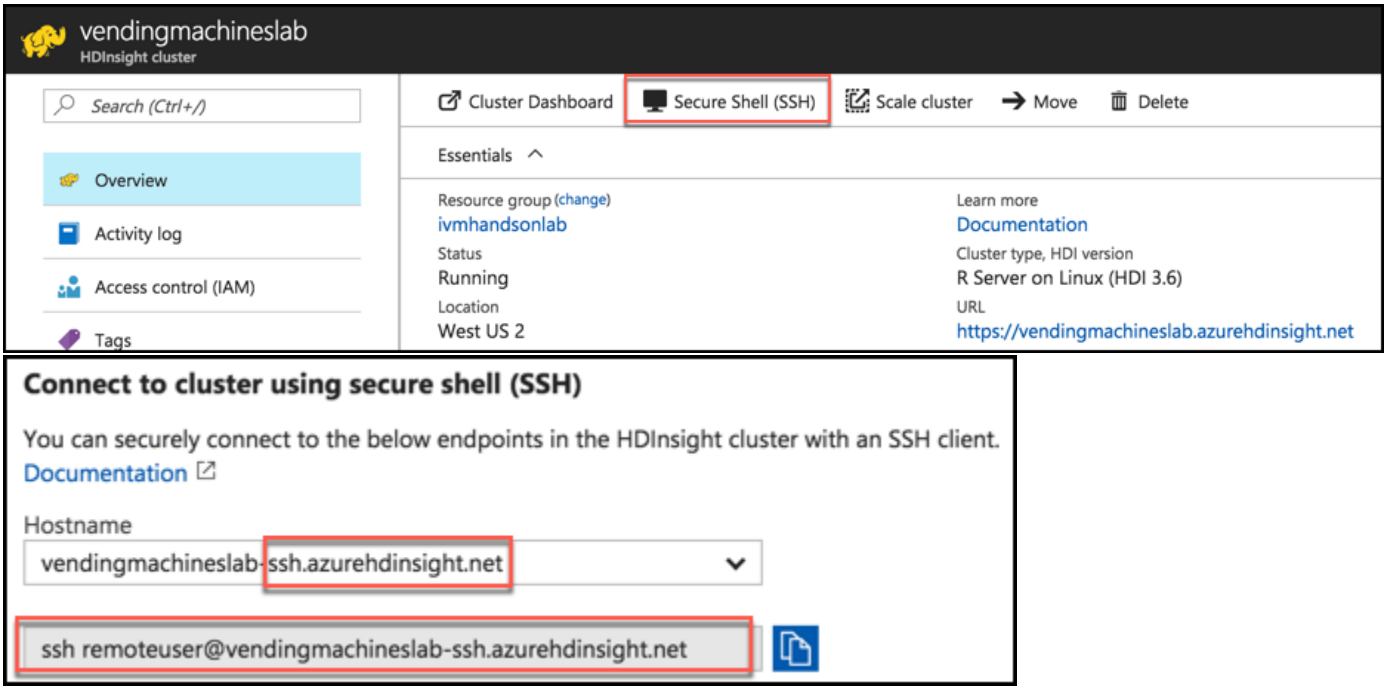
5. Using Variable Explorer, expand the prediction variable and observe the price the model suggested to use for purchasing the coconut water for input of a 30-year-old female.

prediction	1 obs. of 1 variable	data.frame	list
@.Data	List of 1	list	list
@names	"purchasePrice_Pred"	character	character
@row.names	1	integer	integer
@.S3Class	"data.frame"	character	character
purchasePrice_Prec	0.949	numeric	double

Task 3: Create the model in R Server on HDInsight

1. On your LabVM, open a Git Bash shell like you did in the Before the Hands-on Lab, Task 4, Step 7.

2. SSH into your deployed R Server in HDInsight cluster. (You can get the SSH connection string for your cluster from the HDInsight Blade in the Azure Portal).



3. If prompted to continue connecting, enter yes.

4. Enter your password.

5. At the command prompt, type **R** to load the R shell (be sure to use a capital letter "R").

6. Run the following command to create a spark context for R:

```
sparkCluster <- RxSpark()

rxSetComputeContext(sparkCluster)
```

7. In Visual Studio, open **TrainModel.r**, and copy the entire script.

8. Paste the script in the R shell, and press ENTER. (You may need to press ENTER a few times until you get to the last line of the script.)

9. When the script has finished executing, type the following:

```
dir()
```

10. You should see it list the three files created by the script, as follows:

```
> dir()
[1] "inputExample.RData" "pricingModel.RData" "sampleData.RData"
```

11. Now, copy those files from local storage to Blob storage by using the Hadoop File System. First, create a folder in which to store your output.

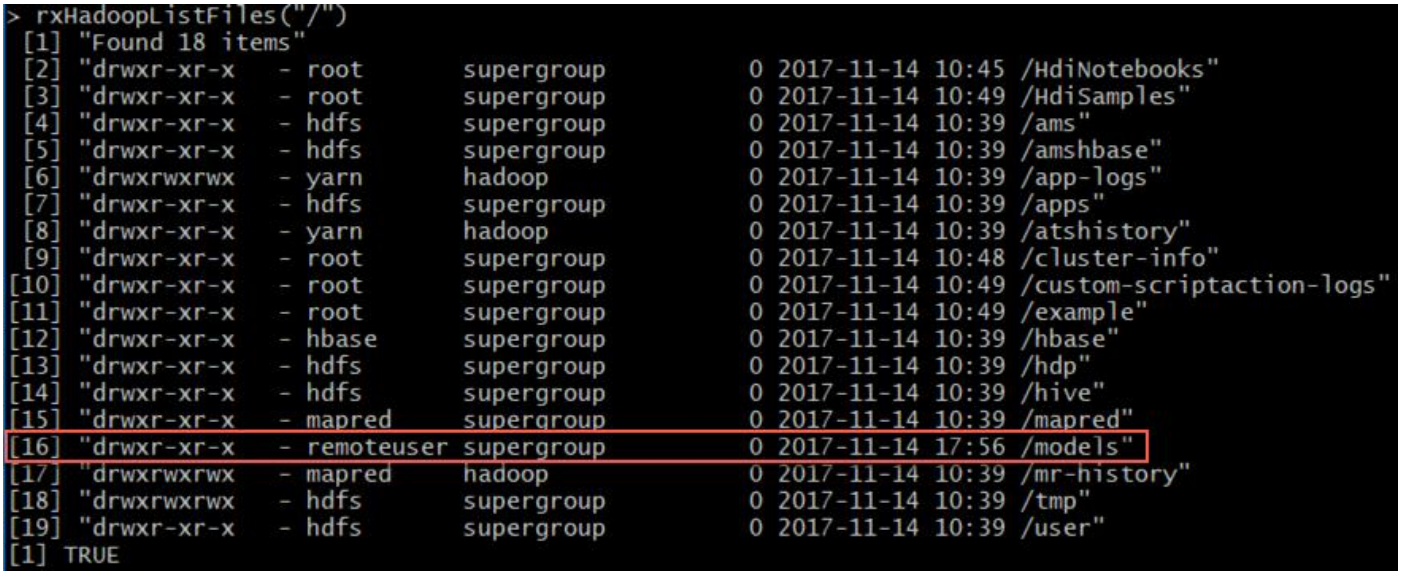
```
modelExportDir <- "/models"
```



```
rxHadoopMakeDir(modelExportDir)
```

12. List the contents of the root ("/") directory, and confirm your "**/models**" folder has been created. Notice that the list you are looking at is folders directly underneath the container in Azure Storage that was created with your cluster.

    ```
    rxHadoopListFiles("/")
    ```



13. Copy the **pricingModel.RData** from the local directory to the **/models** folder in HDFS by running the following command:

    ```
    rxHadoopCopyFromLocal("pricingModel.RData", modelExportDir)
    ```

14. Repeat the previous step for inputExample.RData and sampleData.RData.

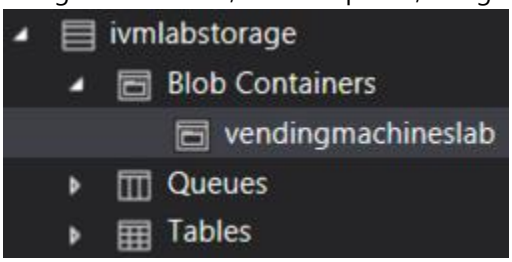
15. Run the following command to verify the three files now exist in HDFS (and Blob storage), under /models.

    ```
    rxHadoopListFiles("/models")
    ```

16. The output should look similar to the following:

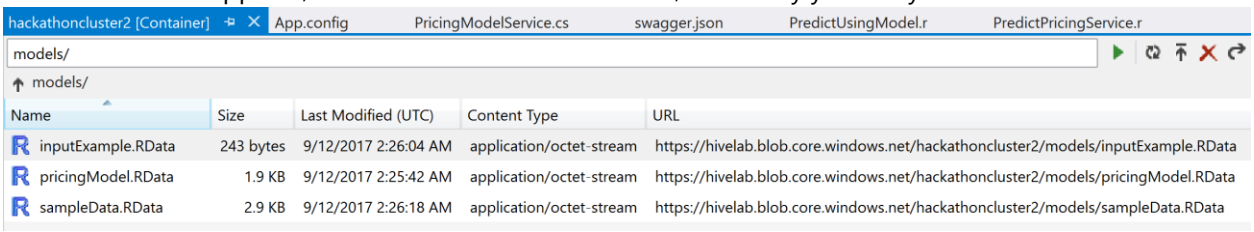
```
> rxHadoopListFiles("/models")
[1] "Found 3 items"
[2] "-rw-r--r--   1 sshuser supergroup        243 2017-09-12 02:26 /models/inputExample.RData"
[3] "-rw-r--r--   1 sshuser supergroup       1895 2017-09-12 02:25 /models/pricingModel.RData"
[4] "-rw-r--r--   1 sshuser supergroup       2954 2017-09-12 02:26 /models/sampleData.RData"
[1] TRUE
```

17. Using Visual Studio, Cloud Explorer, navigate to the storage account for your HDInsight cluster, expand:



18. Right-click the storage container, and select **Open.**

19. In the editor that appears, double-click the models folder, and verify you see your files.



20. Right-click **inputExample.RData** and select Save As… and choose the directory for your **PricingModel** project, overwriting files if prompted.

21. Repeat the previous step for **pricingModel.RData** and **sampleData.RData**.

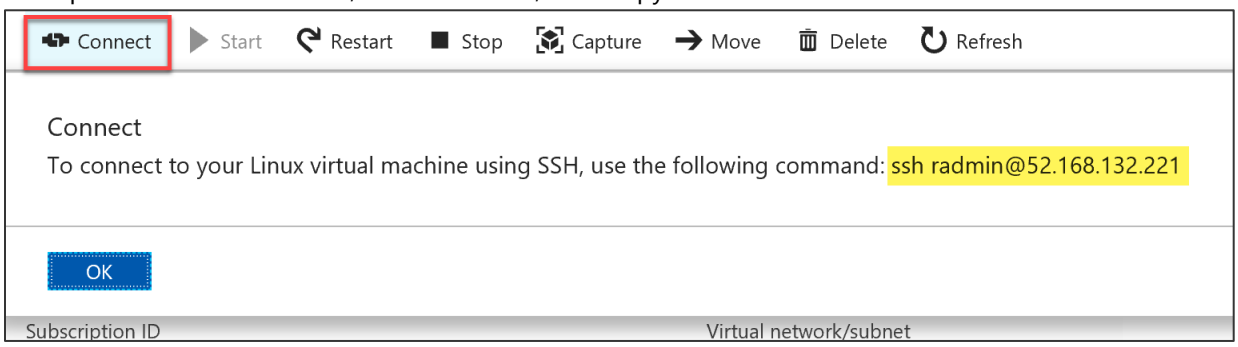
22. You have now used R Server on HDInsight to train a model that you can then upload to R Server Operationalization to expose it as a web service.

Task 4: Create predictive service in R Server Operationalization

After training a model, you want to operationalize the model so that it becomes available for integration by developers. One way to operationalize a trained model is to take the model you trained in HDInsight, and then to expose that as a predictive web service. In this task, you take a version of the scripts you have been running locally and in HDInsight and migrate them to run in the VM that is running R.

16. In the Azure portal, navigate to the Microsoft Machine Learning Server Virtual Machine you created in Exercise 1, Task 3.

17. On top of the Overview blade, select Connect, then copy the SSH command.



18. Using a new Git Bash window on your Lab VM, SSH into your Microsoft Machine Learning Server VM by pasting the SSH command you copied above at the command prompt. For example:
ssh radmin@<your-server-ip>.

19. When prompted if you want to continue connecting, enter yes.

20. Enter your password, **Password.1!!**

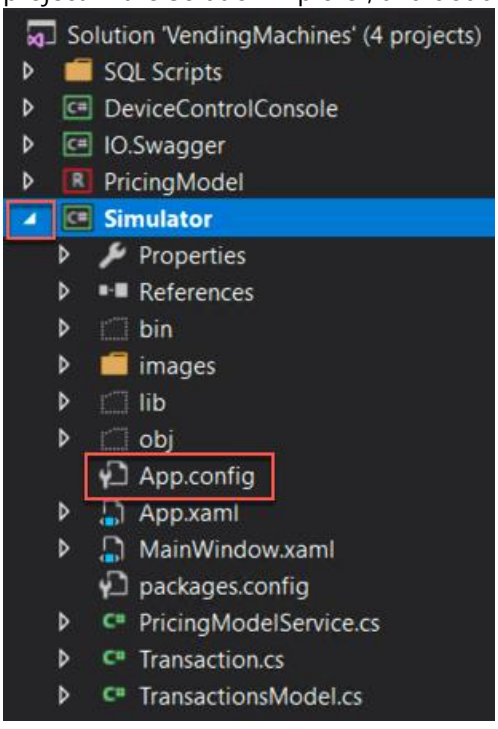
21. At the prompt, after successfully logging in, you will need to complete a few tasks to configure and operationalize the environment.

22. Run the following command to act as root: **sudo -i**

23. Now that you are acting as root run the following command to: **az ml admin node setup --onebox --admin-password Password.1!! --confirm-password Password.1!!**

```
root@ivmMLserver:~# az ml admin node setup --onebox --admin-password Password.1!! --co
az ml admin node setup --onebox --admin-password Password.1clear --confirm-password Pa



Starting Compute Node
SUCCESS! Compute Node started. (PID: 25394, Listening on URI: http://localhost:12805/)
Starting Web Node
SUCCESS! Web Node started. (PID: 25657, Listening on URI: http://localhost:12800/)

root@ivmMLserver:~# exit
logout
radmin@ivmMLserver:~$
```

24. In Visual Studio, open the **App.config** for the Simulator project. This can be done by expanding the Simulator project in the Solution Explorer, and double-clicking on **App.config** under the **Simulator** project.



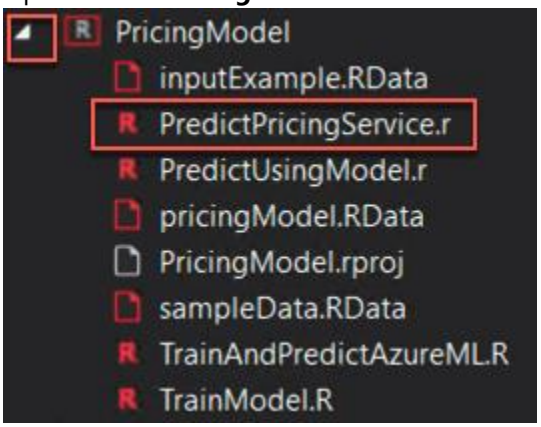
25. Locate the **appSetting rServiceBaseAddress** and enter **http://<your-server-public-ip>:12800** for the value. (For example: http://52.168.132.221:12800). Your server IP address is the same IP address you used for the SSH connection in Step 3 of this task, above.

26. Locate the **appSetting rServicePassword** and update its value with the password you defined when you configured your R server for Operationalization (step 11 above), **Password.1!!**

```
<appSettings>
    <add key="rServiceBaseAddress" value="http://52.250.124.140:12800" />
    <add key="rServiceUsername" value="admin" />
    <add key="rServicePassword" value="Password.1!!" />

    <add key="IoTHubSenderConnectionsString" value="" />
    <add key="IoTHubManagerConnectionsString" value="" />

    <add key="storageConnectionString" value="" />

    <add key="faceAPIKey" value="" />
    <add key="faceEndpoint" value="" />

</appSettings>
```

27. Save **App.config**.

28. Open **PredictPricingService.r** within the **PricingModel** project in the Visual Studio Solution Explorer.



29. Find TODO 1, and replace with the following code block:

```
# TODO: 1. Load packages needed for operationalization

usePackage <- function(p) {

if (!is.element(p, installed.packages()[, 1]))

install.packages(p, dep = TRUE)

library(p, character.only = TRUE)

}
```

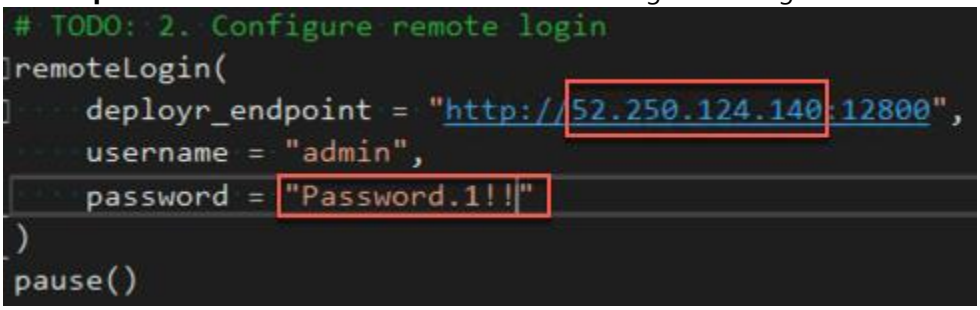


```
usePackage("curl")
usePackage("ggplot2")
usePackage("mrsdeploy")
usePackage("RevoScaleR")
```

30. Find TODO 2, and replace with the following code block to remotely connect to the R Server Operationalization service:

```
# TODO: 2. Configure remote login
remoteLogin(
deployr_endpoint = "http://<your-server-ip>:12800",
username = "admin",
password = "<your-admin-password>"
)
pause()
```

Note: Make sure to replace **<your-server-ip>** with the IP address with your VM's IP address, and enter the password you specified when you configured your R server for Operationalization (step 11) in place of **<your-admin-password>**. The TODO 2 section should looking something like the following screen shot.



31. Highlight all the code in **PredictPricingService.r**, right-click and then **execute in interactive**. The last output status in the R Interactive window should be "Published service".

32. Find TODO 3 and replace with the following code block to consume the API as a test:

```
# TODO: 3. Consume the API as a test
services <- listServices("apiPredictPurchasePrice")
serviceName <- services[[1]]
api <- getService(serviceName$name, serviceName$version)
result <- api$apiPredictPurchasePrice(30, "F", "coconut water")
print("Result: ")
print(result$output("answer"))
```

```
result
```

33. Find TODO 4 and replace with the following code block to generate and save the Swagger JSON file for the API:

```
# TODO: 4. Generate the Swagger JSON file for the API

swagger <- api$swagger()

cat(swagger, file = "swagger.json", append = FALSE)
```

34. Highlight just the TODO 3 and TODO 4 code blocks you added and execute in interactive.

35. When you scroll up through the R Interactive window results, you should see an output with your prediction like the following:

```
$success
[1] TRUE

$errorMessage
[1] ""

$outputParameters
$outputParameters$purchasePrice
[1] 0.9348741
```

36. Also open the swagger.json file in your PricingModel project directory to view its contents. This file can be used within the swagger.io online editor to generate client code to connect to your service. We have already done this for you within the included IO.Swagger project.

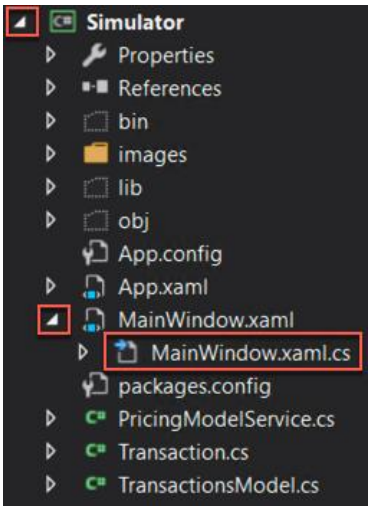
Exercise 3: Implement dynamic pricing

Duration: 45 minutes

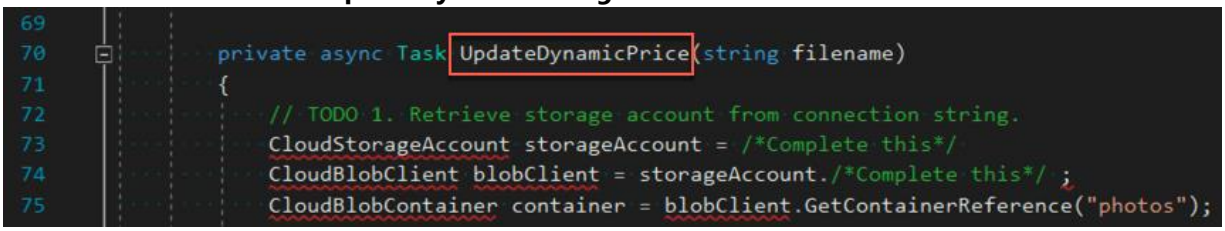
In this exercise, you will implement the code that performs dynamic pricing, capitalizing on the Face API to acquire demographics, and your deployed pricing model to suggest the price based on those demographics. You will then run the vending machine simulator and see the dynamic pricing in action.

Task 1: Implement photo uploads to Azure Storage

1. In Visual Studio Solution Explorer, expand the Simulator project and then **MainWindow.xaml** and then open **MainWindow.xaml.cs**.



2. Scroll down to the method **UpdateDynamicPricing**.



3. Replace TODO 1 with the following:

```
// TODO 1. Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(_storageConnectionString);
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobClient.GetContainerReference("photos");
```

4. Replace TODO 2 with the following:

```
// TODO 2. Retrieve reference to a blob named with the value of fileName.
string blobName = Guid.NewGuid().ToString() + System.IO.Path.GetExtension(filename);
CloudBlockBlob blockBlob = container.GetBlockBlobReference(blobName);
```

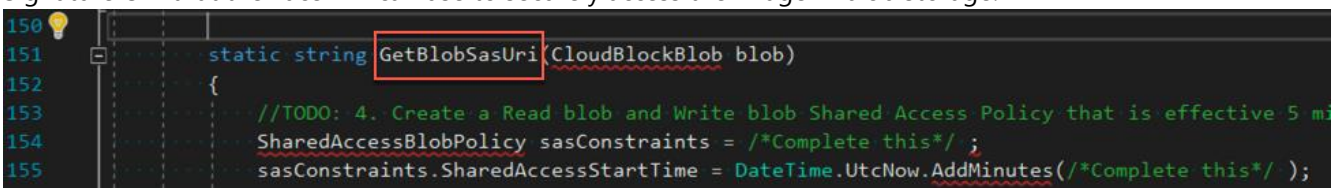
5. Replace TODO 3 with the following:

```
// TODO 3. Create or overwrite the blob with contents from a local file.

using (var fileStream = System.IO.File.OpenRead(filename))

{

       blockBlob.UploadFromStream(fileStream);

}
```

6. Save MainWindow.xaml.cs.

Task 2: Invoke Face API

1. Continuing with **MainWindow.xaml.cs**, scroll down to **GetBlobSasUri**. This method will create a Shared Access Signature URI that the Face API can use to securely access the image in blob storage.



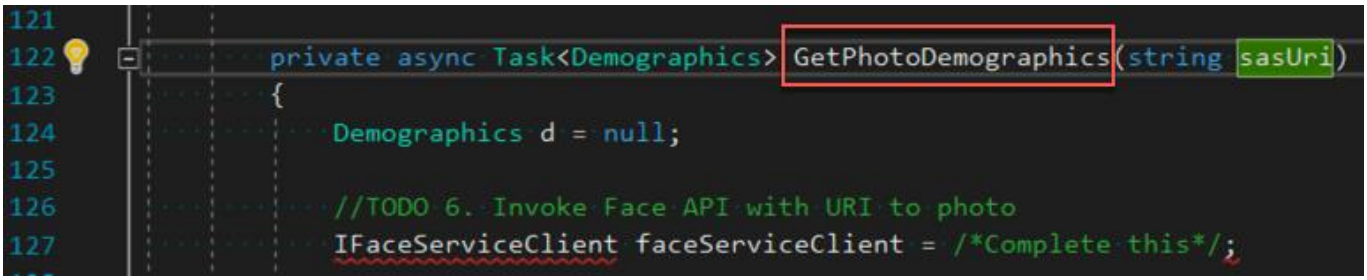
2. Replace TODO 4 with the following:

```
//TODO: 4. Create a Read blob and Write blob Shared Access Policy that is effective 5
minutes ago and for 2 hours into the future

SharedAccessBlobPolicy sasConstraints = new SharedAccessBlobPolicy();

sasConstraints.SharedAccessStartTime = DateTime.UtcNow.AddMinutes(-5);

sasConstraints.SharedAccessExpiryTime = DateTime.UtcNow.AddHours(2);

sasConstraints.Permissions = SharedAccessBlobPermissions.Read |
SharedAccessBlobPermissions.Write;
```

3. Replace TODO 5 with the following:

```
//TODO: 5. construct the full URI with SAS

string sasBlobToken = blob.GetSharedAccessSignature(sasConstraints);

return blob.Uri + sasBlobToken;
```

4. With the SAS URI to the upload photo in hand, scroll to **GetPhotoDemographics** to implement the call to the Face API.



5. Replace TODO 6 with the following:

```
//TODO 6. Invoke Face API with URI to photo

IFaceServiceClient faceServiceClient = new FaceServiceClient(_faceApiKey, _faceEndpoint);
```

6. Replace TODO 7 with the following:

```
//TODO 7. Configure the desired attributes Age and Gender

IEnumerable<FaceAttributeType> desiredAttributes = new FaceAttributeType[] {
FaceAttributeType.Age, FaceAttributeType.Gender };
```

7. Replace TODO 8 with the following:

```
//TODO 8. Invoke the Face API Detect operation

Face[] faces = await faceServiceClient.DetectAsync(sasUri, false, true, desiredAttributes);
```

8. Replace TODO 9 with the following:

```
//TODO 9. Extract the age and gender from the Face API response

double computedAge = faces[0].FaceAttributes.Age;

string computedGender = faces[0].FaceAttributes.Gender;
```

9. Save the file.

Task 3: Invoke pricing model

1. Within **MainWindow.xaml.cs**, scroll to the end of **UpdateDynamicPrice** and replace TODO 10 with the following:

```
//TODO 10. Invoke the actual ML Model

PricingModelService pricingModel = new PricingModelService();

string gender = d.gender == "Female" ? "F" : "M";

suggestedPrice = await pricingModel.GetSuggestedPrice((int)d.age, gender, _itemName);
```

2. Save the file.

Task 4: Configure the Simulator

1. In the Simulator project, open **App.config**.

2. Within the **appSettings** section, set the following settings (there were copied into a text edit previously):
   a. **faceAPIKey**: set this to the KEY 1 value for your Face API as acquired from the Azure Portal.
   b. **faceEndpoint**: set this to the ENDPOINT value for your Face API as acquired from the Azure Portal (for example: https://eastus2.api.cognitive.microsoft.com/face/v1.0)
   c. **storageConnectionString**: set this to the connection string of the Storage Account you created with the photos container.
3. Save the **App.config**. The updated **App.config** file settings should look similar to the following:

```
20
21      <add key="storageConnectionString" value="DefaultEndpointsProtocol=https;AccountName=vendingmachinesphotos;
22
23      <add key="faceAPIKey" value="beead523483b4bfb9258abd9fda6db9b" />
24      <add key="faceEndpoint" value="https://westus2.api.cognitive.microsoft.com/face/v1.0" />
25
26  </appSettings>
```

Task 5: Test dynamic pricing in Simulator

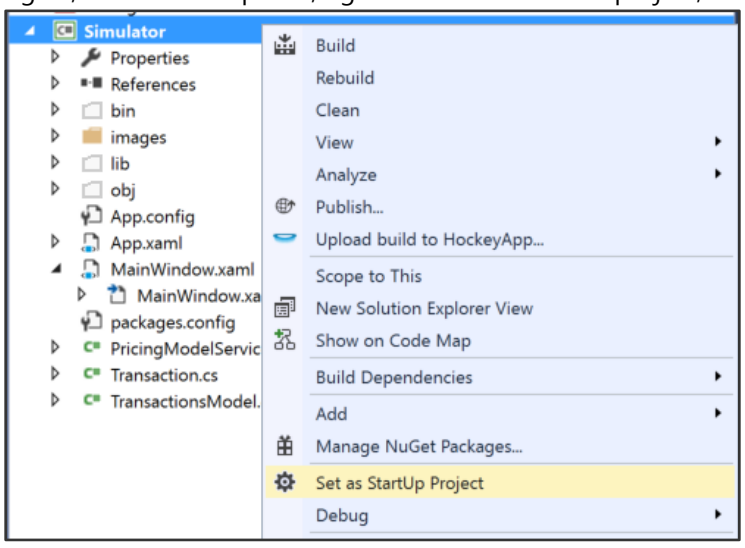
1. Before building the project in Visual Studio, we need to ensure all the NuGet packages are properly restored. In Visual Studio, go to **Tools->NuGet Package Manager->Package Manager Console** and enter the following command:

   ```
   Update-Package -Reinstall
   ```

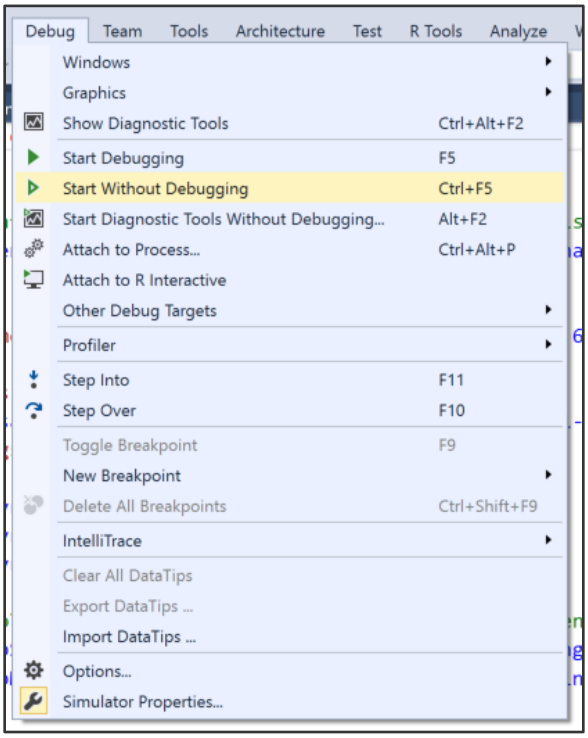
2. That should force the packages to get downloaded again.

3. Now, in Solution Explorer, right-click the Simulator project, and select **Build**.

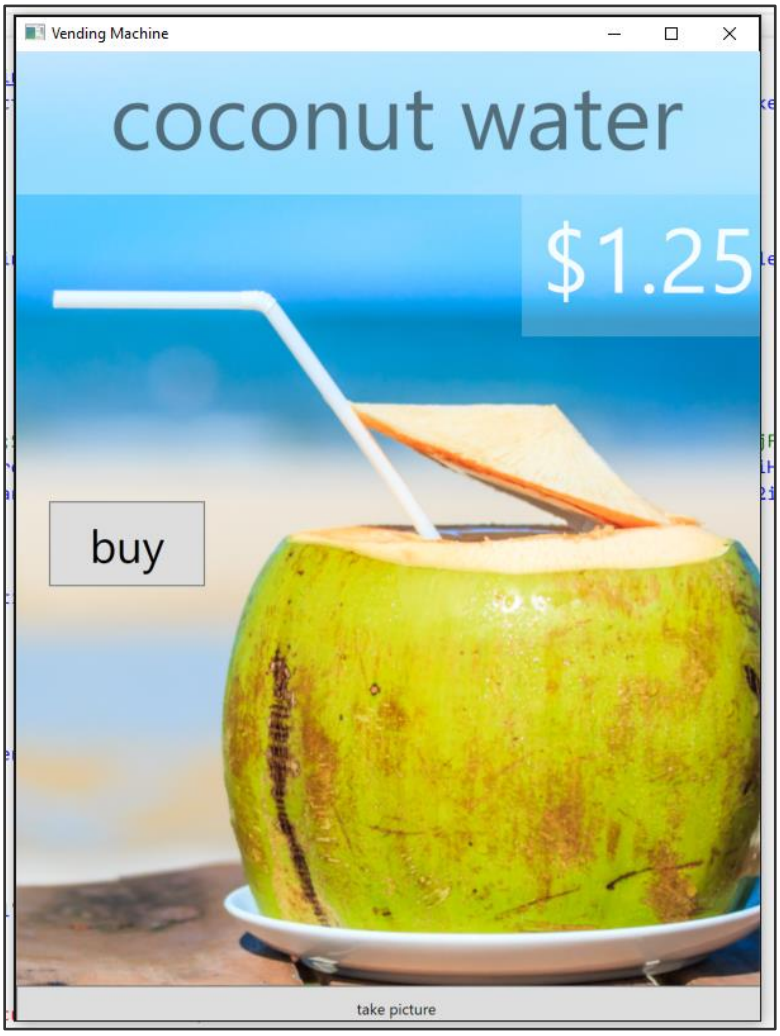
4. Ensure that your build generates no errors (View the Output and Errors windows, under the View menu in Visual Studio).
5. Again, in solution explorer, right-click the Simulator project, and select Set as Startup Project.



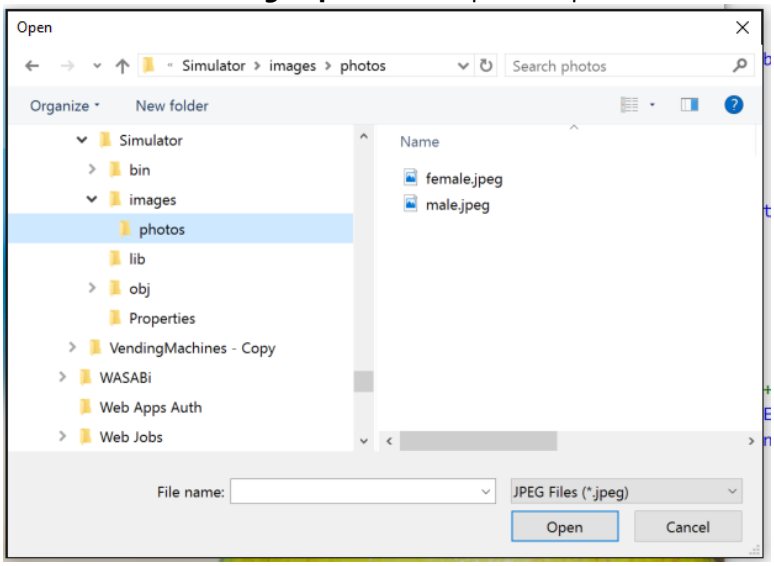
6. From the Debug menu, select Start Without Debugging.



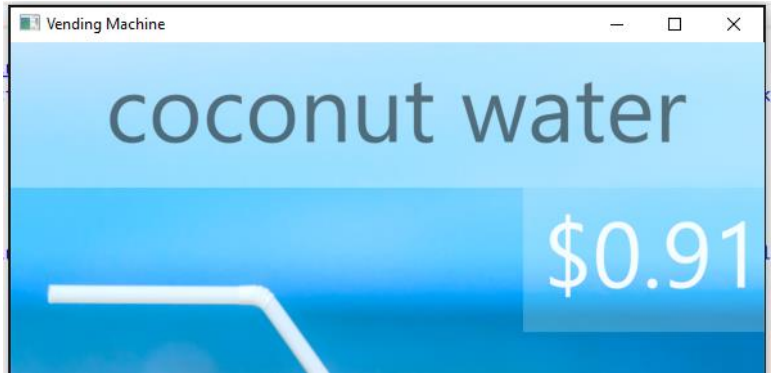
7. When the vending machine simulator appears, **select take picture** at the bottom.



8. In the dialog that appears, navigate to the images folder under **C:\VendingMachines\VendingMachines - Clean\Simulator\images\photos**, and pick the photo of either the man or woman to upload, and select Open.



9. In a few moments, you should see the price change from $1.25 to whatever value the predictive model suggested.



10. Try using the other photo or your own photo to see what prices are suggested.

11. Click the X at the top right of the application to stop it.

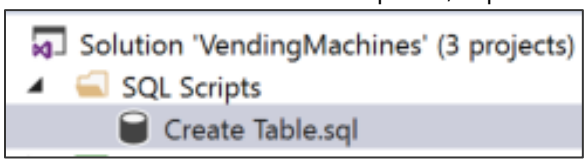
Exercise 4: Implement purchasing

Duration: 15 minutes

In this exercise, you will create an in-memory, columnar index table in SQL DB that will be used to support purchase transactions in a real-time analytics fashion, and then implement the purchasing process in the vending machine simulator. Finally, you will run the simulator and purchase items.

Task 1: Create the transactions table

1. Within Visual Studio Solution Explorer, expand the SQL Scripts folder and open the file Create Table.sql.



2. Replace TODO 1 with the following:

```
TODO: 1. Transaction ID should be a Primary Key, fields with a b-tree index

TransactionId int IDENTITY NOT NULL PRIMARY KEY NONCLUSTERED,
```

3. Replace TODO 2 with the following:

```
TODO: 2. This table should have a columnar index

INDEX Transactions_CCI CLUSTERED COLUMNSTORE
```

4. Replace TODO 3 with the following:

```
TODO: 3. This should be an in-memory table

MEMORY_OPTIMIZED = ON
```

5. Replace TODO 4 with the following:

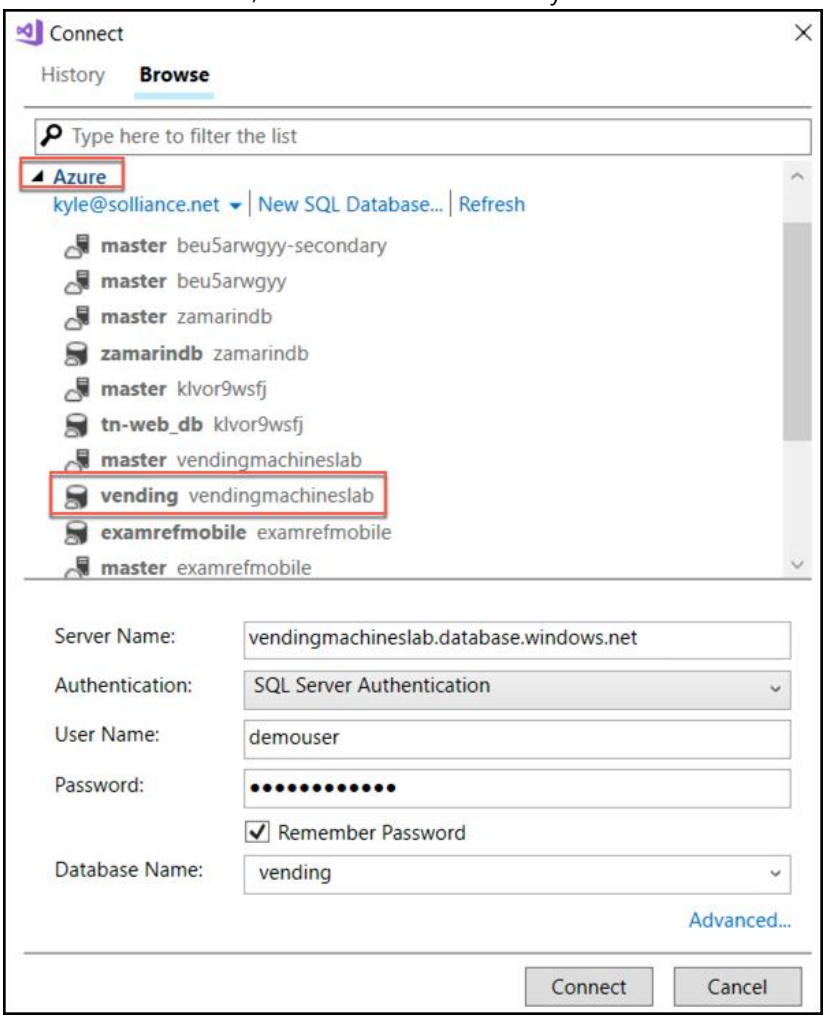
```
TODO: 4. In-memory tables should auto-elevate their transaction level to Snapshot

ALTER DATABASE CURRENT SET MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT=ON;
```

6. Save the script.

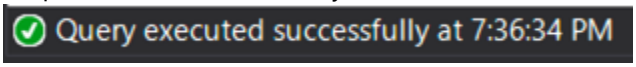
7. Execute the script by pressing the play icon.

8. Expand **Azure**, and if prompted, sign in with your Azure credentials.

9. From the Azure node, and select the database you created for the vending database.



10. In the in fields at the bottom, enter your user name and password for the SQL Server, and select Connect. The script should run successfully.



Task 2: Configure the Simulator

1. In the Simulator project, open **App.config**.

2. Within the **connectionString** section, set the following:
   - **a.** TransactionsModel: set the value of the connectionString attribute to the ADO.NET connection string to your SQL DB instance. This value was copied to a text editor previously, or you can copy it from the Azure Portal. **Do not forget to replace the values for {your_username} and {your_password} with your actual credentials.**
     i. User name: demouser
     ii. Password: Password.1!!

3. Save the **App.config**.

```
<connectionStrings>
    <add name="TransactionsModel" connectionString="Server=tcp:vendingmachineslab.database.windows.net,1433;Init
</connectionStrings>
```

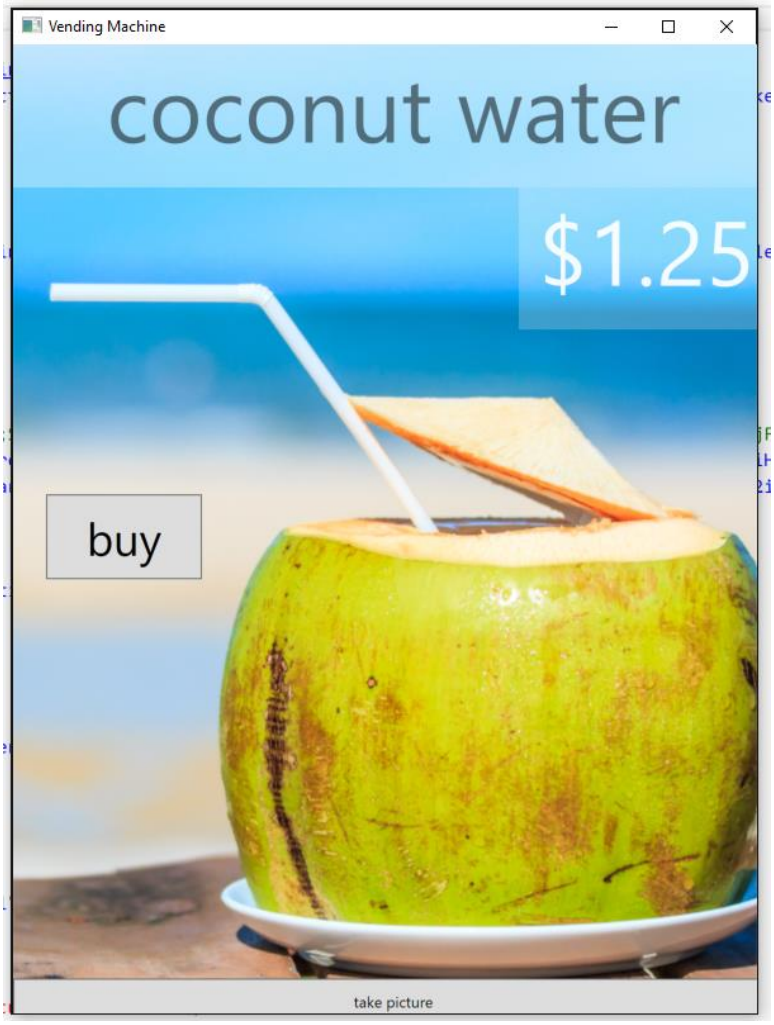
Task 3: Test purchasing

1. In solution explorer, right-click the Simulator project, and select Build. Note: You may need to ensure the previous instance you started has been closed before rebuilding.

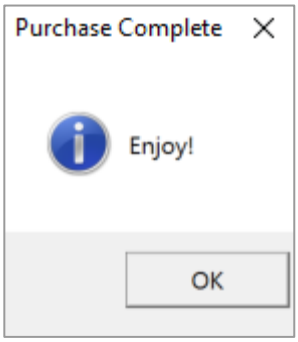
2. Ensure that your build generates no errors.

3. From the Debug menu, select **Start Without Debugging**.

4. In the Simulator, select buy.



5. You should see a confirmation dialog similar to the following:



Exercise 5: Implement device command and control

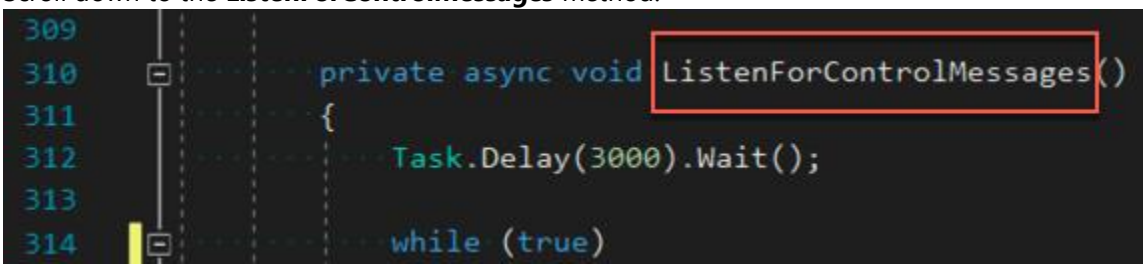
Duration: 30 minutes

In this exercise, you will implement the ability to push new promotions to the vending machine simulator using the command and control features of IoT Hub. You will update the simulator to listen for these messages. You will also update the console application DeviceControlConsole to send selected promotions.

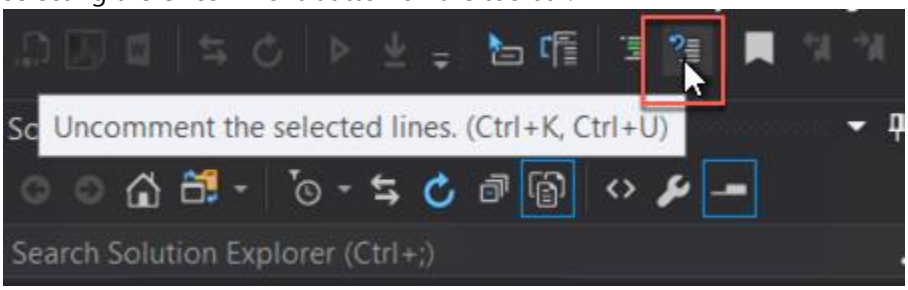
Task 1: Listen for control messages

1. Within Visual Studio Solution Explorer, expand the Simulator project, and open the file **MainWindow.xaml.cs**.

2. Scroll down to the **ListenForControlMessages** method.



3. Uncomment the body of the while(true) loop. You can uncomment a block of code by selecting the code, then selecting the Uncomment button on the toolbar.



4. Replace TODO 1 with the following:

```
//TODO: 1. Receive messages intended for the device via the instance of _deviceClient.

Microsoft.Azure.Devices.Client.Message receivedMessage = await
_deviceClient.ReceiveAsync();
```

5. Replace TODO 2 with the following:

```
//TODO: 2. A null message may be received if the wait period expired, so ignore and call
the receive operation again

if (receivedMessage == null) continue;
```

6. Replace TODO 3 with the following:

```
//TODO: 3. Deserialize the received binary encoded JSON message into an instance of
PromoPackage.
```



```
string receivedJSON = Encoding.ASCII.GetString(receivedMessage.GetBytes());

System.Diagnostics.Trace.TraceInformation("Received message: {0}", receivedJSON);

PromoPackage promo =
Newtonsoft.Json.JsonConvert.DeserializeObject<PromoPackage>(receivedJSON);
```

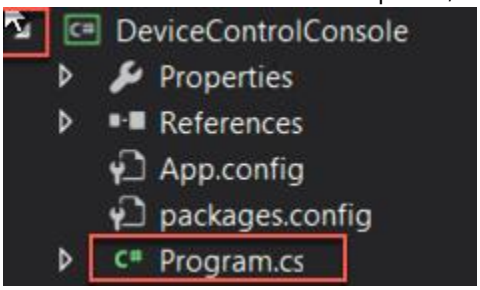
7. Replace TODO 4 with the following:

```
//TODO: 4. Acknowledge receipt of the message with IoT Hub

await _deviceClient.CompleteAsync(receivedMessage);
```

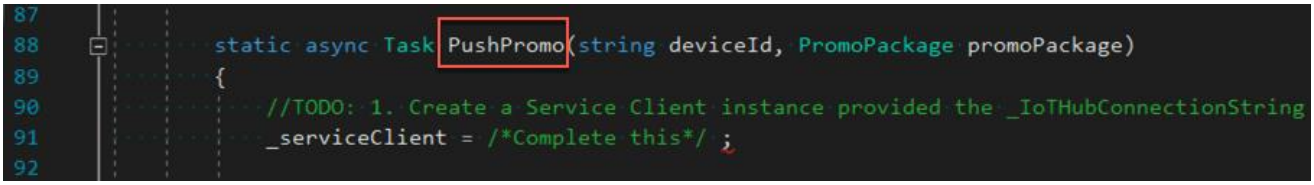
8. Save the file.

Task 2: Send control messages

1. Within Visual Studio Solution Explorer, expand the **DeviceControlConsole** project, and open the file **Program.cs**.



2. Scroll down to the PushPromo method.



3. Replace TODO 1 with the following:

```
//TODO: 1. Create a Service Client instance provided the _IoTHubConnectionString

_serviceClient = ServiceClient.CreateFromConnectionString(_IoTHubConnectionString);
```

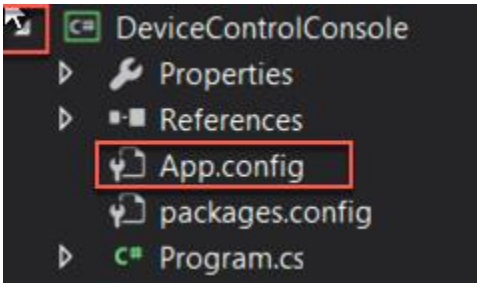
4. Replace TODO 2 with the following:

```
//TODO: 2. Send the command

await _serviceClient.SendAsync(deviceId, commandMessage);
```

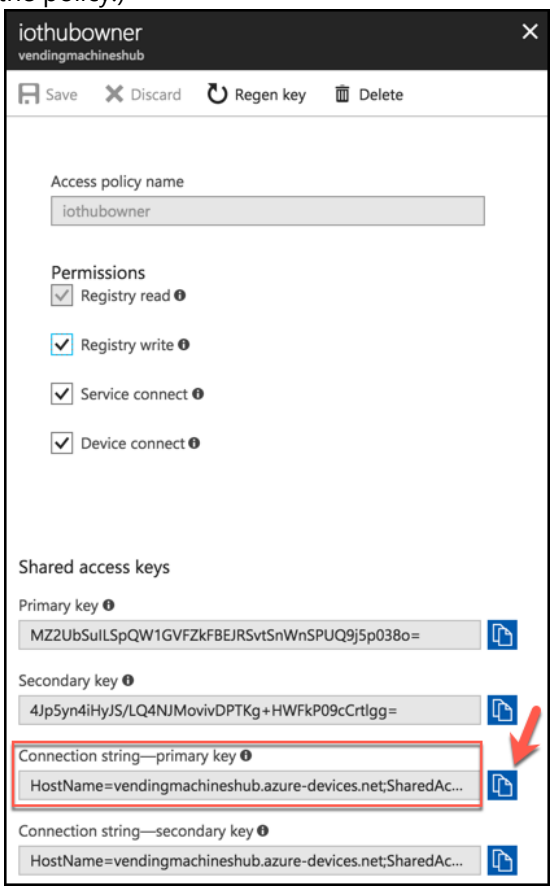
5. Save Program.cs.

Task 3: Configure the DeviceControlConsole and the Simulator

1. In **DeviceControlConsole**, open **App.config**.



2. Set the **IoTHubConnectionString appSetting** to have a value of the connection string for the service policy to your IoT Hub. (Recall you can get this from the Azure Portal IoT Hub blade, Shared access policies, and then select the policy.)



3. Set the **storageConnectionString appSetting** to have the same connection string for your storage account that the **App.config** file in the Simulator project has.

4. Save the file.

5. Now, open the **App.config** file In the Simulator project.

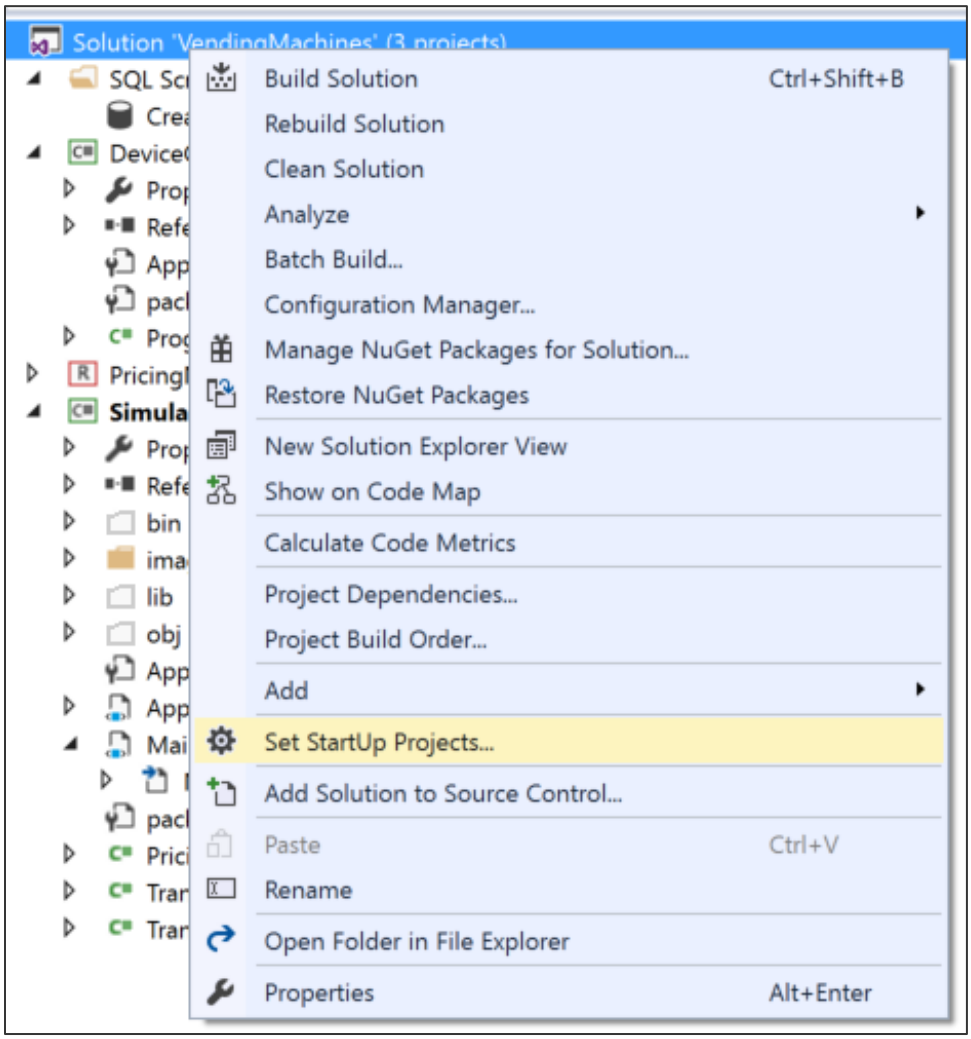
6. Set the **IoTHubSenderConnectionString appSetting** to have a value of the connection string for the device policy to your IoT Hub.

7. Set the **IoTHubManagerConnectionString appSetting** to have a value of the connection string for the **iothubowner** policy to your IoT Hub.

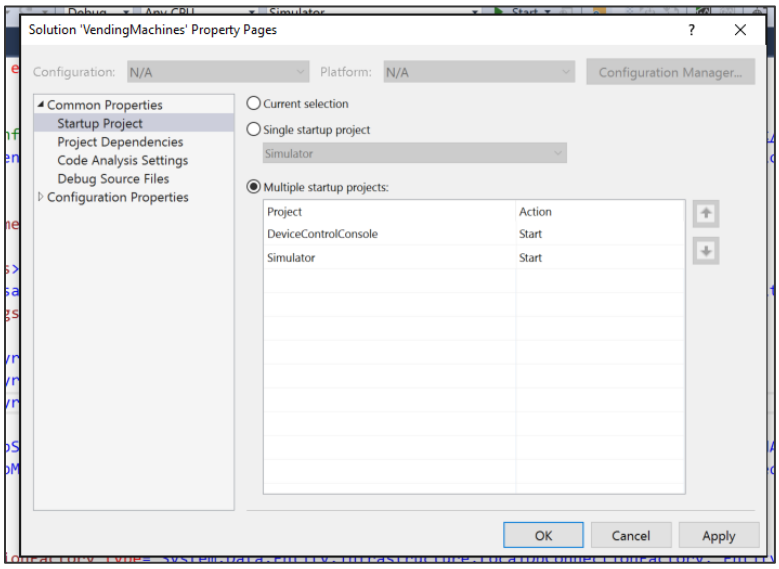
8. Save the file.

9. Build the Simulator and DeviceControlConsole projects.

10. In Solution Explorer, right-click Solution 'Vending Machines,' and select Set StartUp Projects.



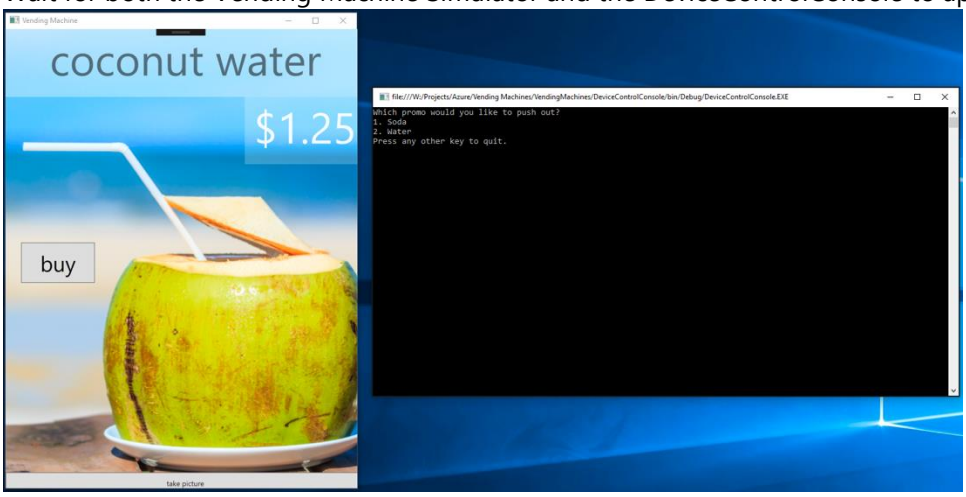
11. In the dialog, select the Multiple startup projects option, and ensure that Action is set to Start for both DeviceControlConsole and Simulator.



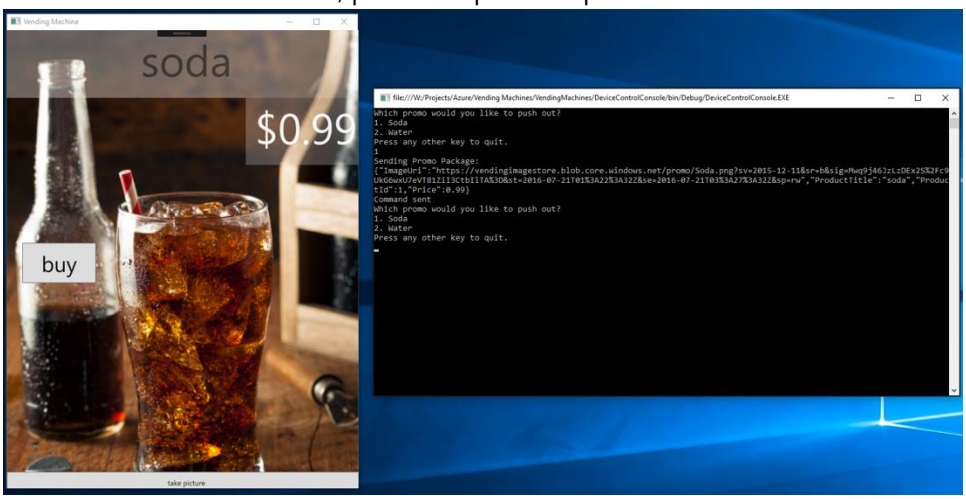
12. Select OK.

13. From the Debug menu, choose Start without Debugging.

14. Wait for both the Vending Machine Simulator and the DeviceControlConsole to appear.



15. In the DeviceControlConsole, press 1 to push the promotion for Soda.



16. Observe that the entire promotion surface of the vending machine changes (product name, price, and image). Note: If the photo does not change, and after a few minutes you receive a DotNetty.Transport... error, you will need to delete and recreate your IoT Hub in the Azure portal. The error is caused by a communication error between the application and your IoT Hub. Be sure to update your App.config file with the new IoT hub connection strings.

17. Experiment sending the other promotion or toggling between promotions.

18. Experiment with making purchases and sending photos to verify the other functions still work with the new promoted products.

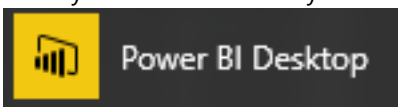
Exercise 6: Analytics with Power BI Desktop

Duration: 15 minutes

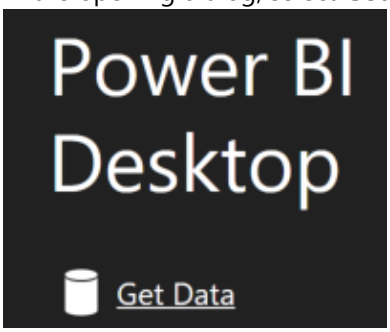
In this exercise, you will use Power BI Desktop to query purchase data from the in-memory table of SQL DB and visualize the result.

Task 1: Build the query and create the visualization

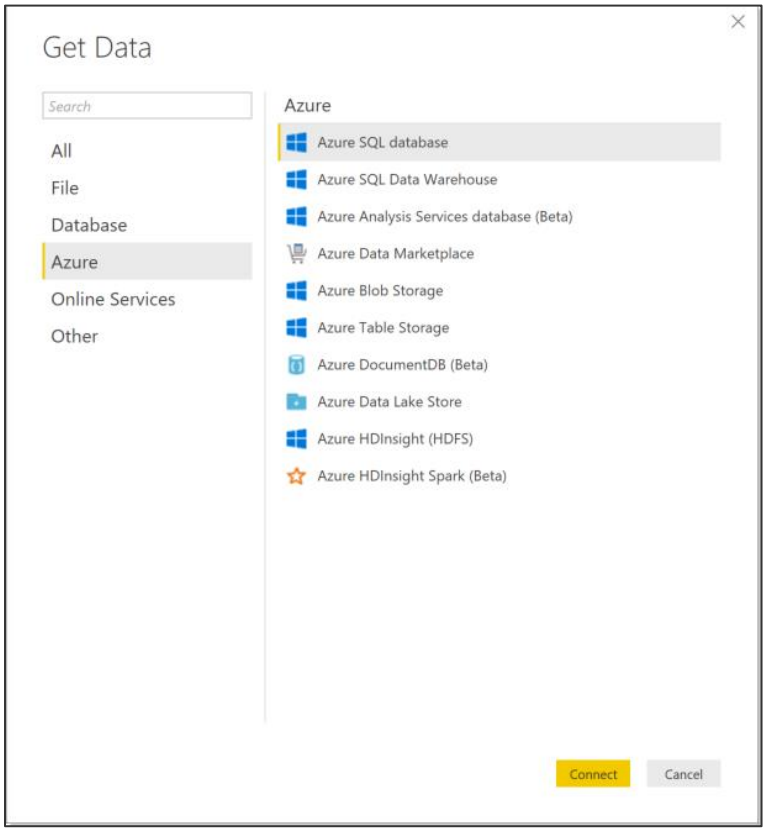
1. From your Start menu on your Lab VM, open Power BI Desktop.



2. In the opening dialog, select **Get Data**.

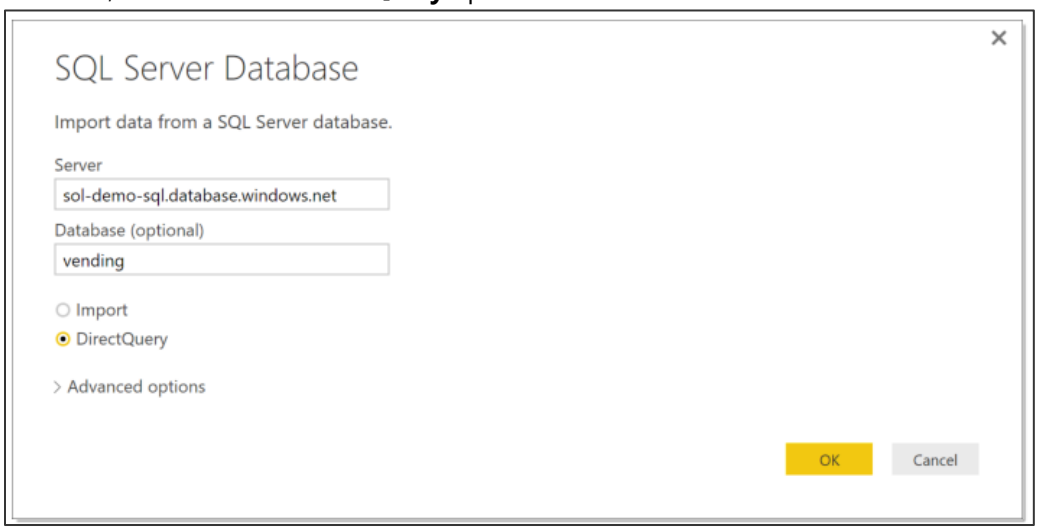


3. In the Get Data dialog, select **Azure** in the categories list and then **Azure SQL Database**.

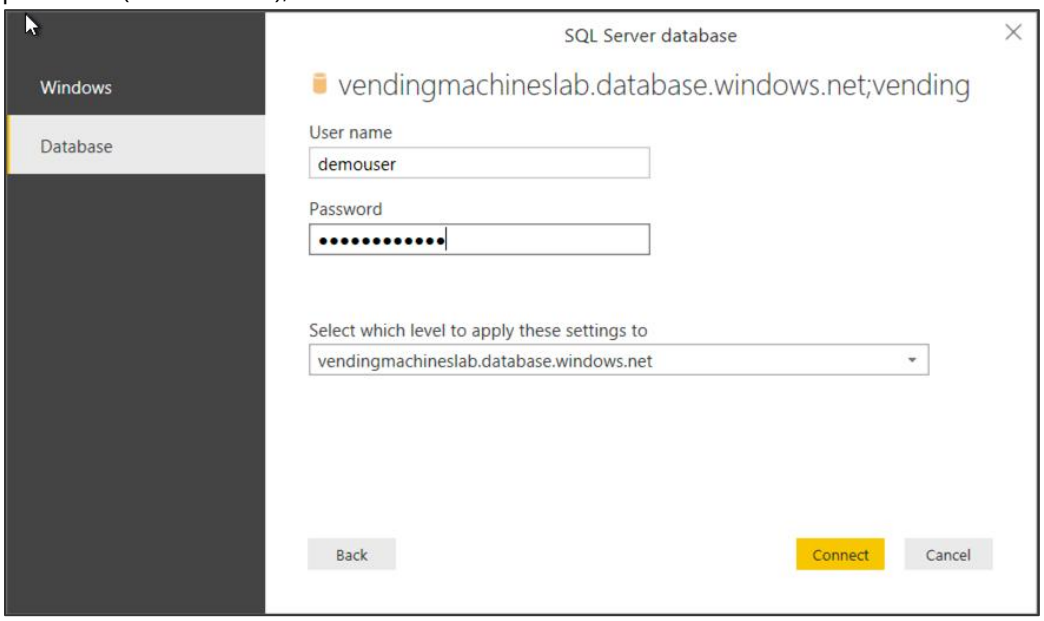


4. Select Connect.

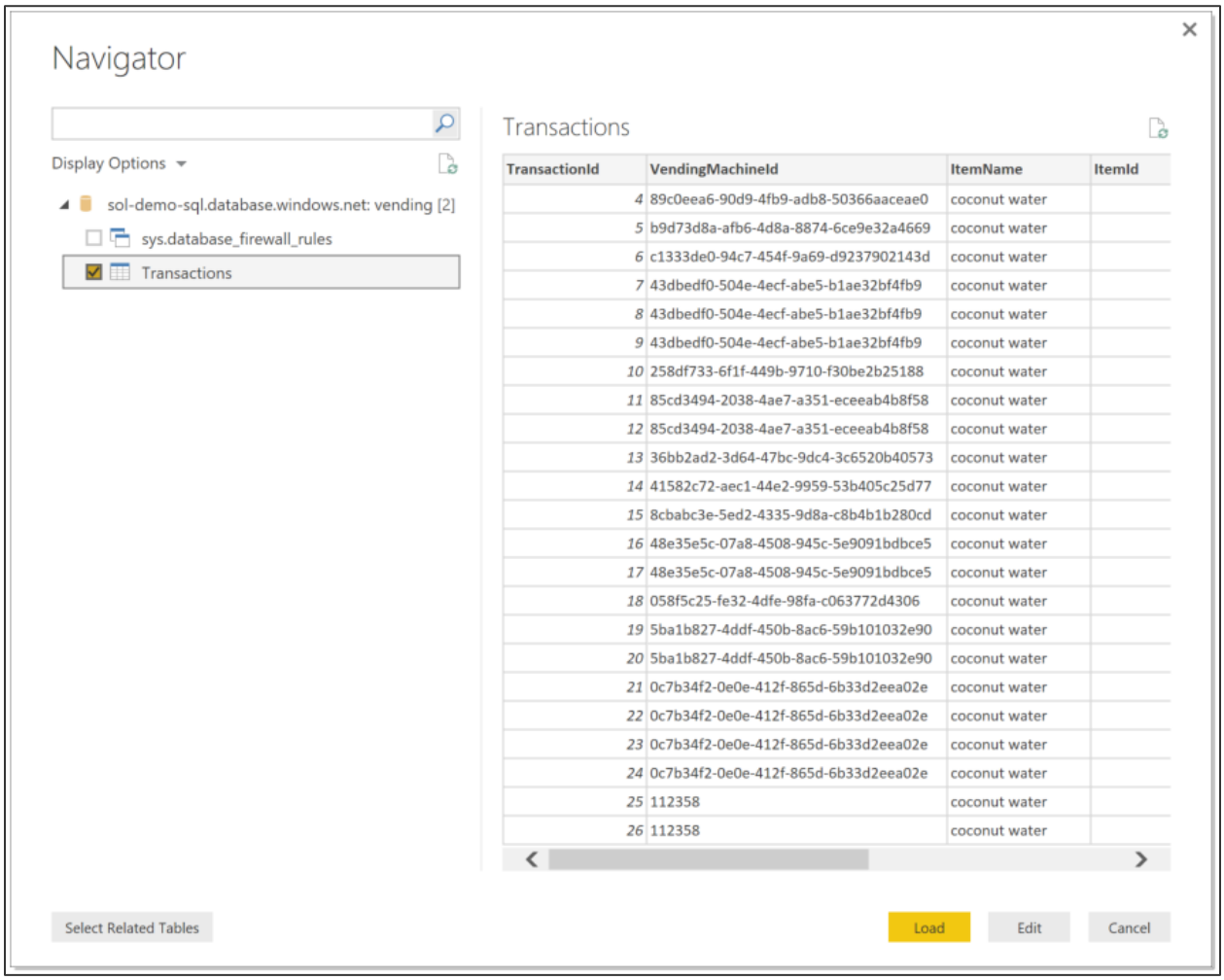
5. In the dialog, enter the name of your SQL Server (e.g., myserver.database.windows.net), the name of your vending database, and select the **DirectQuery** option. Select OK.



6. On the next screen, select the **Database** tab on the left, and provide your SQL username (demouser) and password (Password.1!!), then select **Connect**.

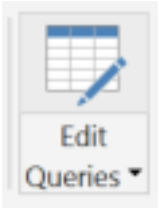


7. In the Navigator dialog, check the box next to **Transactions**.

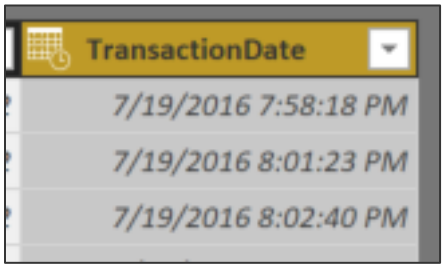


8. Select **Load**.

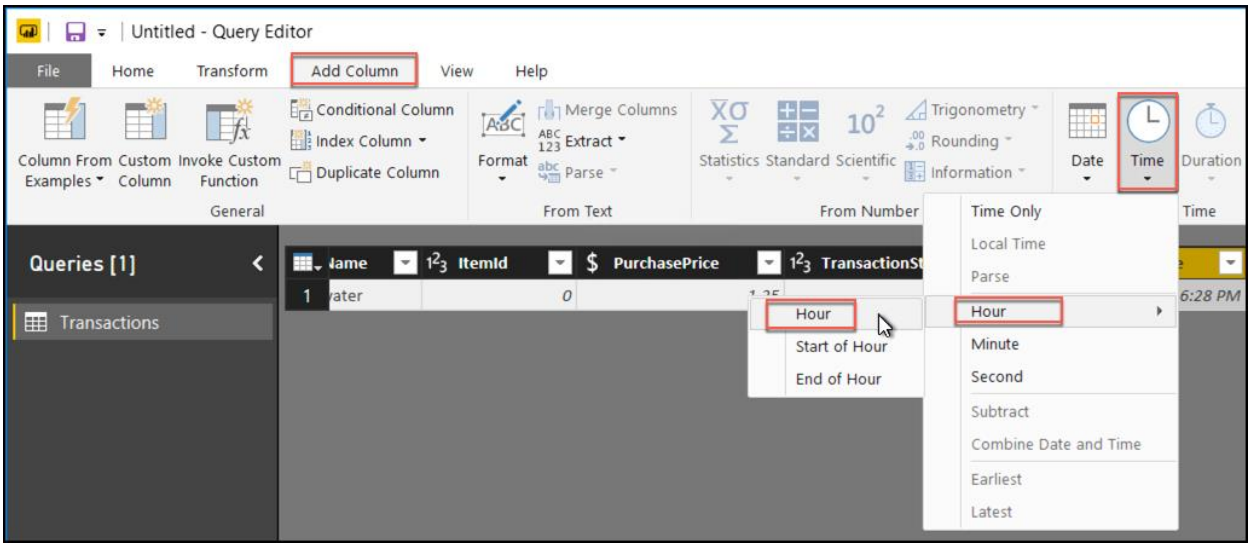
9. In the Ribbon, select **Edit Queries**.



10. In the Query Editor, select the Transaction Date column header to select the column.

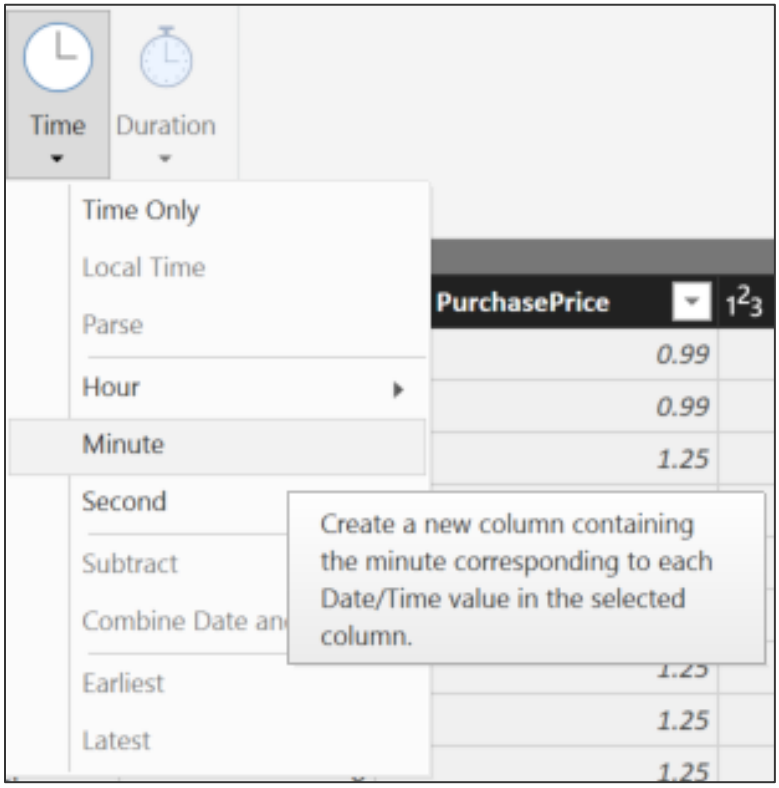


11. In the Ribbon, select the **Add Column** tab and select Time, Hour, Hour.



12. Select the **TransactionDate** column again.

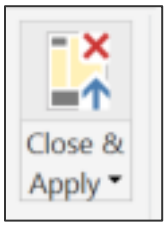
13. In the Ribbon, select **Time, Minute.**



14. Select the **TransactionDate** one more time.

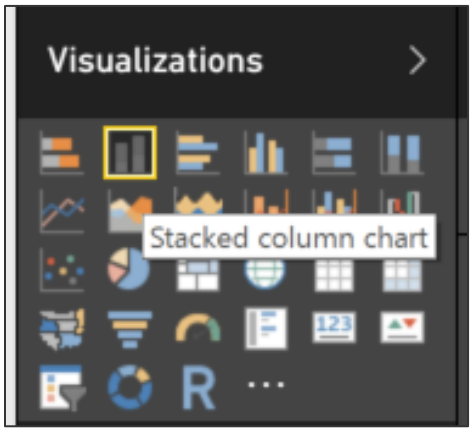
15. In the Ribbon, select **Time, Second**.

16. In the Ribbon, on the Home tab, select **Close & Apply**.

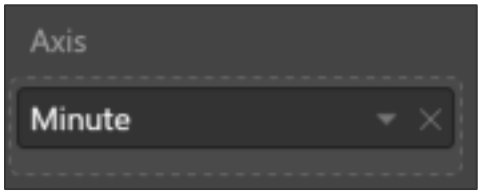


17. In the message that appears, select **Apply Changes**.

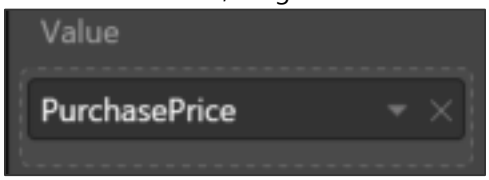
18. In the Visualizations, select **Stacked column chart**.



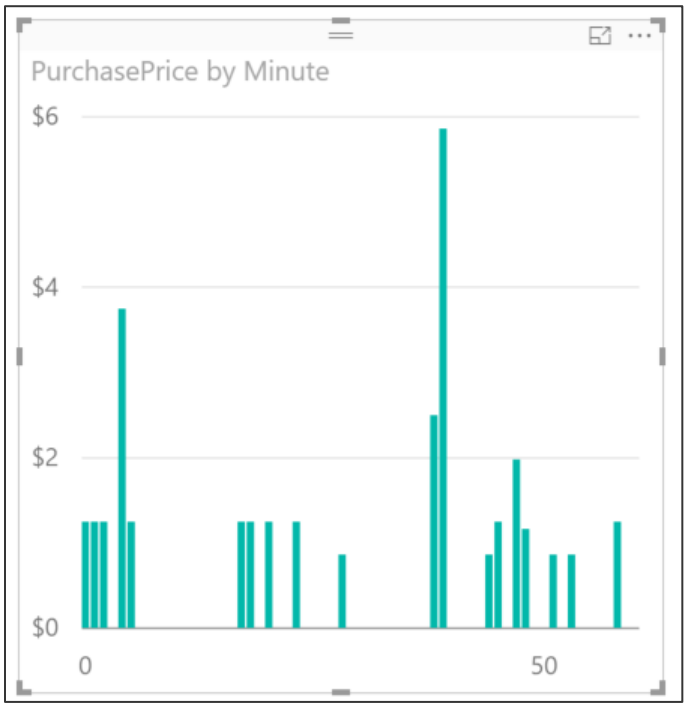
19. From the Fields list, drag the **minute** field over to the **axis** property.



20. From the Fields list, drag the **PurchasePrice** over to the **value** property.



21. Your completed visualization summarizing the most profitable minutes in each hour should appear as follows:



After the hands-on lab

Duration: 10 mins

In this exercise, attendees will deprovision any Azure resources that were created in support of the lab. You should follow all steps provided after attending the Hands-on lab.

Task 1: Delete the resource group

1. Using the Azure portal, navigate to the Resource group you used throughout this hands-on lab by selecting Resource groups in the left menu.
2. Search for the name of your research group, and select it from the list.
3. Select Delete in the command bar, and confirm the deletion by re-typing the Resource group name, and selecting Delete.