Microsoft Cloud Workshop

Intelligent Vending Machines Hackathon Leader guide

September 2017

© 2017 Microsoft Corporation. All rights reserved. This document is confidential and proprietary to Microsoft. Internal use only. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

Contents

[Intelligent Vending Machines 1](#_Toc493690285)

[Overview 1](#_Toc493690286)

[Requirements 1](#_Toc493690287)

[Exercise 0: Before the hackathon 2](#_Toc493690288)

[Task 1: Provision Power BI 2](#_Toc493690289)

[Task 2: Provision Power BI Desktop 2](#_Toc493690290)

[Task 3: Provision an R Server on HDInsight with Spark Cluster 3](#_Toc493690291)

[Task 4: Prepare an SSH client 8](#_Toc493690292)

[Exercise 1: Environment setup 11](#_Toc493690293)

[Task 1: Download and open the vending machines starter project 11](#_Toc493690294)

[Task 2: IoT Hub 11](#_Toc493690295)

[Task 3: Microsoft R Server on Linux 14](#_Toc493690296)

[Task 4: Storage 17](#_Toc493690297)

[Task 5: Cognitive Services Face API 21](#_Toc493690298)

[Task 6: SQL Database 24](#_Toc493690299)

[Exercise 2: Create Dynamic Pricing Model 25](#_Toc493690300)

[Task 1: Create a model locally 25](#_Toc493690301)

[Task 2: Try a prediction locally 28](#_Toc493690302)

[Task 3: Create the model in R Server on HDInsight 30](#_Toc493690303)

[Task 4: Create predictive service in R Server Operationalization 32](#_Toc493690304)

[Exercise 3: Implement dynamic pricing 37](#_Toc493690305)

[Task 1: Implement photo uploads to Azure Storage 37](#_Toc493690306)

[Task 2: Invoke Face API 37](#_Toc493690307)

[Task 3: Invoke pricing model 39](#_Toc493690308)

[Task 4: Configure the Simulator 39](#_Toc493690309)

[Task 5: Test dynamic pricing in Simulator 39](#_Toc493690310)

[Exercise 4: Implement purchasing 43](#_Toc493690311)

[Task 1: Create the transactions table 43](#_Toc493690312)

[Task 2: Configure the Simulator 44](#_Toc493690313)

[Task 3: Test purchasing 46](#_Toc493690314)

[Exercise 5: Implement device command and control 47](#_Toc493690315)

[Task 1: Listen for control messages 47](#_Toc493690316)

[Task 2: Send control messages 48](#_Toc493690317)

[Task 3: Configure the DeviceControlConsole and the Simulator 48](#_Toc493690318)

[Exercise 6: Analytics with Power BI Desktop 52](#_Toc493690319)

[Task 1: Build the query and create the visualization 52](#_Toc493690320)

[Exercise 7: Cleanup 58](#_Toc493690321)

# Intelligent Vending Machines

## 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.

## Requirements

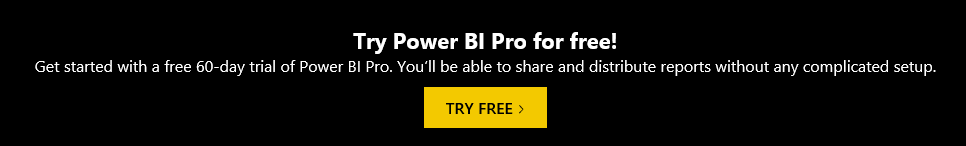
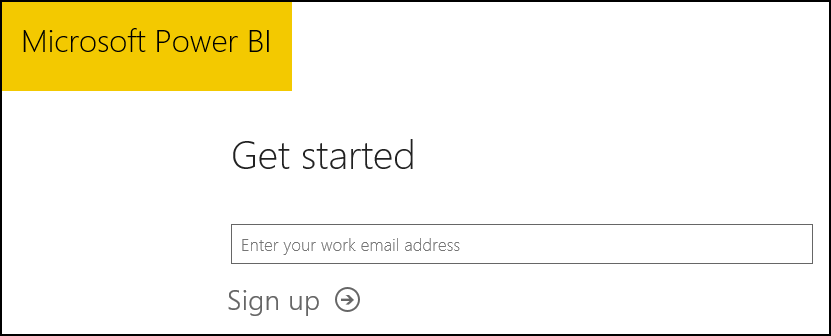
* Microsoft Azure subscription must be pay-as-you-go or MSDN.
  + Trial subscriptions will *not* work.
* Local machine or a virtual machine configured with:
  + Visual Studio Community 2017 or later
  + Azure SDK 2.9 or later for Visual Studio
  + Azure PowerShell 1.0.0 or later
  + [Microsoft R Client 3.3.3 or later](http://aka.ms/rclient/download)
  + [R Tools for Visual Studio](https://aka.ms/rtvs-current) 0.3.2 or later
  + [Power BI Desktop](https://powerbi.microsoft.com/desktop) (June 2016 build or later)
* A running R Server on HD Insight Spark cluster (see Exercise 0).

## Exercise 0: Before the hackathon

Duration: 60 minutes

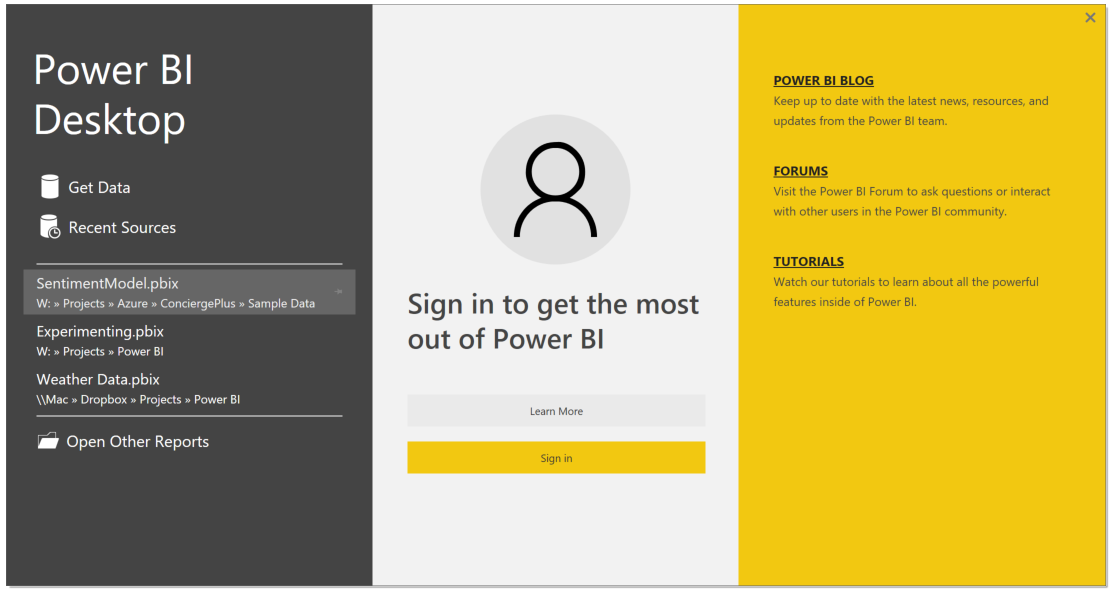
Synopsis: You should follow all of the steps provided in Exercise 0 of the Leader Guide (included in this document) *before* attending the hackathon.

### Task 1: Provision Power BI

1. If you do not already have a Power BI account, go to <https://powerbi.microsoft.com/en-us/features/>.
2. Scroll down until you see the Try Power BI for free section of the page, and click the Try Free button. 
3. On the page, enter your work email address (which should be the same account as the one you use for your Azure subscription) and select **Sign up**. 
4. Follow the on-screen prompts and your Power BI environment should be ready within minutes. You can always return to it via <https://app.powerbi.com>.

### Task 2: Provision Power BI Desktop

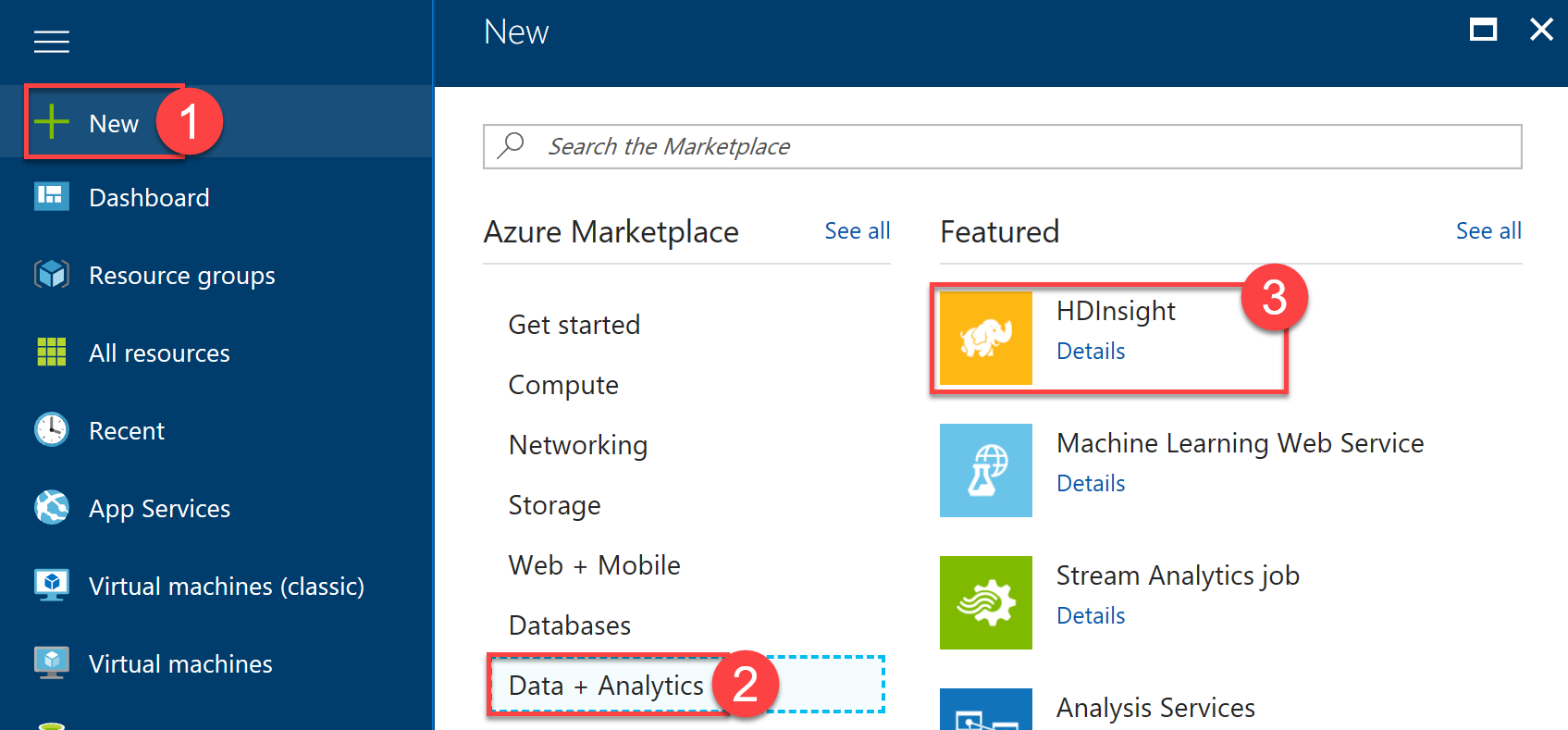
1. Download Power BI Desktop from <https://powerbi.microsoft.com/desktop>.
2. Step through the installation wizard.
3. Run Power BI Desktop from your Start Menu.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%204.53.53%20AM.png
4. At the initial dialog, select Sign In and confirm that you can sign in to Power BI using your work or school credentials.

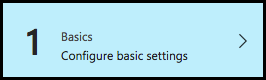
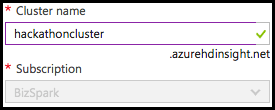
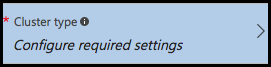
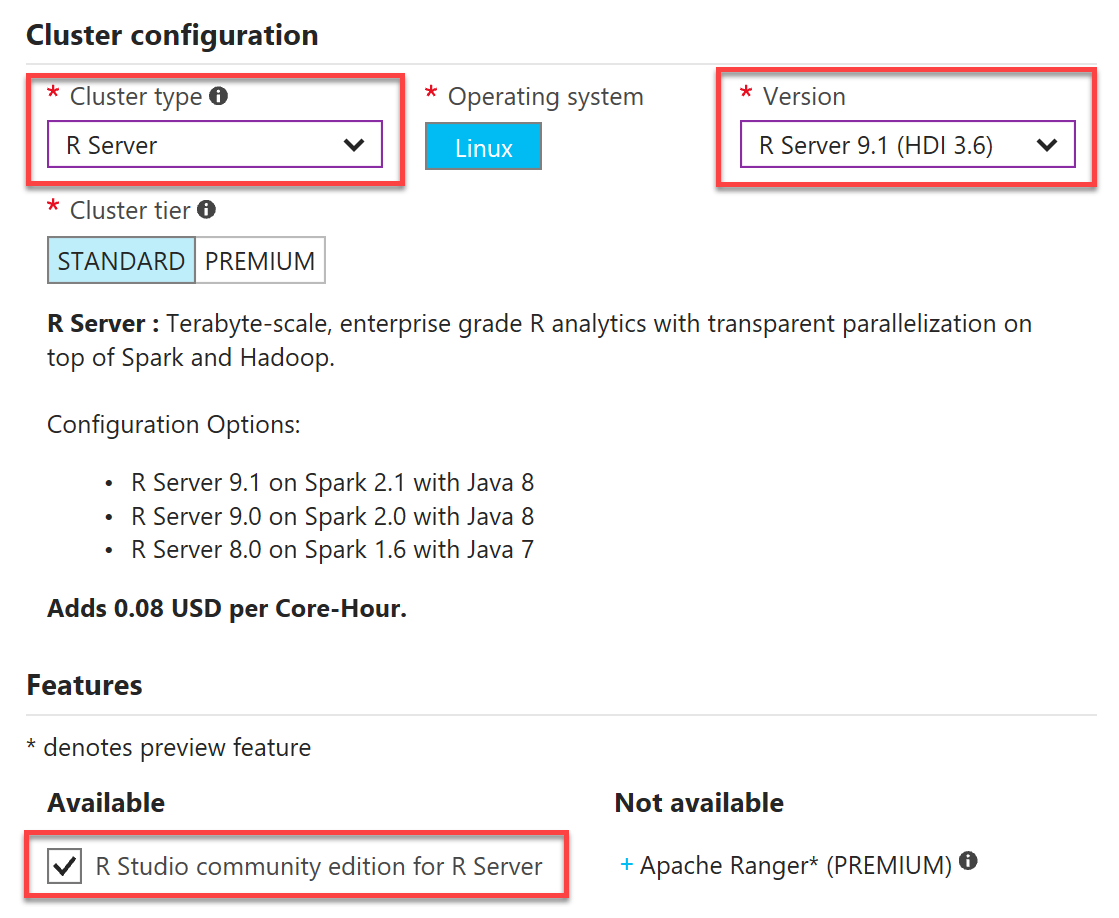
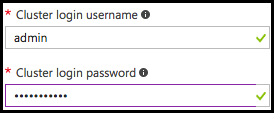
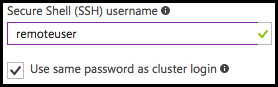
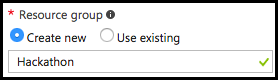
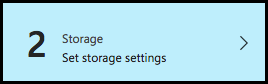
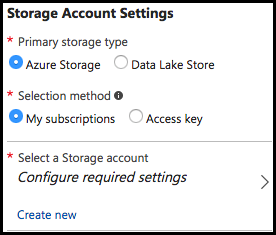
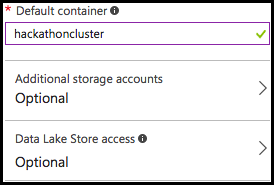
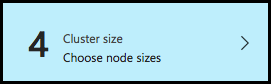
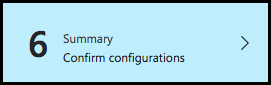


### Task 3: Provision an R Server on HDInsight with Spark Cluster

Using the Azure Portal, provision a new HDInsight cluster.

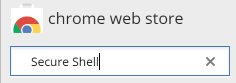
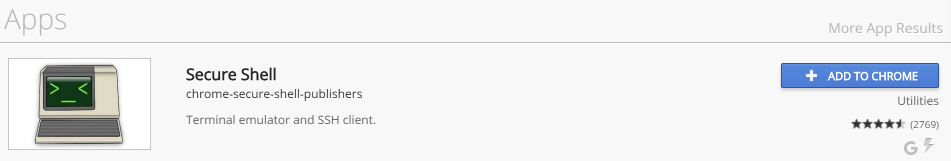
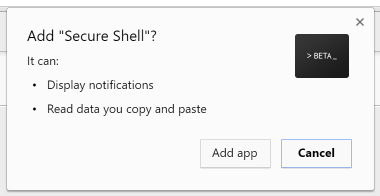
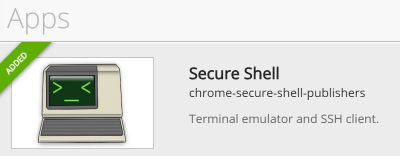
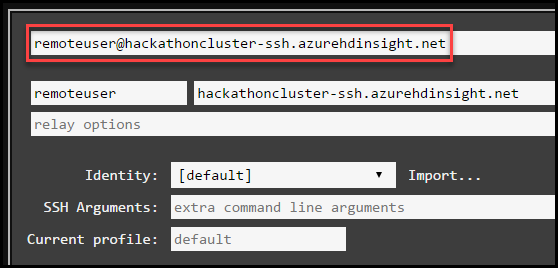
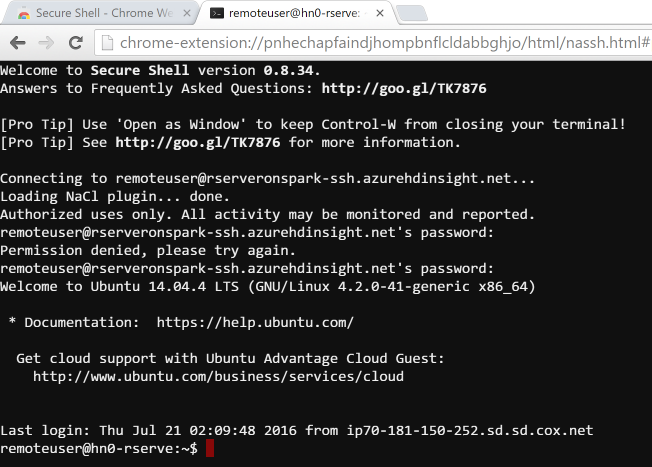
1. Open a browser and go to <https://portal.azure.com>.
2. Select **+New**, select **Data + Analytics**, **HDInsight**.



1. In the HDInsight blade, select the **Custom (size, settings, apps)** option. 
2. Enter the Basics blade to configure basic settings for the cluster.
   1. Enter a unique Cluster name (verified by the green checkmark), and select the Azure subscription into which you want to deploy the cluster. 
   2. Click **Cluster type, *Configure required settings***. 
      1. In the Cluster configuration blade, set the **Cluster type** to **R Server** and the **Version** to **R Server 9.1 (HDI 3.6)**. 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.  
         
      2. Click **Select** to close the Cluster configuration blade.
   3. Leave Cluster login username as **admin**, and enter a Cluster login password.
   4. Set the Secure Shell (SSH) username to **remoteuser** (this is required), and leave the Use same password as cluster login checkbox checked. 
   5. Under Resource group, select the Create new radio button, and enter **Hackathon** for the resource group name. ****
   6. Select the desired location from the dropdown list. 
   7. Click **Next** to move on to the storage settings.
3. Enter the Storage blade to configure storage settings for the cluster. 
   1. Leave the Primary storage type set to **Azure Storage**, and the Select Method set to **My subscriptions**, then choose or create a new storage account. 
   2. Set the Default container to the name of your cluster. Leave Additional storage accounts and Data Last Store access unconfigured. 
   3. Leave the Metastore settings blank.
   4. Click **Next**
4. Click **Next** on the Applications blade to move on the cluster size configuration.
5. Enter the Cluster size blade to configure the size settings for the cluster. 
   1. Set the Number of Worker Nodes to **2.** ****
   2. Select **Worker node size**, select **D12 v2**, and select **Select**. 
   3. Select **Head node size**, select **D12 v2**, and select **Select**. 
   4. Leave the Zookeeper node sizes set to **A2**. 
   5. Select **R-Server edge node size**, select **D12 v2**, and select **Select**. 
   6. Click **Next**.
6. Click **Next** on the Advanced settings blade to move to the Cluster summary blade.
7. Enter the Cluster summary blade to review the cluster settings. 
   1. Click **Create** on the Cluster summary blade.
   2. Your cluster should be ready in 20 – 30 minutes.

### Task 4: Prepare an SSH client

In this task, you will prepare an SSH client that you will use to access your HDInsight cluster.

1. Download and install the Chrome browser from: <https://www.google.com/intl/en/chrome/browser/desktop/index.html>.
2. Launch Chrome.
3. Navigate to the Chrome Web Store at: <https://chrome.google.com/webstore/category/apps>.
4. In the search field, type Secure Shell.  
   
5. Find the app titled Secure Shell and click Add to Chrome.  
   
6. Select Add app when prompted.  
   
7. Return the Chrome Web Store, search for Secure Shell, and select the listing.  
   
8. In the dialog that appears, click Launch App.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%207.07.39%20PM.png
9. The SSH client should appear in a new browser tab. In the box at the center, enter remoteuser@<clustername>-ssh.azurehdinsight.net, replacing <clustername> with the name of the HDInsight cluster created above.  
   
10. Click Connect. Note: you will need to wait for the cluster deployment to complete in Azure before you can complete this step.  
    ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%207.17.49%20PM.png
11. Respond to any prompts in the SSH window, and enter the password for **remoteuser** when prompted.
12. Use this App for SSH during the hackathon for any instructions requiring an SSH connection. You can repeat these steps any time to re-connect.  
       
      
    Note: If you prefer, you can also use the Linux Subsystem for Windows to run a Bash shell to execute commands over SSH, but the setup and use of this is outside the scope of this document.

## 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. Download the starter project from the following URL: <http://bit.ly/2w6t2qz>
2. Unzip the contents.
3. Open VendingMachines.sln with Visual Studio.

**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: IoT Hub

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

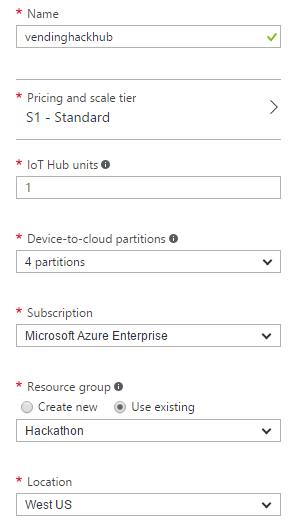
1. In your browser, navigate to the **Azure Portal** (<https://portal.azure.com)>.
2. Select **+New** in the navigation bar at the left.



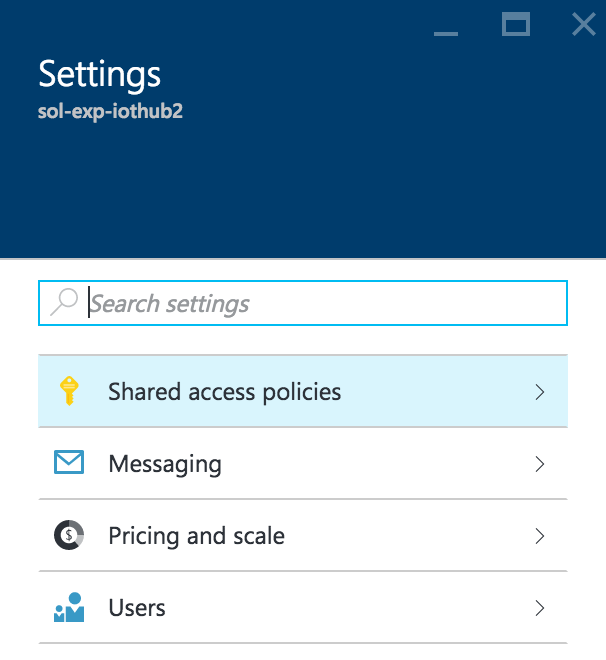
1. Within the **New** blade, select **Internet of Things**, then select **IoT Hub** in the Internet of Things blade.



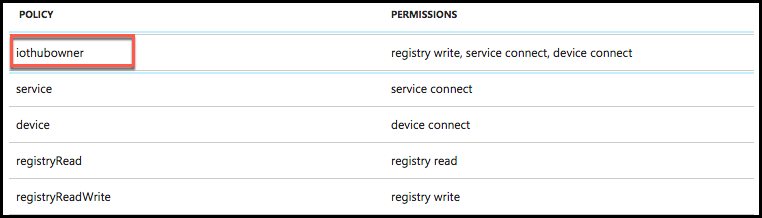
1. In the **IoT Hub** blade, provide a name for your new IoT Hub, choose a **Pricing and scale tier** (**S1** **Standard**), set the **IoT Hub units** to **1**, select the Use existing radio button under Resource group, and select **Hackathon** from the resource group list, and choose the **Location**.



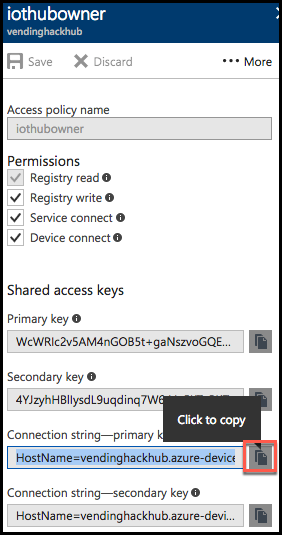
1. Click **Create**.
2. In the **Settings** blade that appears for your new IoT Hub, select **Shared access policies**.



1. Select **iothubowner** policy.

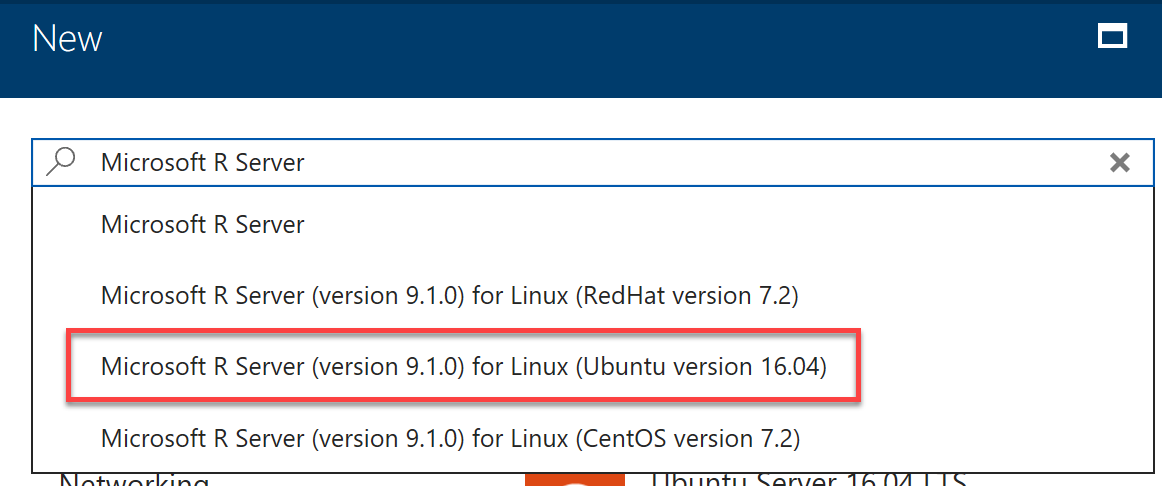
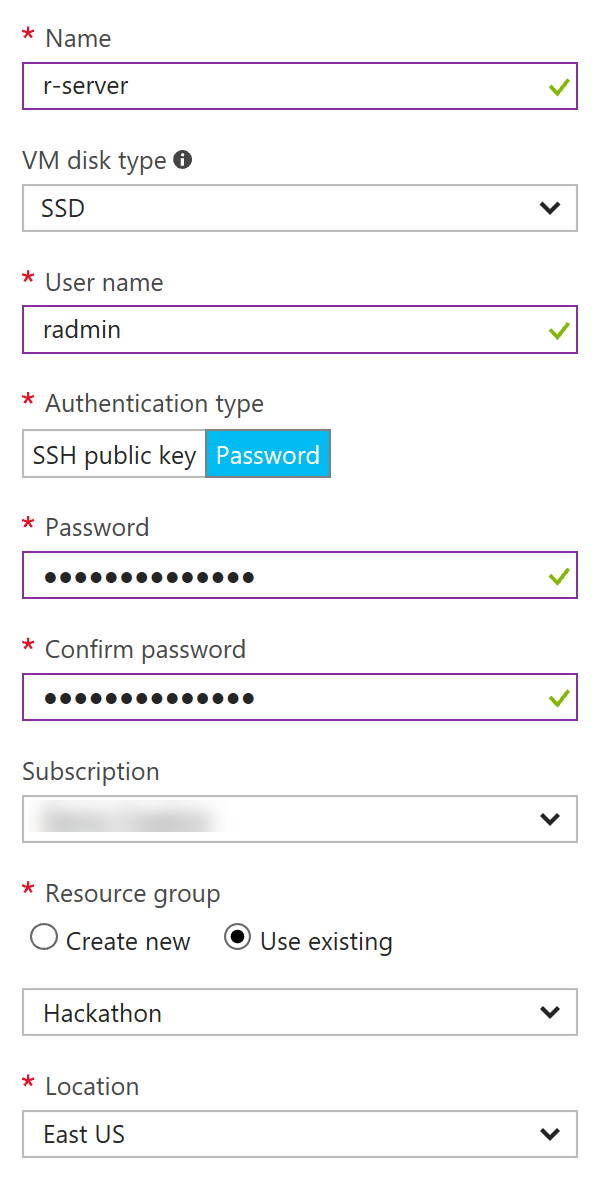
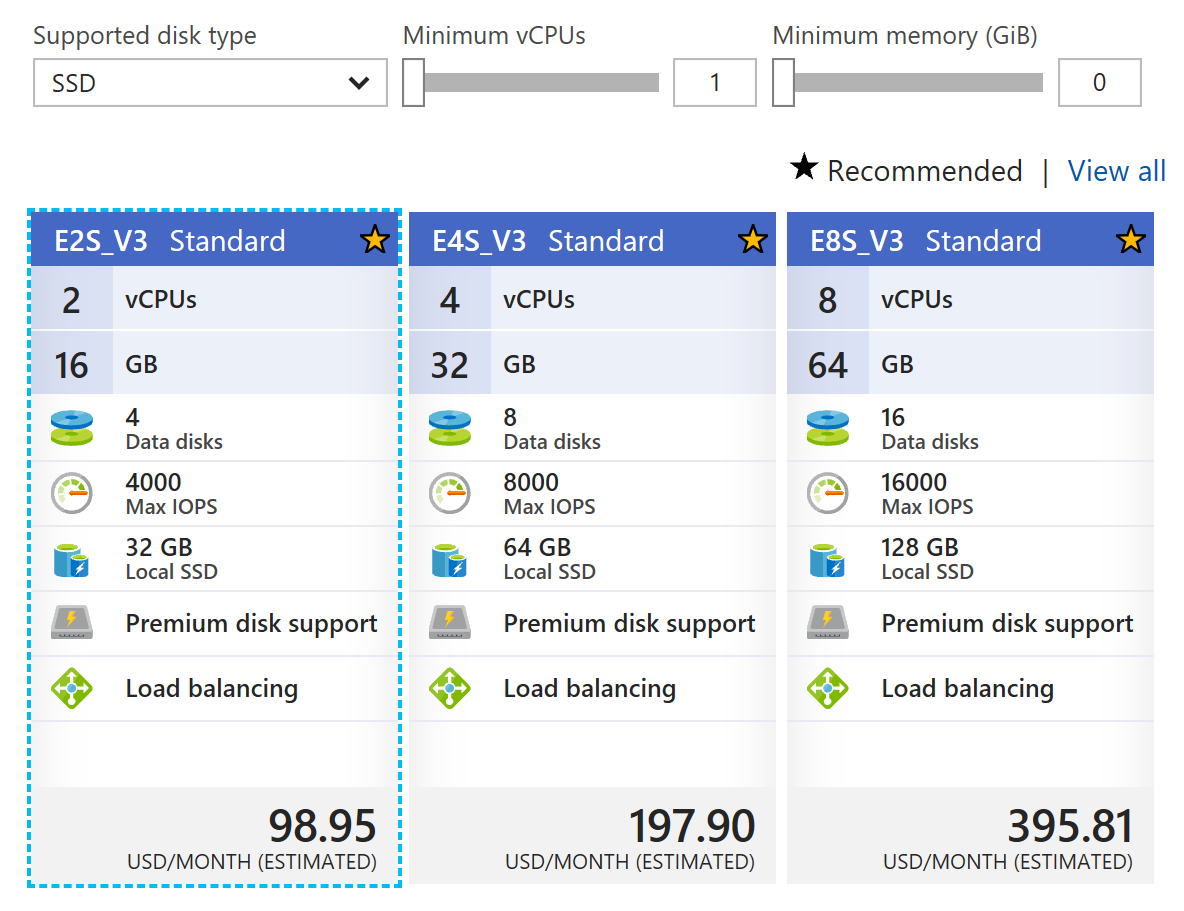


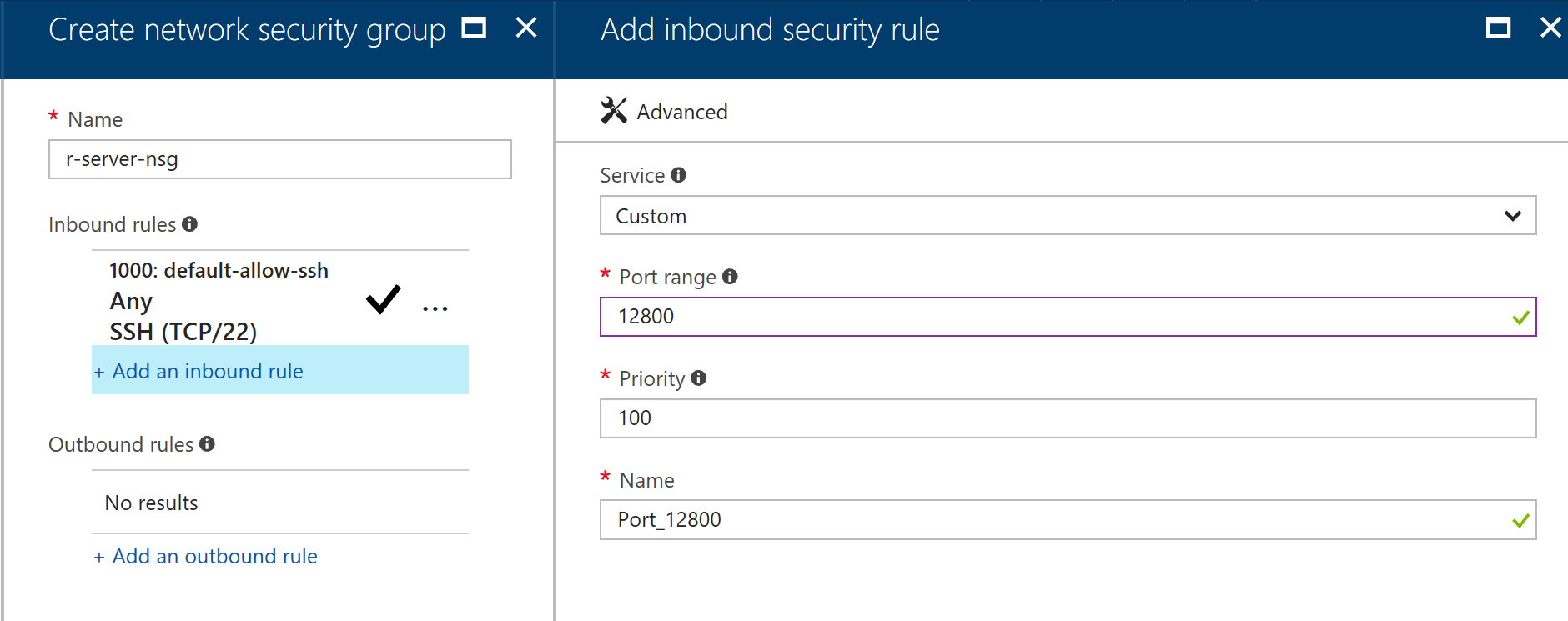
1. In the iothubowner blade, click the **Copy** button to the right of the **Connection string - primary key** field, located under the **Shared access keys** section. Paste the connection string value into a text editor, such as Notepad, as this will be needed later in the hackathon.



### Task 3: Microsoft R Server on Linux

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

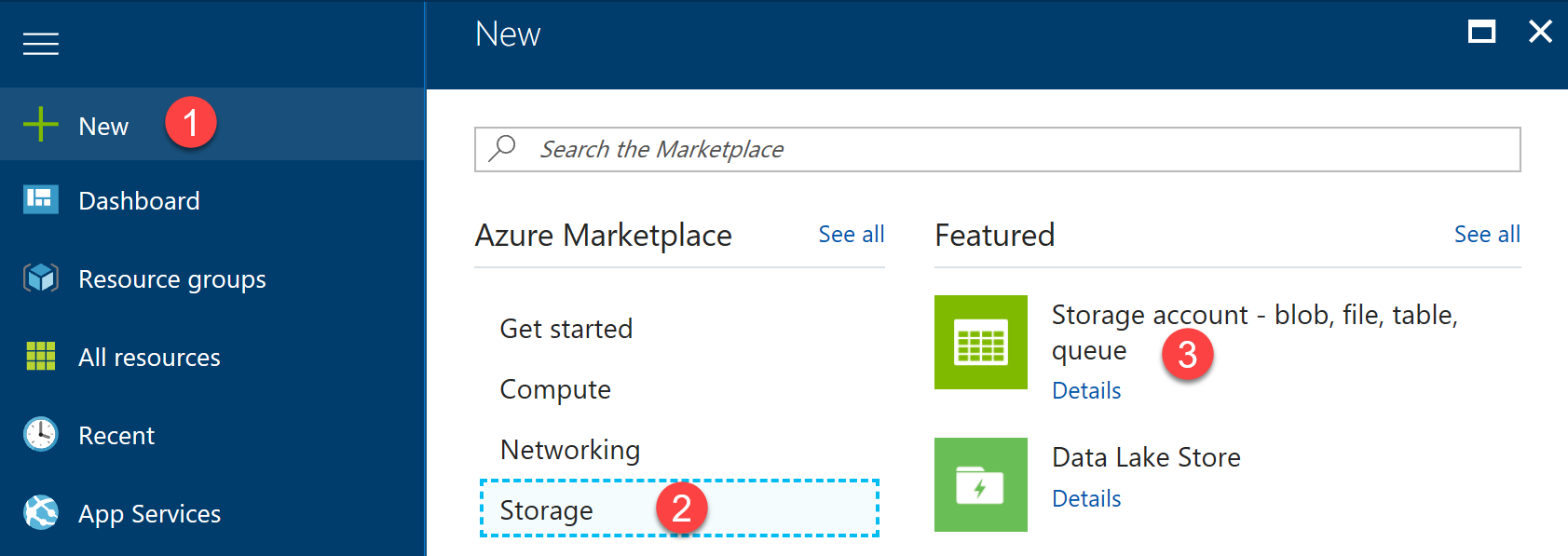
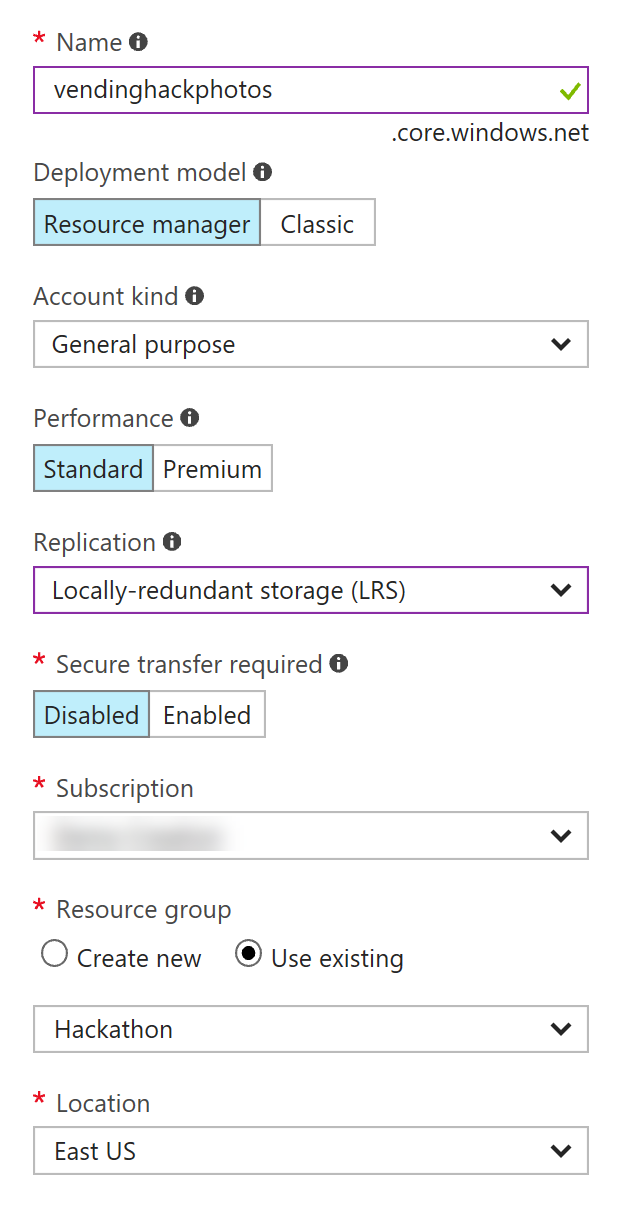
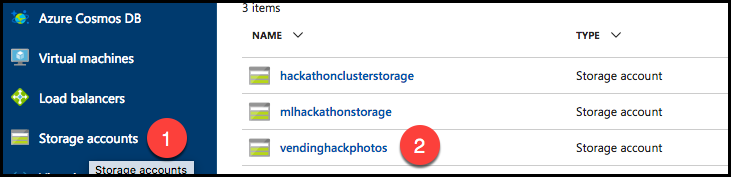
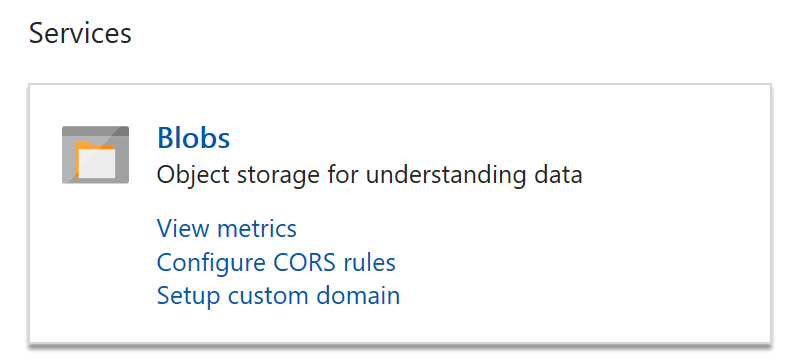
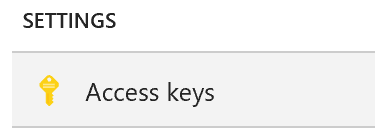
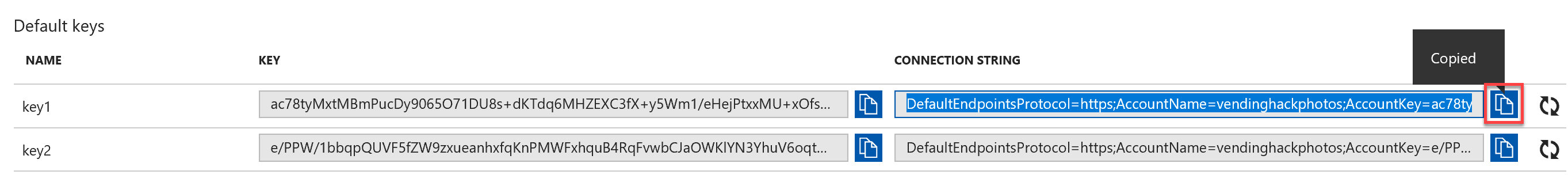
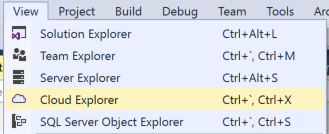
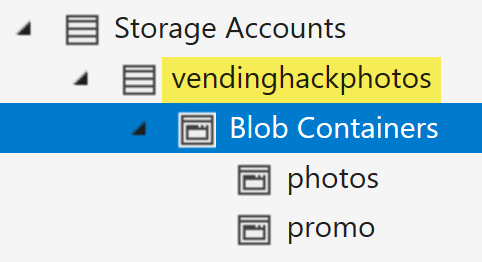
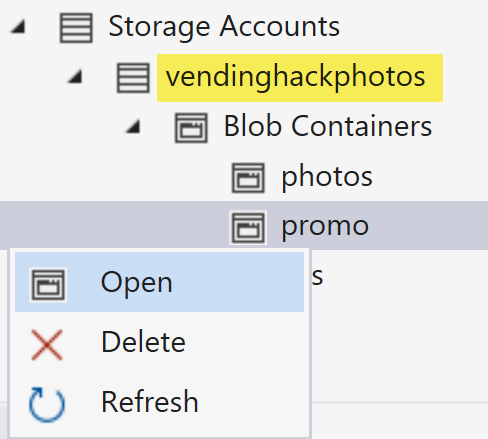
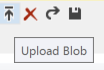
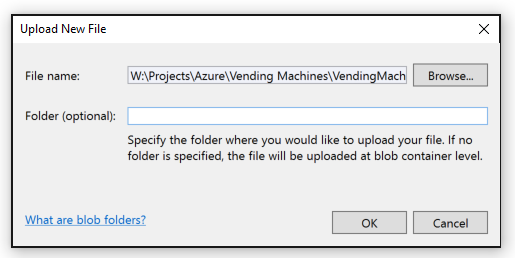
1. Within the Azure Portal, select + New.
2. In the Search field, type Microsoft R Server.
3. In the results list, select Microsoft R Server (version 9.1.0) for Linux (Ubuntu version 16.04).  
   
4. On the blade that appears, select Create.
5. In the Basics blade, enter:
   1. Name: must be unique
   2. User name: such as radmin
   3. Authentication type: select Password
   4. Password: enter and confirm your password (make note of your username and password)
   5. Resource group: select the Resource Group you created earlier
   6. Location: select a location nearest to you  
      
6. Click OK.
7. Under Choose a size, select E2S\_V3 Standard.  
   
8. Under Settings, select Network security group (firewall).
   1. Click Create new under Choose network security group.
   2. Enter a name, such as my-r-nsg.
   3. Select + Add an inbound rule.
   4. Select Custom under Service and enter the following:
      1. Port range: 12800
      2. Priority: 100
      3. Name: Port\_12800 (should be auto-generated)



1. Click Ok.
2. Click Purchase.

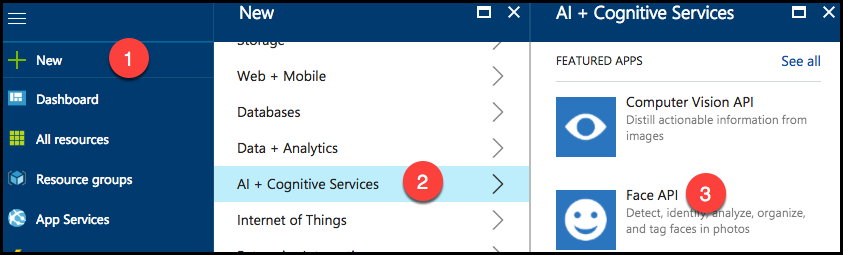
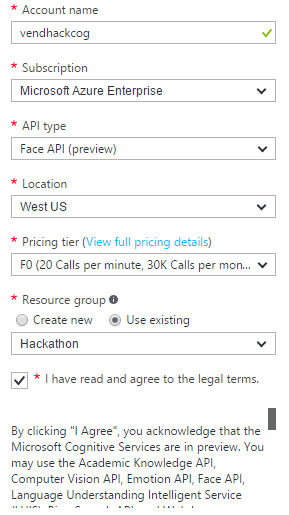
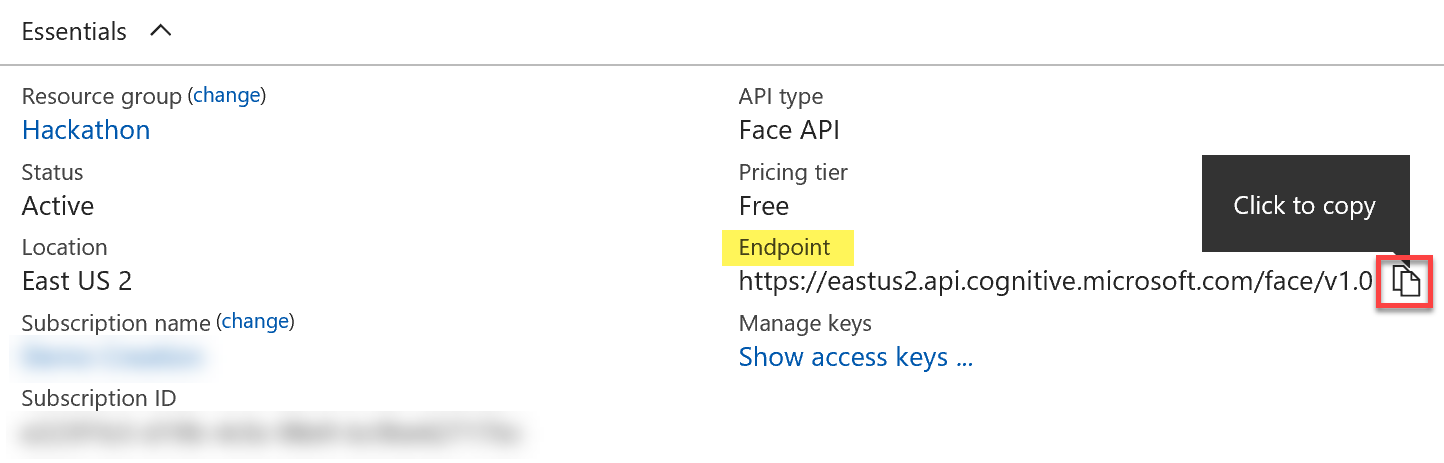
### Task 4: Storage

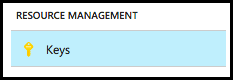
In these steps, you will provision a storage account that will be used for storing photos sent from the vending machine simulator and for the storage of the promotional package resources.

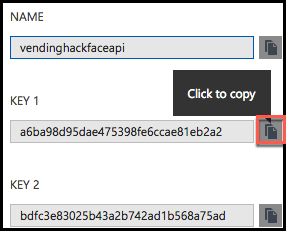
1. Using the Azure Portal, select **+ New, Storage, Storage account**. 
2. On the Create storage account blade:
   1. Enter a unique name for the storage account.
   2. Leave Resource Manager selected for the deployment model.
   3. Leave the Account kind set at General purpose.
   4. Leave Performance set to Standard.
   5. Set Replication to **Locally-redundant storage (LRS)**.
   6. Leave Storage service encryption disabled.
   7. Select your Subscription group
   8. Under Resource group, select the Use existing radio button, and select **Hackathon** from the resource group list.
   9. Select a Location to be consistent with the other resources you have created.  
      
3. Click **Create**.
4. 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. 
5. On the Storage account blade, select **Blobs**.  
   
6. In the Blob service blade, select **+ Container** from the command bar.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%207.09.49%20AM.png
7. On the New container blade, set the name to **photos** and select **Private** as the Access type.  
   
8. Click **OK**.
9. Repeat steps 6-8 to create another container named “promo”.
10. You should now see both containers listed on the Blob service blade. 
11. Close the Blob service blade to go back to the Storage blade. Select Access Keys from the left-hand menu.  
    
12. 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.  
    
13. Open Visual Studio and from the View Menu select Cloud Explorer.  
    
14. Expand the Storage account that you just created, and the Blob Containers item underneath it.  
    
15. Right-click the promo container and select Open.  
    
16. Select the Upload blob button.  
    
17. Select Browse.
18. In the dialog, select the three images CoconutWater.png, Water.png, and Soda.png from the starter solution Simulator\Images folder and select Open.
19. Select OK on the Upload New File Dialog to upload the images to the container.  
    

### Task 5: 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.

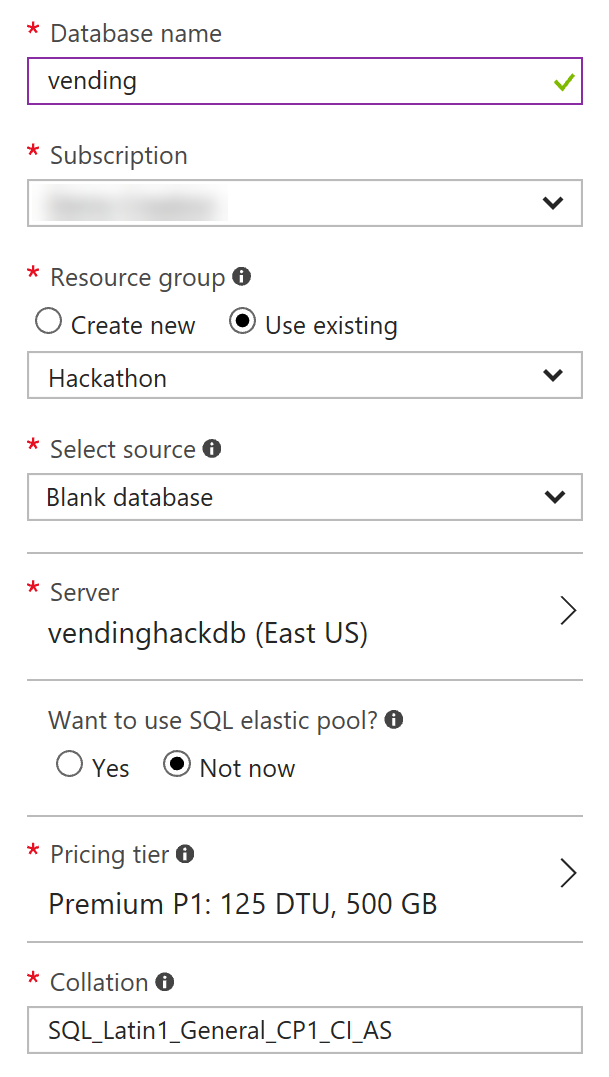
1. In the Azure Portal, select **+ NEW, AI + Cognitive Services,** and select **Face API**.  
   
2. On the Create Face API blade:
   1. Enter a Name.
   2. Choose your Subscription.
   3. Choose a Location near you.
   4. For the Pricing tier, select the Free tier (**F0**) from the drop-down list.
   5. Select Use existing under Resource group, and select the **Hackathon** resource group from the list.
   6. Check the box confirming you have read and understand the legal terms. 
   7. Click **Create** to provision the Cognitive Services account.
3. When it finishes provisioning, browse to the Cognitive Services Face API by clicking on All Resources, and selecting the newly provisioned Face API from the list of available resources.
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.



1. 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: SQL Database

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

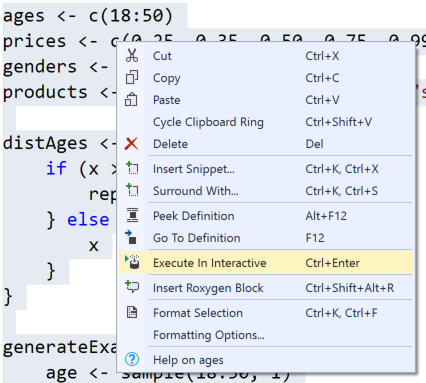
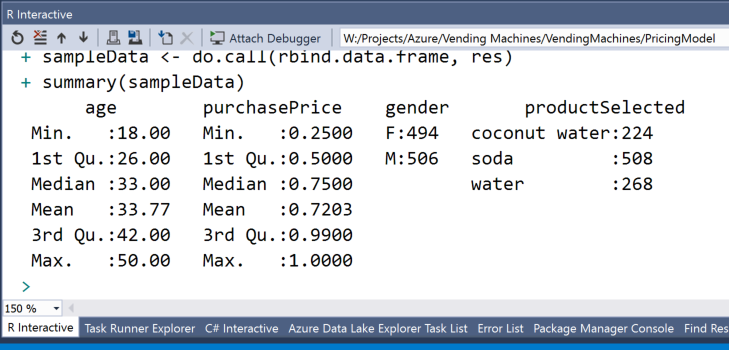
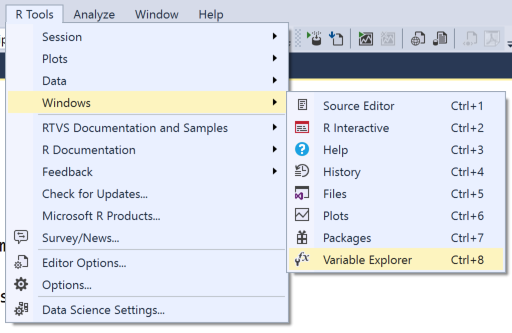
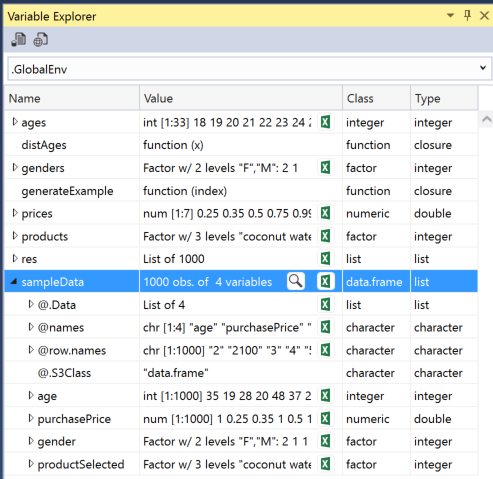
1. In the Azure Portal, select **+ NEW**.
2. Select **Databases**.
3. Select **SQL Database**.
4. In the SQL Database blade:
   1. Enter **vending** in the Database name field.
   2. Choose your Subscription.
   3. Under Resource Group, select Use existing and choose **Hackathon** from the resource group list.
   4. Leave Select source as Blank database.
   5. For Server, choose an existing Server or create a new one.
   6. Leave Want to use SQL elastic pool as **Not now**.
   7. Change the Pricing tier to Premium P1.  
      
5. Click **Create**.
6. Once the SQL Database finishes provisioning, select the Show database connection strings.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%209.09.02%20AM.png
7. 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

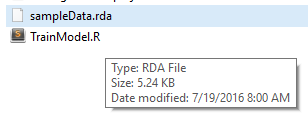
In this exercise, you will create a machine learning model that predicts the purchase price for an item sold by the vending machine provided the demographics of the customer and the item. You will then operationalize this model by exposing it as a web service hosted in Azure Machine Learning and test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

### Task 1: Create a model locally

1. Within Visual Studio Solution Explorer, expand the PricingModel project and open the file TrainModel.R.
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 with the following code:

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

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

1. Highlight the save line and select Execute In Interactive.
2. Open File Explorer and navigate to the location of the PricingModel project on disk. You should see the file sampleData.rda on disk.  
   
3. 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)

1. 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")

1. 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")

1. Save your changes to TrainModel.R.
2. Highlight TODO items 2 through 4 and execute them in interactive.
3. 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**.
2. Replace TODO 1 with the following:

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

inputExample[1,]$age <- 30

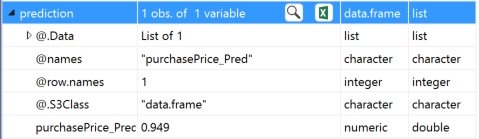
inputExample[1,]$gender <- "F"

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

1. Replace TODO 2 with the following:

# TODO: 2. Execute the prediction

prediction <- rxPredict(pricingModel, data = inputExample)

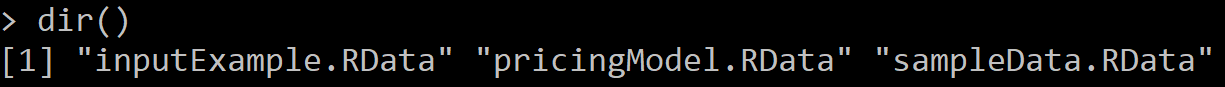
1. Highlight all the script in the file and execute it in interactive.
2. Using Variable Explorer, expand the prediction variable and observe the price the model suggested to use for purchasing the coconut water for input of a 30-year-old female.  
   

### Task 3: Create the model in R Server on HDInsight - Optional

1. SSH into your deployed R Server in HDInsight cluster. (You can get the Host name for your cluster from the HDInsight Blade in the Azure Portal, Settings, and then Secure Shell.) For example:  
    ssh <user>@rserveronspark-ssh.azurehdinsight.net.
2. When prompted if you want to continue connecting, enter yes.
3. Enter your password.
4. At the command prompt, type R to load the R shell (be sure to use a capital letter “R”).
5. Run the following command to create a spark context for R:

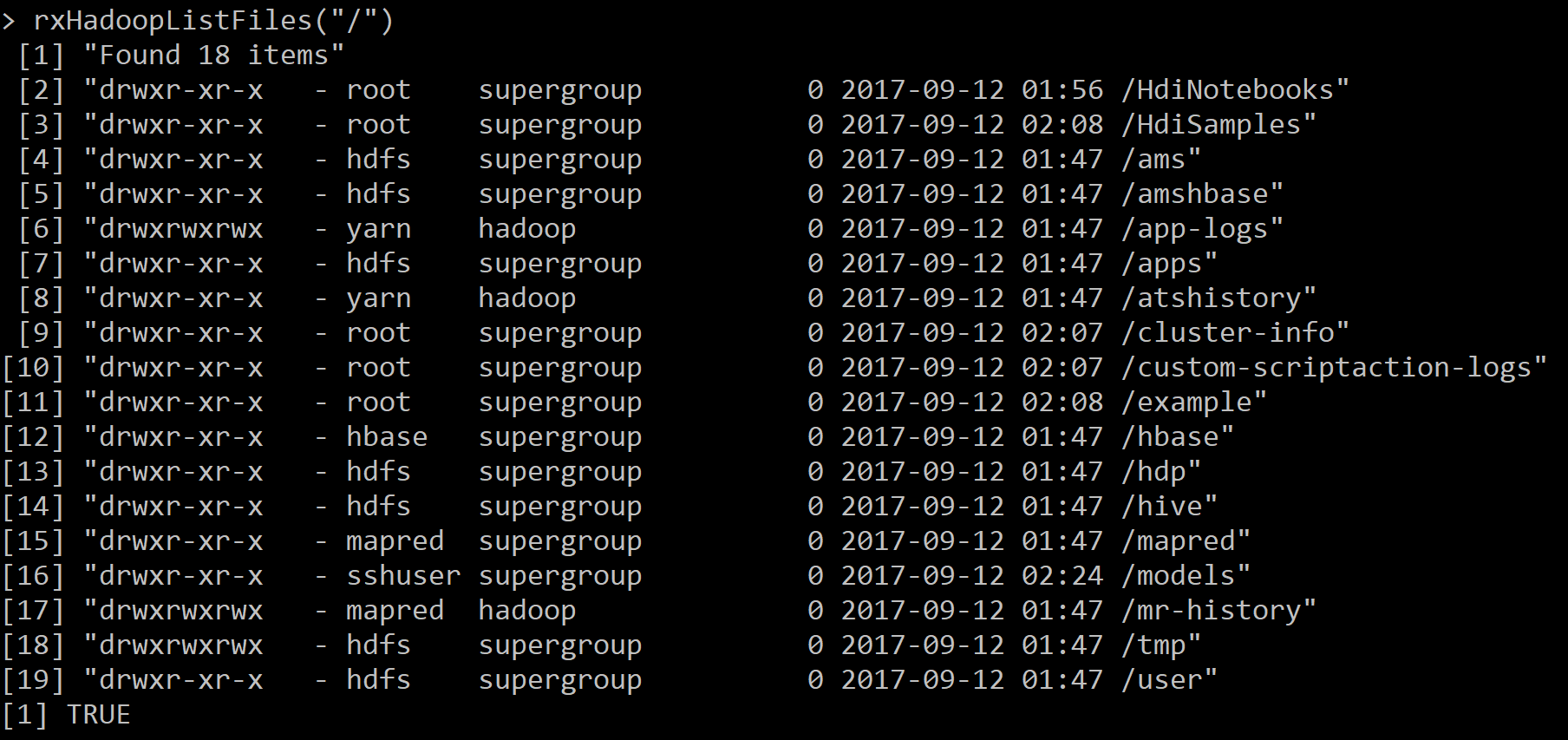
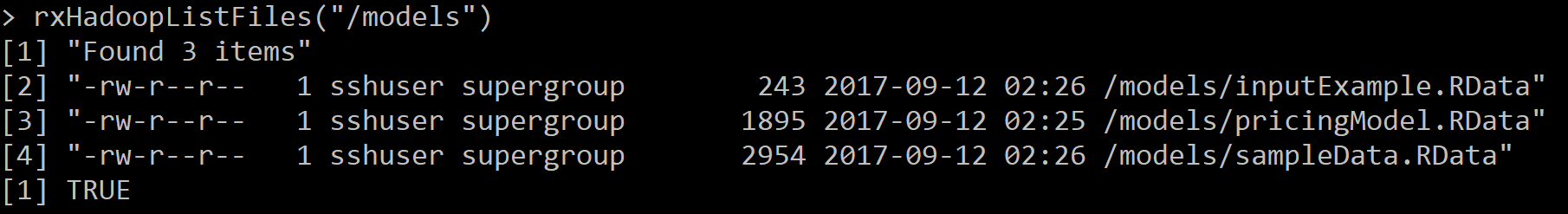
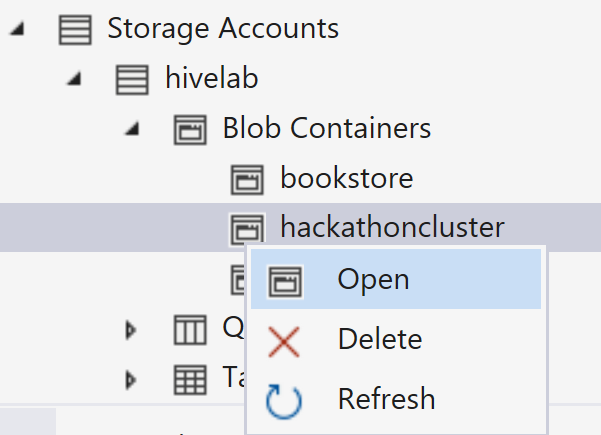
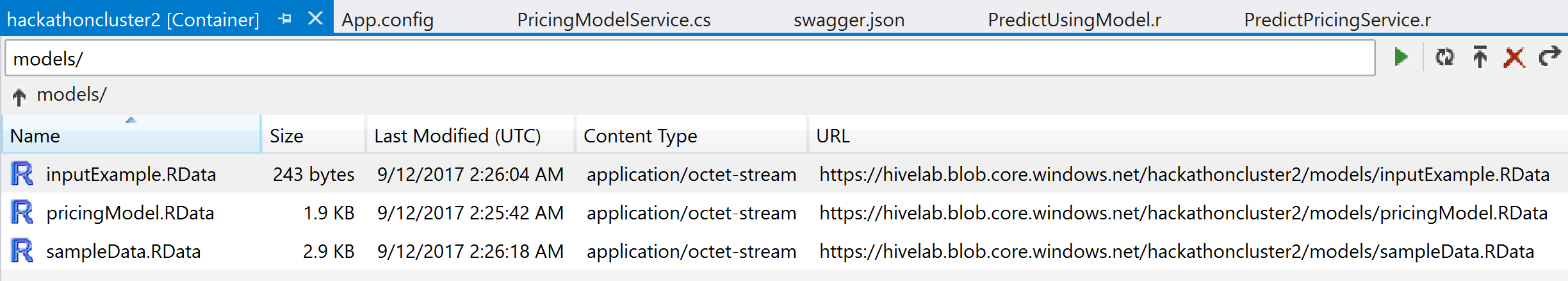
sparkCluster <- RxSpark()

rxSetComputeContext(sparkCluster)

1. In Visual Studio, open TrainModel.r and copy the entire script.
2. 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.)
3. When the script has finished executing, type the following:  
   dir().
4. You should see it list the three files created by the script, as follows:  
    
5. 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.

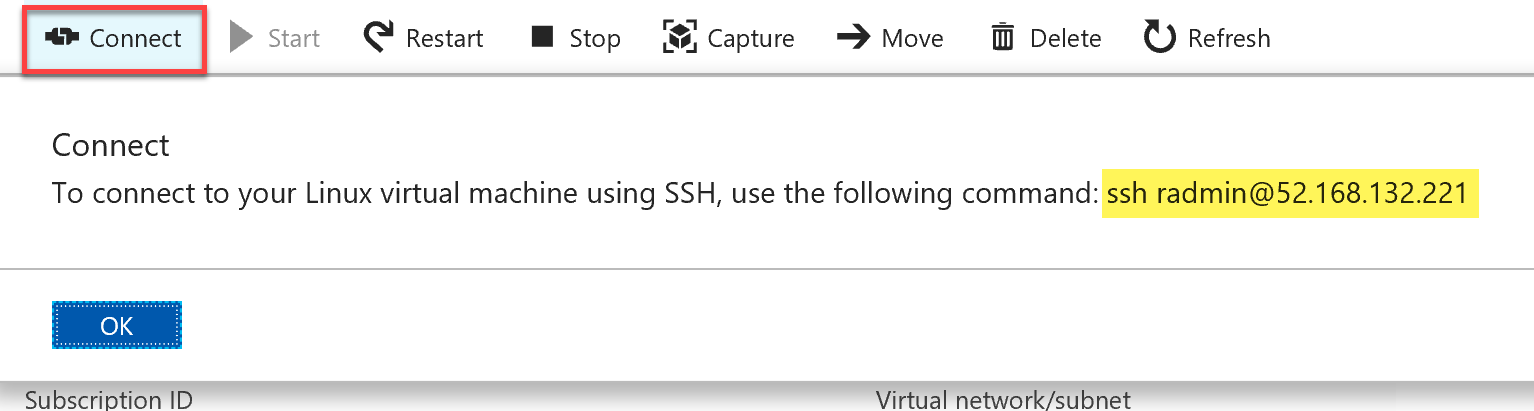
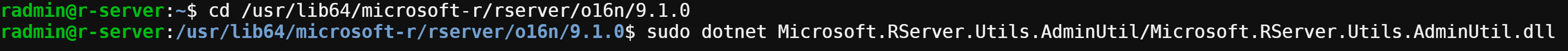
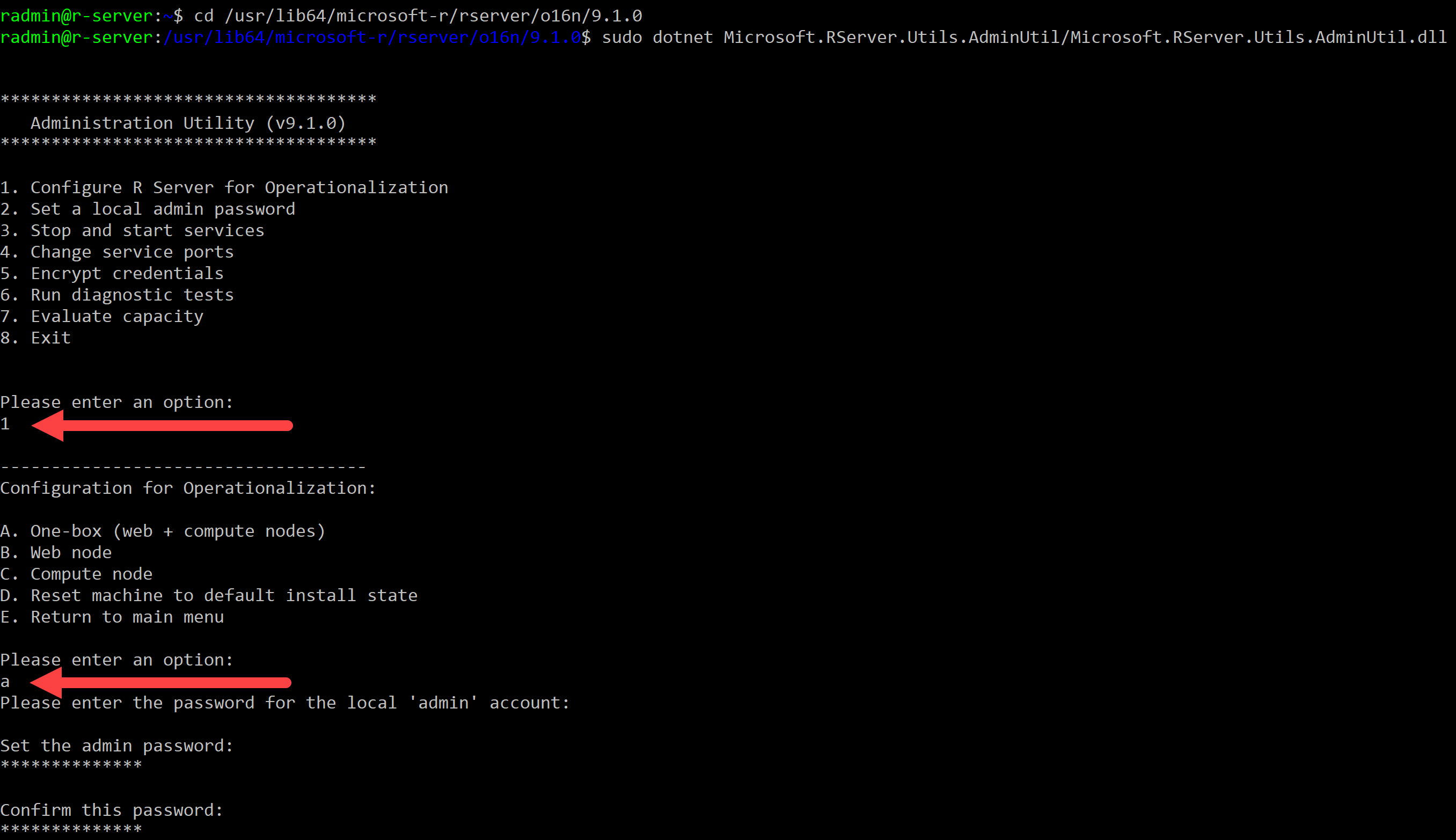
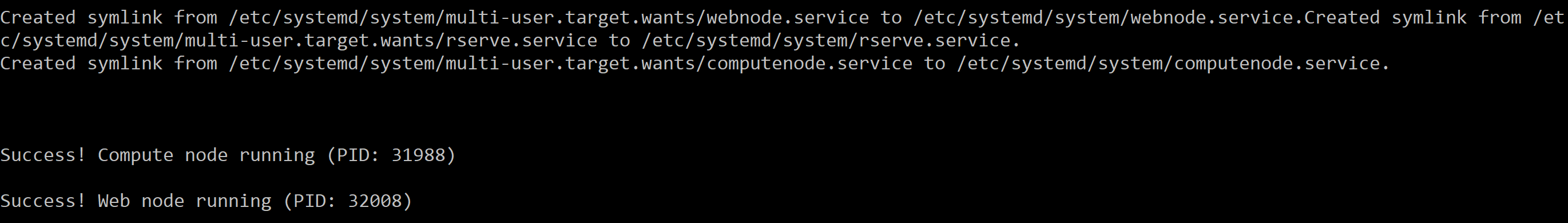
modelExportDir <- "/models"

rxHadoopMakeDir(modelExportDir)

1. List the contents of the /models directory and confirm your 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("/")  
     
   
2. Copy the pricingModel.RData from the local directory to HDFS by running the following command:  
   rxHadoopCopyFromLocal("pricingModel.RData", modelExportDir)
3. Repeat the previous step for inputExample.RData and sampleData.RData.
4. Run the following command to verify the three files now exist in HDFS (and Blob storage), under /Models.  
   rxHadoopListFiles("/models")
5. The output should look similar to the following:  
   
6. Using Visual Studio, Cloud Explorer, navigate to the storage account for your HDInsight cluster, expand:   
    
7. In the editor that appears, double-click the **models** folder and verify you see your files.  
   
8. Right-click inputExample.RData and select Save As… and **choose the directory for your PricingModel project**, overwriting files if prompted.
9. Repeat the previous step for pricingModel.RData and sampleData.RData.
10. 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.

1. Open the Azure Portal.
2. Navigate to your R Server Virtual Machine.
3. On top of the Overview blade, select Connect, then copy the SSH command.  
   
4. SSH into your R Server VM. For example:  
   ssh radmin@<your-server-ip>.
5. When prompted if you want to continue connecting, enter yes.
6. Enter your password.
7. At the prompt, after successfully logging in, enter the following command:  
     
   cd /usr/lib64/microsoft-r/rserver/o16n/9.1.0
8. After changing to that directory, enter the following command to open the Administration Utility:  
     
   sudo dotnet Microsoft.RServer.Utils.AdminUtil/Microsoft.RServer.Utils.AdminUtil.dll  
     
   
9. Enter 1 for “Configure R Server for Operationalization”
10. Enter A for “A. One-box (web + compute nodes)”
11. Enter a password for the admin user. Save this password in a text editor, such as Notepad, for later use.  
    
12. After confirming your password, you should see two Success messages, indicating that the compute node and web node are running.  
    
13. Enter E to return to the main menu.
14. Enter 3 for “Stop and start services”.
15. Enter B for “Start web node”.
16. Enter D for “Start compute node”.
17. Enter E to return to the main menu.
18. Enter 8 to exit.
19. In Visual Studio, open the App.config for the Simulator project.
20. Locate the appSetting rServiceBaseAddress and enter http://<your-server-ip>:12800 for the value. (For example: http://52.168.132.221:12800)
21. Locate the appSetting rServicePassword and update its value with the password you defined when you configured your R server for Operationalization (step 11 above).
22. Save App.config.
23. Open PredictPricingService.r within the PricingModel project.
24. 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")

1. 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()  
  
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>.

1. Highlight all of the code in PredictPricingService.r and execute in interactive. The last output status in the R Interactive window should be “Published service”.
2. 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

1. 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)

1. Highlight just the TODO 3 and TODO 4 code blocks you added and execute in interactive.
2. When you scroll up through the R Interactive window results, you should see an output with your prediction similar to the following:  
     
   $success

[1] TRUE

$errorMessage

[1] ""

$outputParameters

$outputParameters$purchasePrice

[1] 0.9348741

1. 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");

1. 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);

1. 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);

}

1. 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;

1. Replace TODO 5 with the following:

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

string sasBlobToken = blob.GetSharedAccessSignature(sasConstraints);

return blob.Uri + sasBlobToken;

1. With the SAS URI to the upload photo in hand, scroll to GetPhotoDemographics to implement the call to the Face API.
2. Replace TODO 6 with the following:

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

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

1. Replace TODO 7 with the following:

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

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

1. Replace TODO 8 with the following:

//TODO 8. Invoke the Face API Detect operation

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

1. 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;

1. Save the file.

### Task 3: Invoke pricing model

1. Within MainWindow.xaml.cs, scroll to the end of Update Dynamic Price 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);

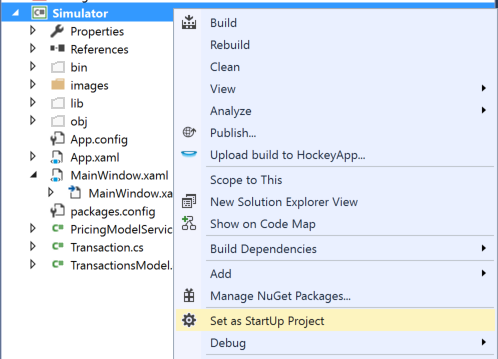
1. Save the file.

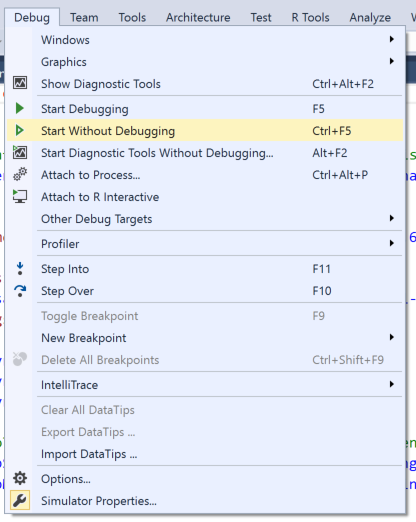
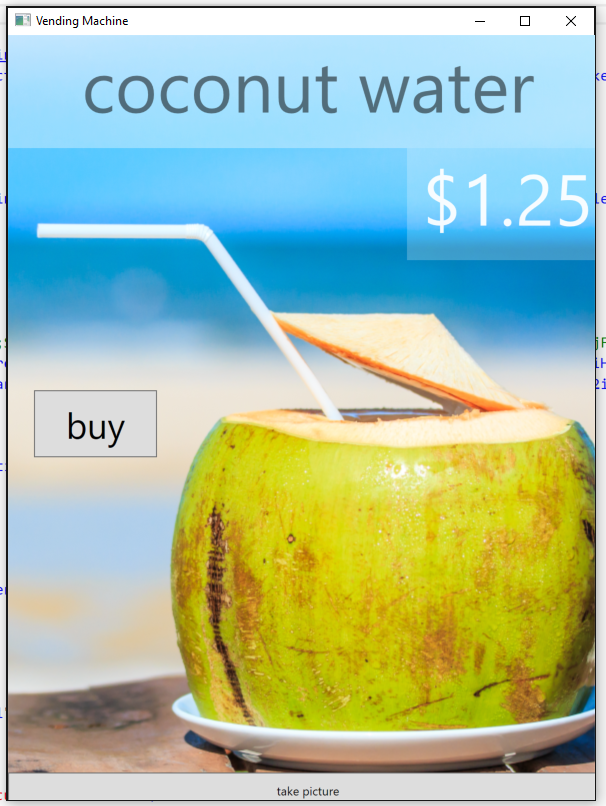
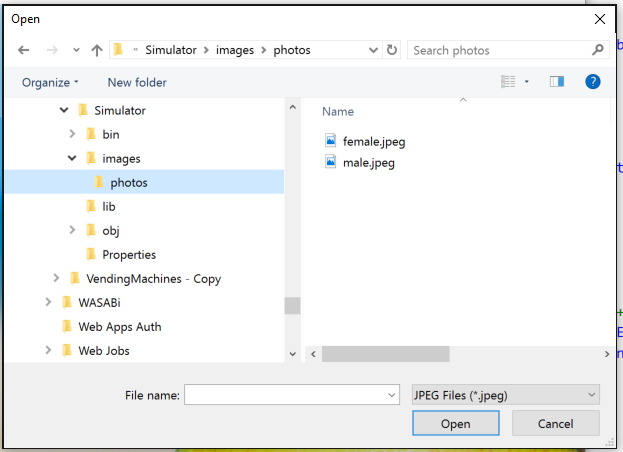
### Task 4: Configure the Simulator

1. In the Simulator project, open App.config.
2. Within the appSettings section, set the following settings:
   1. faceAPIKey: set this to the KEY 1 value for your Face API as acquired from the Azure Portal.
   2. 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)
   3. storageConnectionString: set this to the connection string of the Storage Account you created with the photos container.
3. Save the App.config.

### Task 5: Test dynamic pricing in Simulator

1. In solution explorer, right-click the Simulator project and select Build.
2. Ensure that your build generates no errors.
3. Again, in solution explorer, right-click the Simulator project and select Set as Startup Project.



1. From the Debug menu, select Start Without Debugging.  
   
2. When the vending machine simulator appears, select take picture at the bottom.  
   
3. In the dialog that appears, navigate to the images folder under Simulator\bin\debug and pick the photo of either the man or woman to upload and select Open.  
   
4. In a few moments, you should see the price change from $1.25 to whatever value the predictive model suggested.



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

## 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.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%205.25.27%20PM.png
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,

1. Replace TODO 2 with the following:

-- TODO: 2. This table should have a columanr index

INDEX Transactions\_CCI CLUSTERED COLUMNSTORE

1. Replace TODO 3 with the following:

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

MEMORY\_OPTIMIZED = ON

1. Replace TODO 4 with the following:

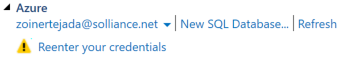
-- TODO: 4. In-memory tables should auto-elevate their transaction level to Snapshot

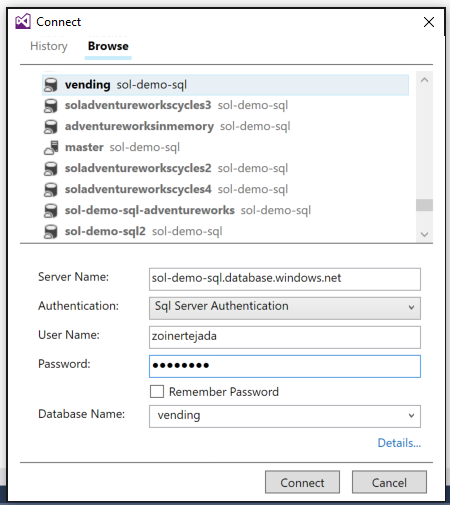
ALTER DATABASE CURRENT SET MEMORY\_OPTIMIZED\_ELEVATE\_TO\_SNAPSHOT=ON;

1. Save the script.
2. Execute the script by pressing the play icon.

../../../../../../Captures/Screen%20Shot%202016-07-20%20at%205.38.58%20PM.png

1. When prompted, sign in with your Azure credentials.

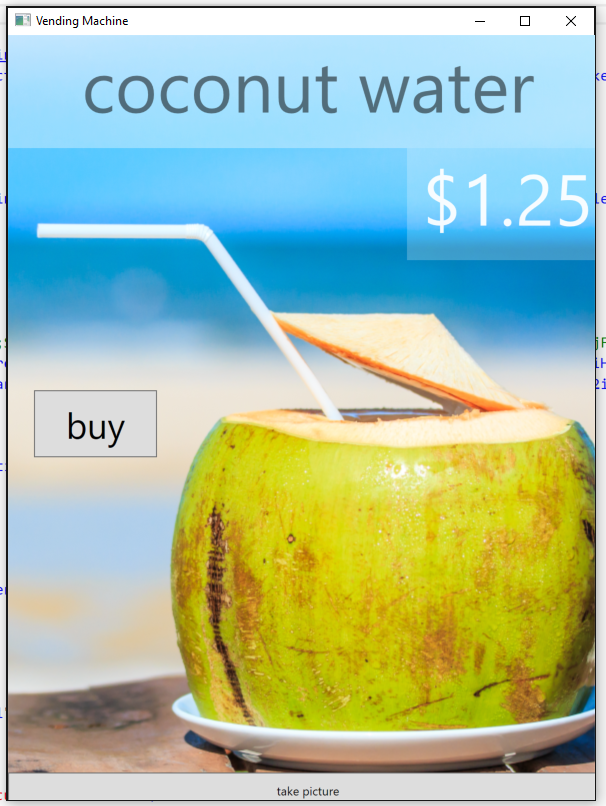
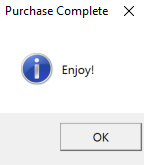


1. Expand the Azure node and select the database you created for the vending database.  
   
2. In the in fields at the bottom, enter your user name and password for the SQL Server and select Connect. The script should run successfully.

### Task 2: Configure the Simulator

1. In the Simulator project, open App.config.
2. Within the connectionString section, set the following:
   1. TransactionsModel: set the value of the connectionString attribute to the ADO.NET connection string to your SQL DB instance. When copying this value from the Azure Portal, do not forget to replace the values for {your\_username} and {your\_password} with your actual credentials.
3. Save the App.config.

### Task 3: Test purchasing

1. In solution explorer, right-click the Simulator project and select Build.
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.
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();

1. 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;

1. 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);

1. Replace TODO 4 with the following:

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

await \_deviceClient.CompleteAsync(receivedMessage);

1. 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);

1. Replace TODO 2 with the following:

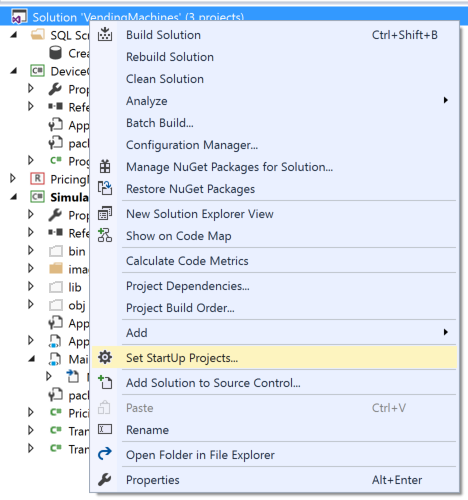
//TODO: 2. Send the command

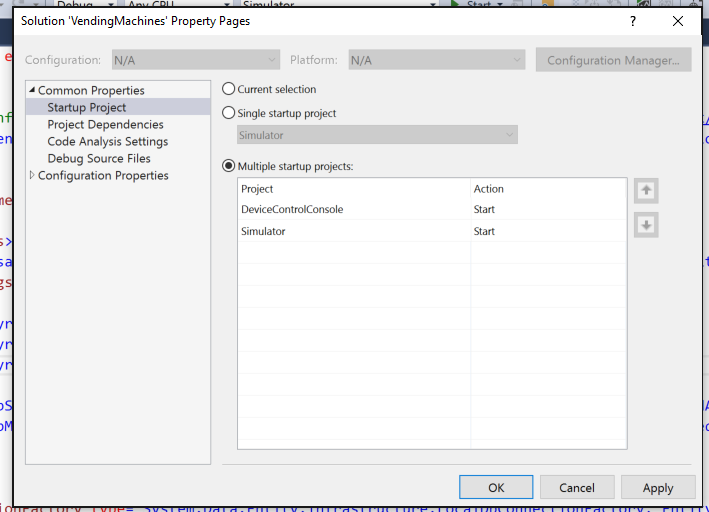
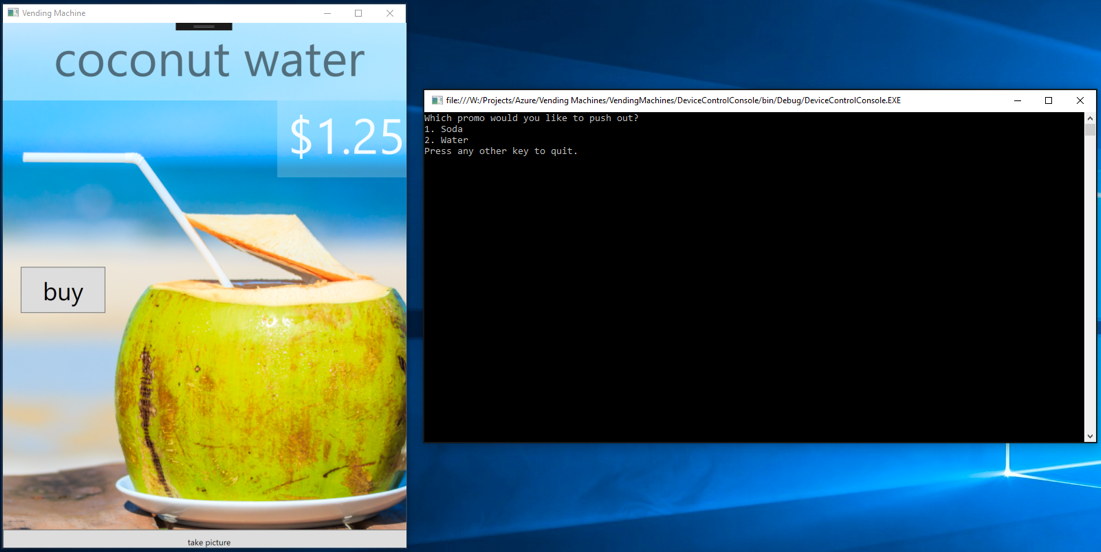
await \_serviceClient.SendAsync(deviceId, commandMessage);

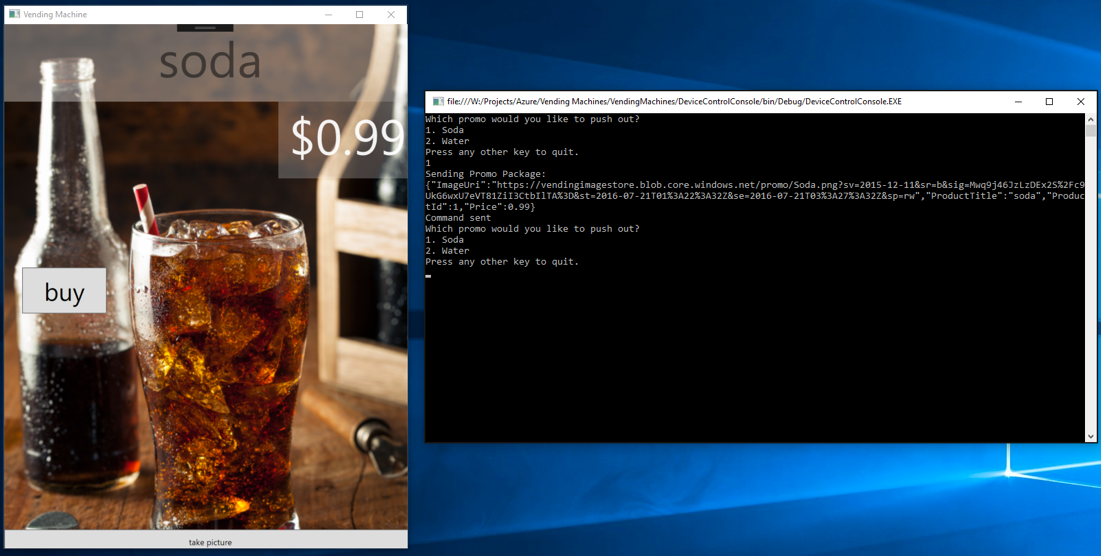
### Task 3: Configure the DeviceControlConsole and the Simulator

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. Save the file.
4. In Simulator, open App.config.

1. Set the IoTHubSenderConnectionString appSetting to have a value of the connection string for the **device** policy to your IoT Hub.
2. Set the IoTHubManagerConnectionString appSetting to have a value of the connection string for the **iothubowner** policy to your IoT Hub.
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. Build the Simulator and DeviceControlConsole projects.
6. In Solution Explorer, right-click Solution Vending Machines and select Set StartUp Projects.



1. In the dialog, select the Multiple startup projects option and ensure that Action is set to Start for both DeviceControlConsole and Simulator.  
   
2. Select OK.
3. From the Debug menu, choose Start without Debugging.
4. Wait for both the Vending Machine Simulator and the DeviceControlConsole to appear.  
   
5. In the DeviceControlConsole, press 1 to push the promotion for Soda.



1. Observe that the entire promotion surface of the vending machine changes (product name, price, and image).
2. Experiment sending the other promotion or toggling between promotions.
3. Experiment with making purchases and sending photos to verify the other functions still work with the new promoted products.

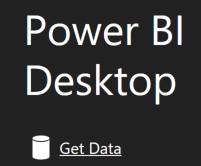
## Exercise 6: Analytics with Power BI Desktop

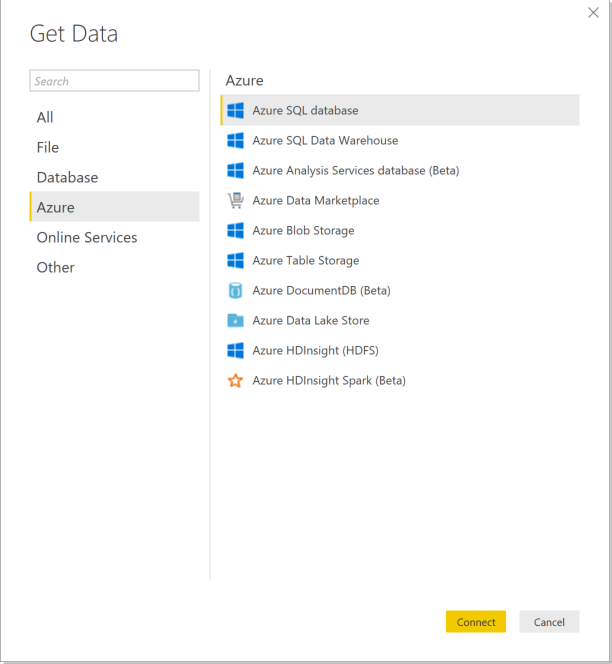
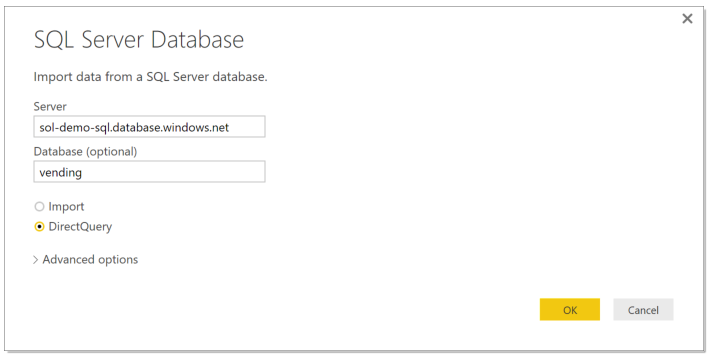
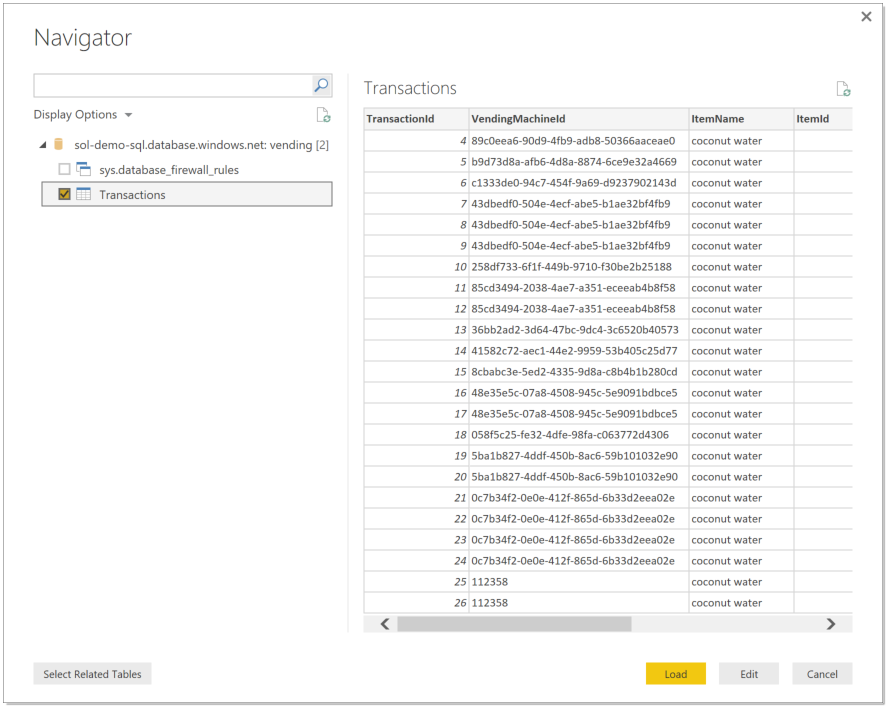
Duration: 15 minutes

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.

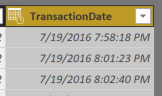
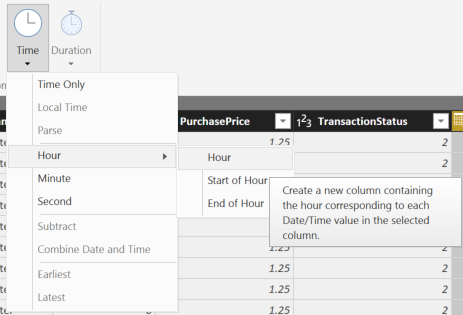
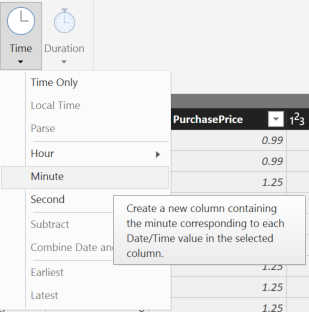
### Task 1: Build the query and create the visualization

1. From your Start menu, open Power BI Desktop.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%206.34.45%20PM.png
2. In the opening dialog, select Get Data.

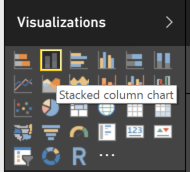


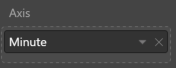
1. In the Get Data dialog, select Azure in the categories list and then Azure SQL Database.  
   
2. Select **Connect**.
3. In the dialog, enter the name of your SQL Server (e.g., myserver.database.windows.net), the name of your vending database, and select the DirectQuery option. Select OK.  
   
4. On the next screen, select the Database tab on the left.
5. Provide your SQL username and password and select Connect.
6. In the Navigator dialog, check the box next to Transactions.  
   
7. Select Load.
8. In the Ribbon, select Edit Queries.

../../../../../../Captures/Screen%20Shot%202016-07-20%20at%206.42.38%20PM.png

1. In the Query Editor, select the Transaction Date column header to select the column.  
   
2. In the Ribbon, select the Add Column tab and select Time, Hour, Hour.  
   
3. Select the TransactionDate column again.
4. In the Ribbon, select Time, Minute.  
   
5. Select the TransactionDate one more time.
6. In the Ribbon, select Time, Second.
7. In the Ribbon, on the Home tab, select Close & Apply.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%206.47.05%20PM.png
8. In the message that appears, select Apply Changes.

../../../../../../Captures/Screen%20Shot%202016-07-20%20at%206.48.12%20PM.png

1. In the Visualizations, select Stacked column chart.  
   
2. From the Fields list, drag the minute field over to the axis property.



1. From the Fields list, drag the PurchasePrice over to the value property.  
   ../../../../../../Captures/Screen%20Shot%202016-07-20%20at%206.51.18%20PM.png
2. Your completed visualization summarizing the most profitable minutes in each hour should appear as follows:

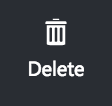


## Exercise 7: Cleanup

Duration: 5 minutes

In this exercise, attendees will de-provision any Azure resources that were created in support of the hackathon.

1. Delete the Resource Group you created.
   1. From the Portal, navigate to the blade of your Resource Group and select **Delete** in the **command bar** at the top.



1. When prompted to confirm the deletion, follow the prompts.