Contents

1 AZ-204: Developing solutions for Microsoft Azure

- **Direct students** to https://aka.ms/az204labs for an easy-to-use list of lab instructions.
- **Download Latest Student Handbook and AllFiles Content**
- **Are you a MCT?** - Have a look at our GitHub User Guide for MCTs
- **Need to manually build the lab instructions?** - Instructions are available in the MicrosoftLearning/Docker-Build repository

1.1 What are we doing?

- To support this course, we will need to make frequent updates to the course content to keep it current with the Azure services used in the course. We are publishing the lab instructions and lab files on GitHub to allow for open contributions between the course authors and MCTs to keep the content current with changes in the Azure platform.
- We hope that this brings a sense of collaboration to the labs like we've never had before - when Azure changes and you find it first during a live delivery, go ahead and make an enhancement right in the lab source. Help your fellow MCTs.

1.2 How should I use these files relative to the released MOC files?

- The instructor handbook and PowerPoints are still going to be your primary source for teaching the course content.
- These files on GitHub are designed to be used in conjunction with the student handbook, but are in GitHub as a central repository so MCTs and course authors can have a shared source for the latest lab files.
- It will be recommended that for every delivery, trainers check GitHub for any changes that may have been made to support the latest Azure services, and get the latest files for their delivery.

1.3 What about changes to the student handbook?

- We will review the student handbook on a quarterly basis and update through the normal MOC release channels as needed.

1.4 How do I contribute?

- Any MCT can submit a pull request to the code or content in the GitHub repro, Microsoft and the course author will triage and include content and lab code changes as needed.
- You can submit bugs, changes, improvement and ideas. Find a new Azure feature before we have? Submit a new demo!

1.5 Notes

1.5.1 Classroom Materials

1.6 It is strongly recommended that MCTs and Partners access these materials and in turn, provide them separately to students. Pointing students directly to GitHub to access Lab steps as part of an ongoing class will require them to access yet another UI as part of the course, contributing to a confusing experience for the student. An explanation to the student regarding why they are receiving separate Lab instructions can highlight the nature of an always-changing cloud-based interface and platform. Microsoft Learning support for accessing files on GitHub and support for navigation of the GitHub site is limited to MCTs teaching this course only.

1.7 title: Online Hosted Instructions permalink: index.html layout: home

1.8 Content Directory

Hyperlinks to each of the lab exercises and demos are listed below.

1.9 Labs

{% assign labs = site.pages | where_exp: "page", "page.url contains '/Instructions/Labs'" %} | Module | Lab | | --- | --- | {% for activity in labs %}{% if activity.lab.az204Module %}| {{ activity.lab.az204Module }} | [{{ activity.lab.az204Title }}{% if activity.lab.type %} - {{ activity.lab.type }}{% endif %}](/home/ll/Azure_clone/Azure_new/AZ-204-DevelopingSolutionsforMicrosoftAzure/{{ site.github.url }}{{ activity.url }}) | {% endif %}{% endfor %}

2 Lab Virtual Machine Setup

2.1 Installed Software

Software	Link
Windows 10 (Build 2004)	https://www.microsoft.com/software-download/windows10
Visual Studio Code	https://code.visualstudio.com
Visual Studio Code Azure Account Extension	https://marketplace.visualstudio.com/items?itemName=m
Visual Studio Code Azure Functions Extension	https://marketplace.visualstudio.com/items?itemName=m
Visual Studio Code Azure Resource Manager Tools Extension	https://marketplace.visualstudio.com/items?itemName=m
Visual Studio Code Azure CLI Tools Extension	https://marketplace.visualstudio.com/items?itemName=m
Visual Studio Code PowerShell Extension	https://marketplace.visualstudio.com/items?itemName=m
Visual Studio Code C# Extension	https://marketplace.visualstudio.com/items?itemName=m
PowerShell 7	https://github.com/PowerShell/PowerShell/releases/tag/v
.NET Core 3.1 SDK	https://dotnet.microsoft.com/download/dotnet-core/3.1
Azure PowerShell	https://docs.microsoft.com/powershell/azure/install-az-ps
Azure CLI	https://docs.microsoft.com/cli/azure/install-azure-cli
Azure Storage Explorer	https://azure.microsoft.com/features/storage-explorer
.NET Tool - HttpRepl	https://github.com/dotnet/HttpRepl
Azure Functions Core Tools	https://docs.microsoft.com/azure/azure-functions/function
Windows Terminal	https://aka.ms/terminal
Edge (Chromium)	https://www.microsoft.com/edge

2.2 Additional Configuration

- Enable ClearType
- Configure Microsoft Edge as the default browser
- Update VSCode configuration

```
{
  "editor.fontFamily": "'Cascadia Code', Consolas, 'Courier New', monospace",
```

```
  "update.enableWindowsBackgroundUpdates": false,
  "update.mode": "manual",
  "terminal.integrated.shell.windows": "C:\\Program Files\\PowerShell\\7\\pwsh.exe",
  "workbench.startupEditor": "none",
  "terminal.integrated.rendererType": "dom",
  "csharp.suppressDotnetInstallWarning": true,
  "csharp.suppressDotnetRestoreNotification": true,
  "csharp.supressBuildAssetsNotification": true,
  "azureFunctions.showProjectWarning": false
}
```

- Update Windows Terminal configuration

```
{
  "$schema": "https://aka.ms/terminal-profiles-schema",
  "defaultProfile": "{574e775e-4f2a-5b96-ac1e-a2962a402336}",
  "profiles": [
    {
      "guid": "{574e775e-4f2a-5b96-ac1e-a2962a402336}",
      "useAcrylic": true,
      "acrylicOpacity": 0.85,
      "colorScheme": "Campbell",
      "fontFace": "Cascadia Code",
      "hidden": false,
      "name": "PowerShell",
      "source": "Windows.Terminal.PowershellCore"
    },
    {
      "guid": "{b453ae62-4e3d-5e58-b989-0a998ec441b8}",
      "hidden": false,
      "name": "Azure Cloud Shell",
      "source": "Windows.Terminal.Azure"
    }
  ],
  "schemes": [],
  "keybindings": []
}
```

- Configure Start Menu & Taskbar to only include the following icons:
  - File Explorer
  - Edge
  - Windows Terminal
  - Visual Studio Code
  - Azure Storage Explorer
- Disable PowerShell 7 update notifications
  1. Create an environment variable named `POWERSHELL_UPDATECHECK`
  2. Set the value of the environment variable to `Off` (case-sensitive)
- Run Azure Functions Core Tools atleast once to configure Windows Firewall

```
func init test --worker-runtime dotnet
cd test
func new --template 'HTTP trigger' --name web
func start --build
```

2.3 lab: az204Title: 'Lab 01: Building a web application on Azure platform as a service offerings' az020Title: 'Lab 01: Building a web application on Azure platform as a service offerings' az204Module: 'Module 01: Creating Azure App Service Web Apps' az020Module: 'Module 01: Creating Azure App Service Web Apps' type: 'Answer Key'

3 Lab 01: Building a web application on Azure platform as a service offerings

4 Student lab answer key

4.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

4.2 Instructions

4.2.1 Before you start

4.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

4.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Windows PowerShell
- Visual Studio Code

4.2.2 Exercise 1: Build a back-end API by using Azure Storage and the Web Apps feature of Azure App Service

4.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. At the sign-in page, enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, a dialog box will display offering a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

4.2.2.2 Task 2: Create a Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. From the **All services** blade, select **Storage Accounts**.

3. From the **Storage accounts** blade, get your list of storage account instances.
4. From the **Storage accounts** blade, select **New**.
5. From the **Create storage account** blade, observe the tabs from the blade, such as **Basics**, **Tags**, and **Review + Create**.

   Note: Each tab represents a step in the workflow to create a new storage account. At any time, you can select **Review + Create** to skip the remaining tabs.
6. Select the **Basics** tab, and in the tab area, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **ManagedPlatform**, and then select **OK**.
   3. In the **Storage account name** text box, enter imgstor*[yourname]***.
   4. In the **Location** list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** list, select **Locally-redundant storage (LRS)**.
   8. Select **Review + Create**.
7. From the **Review + Create** tab, review the options that you specified in the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.
9. From the **Deployment** blade, wait for the creation task to complete before moving forward with this lab.
10. Select the **Go to resource** button from the **Deployment** blade to go to the newly created storage account.
11. From the **Storage account** blade, find the **Settings** section, and then select **Access keys**.
12. From the **Access keys** blade, select any one of the keys, and then record the value of either of the **Connection string** boxes. You'll use this value later in this lab.

    Note: It doesn't matter which connection string you choose. They are interchangeable.

4.2.2.3 Task 3: Upload a sample blob

1. In the Azure portal's navigation pane, select **Resource groups**.
2. From the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. From the **ManagedPlatform** blade, select the imgstor*[yourname]*** storage account that you created earlier in this lab.
4. From the **Storage Account** blade, in the **Blob service** section, select the **Containers** link.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** window, perform the following actions:
   1. In the **Name** text box, enter **images**.
   2. In the **Public access level** list, select **Blob (anonymous read access for blobs only)**, and then select **Create**.
7. In the **Containers** section, select the newly created **images** container.
8. From the **Container** blade, select **Upload**.
9. In the **Upload blob** window, perform the following actions:
   1. In the **Files** section, select the **Folder** icon.
   2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\01\Starter\Images**, select the **grilledcheese.jpg** file, and then select **Open**.

3. Ensure that the **Overwrite if files already exist** check box is selected, and then select **Upload**. Wait for the blob to upload before you continue with this lab.

4.2.2.4 Task 4: Create a web app

1. In the Azure portal's navigation pane, select **Create a resource**.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Web**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Web App** result.
5. From the **Web App** blade, select **Create**.
6. From the second **Web App** blade, find the tabs from the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new web app. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** drop-down list, select **ManagedPlatform**.
   3. In the **Name** text box, enter imgapi*[yourname]***.
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET Core 3.1 (LTS)**.
   6. In the **Operating System** section, select **Windows**.
   7. In the **Region** drop-down list, select the **East US** region.
   8. In the **Windows Plan (East US)** section, select **Create new**, enter the value **ManagedPlan** in the **Name** text box, and then select **OK**.
   9. Leave the **SKU and size** section set to its default value.
   10. Select **Next: Monitoring**.
8. From the **Monitoring** tab, perform the following actions:
   1. In the **Enable Application Insights** section, select **No**.
   2. Select **Review + Create**.
9. From the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the web app by using your specified configuration. Wait for the creation task to complete before you move forward with this lab.

4.2.2.5 Task 5: Configure the web app

1. In the Azure portal's navigation pane, select **Resource groups**.
2. From the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. From the **ManagedPlatform** blade, select the imgapi*[yourname]*** web app that you created earlier in this lab.
4. From the **Web App** blade, in the **Settings** section, select the **Configuration** link.
5. In the **Configuration** section, perform the following actions:
   1. Select the **Application settings** tab, and then select **New application setting**.
   2. In the **Add/Edit application setting** pop-up dialog, in the **Name** text box, enter **StorageConnectionString**.
   3. In the **Value** text box, enter the storage connection string that you copied earlier in this lab.

4. Leave the **Deployment slot setting** text box set to its default value, and then select **OK** to close the pop-up dialog and return to the **Configuration** section.
5. Select **Save** from the blade to persist your settings.

Wait for your application settings to persist before you move forward with the lab.

6. From the **Web App** blade in the **Settings** section, select the **Properties** link.
7. In the **Properties** section, copy the value of the **URL** text box. You'll use this value later in the lab.

> **Note**: At this point, the web server at this URL will return a 404 error. You have not deployed any code to the Web App yet. You will deploy code to the Web App later in this lab.

4.2.2.6 Task 6: Deploy an ASP.NET web application to Web Apps

1. On the taskbar, select the **Visual Studio Code** icon.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\01\Starter\API**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, expand the **Controllers** folder, and then select the **ImagesController.cs** file to open the file in the editor.
5. In the editor, in the **ImagesController** class on line 26, observe the **GetCloudBlobContainer** method and the code used to retrieve a container.
6. In the **ImagesController** class on line 36, observe the **Get** method and the code used to retrieve all blobs asynchronously from the **images** container.
7. In the **ImagesController** class on line 55, observe the **Post** method and the code used to persist an uploaded image to Storage.
8. On the taskbar, select the **Windows Terminal** icon.
9. At the open command prompt, enter the following command, and then select Enter to sign in to the Azure Command-Line Interface (CLI):

   ```
   az login
   ```

10. In the **Microsoft Edge** browser window, perform the following actions:
    1. Enter the email address for your Microsoft account, and then select **Next**.
    2. Enter the password for your Microsoft account, and then select **Sign in**.
11. Return to the currently open **Command Prompt** window. Wait for the sign-in process to finish.
12. At the command prompt, enter the following command, and then select Enter to list all the apps in your **ManagedPlatform** resource group:

    ```
    az webapp list --resource-group ManagedPlatform
    ```

13. Enter the following command, and then select Enter to find the apps that have the **imgapi*** prefix:

    ```
    az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgapi')]"
    ```

14. Enter the following command, and then select Enter to render only the name of the single app that has the **imgapi*** prefix:

    ```
    az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgapi')].{Name:name
    ```

15. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\01\Starter\API** directory that contains the lab files:

    ```
    cd F:\Allfiles\Labs\01\Starter\API\
    ```

16. Enter the following command, and then select Enter to deploy the **api.zip** file to the web app that you created earlier in this lab:

    ```
    az webapp deployment source config-zip --resource-group ManagedPlatform --src api.zip --name <name
    ```

Note: Replace the *<name-of-your-api-app>* placeholder with the name of the web app that you created earlier in this lab. You recently queried this app's name in the previous steps.

Wait for the deployment to complete before you move forward with this lab.

17. In the Azure portal's navigation pane, select the **Resource groups** link.
18. From the **Resource groups** blade, find and select the **ManagedPlatform** resource group that you created earlier in this lab.
19. From the **ManagedPlatform** blade, select the imgapi*[yourname]*** web app that you created earlier in this lab.
20. From the **Web App** blade, select **Browse**.
21. Perform a GET request to the root of the website, and then observe the JavaScript Object Notation (JSON) array that's returned. This array should contain the URL for your single uploaded image in your Storage account.
22. Return to your browser window with the Azure portal.
23. Close the currently running Visual Studio Code and Windows Terminal applications.

4.2.2.7 Review

In this exercise, you created a web app in Azure and then deployed your ASP.NET web application to Web Apps by using the Azure CLI and Apache Kudu zip file deployment utility.

4.2.3 Exercise 2: Build a front-end web application by using Azure Web Apps

4.2.3.1 Task 1: Create a web app

1. In the Azure portal's navigation pane, select **Create a resource**.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Web**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Web App** result.
5. From the **Web App** blade, select **Create**.
6. From the second **Web App** blade, find the tabs from the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new web app. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** drop-down list, select **ManagedPlatform**.
   3. In the **Name** text box, enter imgweb*[yourname]*.
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET Core 3.1 (LTS)**.
   6. In the **Operating System** section, select **Windows**.
   7. In the **Region** drop-down list, select the **East US** region.
   8. In the **Windows Plan (East US)** section, select **ManagedPlan (S1)**.
   9. Select **Next: Monitoring**.
8. From the **Monitoring** tab, perform the following actions:
   1. In the **Enable Application Insights** section, select **No**.
   2. Select **Review + Create**.
9. From the **Review + Create** tab, review the options that you selected during the previous steps.

10. Select **Create** to create the web app by using your specified configuration. Wait for the creation task to complete before you move forward with this lab.

4.2.3.2 Task 2: Configure a web app

1. In the Azure portal's navigation pane, select **Resource groups**.
2. From the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. From the **ManagedPlatform** blade, select the imgweb*[yourname]*** web app that you created earlier in this lab.
4. From the **Web App** blade, in the **Settings** section, select the **Configuration** link.
5. In the **Configuration** section, perform the following actions:
    1. Select the **Application settings** tab, and then select **New application setting**.
    2. In the **Add/Edit application setting** pop-up dialog, in the **Name** text box, enter **ApiUrl**.
    3. In the **Value** text box, enter the web app URL that you copied earlier in this lab.

        Note: Make sure you include the protocol, such as **https://**, in the URL that you copy into the **Value** text box for this application setting.

    4. Leave the **Deployment slot setting** text box set to its default value.
    5. Select **OK** to close the pop-up dialog, and then return to the **Configuration** section.
    6. Select **Save** from the blade to persist your settings.

    Wait for your application settings to persist before you move forward with the lab.

4.2.3.3 Task 3: Deploy an ASP.NET web application to Web Apps

1. On the taskbar, select the **Visual Studio Code** icon.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\01\Starter\Web**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, expand the **Pages** folder, and then select the **Index.cshtml.cs** file to open the file in the editor.
5. In the editor, in the **IndexModel** class on line 30, observe the **OnGetAsync** method and the code used to retrieve the list of images from the API.
6. In the **IndexModel** class on line 41, observe the **OnPostAsync** method and the code used to stream an uploaded image to the back-end API.
7. On the taskbar, select the **Windows Terminal** icon.
8. At the open command prompt, enter the following command, and then select Enter to sign in to the Azure CLI:

    ```
    az login
    ```

9. In the browser window, perform the following actions:
    1. Enter the email address for your Microsoft account, and then select **Next**.
    2. Enter the password for your Microsoft account, and then select **Sign in**.
10. Return to the currently open **Command Prompt** window. Wait for the sign-in process to finish.
11. Enter the following command, and then select Enter to list all the apps in your **ManagedPlatform** resource group:

    ```
    az webapp list --resource-group ManagedPlatform
    ```

12. Enter the following command, and then select Enter to find the apps that have the **imgweb*** prefix:

    ```
    az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgweb')]"
    ```

13. Enter the following command, and then select Enter to render only the name of the single app that has the **imgweb*** prefix:

    ```
    az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgweb')].{Name:name
    ```

14. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\01\Starter\Web** directory that contains the lab files:

    ```
    cd F:\Allfiles\Labs\01\Starter\Web\
    ```

15. Enter the following command, and then select Enter to deploy the **web.zip** file to the web app that you created earlier in this lab:

    ```
    az webapp deployment source config-zip --resource-group ManagedPlatform --src web.zip --name <name
    ```

    > **Note**: Replace the *<name-of-your-web-app>* placeholder with the name of the web app that you created earlier in this lab. You recently queried this app's name in the previous steps.

    Wait for the deployment to complete before you move forward with this lab.

16. In the Azure portal's navigation pane, select **Resource groups**.
17. From the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
18. From the **ManagedPlatform** blade, select the imgweb*[yourname]*** web app that you created earlier in this lab.
19. From the **Web App** blade, select **Browse**.
20. Observe the list of images in the gallery. The gallery should list a single image that was uploaded to Storage earlier in the lab.
21. From the **Contoso Photo Gallery** webpage, find the **Upload a new image** section, and then perform the following actions:
    1. Select **Browse**.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\01\Starter\Images**, select the **bahnmi.jpg** file, and then select **Open**.
    3. Select **Upload**.
22. Observe that the list of gallery images has updated with your new image.

    > **Note**: In some rare cases, you might need to refresh your browser window to retrieve the new image.

23. Return to your browser window with the Azure portal.
24. Close the currently running Visual Studio Code and Windows Terminal applications.

4.2.3.4 Review

In this exercise, you created an Azure web app and deployed an existing web application's code to the resource in the cloud.

4.2.4 Exercise 3: Clean up your subscription

4.2.4.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   > **Note**: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.
   2. Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab.

Note: If you don't notice the Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

4.2.4.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ManagedPlatform** resource group:

   ```
   az group delete --name ManagedPlatform --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

4.2.4.3 Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

4.2.4.4 Review

4.3 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

4.4 lab: az204Title: 'Lab 01: Building a web application on Azure platform as a service offerings' az020Title: 'Lab 01: Building a web application on Azure platform as a service offerings' az204Module: 'Module 01: Creating Azure App Service Web Apps' az020Module: 'Module 01: Creating Azure App Service Web Apps'

5 Lab 01: Building a web application on Azure platform as a service offerings

6 Student lab manual

6.1 Lab scenario

You're the owner of a startup organization and have been building an image gallery application for people to share great images of food. To get your product to market as quickly as possible, you decided to use Microsoft Azure App Service to host your web apps and APIs.

6.2 Objectives

After you complete this lab, you will be able to:

- Create various apps by using App Service.
- Configure application settings for an app.
- Deploy apps by using Kudu, the Azure Command-Line Interface (CLI), and zip file deployment.

6.3 Lab setup

- Estimated time: **45 minutes**

6.4 Instructions

6.4.1 Before you start

6.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

6.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Windows PowerShell
- Visual Studio Code

6.4.2 Exercise 1: Build a back-end API by using Azure Storage and the Web Apps feature of Azure App Service

6.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).

   Note: If this is your first time signing in to the Azure portal, a dialog box will display offering a tour of the portal. Select **Get Started** to skip the tour.

6.4.2.2 Task 2: Create a Storage account

1. Create a new storage account with the following details:
   - New resource group: **ManagedPlatform**
   - Name: imgstor*[yourname]***
   - Location: **(US) East US**
   - Performance: **Standard**
   - Account kind: **StorageV2 (general purpose v2)**
   - Replication: **Locally-redundant storage (LRS)**
2. Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.
3. Access the **Access Keys** blade of your newly created storage account instance.
4. Record the value of the **Connection string** text box. You'll use this value later in this lab.

6.4.2.3 Task 3: Upload a sample blob

1. Access the imgstor*[yourname]*** storage account that you created earlier in this lab.
2. In the **Blob service** section, select the **Containers** link.
3. Create a new **container** with the following settings:
   - Name: **images**
   - Public access level: **Blob (anonymous read access for blobs only)**
4. Go to the new **images** container, and then use the **Upload** button to upload the **grilledcheese.jpg** file in the **Allfiles (F):\Allfiles\Labs\01\Starter\Images** folder on your lab machine.

   Note: We recommended that you enable the **Overwrite if files already exist** option.

6.4.2.4 Task 4: Create a web app

1. Create a new web app with the following details:
   - Existing resource group: **ManagedPlatform**
   - Web App name: imgapi*[yourname]***
   - Publish: **Code**
   - Runtime stack: **.NET Core 3.1 (LTS)**

- Operating System: **Windows**
- Region: **East US**
- New App Service plan: **ManagedPlan**
- SKU and size: **Standard (S1)**
- Application Insights: **Disabled**

2. Wait for Azure to finish creating the web app before you move forward with the lab. You'll receive a notification when the app is created.

6.4.2.5 Task 5: Configure the web app

1. Access the imgapi*[yourname]*** web app that you created earlier in this lab.
2. In the **Settings** section, find the **Configuration** section, and then create a new application setting by using the following details:
   - Name: **StorageConnectionString**
   - Value: ***Storage Connection String copied earlier in this lab***
   - Deployment slot setting: **Not selected**
3. Save your changes to the application settings.
4. In the **Settings** section, find the **Properties** section.
5. In the **Properties** section, copy the value of the **URL** text box. You'll use this value later in the lab.

   Note: At this point, the web server at this URL will return a 404 error. You have not deployed any code to the Web App yet. You will deploy code to the Web App later in this lab.

6.4.2.6 Task 6: Deploy an ASP.NET web application to Web Apps

1. Using Visual Studio Code, open the web application in the **Allfiles (F):\Allfiles\Labs\01\Starter\API** folder.
2. Open the **Controllers\ImagesController.cs** file, and then observe the code in each of the methods.
3. Open the Windows Terminal application.
4. Sign in to the Azure CLI by using your Azure credentials:

   ```
   az login
   ```

5. List all the apps in your **ManagedPlatform** resource group:

   ```
   az webapp list --resource-group ManagedPlatform
   ```

6. Find the apps that have the **imgapi*** prefix:

   ```
   az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgapi')]"
   ```

7. Print only the name of the single app that has the **imgapi*** prefix:

   ```
   az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgapi')].{Name:name
   ```

8. Change the current directory to the **Allfiles (F):\Allfiles\Labs\01\Starter\API** directory that contains the lab files:

   ```
   cd F:\Allfiles\Labs\01\Starter\API\
   ```

9. Deploy the **api.zip** file to the web app that you created earlier in this lab:

   ```
   az webapp deployment source config-zip --resource-group ManagedPlatform --src api.zip --name <name
   ```

   Note: Replace the *<name-of-your-api-app>* placeholder with the name of the web app that you created earlier in this lab. You recently queried this app's name in the previous steps.

10. Access the imgapi*[yourname]*** web app that you created earlier in this lab. Open the imgapi*[yourname]*** web app in your browser.

11. Perform a GET request to the root of the website, and then observe the JavaScript Object Notation (JSON) array that's returned. This array should contain the URL for your single uploaded image in your storage account.
12. Close the currently running Visual Studio Code and Windows Terminal applications.

6.4.2.7 Review

In this exercise, you created a web app in Azure and then deployed your ASP.NET web application to Web Apps by using the Azure CLI and the Kudu zip file deployment utility.

6.4.3 Exercise 2: Build a front-end web application by using Azure Web Apps

6.4.3.1 Task 1: Create a web app

1. In the Azure portal, create a new web app with the following details:
   - Existing resource group: **ManagedPlatform**
   - Web app name: imgweb*[yourname]***
   - Publish: **Code**
   - Runtime stack: **.NET Core 3.1 (LTS)**
   - Operating system: **Windows**
   - Region: **East US**
   - Existing App Service plan: **ManagedPlan**
   - Application Insights: **Disabled**
2. Wait for Azure to finish creating the web app before you move forward with the lab. You'll receive a notification when the app is created.

6.4.3.2 Task 2: Configure a web app

1. Access the imgweb*[yourname]*** web app that you created in the previous task.
2. In the **Settings** section, find the **Configuration** settings.
3. Create a new application setting by using the following details:
   - Name: **ApiUrl**
   - Value: ***Web app URL copied earlier in this lab***
   - Deployment slot setting: **Not selected**

   Note: Make sure you include the protocol, such as **https://**, in the URL that you copy into the **Value** text box for this application setting.
4. Save your changes to the application settings.

6.4.3.3 Task 3: Deploy an ASP.NET web application to Web Apps

1. Using Visual Studio Code, open the web application in the **Allfiles (F):\Allfiles\Labs\01\Starter\Web** folder.
2. Open the **Pages\Index.cshtml.cs** file, and then observe the code in each of the methods.
3. Open the Windows Terminal application, and then sign in to the Azure CLI by using your Azure credentials:

   ```
   az login
   ```
4. List all the apps in your **ManagedPlatform** resource group:

   ```
   az webapp list --resource-group ManagedPlatform
   ```
5. Find the apps that have the **imgweb*** prefix:

   ```
   az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgweb')]"
   ```

6. Print only the name of the single app that has the **imgweb*** prefix:

   ```
   az webapp list --resource-group ManagedPlatform --query "[?starts_with(name, 'imgweb')].{Name:name
   ```

7. Change your current directory to the **Allfiles (F):\Allfiles\Labs\01\Starter\Web** directory that contains the lab files:

   ```
   cd F:\Allfiles\Labs\01\Starter\Web\
   ```

8. Deploy the **web.zip** file to the web app that you created earlier in this lab:

   ```
   az webapp deployment source config-zip --resource-group ManagedPlatform --src web.zip --name <name
   ```

   Note: Replace the *<name-of-your-web-app>* placeholder with the name of the web app that you created earlier in this lab. You recently queried this app's name in the previous steps.

9. Access the imgweb*[yourname]*** web app that you created earlier in this lab. Open the imgweb*[yourname]*** web app in your browser.

10. From the **Contoso Photo Gallery** webpage, find the **Upload a new image** section, and then upload the **bahnmi.jpg** file in the **Allfiles (F):\Allfiles\Labs\01\Starter\Images** folder on your lab machine.

    Note: Ensure you click the **Upload** button to upload the image to Azure.

11. Observe that the list of gallery images has updated with your new image.

    Note: In some rare cases, you might need to refresh your browser window to retrieve the new image.

12. Close the currently running Visual Studio Code and Windows Terminal applications.

6.4.3.4 Review

In this exercise, you created an Azure web app and deployed an existing web application's code to the resource in the cloud.

6.4.4 Exercise 3: Clean up your subscription

6.4.4.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

6.4.4.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ManagedPlatform** resource group:

   ```
   az group delete --name ManagedPlatform --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

6.4.4.3 Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

6.4.4.4 Review

6.5 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

6.6 lab: az204Title: 'Lab 02: Implement task processing logic by using Azure Functions' az020Title: 'Lab 02: Implement task processing logic by using Azure Functions' az204Module: 'Module 02: Implement Azure Functions' az020Module: 'Module 02: Implement Azure Functions' type: 'Answer Key'

7 Lab 02: Implement task processing logic by using Azure Functions

8 Student lab answer key

8.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure user interface (UI) changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

8.2 Instructions

8.2.1 Before you start

8.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

8.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Windows Terminal
- Visual Studio Code

8.2.2 Exercise 1: Create Azure resources

8.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. If you prefer to skip the tour, select **Get Started** to begin using the portal.

8.2.2.2 Task 2: Create an Azure Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. On the **All services** blade, select **Storage Accounts**.
3. On the **Storage accounts** blade, get your list of storage account instances.
4. On the **Storage accounts** blade, select **New**.
5. On the **Create storage account** blade, observe the tabs on the blade, such as **Basics**, **Tags**, and **Review + Create**.

Note: Each tab represents a step in the workflow to create a new storage account. You can select **Review + Create** at any time to skip the remaining tabs.

6. Select the **Basics** tab, and then in the tab area, perform the following actions:
    1. Leave the **Subscription** text box set to its default value.
    2. In the **Resource group** section, select **Create new**, enter **Serverless**, and then select **OK**.
    3. In the **Storage account name** text box, enter **funcstor[yourname]**.
    4. In the **Location** list, select the **(US) East US** region.
    5. In the **Performance** section, select **Standard**.
    6. In the **Account kind** list, select **StorageV2 (general purpose v2)**.
    7. In the **Replication** list, select **Locally-redundant storage (LRS)**.
    8. Select **Review + Create**.
7. On the **Review + Create** tab, review the options that you specified in the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.
    Note: On the **Deployment** blade, wait for the creation task to complete before moving forward with this lab.
9. In the Azure portal's navigation pane, select **All services**.
10. On the **All services** blade, select **Storage Accounts**.
11. On the **Storage accounts** blade, select the **funcstor[yourname]** storage account instance.
12. From the **Storage account** blade, find the **Settings** section, and then select **Access keys**.
13. From the **Access keys** blade, select any one of the keys, and then record the value of either of the **Connection string** boxes.
    Note: You'll use this value later in the lab. It doesn't matter which connection string you choose. They are interchangeable.

8.2.2.3 Task 3: Create a Function app

1. In the Azure portal's navigation pane, select the **Create a resource** link.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Function**, and then select Enter.
4. On the **Everything** search results blade, select the **Function App** result.
5. On the **Function App** blade, select **Create**.
6. Find the tabs on the **Function App** blade, such as **Basics**.
    Note: Each tab represents a step in the workflow to create a new function app. You can select **Review + Create** at any time to skip the remaining tabs.
7. On the **Basics** tab, perform the following actions:
    1. Leave the **Subscription** text box set to its default value.
    2. In the **Resource group** section, select **Use existing**, and then select **Serverless** in the list.
    3. In the **Function app name** text box, enter **funclogic[yourname]**.
    4. In the **Publish** section, select **Code**.
    5. In the **Runtime stack** drop-down list, select **.NET**.
    6. In the **Version** drop-down list, select **3.1**.
    7. In the **Region** drop-down list, select the **East US** region.
    8. Select **Next: Hosting**.
8. On the **Hosting** tab, perform the following actions:
    1. In the **Operating System** section, select **Linux**.
    2. In the **Storage account** drop-down list, select the **funcstor[yourname]** storage account that you created earlier in this lab.
    3. In the **Plan type** drop-down list, select the **Consumption** option.
    4. Select **Review + Create**.
9. On the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the function app by using your specified configuration.
    Note: Wait for the creation task to complete before you move forward with this lab.

Review: In this exercise, you created all the resources that you'll use for this lab.

8.2.3 Exercise 2: Configure a local Azure Functions project

8.2.3.1 Task 1: Initialize a function project

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

```
cd F:\Allfiles\Labs\02\Starter\func
```

3. When you receive the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new local Azure Functions project in the current directory using the **dotnet** runtime:

   ```
   func init --worker-runtime dotnet --force
   ```

   > **Note**: You can review the documentation to [create a new project][azure-functions-core-tools-new-project] using the **Azure Functions Core Tools**.

4. Close the currently running **Windows Terminal** application.

8.2.3.2 Task 2: Configure connection string

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\02\Starter\func**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, open the **local.settings.json** file.
5. Observe the current value of the **AzureWebJobsStorage** setting:

   ```
   "AzureWebJobsStorage": "UseDevelopmentStorage=true",
   ```

6. Update the value of the **AzureWebJobsStorage** by setting it to the **connection string** of the storage account that you recorded earlier in this lab.

8.2.3.3 Task 3: Build and validate a project

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. When you receive the open command prompt, enter the following command, and then select Enter to **build** the .NET Core 3.1 project:

   ```
   dotnet build
   ```

   Review: In this exercise, you created a local project that you'll use for Azure Functions development.

8.2.4 Exercise 3: Create a function that's triggered by an HTTP request

8.2.4.1 Task 1: Create an HTTP-triggered function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. When you receive the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new function named **Echo** using the **HTTP trigger** template:

   ```
   func new --template "HTTP trigger" --name "Echo"
   ```

   > **Note**: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.

4. Close the currently running **Windows Terminal** application.

8.2.4.2 Task 2: Write HTTP-triggered function code

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\02\Starter\func**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, open the **Echo.cs** file.
5. In the code editor, observe the example implementation:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace func
{
    public static class Echo
    {
        [FunctionName("Echo")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest re
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the query st
                : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);
        }
    }
}
```

6. Delete all the content within the **Echo.cs** file.
7. Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Mvc**, **Microsoft.Azure.WebJobs**, **Microsoft.AspNetCore.Http**, and **Microsoft.Extensions.Logging** namespaces:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
```

8. Create a new **public static** class named **Echo**:

```
public static class Echo
{ }
```

9. Observe the **Echo.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;

public static class Echo
{ }
```

10. Within the **Echo** class, add the following code block to create a new **public static** method named **Run** that returns a variable of type **IActionResult** and that also takes in variables of type **HttpRequest** and **ILogger** as parameters named *request* and *logger*:

```
public static IActionResult Run(
    HttpRequest request,
    ILogger logger)
{ }
```

11. Add the following code to append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **Echo**:

```
[FunctionName("Echo")]
public static IActionResult Run(
    HttpRequest request,
    ILogger logger)
{ }
```

12. Add the following code to append an attribute to the **request** parameter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **POST**:

```
[FunctionName("Echo")]
public static IActionResult Run(
    [HttpTrigger("POST")] HttpRequest request,
    ILogger logger)
{ }
```

13. Observe the **Echo.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;

public static class Echo
{
    [FunctionName("Echo")]
    public static IActionResult Run(
        [HttpTrigger("POST")] HttpRequest request,
        ILogger logger)
    { }
}
```

14. In the **Run** method, enter the following line of code to log a fixed message:

```
logger.LogInformation("Received a request");
```

15. Enter the following line of code to echo the body of the HTTP request as the HTTP response:

```
return new OkObjectResult(request.Body);
```

16. Observe the **Echo.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;

public static class Echo
{
```

```
        [FunctionName("Echo")]
        public static IActionResult Run(
            [HttpTrigger("POST")] HttpRequest request,
            ILogger logger)
        {
            logger.LogInformation("Received a request");
            return new OkObjectResult(request.Body);
        }
    }
```

17. Select **Save** to save your changes to the **Echo.cs** file.

8.2.4.3 Task 3: Test the HTTP-triggered function by using httprepl

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

   ```
   func start --build
   ```

   > **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. On the taskbar, select the **Windows Terminal** icon again to open a new instance of the **Windows Terminal** application.
5. When you receive the open command prompt, enter the following command, and then select Enter to start the **httprepl** tool setting the base Uniform Resource Identifier (URI) to `http://localhost:7071`:

   ```
   httprepl http://localhost:7071
   ```

   > **Note**: An error message is displayed by the httprepl tool. This message occurs because the tool is searching for a Swagger definition file to use to ”traverse” the API. Because your functio projectp does not produce a Swagger definition file, you’ll need to traverse the API manually.

6. When you receive the tool prompt, enter the following command, and then select Enter to browse to the relative **api** directory:

   ```
   cd api
   ```

7. Enter the following command, and then select Enter to browse to the relative **echo** directory:

   ```
   cd echo
   ```

8. Enter the following command, and then select Enter to run the **post** command sending in an HTTP request body set to a numeric value of **3** by using the **--content** option:

   ```
   post --content 3
   ```

9. Enter the following command, and then select Enter to run the **post** command sending in an HTTP request body set to a numeric value of **5** by using the **--content** option:

   ```
   post --content 5
   ```

10. Enter the following command, and then select Enter to run the **post** command sending in an HTTP request body set to a string value of **Hello** by using the **--content** option:

    ```
    post --content "Hello"
    ```

11. Enter the following command, and then select Enter to run the **post** command sending in an HTTP request body set to a JavaScript Object Notation (JSON) value of **{”msg”: ”Successful”}** by using the **--content** option:

    ```
    post --content "{"msg": "Successful"}"
    ```

12. Enter the following command, and then select Enter to exit the **httprepl** application:

```
exit
```

13. Close all currently running instances of the **Windows Terminal** application.

    Review: In this exercise, you created a basic function that echoes the content sent via an HTTP POST request.

8.2.5 Exercise 4: Create a function that triggers on a schedule

8.2.5.1 Task 1: Create a schedule-triggered function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. When you receive the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new function named **Recurring** using the **Timer trigger** template:

   ```
   func new --template "Timer trigger" --name "Recurring"
   ```

   > **Note**: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.

4. Close the currently running **Windows Terminal** application.

8.2.5.2 Task 2: Observe function code

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\02\Starter\func**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, open the **Recurring.cs** file.
5. In the code editor, observe the implementation:

```
using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;

namespace func
{
    public static class Recurring
    {
        [FunctionName("Recurring")]
        public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
        {
            log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
        }
    }
}
```

8.2.5.3 Task 3: Observe function runs

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start --build
```

> **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. Observe the function run that occurs about every five minutes. Each function run should render a simple message to the log.
5. Close the currently running **Windows Terminal** application.

8.2.5.4 Task 4: Update the function integration configuration

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\02\Starter\func**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, open the **Recurring.cs** file.
5. In the code editor, observe the existing **Run** method signature:

```
[FunctionName("Recurring")]
public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
```

6. Update the **Run** method signature code block to change the schedule to execute once every **30 seconds**:

```
[FunctionName("Recurring")]
public static void Run([TimerTrigger("*/30 * * * * *")]TimerInfo myTimer, ILogger log)
```

7. Select **Save** to save your changes to the **Recurring.cs** file.

8.2.5.5 Task 5: Observe function runs

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

```
cd F:\Allfiles\Labs\02\Starter\func
```

3. When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start --build
```

> **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. Observe the function run that occurs about every 30 seconds. Each function run should render a simple message to the log.
5. Close the currently running **Windows Terminal** application.

Review: In this exercise, you created a function that runs automatically based on a fixed schedule.

8.2.6 Exercise 5: Create a function that integrates with other services

8.2.6.1 Task 1: Upload sample content to Azure Blob Storage

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
3. On the **Serverless** blade, select the **funcstor[yourname]** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up window, perform the following actions:
    1. In the **Name** text box, enter **content**.
    2. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
    3. Select **OK**.

7. Return to the **Containers** section, and then select the recently created **content** container.
8. On the **Container** blade, select **Upload**.
9. In the **Upload blob** window, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\02\Starter**, select the **settings.json** file, and then select **Open**.
    3. Ensure that the **Overwrite if files already exist** check box is selected, and then select **Upload**. **Note**: Wait for the blob to upload before you continue with this lab.

8.2.6.2 Task 2: Create a HTTP-triggered function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

    ```
    cd F:\Allfiles\Labs\02\Starter\func
    ```

3. When you receive the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new function named **GetSettingInfo** using the **HTTP trigger** template:

    ```
    func new --template "HTTP trigger" --name "GetSettingInfo"
    ```

    Note: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.

4. Close the currently running **Windows Terminal** application.

8.2.6.3 Task 3: Write HTTP-triggered and blob-inputted function code

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\02\Starter\func**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, open the **GetSettingInfo.cs** file.
5. In the code editor, observe the example implementation:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace func
{
    public static class GetSettingInfo
    {
        [FunctionName("GetSettingInfo")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest re
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
```

```
            name = name ?? data?.name;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the query st
                : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);
        }
    }
}
```

6. Delete all the content within the **GetSettingInfo.cs** file.

7. Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Http**, **Microsoft.AspNetCore.Mvc**, and **Microsoft.Azure.WebJobs** namespaces:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
```

8. Create a new **public static** class named **GetSettingInfo**:

```
public static class GetSettingInfo
{ }
```

9. Observe the **GetSettingInfo.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;

public static class GetSettingInfo
{ }
```

10. Within the **GetSettingInfo** class, add the following code block to create a new **public static** expression-bodied method named **Run** that returns a variable of type **IActionResult** and that also takes in variables of type **HttpRequest** and **string** as parameters named *request* and *json*:

```
public static IActionResult Run(
    HttpRequest request,
    string json)
    => null;
```

    Note: You are only temporarily setting the return value to **null**.

11. Add the following code to append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **GetSettingInfo**:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    HttpRequest request,
    string json)
    => null;
```

12. Add the following code to append an attribute to the **request** parameter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **GET**:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    [HttpTrigger("GET")] HttpRequest request,
    string json)
    => null;
```

13. Add the following code to append an attribute to the **json** parameter of type **BlobAttribute** that has its **blobPath** parameter set to a value of **content/settings.json**:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    [HttpTrigger("GET")] HttpRequest request,
```

```
    [Blob("content/settings.json")] string json)
    => null;
```

14. Add the following code to update the **Run** expression-bodied method to return a new instance of the **OkObjectResult** class passing in the value of the **json** method parameter as the sole constructor parameter:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    [HttpTrigger("GET")] HttpRequest request,
    [Blob("content/settings.json")] string json)
    => new OkObjectResult(json);
```

15. Observe the **GetSettingInfo.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;

public static class GetSettingInfo
{
    [FunctionName("GetSettingInfo")]
    public static IActionResult Run(
        [HttpTrigger("GET")] HttpRequest request,
        [Blob("content/settings.json")] string json)
        => new OkObjectResult(json);
}
```

16. Select **Save** to save your changes to the **GetSettingInfo.cs** file.

8.2.6.4 Task 4: Register Azure Storage blob extensions

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

```
cd F:\Allfiles\Labs\02\Starter\func
```

3. When you receive the open command prompt, enter the following command, and then select Enter to **register** the Microsoft.Azure.WebJobs.Extensions.Storage extension:

```
func extensions install --package Microsoft.Azure.WebJobs.Extensions.Storage --version 4.0.4
```

4. Enter the following command, and then select Enter to validate the extensions were installed correctly by **building** the .NET project:

```
dotnet build
```

5. Close all currently running instances of the **Windows Terminal** application.

8.2.6.5 Task 5: Test the function by using httprepl

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

```
cd F:\Allfiles\Labs\02\Starter\func
```

3. When you receive the open command prompt, enter the following command, and then select Enter to run the function app project:

```
func start --build
```

> **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. On the taskbar, select the **Windows Terminal** icon again to open a new instance of the **Windows Terminal** application.

5. When you receive the open command prompt, enter the following command, and then select Enter to start the **httprepl** tool setting the base Uniform Resource Identifier (URI) to `http://localhost:7071`:

   ```
   httprepl http://localhost:7071
   ```

   > **Note**: An error message is displayed by the httprepl tool. This message occurs because the tool is searching for a Swagger definition file to use to "traverse" the API. Because your functio project does not produce a Swagger definition file, you'll need to traverse the API manually.

6. When you receive the tool prompt, enter the following command, and then select Enter to browse to the relative **api** endpoint:

   ```
   cd api
   ```

7. Enter the following command, and then select Enter to browse to the relative **getsettinginfo** endpoint:

   ```
   cd getsettinginfo
   ```

8. Enter the following command, and then select Enter to run the **get** command for the current endpoint:

   ```
   get
   ```

9. Observe the JSON content of the response from the function app, which should now include:

   ```
   {
       "version": "0.2.4",
       "root": "/usr/libexec/mews_principal/",
       "device": {
           "id": "21e46d2b2b926cba031a23c6919"
       },
       "notifications": {
           "email": "joseph.price@contoso.com",
           "phone": "(425) 555-0162 x4151"
       }
   }
   ```

10. Enter the following command, and then select Enter to exit the **httprepl** application:

    ```
    exit
    ```

11. Close all currently running instances of the **Windows Terminal** application.

    Review: In this exercise, you created a function that returns the content of a JSON file in Storage.

8.2.7 Exercise 6: Deploy a local function project to an Azure Functions app

8.2.7.1 Task 1: Deploy using the Azure Functions Core Tools

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. When you receive the open command prompt, enter the following command, and then select Enter to login to the Azure Command-Line Interface (CLI):

   ```
   az login
   ```

4. In the **Microsoft Edge** browser window, perform the following actions:
   1. Enter the email address for your Microsoft account, and then select **Next**.
   2. Enter the password for your Microsoft account, and then select **Sign in**.
5. Return to the currently open **Windows Terminal** window. Wait for the sign-in process to finish.
6. Enter the following command, and then select Enter to publish the function app project:

   ```
   func azure functionapp publish <function-app-name>
   ```

Note: For example, if your **Function App name** is **funclogicstudent**, your command would be `func azure functionapp publish funclogicstudent`. You can review the documentation to [publish the local function app project][azure-functions-core-tools-publish-azure] using the **Azure Functions Core Tools**.

7. Wait for the deployment to finalize before you move forward with the lab.
8. Close the currently running **Windows Terminal** application.

8.2.7.2 Task 2: Validate deployment

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. In the Azure portal's navigation pane, select the **Resource groups** link.
4. On the **Resource groups** blade, find and then select the **Serverless** resource group that you created earlier in this lab.
5. On the **Serverless** blade, select the **funclogic[yourname]** function app that you created earlier in this lab.
6. From the **App Service** blade, select the **Functions** option from the **Functions** section.
7. In the **Functions** pane, select the the existing **GetSettingInfo** function.
8. In the **Function** blade, select the **Code + Test** option from the **Developer** section.
9. In the function editor, select **Test/Run**.
10. In the popup dialog that appears, perform the following actions:
    - In the **HTTP method** list, select **GET**.
11. Select **Run** to test the function.
12. Observe the results of the test run. The JSON content should now include:

```
{
    "version": "0.2.4",
    "root": "/usr/libexec/mews_principal/",
    "device": {
        "id": "21e46d2b2b926cba031a23c6919"
    },
    "notifications": {
        "email": "joseph.price@contoso.com",
        "phone": "(425) 555-0162 x4151"
    }
}
```

Review: In this exercise, you deployed a local function project to Azure Functions and validated that the functions work in Azure.

8.2.8 Exercise 7: Clean up your subscription

8.2.8.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).
2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.
      Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If Cloud Shell configuration options don't display, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

8.2.8.2 Task 2: Delete a resource group

1. When you receive the command prompt, enter the following command, and then select Enter to delete the **Serverless** resource group:

   ```
   az group delete --name Serverless --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

8.2.8.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.

   Review: In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

8.3 [azure-functions-core-tools-new-function]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#create-func [azure-functions-core-tools-new-project]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#create-a-local-functions-project [azure-functions-core-tools-start-function]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#start [azure-functions-core-tools-publish-azure]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#publish

8.4 lab: az204Title: 'Lab 02: Implement task processing logic by using Azure Functions' az020Title: 'Lab 02: Implement task processing logic by using Azure Functions' az204Module: 'Module 02: Implement Azure Functions' az020Module: 'Module 02: Implement Azure Functions'

9 Lab 02: Implement task processing logic by using Azure Functions

10 Student lab manual

10.1 Lab scenario

Your company has built a desktop software tool that parses a local JavaScript Object Notation (JSON) file for its configuration settings. During its latest meeting, your team decided to reduce the number of files that are distributed with your application by serving your default configuration settings from a URL instead of from a local file. As the new developer on the team, you've been tasked with evaluating Microsoft Azure Functions as a solution to this problem.

10.2 Objectives

After you complete this lab, you'll be able to:

- Create an Azure Functions app in the Azure Portal.
- Create a local Azure Functions project using the [Azure Functions Core Tools][azure-functions-core-tools].
- Create various functions by using built-in triggers and input integrations.
- Deploy a local Azure Functions project to Azure.

10.3 Lab setup

- Estimated time: **45 minutes**

10.4 Instructions

10.4.1 Before you start

10.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

10.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Windows Terminal
- Visual Studio Code

10.4.2 Exercise 1: Create Azure resources

10.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, a dialog box offering a tour of the portal will appear. If you prefer to skip the tour, select **Get Started**.

10.4.2.2 Task 2: Create an Azure Storage account

1. Create a new storage account with the following details:
   - New resource group: **Serverless**
   - Name: **funcstor[yourname]**
   - Location: **(US) East US**
   - Performance: **Standard**
   - Account kind: **StorageV2 (general purpose v2)**
   - Replication: **Locally-redundant storage (LRS)**

   Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.
2. Open the **Access Keys** section of your newly created storage account instance.
3. Record the value of the **Connection string** text box.

   Note: You'll use this value later in the lab. It doesn't matter which connection string you choose. They are interchangeable.

10.4.2.3 Task 3: Create a function app

1. Create a new function app with the following details:
   - Existing resource group: **Serverless**
   - App name: **funclogic[yourname]**
   - Publish: **Code**
   - Runtime stack: **.NET**
   - Version: **3.1**
   - Region: **East US**
   - Operating system: **Linux**
   - Storage account: **funcstor[yourname]**
   - Plan: **Consumption**
   - Enable Application Insights: **Yes**

   Note: Wait for Azure to finish creating the function app before you move forward with the lab. You'll receive a notification when the app is created.

Review: In this exercise, you created all the resources that you'll use for this lab.

10.4.3 Exercise 2: Configure local Azure Functions project

10.4.3.1 Task 1: Initialize function project

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\02\Starter\func
   ```

3. Use the **Azure Functions Core Tools** to create a new local Azure Functions project with the following details:
   - worker runtime: **dotnet**

Note: You can review the documentation to [create a new project][azure-functions-core-tools-new-project] using the **Azure Functions Core Tools**.

4. Close the currently running **Windows Terminal** application.

10.4.3.2 Task 2: Configure connection string

1. Open **Visual Studio Code**.
2. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\02\Starter\func**.
3. Open the **local.settings.json** file.
4. Update the **AzureWebJobsStorage** setting by setting its value to the **connection string** of the storage account that you recorded earlier in this lab.

10.4.3.3 Task 3: Build and validate project

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. **Build** the .NET Core 3.1 project:

   ```
   dotnet build
   ```

4. Close the currently running **Windows Terminal** application.

   Review: In this exercise, you created a local project that you'll use for Azure Functions development.

10.4.4 Exercise 3: Create a function that's triggered by an HTTP request

10.4.4.1 Task 1: Create an HTTP-triggered function

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Create a new function with the following details:
   - template: **HTTP trigger**
   - name: **Echo**
     Note: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.
4. Close the currently running **Windows Terminal** application.

10.4.4.2 Task 2: Write HTTP-triggered function code

1. Open **Visual Studio Code**.
2. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\02\Starter\func**.
3. Open the **Echo.cs** file.
4. In the code editor, delete all the code within the **Echo.cs** file.
5. Add **using directives** for the **Microsoft.AspNetCore.Mvc**, **Microsoft.Azure.WebJobs**, **Microsoft.AspNetCore.Http**, and **Microsoft.Extensions.Logging** namespaces for libraries that will be referenced by the application:

   ```
   using Microsoft.AspNetCore.Mvc;
   using Microsoft.Azure.WebJobs;
   using Microsoft.AspNetCore.Http;
   using Microsoft.Extensions.Logging;
   ```

6. Create a new **public static** class named **Echo**:

   ```
   public static class Echo
   { }
   ```

7. Within the **Echo** class, create a new **public static** method named **Run** that returns a variable of type **IActionResult** and that also takes in variables of type **HttpRequest** and **ILogger** as parameters named *request* and *logger*:

```
public static IActionResult Run(
    HttpRequest request,
    ILogger logger)
{ }
```

8. Append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **Echo**:

```
[FunctionName("Echo")]
public static IActionResult Run(
    HttpRequest request,
    ILogger logger)
{ }
```

9. Append an attribute to the **request** parameter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **POST**:

```
[FunctionName("Echo")]
public static IActionResult Run(
    [HttpTrigger("POST")] HttpRequest request,
    ILogger logger)
{ }
```

10. Within the **Run** method, log a fixed message:

```
logger.LogInformation("Received a request");
```

11. Finally, echo the body of the HTTP request as the HTTP response:

```
return new OkObjectResult(request.Body);
```

12. **Save** the **Echo.cs** file.

10.4.4.3 Task 3: Test the HTTP-triggered function by using httprepl

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Start the function app project:

   > **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. Open a new instance of the **Windows Terminal** application.
5. Start the **httprepl** tool, and then set the base Uniform Resource Identifier (URI) to `http://localhost:7071`:

```
httprepl http://localhost:7071
```

   > **Note**: An error message is displayed by the httprepl tool. This message occurs because the tool is searching for a Swagger definition file to use to ”traverse” the API. Because your functio projectp does not produce a Swagger definition file, you’ll need to traverse the API manually.

6. When you receive the tool prompt, browse to the relative **api/echo** directory:

```
cd api
cd echo
```

7. Run the **post** command sending in an HTTP request body set to a numeric value of **3** by using the **--content** option:

```
post --content 3
```

8. Run the **post** command sending in an HTTP request body set to a numeric value of **5** by using the **--content** option:

```
post --content 5
```

9. Run the **post** command sending in an HTTP request body set to a string value of **”Hello”** by using the **--content** option:

```
post --content "Hello"
```

10. Run the **post** command sending in an HTTP request body set to a JSON value of **{"msg": "Successful"}** by using the **--content** option:

    ```
    post --content "{"msg": "Successful"}"
    ```

11. Exit the **httprepl** application:

    ```
    exit
    ```

12. Close all currently running instances of the **Windows Terminal** application.

    Review: In this exercise, you created a basic function that echoes the content sent via an HTTP POST request.

10.4.5 Exercise 4: Create a function that triggers on a schedule

10.4.5.1 Task 1: Create a schedule-triggered function

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Create a new function with the following details:
   - template: **Timer trigger**
   - name: **Recurring**
     Note: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.
4. Close the currently running **Windows Terminal** application.

10.4.5.2 Task 2: Observe function code

1. Open **Visual Studio Code**.
2. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\02\Starter\func**.
3. Open the **Recurring.cs** file.
4. In the code editor, observe the implementation:

```
using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;

namespace func
{
    public static class Recurring
    {
        [FunctionName("Recurring")]
        public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
        {
            log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
        }
    }
}
```

10.4.5.3 Task 3: Observe function runs

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Start the function app project:
   Note: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.
4. Observe the function run that occurs about every five minutes. Each function run should render a simple message to the log.
5. Close the currently running **Windows Terminal** application.

10.4.5.4 Task 4: Update the function integration configuration

1. Open **Visual Studio Code**.
2. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\02\Starter\func**.
3. Open the **Recurring.cs** file.
4. In the code editor, update the **Run** method signature to change the schedule to execute once every **30 seconds**:

```
[FunctionName("Recurring")]
public static void Run([TimerTrigger("*/30 * * * * *")]TimerInfo myTimer, ILogger log)
```

5. **Save** the **Recurring.cs** file.

10.4.5.5 Task 5: Observe function runs

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Start the function app project:
   Note: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.
4. Observe the function run that occurs about every thirty seconds. Each function run should render a simple message to the log.
5. Close the currently running **Windows Terminal** application.

Review: In this exercise, you created a function that runs automatically based on a fixed schedule.

10.4.6 Exercise 5: Create a function that integrates with other services

10.4.6.1 Task 1: Upload sample content to Azure Blob Storage

1. Access the **funcstor[yourname]** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then create a new container with the following settings:
   - Name: **content**
   - Public access level: **Private (no anonymous access)**
3. Select the recently created **content** container.
4. In the **content** container, select **Upload** to upload the **settings.json** file in the **Allfiles (F): \Allfiles\Labs\02\Starter** folder on your lab VM.
   Note: You should enable the **Overwrite if files already exist** option.

10.4.6.2 Task 2: Create an HTTP-triggered function

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Create a new function with the following details:
   - template: **HTTP trigger**
   - name: **GetSettingInfo**
   Note: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.
4. Close the currently running **Windows Terminal** application.

10.4.6.3 Task 3: Write HTTP-triggered and blob-inputted function code

1. Open **Visual Studio Code**.
2. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\02\Starter\func**.
3. Open the **GetSettingInfo.cs** file.
4. In the code editor, delete all the code within the **GetSettingInfo.cs** file.
5. Add **using directives** for the **Microsoft.AspNetCore.Http**, **Microsoft.AspNetCore.Mvc**, and **Microsoft.Azure.WebJobs** namespaces:

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
```

6. Create a new **public static** class named **GetSettingInfo**:

```
public static class GetSettingInfo
{ }
```

7. Within the **GetSettingInfo** class, create a new **public static** expression-bodied method named **Run** that returns a variable of type **IActionResult** and that also takes in variables of type **HttpRequest** and **string** as parameters named *request* and *json*:

```
public static IActionResult Run(
    HttpRequest request,
    string json)
    => null;
```

   Note: You are only temporarily setting the return value to **null**.

8. Append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **GetSettingInfo**:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    HttpRequest request,
    string json)
    => null;
```

9. Append an attribute to the **request** parameter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **GET**:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    [HttpTrigger("GET")] HttpRequest request,
    string json)
    => null;
```

10. Append an attribute to the **json** parameter of type **BlobAttribute** that has its **blobPath** parameter set to a value of **content/settings.json**:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    [HttpTrigger("GET")] HttpRequest request,
    [Blob("content/settings.json")] string json)
    => null;
```

11. Update the **Run** expression-bodied method to return a new instance of the **OkObjectResult** class passing in the value of the **json** method parameter as the sole constructor parameter:

```
[FunctionName("GetSettingInfo")]
public static IActionResult Run(
    [HttpTrigger("GET")] HttpRequest request,
    [Blob("content/settings.json")] string json)
    => new OkObjectResult(json);
```

12. **Save** the **GetSettingInfo.cs** file.

10.4.6.4 Task 4: Register Azure Storage blob extensions

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Use the following command to **register** the Microsoft.Azure.WebJobs.Extensions.Storage extension:

```
func extensions install --package Microsoft.Azure.WebJobs.Extensions.Storage --version 4.0.4
```

4. Use the following command to validate the extensions were installed correctly by **building** the .NET project:

```
dotnet build
```

5. Close all currently running instances of the **Windows Terminal** application.

10.4.6.5 Task 5: Test the function by using httprepl

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Start the function app project:

   Note: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. Open a new instance of the **Windows Terminal** application.
5. Start the **httprepl** tool, and then set the base URI to `http://localhost:7071`:

   ```
   httprepl http://localhost:7071
   ```

   Note: An error message is displayed by the httprepl tool. This message occurs because the tool is searching for a Swagger definition file to use to "traverse" the API. Because your functio projectp does not produce a Swagger definition file, you'll need to traverse the API manually.

6. When you receive the tool prompt, browse to the relative **api/getsettinginfo** endpoint:

   ```
   cd api
   cd getsettinginfo
   ```

7. Run the **get** command for the current endpoint:

   ```
   get
   ```

8. Observe the JSON content of the response from the function app.
9. Exit the **httprepl** application:

   ```
   exit
   ```

10. Close all currently running instances of the **Windows Terminal** application.

    Review: In this exercise, you created a function that returns the content of a JSON file in Storage.

10.4.7 Exercise 6: Deploy a local function project to an Azure Functions app

10.4.7.1 Task 1: Deploy using the Azure Functions Core Tools

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\func** project directory.
3. Log in to the Azure Command-Line Interface (CLI) by using your Azure credentials:

   ```
   az login
   ```

4. Publish the function app project:

   ```
   func azure functionapp publish <function-app-name>
   ```

   Note: For example, if your **Function App name** is **funclogicstudent**, your command would be `func azure functionapp publish funclogicstudent`. You can review the documentation to [publish the local function app project][azure-functions-core-tools-publish-azure] using the **Azure Functions Core Tools**.

5. Wait for the deployment to finalize before you move forward with the lab.
6. Close the currently running **Windows Terminal** application.

10.4.7.2 Task 2: Validate deployment

1. Sign in to the Azure portal (https://portal.azure.com).
2. Access the **funclogic**[**yourname**] function app that you created earlier in this lab.
3. From the **App Service** blade, locate and open the **Functions** section, then locate and open the **GetSettingInfo** function.
4. In the **Function** blade, select the **Code + Test** option from the **Developer** section.
5. In the function editor, select **Test/Run**.
6. In the popup dialog that appears, perform the following actions:
   - In the **HTTP method** list, select **GET**.
7. Select **Run** to test the function.
8. Observe the results of the test run. the JSON content should be the same as the **settings.json** file.

   Review: In this exercise, you deployed a local function project to Azure Functions and validated that the functions work in Azure.

10.4.8 Exercise 7: Clean up your subscription

10.4.8.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

10.4.8.2 Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **Serverless** resource group:

   ```
   az group delete --name Serverless --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

10.4.8.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.

   Review: In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

10.5 [azure-functions-core-tools]: https://docs.microsoft.com/azure/azure-functions/functions-run-local [azure-functions-core-tools-new-function]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#create-func [azure-functions-core-tools-new-project]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#create-a-local-functions-project [azure-functions-core-tools-start-function]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#start [azure-functions-core-tools-publish-azure]: https://docs.microsoft.com/azure/azure-functions/functions-run-local#publish

10.6 lab: az204Title: 'Lab 03: Retrieving Azure Storage resources and metadata by using the Azure Storage SDK for .NET' az020Title: 'Lab 03: Retrieving Azure Storage resources and metadata by using the Azure Storage SDK for .NET' az204Module: 'Module 03: Develop solutions that use blob storage' az020Module: 'Module 03: Develop solutions that use blob storage' type: 'Answer Key'

11 Lab 03: Retrieving Azure Storage resources and metadata by using the Azure Storage SDK for .NET

12 Student lab answer key

12.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

12.2 Instructions

12.2.1 Before you start

12.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

12.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer

12.2.2 Exercise 1: Create Azure resources

12.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

Note: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

12.2.2.2 Task 2: Create a Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. On the **All services** blade, select **Storage Accounts**.
3. On the **Storage accounts** blade, find your list of Storage instances.
4. On the **Storage accounts** blade, select **New**.
5. Find the tabs on the **Create storage account** blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new storage account. You can select **Review + Create** at any time to skip the remaining tabs.
6. On the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **StorageMedia**, and then select **OK**.
   3. In the **Storage account name** text box, enter mediastor*[yourname]***.
   4. In the **Location** drop-down list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** drop-down list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** drop-down list, select **Read-access geo-redundant storage (RA-GRS)**.
   8. Select **Review + Create**.
7. On the **Review + Create** tab, review the options that you selected during the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.

   Note: Wait for the creation task to complete before you move forward with this lab.
9. In the Azure portal's navigation pane, select **All services**.
10. On the **All services** blade, select **Storage Accounts**.
11. On the **Storage accounts** blade, select the mediastor*[yourname]*** storage account instance.
12. On the **Storage account** blade, find the **Settings** section, and then select the **Properties** link.
13. In the **Properties** section, record the value of the **Primary Blob Service Endpoint** text box.

    Note: You'll use this value later in the lab.
14. Still on the **Storage account** blade, find the **Settings** section, and then select the **Access keys** link.
15. In the **Access keys** section, perform the following actions:
    1. Record the value in the **Storage account name** text box.
    2. Select any one of the keys, and then record the value in either of the **Key** boxes.

       Note: All these values will be used later in this lab.

12.2.2.3 Review

In this exercise, you created a new Storage account to use throughout the remainder of the lab.

12.2.3 Exercise 2: Upload a blob into a container

12.2.3.1 Task 1: Create storage account containers

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **StorageMedia** resource group that you created earlier in this lab.

3. On the **StorageMedia** blade, select the mediastor*[yourname]*** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up window, perform the following actions:
    1. In the **Name** text box, enter **raster-graphics**.
    2. In the **Public access level** drop-down list, select **Private (no anonymous access)**, and then select **OK**.
7. Back in the **Containers** section, select **+ Container**.
8. In the **New container** pop-up window, perform the following actions:
    1. In the **Name** text box, enter **compressed-audio**.
    2. In the **Public access level** drop-down list, select **Private (no anonymous access)**, and then select **OK**.
9. Back in the **Containers** section, observe the updated list of containers.

12.2.3.2 Task 2: Upload a storage account blob

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **StorageMedia** resource group that you created earlier in this lab.
3. On the **StorageMedia** blade, select the mediastor*[yourname]*** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select the recently created **raster-graphics** container.
6. On the **Container** blade, select **Upload**.
7. In the **Upload blob** window, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\03\Starter\Images**, select the **graph.jpg** file, and then select **Open**.
    3. Ensure that the **Overwrite if files already exist** check box is selected, and then select **Upload**.

       Note: Wait for the blob to upload before you continue with this lab.

12.2.3.3 Review

In this exercise, you created a couple of placeholder containers in the storage account and populated one of the containers with a blob.

12.2.4 Exercise 3: Access containers by using the .NET SDK

12.2.4.1 Task 1: Create .NET project

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\03\Starter\BlobManager**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to create a new .NET project named **BlobManager** in the current folder:

   ```
   dotnet new console --name BlobManager --output .
   ```

Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

6. At the command prompt, enter the following command, and then select Enter to import version 12.0.0 of **Azure.Storage.Blobs** from NuGet:

```
dotnet add package Azure.Storage.Blobs --version 12.0.0
```

Note: The **dotnet add package** command will add the **Azure.Storage.Blobs** package from NuGet. For more information, go to Azure.Storage.Blobs.

7. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

```
dotnet build
```

8. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

12.2.4.2 Task 2: Modify the Program class to access Storage

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, delete all the code in the existing file.
3. Add the following line of code to import the **Azure.Storage**, **Azure.Storage.Blobs**, and **Azure.Storage.Blobs.Models** namespaces from the **Azure.Storage.Blobs** package imported from NuGet:

```
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
```

4. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

```
using System;
using System.Threading.Tasks;
```

5. Enter the following code to create a new **Program** class:

```
public class Program
{
}
```

6. In the **Program** class, enter the following line of code to create a new string constant named **blobServiceEndpoint**:

```
private const string blobServiceEndpoint = "";
```

7. Update the **blobServiceEndpoint** string constant by setting its value to the **Primary Blob Service Endpoint** of the Storage account that you recorded earlier in this lab.
8. In the **Program** class, enter the following line of code to create a new string constant named **storageAccountName**:

```
private const string storageAccountName = "";
```

9. Update the **storageAccountName** string constant by setting its value to the **Storage account name** of the Storage account that you recorded earlier in this lab.
10. In the **Program** class, enter the following line of code to create a new string constant named **storageAccountKey**:

```
private const string storageAccountKey = "";
```

11. Update the **storageAccountKey** string constant by setting its value to the **Key** of the Storage account that you recorded earlier in this lab.
12. In the **Program** class, enter the following code to create a new asynchronous **Main** method:

```
public static async Task Main(string[] args)
{
}
```

13. Observe the **Program.cs** file, which should now include:

```
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using System;
using System.Threading.Tasks;

public class Program
{
    private const string blobServiceEndpoint = "<primary-blob-service-endpoint>";
    private const string storageAccountName = "<storage-account-name>";
    private const string storageAccountKey = "<key>";

    public static async Task Main(string[] args)
    {
    }
}
```

12.2.4.3 Task 3: Connect to the Azure Storage blob service endpoint

1. In the **Main** method, add the following line of code to create a new instance of the **StorageSharedKeyCredential** class by using the **storageAccountName** and **storageAccountKey** constants as constructor parameters:

```
StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName,
```

2. In the **Main** method, add the following line of code to create a new instance of the **BlobServiceClient** class by using the **blobServiceEndpoint** constant and the *accountCredentials* variable as constructor parameters:

```
BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCrede
```

3. In the **Main** method, add the following line of code to invoke the **GetAccountInfoAsync** method of the **BlobServiceClient** class to retrieve account metadata from the service:

```
AccountInfo info = await serviceClient.GetAccountInfoAsync();
```

4. In the **Main** method, add the following line of code to render a welcome message:

```
await Console.Out.WriteLineAsync($"Connected to Azure Storage Account");
```

5. In the **Main** method, add the following line of code to render the storage account's name:

```
await Console.Out.WriteLineAsync($"Account name:\t{storageAccountName}");
```

6. In the **Main** method, add the following line of code to render the type of storage account:

```
await Console.Out.WriteLineAsync($"Account kind:\t{info?.AccountKind}");
```

7. In the **Main** method, add the following line of code to render the currently selected stock keeping unit (SKU) for the storage account:

```
await Console.Out.WriteLineAsync($"Account sku:\t{info?.SkuName}");
```

8. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountN

    BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountC

    AccountInfo info = await serviceClient.GetAccountInfoAsync();
```

```
    await Console.Out.WriteLineAsync($"Connected to Azure Storage Account");
    await Console.Out.WriteLineAsync($"Account name:\t{storageAccountName}");
    await Console.Out.WriteLineAsync($"Account kind:\t{info?.AccountKind}");
    await Console.Out.WriteLineAsync($"Account sku:\t{info?.SkuName}");
}
```

9. Save the **Program.cs** file.
10. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
11. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.
12. Observe the output from the currently running console application. The output contains metadata for the Storage account that was retrieved from the service.
13. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

12.2.4.4 Task 4: Enumerate the existing containers

1. In the **Program** class, enter the following code to create a new **private static** method named **EnumerateContainersAsync** that's asynchronous and has a single **BlobServiceClient** parameter type:

    ```
    private static async Task EnumerateContainersAsync(BlobServiceClient client)
    {
    }
    ```
2. In the **EnumerateContainersAsync** method, enter the following code to create an asynchronous **foreach** loop that iterates over the results of an invocation of the **GetBlobContainersAsync** method of the **BlobServiceClient** class:

    ```
    await foreach (BlobContainerItem container in client.GetBlobContainersAsync())
    {
    }
    ```
3. Within the **foreach** loop, enter the following code to print the name of each container:

    ```
    await Console.Out.WriteLineAsync($"Container:\t{container.Name}");
    ```
4. Observe the **EnumerateContainersAsync** method, which should now include:

    ```
    private static async Task EnumerateContainersAsync(BlobServiceClient client)
    {
        await foreach (BlobContainerItem container in client.GetBlobContainersAsync())
        {
            await Console.Out.WriteLineAsync($"Container:\t{container.Name}");
        }
    }
    ```
5. In the **Main** method, enter the following code at the end of the method to invoke the **EnumerateContainersAsync** method, passing in the *serviceClient* variable as a parameter:

    ```
    await EnumerateContainersAsync(serviceClient);
    ```
6. Observe the **Main** method, which should now include:

    ```
    public static async Task Main(string[] args)
    {
        // Existing code removed for brevity

        await EnumerateContainersAsync(serviceClient);
    }
    ```

7. Save the **Program.cs** file.
8. In the **Visual Studio Code** window, right-click or access the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
9. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

   ```
   dotnet run
   ```

   > **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

10. Observe the output from the currently running console application. The updated output includes a list of every existing container in the account.
11. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

12.2.4.5 Review

In this exercise, you accessed existing containers by using the Azure Storage SDK.

12.2.5 Exercise 4: Retrieve blob Uniform Resource Identifiers (URIs) by using the .NET SDK

12.2.5.1 Task 1: Enumerate the blobs in an existing container by using the SDK

1. In the **Program** class, enter the following code to create a new **private static** method named **EnumerateBlobsAsync** that's asynchronous and has two parameter types, **BlobServiceClient** and **string**:

   ```
   private static async Task EnumerateBlobsAsync(BlobServiceClient client, string containerName)
   {
   }
   ```

2. In the **EnumerateBlobsAsync** method, enter the following code to get a new instance of the **BlobContainerClient** class by using the **GetBlobContainerClient** method of the **BlobServiceClient** class, passing in the **containerName** parameter:

   ```
   BlobContainerClient container = client.GetBlobContainerClient(containerName);
   ```

3. In the **EnumerateBlobsAsync** method, enter the following code to render the name of the container that will be enumerated:

   ```
   await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");
   ```

4. In the **EnumerateBlobsAsync** method, enter the following code to create an asynchronous **foreach** loop that iterates over the results of an invocation of the **GetBlobsAsync** method of the **BlobContainerClient** class:

   ```
   await foreach (BlobItem blob in container.GetBlobsAsync())
   {
   }
   ```

5. Within the **foreach** loop, enter the following code to print the name of each blob:

   ```
    await Console.Out.WriteLineAsync($"Existing Blob:\t{blob.Name}");
   ```

6. Observe the **EnumerateBlobsAsync** method, which should now include:

   ```
   private static async Task EnumerateBlobsAsync(BlobServiceClient client, string containerName)
   {
       BlobContainerClient container = client.GetBlobContainerClient(containerName);

       await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");

       await foreach (BlobItem blob in container.GetBlobsAsync())
       {
            await Console.Out.WriteLineAsync($"Existing Blob:\t{blob.Name}");
       }
   }
   ```

7. In the **Main** method, enter the following code at the end of the method to create a variable named *existingContainerName* with a value of **raster-graphics**:

```
string existingContainerName = "raster-graphics";
```

8. In the **Main** method, enter the following code at the end of the method to invoke the **EnumerateBlobsAsync** method, passing in the *serviceClient* and *existingContainerName* variables as parameters:

```
await EnumerateBlobsAsync(serviceClient, existingContainerName);
```

9. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    // Existing code removed for brevity

    await EnumerateContainersAsync(serviceClient);

    string existingContainerName = "raster-graphics";
    await EnumerateBlobsAsync(serviceClient, existingContainerName);
}
```

10. Save the **Program.cs** file.
11. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
12. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

13. Observe the output from the currently running console application. The updated output includes metadata about the existing container and blobs.
14. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

12.2.5.2 Task 2: Create a new container by using the SDK

1. In the **Program** class, enter the following code to create a new **private static** method named **GetContainerAsync** that's asynchronous and has two parameter types, **BlobServiceClient** and **string**:

```
private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string
{
}
```

2. In the **GetContainerAsync** method, enter the following code to get a new instance of the **BlobContainerClient** class by using the **GetBlobContainerClient** method of the **BlobServiceClient** class, passing in the **containerName** parameter:

```
BlobContainerClient container = client.GetBlobContainerClient(containerName);
```

3. In the **GetContainerAsync** method, enter the following code to invoke the **CreateIfNotExistsAsync** method of the **BlobContainerClient** class:

```
await container.CreateIfNotExistsAsync(PublicAccessType.Blob);
```

4. In the **GetContainerAsync** method, enter the following code to render the name of the container that was potentially created:

```
await Console.Out.WriteLineAsync($"New Container:\t{container.Name}");
```

5. In the **GetContainerAsync** method, enter the following code to return the instance of the **BlobContainerClient** class named **container** as the result of the **GetContainerAsync** method:

```
return container;
```

6. Observe the **GetContainerAsync** method, which should now include:

```
private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string
{
    BlobContainerClient container = client.GetBlobContainerClient(containerName);

    await container.CreateIfNotExistsAsync(PublicAccessType.Blob);

    await Console.Out.WriteLineAsync($"New Container:\t{container.Name}");

    return container;
}
```

7. In the **Main** method, enter the following code at the end of the method to create a variable named *newContainerName* with a value of **vector-graphics**:

```
string newContainerName = "vector-graphics";
```

8. In the **Main** method, enter the following code at the end of the method to invoke the **GetContainerAsync** method, passing in the *serviceClient* and *newContainerName* variables as parameters, and to store the result in a variable named *containerClient* of type **BlobContainerClient**:

```
BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName);
```

9. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    // Existing code removed for brevity

    await EnumerateContainersAsync(serviceClient);

    string existingContainerName = "raster-graphics";
    await EnumerateBlobsAsync(serviceClient, existingContainerName);

    string newContainerName = "vector-graphics";
    BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName)
}
```

10. Save the **Program.cs** file.
11. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
12. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

13. Observe the output from the currently running console application. The updated output includes metadata about the existing container and blobs.
14. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

12.2.5.3 Task 3: Upload a new blob by using the portal

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **StorageMedia** resource group that you created earlier in this lab.
3. On the **StorageMedia** blade, select the mediastor*[yourname]*** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select the newly created **vector-graphics** container.

6. On the **Container** blade, select **Upload**.
7. In the **Upload blob** window, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\03\Starter\Images**, select the **graph.svg** file, and then select **Open**.
    3. Ensure that the **Overwrite if files already exist** check box is selected, and then select **Upload**.

       Note: Wait for the blob to upload before you continue with this lab.

12.2.5.4 Task 4: Access blob URI by using the SDK

1. In the **Program** class, enter the following code to create a new **private static** method named **GetBlobAsync** that's asynchronous and has two parameter types, **BlobContainerClient** and **string**:

```
private static async Task<BlobClient> GetBlobAsync(BlobContainerClient client, string blobName)
{
}
```

2. In the **GetBlobAsync** method, enter the following code to get a new instance of the **BlobClient** class by using the **GetBlobClient** method of the **BlobContainerClient** class, passing in the **blobName** parameter:

```
BlobClient blob = client.GetBlobClient(blobName);
```

3. In the **GetBlobAsync** method, enter the following code to render the name of the blob that was referenced:

```
await Console.Out.WriteLineAsync($"Blob Found:\t{blob.Name}");
```

4. In the **GetBlobAsync** method, enter the following code to return the instance of the **BlobClient** class named **blob** as the result of the **GetBlobAsync** method:

```
return blob;
```

5. Observe the **GetBlobAsync** method, which should now include:

```
private static async Task<BlobClient> GetBlobAsync(BlobContainerClient client, string blobName)
{
    BlobClient blob = client.GetBlobClient(blobName);
    await Console.Out.WriteLineAsync($"Blob Found:\t{blob.Name}");
    return blob;
}
```

6. In the **Main** method, enter the following code at the end of the method to create a variable named *uploadedBlobName* with a value of **graph.svg**:

```
string uploadedBlobName = "graph.svg";
```

7. In the **Main** method, enter the following code at the end of the method to invoke the **GetBlobAsync** method, passing in the *containerClient* and *uploadedBlobName* variables as parameters, and to store the result in a variable named *blobClient* of type **BlobClient**:

```
BlobClient blobClient = await GetBlobAsync(containerClient, uploadedBlobName);
```

8. In the **Main** method, enter the following code at the end of the method to render the **Uri** property of the *blobClient* variable:

```
await Console.Out.WriteLineAsync($"Blob Url:\t{blobClient.Uri}");
```

9. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    // Existing code removed for brevity

    await EnumerateContainersAsync(serviceClient);

    string existingContainerName = "raster-graphics";
```

```
        await EnumerateBlobsAsync(serviceClient, existingContainerName);

        string newContainerName = "vector-graphics";
        BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName)

        string uploadedBlobName = "graph.svg";
        BlobClient blobClient = await GetBlobAsync(containerClient, uploadedBlobName);

        await Console.Out.WriteLineAsync($"Blob Url:\t{blobClient.Uri}");
    }
```

10. Save the **Program.cs** file.
11. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
12. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.
13. Observe the output from the currently running console application. The updated output includes the final URL to access the blob online. Record the value of this URL to use later in the lab.

    Note: The URL will likely be similar to the following string: **https://mediastor*[yourname]*.blob.core.wind graphics/graph.svg**
14. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

12.2.5.5 Task 5: Test the URI by using a browser

1. On the taskbar, right-click the **Microsoft Edge** icon or activate the shortcut menu, and then select **New window**.
2. In the new browser window, go to the URL that you copied for the blob earlier in this lab.
3. You should now notice the Scalable Vector Graphic (SVG) file in your browser window.

12.2.5.6 Review

In this exercise, you created containers and managed blobs by using the Storage SDK.

12.2.6 Exercise 5: Clean up your subscription

12.2.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).
2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

      Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

12.2.6.2 Task 2: Delete a resource group

1. At the command prompt, enter the following command, and then select Enter to delete the **StorageMedia** resource group:

   ```
   az group delete --name StorageMedia --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

12.2.6.3 Task 3: Close the active application

1. `the currently running Microsoft Edge application.`

12.2.6.4 Review

12.3 In this exercise, you cleaned up your subscription by removing the resource group used in this lab.

12.4 lab: az204Title: 'Lab 03: Retrieving Azure Storage resources and metadata by using the Azure Storage SDK for .NET' az020Title: 'Lab 03: Retrieving Azure Storage resources and metadata by using the Azure Storage SDK for .NET' az204Module: 'Module 03: Develop solutions that use blob storage' az020Module: 'Module 03: Develop solutions that use blob storage'

13 Lab 03: Retrieving Azure Storage resources and metadata by using the Azure Storage SDK for .NET

14 Student lab manual

14.1 Lab scenario

You're preparing to host a web application in Microsoft Azure that uses a combination of raster and vector graphics. As a development group, your team has decided to store any multimedia content in Azure Storage and manage it in an automated fashion by using C# code in .NET. Before you begin this significant milestone, you have decided to take some time to learn the newest version of the .NET SDK that's used to access Storage by creating a simple application to manage and enumerate blobs and containers.

14.2 Objectives

After you complete this lab, you will be able to:

- Create containers and upload blobs by using the Azure portal.
- Enumerate blobs and containers by using the Microsoft Azure Storage SDK for .NET.
- Pull blob metadata by using the Storage SDK.

14.3 Lab setup

- Estimated time: **45 minutes**

14.4 Instructions

14.4.1 Before you start

14.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

14.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer

14.4.2 Exercise 1: Create Azure resources

14.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

14.4.2.2 Task 2: Create a Storage account

1. Create a new storage account with the following details:
   - New resource group: **StorageMedia**
   - Name: mediastor*[yourname]***
   - Location: **East US**
   - Performance: **Standard**
   - Account kind: **StorageV2 (general purpose v2)**
   - Replication: **Read-access geo-redundant storage (RA-GRS)**

     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.
2. Open the **Properties** section of your newly created storage account instance.
3. Record the value in the **Primary Blob Service Endpoint** text box. You'll use this value later in this lab.
4. Open the **Access Keys** section of the storage account instance.
5. Record the value in the **Storage account name** text box and any of the **Key** text boxes. You'll use these values later in this lab.

14.4.2.3 Review

In this exercise, you created a new Storage account to use throughout the remainder of the lab.

14.4.3 Exercise 2: Upload a blob into a container

14.4.3.1 Task 1: Create storage account containers

1. Access the mediastor*[yourname]*** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then create a new container with the following settings:
   - Name: **raster-graphics**
   - Public access level: **Private (no anonymous access)**
3. Select the **Containers** link in the **Blob service** section, and then create a new container with the following settings:
   - Name: **compressed-audio**
   - Public access level: **Private (no anonymous access)**
4. Observe the updated list of containers.

14.4.3.2 Task 2: Upload a storage account blob

1. Access the mediastor*[yourname]*** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then select the recently created **raster-graphics** container.
3. In the **raster-graphics** container, select **Upload** to upload the **graph.jpg** file in the **Allfiles (F):\Allfiles\Labs\03\Starter\Images** folder on your lab VM.

   Note: We recommend that you enable the **Overwrite if files already exist** option.

14.4.3.3 Review

In this exercise, you created a couple of placeholder containers in the storage account and populated one of the containers with a blob.

14.4.4 Exercise 3: Access containers by using the .NET SDK

14.4.4.1 Task 1: Create .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\03\Starter\BlobManager** folder.
2. Using a terminal, create a new .NET project named **BlobManager** in the current folder:

   ```
   dotnet new console --name BlobManager --output .
   ```

   Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.
3. Using the same terminal, import version 12.0.0 of **Azure.Storage.Blobs** from NuGet:

   ```
   dotnet add package Azure.Storage.Blobs --version 12.0.0
   ```

   Note: The **dotnet add package** command will add the **Azure.Storage.Blobs** package from NuGet. For more information, go to Azure.Storage.Blobs.
4. Using the same terminal, build the .NET web application:

   ```
   dotnet build
   ```
5. Close the current terminal.

14.4.4.2 Task 2: Modify the Program class to access Storage

1. Open the **Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that will be referenced by the application:

   ```
   using Azure.Storage;
   using Azure.Storage.Blobs;
   using Azure.Storage.Blobs.Models;
   using System;
   using System.Threading.Tasks;
   ```
4. Create a new **Program** class with three constant string properties named **blobServiceEndpoint**, **storageAccountName**, and **storageAccountKey**, and then create an asynchronous **Main** entry point method:

   ```
   public class Program
   {
       private const string blobServiceEndpoint = "";
       private const string storageAccountName = "";
       private const string storageAccountKey = "";

       public static async Task Main(string[] args)
       {
       }
   }
   ```

5. Update the **blobServiceEndpoint** string constant by setting its value to the **Primary Blob Service Endpoint** of the storage account that you recorded earlier in this lab.
6. Update the **storageAccountName** string constant by setting its value to the **Storage account name** of the Storage account that you recorded earlier in this lab.
7. Update the **storageAccountKey** string constant by setting its value to the **Key** of the Storage account that you recorded earlier in this lab.

14.4.4.3 Task 3: Connect to the Azure Storage blob service endpoint

1. In the **Main** method, add the following block of code to connect to the storage account and to retrieve account metadata:

```
StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName,

BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCrede

AccountInfo info = await serviceClient.GetAccountInfoAsync();
```

2. Still in the **Main** method, add the following block of code to print metadata about the storage account:

```
await Console.Out.WriteLineAsync($"Connected to Azure Storage Account");
await Console.Out.WriteLineAsync($"Account name:\t{storageAccountName}");
await Console.Out.WriteLineAsync($"Account kind:\t{info?.AccountKind}");
await Console.Out.WriteLineAsync($"Account sku:\t{info?.SkuName}");
```

3. Save the **Program.cs** file.
4. Using a terminal, run the console application project:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

5. Observe the output from the currently running console application. The output contains metadata for the Storage account that was retrieved from the service.
6. Close the current terminal.

14.4.4.4 Task 4: Enumerate the existing containers

1. In the **Program** class, create a new **private static** method named **EnumerateContainersAsync** that's asynchronous and has a single parameter of type **BlobServiceClient**:

```
private static async Task EnumerateContainersAsync(BlobServiceClient client)
{
}
```

2. In the **EnumerateContainersAsync** method, create an asynchronous **foreach** loop that iterates over the results of an invocation of the **GetBlobContainersAsync** method of the **BlobServiceClient** class and prints out the name of each container:

```
await foreach (BlobContainerItem container in client.GetBlobContainersAsync())
{
    await Console.Out.WriteLineAsync($"Container:\t{container.Name}");
}
```

3. In the **Main** method, add a new line of code to invoke the **EnumerateContainersAsync** method, passing in the *serviceClient* variable as a parameter:

```
await EnumerateContainersAsync(serviceClient);
```

4. Save the **Program.cs** file.
5. Using a terminal, run the console application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

6. Observe the output from the currently running console application. The updated output includes a list of every existing container in the account.
7. Close the current terminal.

14.4.4.5 Review

In this exercise, you accessed existing containers by using the Storage SDK.

14.4.5 Exercise 4: Retrieve blob Uniform Resource Identifiers (URIs) by using the .NET SDK

14.4.5.1 Task 1: Enumerate the blobs in an existing container by using the SDK

1. In the **Program** class, create a new **private static** method named **EnumerateBlobsAsync** that's asynchronous and has two types of parameters, **BlobServiceClient** and **string**:

```
private static async Task EnumerateBlobsAsync(BlobServiceClient client, string containerName)
{
}
```

2. In the **EnumerateBlobsAsync** method, get a new instance of the **BlobContainerClient** class by using the **GetBlobContainerClient** method of the **BlobServiceClient** class, passing in the **containerName** parameter:

```
BlobContainerClient container = client.GetBlobContainerClient(containerName);
```

3. In the **EnumerateBlobsAsync** method, render the name of the container that will be enumerated:

```
await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");
```

4. In the **EnumerateBlobsAsync** method, create an asynchronous **foreach** loop that iterates over the results of an invocation of the **GetBlobsAsync** method of the **BlobContainerClient** class and prints out the name of each blob:

```
await foreach (BlobItem blob in container.GetBlobsAsync())
{
    await Console.Out.WriteLineAsync($"Existing Blob:\t{blob.Name}");
}
```

5. In the **Main** method, add a new line of code to create a variable named *existingContainerName* with a value of **raster-graphics**:

```
string existingContainerName = "raster-graphics";
```

6. In the **Main** method, add a new line of code to invoke the **EnumerateBlobsAsync** method, passing in the *serviceClient* and *existingContainerName* variables as parameters:

```
await EnumerateBlobsAsync(serviceClient, existingContainerName);
```

7. Save the **Program.cs** file.
8. Using a terminal, run the console application project:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

9. Observe the output from the currently running console application. The updated output includes metadata about the existing container and blobs.
10. Close the current terminal.

14.4.5.2 Task 2: Create a new container by using the SDK

1. In the **Program** class, create a new **private static** method named **GetContainerAsync** that's asynchronous and has two parameter types, **BlobServiceClient** and **string**:

```
private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string
{
}
```

2. In the **GetContainerAsync** method, get a new instance of the **BlobContainerClient** class by using the **GetBlobContainerClient** method of the **BlobServiceClient** class, passing in the **containerName** parameter. Invoke the **CreateIfNotExistsAsync** method of the **BlobContainerClient** class:

```
BlobContainerClient container = client.GetBlobContainerClient(containerName);
await container.CreateIfNotExistsAsync(PublicAccessType.Blob);
```

3. In the **GetContainerAsync** method, render the name of the container that was potentially created:

```
await Console.Out.WriteLineAsync($"New Container:\t{container.Name}");
```

4. Return the instance of the **BlobContainerClient** class named **container** as the result of the **GetContainerAsync** method:

```
return container;
```

5. In the **Main** method, add a new line of code to create a variable named *newContainerName* with a value of **vector-graphics**:

```
string newContainerName = "vector-graphics";
```

6. In the **Main** method, add a new line of code to invoke the **GetContainerAsync** method, passing in the *serviceClient* and *newContainerName* variables as parameters:

```
BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName);
```

7. Save the **Program.cs** file.

8. Using a terminal, run the console application project:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

9. Observe the output from the currently running console application. The updated output includes metadata about the new container and blobs.

10. Close the current terminal.

14.4.5.3 Task 3: Upload a new blob by using the portal

1. Access the mediastor*[yourname]*** storage account that you created earlier in this lab.

2. Select the **Containers** link in the **Blob service** section, and then select the newly created **vector-graphics** container.

3. In the **vector-graphics** container, select **Upload** to upload the **graph.svg** file in the **Allfiles (F):\Allfiles\Labs\03\Starter\Images** folder on your lab VM.

> **Note**: We recommend that you enable the **Overwrite if files already exist** option.

14.4.5.4 Task 4: Access blob URI by using the SDK

1. In the **Program** class, create a new **private static** method named **GetBlobAsync** that's asynchronous and has two types of parameters, **BlobContainerClient** and **string**:

```
private static async Task<BlobClient> GetBlobAsync(BlobContainerClient client, string blobName)
{
}
```

2. In the **GetBlobAsync** method, get a new instance of the **BlobClient** class by using the **GetBlobClient** method of the **BlobContainerClient** class, passing in the **blobName** parameter:

```
BlobClient blob = client.GetBlobClient(blobName);
```

3. In the **GetBlobAsync** method, render the name of the blob that was referenced:

```
await Console.Out.WriteLineAsync($"Blob Found:\t{blob.Name}");
```

4. Return the instance of the **BlobClient** class named **blob** as the result of the **GetBlobAsync** method:

   ```
   return blob;
   ```

5. In the **Main** method, add a new line of code to create a variable named *uploadedBlobName* with a value of **vector-graphics**:

   ```
   string uploadedBlobName = "graph.svg";
   ```

6. In the **Main** method, add a new line of code to invoke the **GetBlobAsync** method, passing in the *containerClient* and *uploadedBlobName* variables as parameters and store the result in a variable named *blobClient* of type **BlobClient**:

   ```
   BlobClient blobClient = await GetBlobAsync(containerClient, uploadedBlobName);
   ```

7. In the **Main** method, add a new line of code to render the **Uri** property of the *blobClient* variable:

   ```
   await Console.Out.WriteLineAsync($"Blob Url:\t{blobClient.Uri}");
   ```

8. Save the **Program.cs** file.
9. Using a terminal, run the console application project:

   ```
   dotnet run
   ```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\03\Solution\BlobManager** folder.

10. Observe the output from the currently running console application. The updated output includes the final URL to access the blob online. Record the value of this URL to use later in the lab.

    Note: The URL will likely be similar to the following string: **https://mediastor*[yourname]*.blob.core.wind graphics/graph.svg**

11. Close the current terminal.

14.4.5.5 Task 5: Test the URI by using a browser

1. Using a new browser window or tab, go to the URL for the blob, and then find the blob's contents.
2. You should now notice the Scalable Vector Graphics (SVG) file in your browser window.

14.4.5.6 Review

In this exercise, you created containers and managed blobs by using the Storage SDK.

14.4.6 Exercise 5: Clean up your subscription

14.4.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

14.4.6.2 Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **StorageMedia** resource group:

   ```
   az group delete --name StorageMedia --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

14.4.6.3 Task 3: Close the active application

1. `the currently running Microsoft Edge application.`

14.4.6.4 Review

14.5 In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

14.6 lab: az204Title: 'Lab 04: Constructing a polyglot data solution' az020Title: 'Lab 04: Constructing a polyglot data solution' az204Module: 'Module 04: Develop solutions that use Cosmos DB storage' az020Module: 'Module 04: Develop solutions that use Cosmos DB storage' type: 'Answer Key'

15 Lab 04: Constructing a polyglot data solution

16 Student lab answer key

16.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes following the development of this training content. These changes might cause the lab instructions and steps to not match up correctly.

Microsoft updates this training course when the community brings needed changes to our attention. However, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

16.2 Instructions

16.2.1 Before you start

16.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

16.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Visual Studio Code

16.2.2 Exercise 1: Creating database resources in Azure

16.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, browse to the Azure portal (portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you will be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

16.2.2.2 Task 2: Create an Azure SQL Database server resource

1. In the Azure portal's navigation pane, select **All services**.
2. From the **All services** blade, select **SQL servers**.
3. From the **SQL servers** blade, find your list of SQL server instances.

4. From the **SQL servers** blade, select **New**.
5. From the **Create SQL Database Server** blade, observe the tabs from the blade, such as **Basics**, **Networking**, and **Additional settings**.

   Note: Each tab represents a step in the workflow to create a new Azure SQL Database server. You can select **Review + Create** at any time to skip the remaining tabs.

   1. From the **Basics** tab, perform the following actions:
   2. Leave the **Subscription** drop-down list set to its default value.
   3. In the **Resource group** section, select **Create new**, enter **PolyglotData**, and then select **OK**.
   4. In the **Server name** text box, enter polysqlsrvr*[yourname]***.
   5. In the **Location** drop-down list, select **(US) East US**.
   6. In the **Server admin login** text box, enter **testuser**.
   7. In the **Password** text box, enter **TestPa55w.rd**.
   8. In the **Confirm password** text box, enter **TestPa55w.rd** again.
   9. Select **Next: Networking**.
6. From the **Networking** tab, perform the following actions:
   1. In the **Allow Azure services and resources to access this server** section, select **Yes**.
   2. Select **Review + Create**.
7. From the **Review + Create** tab, review the options that you selected during the previous steps.
8. Select **Create** to create the SQL Database server by using your specified configuration.

   Note: At this point in the lab, we are only creating the Azure SQL logical server. We will create the Azure SQL database instance later in the lab.

   Note: Wait for the creation task to complete before you move forward with this lab.

16.2.2.3 Task 3: Create an Azure Cosmos DB account resource

1. In the Azure portal's navigation pane, select **All services**.
2. From the **All services** blade, select **Azure Cosmos DB**.
3. From the **Azure Cosmos DB** blade, find your list of Azure Cosmos DB instances.
4. From the **Azure Cosmos DB** blade, select **New**.
5. From the **Create Azure Cosmos DB Account** blade, observe the tabs from the blade, such as **Basics**, **Network**, and **Tags**.

   Note: Each tab represents a step in the workflow to create a new Azure Cosmos DB account. You can select **Review + Create** at any time to skip the remaining tabs.
6. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** list set to its default value.
   2. In the **Resource group** section, select **PolyglotData** from the list.
   3. In the **AccountName** text box, enter polycosmos*[yourname]***.
   4. In the **API** drop-down list, select **Core (SQL)**.
   5. In the **Apply Free Tier Discount** section, select **Do Not Apply**.
   6. In the **Location** drop-down list, select the **(US) East US** region.
   7. In the **Account Type** section, select **Non-Production**.
   8. In the **Multi-region Writes** section, select **Disable**.
   9. Select **Review + Create**.
7. From the **Review + Create** tab, review the options that you selected during the previous steps.

8. Select **Create** to create the Azure Cosmos DB account by using your specified configuration.

   Note: Wait for the creation task to complete before you move forward with this lab.

9. In the Azure portal's navigation pane, select the **Resource groups** link.
10. From the **Resource groups** blade, find and then select the **PolyglotData** resource group that you created earlier in this lab.
11. From the **PolyglotData** blade, select the polycosmos*[yourname]*** Azure Cosmos DB account that you created earlier in this lab.
12. From the **Azure Cosmos DB account** blade, find the **Settings** section from the blade, and then select the **Keys** link.
13. In the Keys pane, record the value in the **PRIMARY CONNECTION STRING** text box. You'll use this value later in this lab.

16.2.2.4 Task 4: Create an Azure Storage account resource

1. In the Azure portal's navigation pane, select **All services**.
2. From the **All services** blade, select **Storage Accounts**.
3. From the **Storage accounts** blade, find your list of Storage instances.
4. From the **Storage accounts** blade, select **New**.
5. From the **Create storage account** blade, observe the tabs from the blade, such as **Basics**, **Advanced**, and **Tags**.

   Note: Each tab represents a step in the workflow to create a new Azure Storage account. You can select **Review + Create** at any time to skip the remaining tabs.

6. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** list set to its default value.
   2. In the **Resource group** section, select **PolyglotData** from the list.
   3. In the **Storage account name** text box, enter polystor*[yourname]***.
   4. In the **Location** drop-down list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** drop-down list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** drop-down list, select **Locally-redundant storage (LRS)**.
   8. Select **Review + Create**.
7. From the **Review + Create** tab, review the options that you selected during the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.

   Note: Wait for the creation task to complete before you move forward with this lab.

16.2.2.5 Review

In this exercise, you created all the Azure resources that you'll need for a polyglot data solution.

16.2.3 Exercise 2: Import and validate data

16.2.3.1 Task 1: Upload image blobs

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
3. From the **PolyglotData** blade, select the polystor*[yourname]*** storage account that you created earlier in this lab.

4. From the **Storage account** blade, select the **Containers** link in the **Blob service** section from the blade.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** ollowing actions:
    1. In the **Name** text box, enter **images**.
    2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**.
    3. Select **Create**.
7. Back in the **Containers** section, select the newly created **images** container.
8. From the **Container** blade, find the **Settings** section from the blade, and then select the **Properties** link.
9. In the Properties pane, record the value in the **URL** text box. You'll use this value later in this lab.
10. Find and select the **Overview** link from the blade.
11. From the blade, select **Upload**.
12. In the **Upload blob** pop-up, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\04\Starter\Images**, select all 42 individual **.jpg** image files, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

       Note: Wait for all the blobs to upload before you continue with this lab.

16.2.3.2 Task 2: Upload an SQL .bacpac file

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
3. From the **PolyglotData** blade, select the polystor*[yourname]*** storage account that you created earlier in this lab.
4. From the **Storage account** blade, select the **Containers** link in the **Blob service** section from the blade.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up, perform the following actions:
    1. In the **Name** text box, enter **databases**.
    2. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
    3. Select **OK**.
7. Back in the **Containers** section, select the newly created **databases** container.
8. From the **Container** blade, select **Upload**.
9. In the **Upload blob** pop-up, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\04\Starter**, select the **AdventureWorks.bacpac** file, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

       Note: Wait for the blob to upload before you continue with this lab.

16.2.3.3 Task 3: Import an SQL database

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
3. From the **PolyglotData** blade, select the polysqlsrvr*[yourname]*** SQL server that you created earlier in this lab.
4. From the **SQL server** blade, select **Import database**.
5. From the **Import database** blade, perform the following actions:
    1. Leave the **Subscription** list set to its default value.
    2. Select the **Storage** option.
    3. From the **Storage accounts** blade, select the polystor*[yourname]*** storage account that you created earlier in this lab.
    4. From the **Containers** blade, select the **databases** container that you created earlier in this lab.
    5. From the **Container** blade, select the **AdventureWorks.bacpac** blob that you created earlier in this lab, and then select **Select** to close the blade.
    6. Back from the **Import database** blade, leave the **Pricing tier** option set to its default value.
    7. In the **Database name** text box, enter **AdventureWorks**.
    8. Leave the **Collation** text box set to its default value.
    9. In the **Server admin login** text box, enter **testuser**.
    10. In the **Password** text box, enter **TestPa55w.rd**.
    11. Select **OK**.

        Note: Wait for the database to be created before you continue with this lab. If you receive a firewall-related error on the import step, it means you did not correctly configure the **Allow Azure services to access server** setting on your SQL Server earlier in the lab. Review your settings, delete the empty **AdventureWorks** database, and then attempt your import again.

16.2.3.4 Task 4: Use an imported SQL database

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
3. From the **PolyglotData** blade, select the polysqlsrvr*[yourname]*** SQL server that you created earlier in this lab.
4. From the **SQL server** blade, find the **Security** section from the blade, and then select the **Firewalls and virtual networks** link.
5. In the Firewalls and virtual networks pane, perform the following actions:
    1. Select **Add client IP**
    2. Select **Save**.
    3. In the **Success!** confirmation dialog, select **OK**.

        Note: This step will ensure that your local machine will have access to the databases that are associated with this server.
6. In the Azure portal's navigation pane, select the **Resource groups** link.
7. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
8. From the **PolyglotData** blade, select the **AdventureWorks** SQL database that you created earlier in this lab.

9. From the **SQL database** blade, find the **Settings** section from the blade, and then select the **Connection strings** link.
10. In the Connection strings pane, record the value in the **ADO.NET (SQL Authentication)** text box. You'll use this value later in this lab.
11. Update the connection string that you recorded by performing the following actions:
    1. Within the connection string, find the *your_username* placeholder and replace it with **testuser**.
    2. Within the connection string, find the *your_password* placeholder and replace it with **TestPa55w.rd**.

       Note: For example, if your connection string was originally `Server=tcp:polysqlsrvrinstructor.database.` `Catalog=AdventureWorks;User ID={your_username};Password={your_password};`, your updated connection string will be `Server=tcp:polysqlsrvrinstructor.database.windows.net,1433;` `Catalog=AdventureWorks;User ID=testuser;Password=TestPa55w.rd;`
12. Find and select the **Query editor (preview)** link from the blade.
13. In the Query editor pane, perform the following actions:
    1. In the **Login** text box, enter **testuser**.
    2. In the **Password** text box, enter **TestPa55w.rd**.
    3. Select **OK**.
14. In the open query editor, enter the following query:

    ```
    SELECT * FROM AdventureWorks.dbo.Models
    ```
15. Select **Run** to run the query, and then observe the results.

    Note: This query will return a list of models from the home page of the web application.
16. In the query editor, replace the existing query with the following query:

    ```
    SELECT * FROM AdventureWorks.dbo.Products
    ```
17. Select **Run** to run the query, and then observe the results.

    Note: This query will return a list of products that are associated with each model.

16.2.3.5 Review

In this exercise, you imported all the resources that you'll use with your web application.

16.2.4 Exercise 3: Open and configure a .NET web application

16.2.4.1 Task 1: Open and build the web application

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\04\Starter\AdventureWorks**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to build the .NET web application:

   ```
   dotnet build
   ```

   Note: The **dotnet build** command will automatically restore any missing NuGet packages prior to building all projects in the folder.
6. Observe the results of the build printed in the terminal. The build should complete successfully with no errors or warning messages.
7. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.4.2 Task 2: Update the SQL connection string

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.
2. Open the **appsettings.json** file.
3. In the JavaScript Object Notation (JSON) object on line 3, find the **ConnectionStrings.AdventureWorksSqlConte** path. Observe that the current value is empty:

```
"ConnectionStrings": {
    "AdventureWorksSqlContext": "",
    ...
},
```

4. Update the value of the **AdventureWorksSqlContext** property by setting its value to the **ADO.NET (SQL Authentication) connection string** of the SQL database that you recorded earlier in this lab.

   Note: It's important that you use your updated connection string here. The original connection string copied from the portal won't have the username and password necessary to connect to the SQL database.

5. Save the **appsettings.json** file.

16.2.4.3 Task 3: Update the blob base URL

1. In the JSON object on line 8, find the **Settings.BlobContainerUrl** path. Observe that the current value is empty:

```
"Settings": {
    "BlobContainerUrl": "",
    ...
}
```

2. Update the value of the **BlobContainerUrl** property by setting its value to the **URL** property of the Azure Storage blob container named **images** that you recorded earlier in this lab.
3. Save the **appsettings.json** file.

16.2.4.4 Task 4: Validate the web application

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Web** folder:

```
cd .\AdventureWorks.Web\
```

3. At the command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

   Note: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

4. On the taskbar, select the **Microsoft Edge** icon.
5. In the open browser window, browse to the currently running web application (http://localhost:5000).
6. In the web application, observe the list of models displayed from the front page.
7. Find the **Water Bottle** model, and then select **View Details**.
8. From the **Water Bottle** product detail page, find **Add to Cart**, and then observe that the checkout functionality is currently disabled.
9. Close the browser window displaying your web application.
10. Return to the **Visual Studio Code** window, and then select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.4.5 Review

In this exercise, you configured your ASP.NET web application to connect to your resources in Azure.

16.2.5 Exercise 4: Migrating SQL data to Azure Cosmos DB

16.2.5.1 Task 1: Create a migration project

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to create a new .NET console project named **AdventureWorks.Migrate** in a folder with the same name:

   ```
   dotnet new console --name AdventureWorks.Migrate
   ```

   > **Note**: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

3. At the command prompt, enter the following command, and then select Enter to add a reference to the existing **AdventureWorks.Models** project:

   ```
   dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Models
   ```

   > **Note**: The **dotnet add reference** command will add a reference to the model classes contained in the **AdventureWorks.Models** project.

4. At the command prompt, enter the following command, and then select Enter to add a reference to the existing **AdventureWorks.Context** project:

   ```
   dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Contex
   ```

   > **Note**: The **dotnet add reference** command will add a reference to the context classes contained in the **AdventureWorks.Context** project.

5. At the command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

   ```
   cd .\AdventureWorks.Migrate\
   ```

6. At the command prompt, enter the following command, and then select Enter to import version 2.2.6 of **Microsoft.EntityFrameworkCore.SqlServer** from NuGet:

   ```
   dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 3.0.1
   ```

   > **Note**: The **dotnet add package** command will add the **Microsoft.EntityFrameworkCore.SqlServer** package from **NuGet**. For more information, go to: Microsoft.EntityFrameworkCore.SqlServer.

7. At the command prompt, enter the following command, and then select Enter to import version 3.4.1 of **Microsoft.Azure.Cosmos** from NuGet:

   ```
   dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
   ```

   > **Note**: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, go to: Microsoft.Azure.Cosmos.

8. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

   ```
   dotnet build
   ```

9. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.5.2 Task 2: Create a .NET class

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Migrate** project.
2. Open the **Program.cs** file.
3. From the code editor tab for the **Program.cs** file, delete all the code in the existing file.

4. Add the following lines of code to import the **AdventureWorks.Models** and **AdventureWorks.Context** namespaces from the referenced **AdventureWorks.Models** and **AdventureWorks.Context** projects:

```
using AdventureWorks.Context;
using AdventureWorks.Models;
```

5. Add the following line of code to import the **Microsoft.Azure.Cosmos** namespace from the **Microsoft.Azure.Cosmos** package imported from NuGet:

```
using Microsoft.Azure.Cosmos;
```

6. Add the following line of code to import the **Microsoft.EntityFrameworkCore** namespace from the **Microsoft.EntityFrameworkCore.SqlServer** package imported from NuGet:

```
using Microsoft.EntityFrameworkCore;
```

7. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

8. Enter the following code to create a new **Program** class:

```
public class Program
{
}
```

9. Within the **Program** class, enter the following line of code to create a new string constant named **sqlDBConnectionString**:

```
private const string sqlDBConnectionString = "";
```

10. Update the **sqlDBConnectionString** string constant by setting its value to the **ADO.NET (SQL Authentication) connection string** of the SQL database that you recorded earlier in this lab.

    Note: It's important that you use your updated connection string here. The original connection string copied from the portal won't have the username and password necessary to connect to the SQL database.

11. Within the **Program** class, enter the following line of code to create a new string constant named **cosmosDBConnectionString**:

```
private const string cosmosDBConnectionString = "";
```

12. Update the **cosmosDBConnectionString** string constant by setting its value to the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.

13. Within the **Program** class, enter the following code to create a new asynchronous **Main** method:

```
public static async Task Main(string[] args)
{
}
```

14. Within the **Main** method, add the following line of code to print an introductory message to the console:

```
await Console.Out.WriteLineAsync("Start Migration");
```

15. Save the **Program.cs** file.

16. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

17. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

18. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

```
dotnet build
```

19. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.5.3 Task 3: Get SQL database records by using Entity Framework

1. Within the **Main** method of the **Program** class within the **Program.cs** file, add the following line of code to create a new instance of the **AdventureWorksSqlContext** class, passing in the *sqlDBConnectionString* variable as the connection string value:

```
using AdventureWorksSqlContext context = new AdventureWorksSqlContext(sqlDBConnectionString);
```

2. Within the **Main** method, add the following block of code to issue a language-integrated query (LINQ) to get all **Models** and child **Products** from the database and store them in an in-memory **List<>** collection:

```
List<Model> items = await context.Models
    .Include(m => m.Products)
    .ToListAsync<Model>();
```

3. Within the **Main** method, add the following line of code to print the number of records imported from SQL Database:

```
await Console.Out.WriteLineAsync($"Total Azure SQL DB Records: {items.Count}");
```

4. Save the **Program.cs** file.

5. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

6. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

7. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

```
dotnet build
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Migrate** folder.

8. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.5.4 Task 4: Insert items into Azure Cosmos DB

1. Within the **Main** method of the **Program** class within the **Program.cs** file, add the following line of code to create a new instance of the **CosmosClient** class, passing in the *cosmosDBConnectionString* variable as the connection string value:

```
using CosmosClient client = new CosmosClient(cosmosDBConnectionString);
```

2. Within the **Main** method, add the following line of code to create a new **database** named **Retail** if it doesn't already exist in the Azure Cosmos DB account:

```
Database database = await client.CreateDatabaseIfNotExistsAsync("Retail");
```

3. Within the **Main** method, add the following block of code to create a new **container** named **Online** if it doesn't already exist in the Azure Cosmos DB account with a partition key path of **/Category** and a throughput of **1000** Request Units:

```
Container container = await database.CreateContainerIfNotExistsAsync("Online",
    partitionKeyPath: $"/{nameof(Model.Category)}",
    throughput: 1000
);
```

4. Within the **Main** method, add the following line of code to create an *int* variable named **count**:

```
int count = 0;
```

5. Within the **Main** method, add the following block of code to create a **foreach** loop that iterates over the objects in the **items** collection:

```
foreach (var item in items)
{
}
```

6. Within the **foreach** loop in the **Main** method, add the following line of code to **upsert** the object into the Azure Cosmos DB collection and save the result in a variable of type *ItemResponse<>* named **document**:

```
ItemResponse<Model> document = await container.UpsertItemAsync<Model>(item);
```

7. Within the **foreach** loop contained in the **Main** method, add the following line of code to print the activity ID of each upsert operation:

```
await Console.Out.WriteLineAsync($"Upserted document #{++count:000} [Activity Id: {document.Activi
```

8. Back within the **Main** method (outside of the **foreach** loop), add the following line of code to print the number of documents exported to Azure Cosmos DB:

```
await Console.Out.WriteLineAsync($"Total Azure Cosmos DB Documents: {count}");
```

9. Save the **Program.cs** file.
10. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
11. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

12. At the command prompt, enter the following command, and then select Enter to build the .NET console application:

```
dotnet build
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Migrate** folder.

16.2.5.5 Task 5: Perform a migration

1. At the open command prompt, enter the following command, and then select Enter to run the .NET console application:

```
dotnet run
```

> **Note**: The **dotnet run** command will start the console application.

2. Observe the various data that prints to the screen, including initial SQL record count, individual upsert activity identifiers, and final Azure Cosmos DB document count.
3. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.5.6 Task 6: Validate the migration

1. Return to the **Microsoft Edge** browser window with the Azure portal.
2. In the Azure portal's navigation pane, select the **Resource groups** link.
3. From the **Resource groups** blade, find and select the **PolyglotData** resource group that you created earlier in this lab.
4. From the **PolyglotData** blade, select the polycosmos*[yourname]*** Azure Cosmos DB account that you created earlier in this lab.
5. From the **Azure Cosmos DB account** blade, find and select the **Data Explorer** link from the blade.
6. In the Data Explorer pane, expand the **Retail** database node.
7. Expand the **Online** container node, and then select **New SQL Query**.

Note: The label for this option might be hidden. You can get labels by hovering over the icons in the Data Explorer pane.

8. From the query tab, enter the following text:

   ```
   SELECT * FROM models
   ```

9. Select **Execute Query**, and then observe the list of JSON models that the query returns.
10. Back in the query editor, replace the existing text with the following text:

    ```
    SELECT VALUE COUNT(1) FROM models
    ```

11. Select **Execute Query**, and then observe the result of the **COUNT** aggregate operation.
12. Return to the **Visual Studio Code** window.

16.2.5.7 Review

In this exercise, you used Entity Framework and the .NET SDK for Azure Cosmos DB to migrate data from SQL Database to Azure Cosmos DB.

16.2.6 Exercise 5: Accessing Azure Cosmos DB by using .NET

16.2.6.1 Task 1: Update library with the Cosmos SDK and references

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Context** folder:

   ```
   cd .\AdventureWorks.Context\
   ```

3. At the command prompt, enter the following command, and then select Enter to import **Microsoft.Azure.Cosmos** from NuGet:

   ```
   dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
   ```

   Note: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, go to: Microsoft.Azure.Cosmos.

4. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

   ```
   dotnet build
   ```

5. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.6.2 Task 2: Write .NET code to connect to Azure Cosmos DB

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Context** project.
2. Access the shortcut menu or right-click or activate the shortcut menu for the **AdventureWorks.Context** folder node, and then select **New File**.
3. At the new file prompt, enter **AdventureWorksCosmosContext.cs**.
4. From the code editor tab for the **AdventureWorksCosmosContext.cs** file, add the following lines of code to import the **AdventureWorks.Models** namespace from the referenced **AdventureWorks.Models** project:

   ```
   using AdventureWorks.Models;
   ```

5. Add the following lines of code to import the **Microsoft.Azure.Cosmos** and **Microsoft.Azure.Cosmos.Linq** namespaces from the **Microsoft.Azure.Cosmos** package imported from NuGet:

   ```
   using Microsoft.Azure.Cosmos;
   using Microsoft.Azure.Cosmos.Linq;
   ```

6. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

7. Enter the following code to add an **AdventureWorks.Context** namespace block:

```
namespace AdventureWorks.Context
{
}
```

8. Within the **AdventureWorks.Context** namespace, enter the following code to create a new **AdventureWorksCosmosContext** class:

```
public class AdventureWorksCosmosContext
{
}
```

9. Update the declaration of the **AdventureWorksCosmosContext** class by adding a specification indicating that this class will implement the **IAdventureWorksProductContext** interface:

```
public class AdventureWorksCosmosContext : IAdventureWorksProductContext
{
}
```

10. Within the **AdventureWorksCosmosContext** class, enter the following line of code to create a new read-only *Container* variable named **__container**:

```
private readonly Container _container;
```

11. Within the **AdventureWorksCosmosContext** class, add a new constructor with the following signature:

```
public AdventureWorksCosmosContext(string connectionString, string database = "Retail", string con
{
}
```

12. Within the constructor, add the following block of code to create a new instance of the **CosmosClient** class and then obtain both a **Database** and **Container** instance from the client:

```
_container = new CosmosClient(connectionString)
    .GetDatabase(database)
    .GetContainer(container);
```

13. Within the **AdventureWorksCosmosContext** class, add a new **FindModelAsync** method with the following signature:

```
public async Task<Model> FindModelAsync(Guid id)
{
}
```

14. Within the **FindModelAsync** method, add the following blocks of code to create a LINQ query, transform it into an iterator, iterate over the result set, and then return the single item in the result set:

```
var iterator = _container.GetItemLinqQueryable<Model>()
    .Where(m => m.id == id)
    .ToFeedIterator<Model>();

List<Model> matches = new List<Model>();
while (iterator.HasMoreResults)
{
    var next = await iterator.ReadNextAsync();
    matches.AddRange(next);
}

return matches.SingleOrDefault();
```

15. Within the **AdventureWorksCosmosContext** class, add a new **GetModelsAsync** method with the following signature:

```
public async Task<List<Model>> GetModelsAsync()
{
}
```

16. Within the **GetModelsAsync** method, add the following blocks of code to run an SQL query, get the query result iterator, iterate over the result set, and then return the union of all results:

```
string query = $@"SELECT * FROM items";

var iterator = _container.GetItemQueryIterator<Model>(query);

List<Model> matches = new List<Model>();
while (iterator.HasMoreResults)
{
    var next = await iterator.ReadNextAsync();
    matches.AddRange(next);
}

return matches;
```

17. Within the **AdventureWorksCosmosContext** class, add a new **FindProductAsync** method with the following signature:

```
public async Task<Product> FindProductAsync(Guid id)
{
}
```

18. Within the **FindProductAsync** method, add the following blocks of code to run an SQL query, get the query result iterator, iterate over the result set, and then return the single item in the result set:

```
string query = $@"SELECT VALUE products
                    FROM models
                    JOIN products in models.Products
                    WHERE products.id = '{id}'";

var iterator = _container.GetItemQueryIterator<Product>(query);

List<Product> matches = new List<Product>();
while (iterator.HasMoreResults)
{
    var next = await iterator.ReadNextAsync();
    matches.AddRange(next);
}

return matches.SingleOrDefault();
```

19. Save the **AdventureWorksCosmosContext.cs** file.

20. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

21. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Context** folder:

```
cd .\AdventureWorks.Context\
```

22. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

```
dotnet build
```

> **Note**: If there are any build errors, review the **AdventureWorksCosmosContext.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Context** folder.

23. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.6.3 Task 3: Update the Azure Cosmos DB connection string

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.
2. Open the **appsettings.json** file.
3. In the JSON object on line 4, find the **ConnectionStrings.AdventureWorksCosmosContext** path. Observe that the current value is empty:

```
"ConnectionStrings": {
    ...
    "AdventureWorksCosmosContext": "",
    ...
},
```

4. Update the value of the **AdventureWorksCosmosContext** property by setting its value to the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.
5. Save the **appsettings.json** file.

16.2.6.4 Task 4: Update .NET application startup logic

1. In the Explorer pane of the **Visual Studio Code** window, expand the **AdventureWorks.Web** project.
2. Open the **Startup.cs** file.
3. In the **Startup** class, find the existing **ConfigureProductService** method:

```
public void ConfigureProductService(IServiceCollection services)
{
    services.AddScoped<IAdventureWorksProductContext, AdventureWorksSqlContext>(provider =>
        new AdventureWorksSqlContext(
            _configuration.GetConnectionString(nameof(AdventureWorksSqlContext))
        )
    );
}
```

   Note: The current product service uses SQL as its database.

4. Within the **ConfigureProductService** method, delete all existing lines of code:

```
public void ConfigureProductService(IServiceCollection services)
{
}
```

5. Within the **ConfigureProductService** method, add the following block of code to change the products provider to the **AdventureWorksCosmosContext** implementation that you created earlier in this lab:

```
services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosContext>(provider =>
    new AdventureWorksCosmosContext(
        _configuration.GetConnectionString(nameof(AdventureWorksCosmosContext))
    )
);
```

6. Save the **Startup.cs** file.

16.2.6.5 Task 5: Validate that the .NET application successfully connects to Azure Cosmos DB

1. In the **Visual Studio Code** window, access the shortcut menu or right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to switch your terminal context to the **AdventureWorks.Web** folder:

```
cd .\AdventureWorks.Web\
```

3. At the command prompt, enter the following command, and then select Enter to run the ASP.NET web application:

   ```
   dotnet run
   ```

   Note: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

4. On the taskbar, select the **Microsoft Edge** icon.
5. In the open browser window, browse to the currently running web application (http://localhost:5000).
6. In the web application, observe the list of models displayed from the front page.
7. Find the **Touring-1000** model, and then select **View Details**.
8. From the **Touring-1000** product detail page, perform the following actions:
   1. In the **Select options** list, select **Touring-1000 Yellow, 50, $2,384.07**.
   2. Find **Add to Cart**, and then observe that the checkout functionality is still disabled.
9. Close the browser window displaying your web application.
10. Return to the **Visual Studio Code** window, and then select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

16.2.6.6 Review

In this exercise, you wrote C# code to query an Azure Cosmos DB collection by using the .NET SDK.

16.2.7 Exercise 6: Clean up your subscription

16.2.7.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

      Note: Wait for the Cloud Shell to finish its initial setup procedures before moving forward with the lab.If you don't notice the Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

16.2.7.2 Task 2: Delete resource groups

1. At the command prompt, enter the following command, and then select Enter to delete the **PolyglotData** resource group:

   ```
   az group delete --name PolyglotData --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

16.2.7.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

16.2.7.4 Review

16.3 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

16.4 lab: az204Title: 'Lab 04: Constructing a polyglot data solution' az020Title: 'Lab 04: Constructing a polyglot data solution' az204Module: 'Module 04: Develop solutions that use Cosmos DB storage' az020Module: 'Module 04: Develop solutions that use Cosmos DB storage'

17 Lab 04: Constructing a polyglot data solution

18 Student lab manual

18.1 Lab scenario

You have been assigned the task of updating your company's existing retail web application to use more than one data service in Microsoft Azure. Your company's goal is to take advantage of the best data service for each application component. After conducting thorough research, you decide to migrate your inventory database from Azure SQL Database to Azure Cosmos DB.

18.2 Objectives

After you complete this lab, you will be able to:

- Create instances of various database services by using the Azure portal.
- Write C# code to connect to SQL Database.
- Write C# code to connect to Azure Cosmos DB.

18.3 Lab setup

- Estimated time: **45 minutes**

18.4 Instructions

18.4.1 Before you start

18.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

18.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Visual Studio Code

18.4.2 Exercise 1: Creating database resources in Azure

18.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select the **Get Started** button to skip the tour.

18.4.2.2 Task 2: Create an Azure SQL Database server resource

1. Create a new **Azure SQL Database server (logical server)** resource with the following details:
   - Server name: polysqlsrvr*[yourname]***
   - New resource group: **PolyglotData**
   - Server admin login: **testuser**
   - Password: **TestPa55w.rd**
   - Location: **(US) East US**
   - Allow Azure services to access server: **Yes**

     Note: At this point in the lab, we are only creating the Azure SQL logical server. We will create the Azure SQL database instance later in the lab.

     Note: Wait for Azure to finish creating the SQL server instance before you move forward with the lab. You'll receive a notification when the SQL server is created.

18.4.2.3 Task 3: Create an Azure Cosmos DB account resource

1. Create a new **Azure Cosmos DB** instance with the following details:
   - Account name: polycosmos*[yourname]***
   - Existing resource group: **PolyglotData**
   - API: **Core (SQL)**
   - Apply Free Tier Discount: **Do Not Apply**
   - Location: **(US) East US**
   - Account Type **Non-Production**
   - Multi-region writes: **Disable**

     Note: Wait for Azure to finish creating the Azure Cosmos DB account before you move forward with the lab. You'll receive a notification when the Azure Cosmos DB account is created.
2. Go to the blade for your newly created Azure Cosmos DB account resource, and then open the Keys pane.
3. In the Keys pane, record the value of the **PRIMARY CONNECTION STRING** text box.

   Note: You'll use these values later in this lab.

18.4.2.4 Task 4: Create an Azure Storage account resource

1. Create a new Azure Storage account with the following details:
   - Storage account name: polystor*[yourname]***
   - Existing resource group: **PolyglotData**
   - Account kind: **StorageV2 (general purpose v2)**
   - Location: **(US) East US**
   - Replication: **Locally-redundant storage (LRS)**
   - Performance: **Standard**

     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the storage account is created.

18.4.2.5 Review

In this exercise, you created all the Azure resources that you'll need for a polyglot data solution.

18.4.3 Exercise 2: Import and validate data

18.4.3.1 Task 1: Upload image blobs

1. Go to the blade for the polystor*[yourname]*** Azure Storage account that you created earlier in this lab.
2. Open the Containers pane, and then create a new container with the following settings:
   - Name: **images**
   - Public access level: **Blob (anonymous read access for blobs only)**
3. Go to the new **images** container, and then open the Properties pane.
4. In the Properties pane, record the value in the **URL** text box.

   Note: You'll use this value later in this lab.
5. Return to the blade for the **images** container.
6. Use the **Upload** button to upload the 42 individual **.jpg** image files in the **Allfiles (F):\Allfiles\Labs\04\Starter\In** folder on your lab machine.

   Note: We recommended that you enable the **Overwrite if files already exist** option.

18.4.3.2 Task 2: Upload an SQL .bacpac file

1. Go back to the blade for the polystor*[yourname]*** Azure Storage account, and then open the Containers pane again.
2. Create a new container with the following settings:
   - Name: **databases**
   - Public access level: **Private (no anonymous access)**
3. Go to the new **databases** container.
4. Use the **Upload** button to upload the **AdventureWorks.bacpac** file in the **Allfiles (F):\Allfiles\Labs\04\Starter** folder on your lab machine.

   Note: We recommended that you enable the **Overwrite if files already exist** option.

18.4.3.3 Task 3: Import an SQL database

1. Go to the blade for the polysqlsrvr*[yourname]*** SQL server resource that you created earlier in this lab.
2. Import the database from your Azure Storage account into the SQL server instance by using the following details:
   - Storage account: polystor*[yourname]***
   - Database backup blob: **databases\AdventureWorks.bacpac**
   - Database name: **AdventureWorks**
   - Server admin login: **testuser**
   - Password: **TestPa55w.rd**

   Note: Wait for the database to be created before you continue with this lab. If you receive a firewall-related error on the import step, it means you did not correctly configure the **Allow Azure services to access server** setting on your SQL Server earlier in the lab. Review your settings, delete the empty **AdventureWorks** database, and then attempt your import again.

18.4.3.4 Task 4: Use an imported SQL database

1. Go back to the blade for the polysqlsrvr*[yourname]*** SQL server resource.
2. Open the Firewalls and virtual networks pane.
3. Add your current client IP address to the list of allowed IP addresses, and then save the list.
4. Go to the blade for the **AdventureWorks** SQL database resource that you recently imported.

5. Open the Connection strings pane, and then record the value of the **ADO.NET (SQL Authentication)** connection string.
6. Update the connection string that you recorded by replacing the placeholder values for *your_username* and *your_password*

   Note: For example, if your connection string was originally `Server=tcp:polysqlsrvrinstructor.database.win` `Catalog=AdventureWorks;User ID={your_username};Password={your_password};`, your updated connection string will be `Server=tcp:polysqlsrvrinstructor.database.windows.net,1433;Initial` `Catalog=AdventureWorks;User ID=testuser;Password=TestPa55w.rd;`
7. Go back to the blade for the **AdventureWorks** SQL database resource.
8. Open the Query editor pane, and then sign in by using the following credentials:
   - Username: **testuser**
   - Password: **TestPa55w.rd**
9. Run the following query, and then observe the results:

   ```
   SELECT * FROM AdventureWorks.dbo.Models
   ```

   Note: This query will return a list of models from the home page of the web application.
10. Run this additional query, and then observe the results:

    ```
    SELECT * FROM AdventureWorks.dbo.Products
    ```

    Note: This query will return a list of products that are associated with each model.

18.4.3.5 Review

In this exercise, you imported all the resources that you'll use with your web application.

18.4.4 Exercise 3: Open and configure a .NET web application

18.4.4.1 Task 1: Open and build the web application

1. Using Visual Studio Code, open the solution folder found at **Allfiles (F):\Allfiles\Labs\04\Starter\AdventureWo**
2. Using a terminal, build the .NET solution:

   ```
   dotnet build
   ```

   Note: The **dotnet build** command will automatically restore any missing NuGet packages prior to building all projects in the folder.
3. Close the current terminal.

18.4.4.2 Task 2: Update the SQL connection string

1. Open the **AdventureWorks.Web/appsettings.json** file in Visual Studio Code.
2. Replace the value of the *ConnectionStrings.AdventureWorksSqlContext* property with the **ADO.NET (SQL Authentication) connection string** of the SQL database that you recorded earlier in this lab.

   Note: It's important that you use your updated connection string here. The original connection string copied from the portal won't have the username and password necessary to connect to the SQL database.
3. Save the **appsettings.json** file.

18.4.4.3 Task 3: Update the blob base URL

1. Replace the value of the *Settings.BlobContainerUrl* property with the *URL* property of the **Azure Storage** blob container named **images** that you recorded earlier in this lab.
2. Save the **appsettings.json** file.

18.4.4.4 Task 4: Validate the web application

1. Using a terminal, change your context to the **AdventureWorks.Web** folder:

   ```
   cd .\AdventureWorks.Web\
   ```

2. Using the same terminal, run the ASP.NET web application project:

   ```
   dotnet run
   ```

   > **Note**: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

3. Open the Microsoft Edge browser.
4. In the open browser window, browse to the web application that's hosted at **localhost** on port **5000**.

   > **Note**: The URL is http://localhost:5000.

5. In the web application, observe the list of models displayed from the front page.
6. Find the **Water Bottle** model, and then select **View Details**.
7. From the **Water Bottle** product detail page, find **Add to Cart**, and then observe that the checkout functionality is currently disabled.
8. Close the browser window that's displaying your web application.
9. Return to the **Visual Studio Code** window.
10. Close the current terminal.

18.4.4.5 Review

In this exercise, you configured your ASP.NET web application to connect to your resources in Azure.

18.4.5 Exercise 4: Migrating SQL data to Azure Cosmos DB

18.4.5.1 Task 1: Create a migration project

1. Using a terminal, create a new .NET console project named **AdventureWorks.Migrate** in a folder with the same name:

   ```
   dotnet new console --name AdventureWorks.Migrate
   ```

   > **Note**: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

2. Using the same terminal, add a reference to the existing **AdventureWorks.Models** project:

   ```
   dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Models
   ```

   > **Note**: The **dotnet add reference** command will add a reference to the model classes contained in the **AdventureWorks.Models** project.

3. Using the same terminal, add a reference to the existing **AdventureWorks.Context** project:

   ```
   dotnet add .\AdventureWorks.Migrate\AdventureWorks.Migrate.csproj reference .\AdventureWorks.Conte
   ```

   > **Note**: The **dotnet add reference** command will add a reference to the context classes contained in the **AdventureWorks.Context** project.

4. Using the same terminal, change your context to the **AdventureWorks.Migrate** folder:

   ```
   cd .\AdventureWorks.Migrate\
   ```

5. Using the same terminal, import version 3.0.1 of **Microsoft.EntityFrameworkCore.SqlServer** from NuGet:

   ```
   dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 3.0.1
   ```

   > **Note**: The **dotnet add package** command will add the **Microsoft.EntityFrameworkCore.SqlServer** package from **NuGet**. For more information, go to: Microsoft.EntityFrameworkCore.SqlServer.

6. Using the same terminal, import version 3.4.1 of **Microsoft.Azure.Cosmos** from NuGet:

```
dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
```

> **Note**: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, go to: Microsoft.Azure.Cosmos.

7. Using the same terminal, build the .NET console application:

```
dotnet build
```

8. Close the current terminal.

18.4.5.2 Task 2: Create a .NET class

1. Open the **AdventureWorks.Migrate/Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that will be referenced by the application:

```
using AdventureWorks.Context;
using AdventureWorks.Models;
using Microsoft.Azure.Cosmos;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

4. Create a new **Program** class with two constant string properties and an asynchronous **Main** entry point method:

```
public class Program
{
    private const string sqlDBConnectionString = "";
    private const string cosmosDBConnectionString = "";

    public static async Task Main(string[] args)
    {
    }
}
```

5. Update the **sqlDBConnectionString** string constant by setting its value to the **ADO.NET (SQL Authentication) connection string** of the SQL database that you recorded earlier in this lab.

> **Note**: It's important that you use your updated connection string here. The original connection string copied from the portal won't have the username and password necessary to connect to the SQL database.

6. Update the **cosmosDBConnectionString** string constant by setting its value to the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.

18.4.5.3 Task 3: Get SQL database records by using Entity Framework

1. Within the **Main** method, add the following blocks of code to export all model and product records from SQL Database to local memory:

```
await Console.Out.WriteLineAsync("Start Migration");

using AdventureWorksSqlContext context = new AdventureWorksSqlContext(sqlDBConnectionString);

List<Model> items = await context.Models
    .Include(m => m.Products)
    .ToListAsync<Model>();

await Console.Out.WriteLineAsync($"Total Azure SQL DB Records: {items.Count}");
```

2. Save the **Program.cs** file.
3. Using a terminal, change your context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

4. Using the same terminal, build the .NET console application project:

```
dotnet build
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Migrate** folder.

5. Close the current terminal.

18.4.5.4 Task 4: Insert items into Azure Cosmos DB

1. Still within the **Main** method, add the following blocks of code to import the in-memory model and product data as documents to Azure Cosmos DB:

```
using CosmosClient client = new CosmosClient(cosmosDBConnectionString);

Database database = await client.CreateDatabaseIfNotExistsAsync("Retail");

Container container = await database.CreateContainerIfNotExistsAsync("Online",
    partitionKeyPath: $"/{nameof(Model.Category)}",
    throughput: 1000
);

int count = 0;
foreach (var item in items)
{
    ItemResponse<Model> document = await container.UpsertItemAsync<Model>(item);
    await Console.Out.WriteLineAsync($"Upserted document #{++count:000} [Activity Id: {document.Ac
}

await Console.Out.WriteLineAsync($"Total Azure Cosmos DB Documents: {count}");
```

2. Save the **Program.cs** file.
3. Using a terminal, change your context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate\
```

4. Using the same terminal, build the .NET console application project:

```
dotnet build
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Migrate** folder.

18.4.5.5 Task 5: Perform a migration

1. Using the same terminal, run the .NET console application project:

```
dotnet run
```

> **Note**: The **dotnet run** command will start the console application.

2. Observe the various data that prints to the screen, including initial SQL record count, individual upsert activity identifiers, and final Azure Cosmos DB document count.
3. Close the current terminal.

18.4.5.6 Task 6: Validate the migration

1. Return to the **Microsoft Edge** browser window with the Azure portal.
2. Go to the blade for the polycosmos*[yourname]*** Azure Cosmos DB account that you created earlier in this lab.
3. Open the Data Explorer pane.
4. Create a new **SQL query** tab within the context of the **Retail** database and **Online** container.

5. Run the following query, and then observe the results:

   ```
   SELECT * FROM models
   ```

6. Run the following query, and then observe the results:

   ```
   SELECT VALUE COUNT(1) FROM models
   ```

18.4.5.7 Review

In this exercise, you used Entity Framework and the .NET SDK for Azure Cosmos DB to migrate data from SQL Database to Azure Cosmos DB.

18.4.6 Exercise 5: Accessing Azure Cosmos DB by using .NET

18.4.6.1 Task 1: Update library with the Cosmos SDK and references

1. Using a terminal, change your context to the **AdventureWorks.Context** folder:

   ```
   cd .\AdventureWorks.Context\
   ```

2. Using the same terminal, import **Microsoft.Azure.Cosmos** from NuGet:

   ```
   dotnet add package Microsoft.Azure.Cosmos --version 3.4.1
   ```

   > **Note**: The **dotnet add package** command will add the **Microsoft.Azure.Cosmos** package from **NuGet**. For more information, go to: Microsoft.Azure.Cosmos.

3. Using the same terminal, build the ASP.NET web application project:

   ```
   dotnet build
   ```

4. Close the current terminal.

18.4.6.2 Task 2: Write .NET code to connect to Azure Cosmos DB

1. Create a new **AdventureWorks.Context/AdventureWorksCosmosContext.cs** file in Visual Studio Code.

2. Add the following **using** directives for libraries that will be referenced by the application:

   ```
   using AdventureWorks.Models;
   using Microsoft.Azure.Cosmos;
   using Microsoft.Azure.Cosmos.Linq;
   using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Threading.Tasks;
   ```

3. Enter the following code to add an **AdventureWorks.Context** namespace block:

   ```
   namespace AdventureWorks.Context
   {
   }
   ```

4. Create a new **AdventureWorksCosmosContext** class that implements the **IAdventureWorksProductContext** interface with a single read-only *Container* variable:

   ```
   public class AdventureWorksCosmosContext : IAdventureWorksProductContext
   {
       private readonly Container _container;
   }
   ```

5. Within the **AdventureWorksCosmosContext** class, add a new constructor that creates a new instance of the **CosmosClient** class, and then obtain both a **Database** and **Container** instance from the client:

   ```
   public AdventureWorksCosmosContext(string connectionString, string database = "Retail", string con
   {
       _container = new CosmosClient(connectionString)
           .GetDatabase(database)
           .GetContainer(container);
   ```

```
}
```

6. Within the **AdventureWorksCosmosContext** class, add a new **FindModelAsync** method that creates a LINQ query, transforms it into an iterator, iterates over the result set, and then returns the single item in the result set:

```
public async Task<Model> FindModelAsync(Guid id)
{
    var iterator = _container.GetItemLinqQueryable<Model>()
        .Where(m => m.id == id)
        .ToFeedIterator<Model>();

    List<Model> matches = new List<Model>();
    while (iterator.HasMoreResults)
    {
        var next = await iterator.ReadNextAsync();
        matches.AddRange(next);
    }

    return matches.SingleOrDefault();
}
```

7. Within the **AdventureWorksCosmosContext** class, add a new **GetModelsAsync** method that runs an SQL query, gets the query result iterator, iterates over the result set, and then returns the union of all results:

```
public async Task<List<Model>> GetModelsAsync()
{
    string query = $@"SELECT * FROM items";

    var iterator = _container.GetItemQueryIterator<Model>(query);

    List<Model> matches = new List<Model>();
    while (iterator.HasMoreResults)
    {
        var next = await iterator.ReadNextAsync();
        matches.AddRange(next);
    }

    return matches;
}
```

8. Within the **AdventureWorksCosmosContext** class, add a new **FindProductAsync** method that runs an SQL query, gets the query result iterator, iterates over the result set, and then returns the single item in the result set:

```
public async Task<Product> FindProductAsync(Guid id)
{
    string query = $@"SELECT VALUE products
                        FROM models
                        JOIN products in models.Products
                        WHERE products.id = '{id}'";

    var iterator = _container.GetItemQueryIterator<Product>(query);

    List<Product> matches = new List<Product>();
    while (iterator.HasMoreResults)
    {
        var next = await iterator.ReadNextAsync();
        matches.AddRange(next);
    }

    return matches.SingleOrDefault();
}
```

9. Save the **AdventureWorksCosmosContext.cs** file.
10. Using a terminal, change your context to the **AdventureWorks.Context** folder:

    ```
    cd .\AdventureWorks.Context\
    ```

11. Using the same terminal, build the ASP.NET web application project:

    ```
    dotnet build
    ```

    Note: If there are any build errors, review the **AdventureWorksCosmosContext.cs** file in the **Allfiles (F):\Allfiles\Labs\04\Solution\AdventureWorks\AdventureWorks.Context** folder.

12. Close the current terminal.

18.4.6.3 Task 3: Update the Azure Cosmos DB connection string

1. Open the **AdventureWorks.Web/appsettings.json** file in Visual Studio Code.
2. Replace the value of the *ConnectionStrings.AdventureWorksCosmosContext* property with the **PRIMARY CONNECTION STRING** of the Azure Cosmos DB account that you recorded earlier in this lab.
3. Save the **appsettings.json** file.

18.4.6.4 Task 4: Update .NET application startup logic

1. Open the **AdventureWorks.Web/Startup.cs** file in Visual Studio Code.
2. In the **Startup** class, find the existing **ConfigureProductService** method.

    Note: The current product service uses SQL as its database.

3. Replace the **ConfigureProductService** method with the following code:

    ```
    public void ConfigureProductService(IServiceCollection services)
    {
        services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosContext>(provider =>
                new AdventureWorksCosmosContext(
                    _configuration.GetConnectionString(nameof(AdventureWorksCosmosContext))
                )
        );
    }
    ```

4. Save the **Startup.cs** file.

18.4.6.5 Task 5: Validate that the .NET application successfully connects to Azure Cosmos DB

1. Using a terminal, change your context to the **AdventureWorks.Web** folder:

    ```
    cd .\AdventureWorks.Web\
    ```

2. Using the same terminal, run the ASP.NET web application project:

    ```
    dotnet run
    ```

    Note: The **dotnet run** command will automatically build any changes to the project and then start the web application without a debugger attached. The command will output the URL of the running application and any assigned ports.

3. Open the Microsoft Edge browser.
4. In the open browser window, browse to the web application that's hosted at **localhost** on port **5000**.

    Note: The URL is http://localhost:5000.

5. In the web application, observe the list of models displayed from the front page.
6. Find the **Touring-1000** model, and then select **View Details**.
7. From the **Touring-1000** product detail page, perform the following actions:

1. In the **Select options** list, select **Touring-1000 Yellow, 50, $2,384.07**.
2. Find **Add to Cart**.

8. Close the browser window displaying your web application.
9. Return to the **Visual Studio Code** window.
10. Close the current terminal.

18.4.6.6 Review

In this exercise, you wrote C# code to query an Azure Cosmos DB collection by using the .NET SDK.

18.4.7 Exercise 6: Clean up your subscription

18.4.7.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

18.4.7.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **PolyglotData** resource group:

   ```
   az group delete --name PolyglotData --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

18.4.7.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

18.4.7.4 Review

18.5 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

18.6 lab: az204Title: 'Lab 05: Deploying compute workloads by using images and containers' az204Module: 'Module 05: Implement IaaS solutions' type: 'Answer Key'

19 Lab 05: Deploying compute workloads by using images and containers

20 Student lab answer key

20.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

20.2 Instructions

20.2.1 Before you start

20.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**

- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

20.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer

20.2.2 Exercise 1: Create a VM by using the Azure Command-Line Interface (CLI)

20.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you will be offered a tour of the portal. Select **Get Started** to begin using the portal.

20.2.2.2 Task 2: Create a resource group

1. In the Azure portal's navigation pane, select the **Create a resource** link.

   Note: If you can't find the **Create a resource** link, the **Create a resource** icon is a plus sign (+) character from the portal.
2. From the **New** blade, find the **Search the Marketplace** text box above the list of featured services.
3. In the search box, enter the text **Resource Group**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Resource group** result.
5. From the **Resource group** blade, select **Create**.
6. From the additional **Resource group** blade, find the tabs from the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new resource group. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** text box, enter the value **ContainerCompute**.
   3. In the **Region** drop-down list, select the **(US) East US** location.
   4. Select **Review + Create**.
8. From the **Review + Create** tab, review the options that you selected during the previous steps.
9. Select **Create** to create the resource group by using your specified configuration.

   Note: Wait for the creation task to complete before moving forward with this lab.

20.2.2.3 Task 3: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign () and underscore character (_).
2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

   Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice the **Cloud Shell** configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

3. At the **Cloud Shell** command prompt in the portal, enter the following command, and then select Enter to get the version of the Azure CLI tool:

   ```
   az --version
   ```

20.2.2.4 Task 4: Use the Azure CLI commands

1. Enter the following command, and then select Enter to get a list of subgroups and commands at the root level of the CLI:

   ```
   az --help
   ```

2. Enter the following command, and then select Enter to get a list of subgroups and commands for Azure Virtual Machines:

   ```
   az vm --help
   ```

3. Enter the following command, and then select Enter to get a list of arguments and examples for the **Create Virtual Machine** command:

   ```
   az vm create --help
   ```

4. Enter the following command, and then select Enter to create a new **virtual machine** with the following settings:

   - Resource group: **ContainerCompute**
   - Name: **quickvm**
   - Image: **Debian**
   - Username: **Student**
   - Password: **StudentPa55w.rd**

   ```
   az vm create --resource-group ContainerCompute --name quickvm --image Debian --admin-username stud
   ```

   Note: Wait for the VM creation process to complete. After the process completes, the command will return a JSON file containing details about the machine.

5. Enter the following command, and then select Enter to get a more detailed JavaScript Object Notation (JSON) file that contains various metadata about the newly created VM:

   ```
   az vm show --resource-group ContainerCompute --name quickvm
   ```

6. Enter the following command, and then select Enter to list all the IP addresses associated with the VM:

   ```
   az vm list-ip-addresses --resource-group ContainerCompute --name quickvm
   ```

7. Enter the following command, and then select Enter to filter the output to only return the first IP address value:

   ```
   az vm list-ip-addresses --resource-group ContainerCompute --name quickvm --query '[].{ip:virtualMa
   ```

8. Enter the following command, and then select Enter to store the results of the previous command in a new Bash shell variable named *ipAddress*:

   ```
   ipAddress=$(az vm list-ip-addresses --resource-group ContainerCompute --name quickvm --query '[].{
   ```

9. Enter the following command, and then select Enter to render the value of the Bash shell variable *ipAddress*:

   ```
   echo $ipAddress
   ```

10. Enter the following command, and then select Enter to connect to the VM that you created earlier in this lab by using the Secure Shell (SSH) tool and the IP address stored in the Bash shell variable *ipAddress*:

```
ssh student@$ipAddress
```

11. The SSH tool will first inform you that the authenticity of the host can't be established and then ask if you want to continue connecting. Enter **yes**, and then select Enter to continue connecting to the VM.
12. The SSH tool will then ask you for a password. Enter **StudentPa55w.rd**, and then select Enter to authenticate with the VM.
13. After connecting to the VM by using SSH, enter the following command, and then select Enter to get metadata describing the Linux VM:

    ```
    uname -a
    ```

14. Use the **exit** command to end your SSH session:

    ```
    exit
    ```

15. Close the Cloud Shell pane in the portal.

20.2.2.5 Review

In this exercise, you used Cloud Shell to create a VM as part of an automated script.

20.2.3 Exercise 2: Create a Docker container image and deploy it to Azure Container Registry

20.2.3.1 Task 1: Open the Cloud Shell and editor

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

   Note: Wait for Cloud Shell to finish connecting to an instance before moving on with the lab.

2. At the **Cloud Shell** command prompt in the portal, enter the following command, and then select Enter to move from the root directory to the ~/**clouddrive** directory:

   ```
   cd ~/clouddrive
   ```

3. Enter the following command, and then select Enter to create a new directory named **ipcheck** in the ~/**clouddrive** directory:

   ```
   mkdir ipcheck
   ```

4. Enter the following command, and then select Enter to change the active directory from ~/**clouddrive** to ~/**clouddrive/ipcheck**:

   ```
   cd ~/clouddrive/ipcheck
   ```

5. Enter the following command, and then select Enter to create a new .NET console application in the current directory:

   ```
   dotnet new console --output . --name ipcheck
   ```

6. Enter the following command, and then select Enter to create a new file in the ~/**clouddrive/ipcheck** directory named **Dockerfile**:

   ```
   touch Dockerfile
   ```

7. Enter the following command, and then select Enter to open the embedded graphical editor in the context of the current directory:

   ```
   code .
   ```

20.2.3.2 Task 2: Create and test a .NET application

1. In the graphical editor, find the FILES pane, and then open the **Program.cs** file to open it in the editor.
2. Delete the entire contents of the **Program.cs** file.
3. Copy and paste the following code into the **Program.cs** file:

```
public class Program
{
    public static void Main(string[] args)
    {
        // Check if network is available
```

```
        if (System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())
        {
            System.Console.WriteLine("Current IP Addresses:");

            // Get host entry for current hostname
            string hostname = System.Net.Dns.GetHostName();
            System.Net.IPHostEntry host = System.Net.Dns.GetHostEntry(hostname);

            // Iterate over each IP address and render their values
            foreach(System.Net.IPAddress address in host.AddressList)
            {
                System.Console.WriteLine($"\t{address}");
            }
        }
        else
        {
            System.Console.WriteLine("No Network Connection");
        }
    }
}
```

4. Save the **Program.cs** file by using the menu in the graphical editor or the Ctrl+S keyboard shortcut. Don't close the graphical editor.
5. Back at the command prompt, enter the following command, and then select Enter to run the application:

   ```
   dotnet run
   ```

6. Find the results of the run. At least one IP address should be listed for the Cloud Shell instance.
7. In the graphical editor, find the FILES pane of the editor, and then open the **Dockerfile** file to open it in the editor.
8. Copy and paste the following code into the **Dockerfile** file:

   ```
   # Start using the .NET Core 3.1 SDK container image
   FROM mcr.microsoft.com/dotnet/sdk:3.1-alpine AS build

   # Change current working directory
   WORKDIR /app

   # Copy existing files from host machine
   COPY . ./

   # Publish application to the "out" folder
   RUN dotnet publish --configuration Release --output out

   # Start container by running application DLL
   ENTRYPOINT ["dotnet", "out/ipcheck.dll"]
   ```

9. Save the **Dockerfile** file by using the menu in the graphical editor or by using the Ctrl+S keyboard shortcut.
10. Close the Cloud Shell pane in the portal.

20.2.3.3 Task 3: Create a Container Registry resource

1. In the Azure portal's navigation pane, select the **Create a resource** link.
2. From the **New** blade, find the **Search the Marketplace** text box above the list of featured services.
3. In the search box, enter **Container Registry**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Container Registry** result.
5. From the **Container Registry** blade, select **Create**.
6. From the **Create container registry** blade, perform the following actions:

1. In the **Registry name** text box, give your registry a globally unique name.

   Note: The blade will automatically check the name for uniqueness and inform you if you're required to choose a different name.

2. Leave the **Subscription** text box set to its default value.
3. In the **Resource group** drop-down list, select the existing **ContainerCompute** option.
4. In the **Location** text box, select **East US**.
5. In the **SKU** drop-down list, select **Basic**.
6. Select **Create**.

   Note: Wait for the creation task to complete before moving forward with this lab.

20.2.3.4 Task 4: Open Azure Cloud Shell and store Container Registry metadata

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: Wait for Cloud Shell to finish connecting to an instance before moving forward with the lab.

2. At the **Cloud Shell** command prompt in the portal, enter the following command, and then select Enter to get a list of all container registries in your subscription:

   ```
   az acr list
   ```

3. Enter the following command, and then select Enter:

   ```
   az acr list --query "max_by([], &creationDate).name" --output tsv
   ```

4. Enter the following command, and then select Enter:

   ```
   acrName=$(az acr list --query "max_by([], &creationDate).name" --output tsv)
   ```

5. Enter the following command, and then select Enter:

   ```
   echo $acrName
   ```

20.2.3.5 Task 5: Deploy a Docker container image to Container Registry

1. Enter the following command, and then select Enter to change the active directory from ~/ to ~/**clouddrive/ipcheck**:

   ```
   cd ~/clouddrive/ipcheck
   ```

2. Enter the following command, and then select Enter to get the contents of the current directory:

   ```
   dir
   ```

3. Enter the following command, and then select Enter to upload the source code to your container registry and build the container image as a Container Registry task:

   ```
   az acr build --registry $acrName --image ipcheck:latest .
   ```

   Note: Wait for the build task to complete before moving forward with this lab.

4. Close the Cloud Shell pane in the portal.

20.2.3.6 Task 6: Validate your container image in Container Registry

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ContainerCompute** resource group that you created earlier in this lab.
3. From the **ContainerCompute** blade, select the container registry that you created earlier in this lab.
4. From the **Container Registry** blade, find the **Services** section, and then select the **Repositories** link.
5. In the **Repositories** section, select the **ipcheck** container image repository.
6. From the **Repository** blade, select the **latest** tag.

7. Find the metadata for the version of your container image with the **latest** tag.

   Note: You can also select the **Run ID** link to find metadata about the build task.

20.2.3.7 Review

In this exercise, you created a .NET console application to display a machine's current IP address. You then added the **Dockerfile** file to the application so that it could be converted into a Docker container image. Finally, you deployed the container image to Container Registry.

20.2.4 Exercise 3: Deploy an Azure container instance

20.2.4.1 Task 1: Enable the admin user in Container Registry

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ContainerCompute** resource group that you created earlier in this lab.
3. From the **ContainerCompute** blade, select the container registry that you created earlier in this lab.
4. From the **Container Registry** blade, select **Update**.
5. From the **Update container registry** blade, perform the following actions:
   1. In the **Admin user** section, select **Enable**.
   2. Select **Save**.
   3. Close the blade.
6. Close the **Update container registry** blade.

20.2.4.2 Task 2: Automatically deploy a container image to an Azure container instance

1. From the **Container Registry** blade, find the **Services** section, and then select the **Repositories** link.
2. In the **Repositories** section, select the **ipcheck** container image repository.
3. From the **Repository** blade, select the ellipsis menu associated with the **latest** tag entry.
4. In the pop-up menu, select the **Run instance** link.
5. From the **Create container instance** blade, perform the following actions:
   1. In the **Container name** text box, enter **managedcompute**.
   2. Leave the **Container image** text box set to its default value.
   3. In the **OS type** section, select **Linux**.
   4. Leave the **Subscription** text box set to its default value.
   5. In the **Resource group** drop-down list, select **ContainerCompute**.
   6. In the **Location** drop-down list, select **East US**.
   7. In the **Number of cores** drop-down list, select **2**.
   8. In the **Memory (GB)** text box, enter **4**.
   9. In the **Public IP address** section, select **No**.
   10. Select **OK**.

   Note: Wait for the creation task to complete before moving forward with this lab.

20.2.4.3 Task 3: Manually deploy a container image to Container Instances

1. In the Azure portal's navigation pane, select the **Create a resource** link.
2. From the **New** blade, find the **Search the Marketplace** text box above the list of featured services.
3. In the search box, enter **container instances**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Container Instances** result.

5. From the **Container Instances** blade, select **Create**.
6. Find the tabs from the **Create Container Instances** blade, such as **Basics**, **Networking**, and **Advanced**.

   Note: Each tab represents a step in the workflow to create a new container instance.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** drop-down list, select **ContainerCompute**.
   3. In the **Container name** text box, enter **manualcompute**.
   4. In the **Region** drop-down list, select **(US) East US**.
   5. In the **Image source** section, select **Azure Container Registry**.
   6. In the **Registry** drop-down list, select the **Azure Container Registry** resource that you created earlier in this lab.
   7. In the **Image** drop-down list, select **ipcheck**.
   8. In the **Image tag** drop-down list, select **latest**.
   9. Select **Review + Create**.
8. From the **Review + Create** tab, review the selected options.
9. Select **Create** to create the container instance by using your specified configuration.

   Note: Wait for the creation task to complete before moving forward with this lab.

20.2.4.4 Task 4: Validate that the container instance ran successfully

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ContainerCompute** resource group that you created earlier in this lab.
3. From the **ContainerCompute** blade, select the **manualcompute** container instance that you created earlier in this lab.
4. From the **Container Instance** blade, find the **Settings** section, and then select the **Containers** link.
5. In the **Containers** section, find the list of **Events**.
6. Select the **Logs** tab, and then find the text logs from the container instance.

   Note: You can also optionally find the **Events** and **Logs** from the **managedcompute** container instance.

   Note: After the application finishes running, the container terminates because it has completed its work. For the manually created container instance, you indicated that a successful exit was acceptable, so the container ran once. The automatically created instance didn't offer this option, and it assumes the container should always be running, so you'll notice repeated restarts of the container.

20.2.4.5 Review

In this exercise, you used multiple methods to deploy a container image to an Azure container instance. By using the manual method, you were also able to customize the deployment further and to run task-based applications as part of a container run.

20.2.5 Exercise 4: Clean up your subscription

20.2.5.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign () and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

    1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

       Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice Cloud Shell configuration options, this is most likely because you are using an existing subscription with this course's labs. The labs are written with the presumption that you are using a new subscription.

20.2.5.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ContainerCompute** resource group:

   ```
   az group delete --name ContainerCompute --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

20.2.5.3 Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

20.2.5.4 Review

20.3 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

20.4 lab: az204Title: 'Lab 05: Deploying compute workloads by using images and containers' az204Module: 'Module 05: Implement IaaS solutions'

21 Lab 05: Deploying compute workloads by using images and containers

22 Student lab manual

22.1 Lab scenario

Your organization is seeking a way to automatically create virtual machines (VMs) to run tasks and immediately terminate. You're tasked with evaluating multiple compute services in Microsoft Azure and determining which service can help you automatically create VMs and install custom software on those machines. As a proof of concept, you have decided to try creating VMs from built-in images and container images so that you can compare the two solutions. To keep your proof of concept simple, you'll create a special "IP check" application written in .NET that you'll automatically deploy to your machines. Your proof of concept will evaluate the Azure Container Instances and Azure Virtual Machines services.

22.2 Objectives

After you complete this lab, you will be able to:

- Create a VM by using the Azure Command-Line Interface (CLI).
- Deploy a Docker container image to Azure Container Registry.
- Deploy a container from a container image in Container Registry by using Container Instances.

22.3 Lab setup

- Estimated time: **45 minutes**

22.4 Instructions

22.4.1 Before you start

22.4.1.1 Sign in to the lab VM

Ensure that you're signed in to your Windows 10 VM by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

22.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer

22.4.2 Exercise 1: Create a VM by using the Azure CLI

22.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

22.4.2.2 Task 2: Create a resource group

1. Create a new resource group with the following details:
    1. Name: **ContainerCompute**
    2. Location: **(US) East US**

       Note: Wait for the creation task to complete before moving forward with this lab.

22.4.2.3 Task 3: Open Azure Cloud Shell

1. Open a new Cloud Shell instance in the Azure portal.
2. If the Cloud Shell is not already configured, configure the shell for Bash by using the default settings.

22.4.2.4 Task 4: Use the Azure CLI commands

1. Use the **az** command with the **--help** flag to find a list of subgroups and commands at the root level of the CLI.
2. Use the **az vm** command with the **--help** flag to find a list of subgroups and commands for Azure Virtual Machines:
3. Use the **az vm create** command with the **--help** flag to find a list of arguments and examples for the **Create Virtual Machine** command:
4. Use the **az vm create** command to create a new VM with the following settings:
    - Resource group: **ContainerCompute**
    - Name: **quickvm**
    - Image: **Debian**
    - Username: **student**
    - Password: **StudentPa55w.rd**

      Note: Wait for the VM creation process to complete. After the process completes, the command will return a JavaScript Object Notation (JSON) file with details about the machine.
5. Use the **az vm show** command to find a more detailed JSON file that contains various metadata about the newly created VM.

6. Use the **az vm list-ip-addresses** command to list all the IP addresses associated with the VM:

   ```
   az vm list-ip-addresses --resource-group ContainerCompute --name quickvm
   ```

7. Use the **az vm list-ip-addresses** command and the **--query** argument to filter the output to only return the first IP address value:

   ```
   az vm list-ip-addresses --resource-group ContainerCompute --name quickvm --query '[].{ip:virtualMa
   ```

8. Use the following script to store the results of the previous command in a new Bash shell variable named *ipAddress*:

   ```
   ipAddress=$(az vm list-ip-addresses --resource-group ContainerCompute --name quickvm --query '[].{
   ```

9. Use the following script to render the value of the Bash shell variable *ipAddress*:

   ```
   echo $ipAddress
   ```

10. Use the following script to connect to the VM that you created earlier in this lab by using the Secure Shell (SSH) tool and the IP address stored in the Bash shell variable *ipAddress*:

    ```
    ssh student@$ipAddress
    ```

11. During the connection process, you'll receive a warning that the authenticity of the host can't be verified. Continue connecting to the host. Finally, use the password **StudentPa55w.rd** when prompted for credentials

12. After connecting to the VM, use the following command to get information about the machine to ensure that you're connected to the correct VM:

    ```
    uname -a
    ```

13. Use the **exit** command to end your SSH session:

    ```
    exit
    ```

14. Close the Cloud Shell pane.

22.4.2.5 Review

In this exercise, you used Cloud Shell to create a VM as part of an automated script.

22.4.3 Exercise 2: Create a Docker container image and deploy it to Container Registry

22.4.3.1 Task 1: Open the Cloud Shell and editor

1. Open a new Cloud Shell instance in the Azure portal.
2. At the **Cloud Shell** command prompt, change the active directory to ~/**clouddrive**.

   Note: The command to change directory in Bash is **cd *path***.

3. At the **Cloud Shell** command prompt, create a new directory named **ipcheck** in the ~/**clouddrive** directory.

   Note: The command to create a new directory in Linux is **mkdir *directory name***.

4. Change the active directory to ~/**clouddrive/ipcheck**.
5. Use the `dotnet new console --output . --name ipcheck` command to create a new .NET console application in the current directory.
6. Create a new file in the ~/**clouddrive/ipcheck** directory named **Dockerfile**.

   Note: The command to create a new file in Bash is **touch *file name***. The file name **Dockerfile** is case sensitive.

7. Open the embedded graphical editor in the context of the current directory.

   Note: You can open the editor by using the **code .** command or by selecting the editor button.

22.4.3.2 Task 2: Create and test a .NET application

1. In the graphical editor, open the **Program.cs** file and replace its contents with the following code, and then save the file:

```
public class Program
{
    public static void Main(string[] args)
    {
        // Check if network is available
        if (System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())
        {
            System.Console.WriteLine("Current IP Addresses:");

            // Get host entry for current hostname
            string hostname = System.Net.Dns.GetHostName();
            System.Net.IPHostEntry host = System.Net.Dns.GetHostEntry(hostname);

            // Iterate over each IP address and render their values
            foreach(System.Net.IPAddress address in host.AddressList)
            {
                System.Console.WriteLine($"\t{address}");
            }
        }
        else
        {
            System.Console.WriteLine("No Network Connection");
        }
    }
}
```

2. Use the `dotnet run` command at the command prompt to run the application and validate that it finds one or more IP addresses.
3. Open the **Dockerfile** file in the graphical editor, replace its contents with the following code, and then save the file:

```
# Start using the .NET Core 3.1 SDK container image
FROM mcr.microsoft.com/dotnet/sdk:3.1-alpine AS build

# Change current working directory
WORKDIR /app

# Copy existing files from host machine
COPY . ./

# Publish application to the "out" folder
RUN dotnet publish --configuration Release --output out

# Start container by running application DLL
ENTRYPOINT ["dotnet", "out/ipcheck.dll"]
```

4. Close the Cloud Shell pane.

22.4.3.3 Task 3: Create a Container Registry resource

- Create a new container registry with the following details:
  - Name: ***Any globally unique name***
  - Resource group: **ContainerCompute**
  - Location: **East US**
  - SKU: **Basic**

Note: Wait for the creation task to complete before moving on with this lab.

22.4.3.4 Task 4: Open Azure Cloud Shell and store Container Registry metadata

1. Open a new Cloud Shell instance.
2. At the **Cloud Shell** command prompt, use the **az acr list** command to get a list of all container registries in your subscription.
3. Use the following command to output the name of the most recently created container registry:

   ```
   az acr list --query "max_by([], &creationDate).name" --output tsv
   ```

4. Use the following command to save the name of the most recently created container registry in a Bash shell variable named *acrName*:

   ```
   acrName=$(az acr list --query "max_by([], &creationDate).name" --output tsv)
   ```

5. Use the following script to render the value of the Bash shell variable *acrName*:

   ```
   echo $acrName
   ```

22.4.3.5 Task 5: Deploy a Docker container image to Container Registry

1. Change the active directory to ~/**clouddrive/ipcheck**.
2. Use the **dir** command to get the contents of the current directory.

   Note: You'll know that you're in the correct directory if both the **Program.cs** and **Dockerfile** files that you edited earlier in this lab are there.

3. Use the following command to upload the source code to your container registry and build the container image as a Container Registry task:

   ```
   az acr build --registry $acrName --image ipcheck:latest .
   ```

   Note: Wait for the build task to complete before moving forward with this lab.

4. Close the Cloud Shell pane.

22.4.3.6 Task 6: Validate your container image in Container Registry

1. Access the container registry that you created earlier in this lab.
2. Select the **Repositories** link to find your images in the registry.
3. Proceed through the **Images** and **Tags** blades to find the metadata associated with the **ipcheck** image with the **latest** tag.

   Note: You can also select the **Run ID** link to find the build task metadata.

22.4.3.7 Review

In this exercise, you created a .NET console application to display a machine's current IP address. You then added the **Dockerfile** file to the application so that it could be converted into a Docker container image. Finally, you deployed the container image to Container Registry.

22.4.4 Exercise 3: Deploy an Azure container instance

22.4.4.1 Task 1: Enable the admin user in Container Registry

1. Access the container registry that you created earlier in this lab.
2. Select **Update** to find the settings for the container registry.
3. **Enable** the **Admin User**, **Save** your changes, and then close the **Update container registry** blade.

22.4.4.2 Task 2: Automatically deploy a container image to an Azure container instance

1. Select the **Repositories** link to find your images in the registry.
2. Select the **ipcheck** image, and then find the **latest** tag for that image.
3. Right-click the **latest** tag or activate the shortcut menu for the **ipcheck** container image to **"run"** a new Azure container instance with the following settings:
   - Container name: **managedcompute**
   - OS type: **Linux**
   - Resource group: **ContainerCompute**
   - Location: **East US**
   - Number of cores: **2**
   - Memory (GB): **4**
   - Public IP address: **No**

     Note: Wait for the creation task to complete before moving on with this lab.

22.4.4.3 Task 3: Manually deploy a container image to Container Instances

1. Create a new container instance with the following information :
   - Resource group: **ContainerCompute**
   - Container name: **manualcompute**
   - Region: **East US**
   - Image source: **Azure Container Registry**
   - Registry: ***Azure Container Registery resource created earlier in the lab**
   - Image: **ipcheck**
   - Image tag: **latest**

     Note: Wait for the creation task to complete before moving forward with this lab.

22.4.4.4 Task 4: Validate that the container instance ran successfully

1. Access the **manualcompute** container instance that you created earlier in this lab.
2. Select the **Containers** link to get a list of the current containers that are running.
3. Find the contents of the **Events** list for the container instance that ran your **ipcheck** application.
4. Select the **Logs** tab, and then find the text logs from the container instance.

   Note: You can also optionally find the **Events** and **Logs** from the **managedcompute** container instance.

22.4.4.5 Review

In this exercise, you used multiple methods to deploy a container image to an Azure container instance. By using the manual method, you were also able to customize the deployment further and run task-based applications as part of a container run.

22.4.5 Exercise 4: Clean up your subscription

22.4.5.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

22.4.5.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ContainerCompute** resource group:

   ```
   az group delete --name ContainerCompute --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

22.4.5.3 Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

22.4.5.4 Review

22.5 In this exercise, you cleaned up your subscription by removing the resource groups that were used in this lab.

22.6 lab: az204Title: 'Lab 06: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs' az020Title: 'Lab 06: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs' az204Module: 'Module 06: Implement user authentication and authorization' az020Module: 'Module 06: Implement user authentication and authorization' type: 'Answer Key'

23 Lab 06: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs

24 Student lab answer key

24.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

24.2 Instructions

24.2.1 Before you start

24.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

24.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Visual Studio Code

24.2.2 Exercise 1: Create an Azure Active Directory (Azure AD) application registration

24.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

24.2.2.2 Task 2: Create an application registration

1. In the Azure portal's navigation pane, select **All services**.
2. From the **All services** blade, select **Azure Active Directory**.
3. From the **Azure Active Directory** blade, select **App registrations** in the **Manage** section.
4. In the **App registrations** section, select **New registration**.
5. In the **Register an application** section, perform the following actions:
   1. In the **Name** text box, enter **graphapp**.
   2. In the **Supported account types** list, select the **Accounts in this organizational directory only (Default Directory only - Single tenant)** check box.
   3. In the **Redirect URI** drop-down list, select **Public client/native (mobile & desktop)**.
   4. In the **Redirect URI** text box, enter **http://localhost**.
   5. Select **Register**.

24.2.2.3 Task 3: Enable the default client type

1. In the **graphapp** application registration blade, select **Authentication** in the **Manage** section.
2. In the **Authentication** section, perform the following actions:
   1. In the **Advanced settings - Allow public client flows** subsection, select **Yes**.
   2. Select **Save**.

24.2.2.4 Task 4: Record unique identifiers

1. On the **graphapp** application registration blade, select **Overview**.
2. In the **Overview** section, find and record the value of the **Application (client) ID** text box. You'll use this value later in the lab.
3. In the **Overview** section, find and record the value of the **Directory (tenant) ID** text box. You'll use this value later in the lab.

24.2.2.5 Review

In this exercise, you created a new application registration and recorded important values that you'll need later in the lab.

24.2.3 Exercise 2: Obtain a token by using the MSAL.NET library

24.2.3.1 Task 1: Create a .NET project

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\06\Starter\GraphClient**, and then select **Select Folder**.

4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to create a new .NET project named **GraphClient** in the current folder:

   ```
   dotnet new console --name GraphClient --output .
   ```

   Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.
6. At the command prompt, enter the following command, and then select Enter to import version 4.7.1 of **Microsoft.Identity.Client** from NuGet:

   ```
   dotnet add package Microsoft.Identity.Client --version 4.7.1
   ```

   Note: The **dotnet add package** command will add the **Microsoft.Identity.Client** package from NuGet. For more information, go to Microsoft.Identity.Client.
7. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

   ```
   dotnet build
   ```
8. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

24.2.3.2 Task 2: Modify the Program class

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, delete all the code in the existing file.
3. Add the following line of code to import the **Microsoft.Identity.Client** namespace from the **Microsoft.Identity.Client** package imported from NuGet:

   ```
   using Microsoft.Identity.Client;
   ```
4. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

   ```
   using System;
   using System.Collections.Generic;
   using System.Threading.Tasks;
   ```
5. Enter the following code to create a new **Program** class:

   ```
   public class Program
   {
   }
   ```
6. In the **Program** class, enter the following code to create a new asynchronous **Main** method:

   ```
   public static async Task Main(string[] args)
   {
   }
   ```
7. In the **Program** class, enter the following line of code to create a new string constant named **_clientId**:

   ```
   private const string _clientId = "";
   ```
8. Update the **_clientId** string constant by setting its value to the **Application (client) ID** that you recorded earlier in this lab.
9. In the **Program** class, enter the following line of code to create a new string constant named **_tenantId**:

   ```
   private const string _tenantId = "";
   ```
10. Update the **_tenantId** string constant by setting its value to the **Directory (tenant) ID** that you recorded earlier in this lab.
11. Observe the **Program.cs** file, which should now include:

```
using Microsoft.Identity.Client;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

public class Program
{
    private const string _clientId = "<app-reg-client-id>";
    private const string _tenantId = "<aad-tenant-id>";

    public static async Task Main(string[] args)
    {
    }
}
```

24.2.3.3 Task 3: Obtain a Microsoft Authentication Library (MSAL) token

1. In the **Main** method, add the following line of code to create a new variable named *app* of type **IPublicClientApplication**:

   ```
   IPublicClientApplication app;
   ```

2. In the **Main** method, perform the following actions to build a public client application instance by using the static **PublicClientApplicationBuilder** class, and then store it in the *app* variable:

   1. Add the following line of code to access the static **PublicClientApplicationBuilder** class:

      ```
      PublicClientApplicationBuilder
      ```

   2. Update the previous line of code by adding another line of code to use the **Create()** method of the **PublicClientApplicationBuilder** class, passing in the *_clientId* variable as a parameter:

      ```
      PublicClientApplicationBuilder
          .Create(_clientId)
      ```

   3. Update the previous line of code by adding another line of code to use the **WithAuthority()** method of the base **AbstractApplicationBuilder<>** class, passing in the enumeration value **AzureCloudInstance.AzurePublic** and the *_tenantId* variable as parameters:

      ```
      PublicClientApplicationBuilder
          .Create(_clientId)
          .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
      ```

   4. Update the previous line of code by adding another line of code to use the **WithRedirectUri()** method of the base **AbstractApplicationBuilder<>** class, passing in a string value of **http://localhost**:

      ```
      PublicClientApplicationBuilder
          .Create(_clientId)
          .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
          .WithRedirectUri("http://localhost")
      ```

   5. Update the previous line of code by adding another line of code to use the **Build()** method of the **PublicClientApplication** class:

      ```
      PublicClientApplicationBuilder
          .Create(_clientId)
          .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
          .WithRedirectUri("http://localhost")
          .Build();
      ```

   6. Update the previous line of code by adding more code to store the result of the expression in the *app* variable:

      ```
      app = PublicClientApplicationBuilder
          .Create(_clientId)
          .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
          .WithRedirectUri("http://localhost")
      ```

```
        .Build();
```

3. In the **Main** method, add the following line of code to create a new generic string **List<>** with a single value of **user.read**:

```
List<string> scopes = new List<string>
{
    "user.read"
};
```

4. In the **Main** method, add the following line of code to create a new variable named *result* of type **AuthenticationResult**:

```
AuthenticationResult result;
```

5. In the **Main** method, perform the following actions to acquire a token interactively and store the output in the *result* variable:

    1. Add the following line of code to access the *app* variable:

```
app
```

    2. Update the previous line of code by adding another line of code to use the **AcquireTokenInteractive()** method of the **IPublicClientApplicationBuilder** interface, passing in the *scopes* variable as a parameter:

```
app
    .AcquireTokenInteractive(scopes)
```

    3. Update the previous line of code by adding another line of code to use the **ExecuteAsync()** method of the **AbstractAcquireTokenParameterBuilder** class:

```
app
    .AcquireTokenInteractive(scopes)
    .ExecuteAsync();
```

    4. Update the previous line of code by adding more code to process the expression asynchronously by using the **await** keyword:

```
await app
    .AcquireTokenInteractive(scopes)
    .ExecuteAsync();
```

    5. Update the previous line of code by adding more code to store the result of the expression in the *result* variable:

```
result = await app
    .AcquireTokenInteractive(scopes)
    .ExecuteAsync();
```

6. In the **Main** method, add the following line of code to use the **Console.WriteLine** method to render the value of the **AuthenticationResult.AccessToken** member to the console:

```
Console.WriteLine($"Token:\t{result.AccessToken}");
```

7. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    IPublicClientApplication app;

    app = PublicClientApplicationBuilder
        .Create(_clientId)
        .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
        .WithRedirectUri("http://localhost")
        .Build();

    List<string> scopes = new List<string>
    {
        "user.read"
```

```
    };

    AuthenticationResult result;

    result = await app
        .AcquireTokenInteractive(scopes)
        .ExecuteAsync();

    Console.WriteLine($"Token:\t{result.AccessToken}");
}
```

8. Save the **Program.cs** file.

24.2.3.4 Task 4: Test the updated application

1. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

   ```
   dotnet run
   ```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\06\Solution\GraphClient** folder.
3. The running console application will automatically open an instance of the default browser.
4. In the open browser window, perform the following actions:
   1. Enter the email address for your Microsoft account, and then select **Next**.
   2. Enter the password for your Microsoft account, and then select **Sign in**.

      Note: You might have the option to select an existing Microsoft account as opposed to signing in again.
5. The browser window will automatically open the **Permissions requested** webpage. On this webpage, perform the following actions:
   1. Review the requested permissions.
   2. Select **Accept**.
6. Return to the currently running Visual Studio Code application.
7. Observe the token rendered in the output from the currently running console application.
8. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

24.2.3.5 Review

In this exercise, you acquired a token from the Microsoft identity platform by using the MSAL.NET library.

24.2.4 Exercise 3: Query Microsoft Graph by using the .NET SDK

24.2.4.1 Task 1: Import the Microsoft Graph SDK from NuGet

1. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the command prompt, enter the following command, and then select Enter to import version 1.21.0 of **Microsoft.Graph** from NuGet:

   ```
   dotnet add package Microsoft.Graph --version 1.21.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.Graph** package from NuGet. For more information, go to Microsoft.Graph.
3. At the command prompt, enter the following command, and then select Enter to import version 1.0.0-preview.2 of **Microsoft.Graph.Auth** from NuGet:

```
dotnet add package Microsoft.Graph.Auth --version 1.0.0-preview.2
```

Note: The **dotnet add package** command will add the **Microsoft.Graph.Auth** package from NuGet. For more information, go to Microsoft.Graph.Auth.

4. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

```
dotnet build
```

5. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

24.2.4.2 Task 2: Modify the Program class

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, add the following line of code to import the **Microsoft.Graph** namespace from the **Microsoft.Graph** package imported from NuGet:

```
using Microsoft.Graph;
```

3. Add the following line of code to import the **Microsoft.Graph.Auth** namespace from the **Microsoft.Graph.Auth** package imported from NuGet:

```
using Microsoft.Graph.Auth;
```

4. Observe the **Program.cs** file, which should now include the following **using** directives:

```
using Microsoft.Graph;
using Microsoft.Graph.Auth;
using Microsoft.Identity.Client;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

24.2.4.3 Task 3: Use the Microsoft Graph SDK to query user profile information

1. Within the **Main** method, perform the following actions to remove unnecessary code:
    1. Delete the following line of code:

    ```
    AuthenticationResult result;
    ```

    2. Delete the following block of code:

    ```
    result = await app
            .AcquireTokenInteractive(scopes)
            .ExecuteAsync();
    ```

    3. Delete the following line of code:

    ```
    Console.WriteLine($"Token:\t{result.AccessToken}");
    ```

2. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    IPublicClientApplication app;

    app = PublicClientApplicationBuilder
        .Create(_clientId)
        .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
        .WithRedirectUri("http://localhost")
        .Build();

    List<string> scopes = new List<string>
    {
        "user.read"
    };
```

```
}
```

3. In the **Main** method, add the following line of code to create a new variable named *provider* of type **DeviceCodeProvider** that passes in the variables *app* and *scopes* as constructor parameters:

```
DeviceCodeProvider provider = new DeviceCodeProvider(app, scopes);
```

4. In the **Main** method, add the following line of code to create a new variable named *client* of type **GraphServiceClient** that passes in the variable *provider* as a constructor parameter:

```
GraphServiceClient client = new GraphServiceClient(provider);
```

5. In the **Main** method, perform the following actions to use the **GraphServiceClient** instance to asynchronously get the response of issuing an HTTP request to the relative **/Me** directory of the REST API:

    1. Add the following line of code to get the **Me** property of the *client* variable:

    ```
    client.Me
    ```

    2. Update the previous line of code by adding another line of code to get an object representing the HTTP request by using the **Request()** method:

    ```
    client.Me
        .Request()
    ```

    3. Update the previous line of code by adding another line of code to issue the request asynchronously by using the **GetAsync()** method:

    ```
    client.Me
        .Request()
        .GetAsync()
    ```

    4. Update the previous line of code by adding more code to process the expression asynchronously by using the **await** keyword:

    ```
    await client.Me
        .Request()
        .GetAsync()
    ```

    5. Update the previous line of code by adding more code to store the result of the expression in a new variable named *myProfile* of type **User**:

    ```
    User myProfile = await client.Me
        .Request()
        .GetAsync();
    ```

6. In the **Main** method, add the following line of code to use the **Console.WriteLine** method to render the value of the **User.DisplayName** member to the console:

```
Console.WriteLine($"Name:\t{myProfile.DisplayName}");
```

7. In the **Main** method, add the following line of code to use the **Console.WriteLine** method to render the value of the **User.Id** member to the console:

```
Console.WriteLine($"AAD Id:\t{myProfile.Id}");
```

8. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    IPublicClientApplication app;

    app = PublicClientApplicationBuilder
        .Create(_clientId)
        .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
        .WithRedirectUri("http://localhost")
        .Build();

    List<string> scopes = new List<string>
    {
```

```
            "user.read"
        };

        DeviceCodeProvider provider = new DeviceCodeProvider(app, scopes);

        GraphServiceClient client = new GraphServiceClient(provider);

        User myProfile = await client.Me
            .Request()
            .GetAsync();

        Console.WriteLine($"Name:\t{myProfile.DisplayName}");
        Console.WriteLine($"AAD Id:\t{myProfile.Id}");
    }
```

9. Save the **Program.cs** file.

24.2.4.4 Task 4: Test the updated application

1. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

   ```
   dotnet run
   ```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\06\Solution\GraphClient** folder.
3. Observe the message in the output from the currently running console application. Record the value of the code in the message. You'll use this value later in the lab.
4. On the taskbar, select the **Microsoft Edge** icon.
5. In the open browser window, go to https://microsoft.com/devicelogin.
6. On the **Enter code** webpage, perform the following actions:
   1. In the **Code** text box, enter the value of the code that you copied earlier in the lab.
   2. Select **Next**.
7. On the login webpage, perform the following actions:
   1. Enter the email address for your Microsoft account, and then select **Next**.
   2. Enter the password for your Microsoft account, and then select **Sign in**.

      Note: You might have the option to select an existing Microsoft account as opposed to signing in again.
8. Return to the currently running Visual Studio Code application.
9. Observe the output from the Microsoft Graph request in the currently running console application.
10. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

24.2.4.5 Review

In this exercise, you queried Microsoft Graph by using the SDK and MSAL-based authentication.

24.2.5 Exercise 4: Clean up your subscription

24.2.5.1 Task 1: Delete the application registration in Azure AD

1. Return to the browser window with the Azure portal.
2. In the Azure portal's navigation pane, select **All services**.
3. From the **All services** blade, select **Azure Active Directory**.

4. From the **Azure Active Directory** blade, select **App registrations** in the **Manage** section.
5. In the **App registrations** section, select the **graphapp** Azure AD application registration that you created earlier in this lab.
6. In the **graphapp** section, perform the following actions:
    1. Select **Delete**.
    2. In the confirmation pop-up dialog box, select **Yes**.

24.2.5.2 Task 2: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

24.2.5.3 Review

24.3 In this exercise, you cleaned up your subscription by removing the application registration used in this lab.

24.4 lab: az204Title: 'Lab 06: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs' az020Title: 'Lab 06: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs' az204Module: 'Module 06: Implement user authentication and authorization' az020Module: 'Module 06: Implement user authentication and authorization'

25 Lab 06: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs

26 Student lab manual

26.1 Lab scenario

As a new employee at your company, you signed in to your Microsoft 365 applications for the first time and discovered that your profile information isn't accurate. You also noticed that the name and profile picture when you sign in aren't correct. Rather than change these values manually, you have decided that this is a good opportunity to learn the Microsoft identity platform and how you can use different libraries such as the Microsoft Authentication Library (MSAL) and the Microsoft Graph SDK to change these values in a programmatic manner.

26.2 Objectives

After you complete this lab, you will be able to:

- Create a new application registration in Azure Active Directory (Azure AD).
- Use the MSAL.NET library to implement the interactive authentication flow.
- Obtain a token from the Microsoft identity platform by using the MSAL.NET library.
- Query Microsoft Graph by using the Microsoft Graph SDK and the device code flow.

26.3 Lab setup

- Estimated time: **45 minutes**

26.4 Instructions

26.4.1 Before you start

26.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

26.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Visual Studio Code

26.4.2 Exercise 1: Create an Azure AD application registration

26.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

26.4.2.2 Task 2: Create an application registration

1. Create a new Azure AD application registration with the following details:
   - Name: **graphapp**
   - Supported account types: **Accounts in this organizational directory only (Default Directory only - Single tenant)**
   - Redirect URI: **Public client/native (mobile & desktop) http://localhost**

     Note: Wait for Azure to finish creating the registration before you move forward with the lab. You'll receive a notification when the vault is created.

26.4.2.3 Task 3: Enable the default client type

1. Find the **Authentication** section of the **graphapp** application registration blade, and in the **Advanced settings - Allow public client flows** subsection, select **Yes**.
2. Save your changes.

26.4.2.4 Task 4: Record unique identifiers

1. Browse to **Overview** of the **graphapp** application registration blade.
2. Find and record the value of the **Application (client) ID** text box. You'll use this value later in the lab.
3. Find and record the value of the **Directory (tenant) ID** text box. You'll use this value later in the lab.

26.4.2.5 Review

In this exercise, you created a new application registration and recorded important values that you'll need later in the lab.

26.4.3 Exercise 2: Obtain a token by using the MSAL.NET library

26.4.3.1 Task 1: Create a .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\06\Starter\GraphClient** folder.
2. Using a terminal, create a new .NET project named **GraphClient** in the current folder:

   ```
   dotnet new console --name GraphClient --output .
   ```

   Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.
3. Using the same terminal, import version 4.7.1 of **Microsoft.Identity.Client** from NuGet:

```
dotnet add package Microsoft.Identity.Client --version 4.7.1
```

Note: The **dotnet add package** command will add the **Microsoft.Identity.Client** package from NuGet. For more information, go to Microsoft.Identity.Client.

4. Using the same terminal, build the .NET web application:

```
dotnet build
```

5. Close the current terminal.

26.4.3.2 Task 2: Modify the Program class

1. Open the **Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that will be referenced by the application:

```
using Microsoft.Identity.Client;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

4. Create a new **Program** class with two constant string properties named **__clientId** and **__tenantId**, and then create an asynchronous **Main** entry point method:

```
public class Program
{
    private const string _clientId = "<app-reg-client-id>";
    private const string _tenantId = "<aad-tenant-id>";

    public static async Task Main(string[] args)
    {
    }
}
```

5. Update the **__clientId** string constant by setting its value to the **Application (client) ID** that you recorded earlier in this lab.
6. Update the **__tenantId** string constant by setting its value to the **Directory (tenant) ID** that you recorded earlier in this lab.

26.4.3.3 Task 3: Obtain an MSAL token

1. In the **Main** method, perform the following actions:
    1. Create a new variable named *app* of type **IPublicClientApplication**.
    2. Add the following block of code to build a public client application instance by using the static **PublicClientApplicationBuilder** class, and then store it in the *app* variable:

```
app = PublicClientApplicationBuilder
    .Create(_clientId)
    .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
    .WithRedirectUri("http://localhost")
    .Build();
```

    3. Add the following block of code to create a new generic string **List<>** with a single value of **user.read**:

```
List<string> scopes = new List<string>
{
    "user.read"
};
```

    4. Create a new variable named *result* of type **AuthenticationResult**.
    5. Add the following block of code to acquire a token interactively and store the output in the *result* variable:

```
result = await app
    .AcquireTokenInteractive(scopes)
    .ExecuteAsync();
```

6. Render the value of the **AuthenticationResult.AccessToken** member to the console:

```
Console.WriteLine($"Token:\t{result.AccessToken}");
```

2. Save the **Program.cs** file.

26.4.3.4 Task 4: Test the updated application

1. Using a terminal, run the .NET console application project:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\06\Solution\GraphClient** folder.

2. The running console application will automatically open an instance of the default browser.
3. In the open browser window, sign in by using your Microsoft account.

> **Note**: You might have the option to select an existing Microsoft account as opposed to signing in again.

4. The browser window will automatically go to the **Permissions requested** webpage. Accept the request for permissions.
5. Close the currently open browser window.
6. In the **Visual Studio Code** window, observe the token render in the output from the currently running console application.
7. Close the current terminal.

26.4.3.5 Review

In this exercise, you acquired a token from the Microsoft identity platform by using the MSAL.NET library.

26.4.4 Exercise 3: Query Microsoft Graph by using the .NET SDK

26.4.4.1 Task 1: Import the Microsoft Graph SDK from NuGet

1. Using a terminal, import version 1.21.0 of **Microsoft.Graph** from NuGet:

```
dotnet add package Microsoft.Graph --version 1.21.0
```

> **Note**: The **dotnet add package** command will add the **Microsoft.Graph** package from NuGet. For more information, go to Microsoft.Graph.

2. Using the same terminal, import version 1.0.0-preview.2 of **Microsoft.Graph.Auth** from NuGet:

```
dotnet add package Microsoft.Graph.Auth --version 1.0.0-preview.2
```

> **Note**: The **dotnet add package** command will add the **Microsoft.Graph.Auth** package from NuGet. For more information, go to Microsoft.Graph.Auth.

3. Using the same terminal, build the .NET web application:

```
dotnet build
```

4. Close the current terminal.

26.4.4.2 Task 2: Modify the Program class

1. Open the **Program.cs** file in Visual Studio Code.
2. Add the following **using** directives for libraries that will be referenced by the application:

```
using Microsoft.Graph;
using Microsoft.Graph.Auth;
```

26.4.4.3 Task 3: Use the Microsoft Graph SDK to query user profile information

1. Within the **Main** method, remove the following block of unnecessary code:

   ```
   AuthenticationResult result;

   result = await app
       .AcquireTokenInteractive(scopes)
       .ExecuteAsync();

   Console.WriteLine($"Token:\t{result.AccessToken}");
   ```

2. In the **Main** method, perform the following actions:

   1. Create a new variable named *provider* of type **DeviceCodeProvider** that passes in the variables *app*, and *scopes* as constructor parameters:

      ```
      DeviceCodeProvider provider = new DeviceCodeProvider(app, scopes);
      ```

   2. Create a new variable named *client* of type **GraphServiceClient** that passes in the variable *provider* as a constructor parameter:

      ```
      GraphServiceClient client = new GraphServiceClient(provider);
      ```

   3. Add the following block of code to use the **GraphServiceClient** instance to asynchronously get the response of issuing an HTTP request to the relative **/Me** directory of the REST API, and then store the result in a new variable named *myProfile* of type **User**:

      ```
      User myProfile = await client.Me
          .Request()
          .GetAsync();
      ```

   4. Render the value of the **User.DisplayName** and **User.Id** members to the console:

      ```
      Console.WriteLine($"Name:\t{myProfile.DisplayName}");
      Console.WriteLine($"AAD Id:\t{myProfile.Id}");
      ```

3. Save the **Program.cs** file.

26.4.4.4 Task 4: Test the updated application

1. Using a terminal, run the .NET console application project:

   ```
   dotnet run
   ```

   > **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\06\Solution\GraphClient** folder.

2. Observe the message in the output from the currently running console application. Record the value of the code in the message. You'll use this value later in the lab.
3. Go to https://microsoft.com/devicelogin, and then enter the code value that you copied earlier in the lab.
4. Sign in by using your Microsoft account.

   > **Note**: You might have the option to select an existing Microsoft account as opposed to signing in again.

5. Close the currently open browser window.
6. In the **Visual Studio Code** window, observe the output from the Microsoft Graph request in the currently running console application.
7. Close the current terminal.

26.4.4.5 Review

In this exercise, you queried Microsoft Graph by using the SDK and MSAL-based authentication.

26.4.5 Exercise 4: Clean up your subscription

26.4.5.1 Task 1: Delete the application registration in Azure AD

1. Access the **graphapp** Azure AD application registration that you created earlier in this lab.
2. Delete the application registration.

26.4.5.2 Task 2: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

26.4.5.3 Review

26.5 In this exercise, you cleaned up your subscription by removing the application registration used in this lab.

26.6 lab: az204Title: 'Lab 07: Access resource secrets more securely across services' az020Title: 'Lab 07: Access resource secrets more securely across services' az204Module: 'Module 07: Implement secure cloud solutions' az020Module: 'Module 07: Implement secure cloud solutions' type: 'Answer Key'

27 Lab 07: Access resource secrets more securely across services

28 Student lab answer key

28.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure user interface (UI) changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

28.2 Instructions

28.2.1 Before you start

28.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

28.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Windows Terminal
- Visual Studio Code

28.2.2 Exercise 1: Create Azure resources

28.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the **Azure portal** (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the **password** for your Microsoft account, and then select **Sign in**.
   Note: If this is your first time signing in to the Azure portal, you will be offered a tour of the portal. If you prefer to skip the tour, select **Get Started** to begin using the portal.

28.2.2.2 Task 2: Create an Azure Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. From the **All services** blade, select **Storage Accounts**.
3. From the **Storage accounts** blade, find your list of Storage instances.
4. From the **Storage accounts** blade, select **New**.
5. Find the tabs from the **Create storage account** blade, such as **Basics**.
   Note: Each tab represents a step in the workflow to create a new storage account. You can select **Review + Create** at any time to skip the remaining tabs.
6. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **ConfidentialStack**, and then select **OK**.
   3. In the **Storage account name** text box, enter **securestor[yourname]**.
   4. In the **Location** drop-down list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** drop-down list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** drop-down list, select **Locally-redundant storage (LRS)**.
   8. Select **Review + Create**.
7. From the **Review + Create** tab, review the options that you selected during the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.
   Note: Wait for the creation task to complete before you move forward with this lab.
9. In the Azure portal's navigation pane, select **All services**.
10. From the **All services** blade, select **Storage Accounts**.
11. From the **Storage accounts** blade, select the **securestor[yourname]** storage account that you created earlier in this lab.
12. From the **Storage account** blade, find the **Settings** section, and then select the **Access keys** link.
13. From the **Access keys** blade, select any one of the keys and record the value in either of the **Connection string** boxes. You'll use this value later in this lab.
    Note: It doesn't matter which connection string you choose. They are interchangeable.

28.2.2.3 Task 3: Create an Azure Key Vault

1. In the Azure portal's navigation pane, select the **Create a resource** link.
2. From the **New** blade, find the **Search the Marketplace** text box above the list of featured services.
3. In the search box, enter **Key Vault**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Key Vault** result.
5. From the **Key Vault** blade, select **Create**.
6. Find the tabs from the **Create key vault** blade, such as **Basics**.
   Note: Each tab represents a step in the workflow to create a new key vault. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Use existing**, and then select **ConfidentialStack** in the list.
   3. In the **Key vault name** text box, enter **securevault[yourname]**.
   4. In the **Region** drop-down list, select the **East US** region.
   5. In the **Pricing tier** drop-down list, select **Standard**.
   6. Select **Review + Create**.
8. From the **Review + Create** tab, review the options that you selected during the previous steps.
9. Select **Create** to create the key vault by using your specified configuration.

Note: Wait for the creation task to complete before you move forward with this lab.

28.2.2.4 Task 4: Create an Azure Functions app

1. In the Azure portal's navigation pane, select the **Create a resource** link.
2. From the **New** blade, find the **Search the Marketplace** text box above the list of featured services.
3. In the search box, enter **Function**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Function App** result.
5. From the **Function App** blade, select **Create**.
6. Find the tabs from the **Function App** blade, such as **Basics**.
   Note: Each tab represents a step in the workflow to create a new function app. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Use existing**, and then select **ConfidentialStack** in the list.
   3. In the **Function app name** text box, enter **securefunc**[**yourname**].
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET**.
   6. In the **Version** drop-down list, select **3.1**.
   7. In the **Region** drop-down list, select the **East US** region.
   8. Select **Next: Hosting**.
8. From the **Hosting** tab, perform the following actions:
   1. In the **Operating System** section, select **Linux**.
   2. In the **Storage account** drop-down list, select the **securestor**[**yourname**] storage account that you created earlier in this lab.
   3. In the **Plan type** drop-down list, select the **Consumption (Serverless)** option.
   4. Select **Review + Create**.
9. From the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the function app by using your specified configuration.
    Note: Wait for the creation task to complete before you move forward with this lab.

Review: In this exercise, you created all the resources that you'll use for this lab.

28.2.3 Exercise 2: Configure secrets and identities

28.2.3.1 Task 1: Configure a system-assigned managed service identity

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.
3. From the **ConfidentialStack** blade, select the **securefunc**[**yourname**] function app that you created earlier in this lab.
4. From the **App Service** blade, select the **Identity** option from the **Settings** section.
5. From the **Identity** pane, find the **System assigned** tab, and then perform the following actions:
   1. In the **Status** section, select **On**, and then select **Save**.
   2. In the confirmation dialog box, select **Yes**.
   Note: Wait for the system-assigned managed identity to be created before you move forward with this lab.

28.2.3.2 Task 2: Create a Key Vault secret

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.
3. From the **ConfidentialStack** blade, select the **securevault**[**yourname**] key vault that you created earlier in this lab.
4. From the **Key Vault** blade, select the **Secrets** link in the **Settings** section.
5. In the **Secrets** pane, select **Generate/Import**.
6. From the **Create a secret** blade, perform the following actions:
   1. In the **Upload options** drop-down list, select **Manual**.
   2. In the **Name** text box, enter **storagecredentials**.

3. In the **Value** text box, enter the storage account connection string that you recorded earlier in this lab.
4. Leave the **Content Type** text box set to its default value.
5. Leave the **Set activation date** text box set to its default value.
6. Leave the **Set expiration date** text box set to its default value.
7. In the **Enabled** section, select **Yes**, and then select **Create**.
   Note: Wait for the secret to be created before you move forward with this lab.

7. Return to the Secrets pane, and then select the **storagecredentials** item in the list.
8. In the Versions pane, select the latest version of the **storagecredentials** secret.
9. In the Secret Version pane, perform the following actions:
   1. Find the metadata for the latest version of the secret.
   2. Select **Show secret value** to find the value of the secret.
   3. Record the value of the **Secret Identifier** text box because you'll use this later in the lab.
      Note: You are recording the value of the **Secret Identifier** text box, not the **Secret Value** text box.

28.2.3.3 Task 3: Configure a Key Vault access policy

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.
3. From the **ConfidentialStack** blade, select the **securevault**[**yourname**] key vault that you created earlier in this lab.
4. From the **Key Vault** blade, select the **Access policies** link in the **Settings** section.
5. In the Access policies pane, select **Add Access Policy**.
6. From the **Add access policy** blade, perform the following actions:
   1. Select the **Select principal** link.
   2. From the **Principal** blade, find and then select the service principal named **securefunc**[**yourname**], and then select **Select**.
      Note: The system-assigned managed identity you created earlier in this lab will have the same name as the Azure Function resource.
   3. Leave the **Key permissions** list set to its default value.
   4. In the **Secret permissions** drop-down list, select the **GET** permission.
   5. Leave the **Certificate permissions** list set to its default value.
   6. Leave the **Authorized application** text box set to its default value.
   7. Select **Add**.
7. Back in the Access policies pane, select **Save**.
   Note: Wait for your changes to the access policies to save before you move forward with this lab.

28.2.3.4 Task 4: Create a Key Vault-derived application setting

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.
3. From the **ConfidentialStack** blade, select the **securefunc**[**yourname**] function app that you created earlier in this lab.
4. From the **App Service** blade, select the **Configuration** option from the **Settings** section.
5. From the **Configuration** pane, perform the following actions:
   1. Select the **Application settings** tab, and then select **New application setting**.
   2. In the **Add/Edit application setting** pop-up window, in the **Name** text box, enter **StorageConnectionString**.
   3. In the **Value** text box, construct a value by using the following syntax: `@Microsoft.KeyVault(SecretUri=*Secret Identifier*)`
      Note: You'll need to build a reference to your ***Secret Identifier*** by using the above syntax.
      For example, if your secret identifier is `https://securevaultstudent.vault.azure.net/secrets/storagec`, your value would be `@Microsoft.KeyVault(SecretUri=https://securevaultstudent.vault.azure.net/s`
   4. Leave the **deployment slot setting** text box set to its default value.
   5. Select **OK** to close the pop-up window and return to the **Configuration** section.
   6. Select **Save** from the blade to save your settings.
   7. In the **Save Changes** confirmation pop-up dialog box, select **Continue**.

Note: Wait for your application settings to save before you move forward with the lab.

Review: In this exercise, you created a system-assigned managed service identity for your function app and then gave that identity the appropriate permissions to get the value of a secret in your key vault. Finally, you created a secret that you referenced within your function app's configuration settings.

28.2.4 Exercise 3: Build an Azure Functions app

28.2.4.1 Task 1: Initialize a function project

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\07\Starter\func
   ```

3. At the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new local Functions project in the current directory using the **dotnet** runtime:

   ```
   func init --worker-runtime dotnet --force
   ```

   Note: You can review the documentation to [create a new project][azure-functions-core-tools-new-project] using the **Azure Functions Core Tools**.

4. Enter the following command, and then select Enter to **build** the .NET Core 3.1 project:

   ```
   dotnet build
   ```

28.2.4.2 Task 2: Create an HTTP-triggered function

1. Still in the open command prompt, enter the following command, and then select Enter to use the **Azure Functions Core Tools** to create a new function named **FileParser** using the **HTTP trigger** template:

   ```
   func new --template "HTTP trigger" --name "FileParser"
   ```

   Note: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.

2. Close the currently running **Windows Terminal** application.

28.2.4.3 Task 3: Configure and read an application setting

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\07\Starter\func**, and then select **Select Folder**.
4. In the Explorer pane of the **Visual Studio Code** window, open the **local.settings.json** file.
5. Observe the current value of the **Values** object:

   ```
   "Values": {
       "AzureWebJobsStorage": "UseDevelopmentStorage=true",
       "FUNCTIONS_WORKER_RUNTIME": "dotnet"
   }
   ```

6. Update the value of the **Values** object by adding a new setting named **StorageConnectionString** and setting it to a string value of **[TEST VALUE]**:

   ```
   "Values": {
       "AzureWebJobsStorage": "UseDevelopmentStorage=true",
       "FUNCTIONS_WORKER_RUNTIME": "dotnet",
       "StorageConnectionString": "[TEST VALUE]"
   }
   ```

7. The **local.settings.json** file should now include:

```
{
    "IsEncrypted": false,
    "Values": {
        "AzureWebJobsStorage": "UseDevelopmentStorage=true",
        "FUNCTIONS_WORKER_RUNTIME": "dotnet",
        "StorageConnectionString": "[TEST VALUE]"
    }
}
```

8. In the Explorer pane of the **Visual Studio Code** window, open the **FileParser.cs** file.
9. In the code editor, observe the example implementation:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace func
{
    public static class FileParser
    {
        [FunctionName("FileParser")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest re
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;

            string responseMessage = string.IsNullOrEmpty(name)
                ? "This HTTP triggered function executed successfully. Pass a name in the query st
                : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);
        }
    }
}
```

10. Delete all of the content within the **FileParser.cs** file.
11. Add the following lines of code to add **using directives** for the **Microsoft.AspNetCore.Mvc**, **Microsoft.Azure.WebJobs**, **Microsoft.AspNetCore.Http**, **System**, and **System.Threading.Tasks** namespaces:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using System;
using System.Threading.Tasks;
```

12. Create a new **public static** class named **FileParser**:

```
public static class FileParser
{ }
```

13. Observe the **FileParser.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using System;
using System.Threading.Tasks;

public static class FileParser
{ }
```

14. Within the **FileParser** class, add the following code block to create a new **public static** *asynchronous* method named **Run** that returns a variable of type **Task<IActionResult>** and that also takes in a variable of type **HttpRequest** named *request*:

```
public static async Task<IActionResult> Run(
    HttpRequest request)
{ }
```

15. Add the following code to append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **FileParser**:

```
[FunctionName("FileParser")]
public static async Task<IActionResult> Run(
    HttpRequest request)
{ }
```

16. Add the following code to append an attribute to the **request** paramter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **GET**:

```
[FunctionName("FileParser")]
public static async Task<IActionResult> Run(
    [HttpTrigger("GET")] HttpRequest request)
{ }
```

17. Observe the **FileParser.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using System;
using System.Threading.Tasks;

public static class FileParser
{
    [FunctionName("FileParser")]
    public static async Task<IActionResult> Run(
        [HttpTrigger("GET")] HttpRequest request)
    { }
}
```

18. In the **Run** method, enter the following line of code to retrieve the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method and storing the result in a **string** variable named **connectionString**:

```
string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
```

19. Enter the following line of code to return the value of the **connectionString** variable as the HTTP response:

```
return new OkObjectResult(connectionString);
```

20. Observe the **FileParser.cs** file again, which should now include:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using System;
```

```
using System.Threading.Tasks;

public static class FileParser
{
    [FunctionName("FileParser")]
    public static async Task<IActionResult> Run(
        [HttpTrigger("GET")] HttpRequest request)
    {
        string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
        return new OkObjectResult(connectionString);
    }
}
```

21. Select **Save** to save your changes to the **FileParser.cs** file.

28.2.4.4 Task 4: Validate the local function

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\07\Starter\func
   ```

3. At the open command prompt, enter the following command, and then select Enter to run the function app project:

   ```
   func start --build
   ```

   > **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. On the taskbar, select the **Windows Terminal** icon again to open a new instance of the **Windows Terminal** application.
5. When you receive the open command prompt, enter the following command, and then select Enter to start the **httprepl** tool setting the base Uniform Resource Identifier (URI) to `http://localhost:7071`:

   ```
   httprepl http://localhost:7071
   ```

   > **Note**: An error message is displayed by the httprepl tool. This message occurs because the tool is searching for a Swagger definition file to use to ”traverse” the API. Because your functio projectp does not produce a Swagger definition file, you’ll need to traverse the API manually.

6. When you receive the tool prompt, enter the following command, and then select Enter to browse to the relative **api** directory:

   ```
   cd api
   ```

7. Enter the following command, and then select Enter to browse to the relative **fileparser** directory:

   ```
   cd fileparser
   ```

8. Enter the following command, and then select Enter to run the **get** command:

   ```
   get
   ```

9. Observe the [**TEST VALUE**] value of the **StorageConnectionString** being returned as the result of the HTTP request:

   ```
   HTTP/1.1 200 OK
   Content-Type: text/plain; charset=utf-8
   Date: Tue, 01 Sep 2020 23:35:39 GMT
   Server: Kestrel
   Transfer-Encoding: chunked

   [TEST VALUE]
   ```

10. Enter the following command, and then select Enter to exit the **httprepl** application:

```
exit
```

11. Close all currently running instances of the **Windows Terminal** application.

28.2.4.5 Task 5: Deploy using the Azure Functions Core Tools

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\07\Starter\func
   ```

3. At the open command prompt, enter the following command, and then select Enter to log in to the Azure Command-Line Interface (CLI):

   ```
   az login
   ```

4. In the **Microsoft Edge** browser window, perform the following actions:
   1. Enter the email address for your Microsoft account, and then select **Next**.
   2. Enter the password for your Microsoft account, and then select **Sign in**.
5. Return to the currently open **Windows Terminal** window. Wait for the sign-in process to finish.
6. Enter the following command, and then select Enter to publish the function app project:

   ```
   func azure functionapp publish <function-app-name>
   ```

   > **Note**: As an example, if your **Function App name** is **securefuncstudent**, your command would be `func azure functionapp publish securefuncstudent`. You can review the documentation to [publish the local function app project][azure-functions-core-tools-publish-azure] using the **Azure Functions Core Tools**.

7. Wait for the deployment to finalize before you move forward with the lab.
8. Close the currently running **Windows Terminal** application.

28.2.4.6 Task 6: Test the Key Vault-derived application setting

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. In the Azure portal's navigation pane, select the **Resource groups** link.
4. On the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.
5. On the **ConfidentialStack** blade, select the **securefunc[yourname]** function app that you created earlier in this lab.
6. From the **App Service** blade, select the **Functions** option from the **Functions** section.
7. In the **Functions** pane, select the the existing **FileParser** function.
8. In the **Function** blade, select the **Code + Test** option from the **Developer** section.
9. In the function editor, select **Test/Run**.
10. In the pop-up dialog box that appears, perform the following actions:
    - In the **HTTP method** list, select **GET**.
11. Select **Run** to test the function.
12. Observe the results of the test run. The result should be your Azure Storage connection string.

> **Review**: In this exercise, you used a service identity to read the value of a secret stored in Key Vault and returned that value as the result of a function app.

28.2.5 Exercise 4: Access Azure Blob Storage data

28.2.5.1 Task 1: Upload a sample storage blob

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. From the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.
3. From the **ConfidentialStack** blade, select the **securestor[yourname]** storage account that you created earlier in this lab.
4. From the **Storage account** blade, select the **Containers** link in the **Blob service** section.

5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up window, perform the following actions:
    1. In the **Name** text box, enter **drop**.
    2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**, and then select **Create**.
7. Return to the **Containers** section, and then select the newly created **drop** container.
8. From the **Container** blade, select **Upload**.
9. In the **Upload blob** pop-up window, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\07\Starter**, select the **records.json** file, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.
    Note: Wait for the blob to upload before you continue with this lab.
10. Return to the **Container** blade, and then select the **records.json** blob in the list of blobs.
11. From the **Blob** blade, find the blob metadata, and then copy the URL for the blob.
12. On the taskbar, right-click the **Microsoft Edge** icon or activate the shortcut menu, and then select **New window**.
13. In the new browser window, go to the URL that you copied for the blob.
14. The JavaScript Object Notation (JSON) contents of the blob should now display. Close the browser window with the JSON contents.
15. Return to the browser window with the Azure portal, and then close the **Blob** blade.
16. Return to the **Container** blade, and then select **Change access level policy**.
17. In the **Change access level** pop-up window, perform the following actions:
    1. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
    2. Select **OK**.
18. On the taskbar, right-click the **Microsoft Edge** icon or activate the shortcut menu, and then select **New window**.
19. In the new browser window, go to the URL that you copied for the blob.
20. An error message indicating that the resource wasn't found should now display.
    Note: If the error message doesn't display, your browser might have cached the file. Press Ctrl+F5 to refresh the page until the error message displays.

28.2.5.2 Task 2: Pull and configure the Azure SDK for .NET

1. On the taskbar, select the **Windows Terminal** icon.
2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

    ```
    cd F:\Allfiles\Labs\07\Starter\func
    ```

3. At the open command prompt, enter the following command, and then select Enter to add version **12.6.0** of the **Azure.Storage.Blobs** package from NuGet:

    ```
    dotnet add package Azure.Storage.Blobs --version 12.6.0
    ```

    Note: The Azure.Storage.Blobs NuGet package references the subset of the Azure SDK for .NET required to write code for Azure Blob Storage.

4. Close the currently running **Windows Terminal** application.
5. On the **Start** screen, select the **Visual Studio Code** tile.
6. From the **File** menu, select **Open Folder**.
7. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\07\Starter\func**, and then select **Select Folder**.
8. In the Explorer pane of the **Visual Studio Code** window, open the **FileParser.cs** file.
9. Add a **using directive** for the **Azure.Storage.Blobs** namespace:

    ```
    using Azure.Storage.Blobs;
    ```

10. Observe the **FileParser.cs** file, which should now include:

    ```
    using Azure.Storage.Blobs;
    using Microsoft.AspNetCore.Mvc;
    ```

```
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using System;
using System.Threading.Tasks;

public static class FileParser
{
    [FunctionName("FileParser")]
    public static async Task<IActionResult> Run(
        [HttpTrigger("GET")] HttpRequest request)
    {
        string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
        return new OkObjectResult(connectionString);
    }
}
```

28.2.5.3 Task 3: Write Azure Blob Storage code using the Azure SDK for .NET

1. Within the **Run** method of the **FileParser** class, delete the following line of code:

   ```
   return new OkObjectResult(connectionString);
   ```

2. Still within the **Run** method, add the following code block to create a new instance of the **BlobClient** class by passing in your *connectionString* variable, a `"drop"` string value, and a `"records.json"` string value to the constructor:

   ```
   BlobClient blob = new BlobClient(connectionString, "drop", "records.json");
   ```

3. Still within the **Run** method, add the following code block to use the **BlobClient.DownloadAsync** method to download the contents of the referenced blob asynchronously and store the result in a variable named *response*:

   ```
   var response = await blob.DownloadAsync();
   ```

4. Still within the **Run** method, add the following code block to return the value of the various content stored in the *content* variable by using the **FileStreamResult** class constructor:

   ```
   return new FileStreamResult(response?.Value?.Content, response?.Value?.ContentType);
   ```

5. Observe the **FileParser.cs** file again, which should now include:

   ```
   using Azure.Storage.Blobs;
   using Microsoft.AspNetCore.Mvc;
   using Microsoft.Azure.WebJobs;
   using Microsoft.AspNetCore.Http;
   using System;
   using System.Threading.Tasks;

   public static class FileParser
   {
       [FunctionName("FileParser")]
       public static async Task<IActionResult> Run(
           [HttpTrigger("GET")] HttpRequest request)
       {
           string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
           BlobClient blob = new BlobClient(connectionString, "drop", "records.json");
           var response = await blob.DownloadAsync();
           return new FileStreamResult(response?.Value?.Content, response?.Value?.ContentType);
       }
   }
   ```

6. Select **Save** to save your changes to the **FileParser.cs** file.

28.2.5.4 Task 4: Deploy and validate the Azure Functions app

1. On the taskbar, select the **Windows Terminal** icon.

2. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\07\Starter\func
   ```

3. At the open command prompt, enter the following command, and then select Enter to log in to the Azure CLI:

   ```
   az login
   ```

4. In the **Microsoft Edge** browser window, perform the following actions:

   1. Enter the email address for your Microsoft account, and then select **Next**.
   2. Enter the password for your Microsoft account, and then select **Sign in**.

5. Return to the currently open **Windows Terminal** window. Wait for the sign-in process to finish.

6. Enter the following command, and then select Enter to publish the function app project again:

   ```
   func azure functionapp publish <function-app-name>
   ```

   > **Note**: As an example, if your **Function App name** is **securefuncstudent**, your command would be `func azure functionapp publish securefuncstudent`. You can review the documentation to [publish the local function app project][azure-functions-core-tools-publish-azure] using the **Azure Functions Core Tools**.

7. Wait for the deployment to finalize before you move forward with the lab.

8. Close the currently running **Windows Terminal** application.

9. On the taskbar, select the **Microsoft Edge** icon.

10. In the open browser window, go to the Azure portal (https://portal.azure.com).

11. In the Azure portal's navigation pane, select the **Resource groups** link.

12. On the **Resource groups** blade, find and then select the **ConfidentialStack** resource group that you created earlier in this lab.

13. On the **ConfidentialStack** blade, select the **securefunc[yourname]** function app that you created earlier in this lab.

14. From the **App Service** blade, select the **Functions** option from the **Functions** section.

15. In the **Functions** pane, select the the existing **FileParser** function.

16. In the **Function** blade, select the **Code + Test** option from the **Developer** section.

17. In the function editor, select **Test/Run**.

18. In the pop-up dialog box that appears, perform the following actions:

    - In the **HTTP method** list, select **GET**.

19. Select **Run** to test the function.

20. Observe the results of the test run. The output will contain the content of the **$/drop/records.json** blob stored in your Azure Storage account.

    Review: In this exercise, you used C# code to access a storage account, and then downloaded the contents of a blob.

28.2.6 Exercise 5: Clean up your subscription

28.2.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

   > **Note**: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

   Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If Cloud Shell configuration options don't display, this is most likely because you are using an existing subscription with this course's labs. The labs are written with the presumption that you are using a new subscription.

28.2.6.2 Task 2: Delete a resource group

1. When you receive the command prompt, enter the following command, and then select Enter to delete the **ConfidentialStack** resource group:

   ```
   az group delete --name ConfidentialStack --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

28.2.6.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.

   Review: In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

28.3 lab: az204Title: 'Lab 07: Access resource secrets more securely across services' az020Title: 'Lab 07: Access resource secrets more securely across services' az204Module: 'Module 07: Implement secure cloud solutions' az020Module: 'Module 07: Implement secure cloud solutions'

29 Lab 07: Access resource secrets more securely across services

30 Student lab manual

30.1 Lab scenario

Your company has a data-sharing business-to-business (B2B) agreement with another local business in which you're expected to parse a file that's dropped off nightly. To keep things simple, the second company has decided to drop the file as a Microsoft Azure Storage blob every night. You're now tasked with devising a way to access the file and generate a secure URL that any internal system can use to access the blob without exposing the file to the internet. You've decided to use Azure Key Vault to store the credentials for the storage account and Azure Functions to write the code necessary to access the file without storing credentials in plaintext or exposing the file to the internet.

30.2 Objectives

After you complete this lab, you'll be able to:

- Create an Azure Key Vault and store secrets in the key vault.
- Create a system-assigned managed identity for an Azure App Service instance.
- Create a Key Vault access policy for an Azure Active Directory identity or application.
- Use the Azure SDK for .NET to download a blob with an Azure Function.

30.3 Lab setup

- Estimated time: **45 minutes**

30.4 Instructions

30.4.1 Before you start

30.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

30.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Windows Terminal
- Visual Studio Code

30.4.2 Exercise 1: Create Azure resources

30.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, a dialog box offering a tour of the portal displays. If you prefer to skip the tour, select **Get Started** to skip the tour.

30.4.2.2 Task 2: Create an Azure Storage account

1. Create a new storage account with the following details:
   - New resource group: **ConfidentialStack**
   - Name: **securestor**[**yourname**]
   - Location: **East US**
   - Performance: **Standard**
   - Account kind: **StorageV2 (general purpose v2)**
   - Replication: **Locally-redundant storage (LRS)**
     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.
2. Open the **Access Keys** blade of your newly created storage account instance.
3. Record the value in the **Connection string** text box. You'll use this value later in this lab.

30.4.2.3 Task 3: Create an Azure key vault

1. Create a new key vault with the following details:
   - Existing resource group: **ConfidentialStack**
   - Name: **securevault**[**yourname**]
   - Region: **East US**
   - Pricing tier: **Standard**
     Note: Wait for Azure to finish creating the key vault before you move forward with the lab. You'll receive a notification when the vault is created.

30.4.2.4 Task 4: Create an Azure Functions app

1. Create a new function app with the following details:

   - Existing resource group: **ConfidentialStack**
   - App name: **securefunc**[**yourname**]
   - Publish: **Code**
   - Runtime Stack: **.NET**
   - Version: **3.1**
   - Region: **East US**
   - Operating system: **Linux**
   - Storage account: **securestor**[**yourname**]
   - Plan: **Consumption (Serverless)**
   - Enable Application Insights: **Yes**

   Note: Wait for Azure to finish creating the function app before you move forward with the lab. You'll receive a notification when the app is created.

Review: In this exercise, you created all the resources that you'll use for this lab.

30.4.3 Exercise 2: Configure secrets and identities

30.4.3.1 Task 1: Configure a system-assigned managed service identity

1. Access the **securefunc[yourname]** function app that you created earlier in this lab.
2. Browse to the **Identity** option from the **Settings** section.
3. Enable the system-assigned managed identity, and then save your changes.

30.4.3.2 Task 2: Create a Key Vault secret

1. Access the **securevault[yourname]** key vault that you created earlier in this lab.
2. Select the **Secrets** link in the **Settings** section.
3. Create a new secret with the following settings:
   - Name: **storagecredentials**
   - Value: ***Storage connection string***
   - Enabled: **Yes**
     Note: Use the storage account connection string that you recorded earlier in this lab for the value of this secret.
4. Select through the secret to find the metadata for its latest version.
5. Record the value of the **Secret Identifier** text box because you'll use this later in the lab.

30.4.3.3 Task 3: Configure a Key Vault access policy

1. Access the **securevault[yourname]** key vault that you created earlier in this lab.
2. Browse to the **Access Policies** link in the **Settings** section.
3. Create a new access policy with the following settings:
   - Principal: **securefunc[yourname]**
     Note: The system-assigned managed identity you created earlier in this lab will have the same name as the Azure Functions resource.
   - Key permissions: **None**
   - Secret permissions: **GET**
   - Certificate permissions: **None**
   - Authorized application: **None**
4. Save your changes to the list of **Access Policies**.

30.4.3.4 Task 4: Create a Key Vault-derived application setting

1. Access the **securefunc[yourname]** function app that you created earlier in this lab.
2. Browse to the **Configuration** option from the **Settings** section.
3. Create a new application setting by using the following details:
   - Name: **StorageConnectionString**
   - Value: **@Microsoft.KeyVault(SecretUri=*Secret Identifier*)**
   - Deployment slot setting: **Not selected**
     Note: You'll need to build a reference to your ***Secret Identifier*** by using the previous syntax.
     For example, if your ***Secret Identifier*** is `https://securevaultstudent.vault.azure.net/secrets/storagecr`
     your value would be `@Microsoft.KeyVault(SecretUri=https://securevaultstudent.vault.azure.net/secre`
4. Save your changes to the application settings.

Review: In this exercise, you created a system-assigned managed service identity for your function app and then gave that identity the appropriate permissions to get the value of a secret in your key vault. Finally, you created a secret that you referenced within your function app's configuration settings.

30.4.4 Exercise 3: Build an Azure Functions app

30.4.4.1 Task 1: Initialize a function project

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** empty directory:

   ```
   cd F:\Allfiles\Labs\07\Starter\func
   ```

3. Use the **Azure Functions Core Tools** to create a new local Azure Functions project with the following details:

- worker runtime: **dotnet**

```
func init --worker-runtime dotnet --force
```

Note: You can review the documentation to [create a new project][azure-functions-core-tools-new-project] using the **Azure Functions Core Tools**.

4. **Build** the .NET Core 3.1 project:

```
dotnet build
```

30.4.4.2 Task 2: Create an HTTP-triggered function

1. Still in the open command prompt,create a new function with the following details:

   - template: **HTTP trigger**
   - name: **FileParser**

```
func new --template "HTTP trigger" --name "FileParser"
```

Note: You can review the documentation to [create a new function][azure-functions-core-tools-new-function] using the **Azure Functions Core Tools**.

2. Close the currently running **Windows Terminal** application.

30.4.4.3 Task 3: Configure and read an application setting

1. Open **Visual Studio Code**.
2. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\07\Starter\func**.
3. Open the **local.settings.json** file.
4. Update the value of the **Values** object by adding a new setting named **StorageConnectionString** and setting it to a string value of [**TEST VALUE**]:

```
"Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet",
    "StorageConnectionString": "[TEST VALUE]"
}
```

5. Open the **FileParser.cs** file.
6. In the code editor, delete all the code within the **FileParser.cs** file.
7. Add **using directives** for the **Microsoft.AspNetCore.Mvc**, **Microsoft.Azure.WebJobs**, **Microsoft.AspNetCore.Http**, **System**, and **System.Threading.Tasks** namespaces:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using System;
using System.Threading.Tasks;
```

8. Create a new **public static** class named **FileParser**:

```
public static class FileParser
{ }
```

9. Within the **FileParser** class, create a new **public static** *asynchronous* method named **Run** that returns a variable of type **Task<IActionResult>** and that also takes in a variable of type **HttpRequest** named *request*:

```
public static async Task<IActionResult> Run(
    HttpRequest request)
{ }
```

10. Append an attribute to the **Run** method of type **FunctionNameAttribute** that has its **name** parameter set to a value of **FileParser**:

```
[FunctionName("FileParser")]
public static async Task<IActionResult> Run(
    HttpRequest request)
{ }
```

11. Append an attribute to the **request** parameter of type **HttpTriggerAttribute** that has its **methods** parameter array set to a single value of **GET**:

```
[FunctionName("FileParser")]
public static async Task<IActionResult> Run(
    [HttpTrigger("GET")] HttpRequest request)
{ }
```

12. Within the **Run** method, retrieve the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method and storing the result in a **string** variable named **connectionString**:

```
string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
```

13. Finally, return the value of the **connectionString** variable as the HTTP response:

```
return new OkObjectResult(connectionString);
```

14. **Save** the **FileParser.cs** file.

30.4.4.4 Task 4: Validate the local function

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** project directory.
3. Start the function app project:

```
func start --build
```

> **Note**: You can review the documentation to [start the function app project locally][azure-functions-core-tools-start-function] using the **Azure Functions Core Tools**.

4. Open a new instance of the **Windows Terminal** application.
5. Start the **httpprepl** tool, and then set the base Uniform Resource Identifier (URI) to `http://localhost:7071`:

```
httprepl http://localhost:7071
```

> **Note**: An error message is displayed by the httprepl tool. This message occurs because the tool is searching for a Swagger definition file to use to "traverse" the API. Because your functio projectp does not produce a Swagger definition file, you'll need to traverse the API manually.

6. When you receive the tool prompt, browse to the relative **api/fileparser** directory:

```
cd api
cd fileparser
```

7. Run the **get** command:

```
get
```

8. Observe the [**TEST VALUE**] value of the **StorageConnectionString** being returned as the result of the HTTP request.
9. Exit the **httpprepl** application:

```
exit
```

10. Close all currently running instances of the **Windows Terminal** application.

30.4.4.5 Task 5: Deploy using the Azure Functions Core Tools

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** project directory.
3. Log in to the Azure Command-Line Interface (CLI) by using your Azure credentials:

```
az login
```

4. Publish the function app project:

```
func azure functionapp publish <function-app-name>
```

> **Note**: For example, if your **Function App name** is **securefuncstudent**, your command would be `func azure functionapp publish securefuncstudent`. You can review the documentation to [publish the local function app project][azure-functions-core-tools-publish-azure] using the **Azure Functions Core Tools**.

5. Wait for the deployment to finalize before you move forward with the lab.
6. Close the currently running **Windows Terminal** application.

30.4.4.6 Task 6: Test the Key Vault-derived application setting

1. Sign in to the Azure portal (https://portal.azure.com).
2. Access the **securefunc[yourname]** function app that you created earlier in this lab.
3. From the **App Service** blade, locate and open the **Functions** section, and then locate and open the **FileParser** function.
4. In the **Function** blade, select the **Code + Test** option from the **Developer** section.
5. In the function editor, select **Test/Run**.
6. In the pop-up dialog that appears, perform the following actions:
   - In the **HTTP method** list, select **GET**.
7. Select **Run** to test the function.
8. Observe the result of the test run. The result should be your Azure Storage connection string.

> **Review**: In this exercise, you used a service identity to read the value of a secret stored in Key Vault and returned that value as the result of a function app.

30.4.5 Exercise 4: Access Azure Blob Storage data

30.4.5.1 Task 1: Upload a sample Storage blob

1. Access the **securestor[yourname]** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then create a new container with the following settings:
   - Name: **drop**
   - Public access level: **Blob (anonymous read access for blobs only)**
3. Browse to the new **drop** container, and then select **Upload** to upload the **records.json** file in the **Allfiles (F): \Allfiles\Labs\07\Starter** folder on your lab VM.
   Note: You should enable the **Overwrite if files already exist** option.
4. Find the metadata for the **records.json** blob by selecting the blob entry in the list of blobs.
5. Using a new browser tab, go to to the URL for the blob, and then find the blob's contents.
6. Update the container's access level by changing the **Public access level** to **Private (no anonymous access)**.
7. Using a new browser window or tab, go to to the URL for the blob, and then find the blob's contents. You should receive an error message indicating that the resource wasn't found.
   > **Note**: If you don't receive the error message, your browser might have cached the file. Refresh the page until you receive the error message.

30.4.5.2 Task 2: Pull and configure the Azure SDK for .NET

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** project directory.
3. When you receive the open command prompt, add version **12.6.0** of the **Azure.Storage.Blobs** package from NuGet:

```
dotnet add package Azure.Storage.Blobs --version 12.6.0
```

> **Note**: The Azure.Storage.Blobs NuGet package references the subset of the Azure SDK for .NET required to write code for Azure Blob Storage.

4. Close the currently running **Windows Terminal** application.

5. Using **Visual Studio Code**, open the solution folder found at **Allfiles (F):\Allfiles\Labs\07\Starter\func**.
6. Open the **FileParser.cs** file.
7. Add a **using directive** for the **Azure.Storage.Blobs** namespace:

   ```
   using Azure.Storage.Blobs;
   ```

30.4.5.3 Task 3: Write Azure Blob Storage code using the Azure SDK for .NET

1. Within the **Run** method of the **FileParser** class, delete the following line of code:

   ```
   return new OkObjectResult(connectionString);
   ```

2. Still within the **Run** method, create a new instance of the **BlobClient** class by passing in your *connectionString* variable, a `"drop"` string value, and a `"records.json"` string value to the constructor:

   ```
   BlobClient blob = new BlobClient(connectionString, "drop", "records.json");
   ```

3. Still within the **Run** method, use the **BlobClient.DownloadAsync** method to download the contents of the referenced blob asynchronously and store the result in a variable named *response*:

   ```
   var response = await blob.DownloadAsync();
   ```

4. Still within the **Run** method, return the value of the various content stored in the *content* variable by using the **FileStreamResult** class constructor:

   ```
   return new FileStreamResult(response?.Value?.Content, response?.Value?.ContentType);
   ```

5. **Save** the **FileParser.cs** file.

30.4.5.4 Task 4: Deploy and validate the Azure Functions app

1. Open the **Windows Terminal** application.
2. Change the current directory to the **Allfiles (F):\Allfiles\Labs\07\Starter\func** project directory.
3. Log in to the Azure CLI by using your Azure credentials:

   ```
   az login
   ```

4. Publish the function app project again:

   ```
   func azure functionapp publish <function-app-name>
   ```

   > **Note**: As an example, if your **Function App name** is **securefuncstudent**, your command would be `func azure functionapp publish securefuncstudent`. You can review the documentation to [publish the local function app project][azure-functions-core-tools-publish-azure] using the **Azure Functions Core Tools**.

5. Wait for the deployment to finalize before you move forward with the lab.
6. Close the currently running **Windows Terminal** application.
7. Sign in to the Azure portal (https://portal.azure.com).
8. Access the **securefunc[yourname]** function app that you created earlier in this lab.
9. From the **App Service** blade, locate and open the **Functions** section, then locate and open the **FileParser** function.
10. In the **Function** blade, select the **Code + Test** option from the **Developer** section.
11. In the function editor, select **Test/Run**.
12. In the pop-up dialog that appears, perform the following actions:
    - In the **HTTP method** list, select **GET**.
13. Select **Run** to test the function.
14. Observe the results of the test run. The output will contain the content of the **$/drop/records.json** blob stored in your Azure Storage account.

    Review: In this exercise, you used C# code to access a storage account and then downloaded the contents of a blob.

30.4.6 Exercise 5: Clean up your subscription

30.4.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

30.4.6.2 Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **ConfidentialStack** resource group:

   ```
   az group delete --name ConfidentialStack --no-wait --yes
   ```

2. Close the Cloud Shell pane from the portal.

30.4.6.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.

   Review: In this exercise, you cleaned up your subscription by removing the resource groups that were used in this lab.

30.5 lab: az204Title: 'Lab 08: Creating a multi-tier solution by using services in Azure' az020Title: 'Lab 08: Creating a multi-tier solution by using services in Azure' az204Module: 'Module 08: Implement API Management' az020Module: 'Module 08: Implement API Management' type: 'Answer Key'

31 Lab 08: Creating a multi-tier solution by using services in Azure

32 Student lab answer key

32.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention. However, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

32.2 Instructions

32.2.1 Before you start

32.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

32.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge

32.2.2 Exercise 1: Creating an Azure App Service resource by using a Docker container image

32.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. At the sign-in page, enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you will be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

32.2.2.2 Task 2: Create a web app by using Azure App Service resource by using an httpbin container image

1. In the Azure portal's navigation pane, select **Create a resource**.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Web**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Web App** result.
5. From the **Web App** blade, select **Create**.
6. From the second **Web App** blade, find the tabs from the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new web app. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **ApiService**, and then select **OK**.
   3. In the **Name** text box, enter httpapi*[yourname]***.
   4. In the **Publish** section, select **Docker Container**.
   5. In the **Operating System** section, select **Linux**.
   6. In the **Region** drop-down list, select the **East US** region.
   7. In the **Linux Plan (East US)** section, select **Create new**, enter the value **ApiPlan** in the **Name** text box, and then select **OK**.
   8. Leave the **SKU and size** section set to its default value.
   9. Select **Next: Docker**.
8. From the **Docker** tab, perform the following actions:
   1. In the **Options** drop-down list, select **Single Container**.
   2. In the **Image Source** drop-down list, select **Docker Hub**.
   3. In the **Access Type** drop-down list, select **Public**.
   4. In the **Image and tag** text box, enter **kennethreitz/httpbin:latest**.
   5. Select **Review + Create**.
9. From the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the web app by using your specified configuration.

    Note: Wait for the creation task to complete before you move forward with this lab.

32.2.2.3 Task 3: Test the httpbin web application

1. In the Azure portal's navigation pane, select **Resource groups**.
2. From the **Resource groups** blade, select the **ApiService** resource group that you created earlier in this lab.
3. From the **ApiService** blade, select the httpapi*[yourname]*** web app that you created earlier in this lab.
4. From the **Web App** blade, select **Browse**.
5. Within the web application, perform the following actions:
    1. Select **Response formats**.
    2. Select **GET /xml**.
    3. Select **Try it out**.
    4. Select **Execute**.
    5. Observe the value of the **Response body** and **Response headers** text boxes.
    6. Observe the value of the **Request URL** text box.
6. Close the browser window for the web application.
7. Find the **Web App** blade for the httpapi*[yourname]*** web app back in the Azure portal.
8. From the **Web App** blade, in the **Settings** section, select the **Properties** link.
9. In the **Properties** section, record the value of the **URL** text box. You'll use this value later in the lab to make requests against the API.

32.2.2.4 Review

In this exercise, you created a new Azure web app by using a container image sourced from Docker Hub.

32.2.3 Exercise 2: Build an API proxy tier by using Azure API Management

32.2.3.1 Task 1: Create an API Management resource

1. In the Azure portal's navigation pane, select **Create a resource**.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **API**, and then select Enter.
4. From the **Marketplace** search results blade, select the **API Management** result.
5. From the **API Management** blade, select **Create**.
6. From the **API Management Service** blade, perform the following actions:
    1. In the **Name** text box, enter prodapi*[yourname]*.
    2. Leave the **Subscription** text box set to its default value.
    3. In the **Resource group** list, select the **ApiService** group that you created earlier in the lab.
    4. In the **Location** list, select **East US**.
    5. In the **Organization name** text box, enter **Contoso**.
    6. Leave the **Administrator email** text box set to its default value.

        Note: If this field is empty, you can enter any non-working email address.
    7. In the **Pricing tier** list, select **Consumption (99.9 SLA, %)**.
    8. Select **Create**.

    Note: Wait for the creation task to complete before you move forward with this lab.

32.2.3.2 Task 2: Define a new API

1. In the Azure portal's navigation pane, select **Resource groups**.
2. From the **Resource groups** blade, select the **ApiService** resource group that you created earlier in this lab.
3. From the **ApiService** blade, select the prodapi*[yourname]*** API Management account that you created earlier in this lab.
4. From the **API Management Service** blade, in the **API Management** section, select **APIs** .
5. In the **Add a new API** section, select **Blank API**.
6. In the **Create a blank API** window, perform the following actions:
    1. In the **Display name** text box, enter **HTTPBin API**.
    2. In the **Name** text box, enter **httpbin-api**.
    3. In the **Web service URL** text box, enter the URL for the web app that you copied earlier in this lab.

        Note: Depending on how you copy the URL, you might need to add an "http://" prefix to create a valid URL value.
    4. Leave the **API URL suffix** text box empty.
    5. Select **Create**.

    Note: Wait for the new API to finish being created.
7. From the **Design** tab, select **Add operation**.
8. In the **Add operation** section, perform the following actions:
    1. In the **Display name** text box, enter **Echo Headers**.
    2. In the **Name** text box, enter **echo-headers**.
    3. In the **URL** list, select **GET**.
    4. In the **URL** text box, enter /.
    5. Select **Save**.
9. Back from the **Design** tab, in the list of operations, select **All Operations**.
10. In the **Design** section for **All Operations**, find the **Inbound processing** tile, and then select **Add policy**.
11. In the **Add inbound policy** section, select the **Set headers** tile.
12. In the **Inbound processing, Set Headers** section, perform the following actions:
    1. In the **Name** text box, enter **source**.
    2. In the **Value** text box, select the list, select **Add Value**, and then enter **azure-api-mgmt**.
    3. In the **Action** list, select **append**.
    4. Select **Save**.
13. Back from the **Design** tab, in the list of operations, select **Echo Headers**.
14. In the **Design** section for **Echo Headers**, find the **Backend** tile, and then select the pencil icon.
15. In the **Backend** section, perform the following actions:
    1. In the **Service URL** section, select the **Override** check box.
    2. In the **Service URL** text box, append the value **/headers** to its current value.

        Note: For example, if the current value is **http://httpapi*[yourname]*.azurewebsites.net**, the new value will be **http://httpapi*[yourname]*.azurewebsites.net/headers**
    3. Select **Save**.
16. Back from the **Design** tab, in the list of operations, select **Echo Headers**.

17. From the **Test** tab, select the **Echo Headers** operation.
18. In the **Echo Headers** section, select **Send**.
19. Observe the results of the API request.

    Note: Observe how there's many headers sent as part of your request that are echoed in the response. Specifically, you'll notice the new **Source** header that you created as part of this task.
20. Select the **Design** tab to return to the list of operations.

32.2.3.3 Task 3: Manipulate an API response

1. From the **Design** tab, select **Add operation**.
2. In the **Add operation** section, perform the following actions:
    1. In the **Display name** text box, enter **Get Legacy Data**.
    2. In the **Name** text box, enter **get-legacy-data**.
    3. In the **URL** list, select **GET**.
    4. In the **URL** text box, enter **/xml**.
    5. Select **Save**.
3. Back from the **Design** tab, in the list of operations, select **Get Legacy Data**.
4. From the **Test** tab, select the **Get Legacy Data** operation.
5. In the **Get Legacy Data** section, select **Send**.
6. Observe the results of the API request.

    Note: At this point, the results should be in XML format.
7. Back from the **Design** tab, in the list of operations, select **Get Legacy Data**.
8. In the **Design** section for the **Get Legacy Data** operation, find the **Outbound processing** tile, and then select **Add policy**.
9. In the **Add outbound policy** section, select the **Other policies** tile.
10. In the policy code editor, find the following block of XML content:

```
<outbound>
    <base />
</outbound>
```

11. Replace that block of XML with the following XML:

```
<outbound>
    <base />
    <xml-to-json kind="direct" apply="always" consider-accept-header="false" />
</outbound>
```

12. In the policy code editor, select **Save**.
13. Back from the **Design** tab, in the list of operations, select **Get Legacy Data**.
14. From the **Test** tab, select the **Get Legacy Data** operation.
15. In the **Get Legacy Data** section, select **Send**.
16. Observe the results of the API request.

    Note: The new results are in JavaScript Object Notation (JSON) format.
17. Within the **HTTP response** section, perform the following actions:
    1. Select **Trace**.
    2. Observe the content in the **Backend** and **Outbound** text boxes.

32.2.3.4 Review

In this exercise, you built a proxy tier between your App Service resource and any developers who wish to make queries.

32.2.4 Exercise 3: Clean up your subscription

32.2.4.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

   - A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

     Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

32.2.4.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ApiService** resource group:

   ```
   az group delete --name ApiService --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

32.2.4.3 Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

32.2.4.4 Review

32.3 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

32.4 lab: az204Title: 'Lab 08: Creating a multi-tier solution by using services in Azure' az020Title: 'Lab 08: Creating a multi-tier solution by using services in Azure' az204Module: 'Module 08: Implement API Management' az020Module: 'Module 08: Implement API Management'

33 Lab 08: Creating a multi-tier solution by using services in Azure

34 Student lab manual

34.1 Lab scenario

The developers in your company have successfully adopted and used the https://httpbin.org/ website to test various clients that issue HTTP requests. Your company would like to use one of the publicly available containers on Docker Hub to host the httpbin web application in an enterprise-managed environment with a few caveats. First, developers who are issuing Representational State Transfer (REST) queries should receive standard headers that are used throughout the company's applications. Second, developers should be able to get responses by using JavaScript Object Notation (JSON) even if the API that's used behind the scenes doesn't support the data format. You're tasked with using Microsoft Azure API Management to create a proxy tier in front of the httpbin web application to implement your company's policies.

34.2 Objectives

After you complete this lab, you will be able to:

- Create a web application from a Docker Hub container image.
- Create an API Management account.
- Configure an API as a proxy for another Azure service with header and payload manipulation.

34.3 Lab setup

- Estimated time: **45 minutes**

34.4 Instructions

34.4.1 Before you start

34.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

34.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge

34.4.2 Exercise 1: Creating an Azure App Service resource by using a Docker container image

34.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select the **Get Started** button to skip the tour.

34.4.2.2 Task 2: Create a web app by using Azure App Service resource by using an httpbin container image

1. Create a new web app with the following details:
   - New resource group: **ApiService**
   - Name: httpapi*[yourname]***
   - Publish: **Docker Container**
   - Operating system: **Linux**
   - Region: **East US**
   - New App Service plan: **ApiPlan**
   - SKU and size: **Premium V2 P1v2**
   - Docker options: **Single Container**
   - Image source: **Docker Hub**
   - Access type: **Public**
   - Image and tag: **kennethreitz/httpbin:latest**

     Note: Wait for Azure to finish creating the web app before you move forward with the lab. You'll receive a notification when the app is created.

34.4.2.3 Task 3: Test the httpbin web application

1. Access the httpapi*[yourname]*** web app that you created earlier in this lab.
2. Open the httpapi*[yourname]*** web app in your browser.
3. Test the API by selecting the **Response formats** option, selecting **GET /xml**, selecting **Try it out**, and then finally by selecting **Execute**.
4. Observe the response from the HTTP request. Specifically, observe the content of the **Request URL**, **Response body**, and **Response headers** text boxes.
5. Return to the Azure portal and the **Web App** blade for the httpapi*[yourname]*** web app.
6. Access the **Properties** section of the **Web App** blade.
7. In the **Properties** section, record the value of the **URL** text box. You'll use this value later in the lab to make requests against the API.

34.4.2.4 Review

In this exercise, you created a new Azure web app by using a container image sourced from Docker Hub.

34.4.3 Exercise 2: Build an API proxy tier by using Azure API Management

34.4.3.1 Task 1: Create an API Management resource

1. In the Azure portal, create a new API Management service instance with the following details:
   - Existing resource group: **ApiService**
   - Name: prodapi*[yourname]***
   - Location: **East US**
   - Organization name: **Contoso**
   - Administrator email: **Leave set to default value**

     Note: If this field is empty, you can enter any non-working email address.
   - Pricing tier: **Consumption (99.9 SLA, %)**

34.4.3.2 Task 2: Define a new API

1. Access the prodapi*[yourname]*** API Management service instance that you created earlier in this lab.
2. Create a new **Blank API** with the following details:
   - Display name: **HTTPBin API**
   - Name: **httpbin-api**
   - Web service URL: *Enter the URL for the web app that you copied earlier in this lab*

     Note: Depending on how you copy the URL, you might need to add an "http://" prefix to create a valid URL value.
3. Add a new **operation** to the recently created API with the following details:
   - Display name: **Echo Headers**
   - Name: **echo-headers**
   - URL: **GET /**
4. Add a new **Set Headers** inbound policy to **All Operations** with the following details:
   - Name: **source**
   - Value: **azure-api-mgmt**
   - Action: **append**
5. Update the **Backend** for the **Echo Headers** operation by overriding the **Service URL** and appending **/headers** to its current value.

Note: For example, if the current value is **http://httpapi*[yourname]*.azurewebsites.net**, the new value will be **http://httpapi*[yourname]*.azurewebsites.net/headers**

6. Test the **Echo Headers** operation in the **HTTPBin API**, observing the results of the API request.

   Note: Observe how there's many headers sent as part of your request that are echoed in the response. Specifically, you'll notice the new **Source** header that you created as part of this task.

34.4.3.3 Task 3: Manipulate an API response

1. Add a new **operation** to the API with the following details:
   - Display name: **Get Legacy Data**
   - Name: **get-legacy-data**
   - URL: **GET /xml**
2. Test the **Get Legacy Data** operation in the **HTTPBin API**, observing the results of the API request.

   Note: At this point, the results should be in XML format.
3. Add a new custom outbound policy scoped to the **Get Legacy Data** operation by first locating the following block of XML content:

```
<outbound>
    <base />
</outbound>
```

4. Replace that block of XML with the following XML, and then save the policy:

```
<outbound>
    <base />
    <xml-to-json kind="direct" apply="always" consider-accept-header="false" />
</outbound>
```

5. Test the **Get Legacy Data** operation in the **HTTPBin API**, observing the results of the API request.

   Note: The new results are in JSON format.
6. Use the **Trace** feature of the test tool to observe the request sent to the back-end service.

34.4.3.4 Review

In this exercise, you built a proxy tier between your App Service resource and any developers who wish to make queries.

34.4.4 Exercise 3: Clean up your subscription

34.4.4.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

34.4.4.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ApiService** resource group:

```
az group delete --name ApiService --no-wait --yes
```

2. Close the Cloud Shell pane in the portal.

34.4.4.3 Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

34.4.4.4 Review

34.5 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

34.6 lab: az204Title: 'Lab 09: Publishing and subscribing to Event Grid events' az020Title: 'Lab 09: Publishing and subscribing to Event Grid events' az204Module: 'Module 09: Develop event-based solutions' az020Module: 'Module 09: Develop event-based solutions' type: 'Answer Key'

35 Lab 09: Publishing and subscribing to Event Grid events

36 Student lab answer key

36.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

36.2 Instructions

36.2.1 Before you start

36.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

36.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Microsoft Visual Studio Code

36.2.2 Exercise 1: Create Azure resources

36.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

36.2.2.2 Task 2: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell. Perform the following actions in the wizard:

- When a dialog box prompts you to create a new storage account to begin using the shell, accept the default settings, and then select **Create storage**.

  Note: Wait for Cloud Shell to finish its initial setup procedures before continuing with the lab. If you don't notice the **Cloud Shell** configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

3. In Azure portal, at the **Cloud Shell** command prompt enter the following command, and then select Enter to get the version of the Azure Command-Line Interface (Azure CLI) tool:

   ```
   az --version
   ```

36.2.2.3 Task 3: View the Microsoft.EventGrid provider registration

1. At the **Cloud Shell** command prompt in the portal, perform the following actions:
   1. Enter the following command, and then select Enter to get a list of subgroups and commands at the root level of the Azure CLI:

      ```
      az --help
      ```

   2. Enter the following command, and then select Enter to get a list of the commands that are available for resource providers:

      ```
      az provider --help
      ```

   3. Enter the following command, and then select Enter to list all currently registered providers:

      ```
      az provider list
      ```

   4. Enter the following command, and then select Enter to list just the namespaces of the currently registered providers:

      ```
      az provider list --query "[].namespace"
      ```

   5. Review the list of currently registered providers. Notice that the **Microsoft.EventGrid** provider is currently included in the list of providers.
2. Close the Cloud Shell pane.

36.2.2.4 Task 4: Create a custom Event Grid topic

1. In the Azure portal's navigation pane, select **Create a resource**.
2. On the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Event Grid Topic**, and then select Enter.
4. On the **Everything** search results blade, select the **Event Grid Topic** result.
5. On the **Event Grid Topic** blade, select **Create**.
6. On the **Create Topic** blade, perform the following actions:
   1. In the **Name** text box, enter hrtopic*[yourname]***.
   2. In the **Resource group** section, select **Create new**, enter **PubSubEvents**, and then select **OK**.
   3. From the **Location** drop-down list, select the **(US) East US** region.
   4. From the **Event Schema** drop-down list, select **Event Grid Schema**, and then select **Create**.

      Note: Wait for Azure to finish creating the topic before you continue with the lab. You'll receive a notification when the topic is created.

36.2.2.5 Task 5: Deploy the Azure Event Grid viewer to a web app

1. In the Azure portal's navigation pane, select **Create a resource**.
2. On the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Web**, and then select Enter.
4. On the **Everything** search results blade, select the **Web App** result.

5. On the **Web App** blade, select **Create**.
6. On the second **Web App** blade, find the tabs on the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new web app. You can select **Review + Create** at any time to skip the remaining tabs.
7. On the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **PubSubEvents**.
   3. In the **Name** text box, enter eventviewer*[yourname]***.
   4. In the **Publish** section, select **Docker Container**.
   5. In the **Operating System** section, select **Linux**.
   6. From the **Region** drop-down list, select the **East US** region.
   7. In the **Linux Plan (East US)** section, select **Create new**.
   8. In the **Name** text box, enter the value **EventPlan**, and then select **OK**.
   9. Leave the **SKU and size** section set to its default value.
   10. Select **Next: Docker**.
8. On the **Docker** tab, perform the following actions:
   1. From the **Options** drop-down list, select **Single Container**.
   2. From the **Image Source** drop-down list, select **Docker Hub**.
   3. From the **Access Type** drop-down list, select **Public**.
   4. In the **Image and tag** text box, enter **microsoftlearning/azure-event-grid-viewer:latest**.
   5. Select **Review + Create**.
9. On the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the web app using your specified configuration.

    Note: Wait for Azure to finish creating the web app before you continue with the lab. You'll receive a notification when the app is created.

36.2.2.6 Review

In this exercise, you created the Event Grid topic and a web app that you will use throughout the remainder of the lab.

36.2.3 Exercise 2: Create an Event Grid subscription

36.2.3.1 Task 1: Access the Event Grid Viewer web application

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **PubSubEvents** resource group that you created earlier in this lab.
3. On the **PubSubEvents** blade, select the eventviewer*[yourname]*** web app that you created earlier in this lab.
4. On the **App Service** blade, in the **Settings** category, select the **Properties** link.
5. In the **Properties** section, record the value of the **URL** text box. You'll use this value later in the lab.
6. Select **Overview**.
7. In the **Overview** section, select **Browse**.
8. Observe the currently running **Azure Event Grid viewer** web application. Leave this web application running for the remainder of the lab.

Note: This web application will update in real-time as events are sent to its endpoint. We will use this to monitor events throughout the lab.

9. Return to your currently open browser window that's displaying the Azure portal.

36.2.3.2 Task 2: Create new subscription

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **PubSubEvents** resource group that you created earlier in this lab.
3. On the **PubSubEvents** blade, select the hrtopic*[yourname]*** Event Grid topic that you created earlier in this lab.
4. On the **Event Grid Topic** blade, select **+ Event Subscription**.
5. On the **Create Event Subscription** blade, perform the following actions:
    1. In the **Name** text box, enter **basicsub**.
    2. In the **Event Schema** list, select **Event Grid Schema**.
    3. In the **Endpoint Type** list, select **Web Hook**.
    4. Select **Endpoint**.
    5. In the **Select Web Hook** dialog box, in the **Subscriber Endpoint** text box, enter the **Web App URL** value that you recorded earlier, ensure it uses an **https://** prefix, add the suffix **/api/updates**, and then select **Confirm Selection**.

        Note: For example, if your **Web App URL** value is `http://eventviewerstudent.azurewebsites.net/`, then your **Subscriber Endpoint** would be `https://eventviewerstudent.azurewebsites.net/api/update`
    6. Select **Create**.

        Note: Wait for Azure to finish creating the subscription before you continue with the lab. You'll receive a notification when the subscription is created.

36.2.3.3 Task 3: Observe the subscription validation event

1. Return to the browser window displaying the **Azure Event Grid viewer** web application.
2. Review the **Microsoft.EventGrid.SubscriptionValidationEvent** event that was created as part of the subscription creation process.
3. Select the event and review its JSON content.
4. Return to your currently open browser window with the Azure portal.

36.2.3.4 Task 4: Record subscription credentials

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **PubSubEvents** resource group that you created earlier in this lab.
3. On the **PubSubEvents** blade, select the hrtopic*[yourname]*** Event Grid topic that you created earlier in this lab.
4. On the **Event Grid Topic** blade, record the value of the **Topic Endpoint** field. You'll use this value later in the lab.
5. In the **Settings** category, select the **Access keys** link.
6. In the **Access keys** section, record the value of the **Key 1** text box. You'll use this value later in the lab.

36.2.3.5 Review

In this exercise, you created a new subscription, validated its registration, and then recorded the credentials required to publish a new event to the topic.

36.2.4 Exercise 3: Publish Event Grid events from .NET

36.2.4.1 Task 1: Create a .NET project

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\09\Starter\EventPublisher**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to create a new .NET project named **EventPublisher** in the current folder:

   ```
   dotnet new console --name EventPublisher --output .
   ```

   > **Note**: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

6. At the command prompt, enter the following command, and then select Enter to import version 4.1.0 of **Azure.Messaging.EventGrid** from NuGet:

   ```
   dotnet add package Azure.Messaging.EventGrid --version 4.1.0
   ```

   > **Note**: The **dotnet add package** command will add the **Microsoft.Azure.EventGrid** package from NuGet. For more information, go to Azure.Messaging.EventGrid.

7. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

   ```
   dotnet build
   ```

8. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

36.2.4.2 Task 2: Modify the Program class to connect to Event Grid

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, delete all the code in the existing file.
3. Add the following line of code to import the **Azure**, and **Azure.Messaging.EventGrid** namespaces from the **Azure.Messaging.EventGrid** package imported from NuGet:

   ```
   using Azure;
   using Azure.Messaging.EventGrid;
   ```

4. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

   ```
   using System;
   using System.Threading.Tasks;
   ```

5. Enter the following code to create a new **Program** class:

   ```
   public class Program
   {
   }
   ```

6. In the **Program** class, enter the following line of code to create a new string constant named **topicEndpoint**:

   ```
   private const string topicEndpoint = "";
   ```

7. Update the **topicEndpoint** string constant by setting its value to the **Topic Endpoint** of the Event Grid topic that you recorded earlier in this lab.
8. In the **Program** class, enter the following line of code to create a new string constant named **topicKey**:

   ```
   private const string topicKey = "";
   ```

9. Update the **topicKey** string constant by setting its value to the **Key** of the Event Grid topic that you recorded earlier in this lab.
10. In the **Program** class, enter the following code to create a new asynchronous **Main** method:

```
public static async Task Main(string[] args)
{
}
```

11. Observe the **Program.cs** file, which should now include the following lines of code:

```
using System;
using System.Threading.Tasks;
using Azure;
using Azure.Messaging.EventGrid;

public class Program
{
    private const string topicEndpoint = "<topic-endpoint>";
    private const string topicKey = "<topic-key>";

    public static async Task Main(string[] args)
    {
    }
}
```

36.2.4.3 Task 3: Publish new events

1. In the **Main** method, perform the following actions to publish a list of events to your topic endpoint:
    1. Add the following line of code to create a new variable named **endpoint** of type **Uri**, using the **topicEndpoint** string constant as a constructor parameter:

    ```
    Uri endpoint = new Uri(topicEndpoint);
    ```

    2. Add the following line of code to create a new variable named **credential** of type **AzureKeyCredential**, using the **topicKey** string constant as a constructor parameter:

    ```
    AzureKeyCredential credential = new AzureKeyCredential(topicKey);
    ```

    3. Add the following line of code to create a new variable named **client** of type **EventGridPublisherClient**, using the **endpoint** and **credential** variables as constructor parameters:

    ```
    EventGridPublisherClient client = new EventGridPublisherClient(endpoint, credential);
    ```

    4. Add the following block of code to create a new variable named **firstEvent** of type **EventGridEvent** and populate that variable with sample data:

    ```
    EventGridEvent firstEvent = new EventGridEvent(
        subject: $"New Employee: Alba Sutton",
        eventType: "Employees.Registration.New",
        dataVersion: "1.0",
        data: new
        {
            FullName = "Alba Sutton",
            Address = "4567 Pine Avenue, Edison, WA 97202"
        }
    );
    ```

    5. Add the following block of code to create a new variable named **secondEvent** of type **EventGridEvent** and populate that variable with sample data:

    ```
    EventGridEvent secondEvent = new EventGridEvent(
        subject: $"New Employee: Alexandre Doyon",
        eventType: "Employees.Registration.New",
        dataVersion: "1.0",
        data: new
        {
    ```

```
            FullName = "Alexandre Doyon",
            Address = "456 College Street, Bow, WA 98107"
        }
    );
```

6. Add the following line of code to asynchronously invoke the **EventGridPublisherClient.SendEventAsync** method using the **firstEvent** variable as a parameter:

   ```
   await client.SendEventAsync(firstEvent);
   ```

7. Add the following line of code to render the **"First event published"** message to the console:

   ```
   Console.WriteLine("First event published");
   ```

8. Add the following line of code to asynchronously invoke the **EventGridPublisherClient.SendEventAsync** method using the **secondEvent** variable as a parameter:

   ```
   await client.SendEventAsync(secondEvent);
   ```

9. Add the following line of code to render the **"Second event published"** message to the console:

   ```
   Console.WriteLine("Second event published");
   ```

2. Review the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    Uri endpoint = new Uri(topicEndpoint);
    AzureKeyCredential credential = new AzureKeyCredential(topicKey);
    EventGridPublisherClient client = new EventGridPublisherClient(endpoint, credential);

    EventGridEvent firstEvent = new EventGridEvent(
        subject: $"New Employee: Alba Sutton",
        eventType: "Employees.Registration.New",
        dataVersion: "1.0",
        data: new
        {
            FullName = "Alba Sutton",
            Address = "4567 Pine Avenue, Edison, WA 97202"
        }
    );

    EventGridEvent secondEvent = new EventGridEvent(
        subject: $"New Employee: Alexandre Doyon",
        eventType: "Employees.Registration.New",
        dataVersion: "1.0",
        data: new
        {
            FullName = "Alexandre Doyon",
            Address = "456 College Street, Bow, WA 98107"
        }
    );

    await client.SendEventAsync(firstEvent);
    Console.WriteLine("First event published");

    await client.SendEventAsync(secondEvent);
    Console.WriteLine("Second event published");
}
```

3. Save the **Program.cs** file.

4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

5. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\09\Solution\EventPublisher** folder.

6. Observe the success message output from the currently running console application.
7. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

36.2.4.4 Task 4: Observe published events

1. Return to the browser window with the **Azure Event Grid viewer** web application.
2. Review the **Employees.Registration.New** events that were created by your console application.
3. Select any of the events and review its JSON content.
4. Return to the Azure portal.

36.2.4.5 Review

In this exercise, you published new events to your Event Grid topic using a .NET console application.

36.2.5 Exercise 4: Clean up your subscription

36.2.5.1 Task 1: Open Azure Cloud Shell

1. In Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

      Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice the Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

36.2.5.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **PubSubEvents** resource group:

   ```
   az group delete --name PubSubEvents --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

36.2.5.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

36.2.5.4 Review

36.3 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

36.4 lab: az204Title: 'Lab 09: Publishing and subscribing to Event Grid events' az020Title: 'Lab 09: Publishing and subscribing to Event Grid events' az204Module: 'Module 09: Develop event-based solutions' az020Module: 'Module 09: Develop event-based solutions'

37 Lab 09: Publishing and subscribing to Event Grid events

38 Student lab manual

38.1 Lab scenario

Your company builds a human resources (HR) system used by various customers around the world. While the system works fine today, your development managers have decided to begin re-architecting the solution by decoupling application components. This decision was driven by a desire to make any future development simpler through modularity. As the developer who manages component communication, you have decided to introduce Microsoft Azure Event Grid as your solution-wide messaging platform.

38.2 Objectives

After you complete this lab, you will be able to:

- Create an Event Grid topic.
- Use the Azure Event Grid viewer to subscribe to a topic and illustrate published messages.
- Publish a message from a .NET application.

38.3 Lab setup

- Estimated time: **45 minutes**

38.4 Instructions

38.4.1 Before you start

38.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

38.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Microsoft Visual Studio Code

38.4.2 Exercise 1: Create Azure resources

38.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

38.4.2.2 Task 2: Open Azure Cloud Shell

1. Open a new Cloud Shell instance in the Azure portal.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.
3. At the **Cloud Shell** command prompt in the portal, use the **az** command with the **--version** flag to get the version of the Azure Command-Line Interface (Azure CLI) tool.

38.4.2.3 Task 3: View the Microsoft.EventGrid provider registration

1. Use the **az** command with the **--help** flag to find a list of subgroups and commands at the root level of the Azure CLI.
2. Use the **az provider** command with the **--help** flag to get a list of commands available for resource providers.
3. Use the **az provider list** command to get a list of all currently registered providers.
4. Use the **az provider list** command again with the **--query "[].namespace"** flag to list just the namespaces of the currently registered providers.
5. Review the list of currently registered providers. Note that the **Microsoft.EventGrid** provider is currently in the list of providers.
6. Close the Cloud Shell pane.

38.4.2.4 Task 4: Create a custom Event Grid topic

1. Create a new Event Grid topic with the following details:
   - Name: hrtopic*[yourname]***
   - New resource group: **PubSubEvents**
   - Location: **East US**
   - Event Schema: **Event Grid Schema**

     Note: Wait for Azure to finish creating the topic before you continue with the lab. You'll receive a notification when the app is created.

38.4.2.5 Task 5: Deploy the Azure Event Grid viewer to a web app

1. Create a new web app with the following details:
   - Existing resource group: **PubSubEvents**
   - Name: eventviewer*[yourname]***
   - Publish: **Docker Container**
   - Operating system: **Linux**
   - Region: **East US**
   - New App Service plan: **EventPlan**
   - SKU and size: **Premium V2 P1v2**
   - Docker options: **Single Container**
   - Image source: **Docker Hub**
   - Access type: **Public**
   - Image and tag: **microsoftlearning/azure-event-grid-viewer:latest**

     Note: Wait for Azure to finish creating the web app before you continue with the lab. You'll receive a notification when the app is created.

38.4.2.6 Review

In this exercise, you created the Event Grid topic and web app that you will use throughout the remainder of the lab.

38.4.3 Exercise 2: Create an Event Grid subscription

38.4.3.1 Task 1: Access the Event Grid Viewer web application

1. Access the eventviewer*[yourname]*** web app that you created earlier in this lab.
2. In the **Settings** section, go to the **Properties** section, and then record the value in the **URL** text box. You'll use this value later in the lab.
3. Browse to the currently running web app.
4. Observe the currently running **Azure Event Grid viewer** web application. Leave this web application running for the remainder of the lab.

   Note: This web application will update in real-time as events are sent to its endpoint. We will use this to monitor events throughout the lab.
5. Return to the Azure portal.

38.4.3.2 Task 2: Create new subscription

1. Access the hrtopic*[yourname]*** Event Grid topic that you created earlier in this lab.
2. Create a new **Event Subscription** with the following details:
   - Name: **basicsub**
   - Event Schema: **Event Grid Schema**
   - Endpoint Type: **Web Hook**
   - Endpoint: ***Web App URL recorded earlier in the lab, with an *https://* prefix and an */api/updates* suffix**

     Note: For example, if your **Web App URL** value is `http://eventviewerstudent.azurewebsites.net/`, then your endpoint would be `https://eventviewerstudent.azurewebsites.net/api/updates`.

   Note: Wait for Azure to finish creating the subscription before you continue with the lab. You'll receive a notification when the app is created.

38.4.3.3 Task 3: Observe the subscription validation event

1. Return to Azure Event Grid viewer.
2. Review the **Microsoft.EventGrid.SubscriptionValidationEvent** event that was created as part of the subscription creation process.
3. Select the event and review its JSON content.
4. Return to Azure portal.

38.4.3.4 Task 4: Record subscription credentials

1. Access the hrtopic*[yourname]*** Event Grid topic that you created earlier in this lab.
2. Record the value of the **Topic Endpoint** field. You'll use this value later in the lab.
3. In the **Settings** section, go to the **Access keys** section, and then record the value in the **Key 1** text box. You'll use this value later in the lab.

38.4.3.5 Review

In this exercise, you created a new subscription, validated its registration, and then recorded the credentials required to publish a new event to the topic.

38.4.4 Exercise 3: Publish Event Grid events from .NET

38.4.4.1 Task 1: Create .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\09\Starter\EventPublisher** folder.
2. Using a terminal, create a new .NET project named **EventPublisher** in the current folder:

```
dotnet new console --name EventPublisher --output .
```

> **Note**: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

3. Using the same terminal, import version 4.1.0 of **Azure.Messaging.EventGrid** from NuGet:

```
dotnet add package Azure.Messaging.EventGrid --version 4.1.0
```

> **Note**: The **dotnet add package** command will add the **Microsoft.Azure.EventGrid** package from NuGet. For more information, go to Azure.Messaging.EventGrid.

4. Using the same terminal, build the .NET web application:

```
dotnet build
```

5. Close the current terminal.

38.4.4.2 Task 2: Modify the Program class to connect to Event Grid

1. Open the **Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that the application will reference:

```
using System;
using System.Threading.Tasks;
using Azure;
using Azure.Messaging.EventGrid;
```

4. Create a new **Program** class with two constant string properties named **topicEndpoint** and **topicKey**, and then create an asynchronous **Main** entry point method:

```
public class Program
{
    private const string topicEndpoint = "";
    private const string topicKey = "";

    public static async Task Main(string[] args)
    {
    }
}
```

5. Update the **topicEndpoint** string constant by setting its value to the **Topic Endpoint** of the Event Grid topic that you recorded earlier in this lab.
6. Update the **topicKey** string constant by setting its value to the **Key** of the Event Grid topic that you recorded earlier in this lab.

38.4.4.3 Task 3: Publish new events

1. In the **Main** method, perform the following actions:
    1. Add the following block of code to connect to the Event Grid using the credentials you specified earlier in the lab:

```
Uri endpoint = new Uri(topicEndpoint);
AzureKeyCredential credential = new AzureKeyCredential(topicKey);
EventGridPublisherClient client = new EventGridPublisherClient(endpoint, credential);
```

    2. Add the following block of code to create a new variable named **firstEvent** of type **EventGridEvent** and populate that variable with sample data:

```
EventGridEvent firstEvent = new EventGridEvent(
    subject: $"New Employee: Alba Sutton",
    eventType: "Employees.Registration.New",
    dataVersion: "1.0",
    data: new
    {
```

```
            FullName = "Alba Sutton",
            Address = "4567 Pine Avenue, Edison, WA 97202"
        }
);
```

3. Add the following block of code to create a new variable named **secondEvent** of type **EventGridEvent** and populate that variable with sample data:

```
EventGridEvent secondEvent = new EventGridEvent(
    subject: $"New Employee: Alexandre Doyon",
    eventType: "Employees.Registration.New",
    dataVersion: "1.0",
    data: new
    {
        FullName = "Alexandre Doyon",
        Address = "456 College Street, Bow, WA 98107"
    }
);
```

4. Add the following block of code to asynchronously invoke the **EventGridPublisherClient.SendEventAsync** method using the **firstEvent** variable as a parameter and then render the **"First event published"** message to the console:

```
await client.SendEventAsync(firstEvent);
Console.WriteLine("First event published");
```

5. Add the following block of code to asynchronously invoke the **EventGridPublisherClient.SendEventAsync** method using the **secondEvent** variable as a parameter and then render the **"Second event published"** message to the console:

```
await client.SendEventAsync(secondEvent);
Console.WriteLine("Second event published");
```

2. Save the **Program.cs** file.
3. Using a terminal, run the .NET console application project:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\09\Solution\EventPublisher** folder.

4. Review the success message output from the currently running console application.
5. Close the current terminal.

38.4.4.4 Task 4: Observe published events

1. Return to the browser window with the **Azure Event Grid viewer** web application.
2. Review the **Employees.Registration.New** events that were created by your console application.
3. Select any of the events and review its JSON content.
4. Return to Azure portal.

38.4.4.5 Review

In this exercise, you published new events to your Event Grid topic using a .NET console application.

38.4.5 Exercise 4: Clean up your subscription

38.4.5.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

38.4.5.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **PubSubEvents** resource group:

   ```
   az group delete --name PubSubEvents --no-wait --yes
   ```

2. Close the Cloud Shell pane.

38.4.5.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

38.4.5.4 Review

38.5 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

38.6 lab: az204Title: 'Lab 10: Asynchronously processing messages by using Azure Queue Storage' az020Title: 'Lab 10: Asynchronously processing messages by using Azure Queue Storage' az204Module: 'Module 10: Develop message-based solutions' az020Module: 'Module 10: Develop message-based solutions' type: 'Answer Key'

39 Lab 10: Asynchronously processing messages by using Azure Queue Storage

40 Student lab answer key

40.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

40.2 Instructions

40.2.1 Before you start

40.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

40.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Visual Studio Code
- Azure Storage Explorer

40.2.2 Exercise 1: Create Azure resources

40.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

40.2.2.2 Task 2: Create a Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. On the **All services** blade, select **Storage Accounts**.
3. On the **Storage accounts** blade, get your list of storage account instances.
4. On the **Storage accounts** blade, select **New**.
5. On the **Create storage account** blade, observe the tabs on the blade, such as **Basics**, **Tags**, and **Review + Create**.

   Note: Each tab represents a step in the workflow to create a new storage account. You can select **Review + Create** at any time to skip the remaining tabs.
6. Select the **Basics** tab, and then in the tab area, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **AsyncProcessor**, and then select **OK**.
   3. In the **Storage account name** text box, enter asyncstor*[yourname]***.
   4. In the **Location** list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** list, select **Locally-redundant storage (LRS)**.
   8. Select **Review + Create**.
7. On the **Review + Create** tab, review the options that you specified in the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.

   Note: On the **Deployment** blade, wait for the creation task to complete before moving forward with this lab.
9. Select the **Go to resource** button on the **Deployment** blade to go to the newly created storage account.
10. On the **Storage account** blade, find the **Settings** section, and then select **Access keys**.
11. On the **Access keys** blade, select any one of the keys, and then record the value of either of the **Connection string** boxes. You'll use this value later in this lab.

    Note: It doesn't matter which connection string you choose. They are interchangeable.

40.2.2.3 Review

In this exercise, you created a new Azure Storage account that you'll use through the remainder of the lab.

40.2.3 Exercise 2: Configure the Azure Storage SDK in a .NET project

40.2.3.1 Task 1: Create a .NET project

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\10\Starter\MessageProcessor**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to create a new .NET project named **MessageProcessor** in the current folder:

   ```
   dotnet new console --name MessageProcessor --output .
   ```

   > **Note**: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

6. At the command prompt, enter the following command, and then select Enter to import version 12.0.0 of **Azure.Storage.Queues** from NuGet:

   ```
   dotnet add package Azure.Storage.Queues --version 12.0.0
   ```

   > **Note**: The **dotnet add package** command will add the **Azure.Storage.Queues** package from NuGet. For more information, go to Azure.Storage.Queues.

7. At the command prompt, enter the following command, and then select Enter to build the .NET web application:

   ```
   dotnet build
   ```

8. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

40.2.3.2 Task 2: Write code to access Azure Storage

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, delete all the code in the existing file.
3. Add the following line of code to import the **Azure**, **Azure.Storage.Queues**, and **Azure.Storage.Queues.Models** namespaces from the **Azure.Storage.Queues** package imported from NuGet:

   ```
   using Azure;
   using Azure.Storage.Queues;
   using Azure.Storage.Queues.Models;
   ```

4. Add the following lines of code to add **using** directives for the built-in namespaces that will be used in this file:

   ```
   using System;
   using System.Text;
   using System.Threading.Tasks;
   ```

5. Enter the following code to create a new **Program** class:

   ```
   public class Program
   {
   }
   ```

6. In the **Program** class, enter the following line of code to create a new string constant named **storageConnectionString**:

   ```
   private const string storageConnectionString = "";
   ```

7. Update the **storageConnectionString** string constant by setting its value to the **Connection string** of the Storage account that you recorded earlier in this lab.

8. In the **Program** class, enter the following line of code to create a new string constant named **queueName** with a value of **messagequeue**:

```
private const string queueName = "messagequeue";
```

9. In the **Program** class, enter the following code to create a new asynchronous **Main** method:

```
public static async Task Main(string[] args)
{
}
```

10. Observe the **Program.cs** file, which should now include:

```
using Azure;
using Azure.Storage.Queues;
using Azure.Storage.Queues.Models;
using System;
using System.Text;
using System.Threading.Tasks;

public class Program
{
    private const string storageConnectionString = "<storage-connection-string>";
    private const string queueName = "messagequeue";

    public static async Task Main(string[] args)
    {
    }
}
```

40.2.3.3 Task 3: Validate Azure Storage access

1. In the **Main** method, add the following line of code to connect to the storage account by creating a new variable named *client* of type **QueueClient**:

```
QueueClient client = new QueueClient(storageConnectionString, queueName);
```

2. In the **Main** method, add the following line of code to asynchronously create the queue if it doesn't already exist:

```
await client.CreateAsync();
```

3. In the **Main** method, add the following line of code to render a header for the "Account Metadata" section:

```
Console.WriteLine($"---Account Metadata---");
```

4. In the **Main** method, add the following line of code to render the Uniform Resource Identifier (URI) of the queue endpoint:

```
Console.WriteLine($"Account Uri:\t{client.Uri}");
```

5. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    QueueClient client = new QueueClient(storageConnectionString, queueName);
    await client.CreateAsync();

    Console.WriteLine($"---Account Metadata---");
    Console.WriteLine($"Account Uri:\t{client.Uri}");
}
```

6. Save the **Program.cs** file.

7. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

8. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

9. Observe the output from the currently running console application. The output contains metadata for the queue endpoint.
10. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

40.2.3.4 Review

In this exercise, you configured your .NET project to access the Storage service and manipulate a queue made available through the service.

40.2.4 Exercise 3: Read messages from the queue

40.2.4.1 Task 1: Write code to access queue messages

1. In the **Main** method, add the following line of code to render a header for the "Existing Messages" section:

   ```
   Console.WriteLine($"---Existing Messages---");
   ```

2. Within the **Main** method, perform the following actions to create variables that will be used when retrieving queue messages:

   1. Add the following line of code to create a variable of type **int** named *batchSize* with a value of **10**:

      ```
      int batchSize = 10;
      ```

   2. Add the following line of code to create a variable of type **TimeSpan** named *visibilityTimeout* with a value of **2.5 seconds**:

      ```
      TimeSpan visibilityTimeout = TimeSpan.FromSeconds(2.5d);
      ```

3. Within the **Main** method, perform the following actions to retrieve a batch of messages asynchronously from the queue service:

   1. Add the following line of code to invoke the **ReceiveMessagesAsync** asynchronous method of the **QueueClient** class, passing in the *batchSize* and *visibilityTimeout* variables as parameters:

      ```
      client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
      ```

   2. Update the previous line of code by adding more code to process the expression asynchronously by using the **await** keyword:

      ```
      await client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
      ```

   3. Update the previous line of code by adding more code to store the result of the expression in a new variable named *messages* of type **Response<QueueMessage[]>**:

      ```
      Response<QueueMessage[]> messages = await client.ReceiveMessagesAsync(batchSize, visibilityTim
      ```

4. Within the **Main** method, perform the following actions to iterate over and render the properties of each message:

   1. Add the following line of code to create a **foreach** loop that iterates over each message that's stored in the **Value** property of the *messages* variable of type **QueueMessage[]**:

      ```
      foreach(QueueMessage message in messages?.Value)
      {
      }
      ```

   2. Within the **foreach** loop, add another line of code to render the **MessageId** and **MessageText** properties of each **QueueMessage** instance:

      ```
      Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
      ```

5. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    // Existing code removed for brevity

    Console.WriteLine($"---Existing Messages---");
    int batchSize = 10;
    TimeSpan visibilityTimeout = TimeSpan.FromSeconds(2.5d);

    Response<QueueMessage[]> messages = await client.ReceiveMessagesAsync(batchSize, visibilityTim

    foreach(QueueMessage message in messages?.Value)
    {
        Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
    }
}
```

6. Save the **Program.cs** file.
7. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
8. At the open command prompt, enter the following command, and then select Enter to build the .NET web application:

   ```
   dotnet build
   ```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.
9. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

40.2.4.2 Task 2: Test message queue access

1. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
2. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

   ```
   dotnet run
   ```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.
3. Observe the output from the currently running console application. The output indicates that no messages are in the queue.
4. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.
5. In the Azure portal's navigation pane, select the **Resource groups** link.
6. On the **Resource groups** blade, find and then select the **AsyncProcessor** resource group that you created earlier in this lab.
7. On the **AsyncProcessor** blade, select the asyncstor*[yourname]*** storage account that you created earlier in this lab.
8. On the **Storage account** blade, select **Overview**.
9. In the **Overview** section, select **Open in Explorer**.
10. In the **Azure Storage Explorer** window, select **Open Azure Storage Explorer**.

    Note: If this is your first time opening Storage Explorer by using the portal, you might be prompted to allow the portal to open these types of links in the future. You should accept the prompt.

11. In the **Azure Storage Explorer** application, you will notice a prompt to sign in to your Azure account. Sign in by performing the following actions:
    1. In the popup dialog, select **Sign in**.
    2. In the **Connect to Azure Storage** window, select **Add an Azure Account**, in the **Azure environment** list select **Azure**, and then select **Next**.
    3. In the **Sign in to your account** popup window, enter the email address for your Microsoft account, and then select **Next**.
    4. Still within the **Sign in to your account** popup window, enter the password for your Microsoft account, and then select **Sign in**.
    5. In the **ACCOUNT MANAGEMENT** pane, select **Apply**.
    6. Observe that you are returned back to the **EXPLORER** pane with your subscription information populated.
12. From the **Azure Storage Explorer** application, in the **EXPLORER** pane, find and expand the asyncstor*[yourname]*** storage account that you created earlier in this lab.
13. Within the asyncstor*[yourname]*** storage account, find and expand the **Queues** node.
14. In the **Queues** node, open the **messagequeue** queue that you created earlier in this lab by using .NET code.
15. On the **messagequeue** tab, select **Add Message**.
16. In the **Add Message** pop-up window, perform the following actions:
    1. In the **Message text** text box, enter the value **Hello World**.
    2. In the **Expires in** text box, enter the value **12**.
    3. In the **Expires in** drop-down list, select **Hours**.
    4. Ensure that the **Encode message body in Base 64** check box isn't selected.
    5. Select **OK**.
17. Return to the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
18. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.
19. Observe the output from the currently running console application. The output includes the new message that you created.
20. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

40.2.4.3 Task 3: Delete queued messages

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, find the existing **foreach** loop within the **Main** method:

    ```
    foreach(QueueMessage message in messages?.Value)
    {
        Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
    }
    ```
3. Within the **foreach** loop, add a new line of code to invoke the **DeleteMessageAsync** method of the **QueueMessage** class, passing in the **MessageId** and **PopReceipt** properties of the *message* variable:

    ```
    await client.DeleteMessageAsync(message.MessageId, message.PopReceipt);
    ```

4. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    // Existing code removed for brevity

    foreach(QueueMessage message in messages?.Value)
    {
        Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
        await client.DeleteMessageAsync(message.MessageId, message.PopReceipt);
    }
}
```

5. **Save** the **Program.cs** file.
6. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
7. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

8. Observe the output from the currently running console application. The message that you created earlier in the lab still exists because it hasn't been deleted previously.
9. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.
10. Return Storage Explorer, and then find and expand the asyncstor*[yourname]*** storage account that you created earlier in this lab.
11. In the asyncstor*[yourname]*** storage account, find and expand the **Queues** node.
12. In the **Queues** node, open the **messagequeue** queue that you created earlier in this lab by using .NET code.
13. Observe the empty list of messages in the queue.

    Note: You might need to refresh the queue.

40.2.4.4 Review

In this exercise, you read and deleted existing messages from the Storage queue by using the .NET library.

40.2.5 Exercise 4: Queue new messages by using .NET

40.2.5.1 Task 1: Write code to create queue messages

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, find the existing **Main** method.
3. Within the **Main** method, add a new line of code to render a header for the "New Messages" section:

```
Console.WriteLine($"---New Messages---");
```

4. In the **Main** method, perform the following actions to create and send a message asynchronously:
   1. Add the following line of code to create a new string variable named *greeting* with a value of **Hi, Developer!**:

```
string greeting = "Hi, Developer!";
```

   2. Add the following line of code to invoke the **SendMessageAsync** method of the **QueueClient** class by using the *greeting* variable as a parameter

```
await client.SendMessageAsync(Convert.ToBase64String(Encoding.UTF8.GetBytes(greeting)));
```

   3. Add the following line of code to render the content of the message that you sent:

```
    Console.WriteLine($"Sent Message:\t{greeting}");
```

5. Observe the **Main** method, which should now include:

```
public static async Task Main(string[] args)
{
    // Existing code removed for brevity

    Console.WriteLine($"---New Messages---");
    string greeting = "Hi, Developer!";
    await client.SendMessageAsync(Convert.ToBase64String(Encoding.UTF8.GetBytes(greeting)));

    Console.WriteLine($"Sent Message:\t{greeting}");
}
```

6. **Save** the **Program.cs** file.
7. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
8. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

9. Observe the output from the currently running console application. The content of the new message that you sent should be in the output.
10. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

40.2.5.2 Task 2: View queued messages by using Storage Explorer

1. Return to Storage Explorer, and then find and expand the asyncstor*[yourname]*** storage account that you created earlier in this lab.
2. In the asyncstor*[yourname]*** storage account, find and expand the **Queues** node.
3. In the **Queues** node, open the **messagequeue** queue that you created earlier in this lab by using .NET code.
4. Observe the single new message in the list of messages in the queue.

> **Note**: You might need to refresh the queue.

40.2.5.3 Review

In this exercise, you created new messages in the queue by using the .NET library for Storage queues.

40.2.6 Exercise 5: Clean up your subscription

40.2.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

> **Note**: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   - A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

40.2.6.2 Task 2: Delete a resource group

1. At the command prompt, enter the following command, and then select Enter to delete the **AsyncProcessor** resource group:

   ```
   az group delete --name AsyncProcessor --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

40.2.6.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.
3. Close the currently running Azure Storage Explorer application.

40.2.6.4 Review

40.3 In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

40.4 lab: az204Title: 'Lab 10: Asynchronously processing messages by using Azure Queue Storage' az020Title: 'Lab 10: Asynchronously processing messages by using Azure Queue Storage' az204Module: 'Module 10: Develop message-based solutions' az020Module: 'Module 10: Develop message-based solutions'

41 Lab 10: Asynchronously processing messages by using Azure Queue Storage

42 Student lab manual

42.1 Lab scenario

You're studying various ways to communicate between isolated service components in Microsoft Azure, and you have decided to evaluate the Azure Storage service and its Queue service offering. As part of this evaluation, you'll build a prototype application in .NET that can send and receive messages so that you can measure the complexity involved in using this service. To help you with your evaluation, you've also decided to use Azure Storage Explorer as the queue message producer/consumer throughout your tests.

42.2 Objectives

After you complete this lab, you will be able to:

- Add **Azure.Storage** libraries from NuGet.
- Create a queue in .NET.
- Produce a new message in the queue by using .NET.
- Consume a message from the queue by using .NET.
- Manage a queue by using Storage Explorer.

42.3 Lab setup

- Estimated time: **45 minutes**

42.4 Instructions

42.4.1 Before you start

42.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

42.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Visual Studio Code
- Storage Explorer

42.4.2 Exercise 1: Create Azure resources

42.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

42.4.2.2 Task 2: Create a Storage account

1. Create a new storage account with the following details:
   - New resource group: **AsyncProcessor**
   - Name: asyncstor*[yourname]***
   - Location: **(US) East US**
   - Performance: **Standard**
   - Account kind: **StorageV2 (general purpose v2)**
   - Replication: **Locally-redundant storage (LRS)**

     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.
2. Find the **Access Keys** blade of your newly created storage account instance.
3. Record the value of the **Connection string** text box. You'll use this value later in this lab.

42.4.2.3 Review

In this exercise, you created a new Storage account that you'll use through the remainder of the lab.

42.4.3 Exercise 2: Configure the Azure Storage SDK in a .NET project

42.4.3.1 Task 1: Create a .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\10\Starter\MessageProcessor** folder.
2. Using a terminal, create a new .NET project named **MessageProcessor** in the current folder:

   ```
   dotnet new console --name MessageProcessor --output .
   ```

   Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.
3. Using the same terminal, import version 12.0.0 of **Azure.Storage.Queues** from NuGet:

```
dotnet add package Azure.Storage.Queues --version 12.0.0
```

> **Note**: The **dotnet add package** command will add the **Azure.Storage.Queues** package from NuGet. For more information, go to Azure.Storage.Queues.

4. Using the same terminal, build the .NET web application:

```
dotnet build
```

5. Close the current terminal.

42.4.3.2 Task 2: Write code to access Azure Storage

1. Open the **Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that will be referenced by the application:

```
using Azure;
using Azure.Storage.Queues;
using Azure.Storage.Queues.Models;
using System;
using System.Text;
using System.Threading.Tasks;
```

4. Create a new **Program** class with two constant string properties named **storageConnectionString** and **messagequeue**, and then create an asynchronous **Main** entry point method:

```
public class Program
{
    private const string storageConnectionString = "<storage-connection-string>";
    private const string queueName = "messagequeue";

    public static async Task Main(string[] args)
    {
    }
}
```

5. Update the **storageConnectionString** string constant by setting its value to the **Connection string** of the storage account that you recorded earlier in this lab.

42.4.3.3 Task 3: Validate Azure Storage access

1. In the **Main** method, add the following block of code to connect to the storage account and to asynchronously create the queue if it doesn't already exist:

```
QueueClient client = new QueueClient(storageConnectionString, queueName);
await client.CreateAsync();
```

2. Still in the **Main** method, add the following block of code to render the Uniform Resource Identifier (URI) of the queue endpoint:

```
Console.WriteLine($"---Account Metadata---");
Console.WriteLine($"Account Uri:\t{client.Uri}");
```

3. Save the **Program.cs** file.
4. Using a terminal, run the ASP.NET web application project:

```
dotnet run
```

> **Note**: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

5. Observe the output from the currently running console application. The output contains metadata for the queue endpoint.
6. Close the current terminal.

42.4.3.4 Review

In this exercise, you configured your .NET project to access the Storage service and manipulate a queue made available through the service.

42.4.4 Exercise 3: Read messages from the queue

42.4.4.1 Task 1: Write code to access queue messages

1. In the **Main** method, perform the following actions:
    1. Render a header by using the **Console.WriteLine** static method:

        ```
        Console.WriteLine($"---Existing Messages---");
        ```

    2. Add the following block of code to create variables that will be used when retrieving queue messages:

        ```
        int batchSize = 10;
        TimeSpan visibilityTimeout = TimeSpan.FromSeconds(2.5d);
        ```

    3. Add the following block of code to retrieve a batch of messages asynchronously from the queue service and iterate over the messages:

        ```
        Response<QueueMessage[]> messages = await client.ReceiveMessagesAsync(batchSize, visibilityTim
        foreach(QueueMessage message in messages?.Value)
        {
        }
        ```

    4. Within the **foreach** block, render the **MessageId** and **MessageText** properties of the **QueueMessage** instance:

        ```
        Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
        ```

2. Save the **Program.cs** file.
3. Using a terminal, build the ASP.NET web application project:

    ```
    dotnet build
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

4. Close the current terminal.

42.4.4.2 Task 2: Test message queue access

1. Using a terminal, run the ASP.NET web application project:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

2. Observe the output from the currently running console application. The output indicates that no messages are in the queue.
3. Close the current terminal.
4. Find the asyncstor*[yourname]*** Storage account that you created earlier in this lab.
5. In the **Overview** section of the blade, select **Open in Explorer** to open the Storage account by using Storage Explorer.
6. In the **Azure Storage Explorer** application, sign in to your Azure account.
7. From the **Azure Storage Explorer** application, in the **EXPLORER** pane, find and expand the asyncstor*[yourname]*** storage account that you created earlier in this lab.
8. Within the asyncstor*[yourname]*** node for the Storage account that you created earlier in this lab, find and open the **messagequeue** queue.
9. Add a new message to the queue with the following properties:
    - Message text: **Hello World**

- Expires in: **12 Hours**
- Encode message body in Base 64: **No**

10. Return to the Visual Studio Code application.
11. Using a terminal, run the ASP.NET web application project:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

12. Observe the output from the currently running console application. The output includes the new message that you created.
13. Close the current terminal.

42.4.4.3 Task 3: Delete queued messages

1. In the **Visual Studio Code** window, open the **Program.cs** file.
2. Update the **foreach** loop's block of code to invoke the **DeleteMessageAsync** method of the **QueueMessage** class, passing in the **MessageId** and **PopReceipt** properties of the *message* variable:

    ```
    foreach(QueueMessage message in messages?.Value)
    {
        Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
        await client.DeleteMessageAsync(message.MessageId, message.PopReceipt);
    }
    ```

3. **Save** the **Program.cs** file.
4. Using a terminal, run the ASP.NET web application project:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

5. Observe the output from the currently running console application. The message that you created earlier in the lab still exists because it hasn't been deleted previously.
6. Close the current terminal.
7. In the **Azure Storage Explorer** window, within the asyncstor*[yourname]*** node for the Storage account that you created earlier in this lab, find and open the **messagequeue** queue.
8. Observe the empty list of messages in the queue.

    Note: You might need to refresh the queue.

42.4.4.4 Review

In this exercise, you read and deleted existing messages from the Storage queue by using the .NET library.

42.4.5 Exercise 4: Queue new messages by using .NET

42.4.5.1 Task 1: Write code to create queue messages

1. In the **Visual Studio Code** window, open the **Program.cs** file.
2. In the **Main** method, perform the following actions:
    1. Render a header by using the **Console.WriteLine** static method:

        ```
        Console.WriteLine($"---New Messages---");
        ```

    2. Add the following block of code to create a new string variable named *greeting* with a value of **Hi, Developer!**, and then invoke the **SendMessageAsync** method of the **QueueClient** class by using the *greeting* variable as a parameter:

        ```
        string greeting = "Hi, Developer!";
        await client.SendMessageAsync(Convert.ToBase64String(Encoding.UTF8.GetBytes(greeting)));
        ```

3. Add the following block of code to render the content of the message that you sent:

   ```
   Console.WriteLine($"Sent Message:\t{greeting}");
   ```

3. **Save** the **Program.cs** file.
4. Using a terminal, run the ASP.NET web application project:

   ```
   dotnet run
   ```

   Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

5. Observe the output from the currently running console application. The content of the new message that you sent should be in the output.
6. Close the current terminal.

42.4.5.2 Task 2: View queued messages by using Storage Explorer

1. In the **Azure Storage Explorer** window, within the asyncstor*[yourname]*** node for the Storage account that you created earlier in this lab, find and open the **messagequeue** queue.
2. Observe the single new message in the list of messages in the queue.

   Note: You might need to refresh the queue.

42.4.5.3 Review

In this exercise, you created new messages in the queue by using the .NET library for Storage queues.

42.4.6 Exercise 5: Clean up your subscription

42.4.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

42.4.6.2 Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **AsyncProcessor** resource group:

   ```
   az group delete --name AsyncProcessor --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

42.4.6.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.
3. Close the currently running Azure Storage Explorer application.

42.4.6.4 Review

42.5 In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

42.6 lab: az204Title: 'Lab 11: Monitoring services that are deployed to Azure' az020Title: 'Lab 11: Monitoring services that are deployed to Azure' az204Module: 'Module 11: Monitor and optimize Azure solutions' az020Module: 'Module 11: Monitor and optimize Azure solutions' type: 'Answer Key'

43 Lab 11: Monitoring services that are deployed to Azure

44 Student lab answer key

44.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention. However, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

44.2 Instructions

44.2.1 Before you start

44.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

44.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Visual Studio Code
- Windows PowerShell

44.2.2 Exercise 1: Create and configure Azure resources

44.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. From the sign-in page, enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, a dialog box will display an offer to tour the portal. Select **Get Started** to skip the tour and begin using the portal.

44.2.2.2 Task 2: Create an Application Insights resource

1. In the Azure portal's navigation pane, select **Create a resource**.
2. From the **New** blade, find the **Search the Marketplace** text box.

3. In the search box, enter **Insights**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Application Insights** result.
5. From the **Application Insights** blade, select **Create**.
6. Find the tabs from the second **Application Insights** blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new Application Insights instance. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **MonitoredAssets**, and then select **OK**.
   3. In the **Name** text box, enter instrm*[yourname]***.
   4. In the **Region** drop-down list, select the **(US) East US** region.
   5. In the **Resource Mode** section, select the **Classic** option.
   6. Select **Review + Create**.
8. From the **Review + Create** tab, review the options that you selected during the previous steps.
9. Select **Create** to create the Application Insights instance by using your specified configuration.

   Note: Wait for the creation task to complete before you move forward with this lab.
10. In the Azure portal's navigation pane, select **Resource groups**.
11. From the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
12. From the **MonitoredAssets** blade, select the instrm*[yourname]*** Application Insights account that you created earlier in this lab.
13. From the **Application Insights** blade, in the **Configure** category, select the **Properties** link.
14. In the **Properties** section, find the value of the **Instrumentation Key** text box. This key is used by client applications to connect to Application Insights.

44.2.2.3 Task 3: Create a web app by using Azure App Services resource

1. In the Azure portal's navigation pane, select **Create a resource**.
2. From the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Web**, and then select Enter.
4. From the **Marketplace** search results blade, select the **Web App** result.
5. From the **Web App** blade, select **Create**.
6. Find the tabs from the second **Web App** blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new web app. You can select **Review + Create** at any time to skip the remaining tabs.
7. From the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** drop-down list, select **MonitoredAssets**.
   3. In the **Name** text box, enter ***smpapi***[yourname]***.
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET Core 3.1 (LTS)**.
   6. In the **Operating System** section, select **Windows**.
   7. In the **Region** drop-down list, select the **East US** region.

8. In the **Windows Plan (East US)** section, select **Create new**, enter the value **MonitoredPlan** into the **Name** text box, and then select **OK**.
9. Leave the **SKU and size** section set to its default value.
10. Select **Next: Monitoring**.

8. From the **Monitoring** tab, perform the following actions:
    1. In the **Enable Application Insights** section, select **Yes**.
    2. In the **Application Insights** drop-down list, select the instrm*[yourname]*** Application Insights account that you created earlier in this lab.
    3. Select **Review + Create**.
9. From the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the web app by using your specified configuration.

    Note: Wait for the creation task to complete before you move forward with this lab.
11. In the Azure portal's navigation pane, select **Resource groups**.
12. From the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
13. From the **MonitoredAssets** blade, select the ***smpapi****[yourname]*** web app that you created earlier in this lab.
14. From the **App Service** blade, in the **Settings** category, select the **Configuration** link.
15. In the **Configuration** section, perform the following actions:
    1. Select the **Application settings** tab.
    2. Select **Show Values** to get the secrets associated with your API.
    3. Find the value corresponding to the **APPINSIGHTS_INSTRUMENTATIONKEY** key. This value was set automatically when you built your Web Apps resource.
16. From the **App Service** blade, in the **Settings** category, select the **Properties** link.
17. In the **Properties** section, record the value of the **URL** text box. You'll use this value later in the lab to make requests against the API.

44.2.2.4 Task 4: Configure web app autoscale options

1. From the **App Service** blade, in the **Settings** category, select the **Scale out (App Service Plan)** link.
2. In the **Scale out** section, perform the following actions:
    1. Select **Custom autoscale**.
    2. In the **Autoscale setting name** text box, enter **ComputeScaler**.
    3. In the **Resource group** list, select **MonitoredAssets**.
    4. In the **Scale mode** section, select **Scale based on a metric**.
    5. In the **Minimum** text box in the **Instance limits** section, enter **2**.
    6. In the **Maximum** text box in the **Instance limits** section, enter **8**.
    7. In the **Default** text box in the **Instance limits** section, enter **3**.
    8. Select **Add a rule**. In the **Scale rule** pop-up dialog, leave all boxes set to their default values, and then select **Add**.
    9. Within the section, select **Save**.

    Note: Wait for the save operation to complete before you move forward with this lab.

44.2.2.5 Review

In this exercise, you created the resources that you'll use for the remainder of the lab.

44.2.3 Exercise 2: Monitor a local web application by using Application Insights

44.2.3.1 Task 1: Build a .NET Web API project

1. On the taskbar, select the **Visual Studio Code** icon.
2. From the **File** menu, select **Open Folder**.
3. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\11\Starter\Api**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, right-click the Explorer pane or activate the shortcut menu, and then select **Open in Terminal**.
5. At the **Open** command prompt, enter the following command, and then select Enter to create a new .NET Web API application named **SimpleApi** in the current directory:

   ```
   dotnet new webapi --output . --name SimpleApi
   ```

6. At the command prompt, enter the following command, and then select Enter to import version 2.14.0 of **Microsoft.ApplicationInsights** from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights --version 2.14.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.ApplicationInsights** package from NuGet. For more information, go to Microsoft.ApplicationInsights.

7. At the command prompt, enter the following command, and then select Enter to import version 2.14.0 of **Microsoft.ApplicationInsights.AspNetCore** from NuGet:

   ```
   dotnet add package Microsoft.ApplicationInsights.AspNetCore --version 2.14.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.ApplicationInsights.AspNetCore** package from NuGet. For more information, go to Microsoft.ApplicationInsights.AspNetCore.

8. At the command prompt, enter the following command, and then select Enter to import version 2.14.0 of **Microsoft.ApplicationInsights.PerfCounterCollector** from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights.PerfCounterCollector  --version 2.14.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.ApplicationInsights.PerfCounterCollect** package from NuGet. For more information, go to Microsoft.ApplicationInsights.PerfCounterCollector.

9. At the command prompt, enter the following command, and then select Enter to build the .NET web app:

   ```
   dotnet build
   ```

44.2.3.2 Task 2: Update application code to disable HTTPS and use Application Insights

1. In the **Visual Studio Code** window, in the Explorer pane, select the **Startup.cs** file to open the file in the editor.
2. In the editor, in the **Startup** class, find and delete the following line of code at line 39:

   ```
   app.UseHttpsRedirection();
   ```

   Note: This line of code forces the web app to use HTTPS. For this lab, this is unnecessary.

3. In the **Startup** class, add a new static string constant named **INSTRUMENTATION_KEY** with its value set to the instrumentation key that you copied from the Application Insights resource you created earlier in this lab:

   ```
   private const string INSTRUMENTATION_KEY = "{your_instrumentation_key}";
   ```

   Note: For example, if your instrumentation key is `d2bb0eed-1342-4394-9b0c-8a56d21aaa43`, your line of code would be `private const string INSTRUMENTATION_KEY = "d2bb0eed-1342-4394-9b0c-8a56d`

4. Find the **ConfigureServices** method in the **Startup** class:

   ```
   public void ConfigureServices(IServiceCollection services)
   {
       services.AddControllers();
   }
   ```

5. Add a new line of code at the end of the **ConfigureServices** method to configure Application Insights using the provided instrumentation key:

   ```
   services.AddApplicationInsightsTelemetry(INSTRUMENTATION_KEY);
   ```

6. Observe the **ConfigureServices** method, which should now include:

   ```
   public void ConfigureServices(IServiceCollection services)
   {
       services.AddControllers();
       services.AddApplicationInsightsTelemetry(INSTRUMENTATION_KEY);
   }
   ```

7. Save the **Startup.cs** file.

8. At the command prompt, enter the following command, and then select Enter to build the .NET web application.

   ```
   dotnet build
   ```

44.2.3.3 Task 3: Test an API application locally

1. At the command prompt, enter the following command, and then select Enter to run the .NET web application.

   ```
   dotnet run
   ```

2. On the taskbar, open the context menu for the **Microsoft Edge** icon, and then open a new browser window.

3. In the open browser window, go to the **/weatherforecast** relative path of your test application that's hosted at **localhost** on port **5000**.

   Note: The full URL is http://localhost:5000/weatherforecast

4. Close the browser window that's displaying the http://localhost:5000/weatherforecast address.

5. Close the currently running Visual Studio Code application.

44.2.3.4 Task 4: Get metrics in Application Insights

1. Return to your currently open browser window that's displaying the Azure portal.

2. In the portal, select **Resource groups**.

3. From the **Resource groups** blade, find and select the **MonitoredAssets** resource group that you created earlier in this lab.

4. From the **MonitoredAssets** blade, select the instrm*[yourname]*** Application Insights account that you created earlier in this lab.

5. From the **Application Insights** blade, in the tiles in the center of the blade, find the displayed metrics. Specifically, find the number of server requests that have occurred and the average server response time.

   Note: It can take up to five minutes to observe requests in the Application Insights metrics charts.

44.2.3.5 Review

In this exercise, you created an API by using ASP.NET and configured it to stream application metrics to Application Insights. You then used the Application Insights dashboard to get performance details about your API.

44.2.4 Exercise 3: Monitor a web app using Application Insights

44.2.4.1 Task 1: Deploy an application to the web app

1. On the taskbar, select the **Visual Studio Code** icon.

2. From the **File** menu, select **Open Folder**.

3. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\11\Starter\Api**, and then select **Select Folder**.
4. In the Visual Studio Code window, right-click the Explorer pane or activate the shortcut menu, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to sign in to the Azure Command-Line Interface (CLI):

   ```
   az login
   ```

6. In the browser window, perform the following actions:
   1. Enter the email address for your Microsoft account, and then select **Next**.
   2. Enter the password for your Microsoft account, and then select **Sign in**.
7. Return to the currently open command prompt application.

   Note: Wait for the sign-in process to finish.
8. At the command prompt, enter the following command, and then select Enter to list all the apps in your **MonitoredAssets** resource group:

   ```
   az webapp list --resource-group MonitoredAssets
   ```

9. Enter the following command, and then select Enter to find the apps that have the prefix **smpapi***:

   ```
   az webapp list --resource-group MonitoredAssets --query "[?starts_with(name, 'smpapi')]"
   ```

10. Enter the following command, and then select Enter to render out only the name of the single app that has the **smpapi***:

    ```
    az webapp list --resource-group MonitoredAssets --query "[?starts_with(name, 'smpapi')].{Name:name
    ```

11. Enter the following command, and then select Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\11\Starter** directory that contains the deployment files:

    ```
    cd F:\Allfiles\Labs\11\Starter\
    ```

12. Enter the following command, and then select Enter to deploy the **api.zip** file to the web app that you created earlier in this lab:

    ```
    az webapp deployment source config-zip --resource-group MonitoredAssets --src api.zip --name <name
    ```

    Note: Replace the *name-of-your-api-app* placeholder with the name of the web app that you created earlier in this lab. You recently queried this app's name in the previous steps.

    Note: Wait for the deployment to complete before you move forward with this lab.
13. Close the currently running Visual Studio Code application.
14. Return to your currently open browser window that's displaying the Azure portal.
15. In the Azure portal's navigation pane, select **Resource groups**.
16. From the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
17. From the **MonitoredAssets** blade, select the ***smpapi****[yourname]** web app that you created earlier in this lab.
18. From the **App Service** blade, select **Browse**. A new browser window or tab will open and return a "404 (Not Found)" error.
19. In the browser address bar, update the URL by appending the suffix **/weatherforecast** to the end of the current URL, and then select Enter.

    Note: For example, if your URL is https://smpapistudent.azurewebsites.net, the new URL would be https://smpapistudent.azurewebsites.net/weatherforecast.
20. Find the JavaScript Object Notation (JSON) array that's returned as a result of using the API.

44.2.4.2 Task 2: Configure in-depth metric collection for Web Apps

1. Return to your currently open browser window that's displaying the Azure portal.
2. In the Azure portal's navigation pane, select **Resource groups**.
3. From the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
4. From the **MonitoredAssets** blade, select the ***smpapi****[yourname]*** web app that you created earlier in this lab.
5. From the **App Service** blade, select **Application Insights**.
6. From the **Application Insights** blade, perform the following actions:
    1. Ensure that the **Application Insights** section is set to **Enable**.
    2. In the **Instrument your application** section, select the **.NET** tab.
    3. In the **Collection level** section, select **Recommended**.
    4. In the **Profiler** secton, select **On**.
    5. In the **Snapshot debugger** section, select **Off**.
    6. In the **SQL Commands** section, select **Off**.
    7. Select **Apply**.
    8. In the confirmation dialog, select **Yes**.
7. Close the **Application Insights** blade.
8. Back from the **App Service** blade, select **Browse**. A new browser window or tab will open and return a "404 (Not Found)" error.
9. In the browser address bar, update the URL by appending the suffix **/weatherforecast** to the end of the current URL, and then select Enter.

    Note: For example, if your URL is https://smpapistudent.azurewebsites.net, the new URL would be https://smpapistudent.azurewebsites.net/weatherforecast.
10. Observe the JSON array that's returned as a result of using the API.
11. Record the URL that you used to access the JSON array.

    Note: Using the example from the previous step, you would record the URL `https://smpapistudent.azurewebsi`

44.2.4.3 Task 3: Get updated metrics in Application Insights

1. Return to your currently open browser window that's displaying the Azure portal.
2. In the portal, select **Resource groups**.
3. From the **Resource groups** blade, find and select the **MonitoredAssets** resource group that you created earlier in this lab.
4. From the **MonitoredAssets** blade, select the instrm*[yourname]** Application Insights account that you created earlier in this lab.
5. From the **Application Insights** blade, in the tiles in the center of the blade, find the displayed metrics. Specifically, find the number of server requests that have occurred and the average server response time.

    Note: It can take up to five minutes to observe requests in the Application Insights metrics charts.

44.2.4.4 Task 4: View real-time metrics in Application Insights

1. Return to your currently open browser window that's displaying the Azure portal.
2. In the portal, select **Resource groups**.
3. From the **Resource groups** blade, find and select the **MonitoredAssets** resource group that you created earlier in this lab.

4. From the **MonitoredAssets** blade, select the instrm*[yourname]*** Application Insights account that you created earlier in this lab.
5. From the **Application Insights** blade, select **Live Metrics Stream** in the **Investigate** section.
6. On the taskbar, open the context menu for the **Microsoft Edge** icon, and then open a new browser window.
7. In the new browser window, go to the URL that you recorded earlier in this lab.
8. Observe the JSON array result.
9. Return to your currently open browser window that's displaying the Azure portal.
10. Observe the updated **Live Metrics Stream** blade.

    Note: The **Incoming Requests** section should update within seconds, showing the requests that you made to the web app.

44.2.4.5 Review

In this exercise, you deployed your web application to Azure App Service and monitored your metrics from the same Application Insights instance.

44.2.5 Exercise 4: Clean up your subscription

44.2.5.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).
2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

      Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice the Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

44.2.5.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **MonitoredAssets** resource group:

   ```
   az group delete --name MonitoredAssets --no-wait --yes
   ```
2. Close the Cloud Shell pane in the portal.

44.2.5.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

44.2.5.4 Review

44.3 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

44.4 lab: az204Title: 'Lab 11: Monitoring services that are deployed to Azure' az020Title: 'Lab 11: Monitoring services that are deployed to Azure' az204Module: 'Module 11: Monitor and optimize Azure solutions' az020Module: 'Module 11: Monitor and optimize Azure solutions'

45 Lab 11: Monitoring services that are deployed to Azure

46 Student lab manual

46.1 Lab scenario

You have created an API for your next big startup venture. Even though you want to get to market quickly, you have witnessed other ventures fail when they don't plan for growth and have too few resources or too many users. To plan for this, you have decided to take advantage of the scale-out features of Microsoft Azure App Service, the telemetry features of Application Insights, and the performance-testing features of Azure DevOps.

46.2 Objectives

After you complete this lab, you will be able to:

- Create an Application Insights resource.
- Integrate Application Insights telemetry tracking into an ASP.NET web app and a resource built using the Web Apps feature of Azure App Service.

46.3 Lab setup

- Estimated time: **45 minutes**

46.4 Instructions

46.4.1 Before you start

46.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

46.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer
- Visual Studio Code
- Windows PowerShell

46.4.2 Exercise 1: Create and configure Azure resources

46.4.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. Sign in to the Azure portal (https://portal.azure.com).
3. At the sign-in page, enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

Note: If this is your first time signing in to the Azure portal, a dialog box will display offering a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

46.4.2.2 Task 2: Create an Application Insights resource

1. Create a new Application Insights account with the following details:
   - New resource group: **MonitoredAssets**
   - Name: instrm*[yourname]***
   - Region: **(US) East US**
   - Resource Mode: **Classic**

     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.
2. Access the **Properties** section of the **Application Insights** blade.
3. Get the value of the **Instrumentation Key** text box. This key is used by client applications to connect to Application Insights.

46.4.2.3 Task 3: Create a web app by using Azure App Services resource

1. Create a new **web app** with the following details:
   - Existing resource group: **MonitoredAssets**
   - Web App name: ***smpapi****[yourname]***
   - Publish: **Code**
   - Runtime stack: **.NET Core 3.1 (LTS)**
   - Operating System: **Windows**
   - Region: **East US**
   - New App Service plan: **MonitoredPlan**
   - SKU and size: **Standard (S1)**
   - Application Insights: **Enabled**
   - Existing Application Insights resource: instrm*[yourname]***

     Note: Wait for Azure to finish creating the web app before you move forward with the lab. You'll receive a notification when the app is created.
2. Access the web app with a prefix of **smpapi*** that you created earlier in this lab.
3. In the **Settings** section, go to the **Configuration** section.
4. Find and access the **Application Settings** tab in the **Configuration** section.
5. Get the value corresponding to the **APPINSIGHTS_INSTRUMENTATIONKEY** application settings key. This value was set automatically when you built your web app.
6. Access the **Properties** section of the **App Service** blade.
7. Record the value of the **URL** text box. You'll use this value later in the lab to make requests against the API.

46.4.2.4 Task 4: Configure web app autoscale options

1. Go to the **Scale out** section of the **App Services** blade.
2. In the **Scale out** section, enable **Custom autoscale** with the following details:
   - Name: **ComputeScaler**
   - In the **Scale mode** section, select **Scale based on a metric**.
   - Minimum instances: **2**

- Maximum instances: **8**
- Default instances: **3**
- Scale rules: **Single scale-out rule with default values**

3. Save your changes to the **autoscale** configuration.

46.4.2.5 Review

In this exercise, you created the resources that you'll use for the remainder of the lab.

46.4.3 Exercise 2: Monitor a local web application by using Application Insights

46.4.3.1 Task 1: Build a .NET Web API project

1. Open Visual Studio Code.
2. In Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\11\Starter\Api** folder.
3. Use the Explorer pane in Visual Studio Code to open a new terminal that has the context set to the current working directory.
4. At the command prompt, create a new .NET Web API application named **SimpleApi** in the current directory:

   ```
   dotnet new webapi --output . --name SimpleApi
   ```

5. Import version 2.14.0 of **Microsoft.ApplicationInsights** from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights --version 2.14.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.ApplicationInsights** package from NuGet. For more information, go to Microsoft.ApplicationInsights.

6. Import version 2.14.0 of **Microsoft.ApplicationInsights.AspNetCore** from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights.AspNetCore --version 2.14.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.ApplicationInsights.AspNetCore** package from NuGet. For more information, go to Microsoft.ApplicationInsights.AspNetCore.

7. Import version 2.14.0 of **Microsoft.ApplicationInsights.PerfCounterCollector** from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights.PerfCounterCollector  --version 2.14.0
   ```

   Note: The **dotnet add package** command will add the **Microsoft.ApplicationInsights.PerfCounterCollect** package from NuGet. For more information, go to Microsoft.ApplicationInsights.PerfCounterCollector.

8. Build the .NET web app:

   ```
   dotnet build
   ```

46.4.3.2 Task 2: Update application code to disable HTTPS and use Application Insights

1. Use the Explorer in Visual Studio Code to open the **Startup.cs** file in the editor.
2. Find and delete the following line of code at line 39:

   ```
   app.UseHttpsRedirection();
   ```

   Note: This line of code forces the web app to use HTTPS. For this lab, this is unnecessary.

3. In the **Startup** class, add a new static string constant named **INSTRUMENTATION_KEY** with its value set to the instrumentation key that you copied from the Application Insights resource you created earlier in this lab:

   ```
   private static string INSTRUMENTATION_KEY = "{your_instrumentation_key}";
   ```

   Note: For example, if your instrumentation key is `d2bb0eed-1342-4394-9b0c-8a56d21aaa43`, your line of code would be `private static string INSTRUMENTATION_KEY = "d2bb0eed-1342-4394-9b0c-8a56`

4. Add a new line of code in the **ConfigureServices** method to configure Application Insights using the provided instrumentation key:

   ```
   services.AddApplicationInsightsTelemetry(INSTRUMENTATION_KEY);
   ```

5. Save the **Startup.cs** file.
6. If it's not already open, use the Explorer in Visual Studio Code to open a new terminal with the context set to the current working directory.
7. Build the .NET web app:

   ```
   dotnet build
   ```

46.4.3.3 Task 3: Test an API application locally

1. If it's not already open, use the Explorer in Visual Studio Code to open a new terminal with the context set to the current working directory.
2. Run the .NET web app.

   ```
   dotnet run
   ```

3. Open a new **Microsoft Edge** browser window.
4. In the open browser window, go to the **/weatherforecast** relative path of your test application that's hosted at **localhost** on port **5000**.

   Note: The full URL is http://localhost:5000/weatherforecast.

5. Close the browser window that you recently opened.
6. Close the currently running Visual Studio Code application.

46.4.3.4 Task 4: Get metrics in Application Insights

1. Return to your currently open browser window that's displaying the Azure portal.
2. Access the instrm*[yourname]*** Application Insights account that you created earlier in this lab.
3. From the **Application Insights** blade, find the metrics displayed in the tiles in the center of the blade. Specifically, find the number of server requests that have occurred and the average server response time.

   Note: It can take up to five minutes to observe requests in the Application Insights metrics charts.

46.4.3.5 Review

In this exercise, you created an API by using ASP.NET and configured it to stream application metrics to Application Insights. You then used the Application Insights dashboard to get performance details about your API.

46.4.4 Exercise 3: Monitor a web app using Application Insights

46.4.4.1 Task 1: Deploy an application to the web app

1. Open Visual Studio Code.
2. In Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\11\Starter\Api** folder.
3. Use the Explorer in Visual Studio Code to open a new terminal with the context set to the current working directory.
4. Sign in to the Azure Command-Line Interface (CLI) by using your Azure credentials:

   ```
   az login
   ```

5. List all the apps in your **MonitoredAssets** resource group:

   ```
   az webapp list --resource-group MonitoredAssets
   ```

6. Find the apps that have the prefix **smpapi***:

   ```
   az webapp list --resource-group MonitoredAssets --query "[?starts_with(name, 'smpapi')]"
   ```

7. Print only the name of the single app that has the prefix **smpapi***:

   ```
   az webapp list --resource-group MonitoredAssets --query "[?starts_with(name, 'smpapi')].{Name:name
   ```

8. Change the current directory to the **Allfiles (F):\Allfiles\Labs\11\Starter** directory that contains the lab files:

   ```
   cd F:\Allfiles\Labs\11\Starter\
   ```

9. Deploy the **api.zip** file to the web app that you created earlier in this lab:

   ```
   az webapp deployment source config-zip --resource-group MonitoredAssets --src api.zip --name <name
   ```

   Note: Replace the *name-of-your-api-app* placeholder with the name of the web app that you created earlier in this lab. You recently queried this app's name in the previous steps.

10. Return to your currently open browser window that's displaying the Azure portal.
11. Access the ***smpapi****[yourname]*** web app that you created earlier in this lab.
12. Open the ***smpapi****[yourname]*** web app in your browser.
13. Perform a GET request to the **/weatherforecast** relative path of the website, and then find the JavaScript Object Notation (JSON) array that's returned as a result of using the API.

    Note: For example, if your URL is https://smpapistudent.azurewebsites.net, the new URL would be https://smpapistudent.azurewebsites.net/weatherforecast.

46.4.4.2 Task 2: Configure in-depth metric collection for Web Apps

1. Return to your currently open browser window that's displaying the Azure portal.
2. Access the ***smpapi****[yourname]*** web app that you created earlier in this lab.
3. Go to the **Application Insights** configuration section.
4. **Enable** Application Insights instrumentation for **.NET** by using the following settings:
   - Collection level: **Recommended**
   - Profiler: **On**
   - Snapshot debugger: **Off**
   - SQL Commands: **Off**
5. Save your Application Insights instrumentation configuration.
6. Open the ***smpapi****[yourname]*** web app in your browser.
7. Perform a GET request to the **/weatherforecast** relative path of the website, and then observe the JSON array that's returned as a result of using the API.

   Note: For example, if your URL is https://smpapistudent.azurewebsites.net, the new URL would be https://smpapistudent.azurewebsites.net/weatherforecast.

8. Record the URL that you used to access the JSON array.

   Note: Using the example from the previous step, you would record the URL `https://smpapistudent.azurewebsi`

46.4.4.3 Task 3: Get updated metrics in Application Insights

1. Return to your currently open browser window that's displaying the Azure portal.
2. Access the instrm*[yourname]*** Application Insights account that you created earlier in this lab.
3. From the **Application Insights** blade, find the metrics displayed in the tiles in the center of the blade. Specifically, find the number of server requests that have occurred and the average server response time.

   Note: It can take up to five minutes to observe requests in the Application Insights metrics charts.

46.4.4.4 Task 4: View real-time metrics in Application Insights

1. Return to your currently open browser window that's displaying the Azure portal.
2. Access the instrm*[yourname]*** Application Insights account that you created earlier in this lab.
3. Open the **Live Metrics Stream** blade for your account.
4. Open a new **Microsoft Edge** browser window, go to the URL that you recorded earlier in this lab, and then observe the JSON array result.
5. Return to your currently open browser window that's displaying the Azure portal, and then observe the updated **Live Metrics Stream** blade.

   Note: The **Incoming Requests** section should update in seconds, showing the requests that you made to the web app.

46.4.4.5 Review

In this exercise, you deployed your web app to App Service and monitored your metrics from the same Application Insights instance.

46.4.5 Exercise 4: Clean up your subscription

46.4.5.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

46.4.5.2 Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **MonitoredAssets** resource group:

   ```
   az group delete --name MonitoredAssets --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

46.4.5.3 Task 3: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

46.4.5.4 Review

46.5 In this exercise, you cleaned up your subscription by removing the resource groups used in this lab.

46.6 lab: az204Title: 'Lab 12: Enhancing a web application by using the Azure Content Delivery Network' az204Module: 'Module 12: Integrate caching and content delivery within solutions' type: 'Answer Key'

47 Lab 12: Enhancing a web application by using the Azure Content Delivery Network

48 Student lab answer key

48.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure UI changes after the development of this training content. These changes might cause the lab instructions and lab steps to not match up.

Microsoft updates this training course when the community brings needed changes to our attention; however, because cloud updates occur frequently, you might encounter UI changes before this training content updates. **If this occurs, adapt to the changes, and then work through them in the labs as needed.**

48.2 Instructions

48.2.1 Before you start

48.2.1.1 Sign in to the lab virtual machine

Sign in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

  Note: Instructions to connect to the virtual lab environment will be provided by your instructor.

48.2.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icon for the application that you'll use in this lab:

- Microsoft Edge

48.2.2 Exercise 1: Create Azure resources

48.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, go to the Azure portal (https://portal.azure.com).
3. Enter the email address for your Microsoft account, and then select **Next**.
4. Enter the password for your Microsoft account, and then select **Sign in**.

   Note: If this is your first time signing in to the Azure portal, you'll be offered a tour of the portal. Select **Get Started** to skip the tour and begin using the portal.

48.2.2.2 Task 2: Create a Storage account

1. In the Azure portal's navigation pane, select **All services**.
2. On the **All services** blade, select **Storage Accounts**.
3. On the **Storage accounts** blade, find your list of Storage instances.
4. On the **Storage accounts** blade, select **New**.
5. Find the tabs on the **Create storage account** blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new storage account. You can select **Review + Create** at any time to skip the remaining tabs.
6. On the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **Create new**, enter **MarketingContent**, and then select **OK**.
   3. In the **Storage account name** text box, enter contenthost*[yourname]***.
   4. In the **Location** drop-down list, select the **(US) East US** region.
   5. In the **Performance** section, select **Standard**.
   6. In the **Account kind** drop-down list, select **StorageV2 (general purpose v2)**.
   7. In the **Replication** drop-down list, select **Read-access geo-redundant storage (RA-GRS)**.
   8. Select **Review + Create**.
7. On the **Review + Create** tab, review the options that you selected during the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.

   Note: Wait for the creation task to complete before you move forward with this lab.

48.2.2.3 Task 3: Create a web app by using Azure App Service

1. In the Azure portal's navigation pane, select **Create a resource**.
2. On the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **Web**, and then select Enter.
4. On the **Everything** search results blade, select the **Web App** result.
5. On the **Web App** blade, select **Create**.
6. On the second **Web App** blade, find the tabs on the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new web app. You can select **Review + Create** at any time to skip the remaining tabs.
7. On the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, select **MarketingContent**.
   3. In the **Name** text box, enter landingpage*[yourname]***.
   4. In the **Publish** section, select **Docker Container**.
   5. In the **Operating System** section, select **Linux**.
   6. In the **Region** drop-down list, select the **East US** region.
   7. In the **Linux Plan (East US)** section, select **Create new**, enter the value **MarketingPlan** in the **Name** text box, and then select **OK**.
   8. Leave the **SKU and size** section set to its default value.
   9. Select **Next: Docker**.
8. On the **Docker** tab, perform the following actions:
   1. In the **Options** drop-down list, select **Single Container**.
   2. In the **Image Source** drop-down list, select **Docker Hub**.
   3. In the **Access Type** drop-down list, select **Public**.
   4. In the **Image and tag** text box, enter **microsoftlearning/edx-html-landing-page:latest**.
   5. Select **Review + Create**.
9. On the **Review + Create** tab, review the options that you selected during the previous steps.
10. Select **Create** to create the web app by using your specified configuration.

    Note: Wait for the creation task to complete before you move forward with this lab.
11. In the Azure portal's navigation pane, select **Resource groups**.
12. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
13. On the **MarketingContent** blade, select the landingpage*[yourname]*** web app that you created earlier in this lab.
14. On the **App Service** blade, in the **Settings** category, select the **Properties** link.
15. In the **Properties** section, record the value of the **URL** text box. You'll use this value later in the lab.

48.2.2.4 Review

In this exercise, you created an Azure Storage account and an Azure Web App that you'll use later in this lab.

48.2.3 Exercise 2: Configure Content Delivery Network and endpoints

48.2.3.1 Task 1: Open Azure Cloud Shell

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign () and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

   - A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

     Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice the **Cloud Shell** configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

3. At the **Cloud Shell** command prompt in the portal, enter the following command, and then select Enter to get the version of the Azure Command-Line Interface (Azure CLI) tool:

   ```
   az --version
   ```

48.2.3.2 Task 2: Register the Microsoft.CDN provider

1. At the **Cloud Shell** command prompt in the portal, perform the following actions:

   1. Enter the following command, and then select Enter to get a list of subgroups and commands at the root level of the Azure CLI:

      ```
      az --help
      ```

   2. Enter the following command, and then select Enter to get a list of the commands that are available for resource providers:

      ```
      az provider --help
      ```

   3. Enter the following command, and then select Enter to list all currently registered providers:

      ```
      az provider list
      ```

   4. Enter the following command, and then select Enter to list just the namespaces of the currently registered providers:

      ```
      az provider list --query "[].namespace"
      ```

   5. Observe the list of currently registered providers. The **Microsoft.CDN** provider isn't currently in the list of providers.

   6. Enter the following command, and then select Enter to get the required flags to register a new provider:

      ```
      az provider register --help
      ```

   7. Enter the following command, and then select Enter to register the **Microsoft.CDN** namespace with your current subscription:

      ```
      az provider register --namespace Microsoft.CDN
      ```

2. Close the Cloud Shell pane in the portal.

48.2.3.3 Task 3: Create a Content Delivery Network profile

1. In the Azure portal's navigation pane, select **Create a resource**.
2. On the **New** blade, find the **Search the Marketplace** text box.
3. In the search box, enter **CDN**, and then select Enter.
4. On the **Everything** search results blade, select the **CDN** result.

5. On the **CDN** blade, select **Create**.
6. On the **CDN profile** blade, perform the following actions:
    1. In the **Name** text box, enter **contentdeliverynetwork**.
    2. Leave the **Subscription** text box set to its default value.
    3. In the **Resource group** section, select **MarketingContent**.
    4. Leave the **Resource group location** drop-down list set to its default value.
    5. In the **Pricing tier** drop-down list, select **Standard Akamai**.
    6. Ensure that the **Create a new CDN endpoint now** check box is cleared.
    7. Select **Create**.

    Note: Wait for Azure to finish creating the CDN profile before you move forward with the lab. You'll receive a notification when the app is created.

48.2.3.4 Task 4: Configure Storage containers

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the contenthost*[yourname]*** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select **+ Container**.
6. In the **New container** pop-up window, perform the following actions:
    1. In the **Name** text box, enter **media**.
    2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**,
    3. Select **Create**.
7. Back in the **Containers** section, select **+ Container** again.
8. In the **New container** pop-up window, perform the following actions:
    1. In the **Name** text box, enter **video**.
    2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**,
    3. Select **Create**.
9. Observe the updated list of containers.

48.2.3.5 Task 5: Create Content Delivery Network endpoints

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the **contentdeliverynetwork** CDN profile that you created earlier in this lab.
4. On the **CDN profile** blade, select **+ Endpoint**.
5. In the **Add an endpoint** pop-up dialog box, perform the following actions:
    1. In the **Name** text box, enter cdnmedia*[yourname]***.
    2. In the **Origin type** drop-down list, select **Storage**.

3. In the **Origin hostname** drop-down list, select the contenthost*[yourname]*.blob.core.windows.net** option for the Storage account that you created earlier in this lab.
4. In the **Origin path** text box, enter **/media**.
5. Leave the **Origin host header** text box set to its default value.
6. Leave the **Protocol** and **Origin port** sections set to their default values.
7. In the **Optimized for** drop-down list, select **General web delivery**.
8. Select **Add**.

6. Back on the **CDN profile** blade, select **+ Endpoint** again.
7. In the **Add an endpoint** pop-up dialog box, perform the following actions:
   1. In the **Name** text box, enter cdnvideo*[yourname]*.
   2. In the **Origin type** drop-down list, select **Storage**.
   3. In the **Origin hostname** drop-down list, select the contenthost*[yourname]*.blob.core.windows.net** option for the Storage account that you created earlier in this lab.
   4. In the **Origin path** text box, enter **/video**.
   5. Leave the **Origin host header** text box set to its default value.
   6. Leave the **Protocol** and **Origin port** sections set to their default values.
   7. In the **Optimized for** drop-down list, select **Video on demand media streaming**.
   8. Select **Add**.
8. Back on the **CDN profile** blade, select **+ Endpoint** again.
9. In the **Add an endpoint** pop-up dialog box, perform the following actions:
   1. In the **Name** text box, enter cdnweb*[yourname]*.
   2. In the **Origin type** drop-down list, select **Web App**.
   3. In the **Origin hostname** drop-down list, select the landingpage*[yourname]*.azurewebsites.net** option for the Web App that you created earlier in this lab.
   4. Leave the **Origin path** text box set to its default value.
   5. Leave the **Origin host header** text box set to its default value.
   6. Leave the **Protocol** and **Origin port** sections set to their default values.
   7. In the **Optimized for** drop-down list, select **General web delivery**.
   8. Select **Add**.

48.2.3.6 Review

In this exercise, you registered the resource provider for Content Delivery Network and then used the provider to create both CDN profile and endpoint resources.

48.2.4 Exercise 3: Upload and configure static web content

48.2.4.1 Task 1: Observe the landing page

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the landingpage*[yourname]*** web app that you created earlier in this lab.
4. On the **App Service** blade, select **Browse**. A new browser window or tab will open and return the current website.

5. Observe the error message displayed on the screen. The website won't work until you configure the specified settings to reference multimedia content.
6. Return to your currently open browser window that's displaying the Azure portal.

48.2.4.2 Task 2: Upload Storage blobs

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the contenthost*[yourname]*** storage account that you created earlier in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Blob service** section.
5. In the **Containers** section, select the **media** container.
6. On the **Container** blade, select **Upload**.
7. In the **Upload blob** pop-up window, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\12\Starter**, select the following files, and then select **Open**:
        - **campus.jpg**
        - **conference.jpg**
        - **poster.jpg**
    3. Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

       Note: Wait for the blob to upload before you continue with this lab.
8. Back on the **Container** blade, select **Properties** in the **Settings** section.
9. Record the value in the **URL** text box. You will use this value later in the lab.
10. Close the **Container** blade.
11. Back on the **Containers** blade, select the **video** container.
12. On the **Container** blade, select **Upload**.
13. In the **Upload blob** pop-up window, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the **File Explorer** window, browse to **Allfiles (F):\Allfiles\Labs\12\Starter**, select the **welcome.mp4** file, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected, and then select **Upload**.

       Note: Wait for the blob to upload before you continue with this lab.
14. Back on the **Container** blade, select **Properties** in the **Settings** section.
15. Record the value in the **URL** text box. You will use this value later in the lab.

48.2.4.3 Task 3: Configure Web App settings

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the landingpage*[yourname]*** web app that you created earlier in this lab.
4. On the **App Service** blade, in the **Settings** category, select the **Configuration** link.
5. In the **Configuration** section, perform the following actions:

1. Select the **Application settings** tab, and then select **New application setting**.
2. In the **Add/Edit application setting** pop-up window, in the **Name** text box, enter **CDNMediaEndpoint**.
3. In the **Value** text box, enter the **URI** value of the **media** container in the contenthost*[yourname]*** storage account that you recorded earlier in this lab.
4. Leave the **deployment slot setting** text box set to its default value, and then select **OK** to close the pop-up window.
5. Return to the **Configuration** section, and then select **New application setting**.
6. In the **Add/Edit application setting** pop-up window, in the **Name** text box, enter **CDNVideoEndpoint**.
7. In the **Value** text box, enter the **URI** value of the **video** container in the contenthost*[yourname]*** storage account that you recorded earlier in this lab.
8. Leave the **deployment slot setting** text box set to its default value, and then select **OK** to close the pop-up window.
9. Return to the **Configuration** section, and then select **Save** on the blade to persist your settings.

   Note: Wait for your application settings to persist before you move forward with the lab.

48.2.4.4 Task 4: Validate the corrected landing page

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the landingpage*[yourname]*** web app that you created earlier in this lab.
4. On the **App Service** blade, select **Restart**. This operation will restart the Web App.

   Note: Wait for the restart operation to complete before you move forward with the lab. You'll receive a notification when the operation is done.
5. Back on the **App Service** blade, select **Browse**. A new browser window or tab will open and return the current website.
6. Observe the updated website rendering multimedia content of various types.
7. Return to your currently open browser window that's displaying the Azure portal.

48.2.4.5 Review

In this exercise, you uploaded multimedia content as blobs to Storage containers and then updated your Web App to point directly to the storage blobs.

48.2.5 Exercise 4: Use Content Delivery Network endpoints

48.2.5.1 Task 1: Retrieve endpoint URIs

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the **contentdeliverynetwork** CDN profile that you created earlier in this lab.
4. On the **CDN profile** blade, select the cdnmedia*[yourname]*** endpoint.
5. On the **Endpoint** blade, copy the value of the **Endpoint hostname** text box. You will use this value later in the lab.
6. Close the **Endpoint** blade.
7. Back on the **CDN profile** blade, select the cdnvideo*[yourname]*** endpoint.

8. On the **Endpoint** blade, copy the value of the **Endpoint hostname** text box. You will use this value later in the lab.
9. Close the **Endpoint** blade.

48.2.5.2 Task 2: Test multimedia content

1. Construct a URL for the **campus.jpg** resource by combining the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/campus.jpg**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnmediastudent.azureedge.net/**, your newly constructed URL would be **https://cdnmediastudent.azureedge.net/campus.jpg**.

2. Construct a URL for the **conference.jpg** resource by combining the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/conference.jpg**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnmediastudent.azureedge.net/**, your newly constructed URL would be **https://cdnmediastudent.azureedge.net/conference.jpg**.

3. Construct a URL for the **poster.jpg** resource by combining the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/poster.jpg**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnmediastudent.azureedge.net/**, your newly constructed URL would be **https://cdnmediastudent.azureedge.net/poster.jpg**.

4. Construct a URL for the **welcome.mp4** resource by combining the **Endpoint hostname** URL from the cdnvideo*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/welcome.mp4**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnvideostudent.azureedge.net/**, your newly constructed URL would be **https://cdnvideostudent.azureedge.net/welcome.mp4**.

5. On the taskbar, right-click the **Microsoft Edge** icon or activate the shortcut menu, and then select **New window**.
6. In the new browser window, go to the URL that you constructed for the **campus.jpg** media resource, and then verify that the resource was successfully found.

   Note: If the content isn't available yet, the CDN endpoint is still initializing. This initialization process can take anywhere from 5 to 15 minutes.

7. Go to the URL that you constructed for the **conference.jpg** media resource, and then verify that the resource was successfully found.
8. Go to the URL that you constructed for the **poster.jpg** media resource, and then verify that the resource was successfully found.
9. Go to the URL that you constructed for the **welcome.mp4** video resource, and then verify that the resource was successfully found.
10. Close the browser window that you created in this task.

48.2.5.3 Task 3: Update the Web App settings

1. In the Azure portal's navigation pane, select **Resource groups**.
2. On the **Resource groups** blade, select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the landingpage*[yourname]*** web app that you created earlier in this lab.
4. On the **App Service** blade, in the **Settings** category, select the **Configuration** link.
5. In the **Configuration** section, perform the following actions:
   1. Select the **Application settings** tab.
   2. Select the existing **CDNMediaEndpoint** application setting.

3. In the **Add/Edit application setting** pop-up dialog box, update the **Value** text box by entering the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab, and then select **OK**.
4. Select the existing **CDNVideoEndpoint** application setting.
5. In the **Add/Edit application setting** pop-up dialog box, update the **Value** text box by entering the **Endpoint hostname** URL from the cdnvideo*[yourname]*** endpoint that you copied earlier in the lab, and then selec **OK**.
6. Select **Save** on the blade to persist your settings.

   Note: Wait for your application settings to persist before you move forward with the lab.

6. Back in the **Configuration** section, select **Overview**.
7. In the **Overview** section, select **Restart**. This operation will restart the Web App.

   Note: Wait for the restart operation to complete before you move forward with the lab. You'll receive a notification when the operation is done.

48.2.5.4 Task 4: Test the web content

1. In the Azure portal's navigation pane, select the **Resource groups** link.
2. On the **Resource groups** blade, find and then select the **MarketingContent** resource group that you created earlier in this lab.
3. On the **MarketingContent** blade, select the **contentdeliverynetwork** CDN profile that you created earlier in this lab.
4. On the **CDN profile** blade, select the cdnweb*[yourname]*** endpoint.
5. On the **Endpoint** blade, copy the value of the **Endpoint hostname** text box.
6. On the taskbar, right-click the **Microsoft Edge** icon or activate the shortcut menu, and then select **New window**.
7. In the new browser window, go to the **Endpoint hostname** URL for the cdnweb*[yourname]*** endpoint.
8. Observe the website and multimedia content that are all served using Content Delivery Network.

48.2.5.5 Review

In this exercise, you updated your Web App to use Content Delivery Network to serve multimedia content and to serve the web application itself.

48.2.6 Exercise 5: Clean up your subscription

48.2.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal's navigation pane, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than sign (>) and underscore character (_).

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:

   1. A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

      Note: Wait for Cloud Shell to finish its initial setup procedures before moving forward with the lab. If you don't notice Cloud Shell configuration options, this is most likely because you're using an existing subscription with this course's labs. The labs are written with the presumption that you're using a new subscription.

48.2.6.2 Task 2: Delete a resource group

1. Enter the following command, and then select Enter to delete the **MarketingContent** resource group:

   ```
   az group delete --name MarketingContent --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

48.2.6.3 Task 3: Close the active application

1. `the currently running Microsoft Edge application.`

48.2.6.4 Review

48.3 In this exercise, you cleaned up your subscription by removing the resource group that was used in this lab.

48.4 lab: az204Title: 'Lab 12: Enhancing a web application by using the Azure Content Delivery Network' az204Module: 'Module 12: Integrate caching and content delivery within solutions'

49 Lab 12: Enhancing a web application by using the Azure Content Delivery Network

50 Student lab manual

50.1 Lab scenario

Your marketing organization has been tasked with building a website landing page to host content about an upcoming edX course. While designing the website, your team decided that multimedia videos and image content would be the ideal way to convey your marketing message. The website is already completed and available using a Docker container, and your team also decided that it would like to use a content delivery network (CDN) to improve the performance of the images, the videos, and the website itself. You have been tasked with using Microsoft Azure Content Delivery Network to improve the performance of both standard and streamed content on the website.

50.2 Objectives

After you complete this lab, you will be able to:

- Register a Microsoft.CDN resource provider.
- Create Content Delivery Network resources.
- Create and configure Content Delivery Network endpoints that are bound to various Azure services.

50.3 Lab setup

- Estimated time: **45 minutes**

50.4 Instructions

50.4.1 Before you start

50.4.1.1 Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine (VM) by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

50.4.1.2 Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icon for the application that you'll use in this lab:

- Microsoft Edge

50.4.2 Exercise 1: Create Azure resources

50.4.2.1 Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

50.4.2.2 Task 2: Create a Storage account

1. Create a new storage account with the following details:
   - New resource group: **MarketingContent**
   - Name: contenthost*[yourname]***
   - Location: **East US**
   - Performance: **Standard**
   - Account kind: **StorageV2 (general purpose v2)**
   - Replication: **Read-access geo-redundant storage (RA-GRS)**

     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You'll receive a notification when the account is created.

50.4.2.3 Task 3: Create a web app by using Azure App Service

1. Create a new web app with the following details:
   - Existing resource group: **MarketingContent**
   - Name: landingpage*[yourname]***
   - Publish: **Docker Container**
   - Operating system: **Linux**
   - Region: **East US**
   - New App Service plan: **MarketingPlan**
   - SKU and size: **Premium V2 P1v2**
   - Docker options: **Single Container**
   - Image source: **Docker Hub**
   - Access type: **Public**
   - Image and tag: **microsoftlearning/edx-html-landing-page:latest**

     Note: Wait for Azure to finish creating the web app before you move forward with the lab. You'll receive a notification when the app is created.
2. Access the landingpage*[yourname]*** web app that you created earlier in this lab.
3. In the **Settings** section, go to the **Properties** section, and then record the value of the **URL** text box. You'll use this value later in the lab.

50.4.2.4 Review

In this exercise, you created an Azure Storage account and an Azure Web App that you'll use later in this lab.

50.4.3 Exercise 2: Configure Content Delivery Network and endpoints

50.4.3.1 Task 1: Open Azure Cloud Shell

1. Open a new Cloud Shell instance in the Azure portal.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.
3. At the **Cloud Shell** command prompt in the portal, use the **az** command with the **--version** flag to get the version of the Azure Command-Line Interface (Azure CLI) tool.

50.4.3.2 Task 2: Register the Microsoft.CDN provider

1. Use the **az** command with the **--help** flag to find a list of subgroups and commands at the root level of the Azure CLI.
2. Use the **az provider** command with the **--help** flag to get a list of commands available for resource providers.
3. Use the **az provider list** command to list all currently registered providers.
4. Use the **az provider list** command again with the **--query "[].namespace"** flag to list just the namespaces of the currently registered providers.
5. Observe the list of currently registered providers. The **Microsoft.CDN** provider isn't currently in the list of providers.
6. Use the **az provider register** command with the **--help** flag to get the required flags to register a new provider.
7. Use the **az provider register** command to register a namespace with your current subscription by using the following setting:
   - Namespace: **Microsoft.CDN**
8. Close the Cloud Shell pane.

50.4.3.3 Task 3: Create a Content Delivery Network profile

1. Create a new Content Delivery Network profile with the following details:
   - Name: **contentdeliverynetwork**
   - Existing resource group: **MarketingContent**
   - Resource group location: **East US**
   - Pricing tier: **Standard Akamai**
   - Create a new CDN endpoint now: **No**

     Note: Wait for Azure to finish creating the CDN profile before you move forward with the lab. You'll receive a notification when the app is created.

50.4.3.4 Task 4: Configure Storage containers

1. Access the contenthost*[yourname]*** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then create a new container with the following settings:
   - Name: **media**
   - Public access level: **Blob (anonymous read access for blobs only)**
3. Create a new container with the following settings:
   - Name: **video**
   - Public access level: **Blob (anonymous read access for blobs only)**
4. Observe the updated list of containers.

50.4.3.5 Task 5: Create Content Delivery Network endpoints

1. Access the **contentdeliverynetwork** CDN profile that you created earlier in this lab, and then select + **Endpoint**.
2. Add a new endpoint with the following properties:
   - Name: cdnmedia*[yourname]***
   - Origin type: **Storage**
   - Origin hostname: contenthost*[yourname]*.blob.core.windows.net** (the Storage account that you created earlier in this lab)
   - Origin path: **/media**
   - Optimized for: **General web delivery**

     Note: Wait for Azure to finish creating the CDN endpoint before you move forward with the lab. You'll receive a notification when the account is created.
3. Add a new endpoint with the following properties:
   - Name: cdnvideo*[yourname]***
   - Origin type: **Storage**
   - Origin hostname: contenthost*[yourname]*.blob.core.windows.net** (the Storage account that you created earlier in this lab)
   - Origin path: **/video**
   - Optimized for: **Video on demand media streaming**

     Note: Wait for Azure to finish creating the CDN endpoint before you move forward with the lab. You'll receive a notification when the account is created.
4. Add a new endpoint with the following properties:
   - Name: cdnweb*[yourname]***
   - Origin type: **Web App**
   - Origin hostname: landingpage*[yourname]*.azurewebsites.net** (the Web App that you created earlier in this lab)
   - Optimized for: **General web delivery**

     Note: Wait for Azure to finish creating the CDN endpoint before you move forward with the lab. You'll receive a notification when the account is created.

50.4.3.6 Review

In this exercise, you registered the resource provider for Content Delivery Network and then used the provider to create both CDN profile and endpoint resources.

50.4.4 Exercise 3: Upload and configure static web content

50.4.4.1 Task 1: Observe the landing page

1. Access the landingpage*[yourname]*** web app that you created earlier in this lab.
2. Go to the URL for the landingpage*[yourname]*** web app.
3. Observe the error message displayed on the screen. The website won't work until you configure the specified settings to reference multimedia content.
4. Return to the Azure portal.

50.4.4.2 Task 2: Upload Storage blobs

1. Access the contenthost*[yourname]*** storage account that you created earlier in this lab.
2. Select the **Containers** link in the **Blob service** section, and then select the **media** container.
3. Upload the following files from the **Allfiles (F): \Allfiles\Labs\12\Starter** folder on your lab VM:
   - **campus.jpg**
   - **conference.jpg**
   - **poster.jpg**

   Note: We recommend that you enable the **Overwrite if files already exist** option.
4. Record the value in the **URL** text box. You will use this value later in the lab.
5. Back in the **Containers** section, select the **video** container.
6. Upload the **welcome.mp4** file from the **Allfiles (F): \Allfiles\Labs\12\Starter** folder on your lab VM.

   Note: We recommend that you enable the **Overwrite if files already exist** option.
7. Record the value in the **URL** text box. You will use this value later in the lab.

50.4.4.3 Task 3: Configure Web App settings

1. Access the landingpage*[yourname]*** web app that you created earlier in this lab.
2. In the **Settings** section, go to the **Configuration** section.
3. Find and access the **Application Settings** tab in the **Configuration** section.
4. Create a new application setting by using the following details:
   - Name: **CDNMediaEndpoint**
   - Value: The **URI** value of the **media** container in the contenthost*[yourname]*** storage account that you recorded earlier in this lab.
   - Deployment slot setting: **Not selected**
5. Create a new application setting by using the following details:
   - Name: **CDNVideoEndpoint**
   - Value: The **URI** value of the **video** container in the contenthost*[yourname]*** storage account that you recorded earlier in this lab.
   - Deployment slot setting: **Not selected**
6. Save your changes to the application settings.

50.4.4.4 Task 4: Validate the corrected landing page

1. Access the landingpage*[yourname]*** web app that you created earlier in this lab.
2. **Restart** the currently running Web App.

   Note: Wait for the restart operation to complete before you move forward with the lab. You'll receive a notification when the operation is done.
3. Go to the URL for the landingpage*[yourname]*** web app.
4. Observe the updated website rendering multimedia content of various types.
5. Return to the Azure portal.

50.4.4.5 Review

In this exercise, you uploaded multimedia content as blobs to Storage containers and then updated your Web App to point directly to the storage blobs.

50.4.5 Exercise 4: Use Content Delivery Network endpoints

50.4.5.1 Task 1: Retrieve endpoint Uniform Resource Identifiers (URIs)

1. Access the **contentdeliverynetwork** CDN profile that you created earlier in this lab.
2. Select the cdnmedia*[yourname]*** endpoint.
3. Copy the value of the **Endpoint hostname** text box. You will use this value later in the lab.
4. Select the cdnvideo*[yourname]*** endpoint.
5. Copy the value of the **Endpoint hostname** text box. You will use this value later in the lab.

50.4.5.2 Task 2: Test multimedia content by using the CDN

1. Construct a URL for the **campus.jpg** resource by combining the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/campus.jpg**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnmediastudent.azureedge.net/**, your newly constructed URL would be **https://cdnmediastudent.azureedge.net/campus.jpg**.

2. Construct a URL for the **conference.jpg** resource by combining the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/conference.jpg**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnmediastudent.azureedge.net/**, your newly constructed URL would be **https://cdnmediastudent.azureedge.net/conference.jpg**.

3. Construct a URL for the **poster.jpg** resource by combining the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/poster.jpg**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnmediastudent.azureedge.net/**, your newly constructed URL would be **https://cdnmediastudent.azureedge.net/poster.jpg**.

4. Construct a URL for the **welcome.mp4** resource by combining the **Endpoint hostname** URL from the cdnvideo*[yourname]*** endpoint that you copied earlier in the lab with a relative path of **/welcome.mp4**.

   Note: For example, if your **Endpoint hostname** URL is **https://cdnvideostudent.azureedge.net/**, your newly constructed URL would be **https://cdnvideostudent.azureedge.net/welcome.mp4**.

5. Open a new **Microsoft Edge** browser window.
6. In the new browser window, go to the URL that you constructed for the **campus.jpg** media resource, and then verify that the resource was successfully found.

   Note: If the content isn't available yet, the CDN endpoint is still initializing. This initialization process can take anywhere from 5 to 15 minutes.

7. Go to the URL that you constructed for the **conference.jpg** media resource, and then verify that the resource was successfully found.
8. Go to the URL that you constructed for the **poster.jpg** media resource, and then verify that the resource was successfully found.
9. Go to the URL that you constructed for the **welcome.mp4** video resource, and then verify that the resource was successfully found.
10. Close the browser window that you created in this task.

50.4.5.3 Task 3: Update the Web App settings

1. Access the landingpage*[yourname]*** web app that you created earlier in this lab.
2. In the **Settings** section, go to the **Configuration** section.
3. Find and access the **Application Settings** tab in the **Configuration** section.
4. Update the **CDNMediaEndpoint** application setting by changing its value to the **Endpoint hostname** URL from the cdnmedia*[yourname]*** endpoint that you copied earlier in the lab.
5. Update the **CDNVideoEndpoint** application setting by changing its value to the **Endpoint hostname** URL from the cdnvideo*[yourname]*** endpoint that you copied earlier in the lab.

6. Save your changes to the application settings.
7. Restart the currently running Web App.

   Note: Wait for the restart operation to complete before you move forward with the lab. You'll receive a notification when the operation is done.

50.4.5.4 Task 4: Test the web content

1. Access the **contentdeliverynetwork** CDN profile that you created earlier in this lab.
2. Select the cdnweb*[yourname]*** endpoint.
3. Copy the value of the **Endpoint hostname** text box. You will use this value later in the lab.
4. Open a new **Microsoft Edge** browser window.
5. In the new browser window, go to the **Endpoint hostname** URL for the cdnweb*[yourname]*** endpoint.
6. Observe the website and multimedia content that are all served by using Content Delivery Network.

50.4.5.5 Review

In this exercise, you updated your Web App to use Content Delivery Network to serve multimedia content and to serve the web application itself.

50.4.6 Exercise 5: Clean up your subscription

50.4.6.1 Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for **Bash** by using the default settings.

50.4.6.2 Task 2: Delete a resource group

1. At the command prompt, enter the following command, and then select Enter to delete the **Marketing-Content** resource group:

   ```
   az group delete --name MarketingContent --no-wait --yes
   ```

2. Close the Cloud Shell pane in the portal.

50.4.6.3 Task 3: Close the active application

1. `the currently running Microsoft Edge application.`

50.4.6.4 Review

In this exercise, you cleaned up your subscription by removing the resource group used in this lab.