Contents

1 AZ-203: Developing Solutions for Microsoft Azure

Sustained Engineering work will no longer continue for AZ-203T0X course labs since a replacement course, AZ-204T00, with associated labs, is currently available in market. Any questions can be directed by MCTs to the Courseware Support Forum for escalation purposes

- **Download Latest Student Handbook and AllFiles Content**
- **Want to know what is installed on the lab VM?**

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

2 Content Directory

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

2.1 Labs

{% assign labs = site.pages | where_exp:"page", "page.url contains '/Instructions/Labs'" %} | Course | Lab | | --- | --- | {% for activity in labs %}| {{ activity.lab.module }} | [{{ activity.lab.title }}{% if activity.lab.type %} - {{ activity.lab.type }}{% endif %}](/home/ll/Azure_clone/Azure_new/AZ-203-DevelopingSolutionsforMicrosoftAzure/{{ site.github.url }}{{ activity.url }}) | {% endfor %}

3 Lab Virtual Machine Installed Software List

Software	Link
Windows 10	https://www.microsoft.com/software-download/windows10
Office 2016	https://products.office.com/microsoft-office-2016
Visual Studio Code	https://code.visualstudio.com/
Visual Studio Code Azure Account 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
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/
Git for Windows	https://git-scm.com/download/win
.NET Core 2.2 SDK	https://dotnet.microsoft.com/download

3.1 lab: title: 'Lab: Deploying compute workloads by using images and containers' type: 'Answer Key' module: 'AZ-203T01-A: Develop Azure infrastructure as a service (IaaS) compute solutions'

4 Lab: Deploying compute workloads by using images and containers

5 Student lab answer key

5.1 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 up.

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

5.2 Instructions

5.2.1 Before you start

5.2.1.1 Sign in to the lab virtual machine

Sign in to your **Windows 10** virtual machine using the following credentials:

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

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

5.2.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:

- Microsoft Edge
- File Explorer

5.2.2 Exercise 1: Create a virtual machine (VM) by using the Azure portal

5.2.2.1 Task 1: Open the Azure portal

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

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

5.2.2.2 Task 2: Create a resource group

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.

   Note: If you cannot find the **Create a resource** link, the "Create a resource" icon is a plus-sign character located on the left side of the portal.

2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.

3. In the search text box, enter the text **Resource Group** and press Enter.
4. In the **Everything** search results blade, select the **Resource group** result.
5. In the **Resource group** blade, select **Create**.
6. In the additional **Resource group** blade, observe the tabs at the top of the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new **resource group**. At any time, you can select **Review + create** 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** text box, enter the value **ContainerCompute**.
   3. In the **Region** drop-down list, select the **(US) East US** location.
   4. Select **Review + Create**.
8. In 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.
10. Wait for the creation task to complete before moving forward with this lab.

5.2.2.3 Task 3: Create a Linux virtual machine resource

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Ubuntu Server 18** and press Enter.
4. In the **Everything** search results blade, select the **Ubuntu Server 18.04 LTS** result.
5. In the **Ubuntu Server 18.04 LTS** blade, select **Create**.
6. In the **Create a virtual machine** blade, observe the tabs at the top of the blade, such as **Basics** and **Disks**.

   Note: Each tab represents a step in the workflow to create a new **virtual machine**. At any time, you can select **Review + create** to skip the remaining tabs.
7. In 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 the existing **ContainerCompute** option.
   3. In the **Virtual machine name** text box, enter **simplevm**.
   4. In the **Region** drop-down list, select the **(US) East US** location.
   5. In the **Availability options** drop-down list, ensure **No infrastructure redundancy required** is selected.
   6. In the **Image** text box, make sure that the **Ubuntu Server 18.04 LTS** option is selected.
   7. In the **Size** text box, select the **Change size** link.
8. In the **Select a VM size** blade, perform the following actions:
   1. Select the **B1s** option from the list of sizes.
   2. Press **Select**.
9. Go back to the **Basics** tab and perform the following actions:
   1. In the **Authentication type** section, select **Password**.
   2. In the **Username** text box, enter **Student**.
   3. In the **Password** and **Confirm password** fields, enter **StudentPa55w.rd**.
   4. In the **Login with Azure Active Directory (Preview)** section, select **Off**.

5. In the **Public inbound ports** section, select **Allow selected ports**.
6. In the **Select inbound ports** drop-down list, select only **SSH (22)**.
7. Select **Next : Disks >**.

10. In the **Disks** tab, perform the following actions:
    1. In the **OS disk type** section, select **Standard SSD**.
    2. Select **Review + create**.
11. In the **Review + Create** tab, review the options that you selected during the previous steps.
12. Select **Create** to create the VM by using your specified configuration.
13. Wait for the creation task to complete before moving forward with this lab.

5.2.2.4 Task 4: Validate the virtual machine

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **ContainerCompute** resource group that you created earlier in this lab.
3. In the **ContainerCompute** blade, select the **simplevm** VM that you created earlier in this lab.
4. In the **Virtual Machine** blade, select **Connect**.
5. In the **Connect to virtual machine** pop-up that appears, perform the following actions:
    1. In the **IP address** text box, select **Public IP address**.
    2. In the **Port number** text box, enter **22**.
    3. **Copy** the text in the **Login using VM local account** text box.

       Note: The command that you copied will connect to the VM by using SSH from a remote computer. You will use this command later in the lab.
6. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than symbol and underscore character.
7. If this is your first time opening the **Cloud Shell** by using your subscription, a **Welcome to Azure Cloud Shell Wizard** will appear that allows you to configure **Cloud Shell** for first-time usage. Perform the following actions in the wizard:
    1. A dialog box will appear that prompts you to create a new Storage Account to begin using the shell. Accept the default settings and select **Create storage**.
    2. Wait for the **Cloud Shell** to finish its first-time setup procedures before moving forward with the lab.

    Note: If you do not see the configuration options for the **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. The labs are written from the presumption that you are using a new subscription.
8. In the **Cloud Shell** command prompt at the bottom of the portal, **paste** the command you copied earlier in this lab and press Enter to connect to your new VM by using SSH.

   Note: This command will be dependent on your username and IP address. For example, if the username is **Student** and the IP address is **40.125.245.5**, the command would be **ssh Student@40.125.245.5**.
9. 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** in the prompt and then press Enter to continue connecting to the VM.
10. The SSH tool will then ask you for a password. Enter **StudentPa55w.rd** and then press Enter to authenticate with the VM.

    Note: Characters do not show when typing password. Please be advised.

11. After you are connected to the VM by using SSH, you will see a prompt for the Bash shell in the VM. In the prompt, type in the following command and press Enter to view the computer name of the Linux VM:

    ```
    uname -mnipo
    ```

12. In the prompt, type in the following command and press Enter to view information about the distribution and operating system of the Linux VM.

    ```
    uname -srv
    ```

13. Close the **Cloud Shell** pane at the bottom of the portal.

5.2.2.5 Review

In this exercise, you created a new VM manually by using the Azure portal interface and connected to the VM by using the Cloud Shell and SSH.

5.2.3 Exercise 2: Create a virtual machine by using Azure CLI

5.2.3.1 Task 1: Open Cloud Shell

1. In the top navigation bar in the Azure Portal, select the **Cloud Shell** icon to open a new shell instance.
2. Wait for the **Cloud Shell** to finish connecting to an instance before moving forward with the lab.
3. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press Enter to view the version of the Azure CLI tool:

    ```
    az --version
    ```

5.2.3.2 Task 2: Use the Azure CLI commands

1. Type the following command and press Enter to view a list of subgroups and commands at the root level of the CLI:

    ```
    az --help
    ```

2. Type the following command and press Enter to view a list of subgroups and commands for **virtual machines**:

    ```
    az vm --help
    ```

3. Type the following command and press Enter to view a list of arguments and examples for the **Create Virtual Machine** command:

    ```
    az vm create --help
    ```

4. Type the following command and press 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**: If this command fails with an error indicating `Resource group 'ContainerCompute'` could not be found, it likely means you either made an error naming your resource group at the start of this lab, or you have more than one subscription associated with your login name. If you named the resource group wrong, substitute the correct name in the command. If you have more than one subscription, you can use the `az account set --subscription` command to select the proper subscription.

5. Wait for the VM creation process to complete. After the process completes, the command will return a JSON file containing details about the machine.

6. Type the following command and press Enter to view a more detailed JSON file that contains various metadata about the newly created VM:

   ```
   az vm show --resource-group ContainerCompute --name quickvm
   ```

7. Type the following command and press Enter to list all the IP addresses associated with the VM:

   ```
   az vm list-ip-addresses --resource-group ContainerCompute --name quickvm
   ```

8. Type the following command and press 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
   ```

9. Type the following command and press 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 '[].{
   ```

10. Type the following command and press Enter to print the value of the Bash shell variable *ipAddress*:

    ```
    echo $ipAddress
    ```

11. Type the following command and press Enter to connect to the VM that you created earlier in this lab by using the SSH tool and the IP address stored in the Bash shell variable *ipAddress*:

    ```
    ssh student@$ipAddress
    ```

12. 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 press Enter to continue connecting to the VM.
13. The **SSH** tool will then ask you for a password. Enter **StudentPa55w.rd** and then press Enter to authenticate with the VM.
14. After you connect to the VM using SSH, type the following command and press Enter to view metadata describing the Linux VM:

    ```
    uname -a
    ```

15. Close the **Cloud Shell** pane at the bottom of the portal.

5.2.3.3 Review

In this exercise, you used the Azure Cloud Shell to create a VM as part of an automated script.

5.2.4 Exercise 3: Create a Docker container image and deploy it to Azure Container Registry

5.2.4.1 Task 1: Open Cloud Shell and editor

1. In the top navigation bar in the Azure Portal, select the **Cloud Shell** icon to open a new shell instance.
2. Wait for the **Cloud Shell** to finish connecting to an instance before moving on with the lab.
3. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press **Enter** to move from the root directory to the ~/**clouddrive** directory:

   ```
   cd ~/clouddrive
   ```

4. Type the following command and press Enter to create a new directory named **ipcheck** within the ~/**clouddrive** directory:

   ```
   mkdir ipcheck
   ```

5. Type the following command and press Enter to change the active directory from ~/**clouddrive** to ~/**clouddrive/ipcheck**:

   ```
   cd ~/clouddrive/ipcheck
   ```

6. Type the following command and press Enter to create a new .NET Core console application in the current directory:

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

7. Type the following command and press Enter to create a new file in the ~/**clouddrive/ipcheck** directory named **Dockerfile**:

```
touch Dockerfile
```

8. Type the following command and press Enter to open the embedded graphical editor in the context of the current directory:

```
code .
```

5.2.4.2 Task 2: Create and test a .NET Core application

1. Within the graphical editor, locate the **FILES** pane and double-select the **Program.cs** file to open that file 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)
    {
        if (System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())
        {
            System.Console.WriteLine("Current IP Addresses:");
            string hostname = System.Net.Dns.GetHostName();
            System.Net.IPHostEntry host = System.Net.Dns.GetHostEntry(hostname);
            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 either the menu in the the graphical editor, or the **Ctrl+S** keyboard shortcut. Do not close the graphical editor.
5. Back in the command prompt, type the following command and press Enter to execute the application:

```
dotnet run
```

6. Observe the results of the execution. There should be at least one IP address listed for the Cloud Shell instance.
7. Within the graphical editor, locate the **FILES** pane on the left side of the editor and double-select the **Dockerfile** file to open that file in the editor.
8. Copy and paste the following code into the **Dockerfile** file:

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2-alpine AS build
WORKDIR /app

COPY *.csproj ./
RUN dotnet restore

COPY . ./
RUN dotnet publish --configuration Release --output out

FROM mcr.microsoft.com/dotnet/core/runtime:2.2-alpine
WORKDIR /app

COPY --from=build /app/out .

ENTRYPOINT ["dotnet", "ipcheck.dll"]
```

9. **Save** the **Dockerfile** file by using either the menu in the graphical editor or the **Ctrl+S** keyboard shortcut.
10. Close the **Cloud Shell** pane at the bottom of the portal.

5.2.4.3 Task 3: Create an Azure Container Registry resource

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Container Registry** and press **Enter**.
4. In the **Everything** search results blade, select the **Container Registry** result.
5. In the **Container Registry** blade, select **Create**.
6. In 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 are 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 **Admin user** section, select **Disable**.
    6. In the **SKU** drop-down list, select **Basic**.
    7. Select **Create**.
7. Wait for the creation task to complete before moving forward with this lab.

5.2.4.4 Task 4: Open Cloud Shell and store Azure Container Registry metadata

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.
2. Wait for the **Cloud Shell** to finish connecting to an instance before moving forward with the lab.
3. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press Enter to view a list of all container registries in your subscription:

    ```
    az acr list
    ```
4. Type the following command and press Enter:

    ```
    az acr list --query "max_by([], &creationDate).name" --output tsv
    ```
5. Type the following command and press Enter:

    ```
    acrName=$(az acr list --query "max_by([], &creationDate).name" --output tsv)
    ```
6. Type the following command and press Enter:

    ```
    echo $acrName
    ```

5.2.4.5 Task 5: Deploy a Docker container image to the Azure Container Registry

1. Type the following command and press Enter to change the active directory from ~/ to ~/**clouddrive/ipcheck**:

    ```
    cd ~/clouddrive/ipcheck
    ```
2. Type the following command and press Enter to view the contents of the current directory:

    ```
    dir
    ```
3. Type the following command and press Enter to upload the source code to your **Container Registry** and build the container image as an **Azure Container Registry Task**:

```
az acr build --registry $acrName --image ipcheck:latest .
```

4. Wait for the build task to complete before moving forward with this lab.
5. Close the **Cloud Shell** pane at the bottom of the portal.

5.2.4.6 Task 6: Validate your container image in the Azure Container Registry

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **ContainerCompute** resource group that you created earlier in this lab.
3. In the **ContainerCompute** blade, select the container registry that you created earlier in this lab.
4. In the **Container Registry** blade, locate the **Services** section and select the **Repositories** link.
5. In the **Repositories** section, select the **ipcheck** container image repository.
6. In the **Repository** blade, select the **latest** tag.
7. Observe the metadata for the version of your container image with the **latest** tag.

   Note: You can also select the **Run ID** hyperlink to view metadata about the build task.

5.2.4.7 Review

In this exercise, you created a .NET Core 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 Azure Container Registry.

5.2.5 Exercise 4: Deploy an Azure container instance

5.2.5.1 Task 1: Enable Admin User in Azure Container Registry

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **ContainerCompute** resource group that you created earlier in this lab.
3. In the **ContainerCompute** blade, select the container registry that you created earlier in this lab.
4. In the **Container Registry** blade, select **Update** from the top of the blade.
5. In the **Update container registry** blade, perform the following actions:
   1. In the **Admin user** section, select **Enable**.
   2. Select **Save**.
6. Close the **Update container registry** blade.

5.2.5.2 Task 2: Deploy a container image automatically to an Azure Container instance

1. In the **Container Registry** blade, locate the **Services** section and select the **Repositories** link.
2. In the **Repositories** section, select the **ipcheck** container image repository.
3. In the **Repository** blade, select the ellipsis menu located immediately to the right of the **latest** tag entry.
4. In the pop-up menu that appears, select the **Run instance** link.
5. In the **Create container instance** blade that appears, perform the following actions:
   1. In the **Container name** text boxtext 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**.

6. Wait for the creation task to complete before moving forward with this lab.

5.2.5.3 Task 3: Deploy a container image manually to an Azure Container instance

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **ContainerCompute** resource group that you created earlier in this lab.
3. In the **ContainerCompute** blade, select the container registry that you created earlier in this lab.
4. In the **Container Registry** blade, locate the **Settings** section and select the **Access keys** link.
5. In the **Access Keys** section, record the values for the following fields:
    1. **Login server**
    2. **Username**
    3. **Password**

        Note: You will use these values later in this lab when you create another **container instance**.
6. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
7. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
8. In the search text box, enter **Container instance** and press **Enter**.
9. In the **Everything** search results blade, select the **Container Instances** result.
10. In the **Container Instances** blade, select **Create**.
11. In the **Create Container Instances** blade, observe the tabs on the left of the blade, such as **Basics** and **Advanced**.

    Note: Each tab represents a step in the workflow to create a new **container instance**.
12. In 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 type** section, select **Private**.
    6. In the **Image name** text box, enter the **Login server** value that you recorded earlier and then add the suffix **/ipcheck:latest**.

        Note: For example, if your **Login server** value is **azadmin.azurecr.io**, then your container image name would be **azadmin.azurecr.io/ipcheck:latest**
    7. In the **Image registry login server** text box, enter the **Login server** value that you recorded earlier in this lab.
    8. In the **Image registry user name** text box, enter the **Username** value that you recorded earlier in this lab.
    9. In the **Image registry password** text box, enter the **Password** value that you recorded earlier in this lab.
    10. In the **OS type** section, select **Linux**.

11. In the **Size** section, select the **Change size** link.
12. In the **Change container size** blade, perform the following actions:
    1. In the **Number of CPU cores** textbox, enter **1**.
    2. In the **Memory (GiB)** textbox, enter **1.5**.
    3. Leave the default value for the **GPU type** drop-down list.
    4. Select **Ok**.
13. Select **Next: Networking**

13. In the **Networking** tab, perform the following actions:
    1. In the **Include Public IP address** section, select **Yes**.
    2. Ensure in the **Ports** section, the port **80** is there, with the port protocol **TCP**.
    3. Leave the **DNS name label** text box empty.
    4. Select **Next: Advanced**.
14. In the **Advanced** tab, perform the following actions:
    1. In the **Restart policy** drop-down list, select **On failure**.
    2. Leave the **Environment variable** text box empty.
    3. Leave the **Command override** text box empty.
    4. Select **Review + create**.
15. In the **Review + create** tab, review the selected options.
16. Select **Create** to create the container instance by using your specified configuration.
17. Wait for the creation task to complete before moving forward with this lab.

5.2.5.4 Task 4: Validate that the container instance ran successfully

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **ContainerCompute** resource group that you created earlier in this lab.
3. In the **ContainerCompute** blade, select the **manualcompute** container instance that you created earlier in this lab.
4. In the **Container Instance** blade, locate the **Settings** section and select the **Containers** link.
5. In the **Containers** section, observe the list of **Events**.
6. Select the **Logs** tab and observe the text logs from the container instance.

   Note: You can also optionally view the **Events** and **Logs** from the **managedcompute** container instance.

   Note: After the application finished executing, the container was terminated because it had completed its work. For the manually created container instance, you indicated that you indicated that a successful exit was acceptable, so the container shows a single execution run. The automatically created instance did not offer you this option, and assumes the container should always be running, so you will see repeated restarts of the container.

5.2.5.5 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 execute task-based applications as part of a container run.

5.2.6 Exercise 5: Clean up subscription

5.2.6.1 Task 1: Open Cloud Shell and list resource groups

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.
2. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

3. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

5.2.6.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **ContainerCompute** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

5.2.6.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.

5.2.6.4 Review

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

5.4 lab: title: 'Lab: Deploying compute workloads by using images and containers' module: 'AZ-203T01-A: Develop Azure infrastructure as a service (IaaS) compute solutions'

6 Lab: Deploying compute workloads by using images and containers

7 Student lab manual

7.1 Lab scenario

Your organization is seeking a way to automatically create virtual machines (VMs) to run tasks and immediately terminate. You are tasked with evaluating multiple compute services in Microsoft Azure and determining which service can help you automatically create virtualized machines and install custom software on those machines. As a proof of concept, you have decided to try to create VMs from VHD images and container images so that you can compare the two solutions. To keep your proof of concept simple, you will create a special "IP check" application, written in .NET Core, that you will automatically deploy to your machines. Your proof of concept will evaluate the Azure Container Instances and Azure Virtual Machines services.

7.2 Objectives

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

- Create a VM manually or by using tools in the Azure portal.
- Deploy a Docker container image to Azure Container Registry.
- Deploy a container from a container image in Azure Container Registry by using Azure Container Instances.
- Deploy a VM using an Azure Resource Manager template and a container image in Azure Container Registry.

7.3 Lab Setup

- **Estimated Time**: 75 minutes

7.4 Instructions

7.4.1 Before you start

7.4.1.1 Sign in to the lab virtual machine

Ensure that you are signed in to your **Windows 10** virtual machine using the following credentials:

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

7.4.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:

- Microsoft Edge
- File Explorer

7.4.2 Exercise 1: Create a virtual machine using the Azure portal

7.4.2.1 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 will appear and a tour of the portal will start. Select the **Get Started** button to skip the tour.

7.4.2.2 Task 2: Create a resource group

1. Create a new **resource group** with the following details:
    1. **Name**: ContainerCompute
    2. **Location**: (US) East US
2. Wait for the creation task to complete before moving forward with this lab.

7.4.2.3 Task 3: Create a Linux virtual machine resource

1. Create a new **Virtual Machine** with the following details:
    - **Operating system**: Ubuntu Server 18.04 LTS
    - **Name**: simplevm
    - **Disk type**: SSD
    - **Username**: Student
    - **Password**: StudentPa55w.rd
    - **Resource group**: ContainerCompute
    - **Location**: (US) East US
    - **Size**: Standard B1s
    - **Public inbound ports**: SSH (22)
2. Wait for the creation task to complete before moving on with this lab.

7.4.2.4 Task 4: Validate the virtual machine

1. Access the **simplevm** VM that you created earlier in this lab.
2. Open the **Connect to virtual machine** pop-up.
3. Copy the **SSH command** to connect to the VM by using port **22** and the **Public IP address**. You will use this command later in this lab.
4. Open a new **Cloud Shell** instance in the Azure Portal.
5. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
6. Within the **Cloud Shell** command prompt, use the **SSH command** that you copied earlier in this lab to connect to the VM by using SSH.
7. During the connection process, you will receive a warning that the authenticity of the host cannot be verified. Continue connecting to the host. Finally, use the password **StudentPa55w.rd** when prompted for credentials
8. When you are connected to the VM, use the following commands to view information about the machine:

   ```
   uname -mnipo
   ```

   ```
   uname -srv
   ```

9. Close the **Cloud Shell** pane.

7.4.2.5 Review

In this exercise, you created a new VM manually by using the Azure portal interface and connected to the VM by using the Cloud Shell and secure shell (SSH).

7.4.3 Exercise 2: Create a virtual machine by using Azure CLI

7.4.3.1 Task 1: Open Cloud Shell

1. Open a new **Cloud Shell** instance in the Azure Portal.

7.4.3.2 Task 2: Use the Azure CLI commands

1. Use the **az** command with the **--help** flag to view a list of subgroups and commands at the root level of the CLI.
2. Use the **az vm** command with the **--help** flag to view a list of subgroups and commands for **Virtual Machines**:
3. Use the **az vm create** command with the **--help** flag to view a list of arguments and examples for the **Create Virtual Machine** command:
4. Use the **az vm create** command to create a new **Virtual Machine** with the following settings:
   - **Resource group**: ContainerCompute
   - **Name**: quickvm
   - **Image**: Debian
   - **Username**: student
   - **Password**: StudentPa55w.rd
5. Wait for the VM creation process to complete. After the process completes, the command will return a JSON file with details about the machine.
6. Use the **az vm show** command to view a more detailed JSON file that contains various metadata about the newly created VM.
7. 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
   ```

8. 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
```

9. 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 '[].{
```

10. Use the following script to print the value of the Bash shell variable *ipAddress*:

```
echo $ipAddress
```

11. Use the following script to connect to the VM that you created earlier in this lab by using the SSH tool and the IP address stored in the Bash shell variable *ipAddress*:

```
ssh student@$ipAddress
```

12. During the connection process, you will receive a warning that the authenticity of the host cannot be verified. Continue connecting to the host. Finally, use the password **StudentPa55w.rd** when prompted for credentials

13. After you are connected to the VM, use the following commands to observe information about the machine to ensure that you are connected to the correct VM:

```
uname -mnipo
```

```
uname -srv
```

14. Close the **Cloud Shell** pane.

7.4.3.3 Review

In this exercise, you used the Azure Cloud Shell to create a virtual machine as part of an automated script.

7.4.4 Exercise 3: Create a Docker container image and deploy it to Azure Container Registry

7.4.4.1 Task 1: Open the Cloud Shell and editor

1. Open a new **Cloud Shell** instance in the Azure Portal.
2. Within the **Cloud Shell** command prompt, change the active directory to ~/**clouddrive**.

   Note: The command to change directory in Bash is **cd <path>**.

3. In the Cloud Shell command prompt, create a new directory named **ipcheck** within 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** command to create a new .NET Core console application named **ipcheck** in the current directory.

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

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 filename **Dockerfile** is case sensitive.

7. Open the embedded graphical editor in the context of the current directory.

   Note: You can open the editor by either using the **code .** command or selecting the editor button.

7.4.4.2 Task 2: Create and test a .NET Core 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)
    {
```

```
            if (System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())
            {
                System.Console.WriteLine("Current IP Addresses:");
                string hostname = System.Net.Dns.GetHostName();
                System.Net.IPHostEntry host = System.Net.Dns.GetHostEntry(hostname);
                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 in the command prompt to execute 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:

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2-alpine AS build
WORKDIR /app

COPY *.csproj ./
RUN dotnet restore

COPY . ./
RUN dotnet publish --configuration Release --output out

FROM mcr.microsoft.com/dotnet/core/runtime:2.2-alpine
WORKDIR /app

COPY --from=build /app/out .

ENTRYPOINT ["dotnet", "ipcheck.dll"]
```

4. Close the **Cloud Shell** pane.

7.4.4.3 Task 3: Create an Azure Container Registry resource

1. Create a new **container registry** with the following details:
   - **Name**: <Any globally unique name>
   - **Resource group**: ContainerCompute
   - **Location**: East US
   - **Admin user**: Disable
   - **SKU**: Basic
2. Wait for the creation task to complete before moving on with this lab.

7.4.4.4 Task 4: Open Cloud Shell and store Azure Container Registry metadata

1. Open a new **Cloud Shell** instance.
2. Within the **Cloud Shell** command prompt, use the **az acr list** command to view 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 *acrName*:

   ```
   acrName=$(az acr list --query "max_by([], &creationDate).name" --output tsv)
   ```

5. Use the following script to print the value of the Bash shell variable *acrName*:

   ```
   echo $acrName
   ```

7.4.4.5 Task 5: Deploy a Docker container image to Azure Container Registry

1. Change the active directory to ~/**clouddrive/ipcheck**.
2. Use the **dir** command to view the contents of the current directory.

   Note: You will know that you are 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 an **Azure Container Registry Task**:

   ```
   az acr build --registry $acrName --image ipcheck:latest .
   ```

4. Wait for the build task to complete before moving forward with this lab.
5. Close the **Cloud Shell** pane.

7.4.4.6 Task 6: Validate your container image in Azure Container Registry

1. Access the container registry that you created earlier in this lab.
2. Select the **Repositories** link to view your images stored in the registry.
3. Proceed through the **Images** and **Tags** blades to view the metadata associated with the **ipcheck** image with the **latest** tag.

   Note: You can also select the Run ID hyperlink to view the build task metadata.

7.4.4.7 Review

In this exercise, you created a .NET Core 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 Azure Container Registry.

7.4.5 Exercise 4: Deploy an Azure container instance

7.4.5.1 Task 1: Enable Admin User in Azure Container Registry

1. Access the container registry that you created earlier in this lab.
2. Select the **Update** button to view the settings for the Container Registry.
3. **Enable** the **Admin User**.

   Note: Your changes are automatically saved.

7.4.5.2 Task 2: Deploy a container image automatically to Azure container instance

1. Select the **Repositories** link to view your images stored in the registry.
2. Select the **ipcheck** image and view the **latest** tag for that image.
3. Select/right-click the **latest** tag 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

4. Wait for the creation task to complete before moving on with this lab.

7.4.5.3 Task 3: Deploy a container image manually to an Azure container instance

1. Access the container registry that you created earlier in this lab.
2. Select the **Access keys** link to view the credentials necessary to access your container registry from another service. Record the following values from this section to use later in this lab:
   - **Login server**
   - **Username**
   - **Password**

     Note: You will use these values later in this lab when you create another container instance.
3. Create a new **container instance** with the following details, using the **Access keys** credentials you recorded earlier in this lab:
   - **Container name**: manualcompute
   - **Image type**: Private
   - **Image name**: <Login server recorded earlier in the lab >/ipcheck:latest
   - **Image registry login server**: <Login server recorded earlier in the lab>
   - **Image registry user name**: <Username recorded earlier in the lab >
   - **Image registry password**: <Password recorded earlier in the lab >
   - **Resource group**: ContainerCompute
   - **Region**: East US
   - **OS type**: Linux
   - **Number of cores**: 1
   - **Memory (GB)**: 1.5
   - **Public IP address**: Yes
   - **Port**: 80
   - **Open additional ports**: No
   - **Port protocol**: TCP
   - **Restart policy**: On failure
4. Wait for the creation task to complete before moving forward with this lab.

7.4.5.4 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 view a list of the current containers running.
3. View the contents of the **Events** list for the container instance that ran your **ipcheck** application.
4. Select the **Logs** tab and observe the text logs from the container instance.

   Note: You can also optionally view the **Events** and **Logs** from the **managedcompute** container instance.

7.4.5.5 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 execute task-based applications as part of a container run.

7.4.6 Exercise 5: Clean up subscription

7.4.6.1 Task 1: Open Cloud Shell and list resource groups

1. At the top of the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

3. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

7.4.6.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **ContainerCompute** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

7.4.6.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.

7.4.6.4 Review

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

7.6 lab: title: 'Lab: Building a web application on Azure Platform-as-a-Service offerings' type: 'Answer Key' module: 'AZ-203T02-A: Develop Azure platform as a service (PaaS) compute solutions'

8 Lab: Building a web application on Azure Platform-as-a-Service offerings

9 Student lab answer key

9.1 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 steps to not match up.

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

9.2 Instructions

9.2.1 Before you start

9.2.1.1 Sign in to the lab virtual machine

Sign in to your **Windows 10** virtual machine 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.

9.2.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:

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

9.2.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T02** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

9.2.2 Exercise 1: Build a back-end API by using Azure Storage and Web apps

9.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure portal** (portal.azure.com).
3. At the sign-in page, enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. 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.

9.2.2.2 Task 2: Create an Azure Storage account

1. In the left navigation pane of the Azure portal, select **All services**.
2. In the **All services** blade, select **Storage Accounts**.
3. In the **Storage accounts** blade, view your list of Storage instances.
4. At the top of the **Storage accounts** blade, select **Add**.
5. In the **Create storage account** blade, observe the tabs at the top of 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 within the tab area, perform the following actions:
   1. Leave the **Subscription** field 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** field, enter **imgstor**[***your name in lowercase***].

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. In the **Access tier (default)** section, ensure that **Hot** is selected.
9. Select **Review + Create**.

7. In 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. In the **Deployment** blade, Wait for the creation task to complete before moving forward with this lab.
10. Click the **Go to resource** button in the **Deployment** blade to go to the newly created storage account.
11. In the **Storage account** blade, on the left side of the blade, locate the **Settings** section and select **Access keys**.
12. In the **Access keys** blade, select any one of the keys and record the value of either of the **Connection string** fields. You will use this value later in this lab.

    Note: It does not matter which connection string you choose. They are interchangeable.

9.2.2.3 Task 3: Upload a sample blob

1. On the Azure portal left navigation pane, select **Resource groups**.
2. In the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgstor*** storage account you created earlier in this lab.
4. In the **Storage Account** blade, on the left side of the 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** field, enter **images**.
    2. In the **Public access level** list, select **Blob (anonymous read access for blobs only)**.
    3. Select **OK**.
7. In the **Containers** section, select **+ Container** again.
8. In the **New container** window, perform the following actions:
    1. In the **Name** field, enter **images-thumbnails**.
    2. In the **Public access level** list, select **Blob (anonymous read access for blobs only)**.
    3. Select **OK**.
9. In the **Containers** section, select the newly created **images** container.
10. In the **Container** blade, select **Upload**.
11. In the **Upload blob** window that appears, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the File Explorer dialog box that appears, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **grilledcheese.jpg** file, and then select **Open**.
    3. Ensure that the **Overwrite if files already exist** check box is selected.
    4. Select **Upload**.
12. Wait for the blob to be uploaded before you continue with this lab.

9.2.2.4 Task 4: Create a Web app

1. At the top of the **New** blade, locate the **Search the Marketplace** field.
2. In the search field, enter **Web** and press Enter.
3. In the **Everything** search results blade, select the **Web App** result.
4. In the **Web App** blade, select **Create**.
5. In the second **Web App** blade, perform the following actions:
    1. In the **App name** field, enter **imgapi**[***your name in lowercase***].
    2. Leave the **Subscription** field set to its default value.
    3. In the **Resource group** section, select **Use existing**, and then select **ManagedPlatform**.
    4. In the **Publish** section, select **Code**.
    5. In the **Runtime stack** section, select **.NET Core 2.1 (LTS)**.
    6. In the **OS** section, select **Windows**.
    7. In the **Region** drop-down list, select **East US**
    8. Leave the **Windows Plan (East US)** field set to its default value.
    9. Leave the **Sku and size** field set to its default value.
    10. Select **Next: Monitoring**.
6. In the **Monitoring** tab, perform the following actions:
    1. In the **Enable Application Insights** section, select **No**.
    2. Select **Review + Create**.
7. In the **Review and create** tab, observe the settings then click **Create**.
8. Wait for the creation task to complete before you move forward with this lab.

9.2.2.5 Task 5: Configure a Web app

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgapi*** Web app that you created earlier in this lab.
4. In the **Web app** blade, on the left side of the blade in the **Settings** section, select the **Configuration** link.
5. In the **Configuration** section, perform the following actions:
    1. Select the **Application settings** tab.
    2. Select **+ New application setting**.
    3. In the **Add/Edit application setting** popup that appears, in the **Name** field, enter **StorageConnectionString**.
    4. In the **Value** field, enter the **Storage Connection String** you copied earlier in this lab.
    5. Leave the **deployment slot setting** field set to its default value.
    6. Select **OK** to close the popup and return to the **Configuration** section.
    7. Select **Save** at the top of the blade to persist your settings.
6. Wait for your application settings to persist before you move forward with the lab.
7. In the **Web app** blade, on the left side of the blade in the **Settings** section, select the **Properties** link.
8. In the **Properties** section, copy the value of the **URL** field. You will use this value later in the lab.

9.2.2.6 Task 6: Deploy an ASP.NET Core web application to Web app

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\API**, and then select **Select Folder**.
4. In the **Explorer** pane of the Visual Studio Code window, expand the **Controllers** folder and double-click the **ImagesController.cs** file to open the file in the editor.
5. In the editor, in the **ImagesController** class, on line 27, observe the **GetCloudBlobContainer** method and the code used to retrieve a container.
6. In the **ImagesController** class, on line 38, observe the **Get** method and the code used to retrieve all blobs asynchronously from the **images** container.
7. In the **ImagesController** class, on line 74, observe the **Post** method and the code used to persist an uploaded image to Azure Storage.
8. On the taskbar, select the **Windows PowerShell** icon.
9. In the open command prompt, enter the following command and press Enter to sign in to the Azure CLI:

   ```
   az login
   ```

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

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

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

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

14. Enter the following command and press Enter to print out only the name of the single app that has the prefix **imgapi***:

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

15. Enter the following command and press Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\API** directory that contains the lab files:

    ```
    cd F:\Labfiles\02\Starter\API\
    ```

16. Enter the following command and press Enter to deploy the **api.zip** file to the **Web app** 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.

17. Wait for the deployment to complete before you move forward with this lab.
18. On the left side of the portal, select the **Resource groups** link.
19. In the **Resource groups** blade, locate and select the **ManagedPlatform** resource group that you created earlier in this lab.
20. In the **ManagedPlatform** blade, select the **imgapi*** *Web app* that you created earlier in this lab.
21. In the **Web app** blade, select the **Browse** button.

22. The browser will open a new tab that performs a **GET** request to the root of the website. Observe the JSON array that is returned. This array should contain the URL for your single uploaded image in your **Azure Storage** account.
23. Return to your browser window showing the **Azure portal**.

9.2.2.7 Review

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

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

9.2.3.1 Task 1: Create a web app

1. In the Azure portal, on the left navigation pane, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **Web** and press Enter.
4. In the **Everything** search results blade, select the **Web App** result.
5. In the **Web App** blade, select **Create**.
6. In the second **Web App** blade, perform the following actions:
    1. In the **App name** field, enter **imgweb**[***your name in lowercase***].
    2. Leave the **Subscription** field set to its default value.
    3. In the **Resource group** section, select **Use existing**, and then select **ManagedPlatform**.
    4. In the **Publish** section, select **Code**.
    5. In the **Runtime stack** section, select **.NET Core 2.1 (LTS)**.
    6. In the **OS** section, select **Windows**.
    7. In the **Region** drop-down list, select **East US**
    8. Leave the **Windows Plan (East US)** field set to its default value.
    9. Leave the **Sku and size** field set to its default value.
    10. Select **Next: Monitoring**.
7. In the **Monitoring** tab, perform the following actions:
    1. In the **Enable Application Insights** section, select **No**.
    2. Select **Review + Create**.
8. In the **Review and create** tab, observe the settings then click **Create**.
9. Wait for the creation task to complete before you move forward with this lab.

9.2.3.2 Task 2: Configure a web app

1. On the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgweb*** web app that you created earlier in this lab.
4. In the **Web App** bladeblade, on the left side of the blade in the **Settings** section, select the **Configuration** link.
5. In the **Configuration** section, perform the following actions:
    1. Select the **Application settings** tab.
    2. Select **+ New application setting**.
    3. In the **Add/Edit application setting** popup that appears, in the **Name** field, enter **ApiUrl**.

4. In the **Value** field, enter the Web app **URL** you copied earlier in this lab.

   Note: Make sure you include the protocol (ex. *https://*) in the URL you copy into the value field for this application setting.

5. Leave the **deployment slot setting** field set to its default value.
6. Select **OK** to close the popup and return to the **Configuration** section.
7. Select **Save** at the top of the blade to persist your settings.

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

9.2.3.3 Task 3: Deploy an ASP.NET Core web application to web app

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Web**, and then select **Select Folder**.
4. In the **Explorer** pane of the Visual Studio Code window, expand the **Pages** folder and double-click 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 52, 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 PowerShell** icon.
8. In the open command prompt, enter the following command and press Enter to sign in to the Azure CLI:

   ```
   az login
   ```

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

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

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

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

13. Enter the following command and press Enter to print out only the name of the single app that has the prefix **imgweb***:

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

14. Enter the following command and press Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\Web** directory that contains the lab files:

    ```
    cd F:\Labfiles\02\Starter\Web\
    ```

15. Enter the following command and press Enter to deploy the **web.zip** file to the **web app** 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 you created earlier in this lab. You recently queried this app's name in the previous steps.

16. Wait for the deployment to complete before you move forward with this lab.
17. On the left navigation pane of the portal, select **Resource groups**.
18. In the **Resource groups** blade, select the **ManagedPlatform** resource group you created earlier in this lab.
19. In the **ManagedPlatform** blade, select the **imgweb*** web app that you created earlier in this lab.
20. In the **Web App** blade, select **Browse**.
21. Observe the list of images in the gallery. The gallery should list a single image that was uploaded to Azure Storage earlier in the lab.
22. At the top of the **Contoso Photo Gallery** webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **bahnmi.jpg** file, and then select **Open**.
    3. Select **Upload**.
23. Observe that the list of gallery images has been updated with your new image.

    Note: In some rare cases, you might need to refresh your browser window for the new image to appear.
24. Return to your browser window showing the **Azure portal**.

9.2.3.4 Review

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

9.2.4 Exercise 3: Build a background processing job by using Azure Storage and Azure Functions

9.2.4.1 Task 1: Create a function app

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Function** and then press Enter.
4. In the **Everything** search results blade, select the **Function App** result.
5. In the **Function App** blade, select **Create**.
6. In the **Function App** blade, observe the tabs at the top of the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new **function app**. At any time, you can select **Review + create** to skip the remaining tabs.

1. In the **Basics** tab, perform the following actions:
    1. Leave the **Subscription** text box set to its default value.
    2. In the **Resource group** section, **Use existing**, and then select **ManagedPlatform** from the list.
    3. In the **Function app name** text box, enter **imgfunc[your name in lowercase]**.
    4. In the **Publish** section, select **Code**.
    5. In the **Runtime stack** drop-down list, select **.NET Core**.
    6. In the **Region** drop-down list, select the **East US** region.
    7. Select **Next: Hosting**.
2. In the **Hosting** tab, perform the following actions:
    1. In the **Operating System** section, select **Windows**.

2. In the **Plan type** drop-down list, select the **App service plan** option.
3. Leave the **Windows Plan (East US)** field set to its default value.
4. In the **Storage account** drop-down list, select the **imgstor*** Storage account that you created earlier in this lab.

   Note: Changing the plan type or operating system resets the selected Storage account field. This setting must be changed last.
5. Select **Next: Monitoring**.

3. In the **Monitoring** tab, perform the following actions:
   1. In the **Enable Application Insights** section, select **No**.
   2. Select **Review + Create**.
4. In the **Review + Create** tab, review the options that you selected during the previous steps.
5. Select **Create** to create the function app by using your specified configuration.
6. Wait for the creation task to complete before you move forward with this lab.

9.2.4.2 Task 2: Create a .NET Core application setting

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgfunc*** function app that you created earlier in this lab.
4. In the **Function App** blade, select the **Platform features** tab.
5. In the **Platform features** tab, select the **Configuration** link located in the **General Settings** section.
6. In the **Configuration** section, perform the following actions:
   1. Select the **Application settings** tab.
   2. Select **+ New application setting**.
   3. In the **Add/Edit application setting** popup that appears, in the **Name** field, enter **DOTNET_SKIP_FIRST_TIME_EXPERIENCE**.
   4. In the **Value** field, enter **true**.

      Note: The `DOTNET_SKIP_FIRST_TIME_EXPERIENCE` application setting tells .NET Core to disable it's built-in NuGet package caching mechanisms. On a temporary compute instance, this would effectively be a waste of time and cause build issues with your Azure Function.
   5. Leave the **deployment slot setting** field set to its default value.
   6. Select **OK** to close the popup and return to the **Configuration** section.
   7. Select **Save** at the top of the blade to persist your settings.
7. Wait for your application settings to persist before you move forward with the lab.

9.2.4.3 Task 3: Author a function to process blobs

1. On the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, locate and select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgfunc*** function app that you created earlier in this lab.
4. In the **Function App** blade, select **+ New function**.
5. In the **New Azure Function** quickstart, perform the following actions:
   1. Under the **Choose a Development Environment** header, select **In-Portal**.
   2. Select **Continue**.

3. Under the **Create a Function** header, select **More templates...**.
4. Select **Finish and view templates**.
5. In the **Templates** list, select **Azure Blob Storage trigger**.
6. In the **Extensions not Installed** window, select **Install**.

   Note: It can take up to two minutes to install the extensions needed to work with Azure Storage blobs. If the portal does not refresh, simply close the **Extensions not Installed** pop-up window and select **Azure Blob Storage trigger** again.
7. Once the installation has succeeded, select **Continue**.
8. In the **New Function** window, in the **Name** field, enter **ImageManager**.
9. In the **New Function** window, in the **Path** field, enter **images/{name}**.
10. In the **New Function** window, in the **Storage account connection** list, select **AzureWebJobsStorage**.
11. In the **New Function** window, select **Create**.

6. On the right side of the function editor, select **View files** to open the tab.
7. In the **View files** tab, select **Add**.
8. In the filename dialog that appears, enter **function.proj**.
9. In the file editor, insert this configuration content:

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="SixLabors.ImageSharp" Version="1.0.0-beta0006" />
    </ItemGroup>
</Project>
```

10. In the editor, select **Save** button to persist your changes to the configuration.

    Note: This **.proj** file contains the NuGet package reference necessary to import the SixLabors.ImageSharp package.
11. Back in the **View files** tab, select the **function.json** file to view the editor for the function's configuration.
12. In the JSON editor, observe the current configuration:

```
{
  "bindings": [
    {
      "name": "myBlob",
      "type": "blobTrigger",
      "direction": "in",
      "path": "images/{name}",
      "connection": "AzureWebJobsStorage"
    }
  ],
  "disabled": false
}
```

13. Replace the entire contents of the JSON configuration file with the following JSON content:

```
{
    "bindings": [
        {
            "name": "inputBlob",
            "type": "blobTrigger",
            "direction": "in",
            "path": "images/{name}",
```

```
            "connection": "AzureWebJobsStorage"
        },
        {
            "type": "blob",
            "name": "outputBlob",
            "path": "images-thumbnails/{name}",
            "connection": "AzureWebJobsStorage",
            "direction": "out"
        }
    ]
}
```

14. In the editor, select **Save** to persist your changes to the configuration.
15. Back in the **View files** tab, select the **run.csx** file to return to the editor for the **ImageManager** function.
16. Minimize the **View files** tab.

    Note: You can minimize the tab by selecting the arrow immediately to the right of the tab header.

17. In the function editor, observe the example function script:

```
public static void Run(Stream myBlob, string name, ILogger log)
{
    log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Le
}
```

18. **Delete** all the example code.
19. Within the editor, copy and paste the following placeholder function:

```
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Processing;
using SixLabors.ImageSharp.Formats.Jpeg;
using SixLabors.Primitives;

public static void Run(Stream inputBlob, Stream outputBlob, string name, ILogger log)
{
}
```

20. Select **Save** to save the script and compile the code.
21. Add the following line of code within the **Run** method to log information about the function execution:

```
log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {inputBlob.Leng
```

22. Add the following **using** statement to load the **Stream** for the input blob into the image library:

```
using (Image<Rgba32> image = Image.Load(inputBlob))
{
}
```

23. Add the following lines of code within the **using** statement to mutate the image by resizing the image and applying a grayscale filter:

```
image.Mutate(i =>
    i.Resize(new ResizeOptions { Size = new Size(250, 250), Mode = ResizeMode.Max }).Grayscale()
);
```

24. Add the following line of code to save the new image to the **Stream** for the output blob:

```
image.Save(outputBlob, new JpegEncoder());
```

25. Your **Run** method should now resemble this:

```
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Processing;
```

```
using SixLabors.ImageSharp.Formats.Jpeg;
using SixLabors.Primitives;

public static void Run(Stream inputBlob, Stream outputBlob, string name, ILogger log)
{
    log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {inputBlob
    using (Image<Rgba32> image = Image.Load(inputBlob))
    {
        image.Mutate(i =>
            i.Resize(new ResizeOptions { Size = new Size(250, 250), Mode = ResizeMode.Max }).Grays
        );
        image.Save(outputBlob, new JpegEncoder());
    }
}
```

26. Select **Save** to save the script and compile the code again.

9.2.4.4 Task 4: Validate the web solution

1. On the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgstor*** storage account that you created earlier in this lab.
4. In the **Storage Account** blade, on the left side of the blade, in the **Blob service** section, select the **Containers** link.
5. In the **Containers** section, select the **images** container.
6. In the **Container** blade, select **Upload**.
7. In the **Upload blob** window that appears, perform the following actions:
    1. In the **Files** section, select the **Folder** icon.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **veggie.jpg** file, and then select **Open**.
    3. Ensure the **Overwrite if files already exist** check box is selected.
    4. Select **Upload**.
8. Wait for the blob to be uploaded before you continue with this lab.
9. Close the **Container** blade.
10. Back in the **Containers** section, select the **images-thumbnails** container.
11. In the **Container** blade, observe the newly created **veggie.jpg** file in the **images-thumbnails** container.

    Note: It might take one to five minutes for the new image to appear.
12. Select the **veggie.jpg** blob in the **images-thumbnails** container.
13. In the **Blob** blade, select the **Edit blob** tab.
14. Observe the contents of the blob. The webpage will render the image that was uploaded to the container.
15. On the left navigation pane of the portal, select **Resource groups**.
16. In the **Resource groups** blade, select the **ManagedPlatform** resource group that you created earlier in this lab.
17. In the **ManagedPlatform** blade, select the **imgweb*** web app that you created earlier in this lab.
18. In the **Web App** blade, select **Browse**.
19. Observe the list of images in the gallery. The list of thumbnails should now be updated with a new thumbnail image.

20. At the top of the **Contoso Photo Gallery** webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **blt.jpg** file, and then select **Open**.
    3. Select **Upload**.
21. At the top of the **Contoso Photo Gallery** webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **sub.jpg** file, and then select **Open**.
    3. Select **Upload**.
22. At the top of the **Contoso Photo Gallery** webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **burger.jpg** file, and then select **Open**.
    3. Select **Upload**.
23. Observe that the list of gallery images has been updated with your new image.
24. Observe the list of thumbnails at the top of the page. Refresh your page every minute until your thumbnails have been generated.

9.2.4.5 Review

In this exercise, you created a background processing job in Azure Functions to handle the computationally intensive task of modifying and resizing images.

9.2.5 Exercise 4: Clean up subscription

9.2.5.1 Task 1: Open Cloud Shell

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than symbol and underscore character.
2. If this is your first time opening the **Cloud Shell** by using your subscription, a **Welcome to Azure Cloud Shell Wizard** will appear that allows you to configure **Cloud Shell** for first-time usage. Perform the following actions in the wizard:
    1. A dialog box will appear that prompts you to create a new Storage Account to begin using the shell. Accept the default settings and select **Create storage**.
    2. Wait for the **Cloud Shell** to finish its first-time setup procedures before moving forward with the lab.

       Note: If you do not see the configuration options for the **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. The labs are written from the presumption that you are using a new subscription.
3. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```
4. Type in the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

9.2.5.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **ManagedPlatform** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

9.2.5.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Visual Studio Code** application.

9.2.5.4 Review

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

9.4 lab: title: 'Lab: Building a web application on Azure Platform-as-a-Service offerings' module: 'AZ-203T02-A: Develop Azure platform as a service (PaaS) compute solutions'

10 Lab: Building a web application on Azure Platform-as-a-Service offerings

11 Student lab manual

11.1 Lab scenario

You are 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 applications and APIs. Recently, you realized that the images that are being uploaded to your application are simply too large. Out of concern for storage and bandwidth costs, you decided to use Azure Functions to process images in the background and shrink them down to thumbnails that can be loaded quickly.

11.2 Objectives

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

- Create various apps by using the Azure App Service.
- Configure application settings for an app.
- Deploy apps by using Kudu, the Azure CLI, and zip deployment.
- Author a function app to process Azure Storage images.

11.3 Lab setup

- Estimated time: 90 minutes

11.4 Instructions

11.4.1 Before you start

11.4.1.1 Sign in to the lab virtual machine

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

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

11.4.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:

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

11.4.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T02** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

11.4.2 Exercise 1: Build a back-end API by using Azure Storage and Web apps

11.4.2.1 Task 1: Open the Azure portal

1. Sign in to the **Azure portal** (portal.azure.com).
2. 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.

11.4.2.2 Task 2: Create an Azure Storage account

1. Create a new **storage account** with the following details:
   - **New resource group**: ManagedPlatform
   - **Name**: imgstor[your name in lowercase]
   - **Location**: (US) East US
   - **Performance**: Standard
   - **Account kind**: StorageV2 (general purpose v2)
   - **Replication**: Locally-redundant storage (LRS)
   - **Access tier**: Hot

   Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You will receive a notification when the account is created.

1. Access the **Access Keys** blade of your newly created **storage account** instance.
2. Record the value of the **Connection string** field. You will use this value later in this lab.

11.4.2.3 Task 3: Upload a sample blob

1. Access the **imgstor*** 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. Create another new **container** with the following settings:
   - **Name**: images-thumbnails
   - **Public access level**: Blob (anonymous read access for blobs only)
5. Navigate to the new **images** container.
6. Use the **Upload** button to upload the **grilledcheese.jpg** file located in the **Allfiles (F):\Allfiles\Labs\02\Starter** folder on your lab machine.

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

11.4.2.4 Task 4: Create a Web app

1. Create a new **Web app** with the following details:
   - **App name**: imgapi[your name in lowercase]
   - **Existing resource group**: ManagedPlatform
   - **Runtime stack**: .NET Core 2.1 (LTS)
   - **OS**: Windows
   - **Enable Application Insights**: No

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

11.4.2.5 Task 5: Configure a Web app

1. Access the **imgapi*** *Web app* that you created earlier in this lab.
2. In the **Settings** section on the left side of the blade, navigate to the **Configuration** section.
3. 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
4. Save your changes to the **Application settings**.
5. In the **Settings** section on the left side of the blade, navigate to the **Properties** section.
6. In the **Properties** section, copy the value of the **URL** field. You will use this value later in the lab.

11.4.2.6 Task 6: Deploy an ASP.NET Core web application to Web app

1. Using **Visual Studio Code**, open the web application located in the **Allfiles (F):\Allfiles\Labs\02\Starter\API** folder.
2. Open the **Controllers\ImagesController.cs** file and observe the code in each of the methods.
3. Open the **Windows PowerShell** application.
4. Sign in to the Azure CLI by using your Microsoft 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 prefix **imgapi***:

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

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

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

8. Change the current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\API** directory that contains the lab files:

   ```
   cd F:\Labfiles\02\Starter\API\
   ```

9. Deploy the **api.zip** file to the **Web app** 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*** *Web app* you created earlier in this lab.
11. Open the **imgapi*** *Web app* in your browser.
12. The browser will open a new tab that performs a **GET** request to the root of the website. Observe the JSON array that is returned. This array should contain the URL for your single uploaded image in your **Azure Storage** account.

11.4.2.7 Review

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

11.4.3 Exercise 2: Build a front-end web application by using Azure web apps

11.4.3.1 Task 1: Create web app

1. In the Azure portal, create a new **web app** with the following details:
   - **App name**: imgweb[your name in lowercase]
   - **Existing resource group**: ManagedPlatform
   - **Runtime stack**: .NET Core 2.1 (LTS)
   - **OS**: Windows
   - **Enable Application Insights**: No

   Note: Wait for Azure to finish creating the web app pbefore you move forward with the lab. You will receive a notification when the app is created.

11.4.3.2 Task 2: Configure a web app

1. Access the **imgweb*** *Web App* you created in the previous task.
2. In the **Settings** section on the left side of the blade, navigate to the **Configuration** settings.
3. Create a new **application setting** 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 (ex. *https://*) in the URL you copy into the value field for this application setting.
4. **Save** your changes to the Application Settings.

11.4.3.3 Task 3: Deploy an ASP.NET Core web application to Web App

1. Using **Visual Studio Code**, open the web application located in the **Allfiles (F):\Allfiles\Labs\02\Starter\Web** folder.
2. Open the **Pages\Index.cshtml.cs** file and observe the code in each of the methods.
3. Open the **Windows PowerShell** application.

4. Sign in to the Azure CLI by using your Microsoft 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 prefix **imgweb***:

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

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

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

8. Change your current directory to the **Allfiles (F):\Allfiles\Labs\02\Starter\Web** directory that contains the lab files:

   ```
   cd F:\Labfiles\02\Starter\Web\
   ```

9. 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.

10. Access the **imgweb*** web app that you created earlier in this lab.
11. Open the **imgweb*** web app in your browser.
12. Observe that the list of gallery images has been updated with your new image.

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

11.4.3.4 Review

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

11.4.4 Exercise 3: Build a background processing job by using Azure Storage and Azure Functions

11.4.4.1 Task 1: Create a function app

1. In the Azure portal, create a new **function app** with the following details:
   - **Existing resource group**: ManagedPlatform
   - **App name**: imgfunc[your name in lowercase]
   - **Publish**: Code
   - **Runtime Stack**: .NET Core
   - **Region**: East US
   - **Storage account**: imgstor[your name in lowercase here]
   - **Operating System**: Windows
   - **Plan type**: App service plan
   - **Enable Application Insights**: No

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

11.4.4.2 Task 2: Create a .NET Core application setting

1. Access the **imgfunc*** *Function App* that you created earlier in this lab.
2. Navigate to the **Configuration** settings located in the **Platform features** tab.
3. Create a new **application setting** by using the following details:
   - **Name**: DOTNET_SKIP_FIRST_TIME_EXPERIENCE
   - **Value**: true

     Note: The `DOTNET_SKIP_FIRST_TIME_EXPERIENCE` application setting tells .NET Core to disable it's built-in NuGet package caching mechanisms. On a temporary compute instance, this would effectively be a waste of time and cause build issues with your Azure Function.
4. Save your changes to the **Application settings**.

11.4.4.3 Task 3: Author a function to process blobs

1. On the Azure portal left navigation pane, select **Resource groups**.
2. In the **Resource groups** blade, locate and select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgfunc*** function app that you created earlier in this lab.
4. In the **Function App** blade, select **+ New function**.
5. In the **New Azure Function** quickstart, perform the following actions:
   1. Under the **Choose a Development Environment** header, select **In-Portal**.
   2. Select **Continue**.
   3. Under the **Create a Function** header, select **More templates...**.
   4. Select **Finish and view templates**.
   5. In the **Templates** list, select **Azure Blob Storage trigger**.
   6. In the **Extensions not Installed** window, select **Install**.

      Note: It can take up to two minutes to install the extensions needed to work with Azure Storage blobs. If the portal does not refresh, simply close the **Extensions not Installed** pop-up window and select **Azure Blob Storage trigger** again.
   7. Once the installation has succeeded, select **Continue**.
   8. In the **New Function** window, in the **Name** field enter **ImageManager**.
   9. In the **New Function** window, in the **Path** field enter **images/{name}**.
   10. In the **New Function** window, in the **Storage account connection** list, select **AzureWebJobsStorage**.
   11. In the **New Function** window, select **Create**.
6. On the right side of the function editor, select **View files** to open the tab.
7. In the **View files** tab, select **Add**.
8. In the filename dialog that appears, enter **function.proj**.
9. In the file editor, insert this configuration content:

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="SixLabors.ImageSharp" Version="1.0.0-beta0006" />
    </ItemGroup>
</Project>
```

10. In the editor, select **Save** button to persist your changes to the configuration.

    Note: This **.proj** file contains the NuGet package reference necessary to import the SixLabors.ImageSharp package.

11. Back in the **View files** tab, select the **function.json** file to view the editor for the function's configuration.

12. In the JSON editor, observe the current configuration:

```
{
  "bindings": [
    {
      "name": "myBlob",
      "type": "blobTrigger",
      "direction": "in",
      "path": "images/{name}",
      "connection": "AzureWebJobsStorage"
    }
  ],
  "disabled": false
}
```

13. Replace the entire contents of the JSON configuration file with the following JSON content:

```
{
  "bindings": [
    {
      "name": "inputBlob",
      "type": "blobTrigger",
      "direction": "in",
      "path": "images/{name}",
      "connection": "AzureWebJobsStorage"
    },
    {
      "type": "blob",
      "name": "outputBlob",
      "path": "images-thumbnails/{name}",
      "connection": "AzureWebJobsStorage",
      "direction": "out"
    }
  ]
}
```

14. In the JSON editor, select **Save** button to persist your changes to the configuration.

15. Back in the **View files** tab, select the **run.csx** file to return to the editor for the **ImageManager** function.

16. Minimize the **View files** tab.

    Note: You can minimize the tab by selecting the arrow immediately to the right of the tab header.

17. In the function editor, observe the example function script:

```
public static void Run(Stream myBlob, string name, ILogger log)
{
    log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Le
}
```

18. **Delete** all the example code.

19. Within the editor, copy and paste the following placeholder function:

```
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Processing;
using SixLabors.ImageSharp.Formats.Jpeg;
```

```
using SixLabors.Primitives;

public static void Run(Stream inputBlob, Stream outputBlob, string name, ILogger log)
{
}
```

20. In the editor, select **Save** to save the script and compile the code.
21. Add the following line of code to log information about the function execution:

```
log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {inputBlob.Leng
```

22. Add the following **using** statement to load the **Stream** for the input blob into the image library:

```
using (Image<Rgba32> image = Image.Load(inputBlob))
{
}
```

23. Add the following lines of code within the **using** statement to mutate the image by resizing the image and applying a grayscale filter:

```
image.Mutate(i =>
    i.Resize(new ResizeOptions { Size = new Size(250, 250), Mode = ResizeMode.Max }).Grayscale()
);
```

24. Add the following line of code to save the new image to the **Stream** for the output blob:

```
image.Save(outputBlob, new JpegEncoder());
```

25. Your **Run** method should now resemble the following:

```
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Processing;
using SixLabors.ImageSharp.Formats.Jpeg;
using SixLabors.Primitives;

public static void Run(Stream inputBlob, Stream outputBlob, string name, ILogger log)
{
    log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {inputBlob
    using (Image<Rgba32> image = Image.Load(inputBlob))
    {
        image.Mutate(i =>
            i.Resize(new ResizeOptions { Size = new Size(250, 250), Mode = ResizeMode.Max }).Grays
        );
        image.Save(outputBlob, new JpegEncoder());
    }
}
```

26. In the editor, select **Save** to save the script and compile the code again.

11.4.4.4 Task 4: Validate the web solution

1. On the Azure portal left navigation pane, select **Resource groups**.
2. In the **Resource groups** blade, locate and select the **ManagedPlatform** resource group that you created earlier in this lab.
3. In the **ManagedPlatform** blade, select the **imgstor*** storage account that you created earlier in this lab.
4. In the **Storage Account** blade, in the **Blob service** section on the left side of the blade, select the **Containers** link.
5. In the **Containers** section, select the **images** container.
6. In the **Container** blade, select **Upload**.
7. In the **Upload blob** window that appears, perform the following actions:

1. In the **Files** section, select the **Folder** icon.
2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **veggie.jpg** file, and then select **Open**.
3. Ensure that the **Overwrite if files already exist** check box is selected.
4. Select **Upload**.

8. Wait for the blob to be uploaded before you continue with this lab.
9. Close the **Container** blade.
10. Back in the **Containers** section, select the **images-thumbnails** container.
11. In the **Container** blade, observe the newly created **veggie.jpg** file in the **images-thumbnails** container.

    Note: It might take one to five minutes for the new image to appear.

12. In the **images-thumbnails** container, select the **veggie.jpg** blob.
13. In the **Blob** blade, select the **Edit blob** tab.
14. Observe the contents of the blob.
15. On the left side of the portal, select **Resource groups**.
16. In the **Resource groups** blade, locate and select the **ManagedPlatform** resource group that you created earlier in this lab.
17. In the **ManagedPlatform** blade, select the **imgweb*** web app that you created earlier in this lab.
18. In the **Web App** blade, select **Browse**.
19. Observe the list of images in the gallery.
20. At the top of the **Contoso Photo Gallery** webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **blt.jpg** file, and then select **Open**.
    3. Select the **Upload**.
21. At the top of the webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **sub.jpg** file, and then select **Open**.
    3. Select **Upload**.
22. At the top of the webpage, locate the **Upload a new image** section and perform the following actions:
    1. Select **Browse**.
    2. In the File Explorer dialog box that opens, go to **Allfiles (F):\Allfiles\Labs\02\Starter\Images**, select the **burger.jpg** file, and then select **Open**.
    3. Select **Upload**.
23. On the **Contoso Photo Gallery** webpage, observe that the list of gallery images has been updated with your new image.
24. Observe the list of thumbnails at the top of the page. Refresh your page every minute until your thumbnails have been generated.

11.4.4.5 Review

In this exercise, you created a background processing job in Azure Functions to handle the computationally intensive task of modifying and resizing images.

11.4.5 Exercise 4: Clean up subscription

11.4.5.1 Task 1: Open Cloud Shell

1. At the top of the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
3. At the bottom of the portal in the **Cloud Shell** command prompt, type the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

4. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

11.4.5.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **ManagedPlatform** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

11.4.5.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Visual Studio Code** application.

11.4.5.4 Review

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

11.6 lab: title: 'Lab: Constructing a polyglot data solution' type: 'Answer Key' module: 'AZ-203T03-A: Develop for Azure Storage'

12 Lab: Constructing a polyglot data solution

13 Student lab answer key

13.1 Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure user interface (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 as soon as the community brings needed changes to our attention. However, because cloud updates occur frequently, you might encounter UI changes before this training content is updated. **If this occurs, adapt to the changes and work through them in the labs as needed.**

13.2 Instructions

13.2.1 Before you start

13.2.1.1 Sign in to the lab virtual machine

Sign in to your **Windows 10** virtual machine using the following credentials:

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

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

13.2.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:

- Microsoft Edge
- File Explorer
- Visual Studio Code

13.2.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T03** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

13.2.2 Exercise 1: Creating database resources in Azure

13.2.2.1 Task 1: Open the Azure portal

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

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

13.2.2.2 Task 2: Create an Azure Cache for Redis resource

1. In the navigation pane on the left side of the Azure portal, select **All services**.
2. In the **All services** blade, select **Azure Cache for Redis**.
3. In the **Azure Cache for Redis** blade, view the list of your Redis cache instances.
4. At the top of the **Azure Cache for Redis** blade, select **+ Add**.
5. In the **New Redis Cache** blade, perform the following actions:
   1. IN the **DNS name** field, enter the value **polyrediscache[your name in lowercase]**
   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 **Location** drop-down list, select the **East US** option.
   5. In the **Pricing tier** list, select the **Basic C0 (250MB Cache)** option.
   6. In the **Unblock port 6379 (not SSL encrypted)** section, ensure the checkbox is not selected.
   7. Select **Create**.
6. Wait for the creation task to complete before you move forward with this lab.

Note: An Azure Cache for Redis resource can take anywhere from **15-30 minutes** to become ready for use. You can choose to move forward with the lab, but you must remember that this resource and its connection string is required for **Exercise 6: Authoring .NET code to connect to Azure Redis Cache**.

7. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
8. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
9. In the **PolyglotData** blade, select the **polyrediscache*** Azure Cache for Redis instance that you created earlier in this lab.
10. In the **Azure Cache for Redis** blade, locate the **Settings** section on the left side of the blade and select the **Access keys** link.
11. In the **Access keys** pane, record the value in the **Primary connection string (StackExchange.Redis)** field. You will use this value later in this lab.

13.2.2.3 Task 3: Create an Azure SQL server resource

1. In the navigation pane on the left side of the Azure portal, select **All services**.
2. In the **All services** blade, select **SQL servers**.
3. In the **SQL servers** blade, view your list of SQL server instances.
4. At the top of the **SQL servers** blade, select **+ Add**.
5. In the **Create SQL Database Server** blade, observe the tabs at the top of 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**. At any time, you can select **Review + create** to skip the remaining tabs.
6. In the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** drop-down list set to its default value.
   2. In the **Resource group** section, select **PolyglotData** from the list.
   3. In the **Server name** field, enter the value **polysqlsrvr**[**your name in lowercase**].
   4. In the **Location** drop-down list, select the **(US) East US** option.
   5. In the **Server admin login** field, enter the value **testuser**.
   6. In the **Password** field, enter the value **TestPa$$w0rd**.
   7. In the **Confirm password** field, enter the value **TestPa$$w0rd** again.
   8. Select **Next: Networking >**.
7. In 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**.
8. In the **Review + Create** tab, review the options that you selected during the previous steps.
9. Select **Create** to create the Azure SQL Database server by using your specified configuration.
10. Wait for the creation task to complete before you move forward with this lab.

13.2.2.4 Task 4: Create an Azure Cosmos DB account resource

1. In the navigation pane on the left side of the Azure portal, select **All services**.
2. In the **All services** blade, select **Azure Cosmos DB**.
3. In the **Azure Cosmos DB** blade, view your list of Azure Cosmos DB instances.
4. At the top of the **Azure Cosmos DB** blade, select **+ Add**.

5. In the **Create Azure Cosmos DB Account** blade, observe the tabs at the top of 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**. At any time, you can select **Review + create** to skip the remaining tabs.

6. In 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** field, enter **polycosmos**[**your name in lowercase**].
   4. In the **API** drop-down list, select the **Core (SQL)** option.
   5. In the **Location** drop-down list, select the **East US** region.
   6. In the **Geo-Redundancy** section, select the **Disable** option.
   7. In the **Multi-region Writes** section, select the **Disable** option.
   8. Select **Review + Create**.
7. In 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.
9. Wait for the creation task to complete before you move forward with this lab.
10. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
11. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
12. In the **PolyglotData** blade, select the **polycosmos*** Azure Cosmos DB account that you created earlier in this lab.
13. In the **Azure Cosmos DB account** blade, locate the **Settings** section on the left side of the blade and select the **Keys** link.
14. In the **Keys** pane, record the value in the **PRIMARY CONNECTION STRING** field. You will use this value later in this lab.

13.2.2.5 Task 5: Create an Azure Storage account resource

1. In the navigation pane on the left side of the Azure portal, select **All services**.
2. In the **All services** blade, select **Storage Accounts**.
3. In the **Storage accounts** blade, view your list of Storage instances.
4. At the top of the **Storage accounts** blade, select **+ Add**.
5. In the **Create storage account** blade, observe the tabs at the top of the blade, such as **Basics**, **Advanced** and **Tags**.

   Note: Each tab represents a step in the workflow to create a new **Azure Storage Account**. At any time, you can select **Review + create** to skip the remaining tabs.

6. In 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** field, enter **polystor**[**your name in lowercase**].
   4. In the **Location** drop-down list, select the **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. In the **Access tier (default)** section, ensure that **Hot** is selected.

9. Select **Review + Create**.

7. In 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.
9. Wait for the creation task to complete before you move forward with this lab.

13.2.2.6 Review

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

13.2.3 Exercise 2: Import databases and images

13.2.3.1 Task 1: Upload image blobs

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
3. In the **PolyglotData** blade, select the **polystor*** storage account that you created earlier in this lab.
4. In the **Storage account** blade, select the **Containers** link located in the **Blob service** section on the left side of the blade.
5. In the **Containers** section, select + **Container**.
6. In the **New container** pop-up, perform the following actions:
    1. In the **Name** field, enter **images**.
    2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**.
    3. Select **OK**.
7. Back in the **Containers** section, select the newly created **images** container.
8. In the **Container** blade, locate the **Settings** section on the left side of the blade and select the **Properties** link.
9. In the **Properties** pane, record the value in the **URL** field. You will use this value later in this lab.
10. Locate and select the **Overview** link on the left side of the blade.
11. At the top of 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 dialog box, go to **Allfiles (F):\Allfiles\Labs\03\Starter\Images**, select all fourty-two **.jpg** image files, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected.
    4. Select **Upload**.
13. Wait for all of the blobs to be uploaded before you continue with this lab.

13.2.3.2 Task 2: Upload SQL .bacpac file

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
3. In the **PolyglotData** blade, select the **polystor*** storage account that you created earlier in this lab.
4. In the **Storage account** blade, select the **Containers** link located in the **Blob service** section on the left side of the blade.
5. In the **Containers** section, select + **Container**.
6. In the **New container** pop-up, perform the following actions:

1. In the **Name** field, 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. In 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 dialog box, go to **Allfiles (F):\Allfiles\Labs\03\Starter**, select the **AdventureWorks.bacpac** file, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected.
    4. Select **Upload**.
10. Wait for the blob to be uploaded before you continue with this lab.

13.2.3.3 Task 3: Import SQL database

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
3. In the **PolyglotData** blade, select the **polysqlsrvr*** SQL server that you created earlier in this lab.
4. In the **SQL server** blade, select **Import database**.
5. In the **Import database** blade that appears, perform the following actions:
    1. Leave the **Subscription** list set to its default value.
    2. Select the **Storage** option.
    3. In the **Storage accounts** blade that appears, select the **polystor*** storage account that you created earlier in this lab.
    4. In the **Containers** blade that appears, select the **databases** container that you created earlier in this lab.
    5. In the **Container** blade that appears, select the **AdventureWorks.bacpac** blob that you created earlier in this lab and then select **Select** to close the blade.
    6. Back in the **Import database** blade, leave the **Pricing tier** option set to its default value.
    7. In the **Database name** field, enter **AdventureWorks**.
    8. Leave the **Collation** field set to its default value.
    9. In the **Server admin login** field, enter the value **testuser**.
    10. In the **Password** field, enter the value **TestPa$$w0rd**.
    11. Select **OK**.
6. Wait for the database to be created before you continue with this lab.

13.2.3.4 Task 4: Use imported SQL Database

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
3. In the **PolyglotData** blade, select the **polysqlsrvr*** SQL server that you created earlier in this lab.
4. In the **SQL server** blade, locate the **Security** section on the left side of the blade and select the **Firewalls and virtual networks** link.
5. In the **Firewalls and virtual networks** pane, select **Add client IP** and then select **Save**.

Note: This step will ensure that your local machine will have access to the databases associated with this server.

6. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
7. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
8. In the **PolyglotData** blade, select the **AdventureWorks** SQL database that you created earlier in this lab.
9. In the **SQL database** blade, locate the **Settings** section on the left side of the blade and select the **Connection strings** link.
10. In the **Connection strings** pane, record the value in the **ADO.NET (SQL Authentication)** field. You will use this value later in this lab.
11. Update the connection string you recorded by performing the following actions:
    1. Within the connection string, locate the `{your_username}` placeholder and replace it with `testuser`.
    2. Within the connection string, locate the `{your_password}` placeholder and replace it with `TestPa$$w0rd`.

       Note: For example, if your connection string was originally `Server=tcp:polysqlsrvrinstructor.database.windows.net,1433; 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=TestPa$$w0rd;`
12. Locate and select the **Query editor** link on the left side of the blade.
13. In the **Query editor** pane, perform the following actions:
    1. In the **Login** field, enter the value **testuser**.
    2. In the **Password** field, enter the value **TestPa$$w0rd**.
    3. Select **OK**.
14. In the open query editor, enter the following query:

    ```
    SELECT * FROM AdventureWorks.dbo.Models
    ```
15. Select **Run** to execute the query.
16. Observe the results of the query.

    Note: This query will return a list of models that will appear on the homepage of the web application.
17. In the query editor, replace the existing query with the following query:

    ```
    SELECT * FROM AdventureWorks.dbo.Products
    ```
18. Select **Run** to execute the query.
19. Observe the results of the query.

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

13.2.3.5 Review

In this exercise, you imported all of the resources you will use with your web application.

13.2.4 Exercise 3: Open and configure a .NET Core web application

13.2.4.1 Task 1: Open and build the web application

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\03\Starter\AdventureWorks**, and then select **Select Folder**.

4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to build the .NET Core 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 the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.4.2 Task 2: Update SQL connection string

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Web** project.
2. Double-click (or double-select) the **appsettings.json** file.
3. In the JSON object, in line 3, locate the **ConnectionStrings.AdventureWorksSqlContext** path. Observe that the current value is empty:

   ```
   "ConnectionStrings": {
       "AdventureWorksSqlContext": "",
       "AdventureWorksCosmosContext": "",
       "AdventureWorksRedisContext": ""
   },
   ```
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 is important that you use your updated connection string here. The original connection string copied from the portal will not have the username and password necessary to connect to the SQL database.
5. Save the **appsettings.json** file.

13.2.4.3 Task 3: Update blob base URL

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

   ```
   "Settings": {
       "BlobContainerUrl": "",
       "CartAvailable": false,
       "UniqueIdentifier": "az:dev:student"
   }
   ```
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.

13.2.4.4 Task 4: Validate web application

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

   ```
   cd .\AdventureWorks.Web\
   ```
3. In the command prompt, enter the following command and press Enter to run the .NET Core 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, navigate to the your currently running web application (http://localhost:5000).
6. In the web application, observe the list of models displayed on the front page.
7. Locate the **Water Bottle** model and select **View Details**.
8. On the **Water Bottle** product detail page, select **Add to Cart**.
9. Observe that the checkout functionality is currently disabled.

   > **Note**: For now, only the product page functionality is implemented. You will implement the checkout logic later in this lab.

10. Close the browser window showing your web application.
11. Return to the **Visual Studio Code** window.
12. Back in the **Visual Studio Code** window, select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.4.5 Review

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

13.2.5 Exercise 4: Migrating SQL data to Azure Cosmos DB

13.2.5.1 Task 1: Create migration project

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

   ```
   dotnet new console --name AdventureWorks.Migrate --langVersion 7.1
   ```

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

3. In the command prompt, enter the following command and press 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. In the command prompt, enter the following command and press Enter to 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.

5. In the command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

   ```
   cd .\AdventureWorks.Migrate
   ```

6. In the command prompt, enter the following command and press Enter to import version **2.2.6** of the **Microsoft.EntityFrameworkCore.SqlServer** from NuGet:

   ```
   dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 2.2.6
   ```

   > **Note**: The `dotnet add package` command will add the **Microsoft.EntityFrameworkCore.SqlServer** package from **NuGet**.

7. In the command prompt, enter the following command and press Enter to import version **3.7.0** of the **Microsoft.Azure.Cosmos** from NuGet:

```
dotnet add package Microsoft.Azure.Cosmos --version 3.7.0
```

> **Note**: The `dotnet add package` command will add the **Microsoft.Azure.Cosmos** package from **NuGet**.

8. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

```
dotnet build
```

9. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.5.2 Task 2: Create .NET class

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Migrate** project.
2. Double-click (or double-select) the **Program.cs** file.
3. In the code editor tab for the **Program.cs** file, delete all 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 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 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 is important that you use your updated connection string here. The original connection string copied from the portal will not 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 the **cosmosDBConnectionString** string constant by setting its value to the **PRIMARY CONNECTION STRING** of the **Azure Cosmos DB account** that you recorded earier 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 context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
17. In the open command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate
```

18. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

```
dotnet build
```

19. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.5.3 Task 3: Get SQL database records 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:

```
AdventureWorksSqlContext context = new AdventureWorksSqlContext(sqlDBConnectionString);
```

2. Within the **Main** method, add the following block of code to issue a LINQ query 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 out the number of records imported from **Azure 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 context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
6. In the open command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

```
cd .\AdventureWorks.Migrate
```

7. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

```
dotnet build
```

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

8. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.5.4 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:

   ```
   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 does not 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 does not already existing in the Azure Cosmos DB account with a partition key path of **/Category** and a throughput of **1000 RUs**:

   ```
   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 contained in **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 **Main** method, add the following line of code to print out 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 out 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 context menu or right-click the **Explorer** pane and then select **Open in Terminal**.

11. In the open command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Migrate** folder:

    ```
    cd .\AdventureWorks.Migrate
    ```

12. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

    ```
    dotnet build
    ```

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

13.2.5.5 Task 5: Perform migration

1. In the open command prompt, enter the following command and press Enter to run the .NET Core web application:

   ```
   dotnet run
   ```

Note: The `dotnet run` command will start the console application.

2. Observe the various data that is printed to the screen including; initial SQL record count, individual upsert activity identifiers, final Azure Cosmos DB document count.
3. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.5.6 Task 6: Validate migration

1. Return to the **Microsoft Edge** browser window showing the **Azure portal**.
2. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
3. In the **Resource groups** blade, locate and select the **PolyglotData** resource group that you created earlier in this lab.
4. In the **PolyglotData** blade, select the **polycosmos*** Azure Cosmos DB account that you created earlier in this lab.
5. In the **Azure Cosmos DB account** blade, locate and select the **Data Explorer** link on the left side of the blade.
6. In the **Data Explorer** pane, expand the **Retail** database node.
7. Expand the **Online** container node.
8. Select **New SQL Query**.

   Note: The label for this option may be hidden. You can view labels by hovering over the icons at the top of the **Data Explorer** pane.
9. In the query tab that appears, enter the following text:

   ```
   SELECT * FROM models
   ```
10. Select **Execute Query**.
11. Observe the list of JSON models returned by the query.
12. Back in the query editor, replace the existing text with the following text:

    ```
    SELECT VALUE COUNT(1) FROM models
    ```
13. Select **Execute Query**.
14. Observe the result of the **COUNT** aggregate operation.
15. Return to the **Visual Studio Code** window.

13.2.5.7 Review

In this exercise, you used Entity Framework and the .NET SDK for Azure Cosmos DB to migrate data from Azure SQL Database to Azure Cosmos DB.

13.2.6 Exercise 5: Accessing Azure Cosmos DB using .NET

13.2.6.1 Task 1: Update Library with the Cosmos SDK and references

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

   ```
   cd .\AdventureWorks.Context\
   ```
3. In the command prompt, enter the following command and press Enter to import the **Microsoft.Azure.Cosmos** from NuGet:

   ```
   dotnet add package Microsoft.Azure.Cosmos --version 3.0.0
   ```

   Note: The `dotnet add package` command will add the **Microsoft.Azure.Cosmos** package from **NuGet**.

4. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

   ```
   dotnet build
   ```

5. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.6.2 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 context menu or right-click the **AdventureWorks.Context** folder node and then select **New File**.
3. In the prompt that appears, enter the value **AdventureWorksCosmosContext.cs**.
4. In 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 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 a **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 readonly **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 execute a sql query, get the query result iterator, iterator 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 execute a sql query, get the query result iterator, iterates 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 context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
21. In the open command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Context** folder:

    ```
    cd .\AdventureWorks.Context
    ```

22. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

    ```
    dotnet build
    ```

    Note: If there are any build errors, please review the **AdventureWorksCosmosContext.cs** file located in the **Allfiles (F):\Allfiles\Labs\03\Solution\AdventureWorks\AdventureWorks.Context** folder.

23. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.6.3 Task 3: Update Azure Cosmos DB connection string

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Web** project.
2. Double-click (or double-select) the **appsettings.json** file.
3. In the JSON object, in line 4, locate the **ConnectionStrings.AdventureWorksCosmosContext** path. Observe that the current value is empty:

    ```
    "ConnectionStrings": {
        "AdventureWorksSqlContext": "<sql-connection-string>",
        "AdventureWorksCosmosContext": "",
        "AdventureWorksRedisContext": ""
    },
    ```

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 earier in this lab.
5. Save the **appsettings.json** file.

13.2.6.4 Task 4: Update .NET application startup logic

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Web** project.
2. Double-click (or double-select) the **Startup.cs** file.
3. In the **Startup** class, locate 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 you created earlier in this lab:

```
services.AddScoped<IAdventureWorksProductContext, AdventureWorksCosmosContext>(provider =>
    new AdventureWorksCosmosContext(
        _configuration.GetConnectionString(nameof(AdventureWorksCosmosContext))
    )
);
```

6. Save the **Startup.cs** file.

13.2.6.5 Task 5: Validate .NET application successfully connects to Azure Cosmos DB

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

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

3. In the command prompt, enter the following command and press Enter to run the .NET Core 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, navigate to the your currently running web application (http://localhost:5000).
6. In the web application, observe the list of models displayed on the front page.
7. Locate the **Touring-1000** model and select **View Details**.
8. On 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. Select **Add to Cart**.
9. Observe that the checkout functionality is still disabled.

> **Note**: In the next exercise, you will implement the checkout logic.

10. Close the browser window showing your web application.
11. Return to the **Visual Studio Code** window.
12. Back in the **Visual Studio Code** window, select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.6.6 Review

In this exercise, you wrote C# code to query an Azure Cosmos DB collection using the .NET SDK.

13.2.7 Exercise 6: Accessing Azure Cache for Redis using .NET

13.2.7.1 Task 1: Update Library with the StackExchange.Redis SDK and references

1. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane and then select **Open in Terminal**.

2. In the open command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Context** folder:

   ```
   cd .\AdventureWorks.Context\
   ```

3. In the command prompt, enter the following command and press Enter to import the **Newtonsoft.Json** from NuGet:

   ```
   dotnet add package Newtonsoft.Json --version 12.0.2
   ```

   Note: The `dotnet add package` command will add the **Newtonsoft.Json** package from **NuGet**.

4. In the command prompt, enter the following command and press Enter to import the **StackExchange.Redis** from NuGet:

   ```
   dotnet add package StackExchange.Redis --version 2.0.601
   ```

   Note: The `dotnet add package` command will add the **StackExchange.Redis** package from **NuGet**.

5. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

   ```
   dotnet build
   ```

6. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.7.2 Task 2: Write .NET code to connect to Azure Cache for Redis

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Context** project.
2. Access the context menu or right-click the **AdventureWorks.Context** folder node and then select **New File**.
3. In the prompt that appears, enter the value **AdventureWorksRedisContext.cs**.
4. In the code editor tab for the **AdventureWorksRedisContext.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 **Newtonsoft.Json** namespace from the **Newtonsoft.Json** package imported from NuGet:

   ```
   using Newtonsoft.Json;
   ```

6. Add the following lines of code to import the **StackExchange.Redis** namespace from the **StackExchange.Redis** package imported from NuGet:

   ```
   using StackExchange.Redis;
   ```

7. Add the following lines of code to add **using** directives for 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 add a **AdventureWorks.Context** namespace block:

   ```
   namespace AdventureWorks.Context
   {
   }
   ```

9. Within the **AdventureWorks.Context** namespace, enter the following code to create a new **AdventureWorksRedisContext** class:

```
public class AdventureWorksRedisContext
{
}
```

10. Update the declaration of the **AdventureWorksRedisContext** class by adding a specification indicating that this class will implement the **IAdventureWorksCheckoutContext** interface:

```
public class AdventureWorksRedisContext : IAdventureWorksCheckoutContext
{
}
```

11. Within the **AdventureWorksRedisContext** class, enter the following line of code to create a new readonly **IDatabase** variable named **__database**:

```
private readonly IDatabase _database;
```

12. Within the **AdventureWorksRedisContext** class, add a new constructor with the following signature:

```
public AdventureWorksRedisContext(string connectionString)
{
}
```

13. Within the constructor, add the following block of code to create a new instance of the **ConnectionMultiplexer** class and then get the database instance:

```
ConnectionMultiplexer connection = ConnectionMultiplexer.Connect(connectionString);
_database = connection.GetDatabase();
```

14. Within the **AdventureWorksRedisContext** class, add a new **AddProductToCartAsync** method with the following signature:

```
public async Task AddProductToCartAsync(string uniqueIdentifier, Product product)
{
}
```

15. Within the **AddProductToCartAsync** method, add the following blocks of code to get the current value from a key, create a new list if one does not already exists, add the product to the list, and then store the list as the new value for the key in the database:

```
RedisValue result = await _database.StringGetAsync(uniqueIdentifier);
List<Product> products = new List<Product>();
if (!result.IsNullOrEmpty)
{
    List<Product> parsed = JsonConvert.DeserializeObject<List<Product>>(result.ToString());
    products.AddRange(parsed);
}
products.Add(product);
string json = JsonConvert.SerializeObject(products);
await _database.StringSetAsync(uniqueIdentifier, json);
```

16. Within the **AdventureWorksRedisContext** class, add a new **GetProductsInCartAsync** method with the following signature:

```
public async Task<List<Product>> GetProductsInCartAsync(string uniqueIdentifier)
{
}
```

17. Within the **GetProductsInCartAsync** method, add the following lines of code to get the list from the database and parse the JSON value into a collection of **Product** instances:

```
string json = await _database.StringGetAsync(uniqueIdentifier);
List<Product> products = JsonConvert.DeserializeObject<List<Product>>(json ?? "[]");
return products;
```

18. Within the **AdventureWorksRedisContext** class, add a new **ClearCart** method with the following signature:

```
public async Task ClearCart(string uniqueIdentifier)
{
}
```

19. Within the **ClearCart** method, add the following line of code to remove a key and its associated values from the database:

    ```
    await _database.KeyDeleteAsync(uniqueIdentifier);
    ```

20. Save the **AdventureWorksRedisContext.cs** file.
21. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
22. In the open command prompt, enter the following command and press Enter to switch your terminal context to the **AdventureWorks.Context** folder:

    ```
    cd .\AdventureWorks.Context
    ```

23. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

    ```
    dotnet build
    ```

    Note: If there are any build errors, please review the **AdventureWorksRedisContext.cs** file located in the **Allfiles (F):\Allfiles\Labs\03\Solution\AdventureWorks\AdventureWorks.Context** folder.

24. Select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.7.3 Task 3: Update Redis connection string

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Web** project.
2. Double-click (or double-select) the **appsettings.json** file.
3. In the JSON object, in line 4, locate the **ConnectionStrings.AdventureWorksRedisContext** path. Observe that the current value is empty:

    ```
    "ConnectionStrings": {
        "AdventureWorksSqlContext": "<sql-connection-string>",
        "AdventureWorksCosmosContext": "<cosmos-connection-string>",
        "AdventureWorksRedisContext": ""
    },
    ```

4. Update the value of the **AdventureWorksRedisContext** property by setting its value to the **Primary connection string (StackExchange.Redis)** of the **Azure Cache for Redis** instance that you recorded earier in this lab.
5. In the JSON object, in line 9, locate the **Settings.CartAvailable** path. Observe that the current value is **false**:

    ```
    "Settings": {
        "BlobContainerUrl": "<blob-container-base-uri>",
        "CartAvailable": false,
        "UniqueIdentifier": "az:dev:student"
    }
    ```

6. Update the value of the **CartAvailable** property by setting its value to **true**:

    ```
    "CartAvailable": true,
    ```

7. Save the **appsettings.json** file.

13.2.7.4 Task 4: Update .NET application startup logic

1. In the **Explorer** pane of the Visual Studio Code window, expand the **AdventureWorks.Web** project.
2. Double-click (or double-select) the **Startup.cs** file.
3. In the **Startup** class, locate the existing **ConfigureCheckoutService** method:

    ```
    public void ConfigureCheckoutService(IServiceCollection services)
    {
        services.AddScoped<IAdventureWorksCheckoutContext>(provider =>
            new Mock<IAdventureWorksCheckoutContext>().Object
    ```

```
    );
}
```

Note: The current checkout service uses a mock as its database.

4. Within the **ConfigureCheckoutService** method, delete all existing lines of code :

```
public void ConfigureCheckoutService(IServiceCollection services)
{
}
```

5. Within the **ConfigureCheckoutService** method, add the following block of code to change the checkout provider to the **AdventureWorksRedisContext** implementation you created earlier in this lab:

```
services.AddScoped<IAdventureWorksCheckoutContext, AdventureWorksRedisContext>(provider =>
    new AdventureWorksRedisContext(
        _configuration.GetConnectionString(nameof(AdventureWorksRedisContext))
    )
);
```

6. Save the **Startup.cs** file.

13.2.7.5 Task 5: Validate .NET application successfully connects to Azure Cache for Redis

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

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

3. In the command prompt, enter the following command and press Enter to run the .NET Core 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, navigate to the your currently running web application (http://localhost:5000).
6. In the web application, observe the list of models displayed on the front page.
7. Locate the **Mountain-400-W** model and select **View Details**.
8. On the **Mountain-400-W** product detail page, perform the following actions:
    1. In the **Select options** list, select **Mountain-400-W Silver, 40, $769.49**.
    2. Select **Add to Cart**.
9. On the shopping cart page, observe the contents of the cart and then select **Checkout**.
10. On the checkout page, observe the final receipt.
11. Select the **Shopping Cart** icon at the top of the page.
12. On the shopping cart page, observe the empty cart.
13. Close the browser window showing your web application.
14. Back in the **Visual Studio Code** window, select the **Trash Can** icon to dispose of the currently open terminal and any associated processes.

13.2.7.6 Review

In this exercise, you used C# code to store and retrieve data from an Azure Cache for Redis store.

13.2.8 Exercise 7: Clean up subscription

13.2.8.1 Task 1: Open Azure Cloud Shell

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than symbol and underscore character.

2. If this is your first time opening the **Cloud Shell** by using your subscription, a **Welcome to Azure Cloud Shell Wizard** will appear that allows you to configure **Cloud Shell** for first-time usage. Perform the following actions in the wizard:

   1. A dialog box will appear that prompts you to create a new Storage Account to begin using the shell. Accept the default settings and select **Create storage**.
   2. Wait for the **Cloud Shell** to finish its first-time setup procedures before moving forward with the lab.

      Note: If you do not see the configuration options for the **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. The labs are written from the presumption that you are using a new subscription.

3. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

4. In the prompt, type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

13.2.8.2 Task 2: Delete resource groups

1. In the prompt, type the following command and press Enter to delete the **PolyglotData** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

13.2.8.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Visual Studio Code** application.

13.2.8.4 Review

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

13.4 lab: title: 'Lab: Constructing a polyglot data solution' module: 'AZ-203T03-A: Develop for Azure Storage'

14 Lab: Constructing a polyglot data solution

15 Student lab manual

15.1 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 leverage 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. You have also decided to implement your shopping cart functionality using Azure Cache for Redis. Since the existing application code is modular, you will take the existing application and add new providers for these data services hosted in Azure.

15.2 Objectives

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

- Create instances of various database services using the Azure Portal.
- Write C# code to connect to Azure SQL Database.
- Write C# code to connect to Azure Cosmos DB.
- Write C# code to connect to Azure Cache for Redis.

15.3 Lab setup

- **Estimated Time**: 90 minutes

15.4 Instructions

15.4.1 Before you start

15.4.1.1 Sign in to the lab virtual machine

Ensure that you are signed in to your **Windows 10** virtual machine using the following credentials:

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

15.4.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:

- Microsoft Edge
- File Explorer
- Visual Studio Code

15.4.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T03** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

15.4.2 Exercise 1: Creating database resources in Azure

15.4.2.1 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 will see a dialog box offering a tour of the portal. Select the **Get Started** button to skip the tour.

15.4.2.2 Task 2: Create an Azure Cache for Redis resource

1. Create a new **Azure Cache for Redis** resource with the following details:
   - **DNS name**: polyrediscache[your name in lowercase]
   - **New resource group**: PolyglotData
   - **Location**: East US
   - **Pricing tier**: Basic C0 (250MB Cache)

     Note: Wait for Azure to finish creating the Azure Cache for Redis instance before you move forward with the lab. You will receive a notification when the Redis cache is created.
2. Navigate to the blade for your newly created **Azure Cache for Redis** resource.
3. Open the **Access keys** pane.
4. In the **Access keys** pane, record the value in the **Primary connection string (StackExchange.Redis)** field.

   Note: You will use this value later in this lab.

15.4.2.3 Task 3: Create an Azure SQL server resource

1. Create a new **SQL server** resource with the following details:
   - **Server Name**: polysqlsrvr[your name in lowercase]
   - **Existing resource group**: PolyglotData
   - **Server admin login**: testuser
   - **Password**: TestPa$$w0rd
   - **Location**: East US
   - **Allow Azure Services to access server**: Yes

     Note: Wait for Azure to finish creating the SQL server instance before you move forward with the lab. You will receive a notification when the SQL server is created.

15.4.2.4 Task 4: Create an Azure Cosmos DB account resource

1. Create a new **Azure Cosmos DB** instance with the following details:
   - **Account Name**: polycosmos[your name in lowercase]
   - **Existing resource group**: PolyglotData
   - **API**: Core (SQL)
   - **Location**: East US
   - **Geo-Redundancy**: Disable
   - **Multi-region Writes**: Disable

     Note: Wait for Azure to finish creating the Azure Cosmos DB account before you move forward with the lab. You will receive a notification when the Azure Cosmos DB account is created.
2. Navigate to the blade for your newly created **Azure Cosmos DB account** resource.
3. Open the **Keys** pane.
4. In the **Keys** pane, record the value of the **PRIMARY CONNECTION STRING** field.

   Note: You will use these values later in this lab.

15.4.2.5 Task 5: Create an Azure Storage account resource

1. Create a new **Azure Storage account** with the following details:
   - **Storage account name**: polystor[your name in lowercase]
   - **Existing resource group**: PolyglotData
   - **Account kind**: StorageV2 (general purpose v2)
   - **Location**: East US
   - **Replication**: Locally-redundant storage (LRS)
   - **Performance**: Standard
   - **Access tier (default)**: Hot

     Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You will receive a notification when the storage account is created.

15.4.2.6 Review

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

15.4.3 Exercise 2: Import databases and images

15.4.3.1 Task 1: Upload image blobs

1. Navigate to the blade for the **polystor*** Azure Storage account you created earlier in this lab.
2. Open the **Containers** pane.
3. Create a new container with the following settings:
   - **Name**: images
   - **Public access level**: Blob (anonymous read access for blobs only)
4. Navigate to the new **images** container.
5. Open the **Properties** pane.
6. In the **Properties** pane, record the value in the **URL** field.

   Note: You will use this value later in this lab.
7. Return to the blade for the **images** container.
8. Use the **Upload** button to upload the fourty-two **.jpg** image files located in the **Allfiles (F):\Allfiles\Labs\03\Starte** folder on your lab machine.

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

15.4.3.2 Task 2: Upload SQL .bacpac file

1. Navigate back to the blade for the **polystor*** Azure Storage account.
2. Open the **Containers** pane again.
3. Create a new container with the following settings:
   - **Name**: databases
   - **Public access level**: Private (no anonymous access)
4. Navigate to the new **databases** container.
5. Use the **Upload** button to upload the **AdventureWorks.bacpac** file located in the **Allfiles (F):\Allfiles\Labs\03\Starter** folder on your lab machine.

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

15.4.3.3 Task 3: Import SQL database

1. Navigate to the blade for the **polysqlsrvr*** SQL server resource you created earlier in this lab.
2. Import the database from your Azure Storage account into the SQL server instance using the following details:
   - **Storage account**: polystor*
   - **Database backup blob**: databases\AdventureWorks.bacpac
   - **Database name**: AdventureWorks
   - **Server admin login**: testuser
   - **Password**: TestPa$$w0rd

     Note: Wait for the database to be created before you continue with this lab.

15.4.3.4 Task 4: Use imported SQL Database

1. Navigate back to the blade for the **polysqlsrvr*** 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 **Save** the list.
4. Navigate to the blade for the **AdventureWorks** SQL database resource you recently imported.
5. Open the **Connection strings** pane.
6. Record the value of the **ADO.NET (SQL Authentication)** connection string.
7. Update the connection string you recorded by replacing the placeholder values for `{your_username}` and `{your_password}` with the values **testuser** and **TestPa$$w0rd**, respectively.

   Note: For example, if your connection string was originally `Server=tcp:polysqlsrvrinstructor.database.wind 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=TestPa$$w0rd;`
8. Navigate back to the blade for the **AdventureWorks** SQL database resource.
9. Open the **Query editor** pane.
10. Login using the following credentials:
    - **Username**: testuser
    - **Password**: TestPa$$w0rd
11. Execute the following query and observe the results:

    ```
    SELECT * FROM AdventureWorks.dbo.Models
    ```

    Note: This query will return a list of models that will appear on the homepage of the web application.
12. Execute this additional query and observe the results:

    ```
    SELECT * FROM AdventureWorks.dbo.Products
    ```

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

15.4.3.5 Review

In this exercise, you imported all of the resources you will use with your web application.

15.4.4 Exercise 3: Open and configure a .NET Core web application

15.4.4.1 Task 1: Open and build the web application

1. Using Visual Studio Code, open the solution folder found at: **Allfiles (F):\Allfiles\Labs\03\Starter\AdventureW**
2. Using a terminal, build the .NET Core solution:

```
dotnet build
```

> **Note**: The `dotnet build` command will automatically restore any missing NuGet packages prior to building all projects in the folder.

3. Dispose of the current terminal.

15.4.4.2 Task 2: Update 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 is important that you use your updated connection string here. The original connection string copied from the portal will not have the username and password necessary to connect to the SQL database.

3. Save the **appsettings.json** file.

15.4.4.3 Task 3: Update 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.

15.4.4.4 Task 4: Validate 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 Core 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, navigate to the web application hosted at **localhost** on port **5000**.

   > **Note**: The URL is http://localhost:5000.

5. In the web application, observe the list of models displayed on the front page.
6. Locate the **Water Bottle** model and select **View Details**.
7. On the **Water Bottle** product detail page, select **Add to Cart**.
8. Observe that the checkout functionality is currently disabled.

   > **Note**: For now, only the product page functionality is implemented. You will implement the checkout logic later in this lab.

9. Close the browser window showing your web application.
10. Return to the **Visual Studio Code** window.
11. Dispose of the current terminal.

15.4.4.5 Review

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

15.4.5 Exercise 4: Migrating SQL data to Azure Cosmos DB

15.4.5.1 Task 1: Create migration project

1. Using a terminal, create a new .NET project named **AdventureWorks.Migrate** in a folder with the same name:

   ```
   dotnet new console --name AdventureWorks.Migrate --langVersion 7.1
   ```

   > **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.Model
   ```

   > **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 **2.2.6** of the **Microsoft.EntityFrameworkCore.SqlServer** from NuGet:

   ```
   dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 2.2.6
   ```

   > **Note**: The `dotnet add package` command will add the **Microsoft.EntityFrameworkCore.SqlServer** package from **NuGet**.

6. Using the same terminal, import version **3.7.0** of the **Microsoft.Azure.Cosmos** from NuGet:

   ```
   dotnet add package Microsoft.Azure.Cosmos --version 3.7.0
   ```

   > **Note**: The `dotnet add package` command will add the **Microsoft.Azure.Cosmos** package from **NuGet**.

7. Using the same terminal, build the .NET Core web application:

   ```
   dotnet build
   ```

8. Dispose of the current terminal.

15.4.5.2 Task 2: Create .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 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 is important that you use your updated connection string here. The original connection string copied from the portal will not have the username and password necessary to connect to the SQL database.

6. Update the the **cosmosDBConnectionString** string constant by setting its value to the **PRIMARY CONNECTION STRING** of the **Azure Cosmos DB account** that you recorded earier in this lab.

15.4.5.3 Task 3: Get SQL database records using Entity Framework

1. Within the **Main** method, add the following blocks of code to implement an export of all Model and Product records from Azure SQL Database to local memory:

```
await Console.Out.WriteLineAsync("Start Migration");

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 ASP.NET Core web application project:

```
dotnet build
```

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

5. Dispose of the current terminal.

15.4.5.4 Task 4: Insert items into Azure Cosmos DB

1. Still within the **Main** method, add the following blocks of code to implement an import of the in-memory Model and Product data as documents to Azure Cosmos DB:

```
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 ASP.NET Core web application project:

   ```
   dotnet build
   ```

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

15.4.5.5 Task 5: Perform migration

1. Using the same terminal, run the ASP.NET Core web application project:

   ```
   dotnet run
   ```

   Note: The `dotnet run` command will start the console application.

2. Observe the various data that is printed to the screen including; initial SQL record count, individual upsert activity identifiers, final Azure Cosmos DB document count.
3. Dispose of the current terminal.

15.4.5.6 Task 6: Validate migration

1. Return to the **Microsoft Edge** browser window showing the **Azure portal**.
2. Navigate to the blade for the **polycosmos*** Azure Cosmos DB account 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. Execute the following query and observe the results:

   ```
   SELECT * FROM models
   ```

6. Execute the following query and observe the results:

   ```
   SELECT VALUE COUNT(1) FROM models
   ```

15.4.5.7 Review

In this exercise, you used Entity Framework and the .NET SDK for Azure Cosmos DB to migrate data from Azure SQL Database to Azure Cosmos DB.

15.4.6 Exercise 5: Accessing Azure Cosmos DB using .NET

15.4.6.1 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 the **Microsoft.Azure.Cosmos** from NuGet:

   ```
   dotnet add package Microsoft.Azure.Cosmos --version 3.0.0
   ```

   Note: The `dotnet add package` command will add the **Microsoft.Azure.Cosmos** package from **NuGet**.

3. Using the same terminal, build the ASP.NET Core web application project:

   ```
   dotnet build
   ```

4. Dispose of the current terminal.

15.4.6.2 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 a **AdventureWorks.Context** namespace block:

```
namespace AdventureWorks.Context
{
}
```

4. Create a new **AdventureWorksCosmosContext** class that implements the **IAdventureWorksProductContext** interface with a single readonly **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 executes a 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 executes a 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 Core web application project:

```
dotnet build
```

> **Note**: If there are any build errors, please review the **AdventureWorksCosmosContext.cs** file located in the **Allfiles (F):\Allfiles\Labs\03\Solution\AdventureWorks\AdventureWorks.Context** folder.

12. Dispose of the current terminal.

15.4.6.3 Task 3: Update 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 earier in this lab.
3. Save the **appsettings.json** file.

15.4.6.4 Task 4: Update .NET application startup logic

1. Open the **AdventureWorks.Web/Startup.cs** file in Visual Studio Code.
2. In the **Startup** class, locate the existing **ConfigureProductService** method.

> **Note**: The current product service uses SQL as its database.

3. Replace the **ConfigureProductService** method implementation 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.

15.4.6.5 Task 5: Validate .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 Core 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, navigate to the web application hosted at **localhost** on port **5000**.

   Note: The URL is http://localhost:5000.

5. In the web application, observe the list of models displayed on the front page.
6. Locate the **Touring-1000** model and select **View Details**.
7. On 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. Select **Add to Cart**.
8. Observe that the checkout functionality is still disabled.

   Note: In the next exercise, you will implement the checkout logic.

9. Close the browser window showing your web application.
10. Return to the **Visual Studio Code** window.
11. Dispose of the current terminal.

15.4.6.6 Review

In this exercise, you wrote C# code to query an Azure Cosmos DB collection using the .NET SDK.

15.4.7 Exercise 6: Accessing Azure Cache for Redis using .NET

15.4.7.1 Task 1: Update Library with the StackExchange.Redis SDK and references

1. Using a terminal, change your context to the **AdventureWorks.Context** folder:

   ```
   cd .\AdventureWorks.Context\
   ```

2. Using the same terminal, import the **Newtonsoft.Json** from NuGet:

   ```
   dotnet add package Newtonsoft.Json --version 12.0.2
   ```

   Note: The `dotnet add package` command will add the **Newtonsoft.Json** package from **NuGet**.

3. Using the same terminal, import the **StackExchange.Redis** from NuGet:

   ```
   dotnet add package StackExchange.Redis --version 2.0.601
   ```

Note: The `dotnet add package` command will add the **StackExchange.Redis** package from **NuGet**.

4. Using the same terminal, build the ASP.NET Core web application project:

```
dotnet build
```

5. Dispose of the current terminal.

15.4.7.2 Task 2: Write .NET code to connect to Azure Cache for Redis

1. Create a new **AdventureWorks.Context/AdventureWorksRedisContext.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 Newtonsoft.Json;
using StackExchange.Redis;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

3. Enter the following code to add a **AdventureWorks.Context** namespace block:

```
namespace AdventureWorks.Context
{
}
```

4. Create a new **AdventureWorksRedisContext** class that implements the **IAdventureWorksCheckoutContext** interface with a single readonly **IDatabase** variable:

```
public class AdventureWorksRedisContext : IAdventureWorksCheckoutContext
{
    private readonly IDatabase _database;
}
```

5. Within the **AdventureWorksRedisContext** class, add a new constructor that creates a new instance of the **ConnectionMultiplexer** class and then get the database instance:

```
public AdventureWorksRedisContext(string connectionString)
{
    ConnectionMultiplexer connection = ConnectionMultiplexer.Connect(connectionString);
    _database = connection.GetDatabase();
}
```

6. Within the **AdventureWorksRedisContext** class, add a new **AddProductToCartAsync** method that gets the current value from a key, creates a new list if one does not already exists, adds the product to the list, and then stores the list as the new value for the key in the database:

```
public async Task AddProductToCartAsync(string uniqueIdentifier, Product product)
{
    RedisValue result = await _database.StringGetAsync(uniqueIdentifier);
    List<Product> products = new List<Product>();
    if (!result.IsNullOrEmpty)
    {
        List<Product> parsed = JsonConvert.DeserializeObject<List<Product>>(result.ToString());
        products.AddRange(parsed);
    }
    products.Add(product);
    string json = JsonConvert.SerializeObject(products);
    await _database.StringSetAsync(uniqueIdentifier, json);
}
```

7. Within the **AdventureWorksRedisContext** class, add a new **GetProductsInCartAsync** method that gets the list from the database and parse the JSON value into a collection of **Product** instances:

```
public async Task<List<Product>> GetProductsInCartAsync(string uniqueIdentifier)
{
    string json = await _database.StringGetAsync(uniqueIdentifier);
    List<Product> products = JsonConvert.DeserializeObject<List<Product>>(json ?? "[]");
    return products;
}
```

8. Within the **AdventureWorksRedisContext** class, add a new **ClearCart** method that removes a key and its associated values from the database:

```
public async Task ClearCart(string uniqueIdentifier)
{
    await _database.KeyDeleteAsync(uniqueIdentifier);
}
```

9. Save the **AdventureWorksRedisContext.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 Core web application project:

```
dotnet build
```

> **Note**: If there are any build errors, please review the **AdventureWorksRedisContext.cs** file located in the **Allfiles (F):\Allfiles\Labs\03\Solution\AdventureWorks\AdventureWorks.Context** folder.

12. Dispose of the current terminal.

15.4.7.3 Task 3: Update Redis connection string

1. Open the **AdventureWorks.Web/appsettings.json** file in Visual Studio Code.
2. Replace the value of the `ConnectionStrings.AdventureWorksRedisContext` property with the **Primary connection string (StackExchange.Redis)** of the **Azure Cache for Redis** instance that you recorded earier in this lab.
3. Replace the existing **false** value of the `Settings.CartAvailable` property with a new value of **true**.
4. Save the **appsettings.json** file.

15.4.7.4 Task 4: Update .NET application startup logic

1. Open the **AdventureWorks.Web/Startup.cs** file in Visual Studio Code.
2. In the **Startup** class, locate the existing **ConfigureCheckoutService** method.

> **Note**: The current checkout service doesn't use any database.

3. Replace the **ConfigureCheckoutService** method implementation with the following code:

```
public void ConfigureCheckoutService(IServiceCollection services)
{
    services.AddScoped<IAdventureWorksCheckoutContext, AdventureWorksRedisContext>(provider =>
        new AdventureWorksRedisContext(
            _configuration.GetConnectionString(nameof(AdventureWorksRedisContext))
        )
    );
}
```

4. Save the **Startup.cs** file.

15.4.7.5 Task 5: Validate .NET application successfully connects to Azure Cache for Redis

1. Using a terminal, change your context to the **AdventureWorks.Web** folder:

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

2. Using the same terminal, run the ASP.NET Core 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, observe the list of models displayed on the front page.
5. Locate the **Mountain-400-W** model and select **View Details**.
6. On the **Mountain-400-W** product detail page, perform the following actions:
    1. In the **Select options** list, select **Mountain-400-W Silver, 40, $769.49**.
    2. Select **Add to Cart**.
7. On the shopping cart page, observe the contents of the cart and then select **Checkout**.
8. On the checkout page, observe the final receipt.
9. Select the **Shopping Cart** icon at the top of the page.
10. On the shopping cart page, observe the empty cart.
11. Close the browser window showing your web application.
12. Return to the **Visual Studio Code** window.
13. Dispose of the current terminal.

15.4.7.6 Review

In this exercise, you used C# code to store and retrieve data from an Azure Cache for Redis store.

15.4.8 Exercise 7: Clean up subscription

15.4.8.1 Task 1: Open Azure Cloud Shell

1. At the top of the Azure portal, click the **Cloud Shell** icon to open a new shell instance.
2. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
3. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press Enter to list all resource groups in the subscription:

    ```
    az group list
    ```

4. Type the following command and press Enter to view a list of possible commands to delete a resource group:

    ```
    az group delete --help
    ```

15.4.8.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **PolyglotData** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

15.4.8.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Visual Studio Code** application.

15.4.8.4 Review

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

15.6 lab: title: 'Lab: Access resource secrets securely across services' type: 'Answer Key' module: 'AZ-203T04-A: Implement Azure security'

16 Lab: Access resource secrets securely across services

17 Student lab answer key

17.1 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 up.

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

17.2 Instructions

17.2.1 Before you start

17.2.1.1 Sign in to the lab virtual machine

Sign in to your **Windows 10** virtual machine 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.

17.2.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:

- Microsoft Edge
- File Explorer

17.2.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T04** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

17.2.2 Exercise 1: Create Azure resources

17.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure portal** (portal.azure.com).

3. Enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. Select **Sign in**.

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

17.2.2.2 Task 2: Create an Azure Storage account

1. In the navigation pane on the left side of the Azure portal, select **All services**.
2. In the **All services** blade, select **Storage Accounts**.
3. In the **Storage accounts** blade, view your list of Storage instances.
4. At the top of the **Storage accounts** blade, select **+ Add**.
5. In the **Create storage account** blade, observe the tabs at the top of the blade, such as **Basics**.

   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.

1. In 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 **SecureFunction**, and then select **OK**.
   3. In the **Storage account name** text box, enter **securestor**[**your name in lowercase**].
   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. In the **Access tier** section, ensure that **Hot** is selected.
   9. Select **Review + Create**.
2. In the **Review + Create** tab, review the options that you selected during the previous steps.
3. Select **Create** to create the storage account by using your specified configuration.
4. Wait for the creation task to complete before you move forward with this lab.
5. In the navigation pane on the left side of the Azure portal, select **All services**.
6. In the **All services** blade, select **Storage Accounts**.
7. In the **Storage accounts** blade, select the storage account instance with the prefix **securestor**.
8. In the **Storage account** blade, locate the **Settings** section on the left side of the blade and select the **Access keys** link.
9. In the **Access keys** blade, select any one of the keys and record the value in either of the **Connection string** fields. You will use this value later in this lab.

   Note: It does not matter which connection string you choose to use. They are interchangeable.

17.2.2.3 Task 3: Create an Azure Key Vault

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Vault** and then press Enter.
4. In the **Everything** search results blade, select the **Key Vault** result.

5. In the **Key Vault** blade, select **Create**.
6. In the **Create key vault** blade, observe the tabs at the top of the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new **key vault**. At any time, you can select **Review + create** to skip the remaining tabs.

1. In the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, **Use existing**, and then select **SecureFunction** from the list.
   3. In the **Key vault name** text box, enter **securevault**[**your name in lowercase**].
   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**.
2. In the **Review + Create** tab, review the options that you selected during the previous steps.
3. Select **Create** to create the key vault by using your specified configuration.
4. Wait for the creation task to complete before you move forward with this lab.

17.2.2.4 Task 4: Create an Azure Function app

1. On the navigation menu located on the left side of the portal, select the **+ Create a resource** link.
2. At the top of the **New** blade, locate the **Search the Marketplace** text box above the list of featured services.
3. In the search text box, enter **Function** and then press Enter.
4. In the **Everything** search results blade, select the **Function App** result.
5. In the **Function App** blade, select **Create**.
6. In the **Function App** blade, observe the tabs at the top of the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new **function app**. At any time, you can select **Review + create** to skip the remaining tabs.

1. In the **Basics** tab, perform the following actions:
   1. Leave the **Subscription** text box set to its default value.
   2. In the **Resource group** section, **Use existing**, and then select **SecureFunction** from the list.
   3. In the **Function app name** text box, enter **securefunc**[**your name in lowercase**].
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET Core**.
   6. In the **Region** drop-down list, select the **East US** region.
   7. Select **Next: Hosting**.
2. In the **Hosting** tab, perform the following actions:
   1. In the **Storage account** drop-down list, select the **securestor*** Storage account that you created earlier in this lab.
   2. In the **Operating System** section, select **Windows**.
   3. In the **Plan type** drop-down list, select the **Consumption** option.
   4. Select **Next: Monitoring**.
3. In the **Monitoring** tab, perform the following actions:
   1. In the **Enable Application Insights** section, select **No**.
   2. Select **Review + Create**.

4. In the **Review + Create** tab, review the options that you selected during the previous steps.
5. Select **Create** to create the function app by using your specified configuration.
6. Wait for the creation task to complete before you move forward with this lab.

17.2.2.5 Review

In this exercise, you created all the resources that you will use for this lab.

17.2.3 Exercise 2: Configure secrets and identities

17.2.3.1 Task 1: Configure a system-assigned managed service identity

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc*** function app that you created earlier in this lab.
4. In the **Function Apps** blade, select the **Platform features** tab.
5. In the **Platform features** tab, select the **Identity** link located in the **Networking** section.
6. In the **Identity** blade, locate the **System assigned** tab and then perform the following actions:
    1. In the **Status** section, select **On**.
    2. Select **Save**.
    3. Select **Yes** in the confirmation dialog.
7. Wait for the system-assigned managed identity to be created before you move forward with this lab.

17.2.3.2 Task 2: Create a Key Vault secret

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securevault*** Key Vault that you created earlier in this lab.
4. In the **Key Vault** blade, select the **Secrets** link located in the **Settings** section.
5. In the **Secrets** pane, select **+ Generate/Import**.
6. In 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**.
    8. Select **Create**.
7. Wait for the secret to be created before you move forward with this lab.
8. Back in the **Secrets** pane, select the **storagecredentials** item in the list.
9. In the **Versions** pane, select the latest version of the **storagecredentials** secret.
10. In the **Secret Version** pane, perform the following actions.
    1. Observe the metadata for the latest version of the secret.

2. Select **Show secret value** to view the value of the secret.
3. Record the value of the **Secret Identifier** text box because you will use this later in the lab.

   Note: You are recording the value of the **Secret Identifier** field, not the **Secret Value** field.

17.2.3.3 Task 3: Configure a Key Vault access policy

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securevault*** Key Vault that you created earlier in this lab.
4. In the **Key Vault** blade, select the **Access policies** link located in the **Settings** section.
5. In the **Access policies** pane, select **+ Add Access Policy**.
6. In the **Add access policy** blade, perform the following actions:
   1. Select the **Select principal** link.
   2. In the **Principal** blade, locate and select the service principal named **securefunc[your name in lowercase]**, and then select **Select**.
   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**.
8. Wait for your changes to the access policies to be saved before you move forward with this lab.

17.2.3.4 Review

In this exercise, you created a server-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 will use within your function app.

17.2.4 Exercise 3: Write function app code

17.2.4.1 Task 1: Create a Key Vault-derived application setting

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc*** function app that you created earlier in this lab.
4. In the **Function Apps** blade, select the **Platform features** tab.
5. In the **Platform features** tab, select the **Configuration** link located in the **General Settings** section.
6. In the **Configuration** section, perform the following actions:
   1. Select the **Application settings** tab.
   2. Select **+ New application setting**.
   3. In the **Add/Edit application setting** popup that appears, in the **Name** field, enter **StorageConnectionString**.
   4. In the **Value** field, construct a value by using the following syntax: **@Microsoft.KeyVault(SecretUri=<Secret Identifier>)**

Note: You will 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/storagecreden**
then your value would be **@Microsoft.KeyVault(SecretUri=https://securevaultstudent.vault.azure.net/**

1. Leave the **deployment slot setting** field set to its default value.
2. Select **OK** to close the popup and return to the **Configuration** section.
3. Select **Save** at the top of the blade to persist your settings.

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

17.2.4.2 Task 2: Create a HTTP-triggered function

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc*** function app that you created earlier in this lab.
4. In the **Function App** blade, click + next to the **Functions** drop-down.
5. In the **New Azure Function** quickstart, perform the following actions:
   1. Under the **Choose a Development Environment** header, select **In-Portal**.
   2. Select **Continue**.
   3. Under the **Choose a Function** header, select **More templates...**.
   4. Select **Finish and view templates**.
   5. In the list of templates, select **HTTP trigger**.
   6. In the **New Function** pop-up, locate the **Name** text box and enter **FileParser**.
   7. In the **New Function** pop-up, locate the **Authorization level** list and select **Anonymous**.
   8. In the **New Function** pop-up, select **Create**.
6. In the function editor, observe the example function script:

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpRequest req, 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;

    return name != null
        ? (ActionResult)new OkObjectResult($"Hello, {name}")
        : new BadRequestObjectResult("Please pass a name on the query string or in the request bod
}
```

7. **Delete** all the example code.
8. Within the function editor, copy and paste the following placeholder function:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;

public static async Task<IActionResult> Run(HttpRequest req)
{
    return new OkObjectResult("Test Successful");
}
```

9. Select **Save and run** to save the script and perform a test execution of the function.
10. The **Test** and **Logs** panes will automatically appear when the script executes for the first time.
11. Observe the **Output** text box in the **Test** pane. You should now see the value **Test Successful** returned from the function.

17.2.4.3 Task 3: Test the Key Vault-derived application setting

1. Delete the existing code within the **Run** method of the script.
2. The **Run** method should now look like this:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;

public static async Task<IActionResult> Run(HttpRequest req)
{

}
```

3. Add the following line of code to get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

```
string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
```

4. Add the following line of code to return the value of the **connectionString** variable by using the **OkObjectResult** class constructor:

```
return new OkObjectResult(connectionString);
```

5. The **Run** method should now look like this:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;

public static async Task<IActionResult> Run(HttpRequest req)
{
    string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
    return new OkObjectResult(connectionString);
}
```

6. Select **Save and run** to save the script and perform a test execution of the function.
7. Observe the **Output** text box in the **Test** pane. You should now see the connection string returned from the function.

17.2.4.4 Review

In this exercise, you securely used a service identity to read the value of a secret stored in **Azure Key Vault** and return that value as the result of an **Azure Function**.

17.2.5 Exercise 4: Access Storage Account blobs

17.2.5.1 Task 1: Upload a sample storage blob

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.

3. In the **SecureFunction** blade, select the **securestor*** storage account that you created earlier in this lab.
4. In the **Storage account** blade, select the **Containers** link located in the **Blob service** section on the left side of 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 **drop**.
    2. In the **Public access level** drop-down list, select **Blob (anonymous read access for blobs only)**.
    3. Select **OK**.
7. Back in the **Containers** section, select the newly created **drop** container.
8. In 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 dialog box, go to **Allfiles (F):\Allfiles\Labs\04\Starter**, select the **records.json** file, and then select **Open**.
    3. Ensure that **Overwrite if files already exist** is selected.
    4. Select **Upload**.
10. Wait for the blob to be uploaded before you continue with this lab.
11. Back in the **Container** blade, select the **records.json** blob from the list of blobs.
12. In the **Blob** blade, view the blob metadata.
13. Copy the **URL** for the blob.
14. On the taskbar, right-select the **Microsoft Edge** icon and then select **New window**.
15. In the new browser window, navigate to the **URL** that you copied for the blob.
16. You should now see the **JSON** contents of the blob. Close the browser window showing the **JSON** contents.
17. Return to the browser window with the **Azure portal.**
18. Close the **Blob** blade.
19. Back in the **Container** blade, select **Change access level policy** located at the top of the blade.
20. In the **Change access level** pop-up that appears, perform the following actions:
    1. In the **Public access level** drop-down list, select **Private (no anonymous access)**.
    2. Select **OK**.
21. On the taskbar, right-select the **Microsoft Edge** icon and then select **New window**.
22. In the new browser window, navigate to the **URL** that you copied for the blob.
23. You should now see an error message indicating that the resource was not found.

    Note: If you do not see the error message, your browser might have cached the file. Use **Ctrl+F5** to refresh the page until you see the error message.

17.2.5.2 Task 2: Pull the Storage Account SDK from NuGet

1. On the navigation menu located on the left side of the portal, select the **Resource groups** link.
2. In the **Resource groups** blade, locate and select the **SecureFunction** resource group that you created earlier in this lab.
3. In the **SecureFunction** blade, select the **securefunc*** function app that you created earlier in this lab.

4. In the **Function App** blade, locate and select the existing **FileParser** function to open the editor for the function.

   Note: You might need to expand the **Functions** option in the menu on the left side of the blade.

5. On the right side of the editor, select **View files** to open the tab.
6. In the **View files** tab, select **Add**.
7. In the filename dialog that appears, enter **function.proj** and press Enter (displays an empty code editor).
8. In the file editor, insert this configuration content:

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="Azure.Storage.Blobs" Version="12.0.0" />
    </ItemGroup>
</Project>
```

9. In the editor, select **Save** button to persist your changes to the configuration.

   Note: This **.proj** file contains the NuGet package reference necessary to import the Azure.Storage.Blobs package.

10. Select the **run.csx** file to return to the editor for the **FileParser** function.
11. Minimize the **View files** tab.

    Note: You can minimize the tab by selecting the arrow immediately to the right of the tab header.

12. Within the editor, delete the existing code within the **Run** method of the script.
13. At the top of the code file, add the following line of code to create a **using** directive for the **Azure.Storage** namespace:

```
using Azure.Storage;
```

14. At the top of the code file, add the following line of code to create a **using** directive for the **Azure.Storage.Blobs** namespace:

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

15. Add the following line of code to create a **using** directive for the **Azure.Storage.Blobs.Models** namespace:

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

16. The **Run** method should now look like this:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;

public static async Task<IActionResult> Run(HttpRequest req)
{

}
```

17.2.5.3 Task 3: Write storage account code

1. Add the following line of code within the **Run** method to get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

```
string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
```

2. Add the following line of code to create a new instance of the **BlobServiceClient** class by passing in your *connectionString* variable to the constructor:

```
BlobServiceClient serviceClient = new BlobServiceClient(connectionString);
```

3. Add the following line of code to use the **BlobServiceClient.GetBlobContainerClient** method, while passing in the **drop** container name to create a new instance of the **BlobContainerClient** class that references the container that you created earlier in this lab:

```
BlobContainerClient containerClient = serviceClient.GetBlobContainerClient("drop");
```

4. Add the following line of code to use the **BlobContainerClient.GetBlobClient** method, while passing in the **records.json** blob name to create a new instance of the **BlobClient** class that references the blob that you uploaded earlier in this lab:

```
BlobClient blobClient = containerClient.GetBlobClient("records.json");
```

5. The **Run** method should now look like this:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;

public static async Task<IActionResult> Run(HttpRequest req)
{
    string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
    BlobServiceClient serviceClient = new BlobServiceClient(connectionString);
    BlobContainerClient containerClient = serviceClient.GetBlobContainerClient("drop");
    BlobClient blobClient = containerClient.GetBlobClient("records.json");
}
```

17.2.5.4 Task 4: Download a blob

1. Add the following line of code 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 blobClient.DownloadAsync();
```

2. Add the following line of code to return the various content stored in the *content* variable by using the **FileStreamResult** class constructor:

```
return new FileStreamResult(response?.Value?.Content, response?.Value?.ContentType);
```

3. The **Run** method should now look like this:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;

public static async Task<IActionResult> Run(HttpRequest req)
{
    string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
    BlobServiceClient serviceClient = new BlobServiceClient(connectionString);
    BlobContainerClient containerClient = serviceClient.GetBlobContainerClient("drop");
    BlobClient blobClient = containerClient.GetBlobClient("records.json");
    var response = await blobClient.DownloadAsync();
    return new FileStreamResult(response?.Value?.Content, response?.Value?.ContentType);
}
```

4. Select **Save and run** to save the script and perform a test execution of the function.

5. Observe the **Output** text box in the **Test** pane. You should now see the content of the **$/drop/records.json** blob stored in your **storage account**.

17.2.5.5 Review

In this exercise, you used C# code to access a Storage Account securely and then download the contents of a blob.

17.2.6 Exercise 5: Clean up subscription

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

1. In the top navigation bar 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 symbol and underscore character.

2. If this is your first time opening the **Cloud Shell** by using your subscription, a **Welcome to Azure Cloud Shell Wizard** will appear that allows you to configure **Cloud Shell** for first-time usage. Perform the following actions in the wizard:

   1. A dialog box will appear that prompts you to create a new Storage Account to begin using the shell. Accept the default settings and select **Create storage**.
   2. Wait for the **Cloud Shell** to finish its first-time setup procedures before moving forward with the lab.

      Note: If you do not see the configuration options for the **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. The labs are written from the presumption that you are using a new subscription.

3. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

4. In the prompt, type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

17.2.6.2 Task 2: Delete resource group

1. In the prompt, type the following command and press Enter to delete the **SecureFunction** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

17.2.6.3 Task 3: Close active application

Close the currently running **Microsoft Edge** application.

17.2.6.4 Review

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

17.4 lab: title: 'Lab: Access resource secrets securely across services' module: 'AZ-203T04-A: Implement Azure security'

18 Lab: Access resource secrets securely across services

19 Student lab manual

19.1 Lab scenario

Your company has a data-sharing business-to-business (B2B) agreement with another local business in which you are expected to parse a file that is 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 are now tasked with devising a way to securely access the file and generate a secure URL that can be used by any internal system to access the

blob without exposing the file to the internet. You have decided to use Microsoft Azure Key Vault to store the credentials for the storage account and Azure Functions to write the code necessary to access the file securely without storing credentials in plain text or exposing the file to the internet.

19.2 Objectives

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

- Create an Azure Key Vault and store secrets in the Key Vault.
- Create a server-assigned managed identity for an Azure App Service instance.
- Create an Azure Key Vault access policy for an Azure Active Directory identity or application.
- Use the Azure Storage .NET software development kit (SDK) to securely download a blob.

19.3 Lab setup

- **Estimated Time**: 45 minutes

19.4 Instructions

19.4.1 Before you start

19.4.1.1 Sign in to the lab virtual machine

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

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

19.4.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:

- Microsoft Edge
- File Explorer

19.4.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T04** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

19.4.2 Exercise 1: Create Azure resources

19.4.2.1 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 will see a dialog box offering a tour of the portal. Click **Get Started** to skip the tour.

19.4.2.2 Task 2: Create an Azure Storage account

1. Create a new **storage account** with the following details:
   - **New resource group**: SecureFunction
   - **Name**: securestor[your name in lowercase]
   - **Location**: East US
   - **Performance**: Standard
   - **Account kind**: StorageV2 (general purpose v2)
   - **Replication**: Locally redundant storage (LRS)
   - **Access tier**: Hot

   Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You will receive a notification when the account is created.

1. Open the **Access Keys** blade of your newly created **storage account** instance.
2. Record the value in the **Connection string** field. You will use this value later in this lab.

19.4.2.3 Task 3: Create an Azure Key Vault

1. Create a new **Key Vault** with the following details:
   - **Existing resource group**: SecureFunction
   - **Name**: securevault[your name in lowercase]
   - **Region**: East US
   - **Pricing tier**: Standard

     Note: Wait for Azure to finish creating the Key Vault before you move forward with the lab. You will receive a notification when the vault is created.

19.4.2.4 Task 4: Create an Azure Functions app

1. Create a new **function app** with the following details:
   - **Existing resource group**: SecureFunction
   - **App name**: securefunc[your name in lowercase]
   - **Publish**: Code
   - **Runtime Stack**: .NET Core
   - **Region**: East US
   - **Storage account**: securestor[your name in lowercase here]
   - **Operating System**: Windows
   - **Plan**: Consumption
   - **Enable Application Insights**: No

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

19.4.2.5 Review

In this exercise, you created all the resources that you will use for this lab.

19.4.3 Exercise 2: Configure secrets and identities

19.4.3.1 Task 1: Configure a system-assigned managed service identity

1. Access the **securefunc*** function app that you created earlier in this lab.
2. Navigate to the **Identity** settings located in the **Platform features** tab.
3. Enable the **system-assigned** managed identity and save your changes.

19.4.3.2 Task 2: Create a Key Vault secret

1. Access the **securevault*** Key Vault that you created earlier in this lab.
2. Navigate to the **Secrets** link located 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. Click through the secret to view the metadata for its latest version.
5. Record the value of the **Secret Identifier** field because you will use this later in the lab.

19.4.3.3 Task 3: Configure a Key Vault access policy

1. Access the **securevault*** Key Vault that you created earlier in this lab.
2. Navigate to the **Access Policies** link located in the **Settings** section.
3. Create a new **access policy** with the following settings:
   - **Principal**: securefunc[your name in lowercase]
   - **Key permissions**: none
   - **Secret permissions**: GET
   - **Certificate permissions**: none
   - **Authorized application**: none
4. **Save** your changes to the list of **Access Policies**.

19.4.3.4 Review

In this exercise, you created a server-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 will use within your function app.

19.4.4 Exercise 3: Write function app code

19.4.4.1 Task 1: Create a Key Vault-derived application setting

1. Access the **securefunc*** *Function App* that you created earlier in this lab.
2. Navigate to the **Configuration** settings located in the **Platform features** tab.
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 will 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/storag** then your value would be **@Microsoft.KeyVault(SecretUri=https://securevaultstudent.vault.azure.**

4. Save your changes to the **Application settings**.

19.4.4.2 Task 2: Create a HTTP-triggered function

1. Access the **securefunc*** function app that you created earlier in this lab.
2. Create a new **function** by using the following settings:
   - **Development Environment**: In-portal
   - **Template**: HTTP trigger
   - **Name**: FileParser
   - **Authorization level**: Anonymous
3. In the function editor, replace the example function script with the following placeholder C# code:

```
using System.Net;
using Microsoft.AspNetCore.Mvc;

public static async Task<IActionResult> Run(HttpRequest req)
{
    return new OkObjectResult("Test Successful");
}
```

4. Click **Save and run** to perform a test execution of the function. The output from the execution should be **Test Successful**.

19.4.4.3 Task 3: Test the Key Vault-derived application setting

1. Delete all the existing code within the **Run** method.
2. Get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

```
string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
```

3. Return the value of the **connectionString** variable by using the **OkObjectResult** class constructor:

```
return new OkObjectResult(connectionString);
```

4. Click **Save and run** to perform a test execution of the function. The output from the execution should be your **storage account** connection string stored in **Azure Key Vault**.

19.4.4.4 Review

In this exercise, you securely used a service identity to read the value of a secret stored in **Azure Key Vault** and return that value as the result of an **Azure Function**.

19.4.5 Exercise 4: Access Storage Account blobs

19.4.5.1 Task 1: Upload a sample Storage blob

1. Access the **securestor*** storage account that you created earlier in this lab.
2. Navigate to the **Containers** link located in the **Blob service** section.
3. Create a new **container** with the following settings:
   - **Name**: drop
   - **Public access level**: Blob (anonymous read access for blobs only)
4. Navigate to the new **drop** container.
5. Click **Upload** to upload the **records.json** file located in the **Allfiles (F): \Allfiles\Labs\04\Starter** folder on your lab machine.

   Note: We recommend that you enable the **Overwrite if files already exist** option.
6. View the metadata for the **records.json** blob by clicking on the blob entry in the list of blobs.

7. Using a new browser tab, navigate to the **URL** for the blob and view the blob's contents.
8. Update the container's **access level** by changing the **Public access level** to **Private (no anonymous access)**.
9. Using a new browser window or tab, navigate to the **URL** for the blob and view the blob's contents. You should now see an error message indicating that the resource was not found.

   Note: If you do not see the error message, your browser might have cached the file. Use **Ctrl+F5** to refresh the page until you see the error message.

19.4.5.2 Task 2: Pull the Storage Account SDK from NuGet

1. Access the **securefunc*** function app that you created earlier in this lab.
2. Open the editor for the **FileParser** function.
3. Using the **View files** tab, create a new **function.proj** file with the following content:

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <TargetFramework>netstandard2.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="Azure.Storage.Blobs" Version="12.0.0" />
    </ItemGroup>
</Project>
```

4. **Save** the newly created **function.proj** file.

   Note: This **.proj** file contains the NuGet package reference necessary to import the Azure.Storage.Blobs package.

5. View the contents of the **function.proj** file by clicking the file in the **View files** tab.
6. Return to the editor for the **FileParser** function by clicking the **run.csx** file in the **View files** tab.
7. Add two **using** directives for the **Azure.Storage**, **Azure.Storage.Blobs** and the **Azure.Storage.Blobs.Models** namespaces.
8. Delete all the existing code within the **Run** method.

19.4.5.3 Task 3: Write Storage Account code

1. Get the value of the **StorageConnectionString** application setting by using the **Environment.GetEnvironmentVariable** method:

```
string connectionString = Environment.GetEnvironmentVariable("StorageConnectionString");
```

2. Create a new instance of the **BlobServiceClient** class by by passing in your *connectionString* variable to the constructor:

```
BlobServiceClient serviceClient = new BlobServiceClient(connectionString);
```

3. Use the **BlobServiceClient.GetBlobContainerClient** method while passing in the **drop** container name to create a new instance of the **BlobContainerClient** class that references the container that you created earlier in this lab:

```
BlobContainerClient containerClient = serviceClient.GetBlobContainerClient("drop");
```

4. Use the **BlobContainerClient.GetBlobClient** method while passing in the **records.json** blob name to create a new instance of the **BlobClient** class that references the blob that you uploaded earlier in this lab:

```
BlobClient blobClient = containerClient.GetBlobClient("records.json");
```

19.4.5.4 Task 4: Download a blob

1. 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 blobClient.DownloadAsync();
```

2. 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);
   ```

3. Click **Save and run** to perform a test execution of the function. The output from the execution should be the content of the **$/drop/records.json** blob stored in your Storage Account.

19.4.5.5 Review

In this exercise, you used C# code to access a Storage Account securely and then download the contents of a blob.

19.4.6 Exercise 5: Clean up subscription

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

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
3. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

4. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

19.4.6.2 Task 2: Delete a resource group

1. Type the following command and press Enter to delete the **SecureFunction** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

19.4.6.3 Task 3: Close active application

Close the currently running **Microsoft Edge** application.

19.4.6.4 Review

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

19.6 lab: title: 'Lab: Monitoring services deployed to Azure' type: 'Answer Key' module: 'AZ-203T05-A: Monitor, troubleshoot, and optimize Azure solutions '

20 Lab: Monitoring services deployed to Azure

21 Student lab answer key

21.1 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 steps to not match up.

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

21.2 Instructions

21.2.1 Before you start

21.2.1.1 Sign in to the lab virtual machine

Sign in to your **Windows 10** virtual machine 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.

21.2.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:

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

21.2.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T05** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

21.2.2 Exercise 1: Create and configure Azure resources

21.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure portal** (portal.azure.com).
3. At the sign-in page, enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. 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.

21.2.2.2 Task 2: Create an Application Insights resource

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **Insights** and press Enter.
4. In the **Everything** search results blade, select the **Application Insights** result.

5. In the **Application Insights** blade, select **Create**.
6. In the second **Application Insights** blade, locate the tabs in the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new **Application Insights instance**. At any time, you can select **Review + create** to skip the remaining tabs.
7. In 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[your name in lowercase]**.
   4. In the **Location** drop-down list, select the **(US) East US** region.
   5. Select **Review + Create**.
8. In 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.
10. Wait for the creation task to complete before you move forward with this lab.
11. In the left navigation pane of the portal, select **Resource groups**.
12. In the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
13. In the **MonitoredAssets** blade, select the **instrm*** Application Insights account that you created earlier in this lab.
14. In the **Application Insights** blade, on the left side of the blade, within the **Configure** category„ select the **Properties** link.
15. In the **Properties** section, observe the value of the **Instrumentation Key** field. This key is used by client applications to connect to Application Insights.

21.2.2.3 Task 3: Create an Web App resource

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **Web** and press Enter.
4. In the **Everything** search results blade, select the **Web App** result.
5. In the **Web App** blade, select **Create**.
6. In the second **Web App** blade, locate the tabs in the blade, such as **Basics**.

   Note: Each tab represents a step in the workflow to create a new **Web App**. At any time, you can select **Review + create** to skip the remaining tabs.
7. In 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[your name in lowercase]**.
   4. In the **Publish** section, select **Code**.
   5. In the **Runtime stack** drop-down list, select **.NET Core 3.0 (current)**.
   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. In 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*** Application Insights account that you created earlier in this lab.
    3. Select **Review + Create**.
9. In 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.
11. Wait for the creation task to complete before you move forward with this lab.
12. In the left navigation pane of the portal, select **Resource groups**.
13. In the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
14. In the **MonitoredAssets** blade, select the **smpapi*** Web App you that created earlier in this lab.
15. In the **App Service** blade, on the left side of the blade, within the **Settings** category, select the **Configuration** link.
16. In the **Configuration** section, perform the following actions:
    1. Select the **Application settings** tab.
    2. Select **Show Values** to view the secrets associated with your API.
    3. Observe the value corresponding to the **APPINSIGHTS_INSTRUMENTATIONKEY** key. This value was set automatically when you built your Web App resource.
17. In the **App Service** blade, on the left side of the blade within the **Settings** category, select the **Properties** link.
18. In the **Properties** section, record the value of the **URL** field. You will use this value later in the lab to make requests against the API.

21.2.2.4 Task 4: Configure Web App auto-scale options

1. In the **App Service** blade, on the left side of the blade, within 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** field, 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** field within the **Instance limits** section, enter **2**.
    6. In the **Maximum** field within the **Instance limits** section, enter **8**.
    7. In the **Default** field within the **Instance limits** section, enter **3**.
    8. Select **+ Add a rule**. In the **Scale rule** popup that appears, leave all fields set to their default values and then select **Add**.
    9. At the top of the section, select **Save**.
3. Wait for the save operation to complete before you move forward with this lab.

21.2.2.5 Review

In this exercise, you created the resources that you will use for the remainder of the lab.

21.2.3 Exercise 2: Build and deploy an .NET Core Web API application

21.2.3.1 Task 1: Build a .NET Core Web API project

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\05\Starter\Api**, and then select **Select Folder**.
4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to create a new .NET Core Web API application named **SimpleApi** in the current directory:

   ```
   dotnet new webapi --output . --name SimpleApi
   ```

6. In the command prompt, enter the following command and press Enter to add the **2.8.2** version of the **Microsoft.ApplicationInsights.AspNetCore** package from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights.AspNetCore --version 2.8.2
   ```

7. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

   ```
   dotnet build
   ```

21.2.3.2 Task 2: Update application code to disable HTTPS and use Application Insights

1. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Startup.cs** file to open the file in the editor.
2. In the editor, in the **Startup** class, locate and delete the following line of code at line **43**:

   ```
   app.UseHttpsRedirection();
   ```

   Note: This line of code forces the Web App to use HTTPS. For this lab, this is unnecessary.

3. Within the **Startup** class, add a new **static string constant** named **INSTRUMENTATION_KEY** with its value set to the **Instrumentation Key** 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 you **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. Locate the **ConfigureServices** method within the **Startup** class:

   ```
   public void ConfigureServices(IServiceCollection services)
   {
       services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
   }
   ```

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. Your **ConfigureServices** method should now look like this:

   ```
   public void ConfigureServices(IServiceCollection services)
   {
       services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
       services.AddApplicationInsightsTelemetry(INSTRUMENTATION_KEY);
   }
   ```

7. **Save** the **Startup.cs** file.
8. Locate the command prompt at the bottom of the screen. In the command prompt, enter the following command and press Enter to build the .NET Core web application.

```
dotnet build
```

21.2.3.3 Task 3: Test an API application locally

1. Locate the command prompt at the bottom of the screen. In the command prompt, enter the following command and press Enter to execute the .NET Core web application.

   ```
   dotnet run
   ```

2. On the taskbar, select the **Microsoft Edge** icon.
3. In the open browser window, navigate to the **/api/values** relative path of your test application hosted at **localhost** on port **5000**.

   Note: The full URL is http://localhost:5000/api/values

4. In the same browser window, navigate to the **/api/values/7** relative path of your test application hosted at **localhost** on port **5000**.

   Note: The full URL is http://localhost:5000/api/values/7

5. Close the browser window displaying the http://localhost:5000/api/values/7 address.
6. Close the currently running **Visual Studio Code** application.

21.2.3.4 Task 4: View metrics in Application Insights

1. Return to your currently open browser window displaying the **Azure portal**.
2. On the left side of the portal, select **Resource groups**.
3. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
4. In the **MonitoredAssets** blade, select the **instrm*** Application Insights account that you created earlier in this lab.
5. In the **Application Insights** blade, in the tiles located in the center of the blade, observe the metrics displayed. Specifically, observe the number of **server requests** that have occurred and the average **server response time**.

   Note: It can take up to five minutes for the requests to show within the Application Insights metrics charts.

21.2.3.5 Task 5: Deploy an application to Web App

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\Starter\Api**, and then select **Select Folder**.
4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane, and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to sign in to the Azure CLI:

   ```
   az login
   ```

6. In the browser window that appears, perform the following actions:
   1. Enter the **email address** for your Microsoft account.
   2. Select **Next**.
   3. Enter the **password** for your Microsoft account.
   4. Select **Sign in**.
7. Return to the currently open **command prompt** application. Wait for the sign-in process to finish.
8. At the command prompt, enter the following command and press Enter to list all the **apps** in your **MonitoredAssets** resource group:

```
az webapp list --resource-group MonitoredAssets
```

9. Enter the following command and press 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 press Enter to print out 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
```

11. Enter the following command and press Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\05\Starter** directory that contains the deployment files:

```
cd F:\Allfiles\Labs\05\Starter\
```

12. Enter the following command and press 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.

13. Wait for the deployment to complete before you move forward with this lab.
14. Close the currently running **Visual Studio Code** application.
15. In the left navigation pane of the portal, select **Resource groups**.
16. In the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
17. In the **MonitoredAssets** blade, select the **smpapi*** Web App that you created earlier in this lab.
18. In the **App Service** blade, select **Browse** at the top of the blade.
19. A new browser window or tab will open and return a **404 (Not Found)** error. In the browser address bar, update the URL by appending the suffix **/api/values** to the end of the current URL and then press Enter.

> **Note**: For example, if your URL is http://smpapistudent.azurewebsites.net, the new URL would be http://smpapistudent.azurewebsites.net/api/values.

20. Observe the JSON array that is returned as a result of using the API.

21.2.3.6 Review

In this exercise, you created an API by using ASP.NET Core and configured it to stream application metrics to Application Insights. You then used the Application Insights dashboard to view performance details about your Web App and the API running in the app.

21.2.4 Exercise 3: Build a client application by using .NET Core

21.2.4.1 Task 1: Build a .NET Core console project

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Starter\Console**, and then select **Select Folder**.
4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane, and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to create a new .NET Core console application named **SimpleConsole** in the current directory:

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

6. In the command prompt, enter the following command and press Enter to add the **7.1.0** version of the **Polly** package from NuGet to the current project:

```
dotnet add package Polly --version 7.1.0
```

7. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

```
dotnet build
```

21.2.4.2 Task 2: Add HTTP client code

1. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Program.cs** file to open the file in the editor.
2. In the editor, add the following **using** directive for the **System.Net.Http** namespace:

```
using System.Net.Http;
```

3. In the editor, add the following **using** directive for the **System.Threading.Tasks** namespace:

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

4. In the **SimpleConsole** namespace, locate the following class at line **7**:

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

5. Replace the entire **Program** class with the following implementation:

```
class Program
{
    private const string _api = "";
    private static HttpClient _client = new HttpClient(){ BaseAddress = new Uri(_api) };

    static void Main(string[] args)
    {
        Run().Wait();
    }

    static async Task Run()
    {

    }
}
```

6. Locate the **__api** constant at line **9**:

```
private const string _api = "";
```

7. Update the **__api** constant by setting the value of the variable to the **URL** of the Web App you recorded earlier in this lab:

> **Note**: For example, if your URL is http://smpapistudent.azurewebsites.net, the new line of code will be: private const string _api = "http://smpapistudent.azurewebsites.net";

8. Within the **Run** method, add the following line of code to asynchronously invoke the **HttpClient.GetStringAsync** method passing in a string for the relative path of **/api/values/**:

```
string response = await _client.GetStringAsync("/api/values/");
```

9. Within the **Run** method, add an additional line of code to write out the response from the **GET** request to the console:

```
Console.WriteLine(response);
```

10. Your **Program.cs** file should now have the following code:

```
using System;
using System.Net.Http;
using System.Threading.Tasks;

namespace SimpleConsole
{
    class Program
    {
        private const string _api = "http://<your-api-name>.azurewebsites.net/";
        private static HttpClient _client = new HttpClient(){ BaseAddress = new Uri(_api) };

        static void Main(string[] args)
        {
            Run().Wait();
        }

        static async Task Run()
        {
            string response = await _client.GetStringAsync("/api/values/");
            Console.WriteLine(response);
        }
    }
}
```

11. **Save** the **Program.cs** file.

21.2.4.3 Task 3: Test a console application locally

1. At the bottom of the screen, in the command prompt, enter the following command and press Enter to execute the .NET Core web application.

   ```
   dotnet run
   ```

2. Observe that the application successfully invokes the Web App in Azure and returns the same JSON array that you observed earlier in this lab. Your result should appear similar to the following JSON content:

   ```
   ["value1","value2"]
   ```

3. Return to your currently open browser window displaying the **Azure portal**.
4. On the left side of the portal, select **Resource groups**.
5. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
6. In the **MonitoredAssets** blade, select the **smpapi*** Web App that you created earlier in this lab.
7. In the **App Service** blade, select **Stop** at the top of the blade to halt the execution of the Web App.
8. In the **Stop web app** confirmation dialog box, select **Yes**.
9. On the taskbar, select the **Visual Studio Code** icon.
10. On the **File** menu, select **Open Folder**.
11. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\05\Starter\Console**, and then select **Select Folder**.
12. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane, and then select **Open in Terminal**.
13. In the open command prompt, enter the following command and press Enter to execute the .NET Core web application.

    ```
    dotnet run
    ```

14. Observe that the application execution fails and displays a **HttpRequestException** message that is similar to the following exception message:

```
System.Net.Http.HttpRequestException: Response status code does not indicate

success: 403 (Site Disabled).

at System.Net.Http.HttpResponseMessage.EnsureSuccessStatusCode()

at System.Net.Http.HttpClient.GetStringAsyncCore(Task\`1 getTask)

at SimpleConsole.Program.Run() in F:\Allfiles\Labs\05\Starter\Console\Program.cs:line 20
```

Note: This exception occurs because the Web App is no longer available.

21.2.4.4 Task 4: Add retry logic by using Polly

1. On the left side of the **Visual Studio Code** window, in the **Explorer** pane double-click the **PollyHandler.cs** file to open the file in the editor.
2. Within the **PollyHandler** class, observe lines **13-24**. These lines of code use the **Polly** library from the **.NET Foundation** to create a retry policy that will retry a failed HTTP request every five minutes.
3. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Program.cs** file to open the file in the editor.
4. Locate the **__client** constant at line **10**:

   ```
   private static HttpClient _client = new HttpClient(){ BaseAddress = new Uri(_api) };
   ```

5. Update the **__client** constant by updating the **HttpClient** constructor to use a new instance of the **PollyHandler** class:

   ```
   private static HttpClient _client = new HttpClient(new PollyHandler()){ BaseAddress = new Uri(_api
   ```

6. **Save** the **Program.cs** file.

21.2.4.5 Task 5: Validate retry logic

1. At the bottom of the screen, in the command prompt, enter the following command and press Enter to execute the .NET Core web application.

   ```
   dotnet run
   ```

2. Observe that the HTTP request execution continues to fail and is re-attempted every five seconds. You will observe the following message in the console while the application is failing:

   ```
   Failed Attempt
   ```

3. Leave the console application running. It will attempt to access the Web App infinitely until it is successful.
4. Return to your currently open browser window displaying the **Azure portal**.
5. On the left side of the portal, select **Resource groups**.
6. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
7. In the **MonitoredAssets** blade, select the **smpapi*** Web App that you created earlier in this lab.
8. In the **App Service** blade, select **Start** at the top of the blade to resume the Web App.
9. Return to the currently running **Visual Studio Code** application.
10. Observe that the application finally successfully invokes the Web App in Azure and returns the same JSON array that you observed earlier in this lab. Your result should resemble the following JSON content:

    ```
    ["value1","value2"]
    ```

11. Close the currently running **Visual Studio Code** application.

21.2.4.6 Review

In this exercise, you created a console application to access your API by using conditional retry logic. The application continued to work regardless of whether the Web App was available.

21.2.5 Exercise 4: Load test Web App

21.2.5.1 Task 1: Run a performance test on an Web App

1. Return to your currently open browser window displaying the **Azure portal**.
2. On the left side of the portal, select **Resource groups**.
3. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
4. In the **MonitoredAssets** blade, select the **smpapi*** Web App that you created earlier in this lab.
5. In the **App Service** blade, on the left side of the blade within the **Development Tools** category, select **Performance test**.
6. In the **Performance test** section, at the top of the section, select **New**.
7. In the **New performance test** blade, perform the following actions:
    1. In the **Name** field, enter **LoadTest**.
    2. In the **Generate Load From** list, select **East US (Web app Location)**.
    3. In the **User Load** field, enter **1000**.
    4. In the **Duration** field, enter **10**.
    5. Select **Configure Test Using**.
8. In the **Configure test using** blade, perform the following actions:
    1. In the **Test Type** list, select **Manual Test**.
    2. In the **URL** field, update the URL by appending the suffix **/api/values** to the end of the current URL.

        Note: For example, if your URL is http://smpapistudent.azurewebsites.net, the new URL would be http://smpapistudent.azurewebsites.net/api/values.
    3. Select **Done**.
9. Back in the **New performance test** blade, select **Run test**.
10. Back in the **Performance test** section, in the list of **Recent runs,** select **LoadTest**.
11. In the **LoadTest** blade, wait for the test to start before you proceed with the lab.

    Note: Most load tests take about 10 to 15 minutes to gather the resources and start. You can wait at this blade because it will automatically refresh when the load testing is started.
12. Wait for the load test to finish before you proceed with the lab. Observe the live chart updating as your Web App experiences increased usage.

    Note: The load test will take the 10 minutes you specified in the previous steps of the lab.

21.2.5.2 Task 2: Use Azure Monitor metrics after performance test

1. In the left navigation pane of the Azure portal, select **All services**.
2. In the **All services** blade, select **Monitor**.
3. In the **Monitor** blade, on the left side of the blade, select **Metrics**.
4. In the **Metrics** section, perform the following actions:
    1. In the **Resource** section, select **Select a scope**.
    2. In the **Select a resource** window that appears, in the **Resource group** list, select **MonitoredAssets**. Then, in the **Resource** list, select the **instrm*** Application Insights account option. Finally, select **Apply** to close the window and confirm your selection.
    3. In the **Metric Namespace** list, in the **Standard** category, select **Standard metrics (preview)**.
    4. In the **Metric** list, in the **Performance Counter** category, select **Process CPU**.
    5. In the **Aggregation** list, select **Avg**.

6. At the top of the section, select **Last 24 hours (Automatic - 5 minutes)**. In the window that appears, in the **Time range** category, select **Last 30 minutes** and then select **Apply** to save your selection.
7. At the top of the section, select **Line chart**. In the menu that appears, select **Area chart**.

5. Observe your newly created chart.
6. At the top of the section, select **Add metric**.
7. In the **Metrics** section, perform the following actions with the new metric item in the list:
    1. In the **Metric Namespace** list, in the **Standard** category, select **Log-based metrics**.
    2. In the **Metric** list, in the **Server** category, select **Server response time**.
    3. In the **Aggregation** list, select **Avg**.
8. Observe your updated chart. Observe the information displayed in your chart. You can observe how the server response time correlates with the CPU time as load on the application increased.

21.2.5.3 Review

In this exercise, you performed a performance (load) test of your Web App by using the tools available to you in Azure. After you performed the load test, you were able to measure your API app's behavior by using metrics in the Azure Monitor interface.

21.2.6 Exercise 5: Clean up subscription

21.2.6.1 Task 1: Open Cloud Shell

1. At the top of 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 symbol and underscore character.

2. If this is your first time opening the **Cloud Shell** by using your subscription, a **Welcome to Azure Cloud Shell Wizard** will appear that allows you to configure **Cloud Shell** for first-time usage. Perform the following actions in the wizard:
    1. A dialog box will appear that prompts you to create a new Storage Account to begin using the shell. Accept the default settings and select **Create storage**.
    2. Wait for the **Cloud Shell** to finish its first-time setup procedures before moving forward with the lab.

        Note: If you do not see the configuration options for the **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. The labs are written from the presumption that you are using a new subscription.
3. At the bottom of the portal in the **Cloud Shell** command prompt, type the following command and press Enter to list all resource groups in the subscription:

    ```
    az group list
    ```
4. Type the following command and press Enter to view a list of possible commands to delete a resource group:

    ```
    az group delete --help
    ```

21.2.6.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **MonitoredAssets** resource group:

    ```
    az group delete --name MonitoredAssets --no-wait --yes
    ```
2. Close the **Cloud Shell** pane at the bottom of the portal.

21.2.6.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Visual Studio Code** application.

21.2.6.4 Review

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

21.4 lab: title: 'Lab: Monitoring services deployed to Azure' module: 'AZ-203T05-A: Monitor, troubleshoot, and optimize Azure solutions '

22 Lab: Monitoring services deployed to Azure

23 Student lab manual

23.1 Lab scenario

You have created an API for your next big startup venture that needs to get to market quickly. 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. In this project, you will deploy an API to the App Service by using API Apps, capture telemetry and metrics by using Application Insights, and implement a smart client that can handle network issues or other transient faults. You will then load test the API by using Azure DevOps.

23.2 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 Core web application and an Azure Web App resource.
- Use .NET Foundation libraries to implement a retry policy over a service that could have transient faults.
- Performance test an Web App by using Azure DevOps.

23.3 Lab setup

- **Estimated time:** 75 minutes

23.4 Instructions

23.4.1 Before you start

23.4.1.1 Sign in to the lab virtual machine

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

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

23.4.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:

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

23.4.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T05** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

23.4.2 Exercise 1: Create and configure Azure resources

23.4.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure portal** (portal.azure.com).
3. At the sign-in page, enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. 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.

23.4.2.2 Task 2: Create an Application Insights resource

1. Create a new **Application Insights account** with the following details:
   - **New resource group**: MonitoredAssets
   - **Name**: instrm[your name in lowercase]
   - **Location**: (US) East US

   Note: Wait for Azure to finish creating the storage account before you move forward with the lab. You will receive a notification when the account is created.
2. Access the **Properties** section of the **Application Insights** blade.
3. Observe the value of the **Instrumentation Key** field. This key is used by client applications to connect to Application Insights.

23.4.2.3 Task 3: Create an Web App resource

1. Create a new **Web App** with the following details:
   - **Existing resource group**: MonitoredAssets
   - **Web App name**: smpapi[your name in lowercase]
   - **Publish**: Code
   - **Runtime stack**: .NET Core 3.0
   - **Operating System**: Windows
   - **Region**: East US
   - **New App Service plan**: MonitoredPlan
   - **Sku and size**: Standard (S1)

- **Application Insights**: Enabled
- **Existing Application Insights resource**: instrm[your name in lowercase]

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

2. Access the **smpapi*** *Web App* that you created earlier in this lab.
3. In the **Settings** section on the left side of the blade, navigate to the **Configuration** section.
4. Locate and access the **Application Settings** tab within the **Configuration** section.
5. Observe the value corresponding to the **APPINSIGHTS_INSTRUMENTATIONKEY** Application Settings key. This value was set automatically when you built your Web App resource.
6. Access the **Properties** section of the **App Service** blade.
7. Record the value of the **URL** field. You will use this value later in the lab to make requests against the API.

23.4.2.4 Task 4: Configure Web App auto-scale 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:
   1. **Name**: ComputeScaler
   2. **Scale mode** section, select **Scale based on a metric**
   3. **Minimum instances**: 2
   4. **Maximum instances**: 8
   5. **Default instances**: 3
   6. **Scale rules**: Single scale-out rule with default values
3. **Save** your changes to the **autoscale** configuration.

23.4.2.5 Review

In this exercise, you created the resources that you will use for the remainder of the lab.

23.4.3 Exercise 2: Build and deploy an .NET Core Web API application

23.4.3.1 Task 1: Build a .NET Core Web API project

1. Open **Visual Studio Code**.
2. In **Visual Studio Code**, open the **Allfiles (F):\Allfiles\Labs\05\Starter\Api** folder.
3. Use the **Explorer** to open a new terminal that has the context set to the current working directory.
4. In the command prompt, create a new .NET Core Web API application named **SimpleApi** in the current directory:

   ```
   dotnet new webapi --output . --name SimpleApi
   ```

5. Add the **2.7.1** version of the **Microsoft.ApplicationInsights.AspNetCore** package from NuGet to the current project:

   ```
   dotnet add package Microsoft.ApplicationInsights.AspNetCore --version 2.7.1
   ```

6. Build the .NET Core web application:

   ```
   dotnet build
   ```

23.4.3.2 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. Locate and delete the following line of code at line **43**:

   ```
   app.UseHttpsRedirection();
   ```

   Note: This line of code forces the Web App to use HTTPS. For this lab, this is unnecessary.
3. Within the **Startup** class, add a new **static string constant** named **INSTRUMENTATION_KEY** with its value set to the **Instrumentation Key** 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 you **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 within the **ConfigureServices** method to configure Application Insights using the provided instrumentation key:

   ```
   services.AddApplicationInsightsTelemetry(INSTRUMENTATION_KEY);
   ```
5. **Save** the **Startup.cs** file.
6. Use the **Explorer** to open a new terminal, if it is not still already open, that context is set to the current working directory.
7. Build the .NET Core web application:

   ```
   dotnet build
   ```

23.4.3.3 Task 3: Test an API application locally

1. Use the **Explorer** to open a new terminal, if it is not still already open, that has the context set to the current working directory.
2. Execute the .NET Core web application.

   ```
   dotnet run
   ```
3. Open the **Microsoft Edge** browser.
4. In the open browser window, navigate to the **/api/values** relative path of your test application hosted at **localhost** on port **5000**.

   Note: The full URL is http://localhost:5000/api/values.
5. In the same browser window, navigate to the **/api/values/7** relative path of your test application hosted at **localhost** on port **5000**.

   Note: The full URL is http://localhost:5000/api/values/7
6. Close the browser window that you recently opened.
7. Close the currently running **Visual Studio Code** application.

23.4.3.4 Task 4: View metrics in Application Insights

1. Return to your currently open browser window displaying the **Azure portal**.
2. Access the **instrm*** Application Insights account that you created earlier in this lab.
3. In the **Application Insights** blade, observe the metrics displayed in the tiles located in the center of the blade. Specifically, observe the number of **server requests** that have occurred and the average **server response time**.

   Note: It can take up to five minutes for the requests to show within the Application Insights metrics charts.

23.4.3.5 Task 5: Deploy an application to Web App

1. Open **Visual Studio Code**.
2. In **Visual Studio Code**, open the **Allfiles (F):\Allfiles\Labs\05\Starter\Api** folder.
3. Use the **Explorer** to open a new terminal that has the context set to the current working directory.
4. Sign in to the Azure CLI by using your Microsoft 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\05\Starter** directory that contains the lab files:

   ```
   cd F:\Labfiles\05\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. Access the **smpapi*** Web App that you created earlier in this lab.
11. Open the **smpapi*** Web App in your browser.
12. Perform a **GET** request to the **/api/values/** relative path of the website and observe the JSON array that is 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/api/values.

23.4.3.6 Review

In this exercise, you created an API by using ASP.NET Core and configured it to stream application metrics to Application Insights. You then used the Application Insights dashboard to view performance details about your API.

23.4.4 Exercise 3: Build a client application by using .NET Core

23.4.4.1 Task 1: Build a .NET Core console project

1. Open **Visual Studio Code**.
2. In **Visual Studio Code**, open the **Allfiles (F):\Allfiles\Labs\05\Starter\Console** folder.
3. Use the **Explorer** to open a new terminal that has the context set to the current working directory.
4. In the command prompt, create a new .NET Core console application named **SimpleConsole** in the current directory:

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

5. Add the **7.1.0** version of the **Polly** package from NuGet to the current project:

   ```
   dotnet add package Polly --version 7.1.0
   ```

6. Build the .NET Core web application:

   ```
   dotnet build
   ```

23.4.4.2 Task 2: Add HTTP client code

1. Use the **Explorer** in **Visual Studio Code** to open the **Program.cs** file in the editor.
2. Add **using** directives to the top of the file for the following namespaces:
   - **System.Net.Http**
   - **System.Threading.Tasks**

```
using System.Net.Http;
using System.Threading.Tasks;
```

3. Locate the **Program** class at line **7**:

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

4. Replace the entire **Program** class with the following implementation:

```
class Program
{
    private const string _api = "";
    private static HttpClient _client = new HttpClient(){ BaseAddress = new Uri(_api) };

    static void Main(string[] args)
    {
        Run().Wait();
    }

    static async Task Run()
    {

    }
}
```

5. Locate the **__api** constant at line **9**:

```
private const string _api = "";
```

6. Update the **__api** constant by setting the value of the variable to the **URL** of the Web App that you recorded earlier in this lab:

   Note: For example, if your URL is http://smpapistudent.azurewebsites.net, the new line of code will be: private const string _api = "http://smpapistudent.azurewebsites.net";

1. Within the **Run** method, add the following two lines of code to asynchronously invoke the **HttpClient.GetStringAsync** method passing in a string for the relative path of **/api/values/**, and then write out the response:

```
string response = await _client.GetStringAsync("/api/values/");
Console.WriteLine(response);
```

2. **Save** the **Program.cs** file.

23.4.4.3 Task 3: Test a console application locally

1. Use the **Explorer** to open a new terminal, if it is not still already open, that has the context set to the current working directory.
2. Execute the .NET Core web application.

```
dotnet run
```

3. Observe that the application successfully invokes the Web App in Azure and returns the same JSON array that you observed earlier in this lab:

   ```
   ["value1","value2"]
   ```

4. Return to your currently open browser window displaying the **Azure portal**.
5. Access the **smpapi*** Web App that you created earlier in this lab.
6. In the **App Service** blade, select **Stop** to halt the execution of the Web App.
7. Open **Visual Studio Code**.
8. In **Visual Studio Code**, open the **Allfiles (F):\Allfiles\Labs\05\Starter\Console** folder.
9. Use the **Explorer** to open a new terminal that has the context set to the current working directory.
10. In the command prompt, execute the .NET Core web application.

    ```
    dotnet run
    ```

11. Observe that the application fails and displays a **HttpRequestException** message that is similar to the following exception message:

    ```
    System.Net.Http.HttpRequestException: Response status code does not indicate
    success: 403 (Site Disabled).
       at System.Net.Http.HttpResponseMessage.EnsureSuccessStatusCode()
       at System.Net.Http.HttpClient.GetStringAsyncCore(Task`1 getTask)
       at SimpleConsole.Program.Run() in F:\Labfiles\05\Starter\Console\Program.cs:line 20
    ```

    Note: This exception occurs because the Web App is no longer available.

23.4.4.4 Task 4: Add retry logic by using Polly

1. Use the **Explorer** in **Visual Studio Code** to open the **PollyHandler.cs** file in the editor.
2. Within the **PollyHandler** class, observe lines **13-24**. These lines of code use the **Polly** library from the **.NET Foundation** to create a retry policy that will retry a failed HTTP request every five minutes.
3. Use the **Explorer** in **Visual Studio Code** to open the **Program.cs** file in the editor.
4. Locate the **__client** constant at line **10**:

   ```
   private static HttpClient _client = new HttpClient(){ BaseAddress = new Uri(_api) };
   ```

5. Update the **__client** constant by updating the **HttpClient** constructor to use a new instance of the **PollyHandler** class:

   ```
   private static HttpClient _client = new HttpClient(new PollyHandler()){ BaseAddress = new Uri(_api
   ```

6. **Save** the **Program.cs** file.

23.4.4.5 Task 5: Validate retry logic

1. Use the **Explorer** to open a new terminal, if it is not still already open, that has the context set to the current working directory.
2. Execute the .NET Core web application.

   ```
   dotnet run
   ```

3. Observe that the HTTP request execution continues to fail and is being re-attempted every five seconds. Leave the application running. It will attempt to access the Web App infinitely until it is successful.
4. Return to your currently open browser window displaying the **Azure portal**.
5. Access the **smpapi*** Web App that you created earlier in this lab.
6. In the **App Service** blade, select **Start** to resume the Web App.
7. Return to the currently running **Visual Studio Code** application.
8. Observe that the application finally successfully invokes the Web App in Azure and returns the same JSON array that you observed earlier in this lab.

9. Close the currently running **Visual Studio Code** application.

23.4.4.6 Review

In this exercise, you created a console application to access your API by using conditional retry logic. The application continued to work regardless of whether the API was available.

23.4.5 Exercise 4: Load a test Web App

23.4.5.1 Task 1: Run a performance test on an Web App

1. Return to your currently open browser window displaying the **Azure portal**.
2. Access the **smpapi*** Web App that you created earlier in this lab.
3. In the **App Service** blade, select the **Performance test** link.
4. Create a new **Performance test** by using the following details:
   - **Name**: LoadTest
   - **Generate Load From**: East US (Web app Location)
   - **User Load**: 1000
   - **Duration**: 10
   - **Test Type**: Manual Test
   - **URL**: http://<your-api-name>.azurewebsites.net/api/values
5. In the **LoadTest** blade, wait for the test to start and complete before proceeding with the lab. Observe the live chart updating as your Web App experiences increased usage.

   Note: Most load tests take about 10 to 15 minutes to gather the resources and start. You can wait at this blade because it will automatically refresh when the load testing is started. The load test will then take the 10 minutes that you specified in the previous steps of this lab.

23.4.5.2 Task 2: Use Azure Monitor metrics after the performance test

1. Navigate to the **Azure Monitor** service.
2. In the **Monitor** blade, select the **Metrics** link.
3. In the **Metrics** section, create a new chart with the following details:
   - **Resource**: instrm* Application Insights account created earlier in this lab
   - **Time range**: Last 30 minutes (Automatic)
   - **Chart type**: Area chart
4. Create a new metric with the following details:
   - **Metric Namespace**: Standard metrics
   - **Metric**: Process CPU
   - **Aggregation**: Avg
5. Create another new metric with the following details:
   - **Metric Namespace**: Log-based metrics
   - **Metric**: Server response time
   - **Aggregation**: Avg
6. Observe the information displayed in your chart. You can observe how the server response time correlates with the CPU time as load on the application increased.

23.4.5.3 Review

In this exercise, you performed a performance (load) test of your Web App by using the tools available to you in Azure. After you performed the load test, you were able to measure your API app's behavior by using metrics in the Azure Monitor interface.

23.4.6 Exercise 5: Clean up subscription

23.4.6.1 Task 1: Open Cloud Shell

1. At the top of the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
3. At the bottom of the portal in the **Cloud Shell** command prompt, type the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

4. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

23.4.6.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **MonitoredAssets** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

23.4.6.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Visual Studio Code** application.

23.4.6.4 Review

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

23.6 lab: title: 'Lab: Creating a multi-tier solution by using services in Azure' type: 'Answer Key' module: 'AZ-203T06-A: Connect to and consume Azure, and third-party, services'

24 Lab: Creating a multi-tier solution by using services in Azure

25 Student lab answer key

25.1 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 steps to not match up.

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

25.2 Instructions

25.2.1 Before you start

25.2.1.1 Sign in to the lab virtual machine

Sign in to your **Windows 10** virtual machine 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.

25.2.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications you will use in this lab:

- Microsoft Edge
- File Explorer
- Microsoft Azure Storage Explorer

25.2.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T06** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

25.2.2 Exercise 1: Creating an Azure Cognitive Search service in the portal

25.2.2.1 Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure portal** (portal.azure.com).
3. At the sign-in page, enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. 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.

25.2.2.2 Task 2: Create API Management resource

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **API** and press Enter.
4. In the **Everything** search results blade, select the **API Management** result.
5. In the **API Management** blade, select **Create**.

6. In the **API Management Service** blade, perform the following actions:
    1. In the **Name** field, enter **prodapi**[***your name in lowercase***].
    2. Leave the **Subscription** field set to its default value.
    3. In the **Resource group** section, select **Create new**, in the pop-up field enter **MultiTierService**, and then select **OK**.
    4. In the **Location** list, select **West US**.
    5. In the **Organization name** field, enter **Contoso**.
    6. Leave the **Administrator email** field set to its default value.
    7. In the **Pricing tier** list, select **Consumption (99.9 SLA, %)**.
    8. Select **Create**.
7. Wait for the creation task to complete before you move forward with this lab.

25.2.2.3 Task 3: Create an Azure Cognitive Search account

1. In the left navigation pane of the portal, select **+ Create a resource**.

    Note: If you cannot find the link, the Create a resource icon is a plus-sign character located on the left side of the portal.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter the text **Search** and press Enter.
4. In the **Everything** search results blade, select the **Azure Cognitive Search** result.
5. In the **Azure Cognitive Search** blade, select **Create**.
6. In the **New Search Service** blade, observe the tabs at the top of the blade, such as **Basics**.

    Note: Each tab represents a step in the workflow to create a new **search account**. At any time, you can select **Review + create** to skip the remaining tabs.
7. In the **Basics** tab, perform the following actions:
    2. Leave the **Subscription** field set to its default value.
    3. In the **Resource group** list, select **MultiTierService**.
    4. In the **URL** field, enter the value **prodsearch**[***your name in lowercase***].
    5. In the **Location** list, select **East US**.
    6. Select the **Pricing tier** link. In the **Pricing tier** blade, select **Basic** and then select **Select**.
    7. Select **Review + Create**.
8. In the **Review + Create** tab, review the options that you selected during the previous steps.
9. Select **Create** to create the search account by using your specified configuration.
10. Wait for the creation task to complete before you move forward with this lab.
11. In the left navigation pane of the portal, select **Resource groups**.
12. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
13. In the **MultiTierService** blade, select the **prodsearch*** Search service that you created earlier in this lab.
14. In the **Search Service** blade, in the **Settings** section, select the **Keys** link.
15. In the **Keys** section, select any one of the keys and record the value. You will use this value later in the lab.

    Note: It does not matter which connection string you choose to use. They are interchangeable.

25.2.2.4 Task 4: Create an index

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
3. In the **MultiTierService** blade, select the **prodsearch*** Search service that you created earlier in this lab.
4. In the **Search Service** blade, select **Add index**.
5. In the **Add index** blade, perform the following actions:
    1. In the **Index name** field, enter **retail**.
    2. In the **Key** list, select **id**.
    3. Leave the **Suggester name** field blank.
    4. Leave the **Search mode** list blank.
6. Within the **Add index** blade, a list of fields is displayed. Perform the following actions to configure the **id** field:
    1. In the **Field Name** field, observe the hard-coded value of **id**.
    2. In the **Type** list, observe the hard-coded option of **Edm.String**.
    3. In the **Retrievable** option, observe that it is hard-coded to **true**.
    4. Leave **Filterable** unselected.
    5. Select **Sortable**.
    6. Leave **Facetable** unselected.
    7. Leave **Searchable** unselected.
7. Within the **Add index** blade, perform the following actions to configure the new **name** field:
    1. Select + **Add field**.
    2. In the **Field Name** field, enter **name**.
    3. In the **Type** list, select **Edm.String**.
    4. Select **Retrievable**.
    5. Leave **Filterable** unselected.
    6. Select **Sortable**.
    7. Leave **Facetable** unselected.
    8. Select **Searchable**.
    9. In the **Analyzer** list, select **Standard - Lucene**.
8. Within the **Add index** blade, perform the following actions to configure the new **price** field:
    1. Select + **Add field**.
    2. In the **Field Name** field, enter **price**.
    3. In the **Type** list, select **Edm.Double**.
    4. Select **Retrievable**.
    5. Select **Filterable**.
    6. Select **Sortable**.
    7. Select **Facetable**.
9. Within the **Add index** blade, perform the following actions to configure the new **quantity** field:
    1. Select + **Add field**.
    2. In the **Field Name** field, enter **quantity**.

3. In the **Type** list, select **Edm.Int32**.
4. Select **Retrievable**.
5. Select **Filterable**.
6. Select **Sortable**.
7. Select **Facetable**.

10. Within the **Add index** blade, perform the following actions to configure the new **manufacturer** field:
    1. Select **+ Add field**.
    2. In the **Field Name** field, enter **manufacturer**.
    3. In the **Type** list, select **Edm.String**.
    4. Select **Retrievable**.
    5. Select **Filterable**.
    6. Select **Sortable**.
    7. Select **Facetable**.
    8. Leave **Searchable** unselected.
11. Within the **Add index** blade, select **Create**.

25.2.2.5 Review

In this exercise, you created a new Azure Cognitive Search account and built an index within the account.

25.2.3 Exercise 2: Index an Azure Storage table in Azure Cognitive Search

25.2.3.1 Task 1: Create an Azure Storage account

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **Storage** and press Enter.
4. In the **Everything** search results blade, select the **Storage Account** result.
5. In the **Storage Account** blade, select **Create**.
6. In the **Create Storage Account** blade, observe the tabs at the top of the blade.

   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.
7. In the **Basics** tab, perform the following actions:
    1. Leave the **Subscription** field set to its default value.
    2. In the **Resource group** list, select **MultiTierService**.
    3. In the **Storage Account Name** field, enter **prodstorage**[***your name in lowercase***].
    4. In the **Location** list, select **(US) East US**.
    5. In the **Performance** section, select **Standard**.
    6. In the **Account kind** list, select **StorageV2 (general purpose v2)** .
    7. In the **Replication** list, select **Read-access geo-redundant storage (RA-GRS)**.
    8. In the **Access tier** section, ensure that **Hot** is selected.
    9. Select **Review + Create**.
8. In the **Review + Create** tab, review the options that you entered in the previous steps.
9. Select **Create** to create the Storage account by using your specified configuration.
10. Wait for the creation task to complete before you move forward with this lab.

11. In the left navigation pane of the portal, select **Resource groups**.
12. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
13. In the **MultiTierService** blade, select the **prodstorage*** storage account that you created earlier in this lab.
14. In the **Storage Account** blade, on the left side of the blade, in the **Settings** section, select the **Access keys** link.
15. In the **Access keys** blade, select any one of the keys and record the value of either of the **Connection string** fields. You will use this value later in this lab.

    Note: It does not matter which connection string you choose. They are interchangeable.

25.2.3.2 Task 2: Upload table entities to Azure Storage

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
3. In the **MultiTierService** blade, select the **prodstorage*** storage account that you created earlier in this lab.
4. In the **Storage Account** blade, on the left side of the blade, in the **Table service** section, select the **Tables** link.
5. In the **Tables** section, select **+ Table**.
6. In the **Add table** window, perform the following actions:
    1. In the **Table Name** field, enter **products**.
    2. Select **OK**.
7. Back in the **Tables** section, on the left side of the blade, select the **Overview** link.

    Note: You might have to scroll up or down the menu on the left side of the blade.
8. Back in the **Overview** section, select **Open in Explorer**.
9. In the **Azure Storage Explorer** window, select the **Open Azure Storage Explorer** link.

    Note: If this is your first time opening the **Azure 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.
10. In the **Azure Storage Explorer** application that appears, locate and expand the **prodstorage* S**torage account that you created earlier in this lab.
11. Within the **prodstorage*** storage account, locate and expand the **Tables** node.
12. Within the **Tables** node, select the **products** table that you created earlier in this lab.
13. In the **Products Table** tab, select **Import**.
14. In the **File Explorer** dialog box that opens, perform the following actions:
    1. Go to **Allfiles (F):\Allfiles\Labs\06\Starter**.
    2. Select the **products.csv** file.
    3. Select **Open**.
15. In the **Import Entities** window that appears, select **Insert**.
16. Wait for the table entities to be uploaded before you continue with this lab.
17. Observe the five entities that were added to your **products** table.
18. Return to the browser window showing the **Azure portal**.

25.2.3.3 Task 3: Create an Azure Cognitive Search indexer

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
3. In the **MultiTierService** blade, select the **prodsearch*** Search service you created earlier in this lab.
4. In the **Search Service** blade, select **Import Data**.
5. In the **Import Data** blade, observe the tabs at the top of the blade.
6. In the **Connect to your data** tab, perform the following actions:
    1. In the **Data Source** list, select **Azure Table Storage**.
    2. In the **Name** field, enter **tabledatasource**.
    3. In the **Connection string** field, select **Choose an existing connection**. In the **Choose storage account** window, select the **prodstorage*** Storage account that you created earlier in this lab.
    4. In the **Table name** field, enter **products**.
    5. Leave the **Query** field empty.
    6. Leave the **Description** field empty.
    7. Select **Next: Add cognitive search (Optional)**.

        Note: Azure Cognitive Search validates your settings at each step. It can take a few minutes to complete validation and move to the next tab in the list.
7. In the **Add cognitive search (optional)** tab, select **Skip to: Customize target index**.
8. In the **Customize target index** tab, perform the following actions:
    1. In the **Index name** field, enter **products**.
    2. In the **Key** list, select **RowKey**.
9. In the **Customize target index** tab, a list of fields is displayed. Leave the configuration of the **PartitionKey**, **RowKey**, **ETag,** and **Timestamp** fields set to their default values.
10. In the **Customize target index** tab, perform the following actions to configure the **Key** field:
    1. In the **Field Name** field, observe the hard-coded value of **Key**.
    2. In the **Type** list, observe the hard-coded option of **Edm.String**.
    3. Leave **Retrievable** selected.
    4. Leave **Filterable** unselected.
    5. Select **Sortable**.
    6. Leave **Facetable** unselected.
    7. Leave **Searchable** unselected.
11. In the **Customize target index** tab, perform the following actions to configure the **name** field:
    1. In the **Field Name** field, observe the hard-coded value of **name**.
    2. In the **Type** list, observe the hard-coded option of **Edm.String**.
    3. Select **Retrievable**.
    4. Leave **Filterable** unselected.
    5. Select **Sortable**.
    6. Leave **Facetable** unselected.
    7. Select **Searchable**.
    8. In the **Analyzer** list, select **Standard - Lucene**.
12. In the **Customize target index** tab, perform the following actions to configure the **price** field:

1. In the **Field Name** field, observe the hard-coded value of **price**.
2. In the **Type** list, observe the hard-coded option of **Edm.Double**.
3. Select **Retrievable**.
4. Select **Filterable**.
5. Select **Sortable**.
6. Select **Facetable**.

13. In the **Customize target index** tab, perform the following actions to configure the **quantity** field:
    1. In the **Field Name** field, observe the hard-coded value of **quantity**.
    2. In the **Type** list, observe the hard-coded option of **Edm.Int32**.
    3. Select **Retrievable**.
    4. Select **Filterable**.
    5. Select **Sortable**.
    6. Select **Facetable**.
14. In the **Customize target index** tab, select **Next: Create an indexer**.

    Note: Azure Cognitive Search validates your settings at each step. It can take a few minutes to complete validation and move to the next tab in the list.
15. In the **Create an indexer** tab, perform the following actions:
    1. In the **Name** field, enter **tableindexer**.
    2. In the **Schedule** section, select **Custom**.
    3. In the **Interval** field, enter **5**.
    4. Set the **Start time** field to midnight UTC on today's date.
    5. Leave the **Track deletions** field set to its default value.
    6. Leave the **Description** field blank.
    7. Select **Submit**.

       Note: You may need to select and then unselect the **Track deletions** field and **Once/Custom** options to get the **Submit** button to appear. This behavior is due to a portal bug.
16. Back in the **Search Service** blade, select the **Indexers** tab.

    Note: If you do not see your indexer yet, you may need to refresh the blade.
17. In the **Indexers** tab, select the **tableindexer** indexer that you created earlier in this lab.
18. In the **Indexer** blade, perform the following actions:
    1. Select **Run**.
    2. When prompted for confirmation, select **Yes**.
    3. Close the **Indexer** blade.
19. Wait for the indexer to finish running and then select **Refresh** at the top of the blade.

    Note: It can take from one to five minutes for the indexer to run. You will know that the indexer was successful if its status is listed as **Success** in the **Search Service** blade.
20. In the **Indexers** tab, observe the metadata of the **tableindexer** indexer, such as the document count and the status of the last indexing operation.

25.2.3.4 Task 4: Validate the indexed table data

1. In the **Search Service** blade, select **Search Explorer** at the top of the blade.
2. In the **Search Explorer** blade, select **Search**.
3. Observe the results of a search for all documents.
4. In the **Query string** field, enter the following query and then press **Search**:

   ```
   search=seat
   ```

5. Observe the results of the search query.
6. In the **Query string** field, enter the following query and then press **Search**:

   ```
   $filter=price lt 100
   ```

7. Observe the results of the search query.
8. In the **Query string** field, enter the following query and then press **Search**:

   ```
   facet=quantity,interval:25
   ```

9. Observe the results of the search query.
10. In the **Query string** field, enter the following query and then press **Search**:

    ```
    $filter=quantity gt 25&facet=price,values:100|1000|10000
    ```

11. Observe the results of the search query.
12. Close the **Search Explorer** blade.

25.2.3.5 Task 5: Retrieve your Azure Cognitive Search base URL

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
3. In the **MultiTierService** blade, select the **prodsearch*** Search service that you created earlier in this lab.
4. In the **Search Service** blade, copy the value of the **URL** field. You will use this value later in this lab.

25.2.3.6 Review

In this exercise, you created an Azure Storage account and indexed a Storage table within the account by using Azure Cognitive Search. After the table was indexed, you were able to issue search queries against a copy of the entities in the Storage table.

25.2.4 Exercise 3: Build an API proxy tier by using Azure API Management

25.2.4.1 Task 1: Define a new API

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
3. In the **MultiTierService** blade, select the **prodapi*** API Management account that you created earlier in this lab.
4. In the **API Management Service** blade, on the left side of the 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** field, enter **Search API**.
   2. In the **Name** field, enter **search-api**.

3. In the **Web service URL** field, enter the URL from the **Search Service URL** field that you copied earlier in this lab.
4. Append the value of the **Web service URL** field with the following relative URL:

`/indexes/products/docs`

Note: For example, if your web service URL is https://prodsearchstudent.search.windows.net, then your new URL will be https://prodsearchstudent.search.windows.net/indexes/products/docs.

7. In the **API URL suffix** field, enter **search**.
8. Select **Create.**
9. Wait for the new API to finish being created.
10. In the **Design** tab, select **+ Add operation**.
11. In the **Add operation** section, perform the following actions:
    1. In the **Display name** field, enter **List All Documents**.
    2. In the **Name** field, enter **list-all-documents**.
    3. In the **URL** list, select **GET**.
    4. In the **URL** field, enter **/**.
    5. Select **Save**.
12. Back in the **Design** tab, in the list of operations, select **All Operations**.
13. In the **Design** section for **All Operations**, locate the **Inbound processing** tile and select **+ Add policy**.
14. In the **Add inbound policy** section, select the **Set headers** tile.
15. In the **Inbound processing, Set Headers** section, perform the following actions:
    1. In the **Name** field, enter **api-key**.
    2. In the **Value** field, select the list, select **+ Add Value**, and then enter the value for the **Search Service Key** that you recorded earlier in this lab.
    3. In the **Action** list, select the **override** option.
    4. Select **Save**.
16. Back in the **Design** tab, in the list of operations, select **All Operations**.
17. In the **Design** section for **All Operations**, locate the **Inbound processing** tile and select **+ Add policy**.
18. In the **Add inbound policy** section, select the **Set query parameters** tile.
19. In the **Inbound processing, Set Query Parameters** section, perform the following actions:
    1. In the **Name** field, enter **api-version**.
    2. In the **Value** field, enter **2017-11-11**.
    3. In the **Action** list, select **override**.
    4. Select **Save**.
20. Back in the **Design** tab, in the list of operations, select **List All Documents**.
21. In the **Design** section for the **List All Documents** operation, locate the **Inbound processing** tile and select the **+ Add policy** button.
22. In the **Add inbound policy** section, select the **Set query parameters** tile.
23. In the **Inbound processing, Set Query Parameters** section, perform the following actions:
    1. In the **Name** field, enter **search**.
    2. In the **Value** field, enter ***.
    3. In the **Action** list, select **override**.

4. Select **Save**.

24. Back in the **Design** tab, in the list of operations, select **List All Documents**.
25. Select the **Test** tab.
26. Select the **List All Documents** operation.
27. In the **List All Documents** section, select **Send**.
28. Observe the results of the API request.

    Note: Observe how there is a large amount of Azure Cognitive Search metadata in the response. You might not want API users to know the implementation details that occur behind the scenes. In the next task, you will obfuscate much of this data.

29. Select the **Design** tab to return to the list of operations.

25.2.4.2 Task 2: Manipulate an API response

1. Back in the **Design** tab, in the list of operations, select **List All Documents** .
2. In the **Design** section for the **List All Documents** operation, locate the **Outbound processing** tile and select + **Add policy**.
3. In the **Add outbound policy** section, select the **Other policies** tile.
4. In the policy code editor, locate the following block of XML content:

```
<outbound>
    <base />
</outbound>
```

5. Replace that block of XML with the following XML:

```
<outbound>
    <base />
    <set-body>
    @{
        var response = context.Response.Body.As<JObject>();
        return response.Property("value").Value.ToString();
    }
    </set-body>
</outbound>
```

6. In the policy code editor, select **Save**.
7. Back in the **Design** tab, in the list of operations, select **List All Documents**.
8. In the **Design** section for the **List All Documents** operation, locate the **Outbound processing** tile and select + **Add policy**.
9. In the **Add outbound policy** section, select the **Set headers** tile.
10. In the **Outbound processing, Set Headers** section, perform the following actions:
    1. In the **Name** field, enter **preference-applied**.
    2. In the **Action** list, select **delete**.
    3. Select + **Add header**.
    4. In the new **Name** field, enter **odata-version**.
    5. In the new **Action** list, select **delete**.
    6. Select + **Add header**.
    7. In the new **Name** field, enter **powered-by**.
    8. In the new **Value** field, select the list, select the + **Add Value** link, and then enter **Contoso**.
    9. In the new **Action** list, select **override**.
    10. Select **Save**.

11. Back in the **Design** tab, in the list of operations, select **List All Documents**.
12. Select the **Test** tab.
13. Select the **List All Documents** operation.
14. In the **List All Documents** section, select **Send**.
15. Observe the results of the API request.

    Note: You will observe that the **preference-applied** and **odata-version** headers that you specified have been deleted and replaced with a new **powered-by** header. You will also notice that the response doesn't contain context data about the OData response, but instead contains a flattened JSON array as the response body.

25.2.4.3 Review

In this exercise, you built a proxy tier between your Azure Cognitive Search account and any developers who wish to make search queries.

25.2.5 Exercise 4: Create new table entities by using Azure Logic Apps

25.2.5.1 Task 1: Create a Logic Apps resource

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **Logic** and press Enter.
4. In the **Everything** search results blade, select the **Logic App** result.
5. In the **Logic App** blade, select **Create**.
6. In the **Logic App** blade, perform the following actions:
    1. In the **Name** field, enter **prodworkflow**[***your name in lowercase***].
    2. Leave the **Subscription** field set to its default value.
    3. In the **Resource group** section, select **Use existing** and then select the **MultiTierService** option from the list.
    4. In the **Location** list, select **East US**.
    5. In the **Log Analytics** section, select **Off**.
    6. Select **Create**.
7. Wait for the creation task to complete before you move forward with this lab.

25.2.5.2 Task 2: Create a trigger for Logic Apps workflow

1. In the left navigation pane of the portal, select **Resource groups**.
2. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
3. In the **MultiTierService** blade, select the **prodworkflow*** logic app that you created earlier in this lab.
4. In the **Logic Apps Designer** blade, select the **Blank Logic App** template.
5. In the **Designer** area, perform the following actions to add a **HTTP Trigger**:
    1. In the **Search connectors and triggers** field, enter **HTTP**.
    2. In the **Triggers** result list, select **When a HTTP request is received**.
    3. Select **Use sample payload to generate schema**.
    4. In the **Enter or paste a sample JSON payload** window, enter the following JSON object:

```
{
    "id": "",
    "manufacturer": "",
    "price": 0.00,
    "quantity": 0,
    "name": ""
}
```

6. Select **Done**.
7. Observe the schema in the **Request Body JSON Schema** field. This schema is built by Azure automatically based on the JSON content that you entered in the previous step.

25.2.5.3 Task 3: Build a connector for Azure Storage

1. In the **Designer** area, select **+ New step**.
2. In the **Designer** area, perform the following actions to add an **Insert or Replace Entity Action**:
    1. In the **Search connectors and triggers** field, enter **Table**.
    2. In the category list, select **Azure Table Storage**.
    3. In the **Actions** result list, select **Insert or Replace Entity**.
    4. In the **Connection Name** field, enter **tableconnection**.
    5. In the **Storage Account** section, select the **prodstorage*** Storage account that you created earlier in this lab.
    6. Select **Create**.
    7. Wait for the connector resource to finish creating.

        Note: These resources take one to five minutes to create.
3. In the **Table** list, select **products**.
4. On the right side of the **Partition Key** field, in the **Dynamic content** pane, within the **When a HTTP request is received** category, select **manufacturer**.
5. On the right side of the **Row Key** field, in the **Dynamic content** pane, within the **When a HTTP request is received** category, select **id**.
6. On the right side of the **Entity** field, in the **Dynamic content** pane, within the **When a HTTP request is received** category, select **Body**.

25.2.5.4 Task 4: Build a HTTP response action

1. In the **Designer** area, select **+ New step**.
2. In the **Designer** area, perform the following actions to add a **Response Action**:
    1. In the **Search connectors and triggers** field, enter **Response**.
    2. In the **Actions** result list, select **Response**.
    3. In the **Status Code** field, enter **201**.
    4. On the right side of the **Body** field, in the **Dynamic content** pane, within the **Insert of Replace Entity** category, select **Body**.

25.2.5.5 Task 5: Retrieve a HTTP trigger POST URL

1. In the **Designer** area, select **Save**.
2. After the workflow is saved, the **HTTP POST URL** field in the **When a HTTP request is received** trigger will be updated with a new URL that you'll need to start this workflow. Copy the URL in the **HTTP POST URL** field. You will use this URL later in this lab.

    Note: This is a very long URL because it includes the URL with the SAS token. Make sure that you copy the entire URL.

25.2.5.6 Task 6: Validate that logic app results are indexed

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.

   Note: The **Cloud Shell** icon is represented by a greater than symbol and underscore character.

2. If this is your first time opening the **Cloud Shell** by using your subscription, a **Welcome to Azure Cloud Shell** wizard will display how to configure **Cloud Shell** for first-time usage. Perform the following actions:

   1. When offered a choice between **Bash** or **PowerShell**, select **Bash**.
   2. A dialog box prompts you to create a new Storage account to begin by using the shell. Accept the default settings and select **Create storage**.
   3. Wait for the **Cloud Shell** to finish its first-time setup procedures before you move forward with the lab.

      Note: If you the configuration options for the **Cloud Shell** do not appear, this is most likely because you are using an existing subscription with this course's labs. The labs are written from the perspective that you are using a new subscription.

3. At the bottom of the portal, in the **Cloud Shell** command prompt, enter the following partial **CURL** command to issue a **HTTP POST** request to the Logic Apps instance and then press **Enter**:

   ```
   curl \
   --header "Content-Type: application/json" \
   --data '{"id":"6","manufacturer":"VEHTOP","price":750,"quantity":6,"name":"car roof rack"}' \
   ```

4. Next, enter the logic app's **HTTP POST URL** that you copied earlier in this lab, ensuring that you place the URL within **quotation marks** so that the URL characters are not escaped. Press **Enter** to execute the command.

   Note: For example, if the URL is https://prod.eastus.logic.azure.com:443/workflows/test/triggers/invoke?api-version=2016\&sig=3, then you would insert "https://prod.eastus.logic.azure.com:443/workflows/test/triggers/ma version=2016\&sig=3". If you don't include the quotation marks, you will get an error message indicating that the SAS token has been truncated and is required to issue the request. This occurs because the query string separator, **&,** is truncated if not enclosed in quotation marks.

5. In the left navigation pane of the portal, select **Resource groups**.
6. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.
7. In the **MultiTierService** blade, select the **prodsearch*** Search service that you created earlier in this lab.
8. In the **Search Service** blade, select the **Indexers** tab.
9. In the **Indexers** tab, select the **tableindexer** indexer that you created earlier in this lab.
10. In the **Indexer** blade, perform the following actions:

    1. Select **Run**.
    2. When prompted for confirmation, select **Yes**.
    3. Close the **Indexer** blade.

11. Wait for the indexer to finish running, and then select **Refresh** at the top of the blade.
12. Back in the **Search Service** blade, select **Search Explorer**.
13. In the **Search Explorer** blade, select **Search**.
14. Observe the results of a search for all documents.

    Note: At this point, you will notice a sixth document in your index representing the new document that was inserted by the logic app.

15. In the left navigation pane of the portal, select **Resource groups**.
16. In the **Resource groups** blade, select the **MultiTierService** resource group that you created earlier in this lab.

17. In the **MultiTierService** blade, select the **prodapi*** API Management account that you created earlier in this lab.
18. In the **API Management Service** blade, on the left side of the blade, in the **API Management** section, select **APIs** .
19. In the **APIs** section, select **Search API**.
20. In the **Design** tab, select the **Test** tab.
21. Select the **List All Documents** operation.
22. In the **List All Documents** section, select **Send**.
23. Observe the results of the API request.

    Note: Observe that there are six documents now instead of five.

25.2.5.7 Review

In this exercise, you created a logic app that takes a HTTP request and then persists the JSON body of the request as a new Azure Storage table entity.

25.2.6 Exercise 5: Clean up subscription

25.2.6.1 Task 1: Open Cloud Shell

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.
2. At the bottom of the portal, in the **Cloud Shell** command prompt, type the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

3. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

25.2.6.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **MultiTierService** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

25.2.6.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Microsoft Azure Storage Explorer** application.

25.2.6.4 Review

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

25.4 lab: title: 'Lab: Creating a multi-tier solution by using services in Azure' module: 'AZ-203T06-A: Connect to and consume Azure, and third-party, services'

26 Lab: Creating a multi-tier solution by using services in Azure

27 Student lab manual

27.1 Lab scenario

Your company has successfully adopted and used Microsoft Azure Cognitive Search in a variety of use cases in your solutions. In your latest solution, your company will store data in an Azure Storage table and will need to index that table by using Azure Cognitive Search. To accomplish this, you will build an API with minimal code. There are two major requirements for this solution. First, developers who are issuing REST queries should not be aware through any details that Azure Cognitive Search is being used behind the scenes. Second, developers should be able to add a new record to the table by using a simple REST query. You have decided to use Azure Cognitive Search, Azure API Management, and Azure Logic Apps to build a minimal-code solution to meet all these requirements.

27.2 Objectives

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

- Create an Azure Cognitive Search account.
- Manually create an Azure Cognitive Search index.
- Create an indexer to automatically parse data from a data source.
- Create an API Management account.
- Configure an API as a proxy for another Azure service with header substitution and payload manipulation.
- Create a Logic Apps resource.
- Configure a workflow that is triggered by an HTTP request.

27.3 Lab Setup

- **Estimated time**: 105 minutes

27.4 Instructions

27.4.1 Before you start

27.4.1.1 Sign in to the lab virtual machine

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

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

27.4.1.2 Review installed applications

Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:

- Microsoft Edge
- File Explorer
- Microsoft Azure Storage Explorer

27.4.1.3 Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):** path:

   ```
   cd F:
   ```

3. Within the command prompt, enter the following command and press Enter to clone the **microsoftlearning/AZ-203-DevelopingSolutionsforMicrosoftAzure** project hosted on GitHub into the **Allfiles (F):** drive:

   ```
   git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsF
   ```

4. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-203T06** lab:

   ```
   git checkout master -- Allfiles/*
   ```

5. Close the currently running **Windows PowerShell** command prompt application.

27.4.2 Exercise 1: Creating an Azure Cognitive Search service in the portal

27.4.2.1 Task 1: Open the Azure portal

1. Sign in to the **Azure portal** (portal.azure.com).
2. 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.

27.4.2.2 Task 2: Create API Management resource

In the Azure portal, create a new **API Management account** with the following details:

- **New resource group**: MultiTierService
- **Name**: prodapi[your name in lowercase]
- **Location**: West US
- **Organization name**: Contoso
- **Pricing tier**: Consumption (99.9 SLA, %)

Note: Wait for Azure to finish creating the API Management account before you move forward with this lab. You will receive a notification when the account is created.

27.4.2.3 Task 3: Create Azure Cognitive Search account

1. Create a new **Azure Cognitive Search** instance with the following details:
   - **URL**: prodsearch[your name in lowercase]
   - **Existing Resource group**: MultiTierService
   - **Location**: East US
   - **Pricing tier**: Basic

     Note: Wait for Azure to finish creating the Azure Cognitive Search account before you move forward with the lab. You will receive a notification when the Azure Cognitive Search account is created.
2. Access the **Keys** blade of your newly created **Azure Cognitive Search** instance.
3. Record one of the **Search Keys**. You will use this value later in the lab.

27.4.2.4 Task 4: Create an index

1. Access the **prodsearch*** Search service that you created earlier in this lab.
2. In the **Search Service** blade, select **Add index**.
3. Create a new index with the following details:
   - **Index name**: retail

- **Key**: id

4. Configure the index's fields by using the table below:

Field	Type	Retrievable	Filterable	Sortable	Facetable	Searchable	Analyzer
id	Edm.String						
name	Edm.String						**Standard - Lucen**
price	Edm.Double						
quantity	Edm.Int32						
manufacturer	Edm.String						

27.4.2.5 Review

In this exercise, you created a new Azure Cognitive Search account and built an index within the account.

27.4.3 Exercise 2: Index an Azure Storage table in Azure Cognitive Search

27.4.3.1 Task 1: Create an Azure Storage account

1. In the Azure portal, create a new **storage account** with the following details:
   - **Existing resource group**: MultiTierService
   - **Name**: prodstorage[your name in lowercase]
   - **Location**: (US) East US
   - **Performance**: Standard
   - **Account kind**: StorageV2 (general purpose v2)
   - **Replication**: Read-access geo-redundant storage (RA-GRS)
   - **Access tier**: Hot

     Note: Wait for Azure to finish creating the Storage account before you move forward with the lab. You will receive a notification when the account is created.
2. Access the **Access Keys** blade of your newly created **storage account** instance.
3. Record the value of the **Connection string** field. You will use this value later in this lab.

27.4.3.2 Task 2: Upload table entities to Azure Storage

1. Access the **prodstorage*** Storage account that you created earlier in this lab.
2. In the **Tables service** section, select the **Tables** link.
3. Create a new **Table** with the following setting:
   - **Name**: products
4. In the **Overview** section of the blade, select **Open in Explorer** to open the Storage account by using the **Azure Storage Explorer.**
5. In the **Azure Storage Explorer** application that appears, in the **prodstorage*** Storage account that you created earlier in this lab, locate and open the **products** table.
6. In the **Products Table** tab, select **Import** and import the **products.csv** file located in the **Allfiles (F):\Allfiles\Labs\06\Starter** folder on your lab machine.

27.4.3.3 Task 3: Create an Azure Cognitive Search indexer

1. Access the **prodsearch*** Search service that you created earlier in this lab.
2. In the **Search Service** blade, select **Import Data**.
3. In the **Import Data** blade, create a new indexer with the following details:
   - **Data Source**: Azure Table Storage
   - **Data Source Name**: tabledatasource

- **Connection string**: *<select the **prodstorage*** storage account you created earlier in this lab>*
- **Table name**: products
- **Index name**: products
- **Key**: RowKey
- Configure the index's fields by using the table below:

Field	Type	Retrievable	Filterable	Sortable	Facetable	Searchable	Analyzer
Key	Edm.String						
name	Edm.String						**Standard - Lucene**
price	Edm.Double						
quantity	Edm.Int32						

- **Indexer Name**: tableindexer
- **Schedule**: Custom
- **Interval**: 5
- **Start time**: *Midnight UTC on today's date*

  Note: You may need to select and then unselect the **Track deletions** field and **Once/Custom** options to get the **Submit** button to appear. This behavior is due to a portal bug.

4. Manually **run** the **tableindexer** indexer that you just created in this lab.
5. Observe the metadata of the **tableindexer** indexer and its latest run. The metadata includes information such as the document count and the status of the last indexing operation. You will observe that the last indexing operation was successful and resulted in multiple documents being added to the index.

27.4.3.4 Task 4: Validate the indexed table data

1. Open the **Search Explorer** for the **prodsearch*** Search service that you created earlier in this lab.
2. Execute the empty query by leaving all fields set to their default (empty) values and selecting **Enter**. Observe that the results of this query returns a page of results for all documents in the index.
3. Execute the following query and observe the results:

   ```
   search=seat
   ```

4. Execute the following query and observe the results:

   ```
   $filter=price lt 100
   ```

5. Execute the following query and observe the results:

   ```
   facet=quantity,interval:25
   ```

6. Execute the following query and observe the results:

   ```
   $filter=quantity gt 25&facet=price,values:100|1000|10000
   ```

27.4.3.5 Task 5: Retrieve your Azure Cognitive Search base URL

1. Access the **prodsearch*** Search service that you created earlier in this lab.
2. In the **Search Service** blade, copy the value of the **URL** field. You will use this value later in the lab.

27.4.3.6 Review

In this exercise, you created an Azure Storage account and indexed a Storage table within the account using Azure Cognitive Search. After the table was indexed, you were able to issue search queries against a copy of the entities in the Storage table.

27.4.4 Exercise 3: Build an API proxy tier by using Azure API Management

27.4.4.1 Task 1: Define a new API

1. Access the **prodapi*** API Management account that you created earlier in this lab.
2. View the list of **APIs** for the new account.

   Note: All new accounts start with a simple **Echo API**.

1. Create a new **Blank API** with the following details:
   - **Display name**: Search API
   - **Name**: search-api
   - **Web service URL**: Enter the **Search Service URL** that you copied earlier in this lab. Append the value of the **Web service URL** field with the following relative URL:

     `/indexes/products/docs`

     Note: For example, if your web service URL is https://prodsearchstudent.search.windows.net, then your new URL will be https://prodsearchstudent.search.windows.net/indexes/products/docs.
   - **API URL suffix**: search
2. Add a new **operation** to the recently created API with the following details:
   - **Display name**: List All Documents
   - **Name**: list-all-documents
   - **URL**: GET /
3. Add a new **Set Headers** inbound policy to **All Operations** with the following details:
   - **Name**: api-key
   - **Value**: Enter the value for the Search Service Key that you recorded earlier in this lab.
4. Add a new **Set Query Parameters** inbound policy to **All Operations** with the following details:
   - **Name**: api-version
   - **Value**: 2017-11-11
   - **Action**: override
5. Add a new **Set Query Parameters** inbound policy scoped to the **List All Documents** operation with the following details:
   - **Name**: search
   - **Value**: *
   - **Action**: override
6. Test the **List All Documents** operation in the **Search API**, observing the results of the API request.

   Note: Observe how there is a large amount of Azure Cognitive Search metadata in the response. You might not want API users to observe implementation details that occur behind the scenes. In the next task, you will obfuscate much of this data.

27.4.4.2 Task 2: Manipulate an API response

1. Add a new **Set Headers** outbound policy to **All Operations** with the following details:
   1. **Name**: preference-applied
   2. **Action**: delete
2. Add a new **Set Headers** outbound policy to **All Operations** with the following details:
   1. **Name**: odata-version
   2. **Action**: delete
3. Add a new **Set Headers** outbound policy to **All Operations** with the following details:

1. **Name**: powered-by
2. **Value**: Contoso
3. **Action**: override

4. Add a new custom outbound policy scoped to the **List All Documents** operation by first locating the following block of XML content:

```
<outbound>
    <base />
</outbound>
```

And then replacing that block of XML with the following XML:

```
<outbound>
    <base />
    <set-body>
    @{
        var response = context.Response.Body.As<JObject>();
        return response.Property("value").Value.ToString();
    }
    </set-body>
</outbound>
```

5. Test the **List All Documents** operation in the **Search API** observing the results of the API request.

> **Note**: You will observe that the **preference-applied** and **odata-version** headers you specified have been deleted and replaced with a new **powered-by** header. You will also notice that the response doesn't contain context data about the OData response, but instead contains a flattened JSON array as the response body.

27.4.4.3 Review

In this exercise, you built a proxy tier between your Azure Cognitive Search account and any developers who wish to make search queries.

27.4.5 Exercise 4: Create new table entities by using Azure Logic Apps

27.4.5.1 Task 1: Create a Logic Apps resource

1. In the Azure portal, create a new **logic app** with the following details:
   - **Existing resource group**: MultiTierService
   - **Name**: prodworkflow[your name in lowercase]
   - **Location**: East US
   - **Log Analytics**: Off

> **Note**: Wait for Azure to finish creating the Logic Apps resource prior to moving forward with the lab. You will receive a notification when the resource is created.

27.4.5.2 Task 2: Create a trigger for Logic Apps workflow

1. In the **Logic Apps Designer** blade, select the **Blank Logic App** template.
2. In the **Designer** area, add a new **When a HTTP request is received** trigger by using the following sample JSON document to generate the JSON schema:

```
{
    "id": "",
    "manufacturer": "",
    "price": 0.00,
    "quantity": 0,
    "name": ""
}
```

27.4.5.3 Task 3: Build a connector for Azure Storage

1. In the **Designer** area, add a new "**Insert or Replace Entity"** action with the following details:
   - **Connection Name**: tableconnection
   - **Storage Account**: <Select the **prodstorage*** storage account you created earlier in this lab>
   - **Table**: products
   - **Partition Key**: manufacturer (Dynamic Field - When a HTTP request is received)
   - **Row Key**: id (Dynamic Field - When a HTTP request is received)
   - **Entity**: Body (Dynamic Field - When a HTTP request is received)

27.4.5.4 Task 4: Build a HTTP response action

1. In the **Designer** area, add a new **Response Action** with the following details:
   - **Status Code**: 201
   - **Body**: Body (Dynamic Field - Insert or Replace Entity)

27.4.5.5 Task 5: Retrieve a HTTP trigger POST URL

1. In the **Designer** area, Select **Save**.
2. After the workflow is saved, the **HTTP POST URL** field in the **When a HTTP request is received** trigger will be updated with a new URL that you'll need to start this workflow. Copy the URL in the **HTTP POST URL** field. You will use this URL later in the lab.

   Note: This is a very long URL because it includes the URL with the SAS token. Make sure that you copy the entire URL.

27.4.5.6 Task 6: Validate that logic app results are indexed

1. Open a new Azure Cloud Shell instance.
2. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
3. Within the Cloud Shell command prompt, enter the following partial **CURL** command to issue a **HTTP POST** request to the Logic Apps instance, and then press Enter:

```
curl \
--header "Content-Type: application/json" \
--data '{"id":"6","manufacturer":"VEHTOP","price":750,"quantity":6,"name":"car roof rack"}' \
```

4. Next, enter the logic app*'s* **HTTP POST URL** you copied earlier in this lab, ensuring that you place the URL within **quotation marks** so that the URL characters are not escaped. Press **Enter** to execute the command.

   Note: For example, if the URL is https://prod.eastus.logic.azure.com:443/workflows/test/triggers/invoke?api-version=2016\&sig=3, then you would insert "https://prod.eastus.logic.azure.com:443/workflows/test/triggers/ma version=2016\&sig=3". If you do not include the quotation marks, you will get an error message indicating that the SAS token has been truncated and is required to issue the request. This occurs because the query string separator, **&**, is truncated if it is not enclosed in quotation marks.

5. Access the **prodsearch*** Search service that you created earlier in this lab.
6. Manually **run** the **tableindexer** indexer that you just created in this lab.
7. Open the **Search Explorer** for the **prodsearch*** Search service that you created earlier in this lab.
8. Execute the empty query and observe the results.

   Note: At this point, you will notice a sixth document in your index representing the new document that was inserted by the logic app.

9. Access the **prodapi*** API Management account that you created earlier in this lab.
10. Test the **List All Documents** operation in the **Search API**, observing the results of the API request.

Note: Observe that there are six documents now instead of five.

27.4.5.7 Review

In this exercise, you created a logic app that takes a HTTP request and then persists the JSON body of the request as a new Azure Storage table entity.

27.4.6 Exercise 5: Clean up subscription

27.4.6.1 Task 1: Open Cloud Shell

1. At the top of the portal, select the **Cloud Shell** icon to open a new shell instance.
2. At the bottom of the portal, in the **Cloud Shell** command prompt, type the following command and press Enter to list all resource groups in the subscription:

   ```
   az group list
   ```

3. Type the following command and press Enter to view a list of possible commands to delete a resource group:

   ```
   az group delete --help
   ```

27.4.6.2 Task 2: Delete resource groups

1. Type the following command and press Enter to delete the **MultiTierService** resource group:

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

2. Close the **Cloud Shell** pane at the bottom of the portal.

27.4.6.3 Task 3: Close active applications

1. Close the currently running **Microsoft Edge** application.
2. Close the currently running **Microsoft Azure Storage Explorer** application.

27.4.6.4 Review

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