Module 03 (CSR): Develop solutions that use blob storage

1 Hr 15 Min Remaining

Instructions Resources Help  100%

Required Lab

Sign into [AZ-204T00A-SEA-DEV](https://labclient.labondemand.com/Instructions/868813d7-42ef-487d-9270-b9c67294545c?rc=10) with Username **Admin** and Password Pa55w.rd

Prepare cloud shell for later use

Mount storage in Azure

1. Sign in to the Azure Portal https://portal.azure.com using the below credentials:

|  |  |
| --- | --- |
| Username | LabUser-23768128@cloudslice.onmicrosoft.com |
| Password | SJr20!a!eD |

1. In the toolbar at the top of the Azure portal, select the **Cloud Shell** icon.
2. In the Welcome to Azure Cloud Shell dialog, select **Bash**.
3. On the you have no storage mounted screen select **Show advanced settings**.
4. In the advanced settings screen, fill in the following fields, then click **Create Storage**:

|  |  |
| --- | --- |
| Resource Group | Use existing (**StorageMedia-lod23768128**) |
| Cloud Shell Region | **East US** |
| Storage account (Create new) | cloudshell23768128 |
| File share (create new) | shellstorage |

1. After the cloud shell initializes and puts you at a text prompt, exit the shell.

At the end of this lab, you can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Sign in to the lab virtual machine

1. Hello Altaf Hussain, log on to [AZ-204T00A-SEA-DEV](https://labclient.labondemand.com/Instructions/868813d7-42ef-487d-9270-b9c67294545c?rc=10) lick [Ctrl+Alt+Delete](https://labclient.labondemand.com/Instructions/868813d7-42ef-487d-9270-b9c67294545c?rc=10) to activate the **Ctrl + Alt + Delete** sequence and bring up the logon page.
2. Sign in as Admin with the password Pa55w.rd

Any links like the one above will send Ctrl+Alt+Delete to the selected machine. This can also be done the **Commands** menu (lightning bolt) in the upper-left hand corner of the screen.

The **Lightning Bolt** in the upper left hand of the screen can also be used to **Copy /Paste** strings and sentences from the Instructions **into the VM** rather than typing them out.

* + To accomplish this simply Highligh\*\*t the sentence in the instructions you wish to type text. **CTRL + C** (Copy)
  + Click the lightning bolt icon, in the Dropdown menu select **Type Text** > **Type Clipboard Text**
  + Click into the Type Text window, **CTRL + V** (Paste) to Paste the sentence into the window
  + Select OK

This Lab supports **Redirect Clipboard** functionality in addition to TypeText.

You can use **Redirect Clipboard** to quickly input code blocks and other strings from the Instructions and elsewhere into the virtual machine using standard copy and paste from your local machine's Clipboard directly into the VM. You may need to click **Allow** in your browser to allow access to your local Clipboard.

Note that due to an issue within **Azure Cloudshell**, use your mouse: **Right-click + Paste** instead of **CTRL+V** when using **Cloudshell** inside the VM.

Download the lab files

Download Files

This will copy the latest lab files from GitHub and configure the directories as needed for the remainder of the lab exercises. You will see a 'Success!' message above, once the process is complete.

Lab Credentials:

* Username: LabUser-23768128@cloudslice.onmicrosoft.com
* Password: SJr20!a!eD

Sites used:

* https://portal.azure.com/#home
* https://shell.azure.com

All the resource groups you require in this lab have been created for you as part of the lab setup. You can safely ignore any steps that ask you to create a resource group. Please use or select a pre-existing resource group that has a similar name to the one you are asked to create. If no similarly named resource group exists, use any existing resource group.

For access to Azure, use the following credentials:

* Username: LabUser-23768128@cloudslice.onmicrosoft.com
* Password: SJr20!a!eD

You can see your available Resource Groups by checking the Resources tab above.

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

Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure user interface (UI) changes that occur after the development of this training content. As a result, the lab instructions and lab steps might not align correctly.

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

Instructions

Before you start

Sign in to the lab environment

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

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

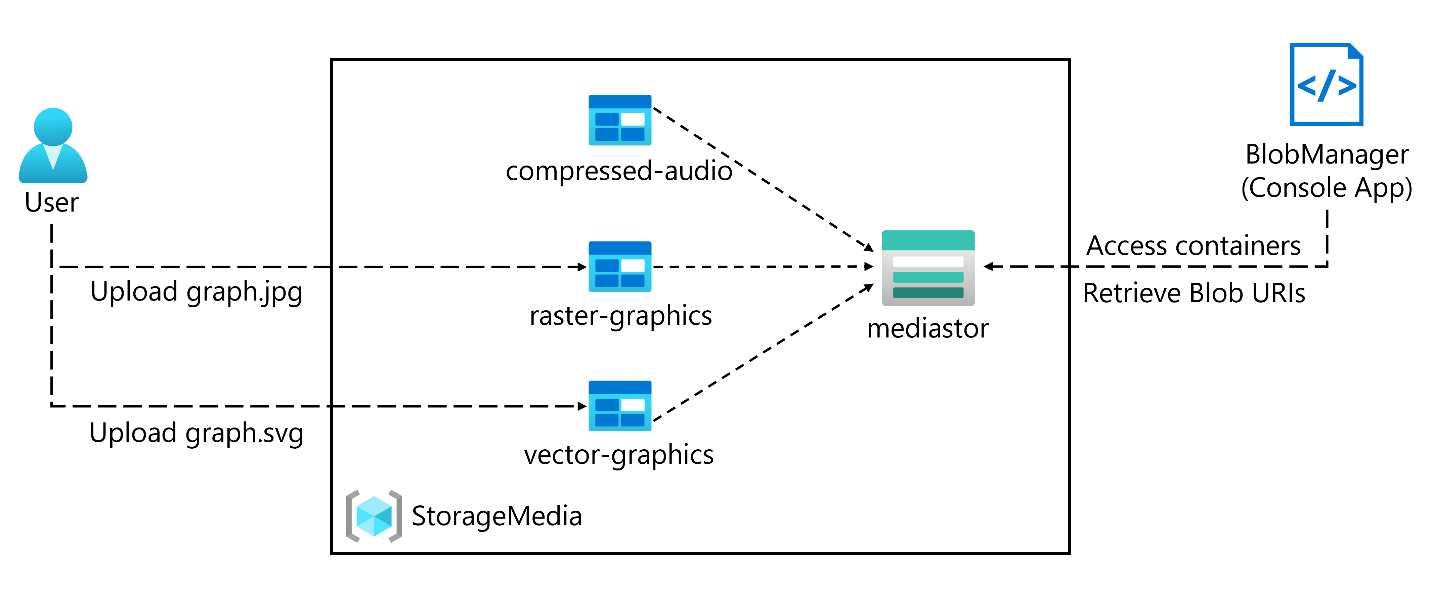
**Note**: Your instructor will provide instructions to connect to the virtual lab environment.

Review the installed applications

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

* Microsoft Edge
* File Explorer

Architecture diagram



Exercise 1: Create Azure resources

Task 1: Open the Azure portal

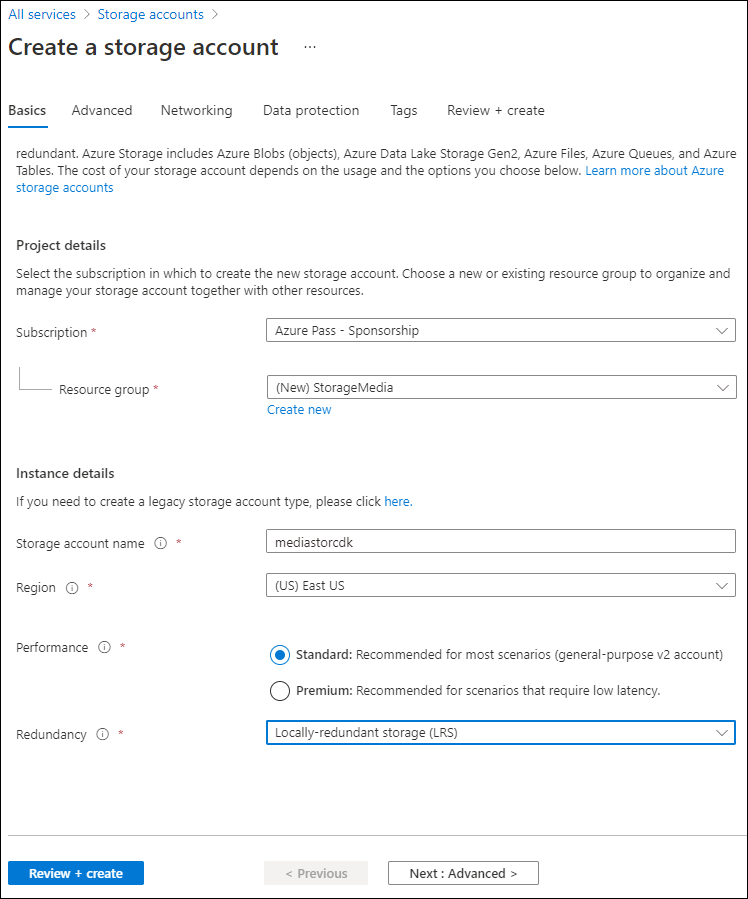
1. On the taskbar, select the **Microsoft Edge** icon.
2. In the browser window, browse to the Azure portal https://portal.azure.com, and then sign in with the account you'll be using for this lab.

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

Task 2: Create a Storage account

1. In the Azure portal, use the **Search resources, services, and docs** text box to search for **Storage Accounts**, and then in the list of results, select **Storage Accounts**.
2. On the **Storage accounts** blade, select **+ Create**.
3. On the **Create a storage account** blade, on the **Basics** tab, perform the following actions, and then select **Review + create**:

| **Setting** | **Action** |
| --- | --- |
| **Subscription** drop-down list | Retain the default value. |
| **Resource group** section | Use existing, and select **StorageMedia-lod23768128**. |
| **Storage account name** text box | Enter **mediastor***[yourname]*. |
| **Region** drop-down list | Select **(US) East US**. |
| **Performance** section | Select the **Standard** option. |
| **Redundancy** drop-down list | select **Locally-redundant storage (LRS)**. |

1. The following screenshot displays the configured settings on the **Create a storage account blade**.
2. 
3. On the **Review + create** tab, review the options that you selected during the previous steps.
4. Select **Create** to create the storage account by using your specified configuration.

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

1. Select **Go to resource**.
2. On the **Storage account** blade, in the **Settings** section, select the **Endpoints** link.
3. In the **Endpoints** section, copy the value of the **Blob Service** text box to the clipboard.

**Note**: You'll use this endpoint value later in the lab.

1. Open Notepad, and then paste the copied blob service value to Notepad.
2. On the **Storage account** blade, in the **Security + networking** section, select **Access keys**.
3. Copy the **Storage account name** value to the clipboard and then paste it into Notepad.
4. On the **Access keys** blade, select **Show keys**.
5. Review any one of the keys, and then copy the value of either of the **Key** boxes to the clipboard.

**Note**: You'll use all these values later in this lab.

Review

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

Exercise 2: Upload a blob into a container

Task 1: Create storage account containers

1. On the **Storage account** blade, select the **Containers** link in the **Data storage** section.
2. In the **Containers** section, select **+ Container**.
3. In the **New container** pop-up window, perform the following actions, and then select **Create**:

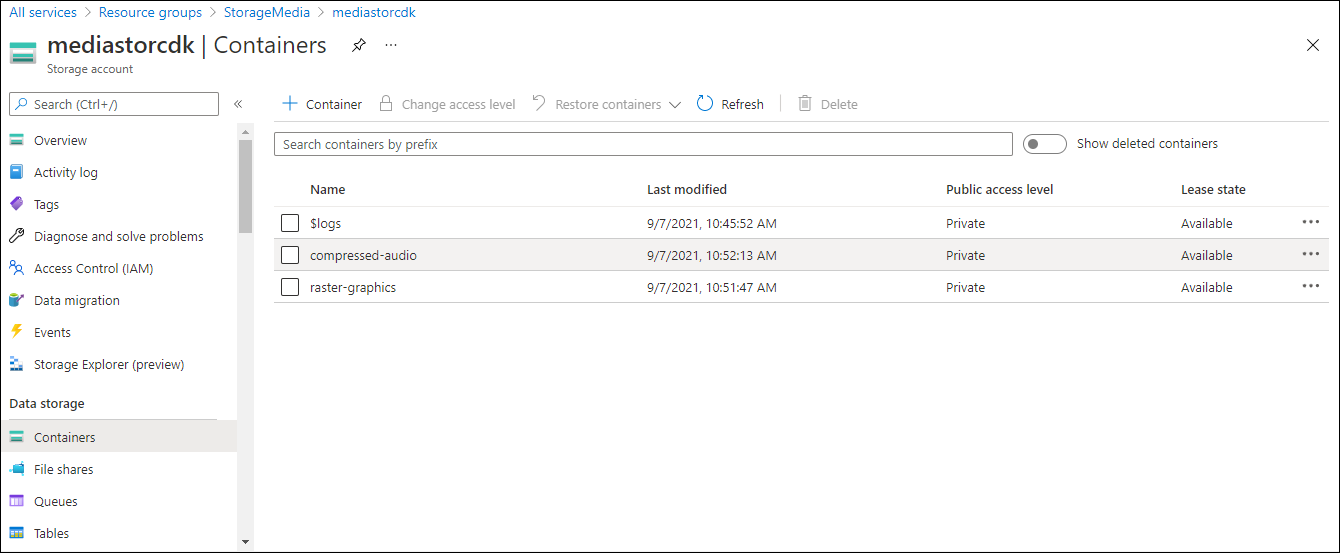
| **Setting** | **Action** |
| --- | --- |
| **Name** text box | Enter **raster-graphics**. |
| **Public access level** drop-down list | Select **Private (no anonymous access)**. |

1. In the **Containers** section, select **+ Container**.
2. In the **New container** pop-up window, perform the following actions and then select **Create**:

| **Setting** | **Action** |
| --- | --- |
| **Name** text box | Enter **compressed-audio**. |
| **Public access level** drop-down list | Select **Private (no anonymous access)**. |

1. In the **Containers** section, observe the updated list of containers.

The following screenshot displays the configured settings on the **Create a storage account blade**.



Task 2: Upload a storage account blob

1. In the **Containers** section, select the recently created **raster-graphics** container.
2. On the **Container** blade, select **Upload**.
3. In the **Upload blob** window, perform the following actions, and then select **Upload**:

| **Setting** | **Action** |
| --- | --- |
| **Files** section | Select the **Folder** icon. |
| **File Explorer** window | Browse to **Allfiles (F):\Allfiles\Labs\03\Starter\Images**, select the **graph.jpg** file, and then select **Open**. |
| **Overwrite if files already exist** check box | Ensure that the check box is selected. |

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

Review

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

Exercise 3: Access containers by using the .NET SDK

Task 1: Create .NET project

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**, browse to **Allfiles (F):\Allfiles\Labs\03\Starter\BlobManager**, and then select **Select Folder**.
3. In the **Visual Studio Code** window, on the Menu Bar, select **Terminal** and then select **New Terminal**.
4. In the terminal, run the following command to create a new .NET project named **BlobManager** in the current folder:

dotnet new console --name BlobManager --output .

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

1. In the terninal, run the following command to import version 12.0.0 of **Azure.Storage.Blobs** from NuGet:

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

**Note**: The **dotnet add package** command will add the **Azure.Storage.Blobs** package from NuGet. For more information, refer to [Azure.Storage.Blobs](https://www.nuget.org/packages/Azure.Storage.Blobs/12.0.0).

1. In the terminal, run the following command to build the .NET web application:

dotnet build

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

Task 2: Modify the Program class to access Storage

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

csharp

using Azure.Storage;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

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

csharp

using System;

using System.Threading.Tasks;

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

csharp

public class Program

{

}

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

csharp

private const string blobServiceEndpoint = "";

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

csharp

private const string storageAccountName = "";

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

csharp

private const string storageAccountKey = "";

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

csharp

public static async Task Main(string[] args)

{

}

1. Review the **Program.cs** file, which should now include:

csharp

using Azure.Storage;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using System;

using System.Threading.Tasks;

public class Program

{

private const string blobServiceEndpoint = "<primary-blob-service-endpoint>";

private const string storageAccountName = "<storage-account-name>";

private const string storageAccountKey = "<key>";

public static async Task Main(string[] args)

{

}

}

Task 3: Connect to the Azure Storage blob service endpoint

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

csharp

StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName, storageAccountKey);

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

csharp

BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCredentials);

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

csharp

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

csharp

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

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

csharp

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

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

csharp

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

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

csharp

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

1. Review the **Main** method, which should now include:

csharp

public static async Task Main(string[] args)

{

StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName, storageAccountKey);

BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCredentials);

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

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

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

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

}

1. Save the **Program.cs** file.
2. In the **Visual Studio Code** window, on the Menu Bar, select **Terminal** and then select **New Terminal**.
3. In the terminal, run the following command to run the .NET web application:

dotnet run

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

1. Observe the output from the currently running console application. The output contains metadata for the storage account that was retrieved from the service.
2. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

Task 4: Enumerate the existing containers

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

csharp

private static async Task EnumerateContainersAsync(BlobServiceClient client)

{

}

1. In the **EnumerateContainersAsync** method, enter the following code to create an asynchronous **foreach** loop that iterates over the results of an invocation of the **GetBlobContainersAsync** method of the **BlobServiceClient** class:

csharp

await foreach (BlobContainerItem container in client.GetBlobContainersAsync())

{

}

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

csharp

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

1. Review the **EnumerateContainersAsync** method, which should now include:

csharp

private static async Task EnumerateContainersAsync(BlobServiceClient client)

{

await foreach (BlobContainerItem container in client.GetBlobContainersAsync())

{

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

}

}

1. In the **Main** method, enter the following code at the end of the method to invoke the **EnumerateContainersAsync** method, passing in the *serviceClient* variable as a parameter:

csharp

await EnumerateContainersAsync(serviceClient);

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

csharp

using Azure.Storage;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using System;

using System.Threading.Tasks;

public class Program

{

private const string blobServiceEndpoint = "your blobServiceEndpoint";

private const string storageAccountName = "your storageAccountName";

private const string storageAccountKey = "your storageAccountKey";

public static async Task Main(string[] args)

{

StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName, storageAccountKey);BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCredentials);

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

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

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

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

await EnumerateContainersAsync(serviceClient);

}

private static async Task EnumerateContainersAsync(BlobServiceClient client)

{

await foreach (BlobContainerItem container in client.GetBlobContainersAsync())

{

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

}

}

}

1. Save the **Program.cs** file.
2. In the **Visual Studio Code** window, on the Menu Bar, select **Terminal** and then select **New Terminal**.
3. In the terminal, run the following command to run the .NET web application:

dotnet run

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

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

Review

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

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

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

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

csharp

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

{

}

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

csharp

BlobContainerClient container = client.GetBlobContainerClient(containerName);

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

csharp

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

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

csharp

await foreach (BlobItem blob in container.GetBlobsAsync())

{

}

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

csharp

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

1. Review the **EnumerateBlobsAsync** method, which should now include:

csharp

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

{

BlobContainerClient container = client.GetBlobContainerClient(containerName);

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

await foreach (BlobItem blob in container.GetBlobsAsync())

{

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

}

}

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

csharp

string existingContainerName = "raster-graphics";

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

csharp

await EnumerateBlobsAsync(serviceClient, existingContainerName);

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

csharp

using Azure.Storage;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using System;

using System.Threading.Tasks;

public class Program

{

private const string blobServiceEndpoint = "your blobServiceEndpoint";

private const string storageAccountName = "your storageAccountName";

private const string storageAccountKey = "your storageAccountKey";

public static async Task Main(string[] args)

{

StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName, storageAccountKey);BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCredentials);

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

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

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

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

await EnumerateContainersAsync(serviceClient);

string existingContainerName = "raster-graphics";

await EnumerateBlobsAsync(serviceClient, existingContainerName);

}

private static async Task EnumerateContainersAsync(BlobServiceClient client)

{

await foreach (BlobContainerItem container in client.GetBlobContainersAsync())

{

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

}

}

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

{

BlobContainerClient container = client.GetBlobContainerClient(containerName);await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");

await foreach (BlobItem blob in container.GetBlobsAsync())

{

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

}

}

}

1. Save the **Program.cs** file.
2. In the **Visual Studio Code** window, on the Menu Bar, select **Terminal** and then select **New Terminal**.
3. In the terminal, run the following command to run the .NET web application:

dotnet run

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

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

Task 2: Create a new container by using the SDK

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

csharp

private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string containerName)

{

}

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

csharp

BlobContainerClient container = client.GetBlobContainerClient(containerName);

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

csharp

await container.CreateIfNotExistsAsync(PublicAccessType.Blob);

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

csharp

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

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

csharp

return container;

1. Review the **GetContainerAsync** method, which should now include:

csharp

private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string containerName)

{

BlobContainerClient container = client.GetBlobContainerClient(containerName);

await container.CreateIfNotExistsAsync(PublicAccessType.Blob);

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

return container;

}

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

csharp

string newContainerName = "vector-graphics";

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

csharp

BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName);

1. Review the **Program.cs** file, which should now include:

csharp

using Azure.Storage;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using System;

using System.Threading.Tasks;

public class Program

{

private const string blobServiceEndpoint = "your blobServiceEndpoint";

private const string storageAccountName = "your storageAccountName";

private const string storageAccountKey = "your storageAccountKey";

public static async Task Main(string[] args)

{

StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName, storageAccountKey);BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCredentials);

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

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

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

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

await EnumerateContainersAsync(serviceClient);

string existingContainerName = "raster-graphics";

await EnumerateBlobsAsync(serviceClient, existingContainerName);

string newContainerName = "vector-graphics";

BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName);

}

private static async Task EnumerateContainersAsync(BlobServiceClient client)

{

await foreach (BlobContainerItem container in client.GetBlobContainersAsync())

{

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

}

}

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

{

BlobContainerClient container = client.GetBlobContainerClient(containerName);await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");

await foreach (BlobItem blob in container.GetBlobsAsync())

{

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

}

}

private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string containerName)

{

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

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

return container;

}

}

1. Save the **Program.cs** file.
2. In the **Visual Studio Code** window, on the Menu Bar, select **Terminal** and then select **New Terminal**.
3. In the terminal, run the following command to run the .NET web application:

dotnet run

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

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

Task 3: Upload a new blob by using the portal

1. On the Azure portal's **navigation** pane, select the **Resource groups** link.
2. On the **Resource groups** blade, select the **StorageMedia-lod23768128** resource group that you created previously in this lab.
3. On the **StorageMedia-lod23768128** blade, select the **mediastor***[yourname]* storage account that you created previously in this lab.
4. On the **Storage account** blade, select the **Containers** link in the **Data storage** section.
5. In the **Containers** section, select the newly created **vector-graphics** container. You might need to refresh the page to observe the new container.
6. On the **Container** blade, select **Upload**.
7. In the **Upload blob** window, perform the following actions, and then select **Upload**:

| **Setting** | **Action** |
| --- | --- |
| **Files** section | Select the **Folder** icon. |
| **File Explorer** window | **Allfiles (F):\Allfiles\Labs\03\Starter\Images**, select the **graph.svg** file, and then select **Open**. |
| **Overwrite if files already exist** check box | Ensure that the check box is selected. |

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

Task 4: Access blob URI by using the SDK

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

csharp

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

{

}

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

csharp

BlobClient blob = client.GetBlobClient(blobName);

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

csharp

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

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

csharp

return blob;

1. Review the **GetBlobAsync** method, which should now include:

csharp

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

{

BlobClient blob = client.GetBlobClient(blobName);

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

return blob;

}

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

csharp

string uploadedBlobName = "graph.svg";

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

csharp

BlobClient blobClient = await GetBlobAsync(containerClient, uploadedBlobName);

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

csharp

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

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

csharp

using Azure.Storage;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using System;

using System.Threading.Tasks;

public class Program

{

private const string blobServiceEndpoint = "your blobServiceEndpoint";

private const string storageAccountName = "your storageAccountName";

private const string storageAccountKey = "your storageAccountKey";

public static async Task Main(string[] args)

{

StorageSharedKeyCredential accountCredentials = new StorageSharedKeyCredential(storageAccountName, storageAccountKey); BlobServiceClient serviceClient = new BlobServiceClient(new Uri(blobServiceEndpoint), accountCredentials);

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

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

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

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

await EnumerateContainersAsync(serviceClient);

string existingContainerName = "raster-graphics";

await EnumerateBlobsAsync(serviceClient, existingContainerName);

string newContainerName = "vector-graphics";

BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName);

string uploadedBlobName = "graph.svg";

BlobClient blobClient = await GetBlobAsync(containerClient, uploadedBlobName);

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

}

private static async Task EnumerateContainersAsync(BlobServiceClient client)

{

await foreach (BlobContainerItem container in client.GetBlobContainersAsync())

{

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

}

}

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

{

BlobContainerClient container = client.GetBlobContainerClient(containerName); await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");

await foreach (BlobItem blob in container.GetBlobsAsync())

{

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

}

}

private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string containerName)

{

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

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

return container;

}

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

{

BlobClient blob = client.GetBlobClient(blobName);

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

return blob;

}

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

{

BlobContainerClient container = client.GetBlobContainerClient(containerName); await Console.Out.WriteLineAsync($"Searching:\t{container.Name}");

await foreach (BlobItem blob in container.GetBlobsAsync())

{

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

}

}

private static async Task<BlobContainerClient> GetContainerAsync(BlobServiceClient client, string containerName)

{

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

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

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

return container;

}

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

{

BlobClient blob = client.GetBlobClient(blobName);

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

return blob;

}

}

1. Save the **Program.cs** file.
2. In the **Visual Studio Code** window, activate the shortcut menu for the **Explorer** pane, and then select **Open in Integrated Terminal**.
3. At the open command prompt, run the following command to run the .NET web application:

dotnet run

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

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

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

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

Task 5: Test the URI by using a browser

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

Review

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

Exercise 5: Clean up your subscription

You can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Task 1: Open Azure Cloud Shell and list resource groups

1. In the Azure portal, select the **Cloud Shell** icon Cloud Shell icon to open a new Bash session. If Cloud Shell defaults to a PowerShell session, select **PowerShell** and, in the drop-down menu, select **Bash**.

**Note**: If this is the first time you're starting **Cloud Shell**, when prompted to select either **Bash** or **PowerShell**, select **PowerShell**. When you're presented with the **You have no storage mounted** message, select the subscription you're using in this lab, and then select **Create storage**.

Task 2: Delete a resource group

1. On the **Cloud Shell** pane, run the following command to delete the **StorageMedia-lod23768128** resource group:

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

**Note**: The command executes asynchronously (as determined by the *--no-wait* parameter), so while you'll be able to run another Azure CLI command immediately afterwards within the same Bash session, it'll take a few minutes before the resource groups are actually removed.

1. Close the **Cloud Shell** pane in the portal.

Task 3: Close the active application

* Close the currently running Microsoft Edge application.

Review

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

Congratulations!

You have successfully completed this **Lab** press **End** to end your lab.

0% Tasks Complete

End

Live Chat