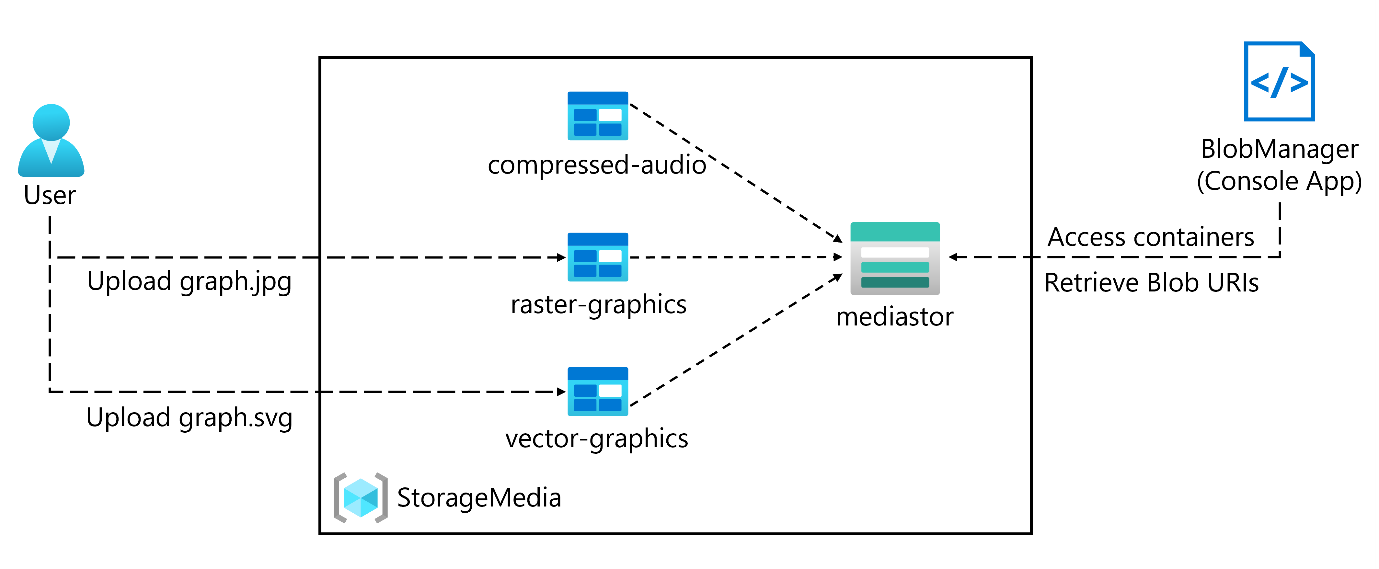
AZ204 LAbworks

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

Lab URL : <https://microsoftlearning.github.io/AZ-204-DevelopingSolutionsforMicrosoftAzure/Instructions/Labs/AZ-204_lab_03.html>

## Architecture diagram

[](https://microsoftlearning.github.io/AZ-204-DevelopingSolutionsforMicrosoftAzure/Instructions/Labs/media/Lab03-Diagram.png)

### Exercise 1: Create Azure resources

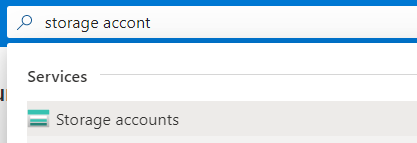
#### Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the browser window, browse to the Azure portal ([https://portal.azure.com](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**.



1. On the **Storage accounts** blade, select **+ Create**.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. 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 | Select **Create new**, enter **StorageMedia**, and then select **OK**. |
| **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)**. |

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

Graphical user interface, text, application, email

Description automatically generated

1. On the **Review + create** tab, review the options that you selected during the previous steps.
2. 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 resouce**.
2. On the **Storage account** blade, in the **Settings** section, select the **Endpoints** link.

Graphical user interface, application

Description automatically generated

1. In the **Endpoints** section, record the value of the **Blob Service** text box.

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



1. On the **Storage account** blade, in the **Security + networking** section, select the **Access keys** link.

Graphical user interface, application

Description automatically generated

1. In the **Access keys** section, perform the following actions:

a. Record the value in the **Storage account name** text box.

b. Select **Show Keys**.

c. Select any one of the keys, and then record the value in either of the **Key** boxes.

**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**.

Graphical user interface, text, application

Description automatically generated

1. 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)**. |

Graphical user interface, text, application, email

Description automatically generated

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)**. |

Graphical user interface, text, application

Description automatically generated

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**.

Graphical user interface, text, application, email

Description automatically generated

#### 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**:

Graphical user interface, application

Description automatically generated

| 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**.

Graphical user interface, text, application

Description automatically generated

1. In the **Visual Studio Code** window, activate the shortcut menu for the **Explorer** pane, and then select **Open in Integrated Terminal**.

Graphical user interface, application

Description automatically generated

**Note**: To see the Explorer pane’s shortcut menu, select Explorer on the left navigation menu and then right-click anywhere in the file explorer’s pane.

1. At the command prompt, run the following command to create a new .NET project named **BlobManager** in the current folder:

CodeCopy

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.

Table

Description automatically generated

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

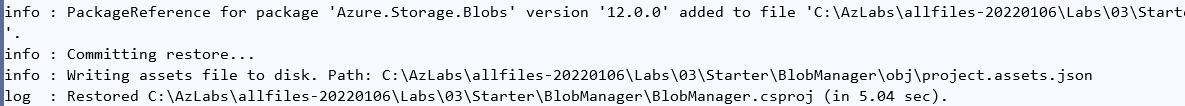
CodeCopy

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).

Text

Description automatically generated



1. At the command prompt, run the following command to build the .NET web application:

CodeCopy

dotnet build

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

Graphical user interface, text, application, email

Description automatically generated

**The build succeeded but ended with a warning since this was done in VS 2019, which does not use target version 6.**

**Instead, run it from command prompt to avoid the warning**

Text

Description automatically generated

#### 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:

C#Copy

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:

C#Copy

using System;

using System.Threading.Tasks;

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

C#Copy

public class Program

{

}

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

C#Copy

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**:

C#Copy

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**:

C#Copy

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:

C#Copy

public static async Task Main(string[] args)

{

}

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

C#Copy

Graphical user interface, text, application, email

Description automatically generated

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:

C#Copy

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:

C#Copy

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:

C#Copy

AccountInfo info = await serviceClient.GetAccountInfoAsync();

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

C#Copy

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:

C#Copy

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:

C#Copy

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:

CodeCopy

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

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

C#Copy

Graphical user interface, text, application

Description automatically generated

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, 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:

CodeCopy

dotnet run

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

Text

Description automatically generated

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:

C#Copy

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:

C#Copy

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

{

}

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

C#Copy

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

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

C#Copy

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:

C#Copy

await EnumerateContainersAsync(serviceClient);

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

C#Copy

public static async Task Main(string[] args)

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}");

}

}

}

Graphical user interface, text, application, email

Description automatically generated

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:

CodeCopy

dotnet run

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

Text

Description automatically generated

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**:

C#Copy

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:

C#Copy

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:

C#Copy

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:

C#Copy

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

{

}

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

C#Copy

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

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

C#Copy

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**:

C#Copy

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:

C#Copy

await EnumerateBlobsAsync(serviceClient, existingContainerName);

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

C#Copy

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())

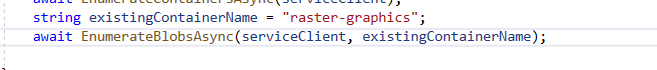
{

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

}

}

}



Graphical user interface, text, application

Description automatically generated

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:

CodeCopy

dotnet run

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

Text

Description automatically generated

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**:

C#Copy

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:

C#Copy

BlobContainerClient container = client.GetBlobContainerClient(containerName);

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

C#Copy

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:

C#Copy

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:

C#Copy

return container;

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

C#Copy

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**:

C#Copy

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**:

C#Copy

BlobContainerClient containerClient = await GetContainerAsync(serviceClient, newContainerName);

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

C#Copy

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;

}

}



Graphical user interface, text, application

Description automatically generated

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:

CodeCopy

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** resource group that you created previously in this lab.
3. On the **StorageMedia** 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.

Graphical user interface, application

Description automatically generated

1. In the **Containers** section, select the newly created **vector-graphics** container. You might need to refresh the page to observe the new container.

Graphical user interface, text, application, email

Description automatically generated

1. On the **Container** blade, select **Upload**.

Graphical user interface, application

Description automatically generated

1. 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**:

C#Copy

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:

C#Copy

BlobClient blob = client.GetBlobClient(blobName);

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

C#Copy

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:

C#Copy

return blob;

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

C#Copy

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**:

C#Copy

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**:

C#Copy

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:

C#Copy

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

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

C#Copy

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($"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)

{

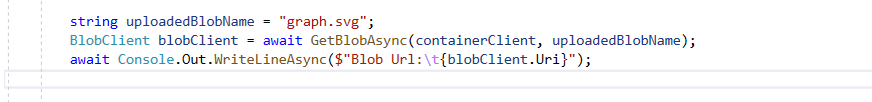
BlobClient blob = client.GetBlobClient(blobName);

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

return blob;

}

}



Graphical user interface, text, application

Description automatically generated

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:

CodeCopy

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

Text

Description automatically generated

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.

Icon

Description automatically generated

1. 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

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

1. In the Azure portal, select the **Cloud Shell** icon [Cloud Shell icon](https://microsoftlearning.github.io/AZ-204-DevelopingSolutionsforMicrosoftAzure/Instructions/Labs/media/az204_lab_CloudShell.png) 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** resource group:

CodeCopy

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.