

AZ-204

**Developing solutions for Microsoft Azure**

**Lab 10**

Asynchronously processing messages by using  
Azure Queue Storage

## Table of Contents

1	Pre-requisites .....	3
1.1	Sign in to the lab virtual machine .....	3
1.2	Review the installed applications .....	3
2	Exercise 1: Create Azure resources .....	4
2.1	Task 1: Open the Azure portal .....	4
2.2	Task 2: Create a Storage account .....	4
2.3	Review .....	6
3	Exercise 2: Configure the Azure Storage SDK in a .NET project .....	7
3.1	Task 1: Create a .NET project .....	7
3.2	Task 2: Write code to access Azure Storage .....	7
3.3	Task 3: Validate Azure Storage access .....	9
3.4	Review .....	10
4	Exercise 3: Read messages from the queue .....	11
4.1	Task 1: Write code to access queue messages .....	11
4.2	Task 2: Test message queue access .....	12
4.3	Task 3: Delete queued messages .....	14
4.4	Review .....	15
5	Exercise 4: Queue new messages by using .NET .....	16
5.1	Task 1: Write code to create queue messages .....	16
5.2	Task 2: View queued messages by using Storage Explorer .....	16
5.3	Review .....	16
6	Exercise 5: Clean up your subscription .....	17
6.1	Task 1: Open Azure Cloud Shell and list resource groups .....	17
6.2	Task 2: Delete a resource group .....	17
6.3	Task 3: Close the active application .....	17
6.4	Review .....	17

# 1 Pre-requisites

## 1.1 Sign in to the lab virtual machine

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

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

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

## 1.2 Review the installed applications

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

- Microsoft Edge
- File Explorer
- Azure CLI
- Windows PowerShell

## 2 Exercise 1: Create Azure resources

### 2.1 Task 1: Open the Azure portal

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

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

### 2.2 Task 2: Create a Storage account

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

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

6. Select the **Basics** tab, and then in the tab area, perform the following actions:
  1. Leave the **Subscription** text box set to its default value.
  2. In the **Resource group** section, select **Create new**, enter **AsyncProcessor**, and then select **OK**.
  3. In the **Storage account name** text box, enter **asyncstor[yourname]**.
  4. In the **Region** list, select the **(US) East US** region.
  5. In the **Performance** section, select **Standard**.
  6. In the **Redundancy** list, select **Locally-redundant storage (LRS)**.

Home > Create a resource > Storage account >

## Create a storage account ...

Basics Advanced Networking Data protection Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

### Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription \*

Resource group \*  [Create new](#)

### Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ \*

Region ⓘ \*

Performance ⓘ \* ☒ Standard: Recommended for most scenarios (general-purpose v2 account)  
☐ Premium: Recommended for scenarios that require low latency.

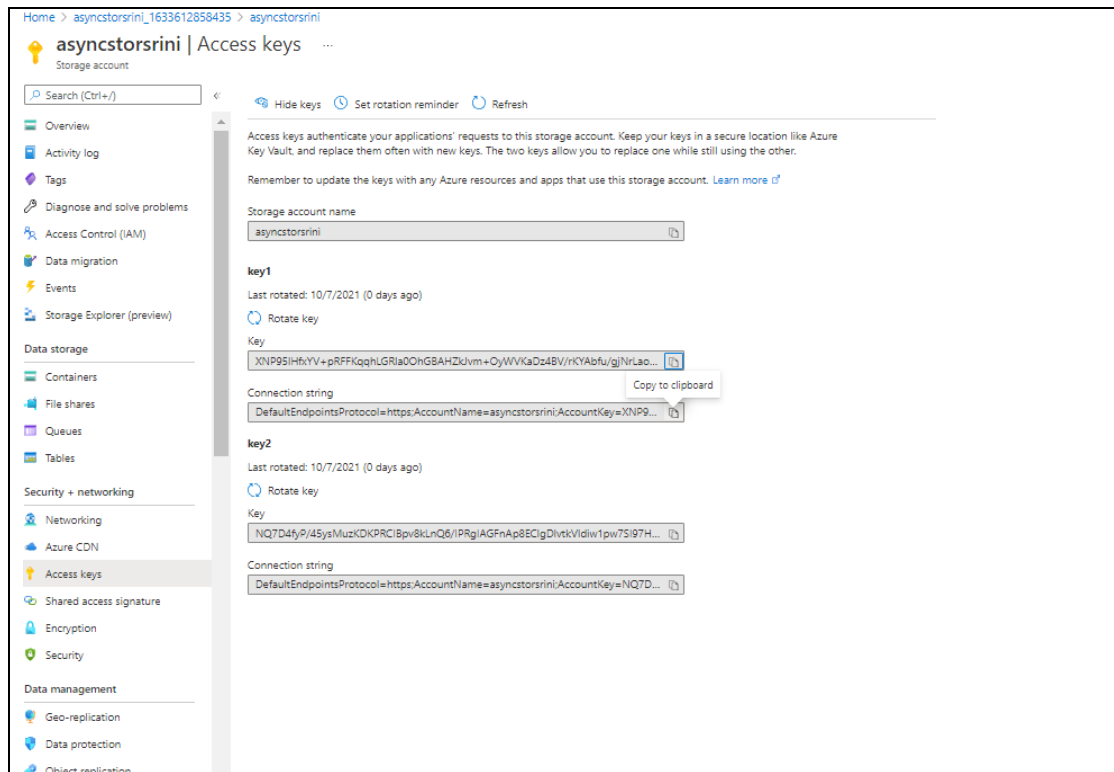
Redundancy ⓘ \*

[Review + create](#) [< Previous](#) [Next : Advanced >](#)

7. Select **Review + Create**.
7. On the **Review + Create** tab, review the options that you specified in the previous steps.
8. Select **Create** to create the storage account by using your specified configuration.

**Note:** On the **Deployment** blade, wait for the creation task to complete before moving forward with this lab.

9. Select the **Go to resource** button on the **Deployment** blade to go to the newly created storage account.
10. On the **Storage account** blade, find the **Security + networking** section, and then select **Access keys**.
11. On the **Access keys** blade, select **Show keys**.
12. Select any one of the keys, and then record the value of either of the **Connection string** boxes. You'll use this value later in this lab.



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

## 2.3 Review

In this exercise, you created a new Azure Storage account that you'll use through the remainder of the lab.

## 3 Exercise 2: Configure the Azure Storage SDK in a .NET project

### 3.1 Task 1: Create a .NET project

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\10\Starter\MessageProcessor**, and then select **Select Folder**.
4. In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
5. At the open command prompt, enter the following command, and then select Enter to create a new .NET project named **MessageProcessor** in the current folder:

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

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

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

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

**Note:** The **dotnet add package** command will add the **Azure.Storage.Queues** package from NuGet. For more information, go to [Azure.Storage.Queues](#).

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

```
dotnet build
```

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

### 3.2 Task 2: Write code to access Azure Storage

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

```
using Azure;  
using Azure.Storage.Queues;  
using Azure.Storage.Queues.Models;
```

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

```
using System;
```

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

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

```
namespace MessageProcessor
{
    class Program
    {
    }
}
```

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

```
private const string storageConnectionString = "";
```

7. Update the **storageConnectionString** string constant by setting its value to the **Connection string** of the Storage account that you recorded earlier in this lab.
8. In the **Program** class, enter the following line of code to create a new string constant named **queueName** with a value of **messagequeue**:

```
private const string queueName = "messagequeue";
```

9. In the **Program** class, enter the following code to create a new asynchronous **Main** method:

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

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

```
using Azure;
using Azure.Storage.Queues;
using Azure.Storage.Queues.Models;

using System;
using System.Text;
using System.Threading.Tasks;

namespace MessageProcessor
```



```
{
    class Program
    {
        private const string storageConnectionString = "";
        private const string queueName = "messagequeue";
        public static async Task Main(string[] args)
        {

        }

    }
}
```

### 3.3 Task 3: Validate Azure Storage access

1. In the **Main** method, add the following line of code to connect to the storage account by creating a new variable named *client* of type **QueueClient**:

```
QueueClient client = new QueueClient(storageConnectionString, queueName);
```

2. In the **Main** method, add the following line of code to asynchronously create the queue if it doesn't already exist:

```
await client.CreateAsync();
```

3. In the **Main** method, add the following line of code to render a header for the "Account Metadata" section:

```
Console.WriteLine($"---Account Metadata---");
```

4. In the **Main** method, add the following line of code to render the Uniform Resource Identifier (URI) of the queue endpoint:

```
Console.WriteLine($"Account Uri:\t{client.Uri}");
```

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

```
public static async Task Main(string[] args)
{
    QueueClient client = new QueueClient(storageConnectionString, queueName);

    await client.CreateAsync();
    Console.WriteLine($"---Account Metadata---");
    Console.WriteLine($"Account Uri:\t{client.Uri}");

}
```

6. Save the **Program.cs** file.

In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

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

```
dotnet run
```

8. **Note:** If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.
9. Observe the output from the currently running console application. The output contains metadata for the queue endpoint.
10. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

### 3.4 Review

1. In this exercise, you configured your .NET project to access the Storage service and manipulate a queue made available through the service.

## 4 Exercise 3: Read messages from the queue

### 4.1 Task 1: Write code to access queue messages

1. In the **Main** method, add the following line of code to render a header for the “Existing Messages” section:

```
Console.WriteLine($"---Existing Messages---");
```

2. Within the **Main** method, perform the following actions to create variables that will be used when retrieving queue messages:
  1. Add the following line of code to create a variable of type **int** named *batchSize* with a value of **10**:

```
int batchSize = 10;
```

2. Add the following line of code to create a variable of type **TimeSpan** named *visibilityTimeout* with a value of 2.5 seconds:

```
TimeSpan visibilityTimeout = TimeSpan.FromSeconds(2.5d);
```

3. Within the **Main** method, perform the following actions to retrieve a batch of messages asynchronously from the queue service:
4. Add the following line of code to invoke the **ReceiveMessagesAsync** asynchronous method of the **QueueClient** class, passing in the **batchSize** and **visibilityTimeout** variables as parameters:

```
client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
```

5. Update the previous line of code by adding more code to process the expression asynchronously by using the **await** keyword:

```
await client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
```

6. Update the previous line of code by adding more code to store the result of the expression in a new variable named *messages* of type **Response<QueueMessage[]>**:

```
Response<QueueMessage[]> messages = await client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
```

7. Within the **Main** method, perform the following actions to iterate over and render the properties of each message:
8. Add the following line of code to create a **foreach** loop that iterates over each message that's stored in the **Value** property of the *messages* variable of type **QueueMessage[]**:

```
foreach(QueueMessage message in messages?.Value)  
{
```

```
}
```

9. Within the foreach loop, add another line of code to render the `MessageId` and `MessageText` properties of each `QueueMessage` instance:

```
Console.WriteLine($"{message.MessageId}]\t{message.MessageText}");
```

10. Observe the `Main` method, which should now include:

```
public static async Task Main(string[] args)
{
    QueueClient client = new QueueClient(storageConnectionString, queueName);
    await client.CreateAsync();
    Console.WriteLine($"---Account Metadata---");
    Console.WriteLine($"Account Uri:\t{client.Uri}");
    Console.WriteLine($"---Existing Messages---");
    int batchSize = 10;
    TimeSpan visibilityTimeout = TimeSpan.FromSeconds(2.5d);
    await client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
    Response<QueueMessage[]> messages = await client.ReceiveMessagesAsync(batchSize, visibilityTimeout);
    foreach(QueueMessage message in messages?.Value)
    {
        Console.WriteLine($"{message.MessageId}]\t{message.MessageText}");
    }
}
```

11. Save the `Program.cs` file.

In the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.

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

**dotnet build**

**Note:** If there are any build errors, review the `Program.cs` file in the `Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor` folder.

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

## 4.2 Task 2: Test message queue access

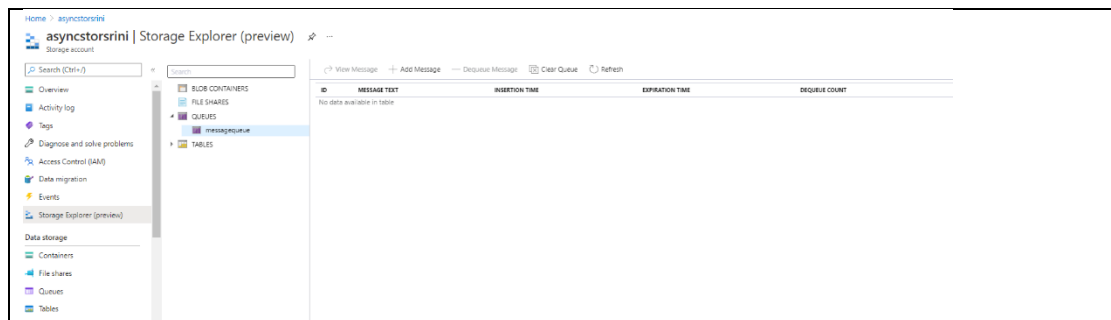
1. In the Visual Studio Code window, right-click or activate the shortcut menu for the Explorer pane, and then select Open in Terminal.

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

**dotnet run**

**Note:** If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

3. Observe the output from the currently running console application. The output indicates that no messages are in the queue.
4. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.
5. In the Azure portal's navigation pane, select the **Resource groups** link.
6. On the **Resource groups** blade, find and then select the **AsyncProcessor** resource group that you created earlier in this lab.
7. On the **AsyncProcessor** blade, select the **asyncstor[yourname]** storage account that you created earlier in this lab.
8. On the **Storage account** blade, select **Overview**.
9. In the **Overview** section, select **Open in Explorer**.
10. In the **Azure Storage Explorer** window, select **Open Azure Storage Explorer**.



**Note:** If this is your first time opening Storage Explorer by using the portal, you might be prompted to allow the portal to open these types of links in the future. You should accept the prompt.

In the **Azure Storage Explorer** application, you will notice a prompt to sign in to your Azure account. Sign in by performing the following actions:

1. In the popup dialog, select **Sign in**.
2. In the **Connect to Azure Storage** window, select **Add an Azure Account**, in the **Azure environment** list select **Azure**, and then select **Next**.
3. In the **Sign in to your account** popup window, enter the email address for your Microsoft account, and then select **Next**.
4. Still within the **Sign in to your account** popup window, enter the password for your Microsoft account, and then select **Sign in**.
5. In the **ACCOUNT MANAGEMENT** pane, select **Apply**.
6. Observe that you are returned back to the **EXPLORER** pane with your subscription information populated.
7. From the **Azure Storage Explorer** application, in the **EXPLORER** pane, find and expand the **asyncstor[yourname]** storage account that you created earlier in this lab.
8. Within the **asyncstor[yourname]** storage account, find and expand the **Queues** node.
9. In the **Queues** node, open the **messagequeue** queue that you created earlier in this lab by using .NET code.
10. On the **messagequeue** tab, select **Add Message**.

In the **Add Message** pop-up window, perform the following actions:

1. In the **Message text** text box, enter the value **Hello World**.
2. In the **Expires in** text box, enter the value **12**.
3. In the **Expires in** drop-down list, select **Hours**.
4. In the **Encoding** section, select **UTF8**.
5. Select **OK**.
6. Return to the **Visual Studio Code** window, right-click or activate the shortcut menu for the Explorer pane, and then select **Open in Terminal**.
7. At the open command prompt, enter the following command, and then select Enter to run the .NET web application:

```
dotnet run
```

18. **Note:** If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.
19. Observe the output from the currently running console application. The output includes the new message that you created.
20. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

### 4.3 Task 3: Delete queued messages

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, find the existing **foreach** loop within the **Main** method:

```
foreach(QueueMessage message in messages?.Value)
{
    Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
}
```

3. Within the **foreach** loop, add a new line of code to invoke the **DeleteMessageAsync** method of the **QueueMessage** class, passing in the **MessageId** and **PopReceipt** properties of the *message* variable:

```
await client.DeleteMessageAsync(message.MessageId, message.PopReceipt);
```

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

```
foreach(QueueMessage message in messages?.Value)
{
    Console.WriteLine($"[{message.MessageId}]\t{message.MessageText}");
    await client.DeleteMessageAsync(message.MessageId, message.PopReceipt);
}
```

5. **Save** the **Program.cs** file.

6. In the Visual Studio Code window, right-click or activate the shortcut menu for the Explorer pane, and then select Open in Terminal.

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

**dotnet run**

**Note:** If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.

7. Observe the output from the currently running console application. The message that you created earlier in the lab still exists because it hasn't been deleted previously.
8. Select Kill Terminal or the Recycle Bin icon to close the currently open terminal and any associated processes.
9. Return Storage Explorer, and then find and expand the **asyncstor[yourname]** storage account that you created earlier in this lab.
10. In the **asyncstor[yourname]** storage account, find and expand the Queues node.
11. In the Queues node, open the **messagequeue** queue that you created earlier in this lab by using .NET code.
12. Observe the empty list of messages in the queue.

**Note:** You might need to refresh the queue.

## 4.4 Review

In this exercise, you read and deleted existing messages from the Storage queue by using the .NET library.

## 5 Exercise 4: Queue new messages by using .NET

### 5.1 Task 1: Write code to create queue messages

1. In the Explorer pane of the **Visual Studio Code** window, open the **Program.cs** file.
2. On the code editor tab for the **Program.cs** file, find the existing **Main** method.
3. Within the **Main** method, add a new line of code to render a header for the “New Messages” section:

```
Console.WriteLine($"---New Messages---");
```

In the **Main** method, perform the following actions to create and send a message asynchronously:

4. Add the following line of code to create a new string variable named *greeting*

```
string greeting = "Hello Queued Storage!" + DateTime.Now.ToString();
```

5. Add the following line of code to invoke the **SendMessageAsync** method of the **QueueClient** class by using the *greeting* variable as a parameter

```
await client.SendMessageAsync(Convert.ToBase64String(Encoding.UTF8.GetBytes(greeting)));
```

6. Add the following line of code to render the content of the message that you sent:

```
Console.WriteLine($"Sent Message:\t{greeting}");
```

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

**dotnet run**

10. **Note:** If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\10\Solution\MessageProcessor** folder.
11. Observe the output from the currently running console application. The content of the new message that you sent should be in the output.
12. Select **Kill Terminal** or the **Recycle Bin** icon to close the currently open terminal and any associated processes.

### 5.2 Task 2: View queued messages by using Storage Explorer

1. Return to Storage Explorer, and then find and expand the **asyncstor[yourname]** storage account that you created earlier in this lab.
2. In the **asyncstor[yourname]** storage account, find and expand the **Queues** node.
3. In the **Queues** node, open the **messagequeue** queue that you created earlier in this lab by using .NET code.
4. Observe the single new message in the list of messages in the queue.

**Note:** You might need to refresh the queue.

### 5.3 Review

In this exercise, you created new messages in the queue by using the .NET library for Storage queues.



## 6 Exercise 5: Clean up your subscription

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

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

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

2. If this is your first time opening Cloud Shell using your subscription, you can use the **Welcome to Azure Cloud Shell Wizard** to configure Cloud Shell for first-time usage. Perform the following actions in the wizard:
  - A dialog box prompts you to configure the shell. Select **Bash**, review the selected subscription, and then select **Create storage**.

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

### 6.2 Task 2: Delete a resource group

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

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

2. Close the Cloud Shell pane in the portal.

### 6.3 Task 3: Close the active application

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.
3. Close the currently running Azure Storage Explorer application.

### 6.4 Review

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