Do not click nested Typetext icons. Doing so will produce the wrong output.

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

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

}
```

Please skip **Exercise 1, Task 3**. The resource provider is already registered.

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.

Lab: Publishing and subscribing to Event Grid events

Student lab manual

Lab scenario

Your company builds a human resources (HR) system used by various customers around the world. While the system works fine today, your development managers have decided to begin re-architecting the solution by decoupling application components. This decision was driven by a desire to make any future development simpler through modularity. As the developer who manages component communication, you have decided to introduce Microsoft Azure Event Grid as your solution-wide messaging platform.

Objectives

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

- Create an Event Grid topic.
- Use the Azure Event Grid viewer to subscribe to a topic and illustrate published messages.
- Publish a message from a .NET application.

Lab setup

- Estimated time: **45 minutes**

Instructions

Before you start

Sign in to the lab virtual machine

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

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

Review the installed applications

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

- Microsoft Edge
- Microsoft Visual Studio Code

Exercise 1: Create Azure resources

Task 1: Open the Azure portal

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

Task 2: Open Azure Cloud Shell

1. Open a new Cloud Shell instance in the Azure portal.
2. If Cloud Shell isn't already configured, configure the shell for Bash by using the default settings.
3. At the **Cloud Shell** command prompt in the portal, use the **az** command with the **--version** flag to get the version of the Azure Command-Line Interface (Azure CLI) tool.

Task 3: View the Microsoft.EventGrid provider registration

1. Use the **az** command with the **--help** flag to find a list of subgroups and commands at the root level of the Azure CLI.
2. Use the **az provider** command with the **--help** flag to get a list of commands available for resource providers.
3. Use the **az provider list** command to get a list of all currently registered providers.
4. Use the **az provider list** command again with the **--query "[].namespace"** flag to list just the namespaces of the currently registered providers.
5. Review the list of currently registered providers. Note that the **Microsoft.EventGrid** provider is currently in the list of providers.
6. Close the Cloud Shell pane.

Task 4: Create a custom Event Grid topic

1. Create a new Event Grid topic with the following details:
    - Name: **hrtopic*[yourname]***
    - New resource group: **PubSubEvents**
    - Location: **East US**
    - Event Schema: **Event Grid Schema**

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

Task 5: Deploy the Azure Event Grid viewer to a web app

1. Create a new web app with the following details:
    - Existing resource group: **PubSubEvents**
    - Name: **eventviewer*[yourname]***
    - Publish: **Docker Container**
    - Operating system: **Linux**
    - Region: **East US**
    - New App Service plan: **EventPlan**
    - SKU and size: **Premium V2 P1v2**
    - Docker options: **Single Container**
    - Image source: **Docker Hub**
    - Access type: **Public**
    - Image and tag: **microsoftlearning/azure-event-grid-viewer:latest**

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

Review

In this exercise, you created the Event Grid topic and web app that you will use throughout the remainder of the lab.

Exercise 2: Create an Event Grid subscription

Task 1: Access the Event Grid Viewer web application

1. Access the **eventviewer*[yourname]*** web app that you created earlier in this lab.
2. In the **Settings** section, go to the **Properties** section, and then record the value in the **URL** text box. You'll use this value later in the lab.
3. Browse to the currently running web app.
4. Observe the currently running **Azure Event Grid viewer** web application. Leave this web application running for the remainder of the lab.

    Note: This web application will update in real-time as events are sent to its endpoint. We will use this to monitor events throughout the lab.

5. Return to the Azure portal.

Task 2: Create new subscription

1. Access the **hrtopic*[yourname]*** Event Grid topic that you created earlier in this lab.

2. Create a new **Event Subscription** with the following details:
    - Name: **basicsub**
    - Event Schema: **Event Grid Schema**
    - Endpoint Type: **Web Hook**
    - Endpoint: ***Web App URL recorded earlier in the lab, with an *https://* prefix and an */api/updates* suffix**

      Note: For example, if your **Web App URL** value is **http://eventviewerstudent.azurewebsites.net/**, then your endpoint would be **https://eventviewerstudent.azurewebsites.net/api/updates**.

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

Task 3: Observe the subscription validation event

1. Return to Azure Event Grid viewer.
2. Review the **Microsoft.EventGrid.SubscriptionValidationEvent** event that was created as part of the subscription creation process.
3. Select the event and review its JSON content.
4. Return to Azure portal.

Task 4: Record subscription credentials

1. Access the **hrtopic*[yourname]*** Event Grid topic that you created earlier in this lab.
2. Record the value of the **Topic Endpoint** field. You'll use this value later in the lab.
3. In the **Settings** section, go to the **Access keys** section, and then record the value in the **Key 1** text box. You'll use this value later in the lab.

Review

In this exercise, you created a new subscription, validated its registration, and then recorded the credentials required to publish a new event to the topic.

Exercise 3: Publish Event Grid events from .NET

Task 1: Create .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\10\Starter\EventPublisher**folder.
2. Using a terminal, create a new .NET project named **EventPublisher** in the current folder:

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

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

3. Using the same terminal, import version 3.2.0 of **Microsoft.Azure.EventGrid** from NuGet:

```
dotnet add package Microsoft.Azure.EventGrid --version 3.2.0
```

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

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

```
dotnet build
```

5. Close the current terminal.

Task 2: Modify the Program class to connect to Event Grid

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

```
using Microsoft.Azure.EventGrid;
using Microsoft.Azure.EventGrid.Models;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

4. Create a new **Program** class with two constant string properties named **topicEndpoint** and **topicKey**, and then create an asynchronous **Main** entry point method:

```
public class Program
{
    private const string topicEndpoint = "";
    private const string topicKey = "";

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

}
```

5. Update the **topicEndpoint** string constant by setting its value to the **Topic Endpoint** of the Event Grid topic that you recorded earlier in this lab.
6. Update the **topicKey** string constant by setting its value to the **Key** of the Event Grid topic that you recorded earlier in this lab.

Task 3: Publish new events

1. In the **Main** method, perform the following actions:

    a. Add the following block of code to connect to the Event Grid using the credentials you specified earlier in the lab:

```
TopicCredentials credentials = new TopicCredentials(topicKey);
EventGridClient client = new EventGridClient(credentials);
```

b. Create a new variable named **events**, of type **List**:

```
List<EventGridEvent> events = new List<EventGridEvent>();
```

c. Add the following block of code to: create two new variables named **firstPerson** of an anonymous type, and **firstEvent** of type **EventGridEvent**; populate the **EventGridEvent**variable with sample data; and add the **firstEvent** instance to your **events** list:

```
var firstPerson = new
{
    FullName = "Alba Sutton",
    Address = "4567 Pine Avenue, Edison, WA 97202"
};

EventGridEvent firstEvent = new EventGridEvent
{
    Id = Guid.NewGuid().ToString(),
    EventType = "Employees.Registration.New",
    EventTime = DateTime.Now,
    Subject = $"New Employee: {firstPerson.FullName}",
    Data = firstPerson.ToString(),
    DataVersion = "1.0.0"
};
events.Add(firstEvent);
```

d. Add the following block of code to: create two new variables named **secondPerson** of an anonymous type, and **secondEvent** of type **EventGridEvent**; populate the **EventGridEvent** variable with sample data; and add the **secondEvent** instance to your **events** list:

```
var secondPerson = new
{
    FullName = "Alexandre Doyon",
    Address = "456 College Street, Bow, WA 98107"
};

EventGridEvent secondEvent = new EventGridEvent
{
    Id = Guid.NewGuid().ToString(),
    EventType = "Employees.Registration.New",
    EventTime = DateTime.Now,
    Subject = $"New Employee: {secondPerson.FullName}",
    Data = secondPerson.ToString(),
    DataVersion = "1.0.0"
};
events.Add(secondEvent);
```

e. Add the following block of code to obtain the **Hostname** from the **topicEndpoint** variable, and then use that hostname as a parameter to the **EventGridClient.PublishEventsAsync**method

invocation:

```
string topicHostname = new Uri(topicEndpoint).Host;
await client.PublishEventsAsync(topicHostname, events);
```

f. Render the **Events published** message to the console:

```
Console.WriteLine("Events published");
```

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

```
dotnet run
```

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

4. Review the success message output from the currently running console application.
5. Close the current terminal.

Task 4: Observe published events

1. Return to the browser window with the **Azure Event Grid viewer** web application.
2. Review the **Employees.Registration.New** events that were created by your console application.
3. Select any of the events and review its JSON content.
4. Return to Azure portal.

Review

In this exercise, you published new events to your Event Grid topic using a .NET console application.

Exercise 4: Clean up your subscription

Task 1: Open Azure Cloud Shell

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

Task 2: Delete resource groups

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

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

2. Close the Cloud Shell pane.

Task 3: Close the active applications

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

Review

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

Congratulations!

You have successfully completed this exercise. Click **End** to advance to the next exercise.