0809 Azure Message queue sandbox Hands-on Lab

▼ **Create a Service Bus queue and topic**

Service Bus 큐 및 토픽 만들기

글로벌 기업의 영업 팀에서 특정 애플리케이션을 사용합니다. 각 팀 멤버는 영업에 사용할 모바일 디바이스에 앱이 설치되어 있습니다. Azure 웹 서비스는 애플리케이션에 대해 구현된 비즈니스 논리를 호스트하고 Azure SQL Database에 정보를 저장합니다. 각 지역에 웹 서비스의 자체 인스턴스가 있습니다.

모바일 앱과 웹 서비스 간의 메시지 교환에 대한 다음 시나리오를 확인했습니다.

- *개별 판매와 관련된 메시지는 사용자 지역의 웹 서비스 인스턴스로만 전송해야 합니다.*
- *영업 성과 관련 메시지는 웹 서비스의 모든 인스턴스로 전송해야 합니다.*

> **첫 번째 사용 사례에는 Service Bus 큐를 구현**하고, **두 번째 사용 사례에는 Service Bus 토픽을 구현**하기로 했습니다.

이 연습에서는 Azure Portal에서 **큐, 토픽 및 구독이 포함된 Service Bus 네임스페이스**를 만듭니다.

Service Bus 네임스페이스 만들기

네임스페이스를 만드는 것으로 시작합니다. **Azure Service Bus**에서 `네임스페이스`는 큐 및 토픽용 컨테이너입니다. **각 네임스페이스에는 기본 및 보조 SAS(공유 액세스 서명) 암호화 키와 함께 고유한 정규화된 도메인 이름**이 있습니다. 전송 또는 수신 구성 요소는 네임스페이스의 개체에 액세스할 수 있는 **SAS 키를 제공**해야 합니다.

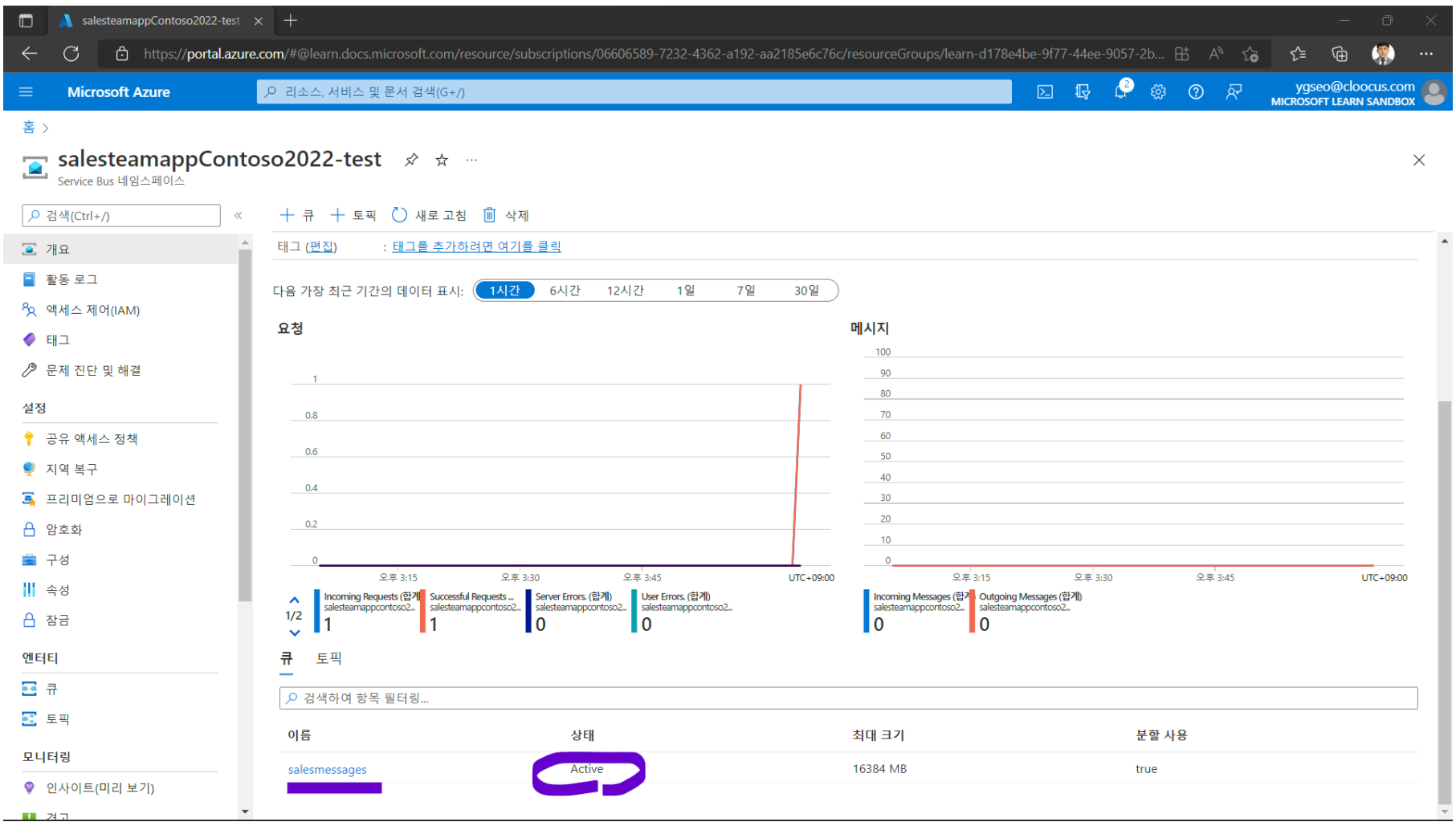
1. 샌드박스를 활성화하는 데 사용한 것과 동일한 자격 증명으로 Azure Portal에 로그인합니다.
2. **Azure 서비스**에서 **리소스 만들기**를 선택합니다.
3. **리소스 만들기** 창에서 전역 검색에 **Service Bus**를 입력합니다. 검색 결과에서, Microsoft에서 Azure용으로 게시한 **Service Bus**에서 **만들기**를 선택합니다.
4. **네임스페이스 만들기** 창의 **기본** 탭에서 각 설정마다 다음 값을 입력하거나 선택합니다.

Service Bus 큐 만들기

그 다음으로 개별 판매량에 대한 메시지 큐를 네임스페이스에 추가합니다.

1. 배포가 완료되면 **리소스로 이동**을 선택합니다.
2. **Service Bus 네임스페이스** 페이지에서 왼쪽 메뉴의 **엔터티**에서 **큐**를 선택합니다.
3. 명령 모음에서 **+ 큐**를 선택합니다.
4. **큐 만들기** 창에서 **이름**에 **salesmessages**를 입력한 다음 **만들기**를 선택합니다.

메시지 큐가 만들어지면 Service Bus 네임스페이스 창 하단에 있는 **큐** 아래에 **salesmessages**가 나열됩니다.



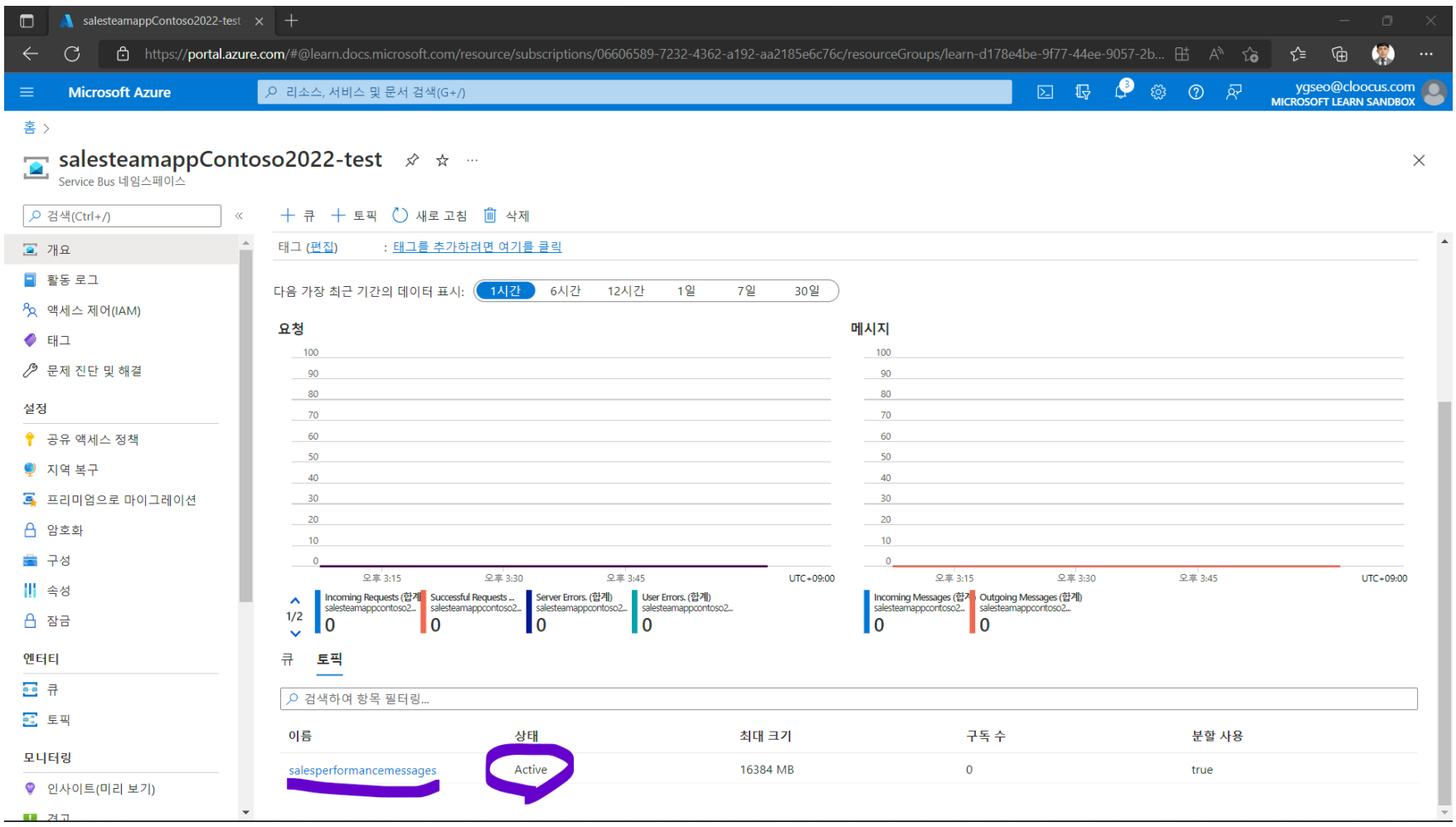
Service Bus 항목과 구독 만들기

영업 성과 관련 메시지에 사용할 항목도 만들려고 합니다. 비즈니스 논리 웹 서비스의 각 인스턴스가 이 토픽을 구독하고, 각 영업 실적 메시지가 모든 웹 서비스 구독에 배달됩니다.

Service Bus 토픽과 구독을 추가하려면 다음을 수행합니다.

1. 왼쪽 메뉴에서 **엔터티** 아래에 있는 **토픽**을 선택한 다음, 명령 모음에서 **+ 토픽**을 선택합니다.
2. **토픽 만들기** 창에서 **이름**에 *salesperformancemessages*를 입력한 다음 **만들기**를 선택합니다.

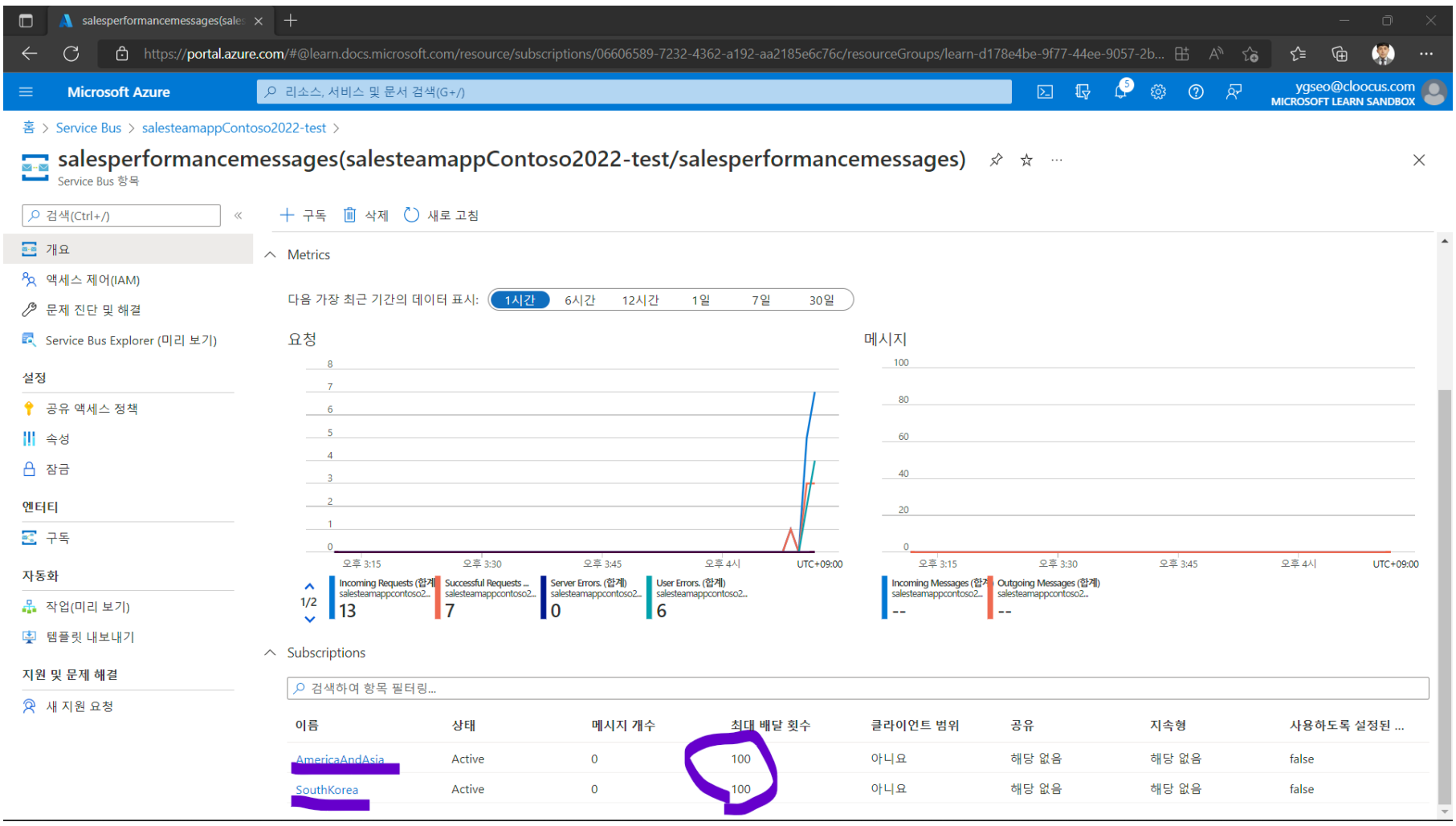
   토픽이 만들어지면 Service Bus 네임스페이스 창 하단에 있는 **토픽** 아래에 **salesperformancemessages**가 나열됩니다.



3. Service Bus 네임스페이스의 탭을 사용하여 큐 및 토픽에서 구독을 추가 또는 제거하거나 Azure Portal 리소스 메뉴를 사용할 수 있습니다. 메뉴 옵션을 사용하려면 왼쪽 메뉴의 **엔터티**에서 **토픽**을 선택한 다음, 토픽 목록에서 **salesperformancemessages**를 선택합니다.
4. **salesperformancemessages** Service Bus 토픽 창의 명령 모음에서 **+ 구독**을 선택합니다.
5. **구독 만들기** 창에서 **이름**에 **Americas**를 입력합니다. **최대 전송 횟수**에 **100**을 입력합니다. **만들기**를 선택합니다.

   salesperformancemessages Service Bus 토픽이 나타나고, **Americas** 구독이 창 하단의 **구독** 섹션에 나열됩니다.
6. 다음으로 두 번째 구독을 추가합니다. 명령 모음에서 **구독**을 선택합니다.
7. **구독 만들기** 창에서 **이름**에 **EuropeAndAsia**를 입력합니다. **최대 전송 횟수**에 **100**을 입력합니다. **만들기**를 선택합니다.

*영업 팀 앱의 **salesperformancemessages** Service Bus 토픽에서 **구독** 섹션에 구독 두 개가 나열됩니다.*



분산 애플리케이션의 복원력을 높이기 위해 Service Bus를 사용하도록 인프라를 빌드했습니다. 그리고 개별 판매 관련 메시지용 큐와 영업 성과 관련 메시지용 항목을 만들었습니다. 토픽 메시지를 전 세계 여러 웹 서비스에 배달할 수 있도록 토픽에 여러 구독을 추가했습니다.

▼ **Write code to send and receive messages by using a queue**

큐를 사용해 메시지를 보내고 받는 코드 작성

분산 애플리케이션은 대상 구성 요소로 배달 대기 중인 메시지의 임시 스토리지 위치로 Service Bus 큐를 사용합니다. 큐를 통해 메시지를 전송하고 수신하려면 원본 구성 요소와 대상 구성 요소 모두에서 코드를 작성해야 합니다.

Contoso Bicycles 애플리케이션의 예를 고려해 보겠습니다. 고객은 웹사이트 또는 모바일 앱을 통해 주문을 할 수 있습니다. 웹 사이트와 모바일 앱은 고객 디바이스에서 실행되므로 동시에 접수될 수 있는 주문 수에는 실제로 제한이 없습니다. 모바일 앱과 웹사이트가 Service Bus 큐에 주문을 보관하도록 하여 백 엔드 구성 요소(웹 앱)는 해당 큐에서 적절한 속도로 주문을 처리할 수 있습니다.

실제로 Contoso Bicycles 애플리케이션은 몇 단계를 거쳐 새 주문을 처리합니다. 모든 단계를 수행하려면 먼저 결제가 승인되어야 하므로 큐를 사용하도록 하겠습니다. 수신 구성 요소에서는 먼저 결제를 처리합니다.

Contoso는 모바일 앱과 웹 사이트에서 메시지를 큐에 추가하는 코드를 작성해야 합니다. 백 엔드 웹앱에서 Contoso는 메시지를 큐에서 선택하는 코드를 작성합니다.

여기서는 Service Bus 큐를 사용하여 메시지를 보내고 받기 위한 코드 작성 프로세스 및 고려 사항을 살펴보겠습니다.

Azure.Messaging.ServiceBus NuGet 패키지

Microsoft에서는 Service Bus를 통해 메시지를 전송하고 수신하는 코드를 쉽게 작성할 수 있도록 .NET 클래스 라이브러리를 제공합니다. 이 라이브러리는 모든 .NET 언어에서 Service Bus 큐 또는 토픽을 조작하는 데 사용할 수 있습니다. **Azure.Messaging.ServiceBus** NuGet 패키지를 추가하여 애플리케이션에 이 라이브러리를 포함할 수 있습니다.

연결 문자열 및 키

Service Bus 네임스페이스에서 큐에 연결하려면 원본 구성 요소와 대상 구성 요소 둘 다에 두 가지 정보가 필요합니다.

- Service Bus 네임스페이스 위치(*엔드포인트*라고도 함). 이 위치는 **servicebus.windows.net** 도메인 내에서 정규화된 도메인 이름으로 지정됩니다. 예를 들면 **bicycleService.servicebus.windows.net**과 같습니다.
- 액세스 키. Service Bus는 유효한 액세스 키를 요구하여 큐 및 토픽에 대한 액세스를 제한합니다.

이 두 정보는 모두 연결 문자열 형식으로 ServiceBusClient 개체에 제공됩니다. 네임스페이스에 대한 올바른 연결 문자열은 Azure Portal에서 가져올 수 있습니다.

비동기식 메서드 호출

Azure의 큐는 전송 및 수신 구성 요소에서 매우 멀리 떨어져 있을 수도 있습니다. 그리고 실제 위치가 가깝더라도 느린 연결 및 대역폭 경합으로 인해 구성 요소가 큐에서 메서드를 호출하는 경우 지연이 발생할 수 있습니다. 이러한 이유로 인해 Service Bus 클라이언트 라이브러리는 큐와 상호 작용할 수 있도록 `async` 메서드를 제공합니다. 호출이 완료되기를 기다리는 동안 스레드가 차단되지 않도록 이러한 메서드를 사용하겠습니다.

예를 들어 큐에 메시지를 보낼 때는 `await` 키워드가 포함된 SendMessageAsync 메서드를 사용합니다.

큐에 메시지 보내기

큐에 메시지를 보내려면 다음 단계를 완료합니다.

모든 전송 또는 수신 구성 요소에서 Service Bus 큐를 호출하는 모든 코드 파일에 다음의 `using` 문을 추가합니다.

```
using System.Threading;
using System.Threading.Tasks;
using Azure.Messaging.ServiceBus;
```

그런 다음, 새 `ServiceBusClient` 개체를 만들고 이 개체에 연결 문자열과 큐 이름을 전달합니다.

```
// Create a ServiceBusClient object using the connection string to the namespace.
await using var client = new ServiceBusClient(connectionString);

// Create a ServiceBusSender object by invoking the CreateSender method on the ServiceBusClient object, and specifying the queue na
me.
ServiceBusSender sender = client.CreateSender(queueName);
```

`ServiceBusSender.SendMessageAsync()` 메서드를 호출하고 `ServiceBusMessage`를 전달하면 큐에 메시지를 전송할 수 있습니다.

```
// Create a new message to send to the queue.
string messageContent = "Order new crankshaft for eBike.";
var message = new ServiceBusMessage(messageContent);

// Send the message to the queue.
await sender.SendMessageAsync(message);
```

큐에서 메시지 받기

메시지를 받으려면 먼저 메시지 처리기를 등록해야 합니다. 메시지 처리기는 큐에서 메시지를 사용할 수 있을 때 호출되는 코드 내 메서드입니다.

```
// Create a ServiceBusProcessor for the queue.
await using ServiceBusProcessor processor = client.CreateProcessor(queueName, options);

// Specify handler methods for messages and errors.
processor.ProcessMessageAsync += MessageHandler;
processor.ProcessErrorAsync += ErrorHandler;
```

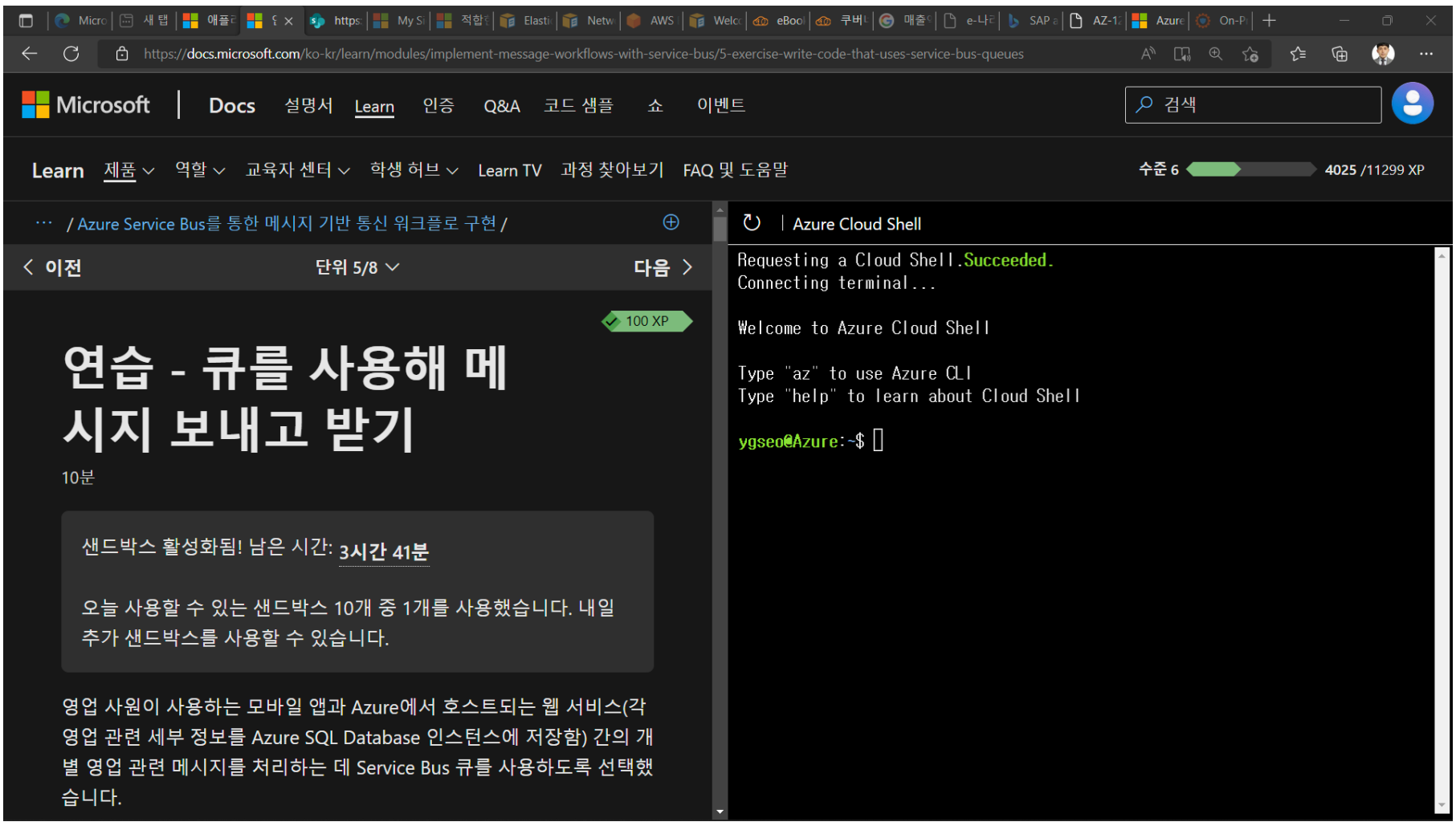
처리 작업을 수행합니다. 그런 다음, 메시지 처리기 내에서 `ProcessMessageEventArgs.CompleteMessageAsync()` 메서드를 호출하여 큐에서 메시지를 제거합니다.

```
await args.CompleteMessageAsync(args.Message);
```

▼ **Exercise - Send and receive messages by using a queue** ← *원격의 큐 .NET 버전(3.1.0)과 호환이 안되어 .NET 구동 명령의 옵션, 'roll-foward'(인자값 Major) 추가하여 메시지 passing 완료함*

연습 - 큐를 사용해 메시지 보내고 받기

영업 사원이 사용하는 모바일 앱과 Azure에서 호스트되는 웹 서비스(각 영업 관련 세부 정보를 Azure SQL Database 인스턴스에 저장함) 간의 개별 영업 관련 메시지를 처리하는 데 Service Bus 큐를 사용하도록 선택했습니다.



이거 아시나요> Docs Learn을 통해 AZ-305: 인프라 솔루션 디자인 과정(애플리케이션 아키텍처 설계 - Learn | Microsoft Docs) 이수하면 AZ-305 응시료 50% 할인 된다는거요? 😒

이전 연습에서는 Azure 구독에서 필요한 개체를 구현했습니다. 이제 메시지를 해당 큐로 보내고 메시지를 검색하는 코드를 작성하려고 합니다.

이 단원에서는 두 가지 콘솔 애플리케이션을 빌드하는데 하나는 Service Bus 큐에 메시지를 넣고, 다른 하나는 Serivce Bus 큐에서 메시지를 검색합니다. 이러한 애플리케이션은 단일 .NET Core 솔루션의 일부분입니다.

Service Bus 네임스페이스에 대한 연결 문자열을 가져옵니다.

Service Bus 네임스페이스에 액세스하고 해당 네임스페이스 내에서 큐를 사용하려면 두 콘솔 앱에서 다음과 같은 두 가지 정보를 구성해야 합니다.

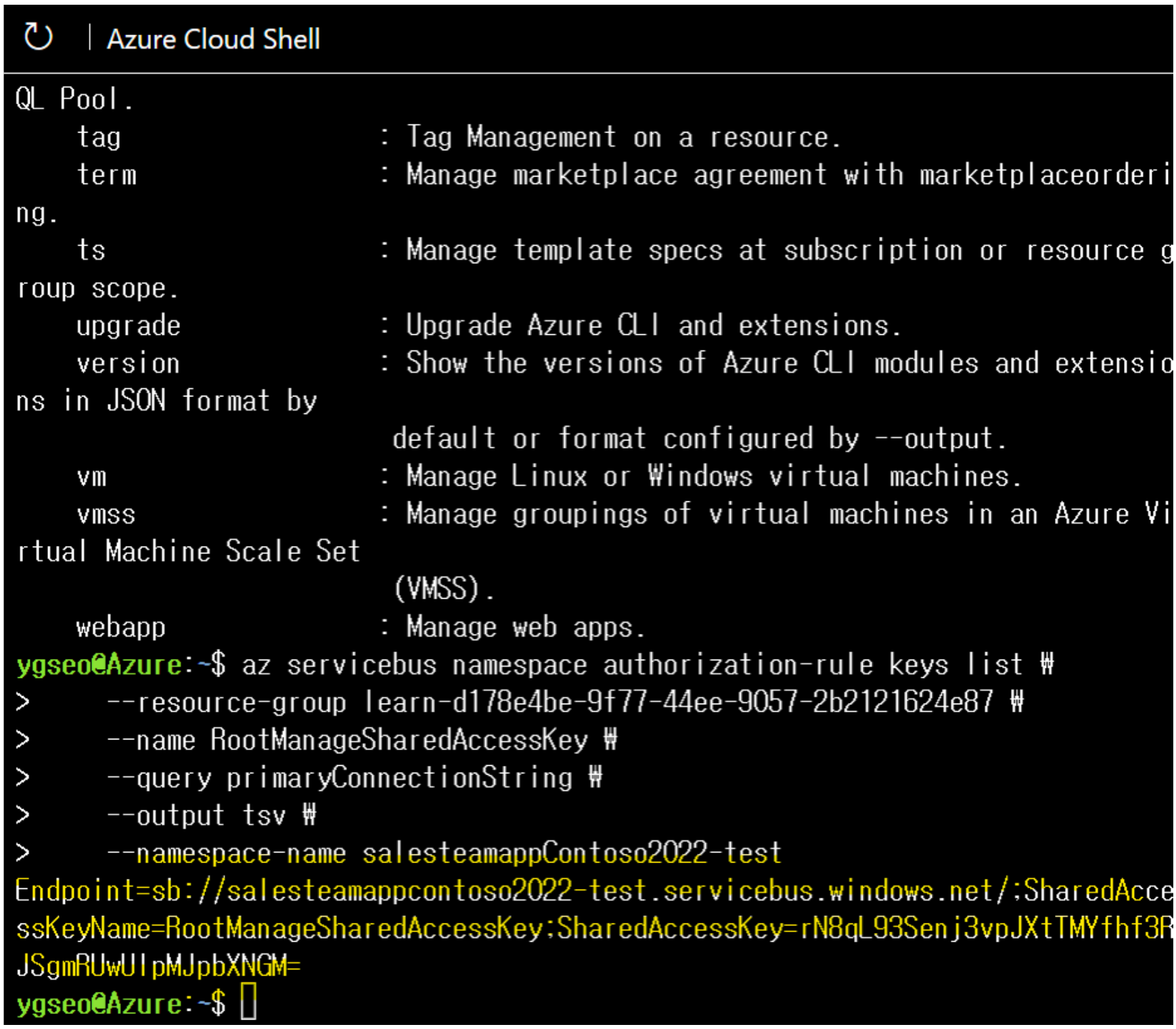
- 네임스페이스의 엔드포인트
- 인증용 공유 액세스 키

연결 문자열에서 이러한 값을 가져올 수 있습니다.

1. Azure Cloud Shell에서 다음 명령을 실행하여 `<namespace-name>`을 마지막 연습에서 만든 Service Bus 네임스페이스로 바꿉니다.

```
az servicebus namespace authorization-rule keys list \
    --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 \
    --name RootManageSharedAccessKey \
    --query primaryConnectionString \
    --output tsv \
    --namespace-name <namespace-name>
```

응답의 마지막 줄이 네임스페이스에 대한 엔드포인트와 공유 액세스 키를 포함하는 연결 문자열입니다. 이는 다음 예와 유사합니다.



2. Cloud Shell에서 연결 문자열을 복사합니다. 모듈 전체에서 이 연결 문자열이 여러 번 필요하므로 어딘가에 붙여넣고 간편하게 사용하는 것이 좋습니다.

시작 애플리케이션 복제 및 열기

참고

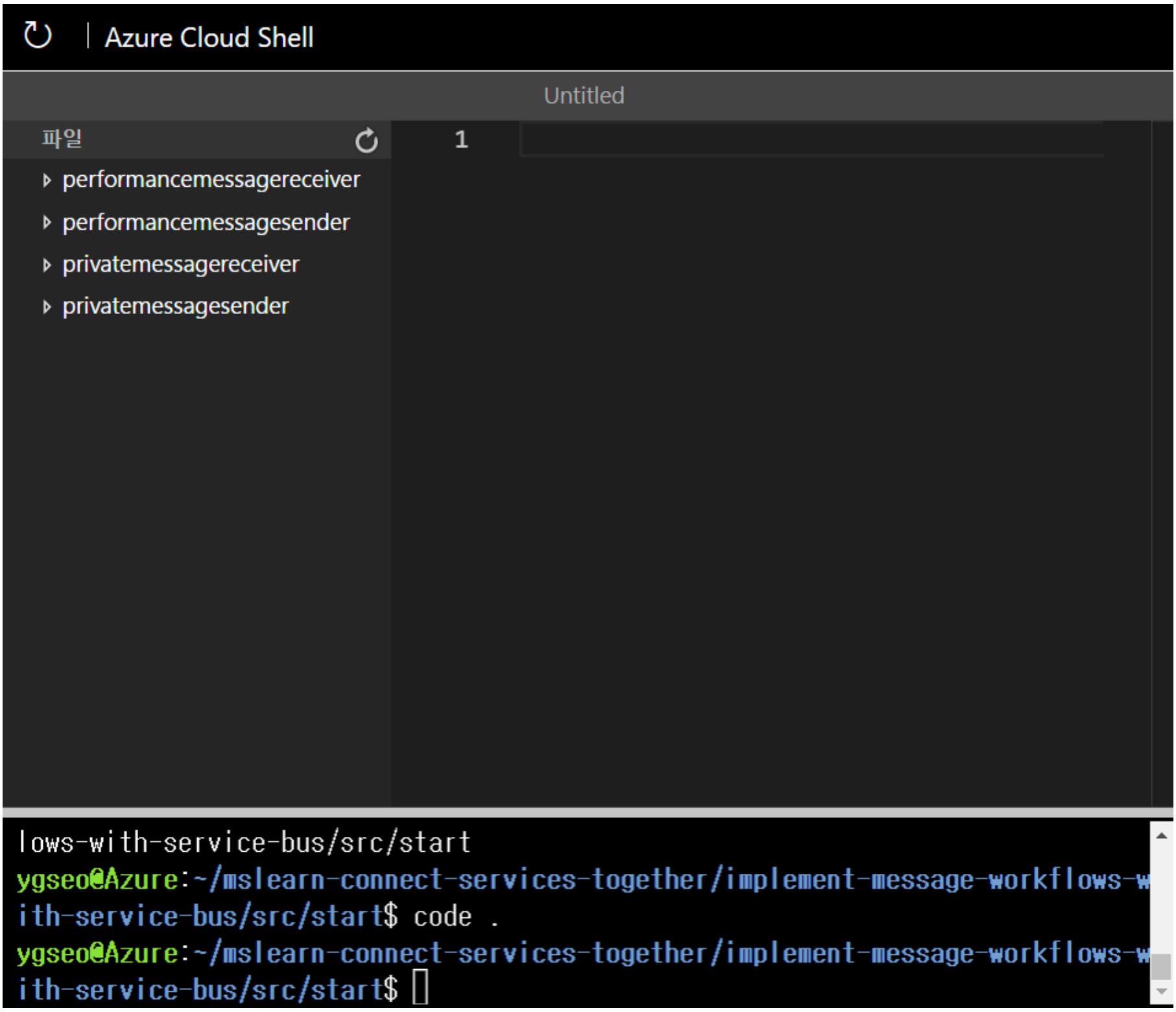
간소화할 수 있도록 다음 작업에서는 두 콘솔 애플리케이션의 *Program.cs* 파일에서 연결 문자열을 하드 코딩하는 방법을 설명합니다. 프로덕션 애플리케이션에서는 구성 파일 또는 Azure Key Vault를 사용하여 연결 문자열을 저장할 수 있습니다.

1. Cloud Shell에서 다음 명령을 실행하여 Git 프로젝트 솔루션을 복제합니다.

```
cd ~
git clone https://github.com/MicrosoftDocs/mslearn-connect-services-together.git
```

2. 다음 명령을 실행하여 복제된 프로젝트의 시작 폴더로 이동하고 Cloud Shell 편집기를 엽니다.

```
cd ~/mslearn-connect-services-together/implement-message-workflows-with-service-bus/src/start
code .
```



큐에 메시지를 보내는 코드 작성

1. Cloud Shell 편집기에서 *privatemessagesender/Program.cs*를 열고 다음 코드 줄을 찾습니다.

   ```
   const string ServiceBusConnectionString = "";
   ```

   연결 문자열을 따옴표 사이에 붙여넣습니다.

2. *privatemessagesender/Program.cs*의 최종 코드가 다음 예제와 유사한지 확인합니다.

   ```
   using System;
   using System.Text;
   using System.Threading.Tasks;
   using Azure.Messaging.ServiceBus;

   namespace privatemessagesender
   {
       class Program
       {
           const string ServiceBusConnectionString = "Endpoint=sb://example.servicebus.windows.net/;SharedAccessKeyName=RootManage
   SharedAccessKey;SharedAccessKey=AbCdEfGhIjKlMnOpQrStUvWxYz==";
           const string QueueName = "salesmessages";

           static void Main(string[] args)
           {
               Console.WriteLine("Sending a message to the Sales Messages queue...");
               SendSalesMessageAsync().GetAwaiter().GetResult();
               Console.WriteLine("Message was sent successfully.");
           }

           static async Task SendSalesMessageAsync()
           {
   ```

```
            await using var client = new ServiceBusClient(ServiceBusConnectionString);

            await using ServiceBusSender sender = client.CreateSender(QueueName);
            try
            {
                string messageBody = $"$10,000 order for bicycle parts from retailer Adventure Works.";
                var message = new ServiceBusMessage(messageBody);
                Console.WriteLine($"Sending message: {messageBody}");
                await sender.SendMessageAsync(message);
            }
            catch (Exception exception)
            {
                Console.WriteLine($"{DateTime.Now} :: Exception: {exception.Message}");
            }
            finally
            {
                // Calling DisposeAsync on client types is required to ensure that network
                // resources and other unmanaged objects are properly cleaned up.
                await sender.DisposeAsync();
                await client.DisposeAsync();
            }
        }
    }
}
```

3. **...** 아이콘 또는 액셀러레이터 키(Windows 및 Linux에서는 Ctrl+S, macOS에서는 Cmd+S)를 사용하여 *privatemessagesender/Program.cs* 파일을 저장합니다.
4. 편집기의 오른쪽 우측 모서리에서 **...**를 선택한 다음, **편집기 닫기**를 선택합니다.

큐에 메시지 보내기

1. 영업 관련 메시지를 보내는 구성 요소를 실행하려면 Cloud Shell에서 다음 명령을 실행합니다. 첫 번째 줄은 올바른 경로에 있는지 확인합니다.

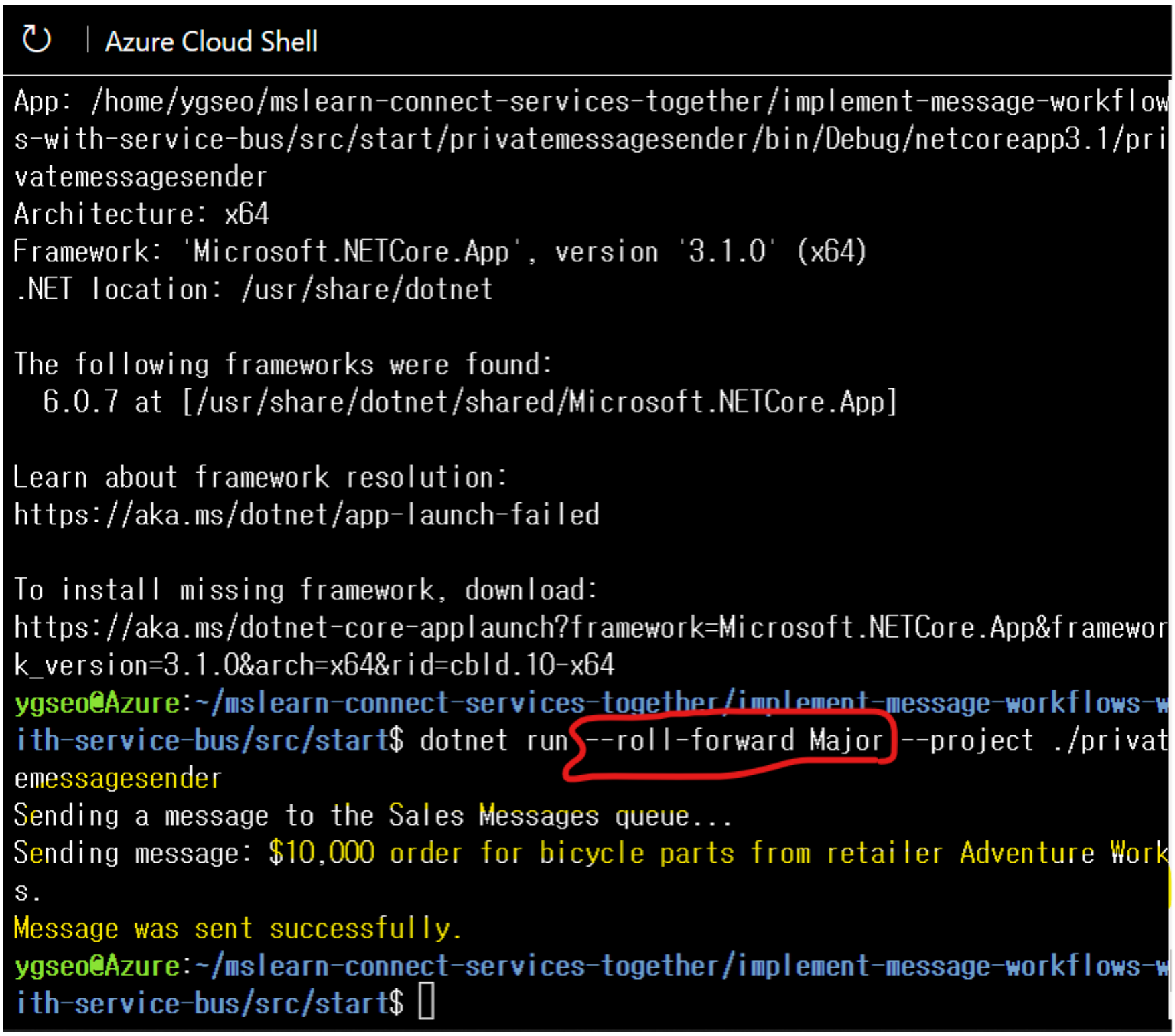
```
cd ~/mslearn-connect-services-together/implement-message-workflows-with-service-bus/src/start
dotnet run -p ./privatemessagesender
```

참고

이 연습에서 앱을 처음 실행할 경우 `dotnet` 이 원격 원본에서 패키지를 복원하고 앱을 빌드하도록 허용합니다.

```
Azure Cloud Shell
ygseo@Azure:/usr/share/dotnet$ dotnet --info
.NET SDK (reflecting any global.json):
 Version:   6.0.302
 Commit:    c857713418

Runtime Environment:
 OS Name:     cbld
 OS Version:  10
 OS Platform: Linux
 RID:         linux-x64
 Base Path:   /usr/share/dotnet/sdk/6.0.302/

global.json file:
  Not found

Host:
  Version:      6.0.7
  Architecture: x64
  Commit:       0ec02c8c96

.NET SDKs installed:
  6.0.302 [/usr/share/dotnet/sdk]

.NET runtimes installed:
  Microsoft.AspNetCore.App 6.0.7 [/usr/share/dotnet/shared/Microsoft.AspNetCo
```

이건 제 랩탑이나 원격의 원본에 있는 .NET 신규 릴리즈 버전(6.0.7 최신)을 업데이트할 게 아니라, 실습랩의 원격의 큐에 .NET 버전 업데이트(3.1.0)가 필요할 거 같은데요...



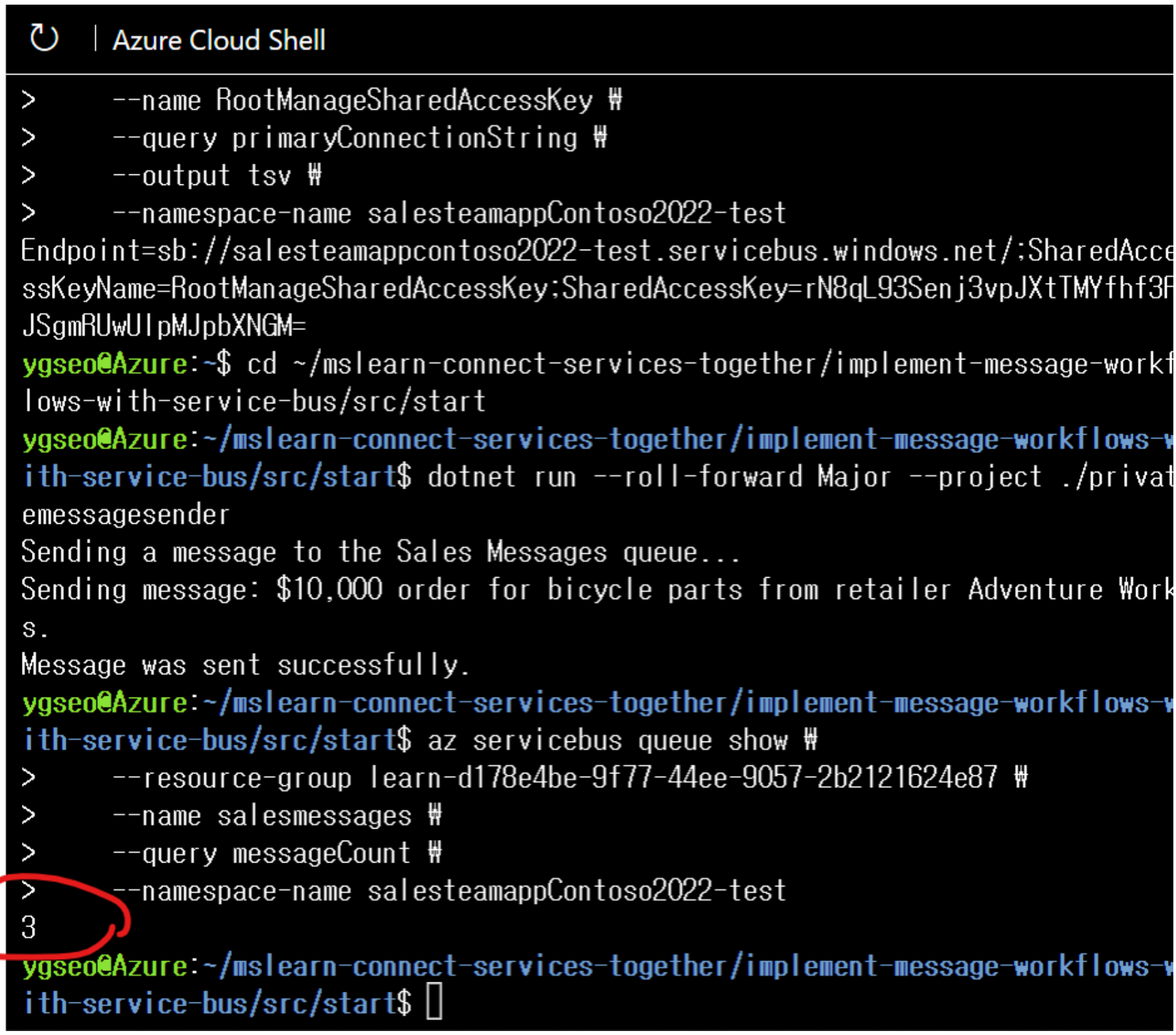
roll-forward라는 옵션을 추가하여 낮은 버전의 .net에 원격의 원본에서 .net 버전을 포워딩시켰습니다. 다행히 큐에 메시지가 보내졌습니다. 참조 : dotnet 명령 - .NET CLI | Microsoft Docs

프로그램이 실행되면 애플리케이션에서 메시지를 보내고 있음을 나타내는 메시지가 콘솔에 출력됩니다.

```
Sending a message to the Sales Messages queue...
Sending message: $10,000 order for bicycle parts from retailer Adventure Works.
Message was sent successfully.
```

2. 앱이 완료되면 다음 명령을 실행하고 <namespace-name>을 Service Bus 네임스페이스의 이름으로 바꿉니다. 이 명령으로 큐에 있는 메시지의 수가 반환됩니다.

```
az servicebus queue show \
    --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 \
    --name salesmessages \
    --query messageCount \
    --namespace-name <namespace-name>
```



잠깐 MSI와 커넥션이 두절되어서 몇 분 지난 후 CLI모드 재진입했습니다. 메시지 큐 보내기를 통신이 끊기기 전에 세 번 보냈었는데, 큐에 적재된 메시지 수가 정확하게 카운트 됐네요.

1. 1단계의 `dotnet run` 명령을 다시 실행한 다음, `servicebus queue show` 명령을 다시 실행합니다. dotnet 앱을 실행할 때마다 새 메시지가 큐에 추가됩니다. Azure 명령을 실행할 때마다 `messageCount`가 증가하는 것을 확인할 수 있습니다.

코드를 작성하여 큐에서 메시지 받기

1. 다음 명령을 실행하여 편집기를 다시 엽니다.

   Bash복사

   ```
   code .
   ```

2. *privatemessagereceiver/Program.cs*의 최종 코드가 다음 예제와 유사한지 확인합니다.

   C#복사

   ```
   using System;
   using System.Text;
   using System.Threading.Tasks;
   using Azure.Messaging.ServiceBus;

   namespace privatemessagereceiver
   {
       class Program
       {

           const string ServiceBusConnectionString = "Endpoint=sb://<examplenamespace.servicebus.windows.net/;SharedAccessKeyName=
   RootManageSharedAccessKey;SharedAccessKey=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
           const string QueueName = "salesmessages";
   ```

```
        static void Main(string[] args)
        {

            ReceiveSalesMessageAsync().GetAwaiter().GetResult();

        }

        static async Task ReceiveSalesMessageAsync()
        {

            Console.WriteLine("========================================================");
            Console.WriteLine("Press ENTER key to exit after receiving all the messages.");
            Console.WriteLine("========================================================");


            var client = new ServiceBusClient(ServiceBusConnectionString);

            var processorOptions = new ServiceBusProcessorOptions
            {
                MaxConcurrentCalls = 1,
                AutoCompleteMessages = false
            };

            await using ServiceBusProcessor processor = client.CreateProcessor(QueueName, processorOptions);

            processor.ProcessMessageAsync += MessageHandler;
            processor.ProcessErrorAsync += ErrorHandler;


            await processor.StartProcessingAsync();

            Console.Read();

            await processor.CloseAsync();

        }

        // handle received messages
        static async Task MessageHandler(ProcessMessageEventArgs args)
        {
            string body = args.Message.Body.ToString();
            Console.WriteLine($"Received: {body}");

            // complete the message. messages is deleted from the queue.
            await args.CompleteMessageAsync(args.Message);
        }

        // handle any errors when receiving messages
        static Task ErrorHandler(ProcessErrorEventArgs args)
        {
            Console.WriteLine(args.Exception.ToString());
            return Task.CompletedTask;
        }
    }
}
```

3. ☰ 메뉴 또는 액셀러레이터 키(Windows 및 Linux에서는 Ctrl+S, macOS에서는 Cmd+S)를 통해 파일을 저장합니다.
4. 편집기의 오른쪽 우측 모서리에서 **...**를 선택한 다음, **편집기 닫기**를 선택합니다.

큐에서 메시지 받기

1. 영업 관련 메시지를 받는 구성 요소를 실행하려면 Cloud Shell에서 다음 명령을 실행합니다.

```
dotnet run --roll-forward Major --project privatemessagereceiver
```

```
Azure Cloud Shell
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ az servicebus queue show ₩
>     --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 ₩
>     --name salesmessages ₩
>     --query messageCount ₩
>     --namespace-name salesteamappContoso2022-test
3
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ code .
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ dotnet run --roll-forward Major -project privateme
ssagereceiver
Couldn't find a project to run. Ensure a project exists in /home/ygseo/mslear
n-connect-services-together/implement-message-workflows-with-service-bus/src/
start, or pass the path to the project using --project.
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ dotnet run --roll-forward Major --project privatem
essagereceiver
=============================================
Press ENTER key to exit after receiving all the messages.
=============================================
Received: $10,000 order for bicycle parts from retailer Adventure Works.
Received: $10,000 order for bicycle parts from retailer Adventure Works.
Received: $10,000 order for bicycle parts from retailer Adventure Works.
```

1. Cloud Shell에서 알림을 확인합니다. Azure Portal에서 Service Bus 네임스페이스로 이동하여 **메시지** 차트를 확인합니다.

```
Received: $10,000 order for bicycle parts from retailer Adventure Works.
```

2. Cloud Shell에서 메시지가 수신된 것으로 확인되면 Enter 키를 눌러 앱을 중지합니다.

메시지 수 확인

다음 코드를 실행하여 이 모든 메시지가 큐에서 제거되었는지 확인하고 <namespace-name>을 Service Bus 네임스페이스로 바꿉니다.

Azure CLI복사

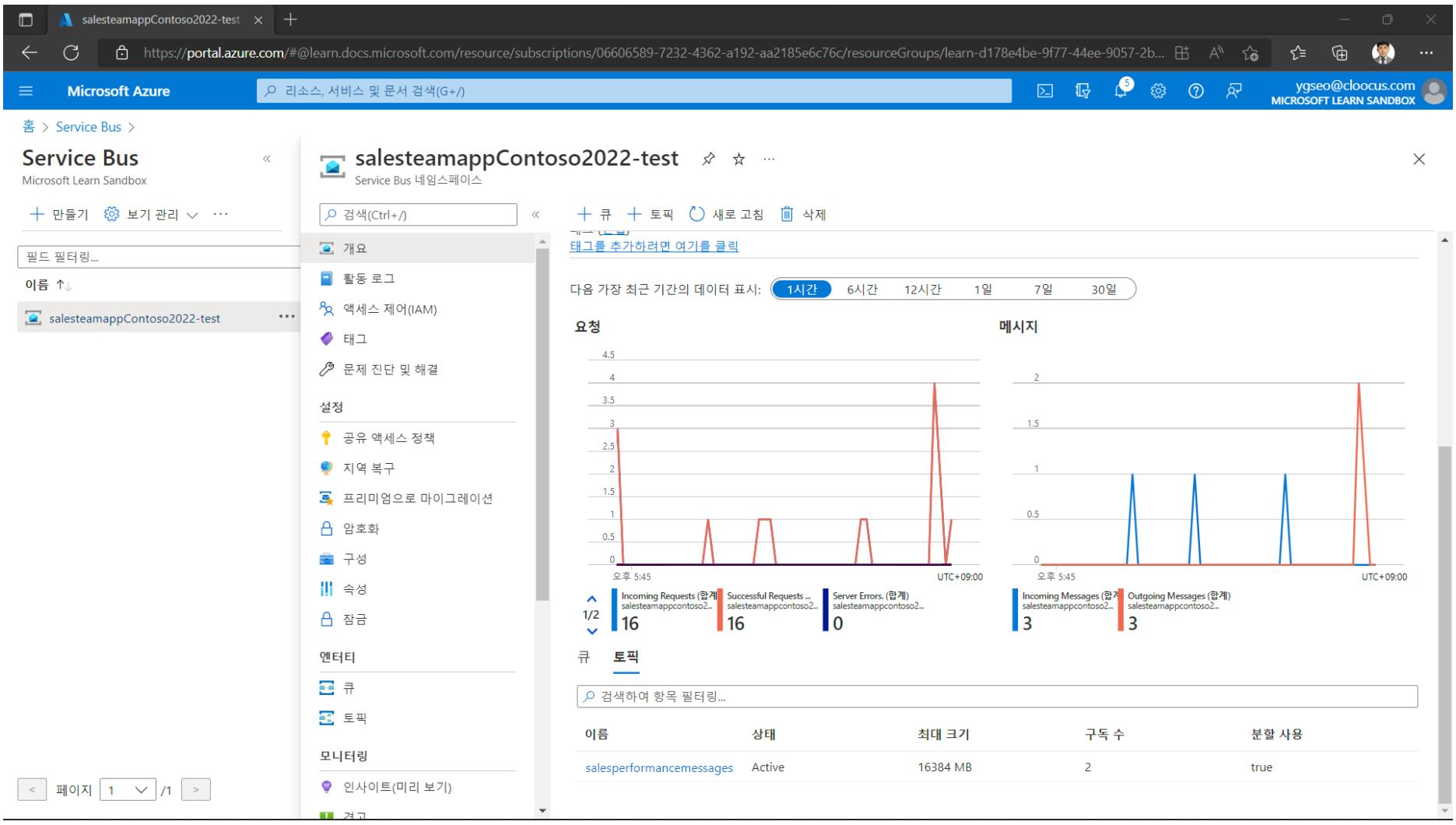
```
az servicebus queue show \
    --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 \
    --name salesmessages \
    --query messageCount \
    --namespace-name <namespace-name>
```

모든 메시지가 제거된 경우에는 `0`이 표시됩니다.

```
Azure Cloud Shell
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ dotnet run --roll-forward Major -project privateme
ssagereceiver
Couldn't find a project to run. Ensure a project exists in /home/ygseo/mslear
n-connect-services-together/implement-message-workflows-with-service-bus/src/
start, or pass the path to the project using --project.
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ dotnet run --roll-forward Major --project privatem
essagereceiver
=============================================
Press ENTER key to exit after receiving all the messages.
=============================================
Received: $10,000 order for bicycle parts from retailer Adventure Works.
Received: $10,000 order for bicycle parts from retailer Adventure Works.
Received: $10,000 order for bicycle parts from retailer Adventure Works.

ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$ az servicebus queue show ₩
>     --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 ₩
>     --name salesmessages ₩
>     --query messageCount ₩
>     --namespace-name salesteamappContoso2022-test
0
ygseo@Azure:~/mslearn-connect-services-together/implement-message-workflows-w
ith-service-bus/src/start$
```

Service Bus 큐에 개별 영업 관련 메시지를 전송하는 코드를 작성했습니다. 영업용 분산 애플리케이션에서는 영업 사원이 디바이스에서 사용하는 모바일 앱에서 이 코드를 작성해야 합니다.

Service Bus 큐에서 메시지를 수신하는 코드도 작성했습니다. 영업용 분산 애플리케이션에서는 Azure에서 실행되며 받은 메시지를 처리하는 웹 서비스에서 이 코드를 작성해야 합니다.



큐(FIFO 자료구조) 저장소를 이용한 메시지 전달과 카운트 수 확인 실습 과정을 Cloud Shell에서 진행한 뒤에 Azure Potal의 Service Bus namespace를 통해서 해당 집계값을 메시지 차트(오른편)로 확인가능

▼ **Write code to send and receive messages by using a topic**

토픽을 사용해 메시지를 보내고 받는 코드 작성

분산 애플리케이션에서는 받는 사람 구성 요소 하나로 보내야 하는 메시지도 있고, 둘 이상의 대상에 보내야 하는 메시지도 있습니다.

사용자가 자전거 주문을 취소하는 경우 발생하는 상황을 생각해봅니다. 주문을 취소하는 것은 최초 주문과 약간 다릅니다. 주문이 완료되면 주문이 결제 처리를 거칠 때까지 전까지 워크플로가 기다렸다가 주문이 로컬 매장으로 전송됩니다. 취소 작업의 경우는 매장과 결제 처리 업체에 동시에 알리게 됩니다. 이 방식은 배송 차량 운전자의 시간 낭비를 최소화합니다.

여러 구성 요소가 같은 메시지를 받을 수 있도록 하기 위해 Azure Service Bus 항목을 사용하겠습니다. 다음으로, 코드를 작성할 때의 프로세스와 고려 사항을 살펴보겠습니다.

토픽을 사용하는 코드 대 큐를 사용하는 코드

전송된 모든 메시지를 구독 중인 모든 구성 요소에 배달하려는 경우에 토픽을 사용합니다. 토픽을 사용하는 코드를 작성하는 것은 큐를 대체하는 방법입니다. 이 경우에도 같은 Azure.Messaging.ServiceBus NuGet 패키지를 사용하고, 연결 문자열을 구성하고, 비동기 프로그래밍 패턴을 사용합니다.

또한 동일한 `ServiceBusClient` 클래스와 `ServiceBusSender` 클래스를 사용하여 메시지를 보내고 `ServiceBusProcessor` 클래스를 사용하여 메시지를 받습니다.

구독에 대해 필터 설정

토픽으로 전송되는 특정 메시지가 특정 구독으로 전송되도록 하려면 토픽에서 하나 이상의 필터를 구독에 담을 수 있습니다. 예를 들어, 자전거 애플리케이션의 경우 매장에서 UWP(유니버설 Windows 플랫폼) 애플리케이션을 실행합니다. 각 매장에서 `OrderCancellation` 토픽을 구독하고 자체 `StoreId`를 필터링할 수 있습니다. 여러 매장 위치로 불필요한 메시지가 전송되지 않아 인터넷 대역폭이 절약됩니다. 단, 결제 처리 구성 요소는 모든 `OrderCancellation` 메시지를 구독합니다.

필터는 다음의 세 가지 유형 중 하나일 수 있습니다.

- **부울 필터**. `TrueFilter`를 사용하면 토픽으로 보내는 모든 메시지가 현재 구독으로 배달됩니다. `FalseFilter`는 현재 구독에 배달된 메시지가 없음을 확인합니다. (이로 인해 구독이 효과적으로 차단되거나 해제됩니다.)
- **SQL 필터**. SQL 필터는 SQL 쿼리의 `WHERE` 절과 같은 구문을 사용하여 조건을 지정합니다. 이 필터를 기준으로 평가했을 때 `True`를 반환하는 메시지만 구독자에게 전송됩니다.

- **상관관계 필터**. 상관관계 필터는 각 메시지 속성과의 일치 여부를 비교하는 조건 집합을 포함합니다. 필터의 속성과 메시지의 속성 값이 같으면 두 속성은 일치하는 것으로 간주됩니다.

`StoreId` 필터의 경우 SQL 필터를 사용하는 것이 가능합니다. SQL 필터는 가장 유연하지만 계산 비용이 가장 많이 들고 필터로 인해 Service Bus 처리 속도가 느려질 수 있습니다. 이 경우에는 상관 관계 필터를 선택합니다.

토픽으로 메시지 보내기

토픽에 메시지를 보내려면 다음 단계를 완료합니다.

모든 전송 또는 수신 구성 요소에서 Service Bus 토픽을 호출하는 모든 코드 파일에 다음의 `using` 문을 추가합니다.

```
using System.Threading;
using System.Threading.Tasks;
using Azure.Messaging.ServiceBus;
```

메시지를 보내려면 먼저 새 `ServiceBusClient` 개체를 만들고 이 개체에 연결 문자열과 토픽 이름을 전달합니다.

```
await using var client = new ServiceBusClient(connectionString);
```

그런 다음, `ServiceBusClient` 개체에서 `CreateSender` 메서드를 호출하고 토픽 이름을 지정하여 `ServiceBusSender` 개체를 만듭니다.

```
ServiceBusSender sender = client.CreateSender(topicName);
```

`ServiceBusSender.SendMessageAsync()` 메서드를 호출하고 `ServiceBusMessage`를 전달하여 토픽에 메시지를 보낼 수 있습니다. 메시지는 큐와 마찬가지로 UTF-8 인코딩 문자열 형식이어야 합니다.

```
string message = "Cancel! I have changed my mind!";
var message = new ServiceBusMessage(message);

// Send the message to the topic.
await sender.SendMessageAsync(message);
```

구독에서 메시지 받기

구독에서 메시지를 받으려면 `ServiceBusProcessor` 개체를 만들고, 토픽 이름 및 구독 이름을 제공해야 합니다.

```
processor = client.CreateProcessor(topicName, subscriptionName, options);
```

그런 다음, 메시지 처리기 및 오류 처리기를 등록합니다.

```
// Specify the handler method for messages.
processor.ProcessMessageAsync += MessageHandler;

// Specify the handler method for errors.
processor.ProcessErrorAsync += ErrorHandler;
```

메시지 처리기 내에서 처리 작업을 수행합니다. 그런 다음, `ProcessMessageEventArgs.CompleteMessageAsync()` 메서드를 호출하여 구독에서 메시지를 제거합니다.

```
// Complete the message. The message is deleted from the subscription.
await args.CompleteMessageAsync(args.Message);
```

▼ **Send and receive messages by using a topic**

연습 - 토픽을 사용해 메시지 보내고 받기

Azure Service Bus 토픽을 사용하여 Salesforce 애플리케이션에서 판매 실적 메시지를 배포하기로 결정했습니다. 영업 직원은 모바일 디바이스에서 앱을 사용하여 각 영역 및 기간의 판매 수치를 요약한 메시지를 보냅니다. 이러한 메시지는 아메리카를 비롯하여 회사가 사

업을 운영하는 부산에 있는 웹 서비스로 배포됩니다.

토픽에 대한 Azure 구독에 필요한 인프라는 이미 구현했습니다. 이제 메시지를 해당 토픽으로 보내는 토픽을 작성하고 구독에서 메시지를 검색하는 코드를 작성하려고 합니다. 그런 다음, 토픽에 메시지를 보내고 특정 구독의 메시지를 검색합니다.

Azure Cloud Shell에서 다음 명령을 실행하여 올바른 디렉터리에서 작업하고 있는지 확인합니다.

```
cd ~/mslearn-connect-services-together/implement-message-workflows-with-service-bus/src/start
code .
```

토픽에 메시지를 보내는 코드 작성

영업 실적 관련 메시지를 보내는 구성 요소를 완성하려면 다음 단계를 완료합니다.

1. Azure Cloud Shell 편집기에서 *performancemessagesender/Program.cs*를 열고 다음 코드 줄을 찾습니다.

   ```
   const string ServiceBusConnectionString = "";
   ```

   이전 연습에서 저장한 연결 문자열을 따옴표 사이에 붙여넣습니다.

2. 최종 코드가 다음 예제와 유사한지 확인합니다.

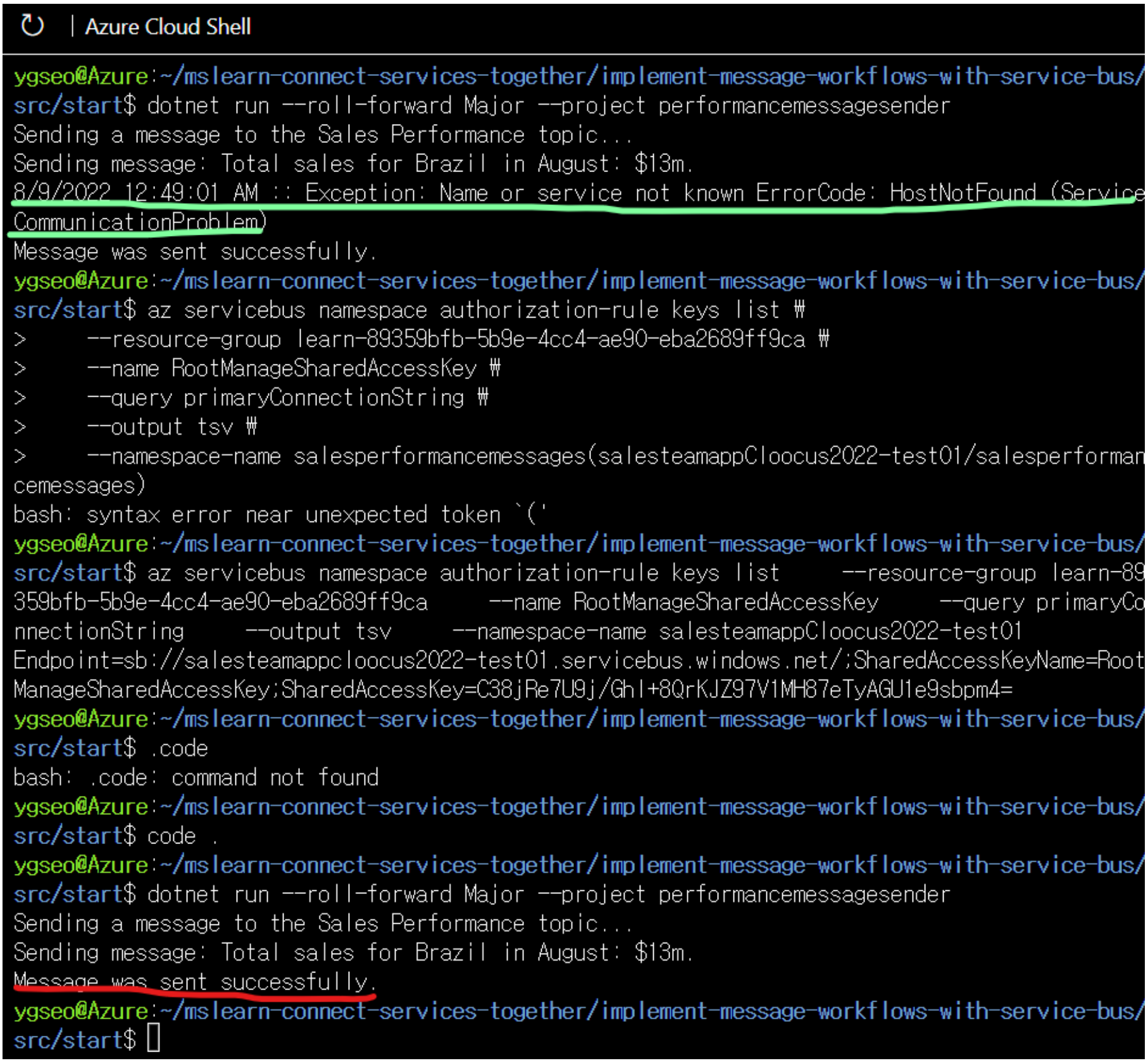
   ```
   using System;
   using System.Threading.Tasks;
   using Azure.Messaging.ServiceBus;

   namespace performancemessagesender
   {
       class Program
       {
           const string ServiceBusConnectionString = "Endpoint=sb://example.servicebus.windows.net/;SharedAccessKeyName=RootManage
   SharedAccessKey;SharedAccessKey=AbCdEfGhIjKlMnOpQrStUvWxYz==";
           const string TopicName = "salesperformancemessages";

           static void Main(string[] args)
           {
               Console.WriteLine("Sending a message to the Sales Performance topic...");
               SendPerformanceMessageAsync().GetAwaiter().GetResult();
               Console.WriteLine("Message was sent successfully.");
           }

           static async Task SendPerformanceMessageAsync()
           {
               // By leveraging "await using", the DisposeAsync method will be called automatically once the client variable goes
    out of scope.
               // In more realistic scenarios, you would store off a class reference to the client (rather than to a local variabl
   e) so that it can be used throughout your program.
               await using var client = new ServiceBusClient(ServiceBusConnectionString);

               await using ServiceBusSender sender = client.CreateSender(TopicName);

               try
               {
                   string messageBody = "Total sales for Brazil in August: $13m.";
                   var message = new ServiceBusMessage(messageBody);
                   Console.WriteLine($"Sending message: {messageBody}");
                   await sender.SendMessageAsync(message);
               }
               catch (Exception exception)
               {
                   Console.WriteLine($"{DateTime.Now} :: Exception: {exception.Message}");
               }
           }
       }
   }
   ```

3. 편집기의 ☰ 메뉴 또는 액셀러레이터 키(Windows 및 Linux에서는 Ctrl+S, macOS에서는 Cmd+S)를 사용하여 파일을 저장합니다.

항목에 메시지 보내기

1. 영업 관련 메시지를 보내는 구성 요소를 실행하려면 Cloud Shell에서 다음 명령을 실행합니다.

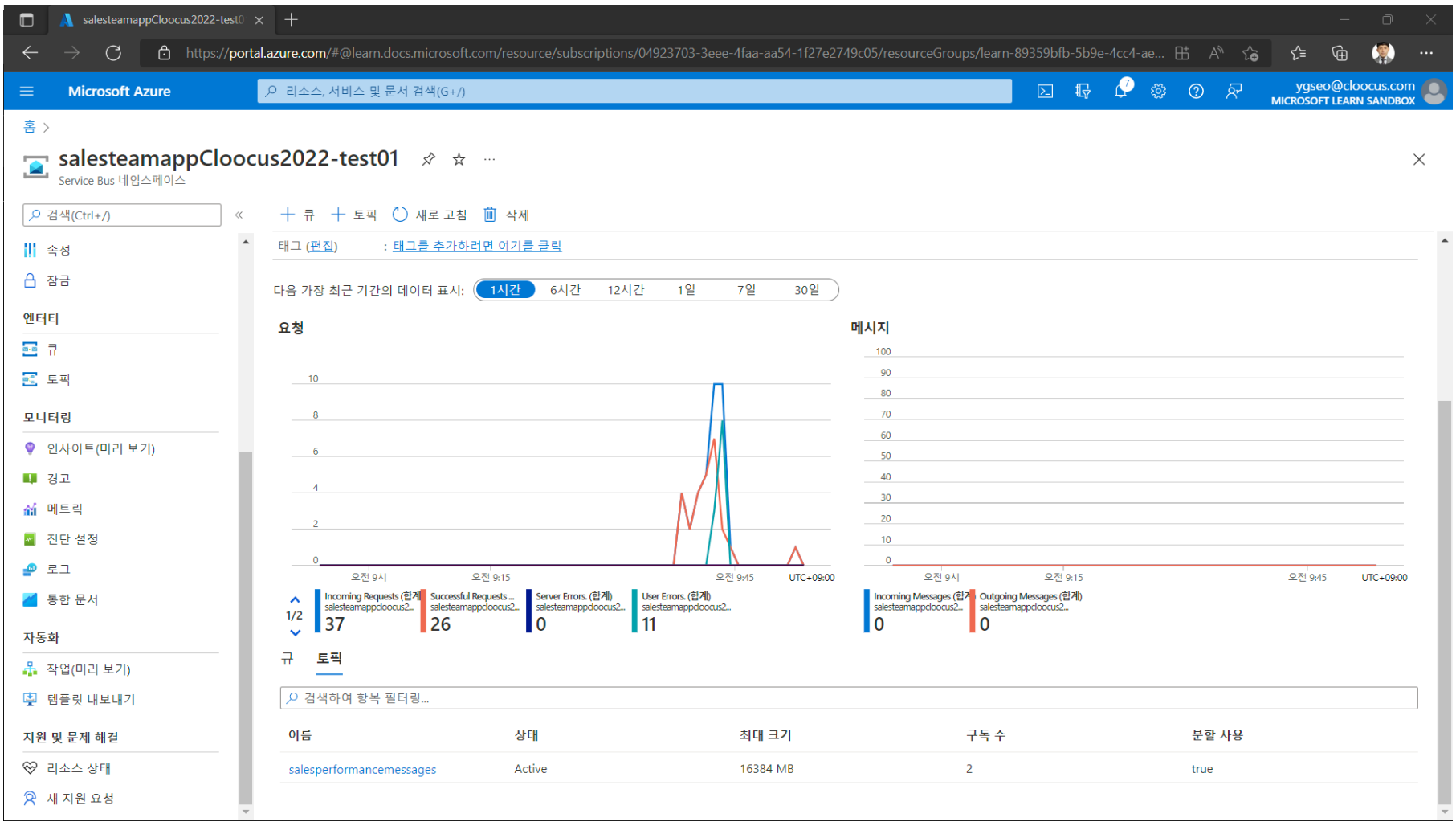
   ```
   dotnet run -p performancemessagesender
   ```



어제 실습 간에 생성한 Endpoint 연결 문자열을 그대로 사용해서 처음엔 호스트를 찾을 수 없다는 예외(전송 코드가 *try-catch* 구문 블럭에 있으므로)가 뜨네요. 그런데 중요한 건 메시지는 성공적으로 보냈다는 결과가 떴다는 거;;; 다시 '연결문자열' 복사해서 *performancemessagesender/Program.cs* 의 해당 코드줄을 치환하고, 닷넷을 재실행하니, 깔끔하게 메시지가 보내졌습니다.

1. 프로그램이 실행되면 Cloud Shell에서 메시지가 전송 중임을 나타내는 알림이 있는지 확인합니다. 앱을 실행할 때마다 토픽에 다른 메시지가 추가되고 각 구독에 복사본을 사용할 수 있게 됩니다.

```
Sending a message to the Sales Performance topic...
Sending message: Total sales for Brazil in August: $13m.
Message was sent successfully.
```



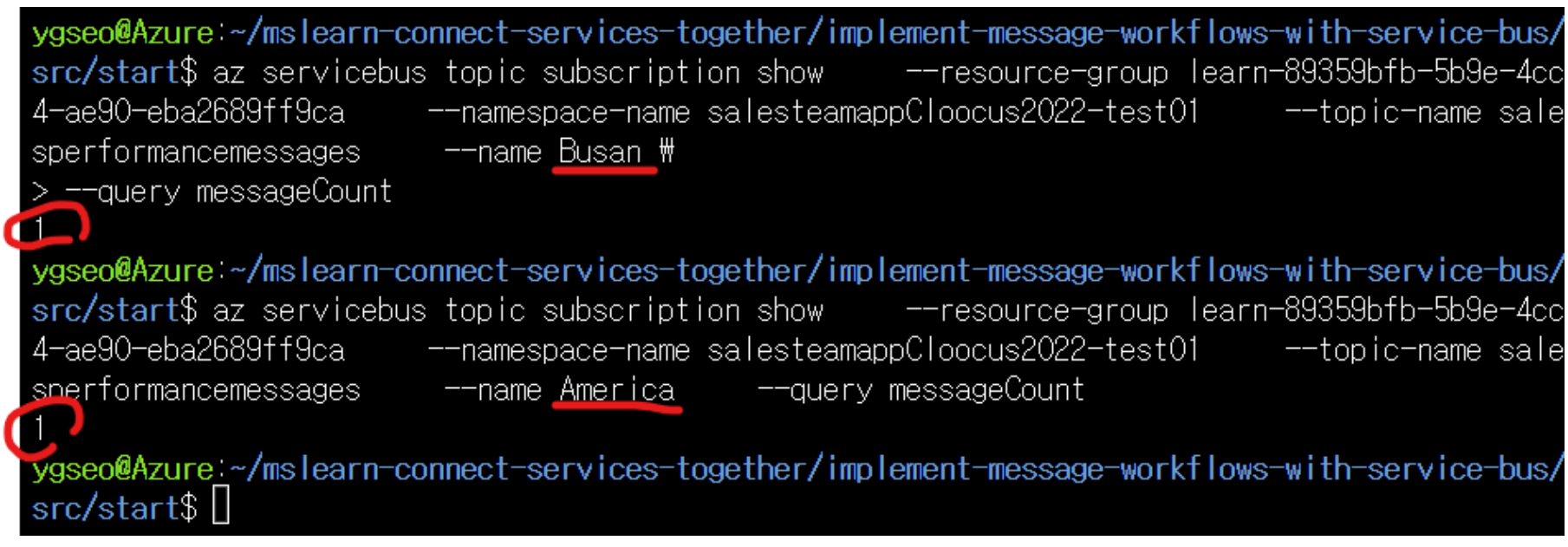
요청 차트에서 User Errors. 가 11개로 치솟더니, Incoming Requests. 합계도 덩달아 치솟았습니다. 아마도 어제와 다르게 오늘 생성한 Service bus의 namespace명(*salesteamappCloocus2022-test01*)으로 인해 호스트를 찾지못한 예외가 발생해서 그런 거 같습니다.

구독에 대한 메시지를 검색하기 전에 메시지 수를 확인합니다.

`Message was sent successfully`가 표시되면 다음 명령을 실행하여 `Busan` 과 `America` 구독에 있는 메시지 수를 확인합니다. <namespace-name>을 Service Bus 네임스페이스의 이름으로 바꾸어야 합니다.

```
az servicebus topic subscription show \
    --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 \
    --namespace-name <namespace-name> \
    --topic-name salesperformancemessages \
    --name Busan \
    --query messageCount
```

`Busan`을 `America`로 바꾸고 명령을 다시 실행하면 두 구독의 메시지가 같아집니다.



구독에 대한 토픽 메시지를 검색하는 코드 작성

영업 실적 관련 메시지를 검색하는 구성 요소를 만들려면 다음 단계를 완료합니다.

1. `code .`를 실행하여 편집기를 시작합니다.
2. 편집기에서 *performancemessagereceiver/Program.cs*를 열고 다음 코드 줄을 찾습니다.

```
const string ServiceBusConnectionString = "";
```

이전 연습에서 저장한 연결 문자열을 따옴표 사이에 붙여넣습니다.

3. 최종 코드가 다음 예제와 유사한지 확인합니다.

```
using System;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Azure.Messaging.ServiceBus;

namespace performancemessagereceiver
{
    class Program
    {
        const string ServiceBusConnectionString = "Endpoint=sb://alexgeddyneil.servicebus.windows.net/;SharedAccessKeyName=Root
ManageSharedAccessKey;SharedAccessKey=LIWIyxs8baqQ0bRf5zJLef6OTfrv0kBEDxFM/ML37Zs=";
        const string TopicName = "salesperformancemessages";
        const string SubscriptionName = "Americas";

        static void Main(string[] args)
        {
            MainAsync().GetAwaiter().GetResult();
        }

        static async Task MainAsync()
        {
            var client = new ServiceBusClient(ServiceBusConnectionString);

            Console.WriteLine("======================================================");
            Console.WriteLine("Press ENTER key to exit after receiving all the messages.");
            Console.WriteLine("======================================================");

            var processorOptions = new ServiceBusProcessorOptions
            {
                MaxConcurrentCalls = 1,
                AutoCompleteMessages = false
            };

            ServiceBusProcessor processor = client.CreateProcessor(TopicName, SubscriptionName, processorOptions);

            processor.ProcessMessageAsync += MessageHandler;
            processor.ProcessErrorAsync += ErrorHandler;

            await processor.StartProcessingAsync();

            Console.Read();

            await processor.DisposeAsync();
            await client.DisposeAsync();
        }

        static async Task MessageHandler(ProcessMessageEventArgs args)
        {
            Console.WriteLine($"Received message: SequenceNumber:{args.Message.SequenceNumber} Body:{args.Message.Body}");
            await args.CompleteMessageAsync(args.Message);
        }

        static Task ErrorHandler(ProcessErrorEventArgs args)
        {
            Console.WriteLine($"Message handler encountered an exception {args.Exception}.");
            Console.WriteLine("Exception context for troubleshooting:");
            Console.WriteLine($"- Endpoint: {args.FullyQualifiedNamespace}");
            Console.WriteLine($"- Entity Path: {args.EntityPath}");
            Console.WriteLine($"- Executing Action: {args.ErrorSource}");
            return Task.CompletedTask;
        }
    }
}
```

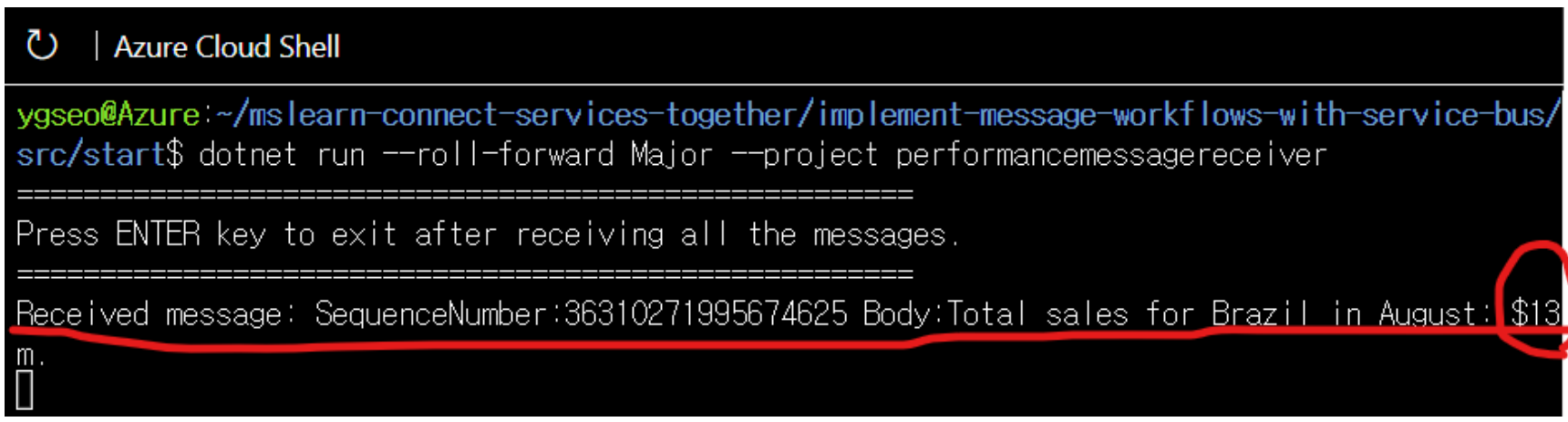
4. ☰ 메뉴를 사용해 파일을 저장하거나 가속기 키(Windows 및 Linux에서는 Ctrl+S, macOS에서는 Cmd+S)를 사용합니다.

구독에 대한 토픽 메시지 검색

1. 구독의 영업 실적 관련 메시지를 검색하는 구성 요소를 실행하려면 다음 명령을 실행합니다.

```
dotnet run -p performancemessagereceiver
```

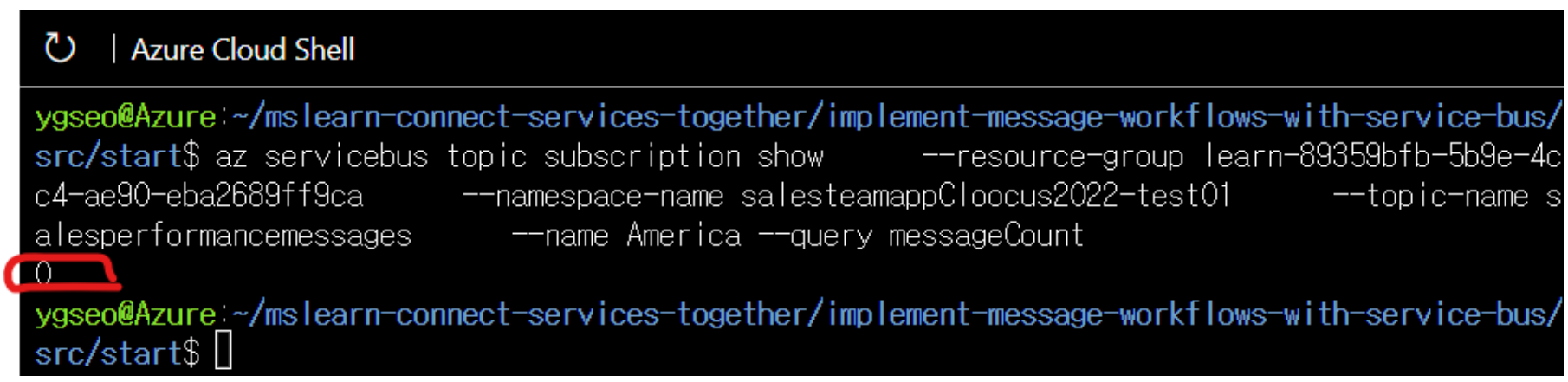
다음 예제와 비슷한 출력이 표시됩니다.



2. 프로그램에서 메시지를 수신하고 있다는 알림이 반환되면 Enter를 눌러 앱을 중지합니다.

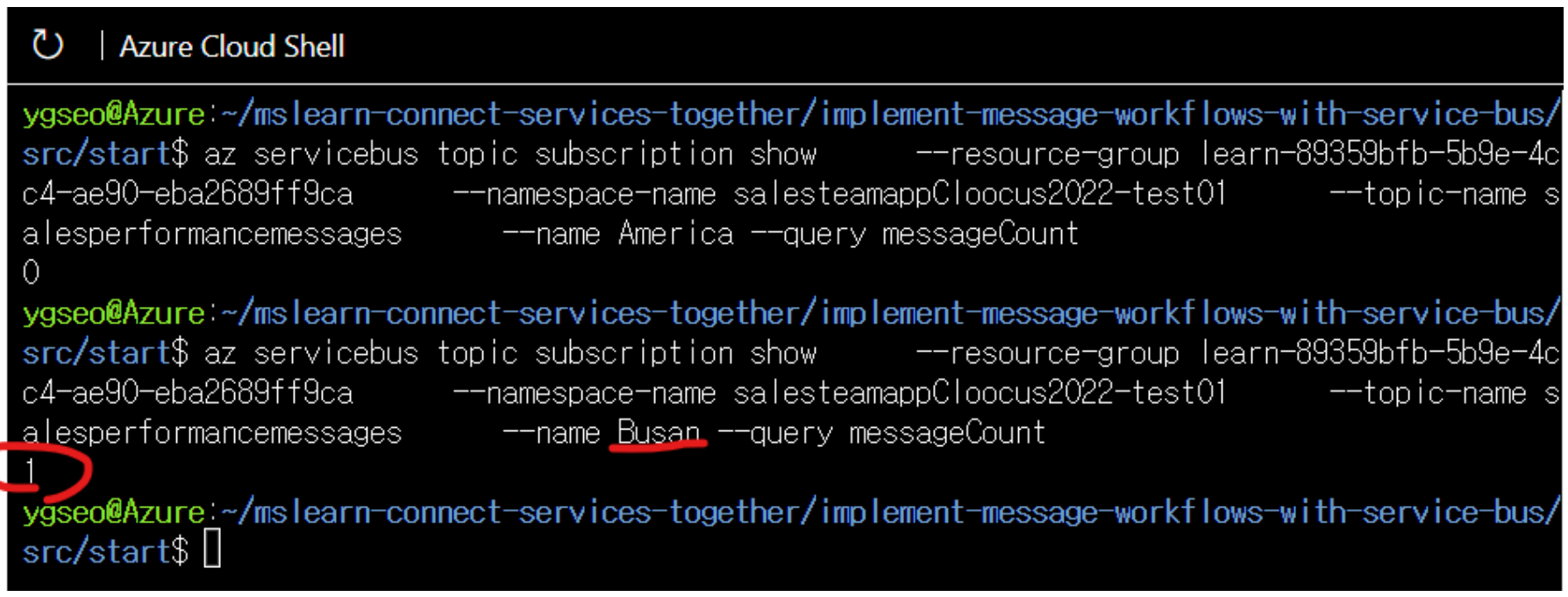
구독에 대한 메시지를 검색한 후 메시지 수를 확인합니다.

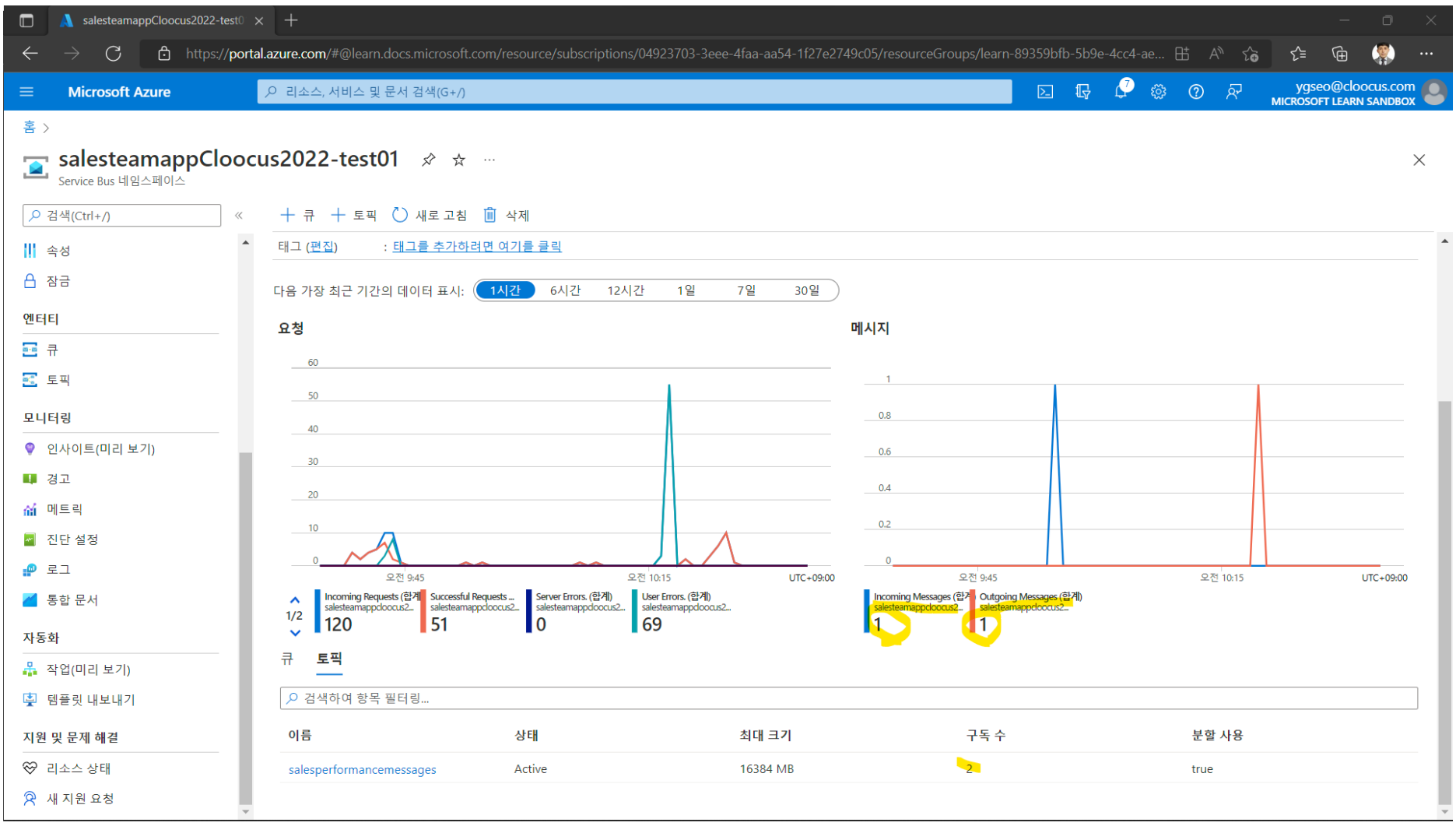
다음 명령을 실행하여 `America` 구독에 남은 메시지가 없는지 확인합니다. <namespace-name>을 Service Bus 네임스페이스의 이름으로 바꾸어야 합니다.




```
az servicebus topic subscription show \
     --resource-group learn-d178e4be-9f77-44ee-9057-2b2121624e87 \
     --namespace-name <namespace-name> \
     --topic-name salesperformancemessages \
     --name Busan \
     --query messageCount
```

`America`를 이 코드의 `Busan`로 바꾸어 `Busan` 구독의 현재 메시지 수를 보면 메시지 수가 `1`입니다. 앞 코드에서는 `America`만 토픽 메시지를 검색하도록 설정되었기 때문에 `Busan` 이 이를 검색할 때까지 메시지가 계속 대기합니다.





performanceMessageReceiver 코드에서는 `America`만 토픽 메시지로 검색하도록 설정되었기 때문에 `Busan`은 이를 검색할 때까지 메시지가 계속 대기하고 있는 상태이고, `America`만 토픽 메시지로 검색되었기 때문에 Queue 저장소에서 해당 메시지가 outgoing한 것을 집계하였음.

▼ **Summary**

이 모듈에서는 Azure 구독에서 토픽에 대한 Service Bus 네임스페이스, 큐, 토픽 및 구독을 만들었습니다. 그런 다음 C# 코드를 사용하여 큐 및 토픽을 통해 메시지를 보내고 받았습니다.

Service Bus 큐 및 토픽은 분산 애플리케이션 내의 통신 복원 기능을 개선하는 데 사용할 수 있는 매우 유용한 도구입니다. **Service Bus 큐 및 토픽은 임시 스토리지 위치 역할을 함으로써 구성 요소 간의 직접 통신에 대한 요구 사항을 없애고 수요 최대치를 원활하게 처리**합니다. 구성이 느슨하게 결합된 다른 구성 요소와 통신할 수 있는 구성 요소가 있을 때는 Service Bus 큐와 토픽을 사용하는 것이 좋습니다.

지식 확인

1. 다음 중 FIFO 메시지 순서(Queue)와 트랜잭션 지원이 필요한 경우에 사용해야 하는 큐는 무엇인가요?

Azure Service Bus 큐

Azure Service Bus 큐는 기본적으로 동일한 순서 메시지에서 메시지를 처리하며 트랜잭션도 지원합니다. 즉, 트랜잭션의 한 메시지가 큐에 추가되지 않으면 트랜잭션의 모든 메시지가 추가되지 않습니다.

2. Azure Service Bus를 사용하여 메시지를 보낼 때 여러 구성 요소에서 이 메시지를 받게 하고 싶다고 가정합니다. 어떤 Azure Service Bus 기능을 사용해야 하나요?

Azure Service Bus 토픽

토픽은 여러 대상 구성 요소가 구독할 수 있습니다. 따라서 각 메시지를 여러 수신자에게 전달할 수 있습니다.

3. True 또는 False: 20MB 크기의 메시지를 Azure Service Bus 큐에 추가할 수 있습니다.

True

프리미엄 계층을 사용하는 경우 최대 100MB 크기의 메시지를 보낼 수 있습니다. 표준 계층의 경우 제한은 256KB입니다.

튜토리얼 완료 후기

1. 엔드포인트 연결 문자열을 소스코드에서 하드코딩으로 사용하는 것은 보안 상 시큐어 코딩 권장사항이 아님. 해당 문자열을 시큐어코딩을 적용하면 패스워드는 외부의 파일에 보관하여 필요할 때마다 복호화 하여 쓸 수 있도록 코딩하는 것을 권장함.
   * 참조 : 하드코드된 암호화 키 - KISA 소프트웨어 개발 보안 가이드 (tistory.com)
2. 메시징 프로토콜 MQTT는 페이스북의 채팅 메시지를 주고 받을 때도 쓰이는 통신규약으로, 보안 상 안전하고 최소한의 패킷과 전력량을 소모하는 효율적인 통신 방법임.

아래 사이트에서 도커에서 해당 프로토콜 중 가장 많이 쓰이는 Mosquitto를 적용하는 실습을 추가적으로 스터디로 진행할 예정임. MQTT란?. MQTT는 M2M, IOT를 위한 프로토콜로서, 최소한의 전력과… | by 박재성 | Medium