## Учреждение образования

## Белорусский государственный технологический университет

Кафедра информационных систем и технологий

**ЛАБОРАТОРНАЯ РАБОТА № 10**

по дисциплине «Корпоративные информационные системы»

Тема

«Azure разработка решений на основе сообщений»

Выполнил студент

4 курса 5 группы, Демьянов В.Р.

Минск 2023

**Цель работы:** изучение возможностей Azure для разработки решений на основе сообщений.

**Ход работы**

Azure поддерживает два типа механизмов очередей: Service Bus и Storage.

Очереди Service Bus входят в более обширную инфраструктуру обмена сообщениями Azure, поддерживающую очереди, публикации и подписки, а также более сложные шаблоны интеграции. Они разработаны на основе более масштабной инфраструктуры обмена сообщениями, предназначенной для интеграции приложений или их компонентов и охватывающей различные протоколы связи, контракты данных, доверенные домены или сети.

Очереди Storage являются частью инфраструктуры службы хранилища Azure. Они позволяют хранить большое количество сообщений. Доступ к сообщениям возможен из любой точки мира с помощью вызовов с проверкой подлинности по протоколу HTTP или HTTPS. Максимальный размер сообщения в очереди составляет 64 КБ. Очередь может содержать миллионы сообщений вплоть до лимита всей емкости учетной записи хранения. Очереди обычно используются для создания списка невыполненных заданий для асинхронной обработки.

В данной лабораторной работе будут использоваться очереди Service Bus.

**1 Создание Azure Service Bus namespace и очереди Service Bus**

Создание Azure Service Bus namespace продемонстрировано на рисунке 1.

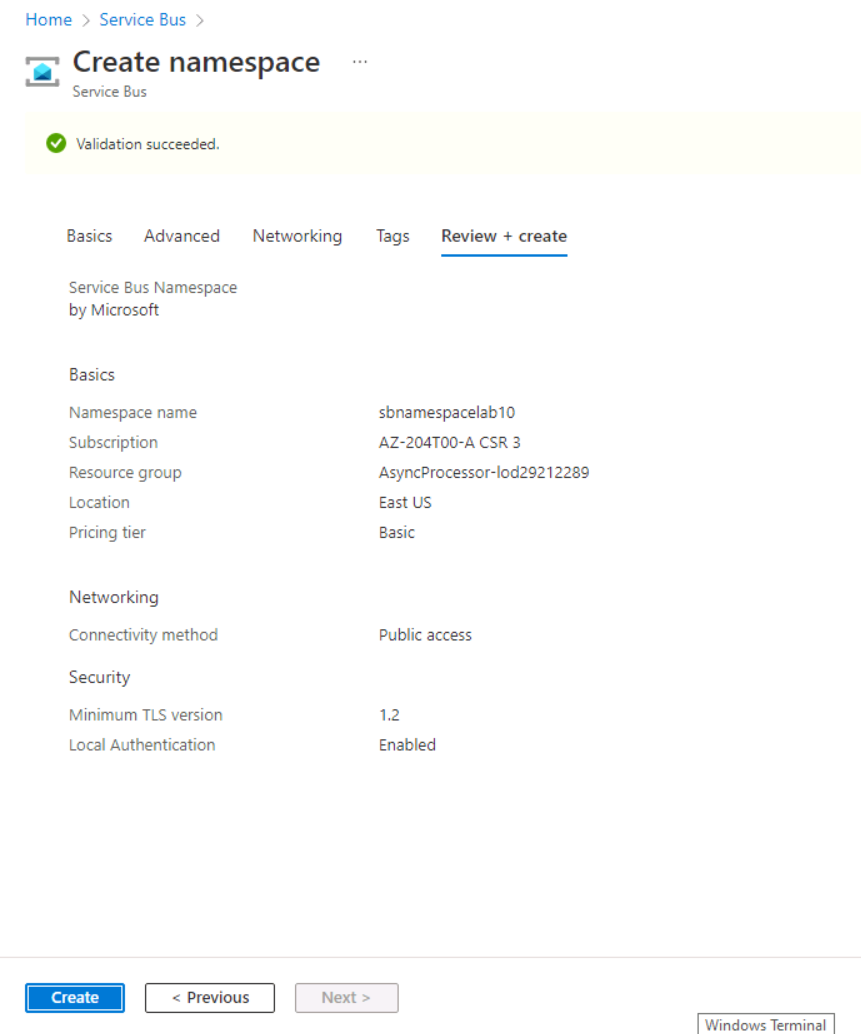


Рисунок 1 – Создание Service Bus namespace

Далее нужно скопировать строку подключения (рисунок 2), которая будет использоваться для подключения к очереди из .Net приложения.

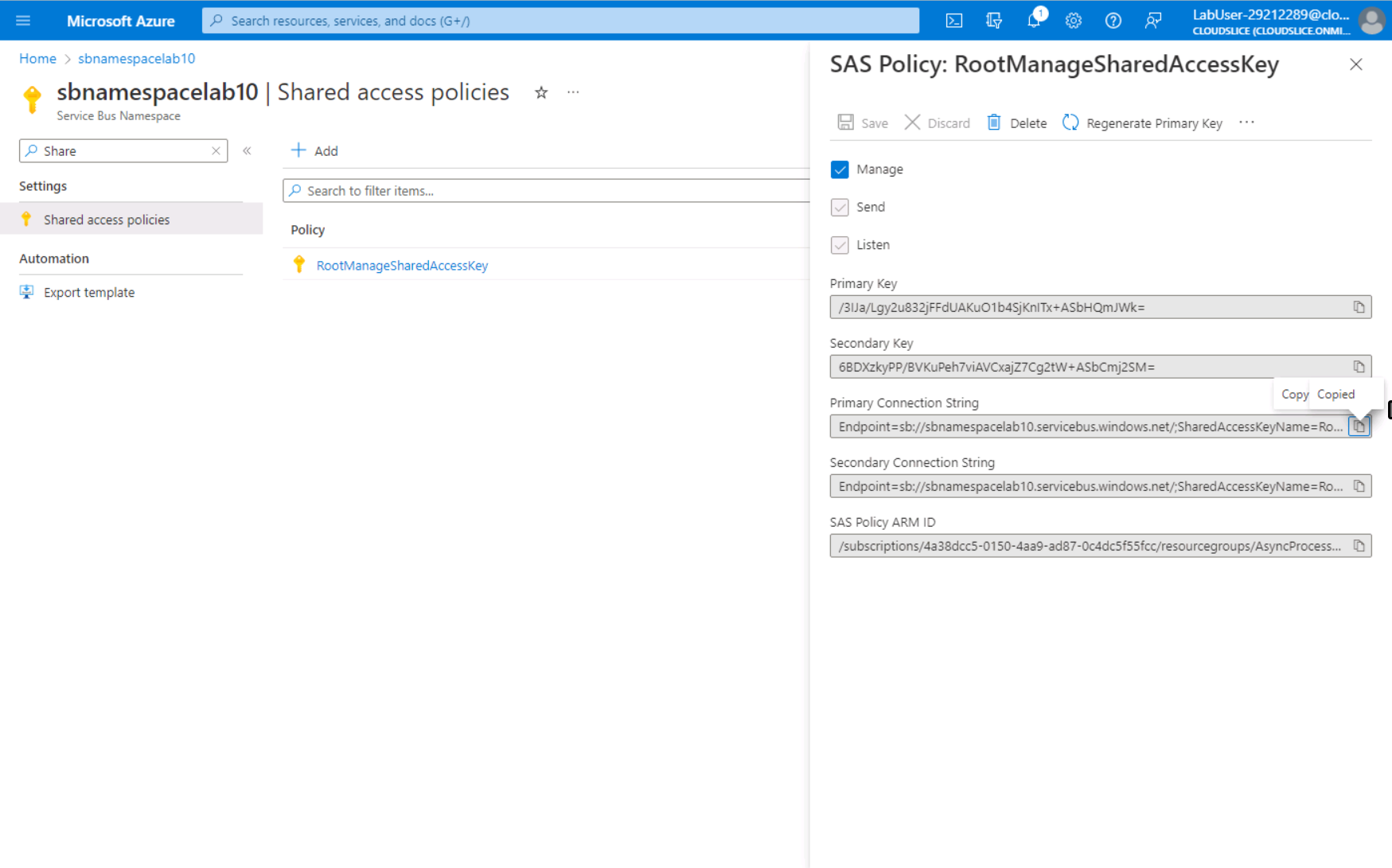


Рисунок 2 – Строка подключение

Далее создаём очередь в созданном ранее namespace.

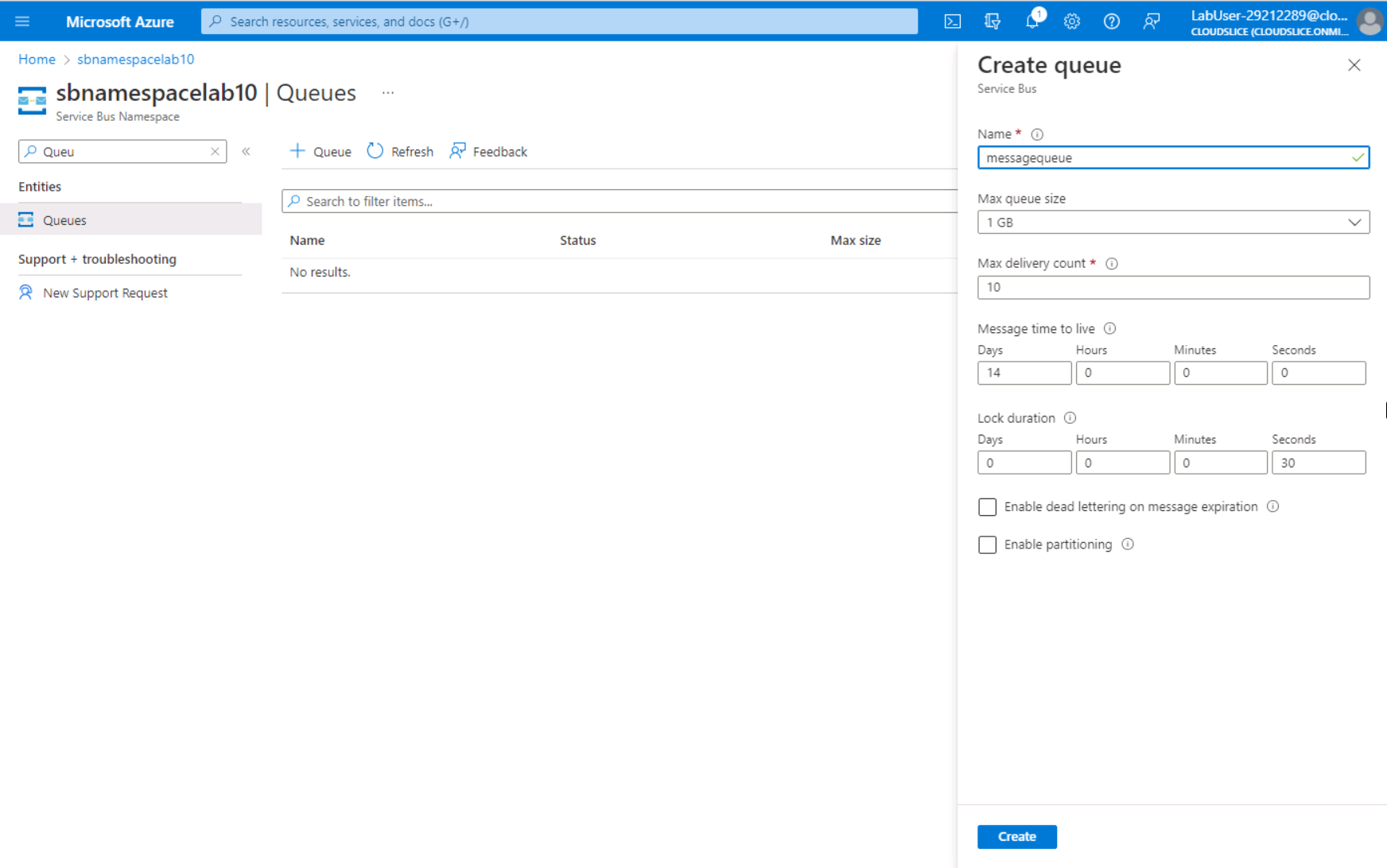


Рисунок 3 – Создание очереди

В результате были созданы Azure Service Bus namespace и очередь Service Bus.

**2 Создание .NET Core приложения для публикации сообщений в очередь**

На рисунке 4 представлен процесс создания консольного .NET Core приложения.

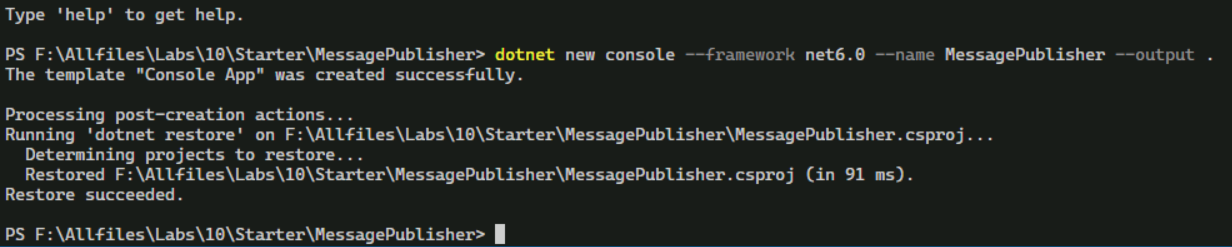


Рисунок 4 – Процесс создания консольного .NET Core приложения

Код класса, в котором происходит публикация сообщений в очередь Azure Service Bus представлен в листинге ниже.

|  |
| --- |
| using System;  using System.Threading.Tasks;  using Azure.Messaging.ServiceBus;  namespace MessagePublisher  {     public class Program     {          private const string serviceBusConnectionString = "Endpoint=sb://sbnamespacelab10.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=/3IJa/Lgy2u832jFFdUAKuO1b4SjKnITx+ASbHQmJWk=";          private const string queueName = "messagequeue";          private const int numOfMessages = 3;          static ServiceBusClient client = default!;          static ServiceBusSender sender = default!;          public static async Task Main(string[] args)          {              client = new ServiceBusClient(serviceBusConnectionString);              sender = client.CreateSender(queueName);              using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();              for (int i = 1; i <= numOfMessages; i++)              {                  if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message {i}")))                  {                      throw new Exception($"The message {i} is too large to fit in the batch.");                  }              }              try              {                  await sender.SendMessagesAsync(messageBatch);                  Console.WriteLine($"A batch of {numOfMessages} messages has been published to the queue.");              }              finally              {                  await sender.DisposeAsync();                  await client.DisposeAsync();              }          }     }  } |

Запуск приложения, которое публикует 3 сообщения в очередь представлен на рисунке 5.

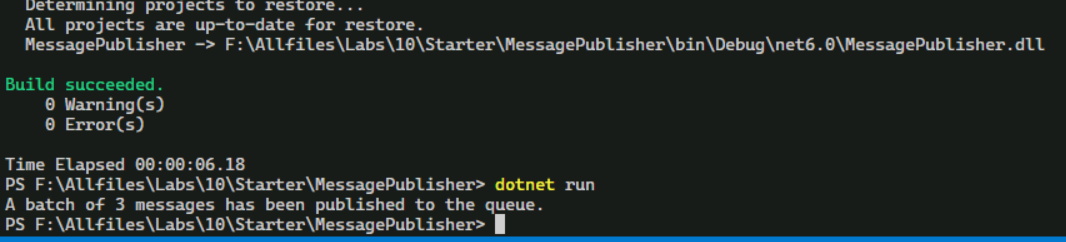


Рисунок 5 – Публикация 3 сообщений

На рисунке 6 продемонстрирована панель с метриками и количество активных сообщений, где мы можем увидеть 3 опубликованных ранее сообщения.

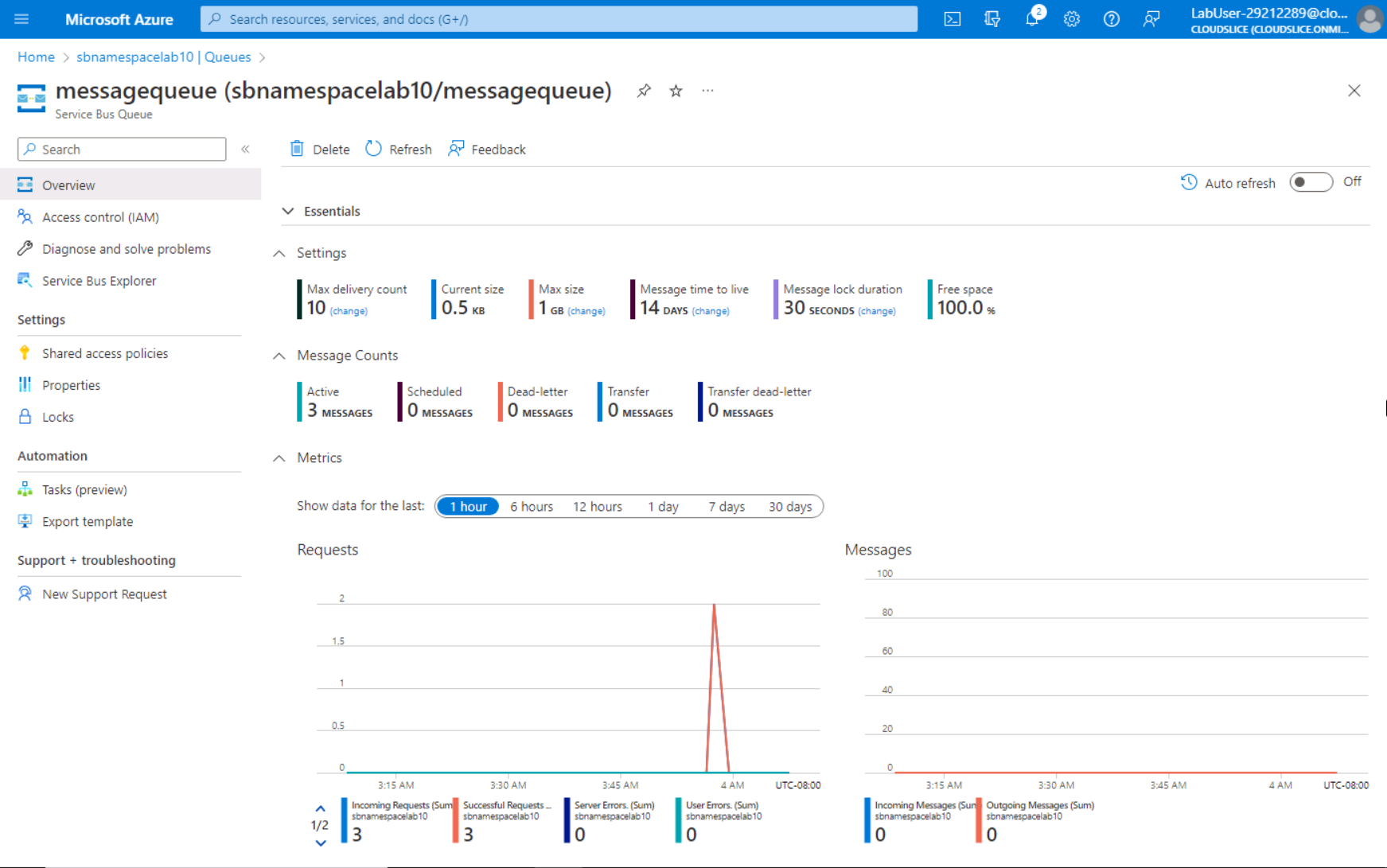


Рисунок 6 – Панель очереди с метриками и количеством активных сообщений

На рисунке 7 продемонстрировано тело одного из сообщений.

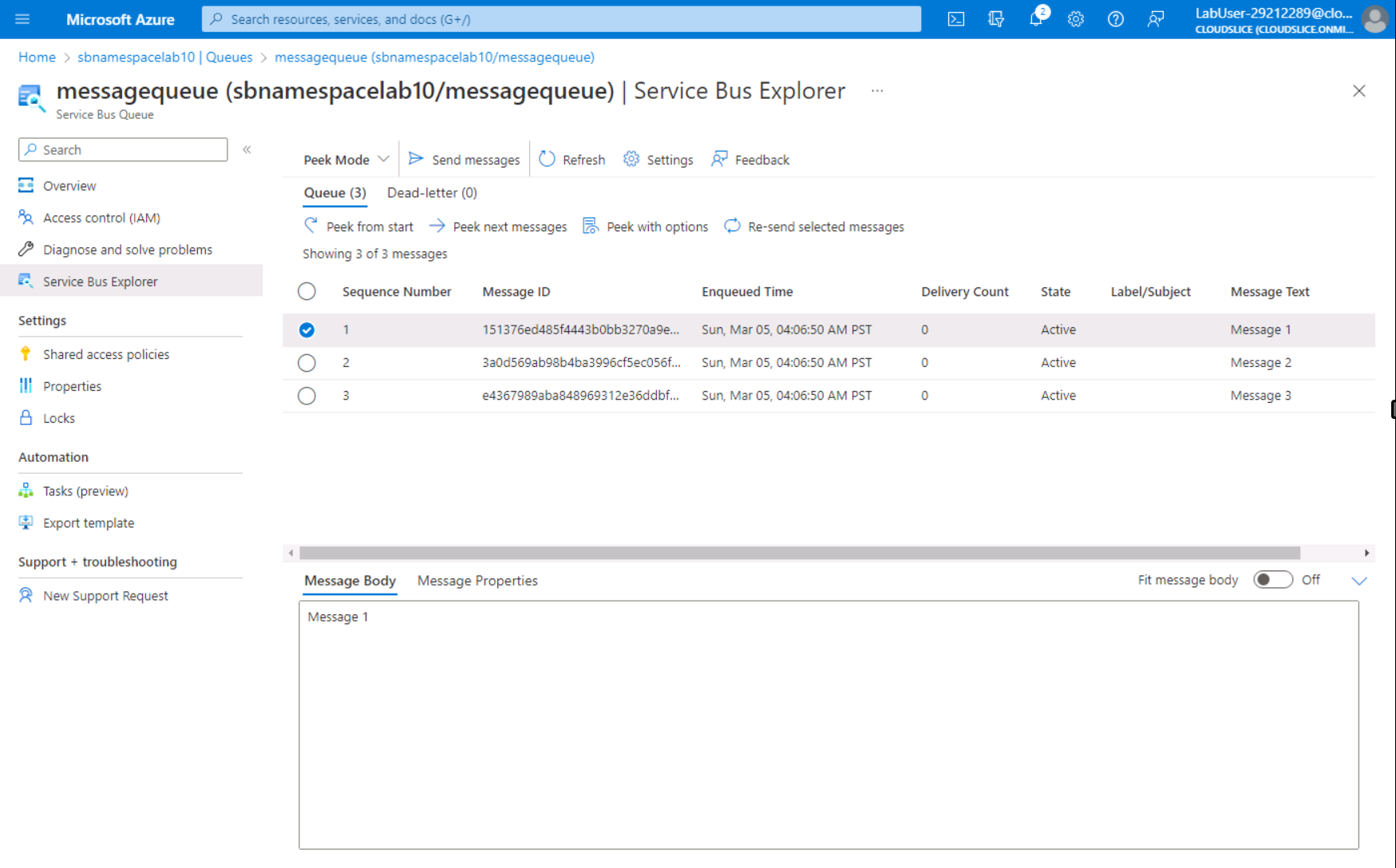


Рисунок 7 – Тело сообщения

В результате было создано приложение, которое публикует сообщения в очередь Azure Service Bus.

**3 Создание приложения читающего сообщения из очереди Azure Service Bus**

Код класса приложение, которое читает сообщения из очереди представлен в листинге ниже.

|  |
| --- |
| using System;  using System.Threading.Tasks;  using Azure.Messaging.ServiceBus;  namespace MessageReader  {     public class Program     {          private const string serviceBusConnectionString = "Endpoint=sb://sbnamespacelab10.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=/3IJa/Lgy2u832jFFdUAKuO1b4SjKnITx+ASbHQmJWk=";          static string queueName = "messagequeue";          static ServiceBusClient client = default!;          static ServiceBusProcessor processor = default!;          static async Task MessageHandler(ProcessMessageEventArgs args)          {              string body = args.Message.Body.ToString();              Console.WriteLine($"Received: {body}");              await args.CompleteMessageAsync(args.Message);          }          static Task ErrorHandler(ProcessErrorEventArgs args)          {              Console.WriteLine(args.Exception.ToString());              return Task.CompletedTask;          }          static async Task Main(string[] args)          {              client = new ServiceBusClient(serviceBusConnectionString);              processor = client.CreateProcessor(queueName, new ServiceBusProcessorOptions());              try              {                  processor.ProcessMessageAsync += MessageHandler;                  processor.ProcessErrorAsync += ErrorHandler;                  await processor.StartProcessingAsync();                  Console.WriteLine("Wait for a minute and then press any key to end the processing");                  Console.ReadKey();                  Console.WriteLine("\nStopping the receiver...");                  await processor.StopProcessingAsync();                  Console.WriteLine("Stopped receiving messages");              }              finally              {                  await processor.DisposeAsync();                  await client.DisposeAsync();              }          }     }  } |

Результат выполенения созданного приложения показан на рисунке 8.

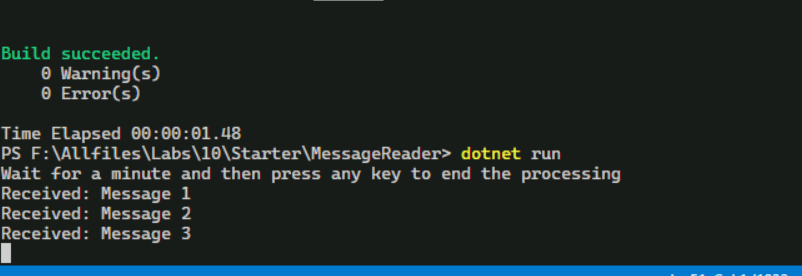


Рисунок 8 – Чтение сообщений из очереди

**Вывод:** в результате выполнения лабораторной работы, я ознакомился с Azure Service Bus queue и создал приложение для публикации и чтения сообщений.