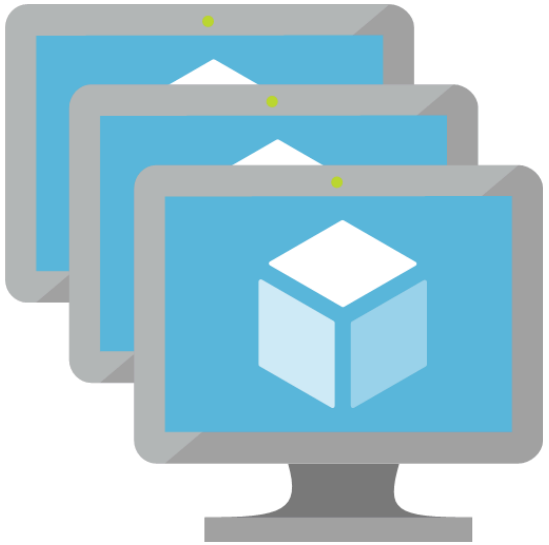
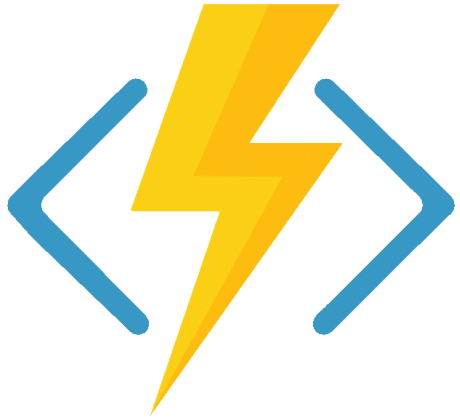
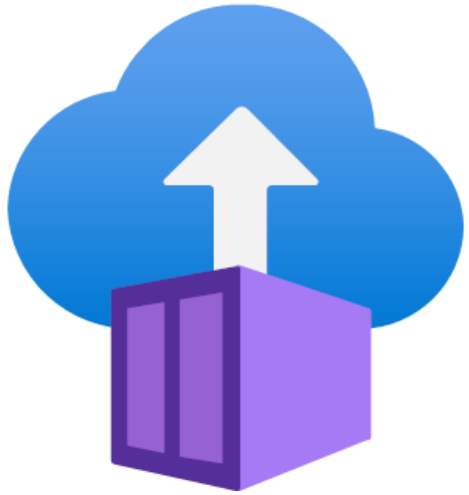
Azure Service Bus
Adding Enterprise Messaging to your Toolkit

Sean Feldman

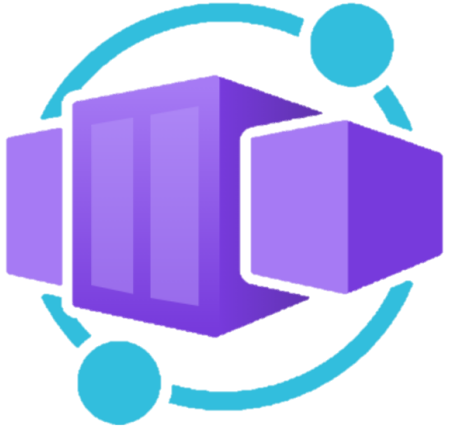
Azure MVP

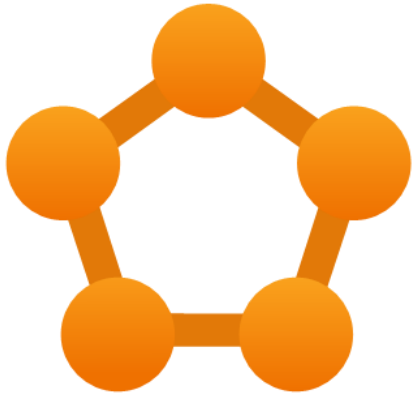
- Consultant / Solutions Architect
- ServiceBus Explorer Contributor
- Contributed to NServiceBus in Azure (transports, hosting, persistence and functions support)
- Azure MVP




.NET Aspire



Azure Messaging Services

Event Grid

Cross-cloud reactive events

Event Hubs

Big data streaming

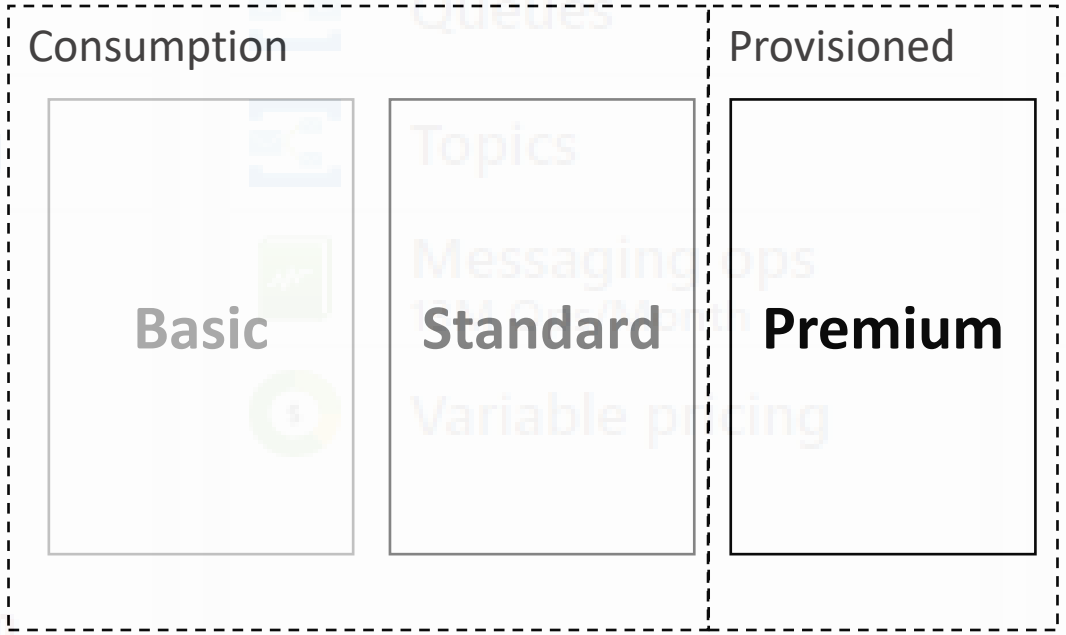
Storage Queues

Simple task queues

Service Bus

Enterprise messaging

Tiers



Broker and Client

.NET	NodeJS
Java	PHP
Python	Ruby

nuget

nuget Packages Upload Statistics Documentation Downloads Blog Sign in

Azure Service Bus

536 packages returned for Azure Service Bus

Sort by Relevance

Azure.Messaging.ServiceBus by: azure-sdk Microsoft

.NET 5.0 .NET Core 2.0 .NET Standard 2.0 .NET Framework 4.6.1

105,532,320 total downloads last updated 11 days ago Latest version: 7.17.4

Azure Service Bus ServiceBus .NET AMQP windowsazureofficial azureofficial

Azure Service Bus is a fully managed enterprise integration message broker. Service Bus can decouple applications and services. Service Bus offers a reliable and secure platform for asynchronous transfer of... More information

0

WindowsAzure.ServiceBus by: azure-sdk Microsoft nugetservicebus

Deprecated

.NET Framework 4.6.2

55,504,778 total downloads last updated 4 months ago Latest version: 7.0.1

ServiceBus Microsoft Azure AppFabric Messaging PubSub Publish Subscribe Queue Topic More tags

Please note, for Azure Service Bus, Azure Event Hubs and Azure Relay, newer packages Azure.Messaging.ServiceBus, Azure.Messaging.EventHubs and Microsoft.Azure.Relay are available as of November 2020, February... More information

1

Microsoft.Azure.ServiceBus by: azure-sdk Microsoft nugetservicebus

Deprecated

.NET 5.0 .NET Core 2.0 .NET Standard 2.0 .NET Framework 4.6.1

112,457,918 total downloads last updated 2021-11-08 Latest version: 5.2.0

Microsoft Azure Service Bus ServiceBus .NET AMQP IoT Queue Topic

Please note, a newer package Azure.Messaging.ServiceBus is available as of November 2020. While this package will continue to receive critical bug fixes, we strongly encourage you to upgrade. Read the migration... More information

Microsoft ♥ OpenSource

github.com/azure/azure-sdk-for-net

AMQP

Advanced Message Queuing Protocol

1.0

Entity Management

Management library

Microsoft.Azure.Management.ServiceBus

```
var subscriptionID = "<SUBSCRIPTION ID>";
var resourceGroupName = "<RESOURCE GROUP NAME>";
var namespaceName = "<SERVICE BUS NAMESPACE NAME>";
var queueName = "<NAME OF QUEUE YOU WANT TO CREATE>";

var token = await GetToken();

var creds = new TokenCredentials(token);
var sbClient = new ServiceBusManagementClient(creds)
{
    SubscriptionId = subscriptionID,
};

var queueParams = new"Creating queue..." SBQueue();

await sbClient.Queues.CreateOrUpdateAsync(resourceGroupName, namespaceName, queueName, queueParams);
```

```
private static async Task<string> GetToken()
{
    try
    {
        var tenantId = "<AZURE AD TENANT ID>";
        var clientId = "<APPLICATION/CLIENT ID>";
        var clientSecret = "<CLIENT SECRET>";

        var context = new AuthenticationContext($"https://login.microsoftonline.com/{tenantId}");

        var result = await context.AcquireTokenAsync(
            "https://management.azure.com/",
            new ClientCredential(clientId, clientSecret)
        );

        // If the token isn't a valid string, throw an error.
        if (string.IsNullOrEmpty(result.AccessToken))
        {
            throw new Exception("Token result is empty!");
        }

        return;
    }
}
```

Client library

Azure.Messaging.ServiceBus

```
var client = new ServiceBusAdministrationClient(connectionString)

if (await client.QueueExistsAsync(destination))
{
    await client.DeleteQueueAsync(destination);
}

await client.CreateQueueAsync(destination);
```

Connection Options

(**string connectionString**)

Initializes a new instance of the ServiceBusClient class.
connectionString: The connection string to use for connecting to the Service Bus namespace.

(**string connectionString**, ServiceBusClientOptions options)

(**string fullyQualifiedNamespace**, AzureNamedKeyCredential credential, ServiceBusClientOptions options = null)

(**string fullyQualifiedNamespace**, AzureSasCredential credential, ServiceBusClientOptions options = null)

(**string fullyQualifiedNamespace**, TokenCredential credential)

- - - - - - -

Sending Messages

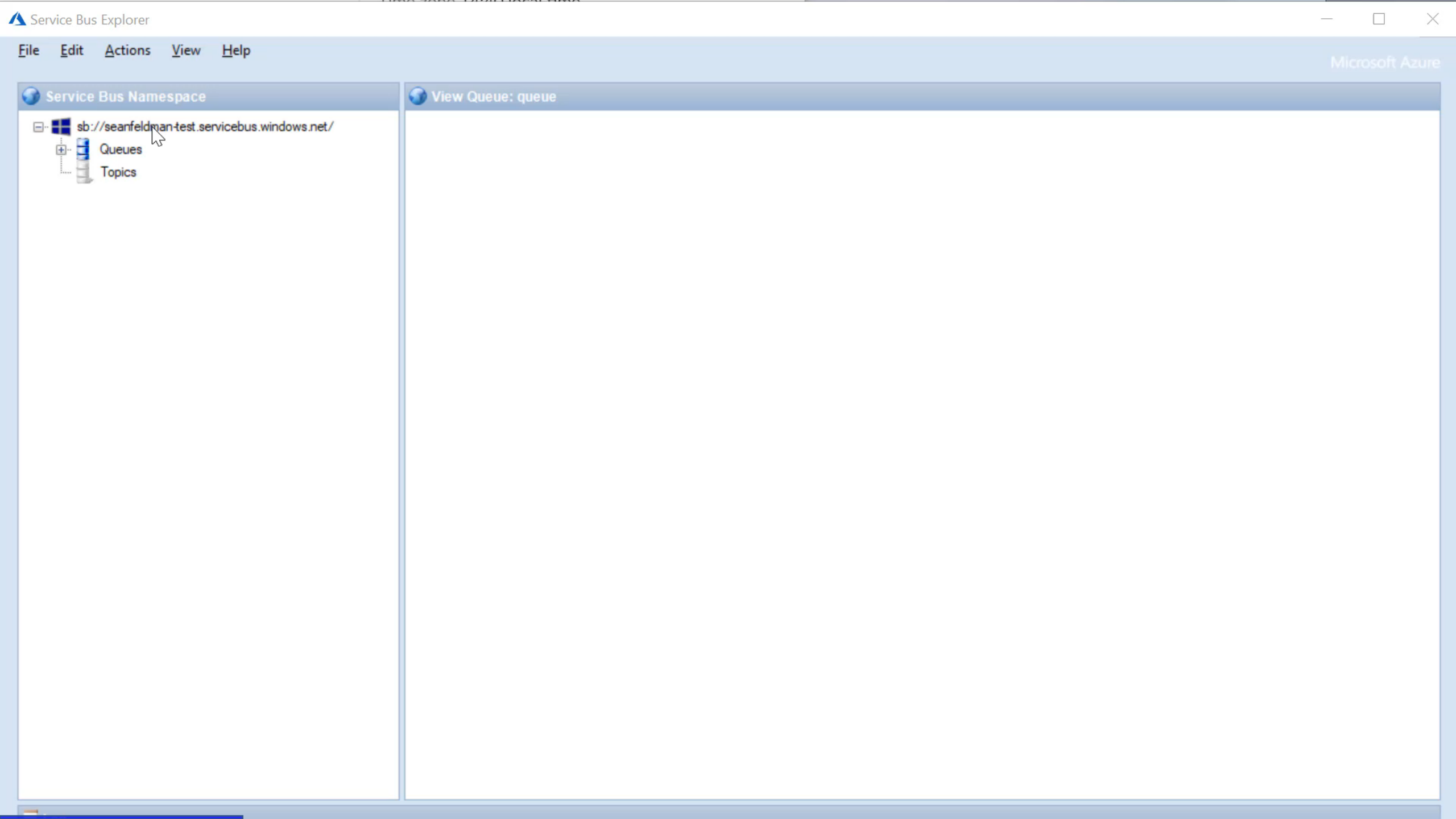
Code

```
<PackageReference Include="Azure.Messaging.ServiceBus" Version="7.17.4" />
```

```
var serviceBusClient = new ServiceBusClient(connectionString);

var client = serviceBusClient.CreateSender(destination);

var message = new ServiceBusMessage("Payload")
{
    Subject = "Deep Dive" // Label
};
message.ApplicationProperties.Add("Machine", Environment.MachineName);

await client.SendMessageAsync(message);

await client.CloseAsync();
```

cmd

```
d:\emos\Sending>dotnet run --no-build --no-restore Sending.csproj
```


Service Bus Explorer
File
Edit
Actions
View
Help
Microsoft Azure
Service Bus Namespace
sb://seanfeldman-test.servicebus.windows.net/
Queues
Topics
View Queue: queue

Receiving Messages

Receiving modes

1. ReceiveAndDelete
2. PeekLock

Simple Scenarios

ServiceBusProcessor

```
var processor = serviceBusClient.CreateProcessor(destination, options);

processor.ProcessMessageAsync += (ProcessMessageEventArgs x) => Task.CompletedTask;

processor.ProcessErrorAsync += (ProcessErrorEventArgs x) => Task.CompletedTask;
```

Advanced Scenarios

ServiceBusReceiver

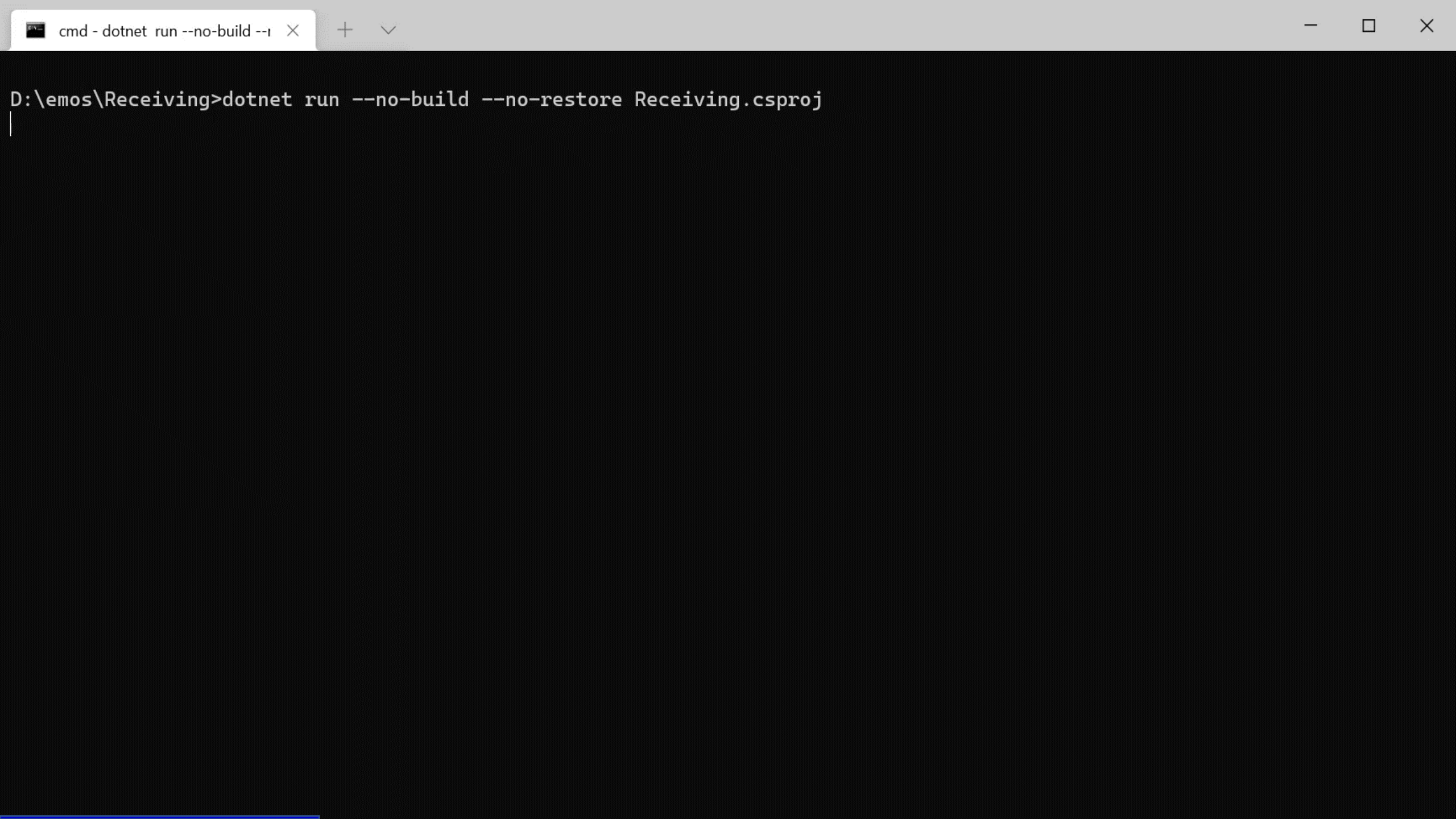
```
var receiver = client.CreateReceiver(source);

message = await receiver.ReceiveMessageAsync();
```

User Callback

```
var options = new ServiceBusProcessorOptions
{
    AutoCompleteMessages = false,
    MaxConcurrentCalls = 1,
    MaxAutoLockRenewalDuration = TimeSpan.FromMinutes(10),
    // PrefetchCount = 10,
    // ReceiveMode = ServiceBusReceiveMode.ReceiveAndDelete,
    // SubQueue = SubQueue.DeadLetter,
    // Identifier = "my-processor-01"
};

var processor = serviceBusClient.CreateProcessor(destination, options);

processor.ProcessMessageAsync += async (ProcessMessageEventArgs x) =>
{
    Console.WriteLine($"Received message with '{x.Message.MessageId}' and content '{x.Message.Body}'");
    //throw new InvalidOperationException();
    await x.CompleteMessageAsync(x.Message);
    syncEvent.TrySetResult(true);
};

processor.ProcessErrorAsync += (ProcessErrorEventArgs x) =>
{
    Console.WriteLine($"EntityPath: {x.EntityPath}");
    Console.WriteLine($"EntityPath: {x.Identifier}");
    Console.WriteLine($"FullyQualifiedNamespace: {x.FullyQualifiedNamespace}");
    Console.WriteLine($"ErrorSource: {x.ErrorSource}");
    Console.WriteLine($"Exception: {x.Exception}");

    return Task.CompletedTask;
};

await processor.StartProcessingAsync();
//await processor.StopProcessingAsync();
```

cmd - dotnet run --no-build --r

```
D:\emos\Receiving>dotnet run --no-build --no-restore Receiving.csproj
```


cmd - dotnet run --no-build --r
D:\emos\Receiving>dotnet run --no-build --no-restore Receiving.csproj

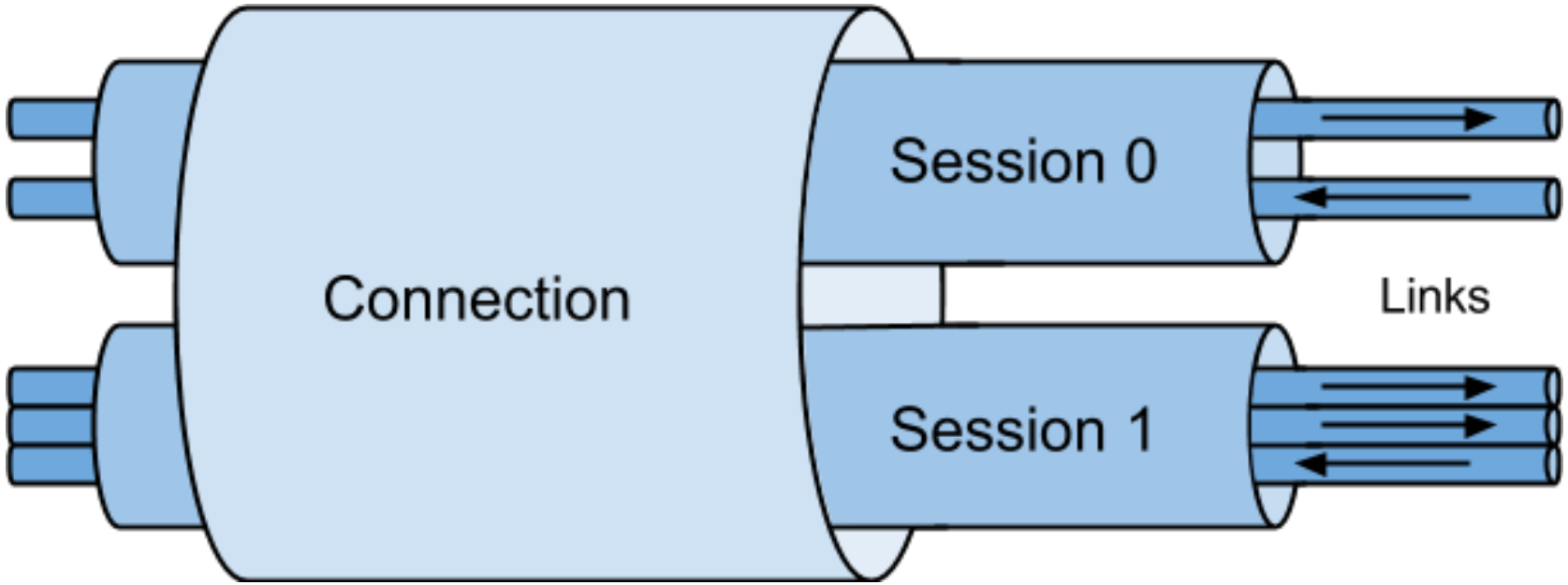
Something to note…

```
message = await receiver.ReceiveMessageAsync();
// or
var operationTimeout = TimeSpan.FromSeconds(4);
message = await receiver.ReceiveMessageAsync(operationTimeout);
```

```
var messageCount = 5;
messages = await receiver.ReceiveMessagesAsync(messageCount);
// or
messages = await receiver.ReceiveMessagesAsync(messageCount, operationTimeout);
```

Connections Management

Connections and Connection Pooling

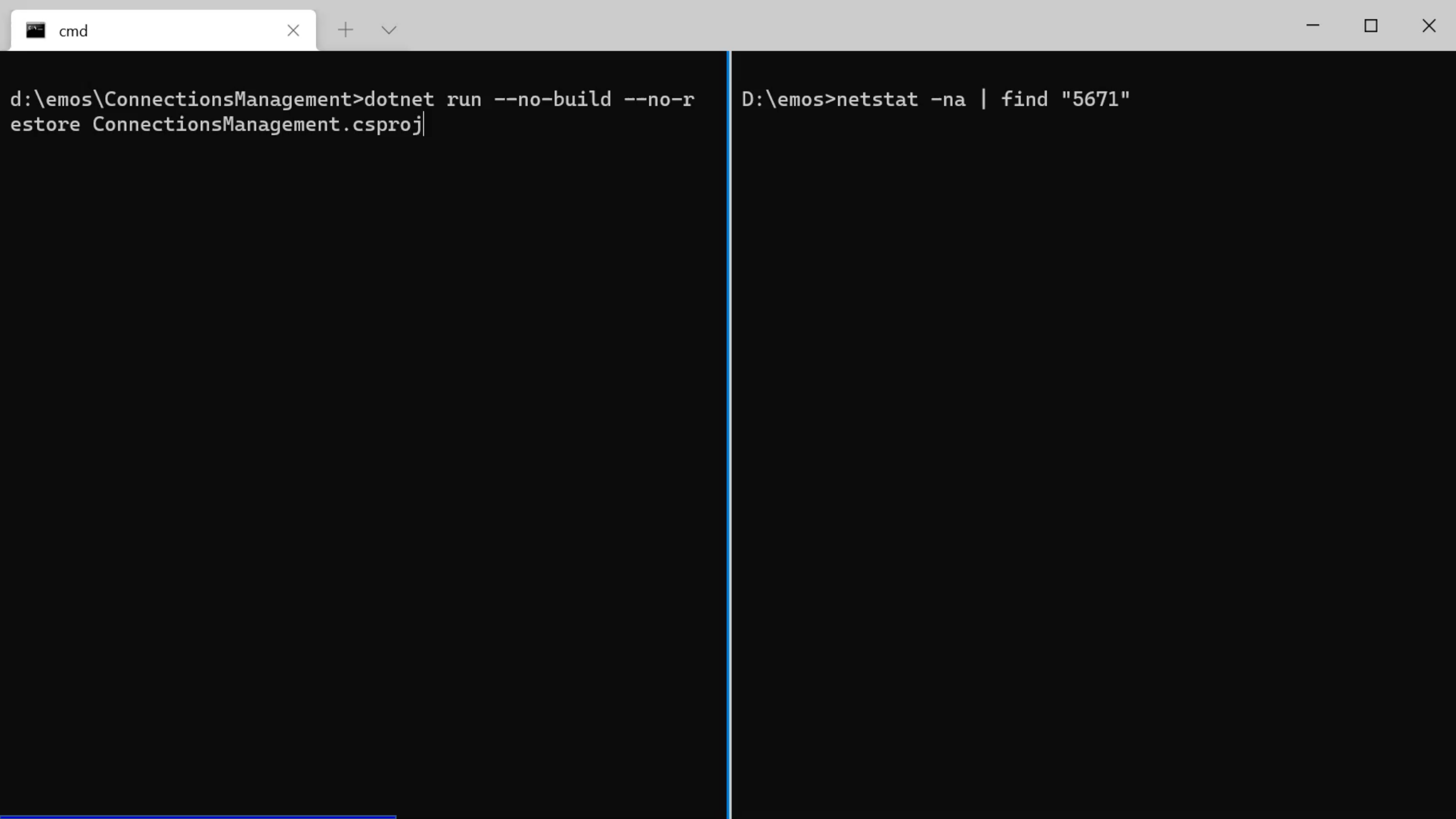


Shared Client

```
var serviceBusClient = new ServiceBusClient(connectionString);

var sender = serviceBusClient.CreateSender(destination);
await sender.SendMessageAsync(new ServiceBusMessage("Deep Dive"));
var receiver = serviceBusClient.CreateReceiver(destination);
await receiver.ReceiveMessageAsync();
```

Multiple Clients

```
var serviceBusClient1 = new ServiceBusClient(connectionString);
var serviceBusClient2 = new ServiceBusClient(connectionString);

var sender = serviceBusClient1.CreateSender(destination);
await sender.SendMessageAsync(new ServiceBusMessage("Deep Dive"));
var receiver = serviceBusClient2.CreateReceiver(destination);
await receiver.ReceiveMessageAsync();
```


cmd
d:\emos\ConnectionsManagement>dotnet run --no-build --no-r
estore ConnectionsManagement.csproj
D:\emos>netstat -na | find "5671"

Recoverability (Retries)

```
var options = new ServiceBusClientOptions
{
    TransportType = ServiceBusTransportType.AmqpTcp,
    //TransportType = ServiceBusTransportType.AmqpWebSockets,
    RetryOptions = new ServiceBusRetryOptions()
    {
        Mode = ServiceBusRetryMode.Fixed,
        //Delay = TimeSpan.FromSeconds(3),
        //MaxRetries = 5,
        //TryTimeout
        //MaxDelay
        //CustomRetryPolicy = new MyRetryPolicy()
    }
};

var cancellationToken = new CancellationToken();

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

Message Scheduling

Schedule w/o a receipt

```
var sender = serviceBusClient.CreateSender(destination);
var due = DateTimeOffset.UtcNow.AddSeconds(10);

await sender.ScheduleMessageAsync(new ServiceBusMessage($"Deep Dive 1 + {due}"), due);
Console.WriteLine($"{DateTimeOffset.UtcNow}: Message scheduled first");
```

Schedule with a receipt

```
var sequenceId = await sender.ScheduleMessageAsync(new ServiceBusMessage($"Deep Dive 2 + {due}"), due);
Console.WriteLine($"{DateTimeOffset.UtcNow}: Message scheduled second");

await sender.CancelScheduledMessageAsync(sequenceId);
Console.WriteLine($"{DateTimeOffset.UtcNow}: Canceled second");
```

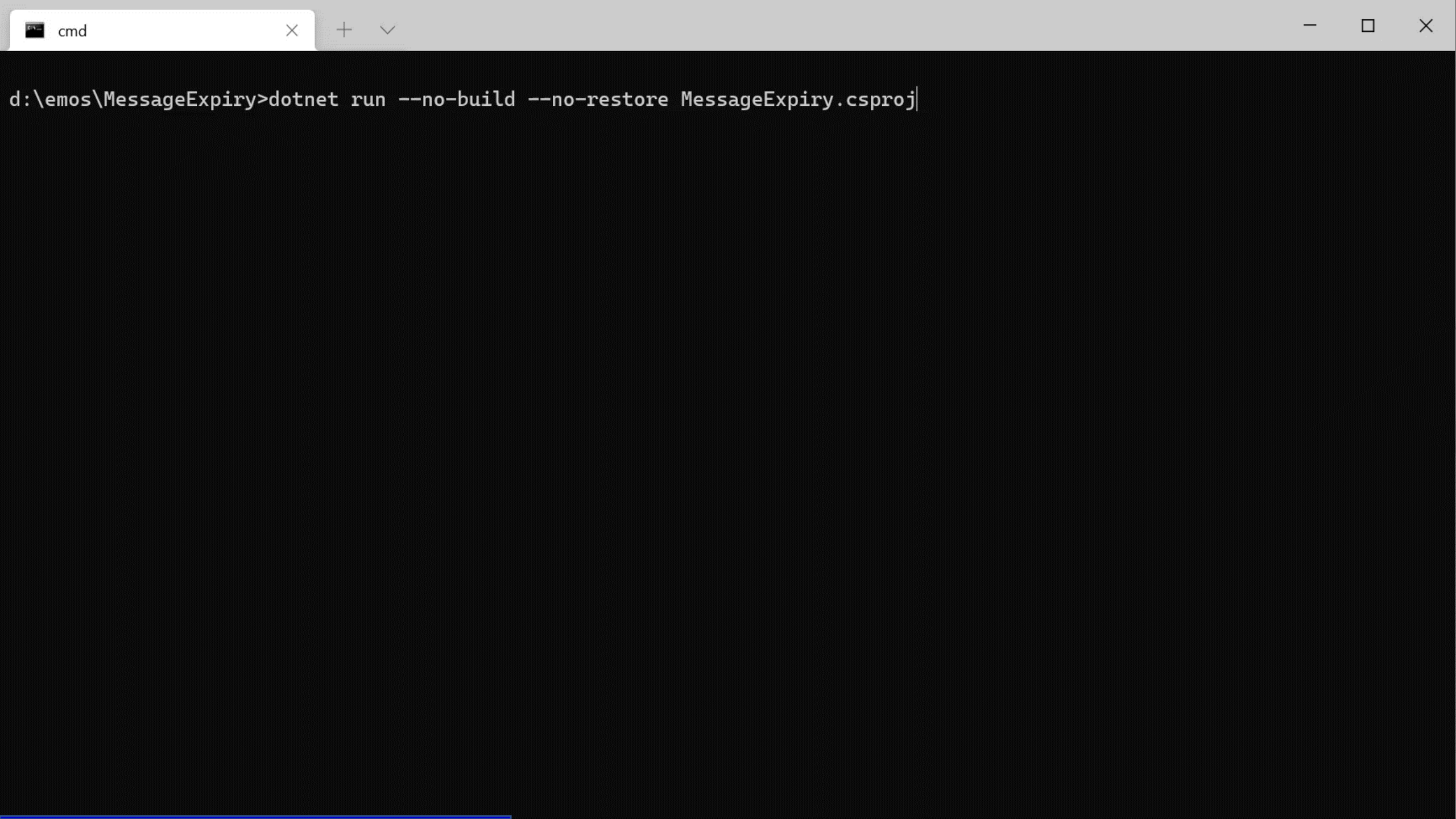
cmd

```
d:\emos\MessageScheduling>dotnet run --no-build --no-restore MessageScheduling.csproj
```

Message Expiration

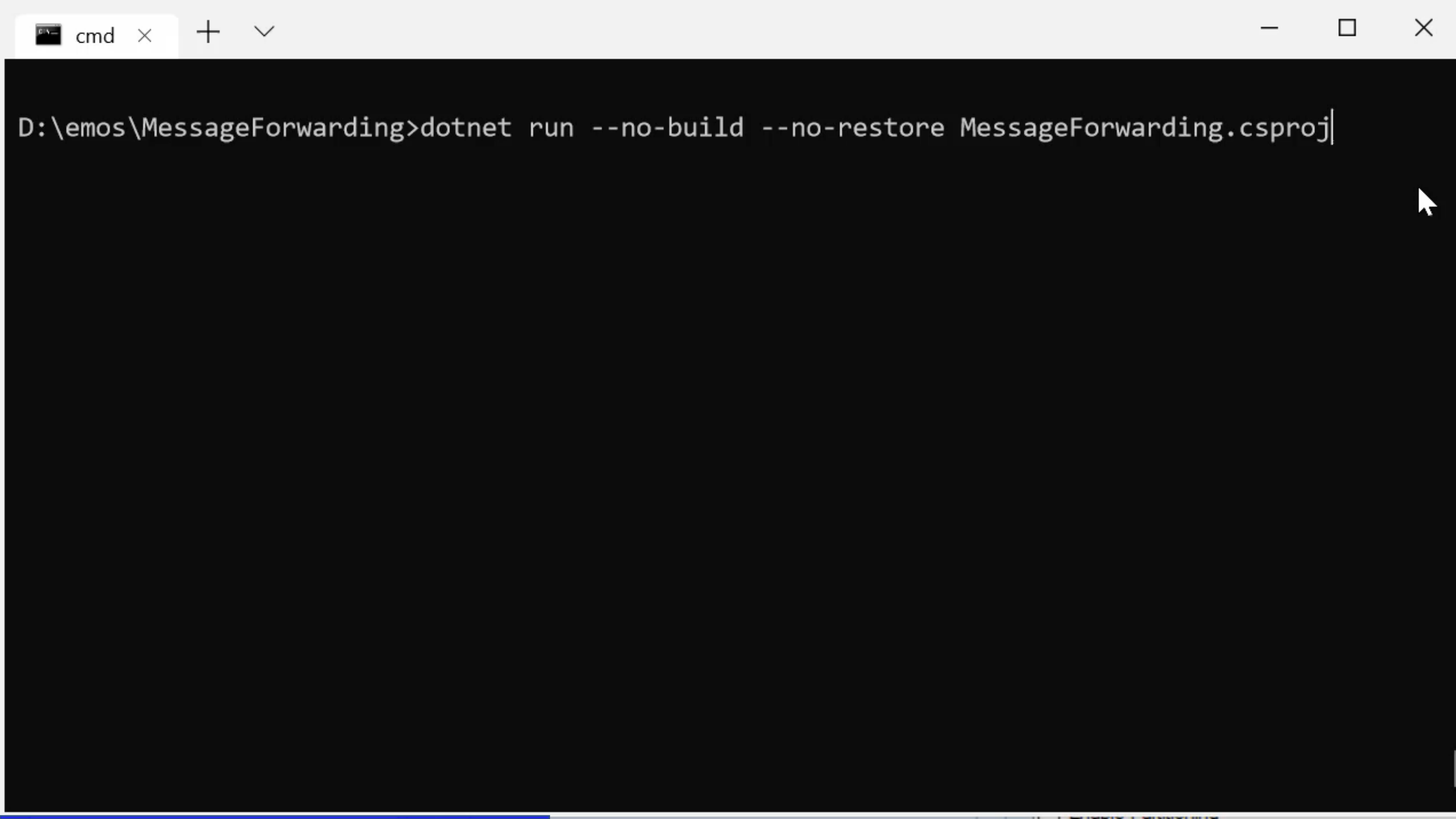
```
var sender = serviceBusClient.CreateSender(destination);

var message = new ServiceBusMessage("Deep Dive")
{
    // if not set the default time to live on the queue counts
    TimeToLive = TimeSpan.FromSeconds(3)
};

await sender.SendMessageAsync(message);
Console.WriteLine("Message sent");

// Note that expired messages are only purged and moved to the DLQ when there is at least one
// active receiver pulling from the main queue or subscription; that behavior is by design.
await Prepare.SimulateActiveReceiver(serviceBusClient, destination);
```

```
var options = new ServiceBusProcessorOptions
{
    AutoCompleteMessages = false,
    MaxConcurrentCalls = 1
};
var processor = client.CreateProcessor(entity, options);

processor.ProcessMessageAsync += async args =>
{
    await args.AbandonMessageAsync(args.Message);
    Console.WriteLine("(Emulating active receiver w/o receiving messages)");
    await Task.Delay(TimeSpan.FromSeconds(5));
};
```


cmd
d:\emos\MessageExpiry>dotnet run --no-build --no-restore MessageExpiry.csproj

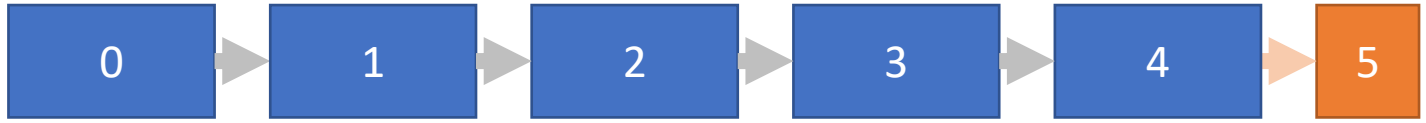
Message Forwarding

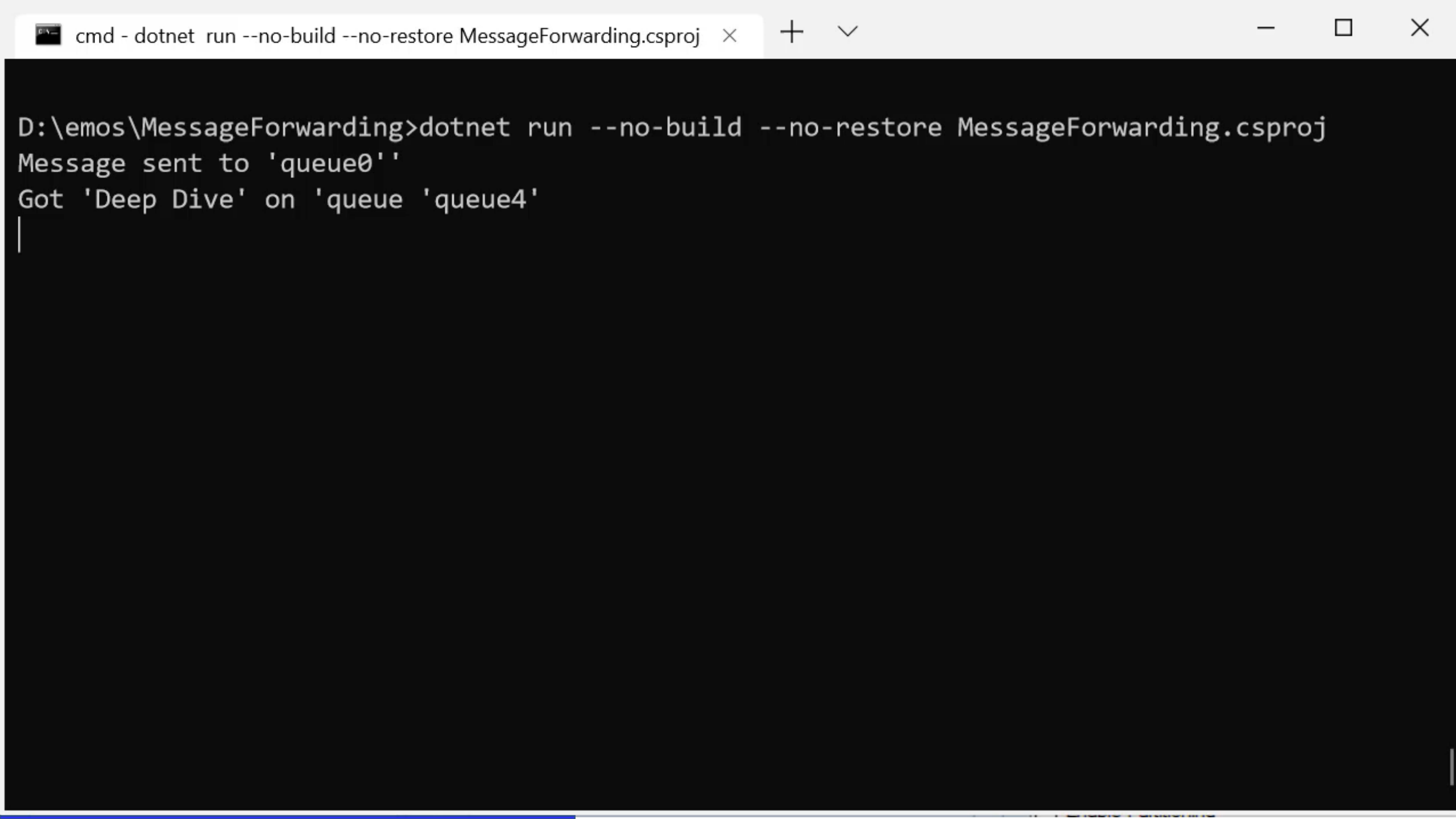
```
var options = new CreateQueueOptions("queue5");
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue4");
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue3")
{
    ForwardTo = "queue4"
};
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue2")
{
    ForwardTo = "queue3"
};
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue1")
{
    ForwardTo = "queue2"
};
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue0")
{
    ForwardTo = "queue1"
};
await client.CreateQueueAsync(options);
```


cmd
D:\emos\MessageForwarding>dotnet run --no-build --no-restore MessageForwarding.csproj

```
var options = new CreateQueueOptions("queue5");
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue4");
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue3")
{
    ForwardTo = "queue4"
};
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue2")
{
    ForwardTo = "queue3"
};
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue1")
{
    ForwardTo = "queue2"
};
await client.CreateQueueAsync(options);

options = new CreateQueueOptions("queue0")
{
    ForwardTo = "queue1"
};
await client.CreateQueueAsync(options);
```

```
QueueProperties properties = await client.GetQueueAsync("queue4");
properties.ForwardTo = "queue5";

await client.UpdateQueueAsync(properties);
```




cmd - dotnet run --no-build --no-restore MessageForwarding.csproj
D:\emos\MessageForwarding>dotnet run --no-build --no-restore MessageForwarding.csproj
Message sent to 'queue0''
Got 'Deep Dive' on 'queue 'queue4'

Message Dead-lettering

queue/$DeadLetterQueue

Setup

```
var createQueueOptions = new CreateQueueOptions(destination)
{
    DeadLetteringOnMessageExpiration = true, // default false
    MaxDeliveryCount = 1
};
```

```
var sender = serviceBusClient.CreateSender(destination);

var message1 = new ServiceBusMessage("Deep Dive 1");
message1.Subject = "first";
message1.TimeToLive = TimeSpan.FromSeconds(1);
await sender.SendMessageAsync(message1);
Console.WriteLine("Sent first message");

var message2 = new ServiceBusMessage("Deep Dive 2");
message2.Subject = "second";
await sender.SendMessageAsync(message2);
Console.WriteLine("Sent second message");

var message3 = new ServiceBusMessage("Deep Dive 3");
message3.Subject = "third";
await sender.SendMessageAsync(message3);
Console.WriteLine("Sent third message");
```

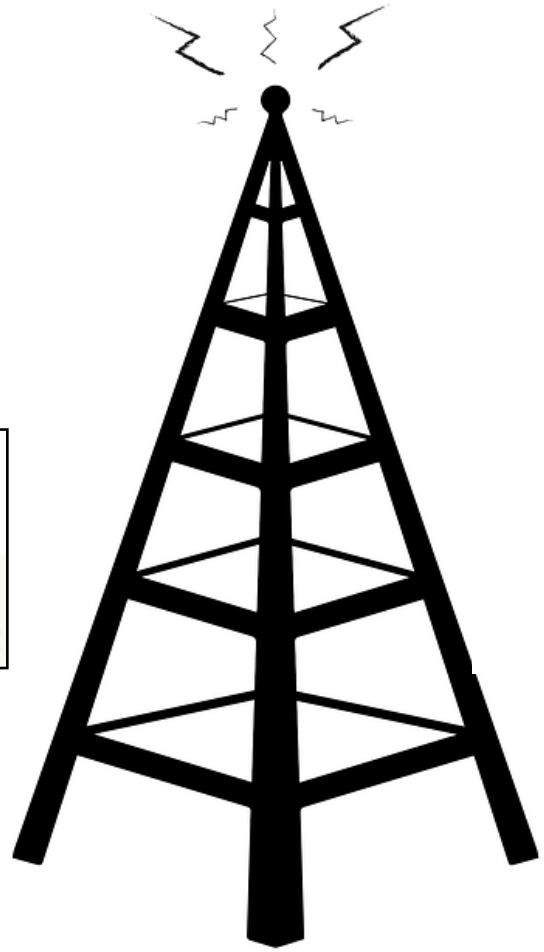
Message pump

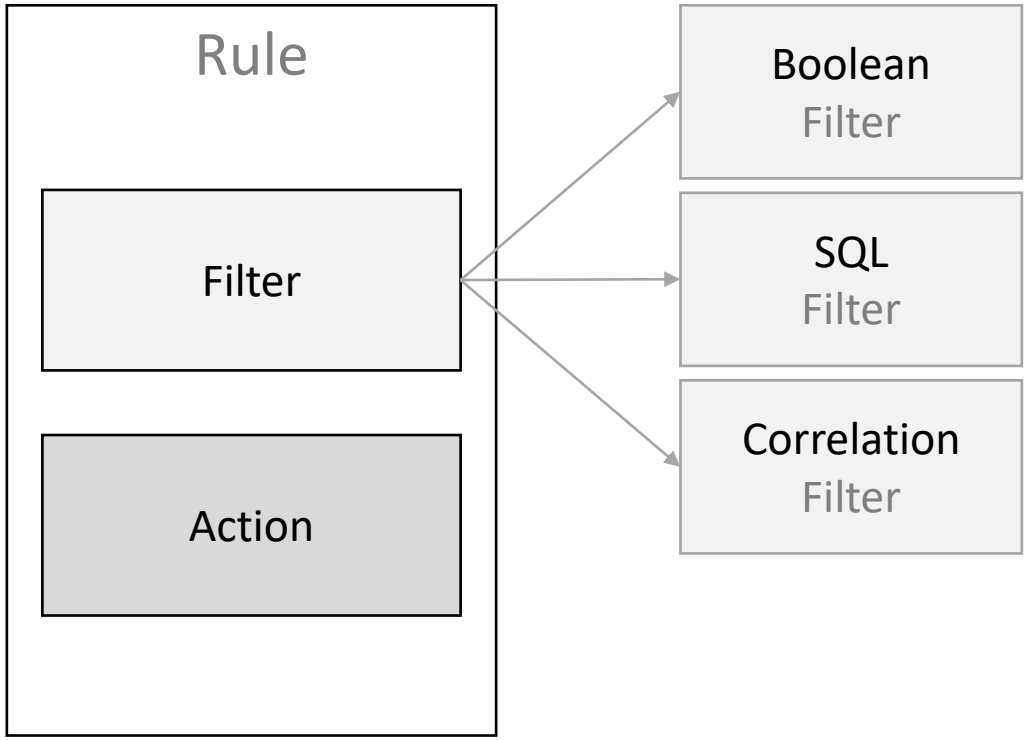
```
var processor = serviceBusClient.CreateProcessor(destination,
    new ServiceBusProcessorOptions { AutoCompleteMessages = false, MaxConcurrentCalls = 3 });

processor.ProcessMessageAsync += async args =>
{
    switch (args.Message.Subject)
    {
        case "first":
            throw new InvalidOperationException("Should never received the first message.");

        case "second":
            await args.AbandonMessageAsync(args.Message);
            Console.WriteLine("Abandon the second message --> delivery count will exceed the maximum");
            break;

        case "third":
            Console.WriteLine("Dead-letter the third message explicitly");
            await args.DeadLetterMessageAsync(args.Message, new Dictionary<string, object>
            {
                { "Reason", "Because we can!" },
                { "When", DateTime.UtcNow }
            }); // Task
            break;
    }
};
```

Pub/Sub

Publish/Subscribe

- Senders only knows Topics
- Receivers only know about Subscriptions
- Subscriptions have Filters (rules) and Actions
- Rules evaluated using OR logic




Rule
Filter
Action
Boolean
Filter
SQL
Filter
Correlation
Filter

SQL vs Correlation Filters

Correlation is faster than SQL

No short-circuiting for SQL

Setup

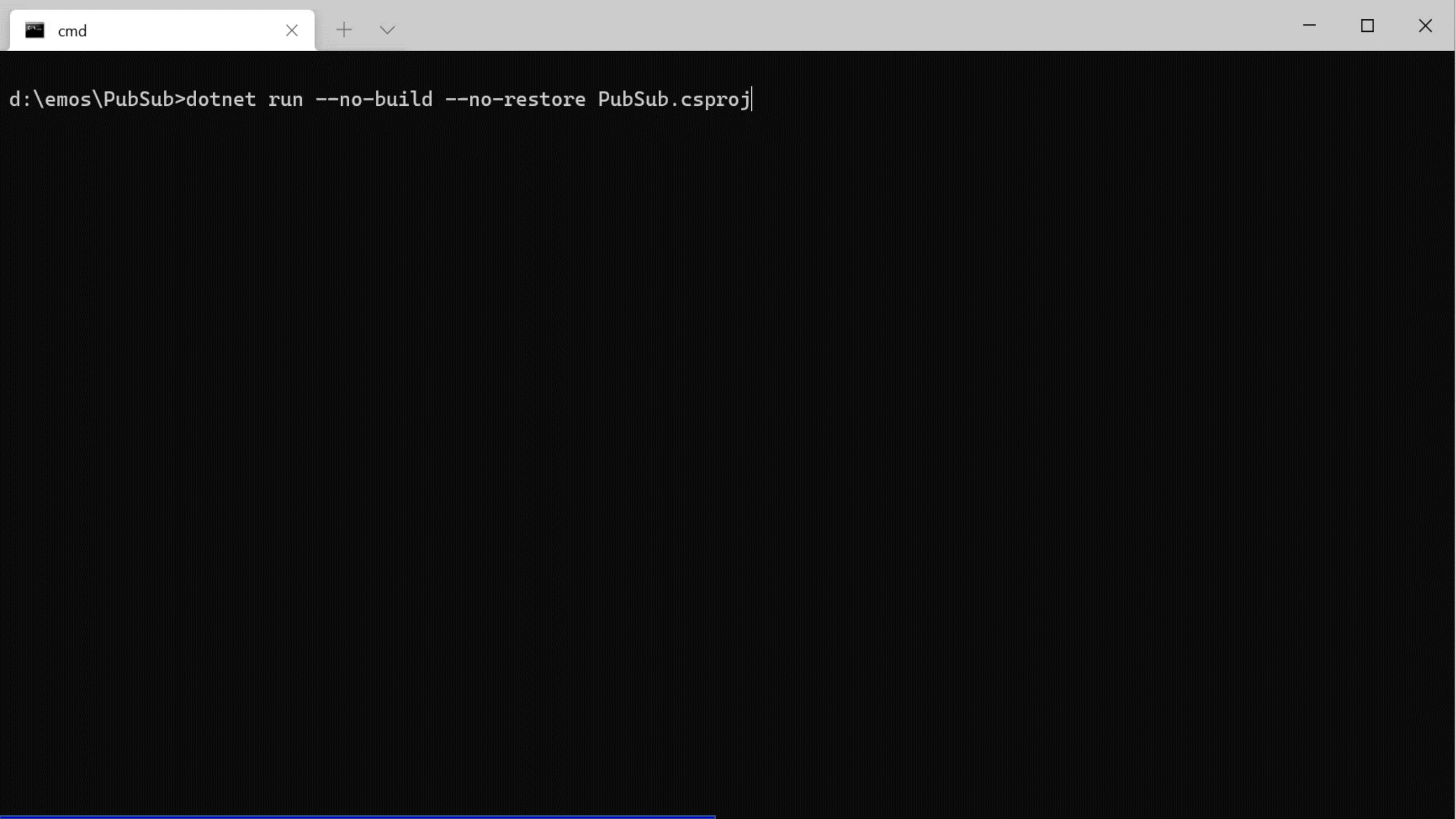
```
var ruleOptions = new CreateRuleOptions
{
    Name = "MessagesWithRushlabel",
    Filter = new CorrelationRuleFilter
    {
        Subject = "rush"
    },
    Action = null
};
```

```
ruleOptions = new CreateRuleOptions
{
    Name = "MessagesWithCurrencyGBP",
    Filter = new SqlRuleFilter("currency = 'GBP' OR currency = '£'"),
    Action = new SqlRuleAction("SET status = 'Keep calm and carry on'")
};
```

Invocation

```
topicName = "topic";
rushSubscription = "alwaysInRush";
currencySubscription = "maybeRich";
```

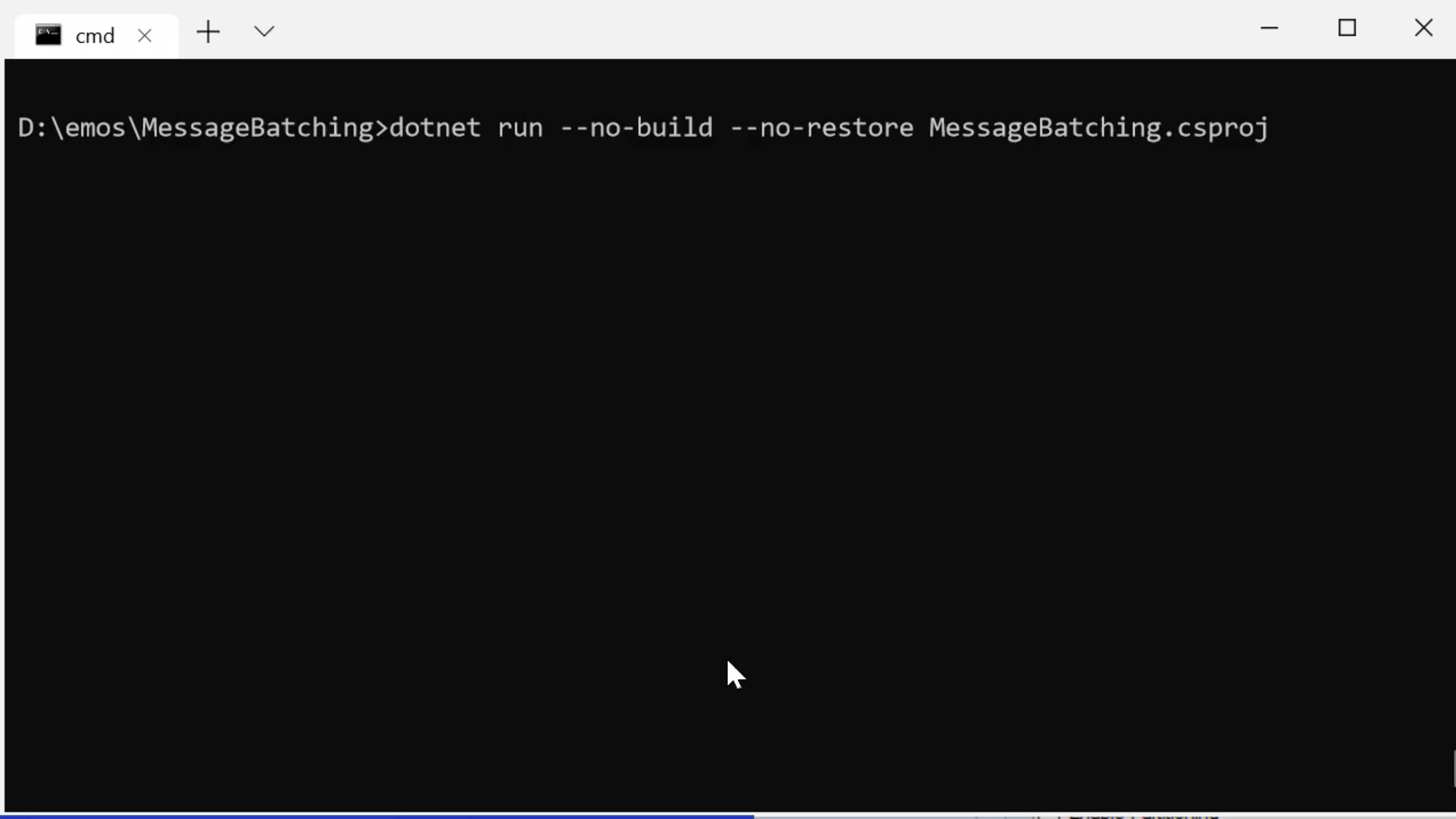
```
var message = new ServiceBusMessage("No time, gotta rush!") { Subject = "rush" };
await sender.SendMessageAsync(message);

message = new ServiceBusMessage("I'm rich! I have 1,000!") { Subject = "rush" };
message.ApplicationProperties.Add("currency", "GBP");
await sender.SendMessageAsync(message);
```


cmd
d:\emos\PubSub>dotnet run --no-build --no-restore PubSub.csproj

Message Batching

Sending a batch

```
var messages = new List<ServiceBusMessage>();
for (var i = 0; i < 10; i++)
{
    var message = new ServiceBusMessage($"Deep Dive{i}");
    messages.Add(message);
}

await sender.SendMessagesAsync(messages);
```


cmd
D:\emos\MessageBatching>dotnet run --no-build --no-restore MessageBatching.csproj

Sending a large batch

```
for (var i = 0; i < 6500; i++)
{
    var message = new ServiceBusMessage($"Deep Dive{i}");
    messages.Add(message);
}

await sender.SendMessagesAsync(messages);
```

cmd

```
d:\emos\MessageBatching>dotnet run --no-build --no-restore MessageBatching.csproj
```

Safe batching

```
var batchOptions = new CreateMessageBatchOptions
{
    MaxSizeInBytes = 150
};
var batch = await sender.CreateMessageBatchAsync(batchOptions);
```

```
var message = new ServiceBusMessage("hello");
for (var i = 0; i < 5; i++)
{
    Console.WriteLine(batch.TryAddMessage(message)
        ? $"Message added to the batch (size: {batch.SizeInBytes})"
        : $"Message cannot fit the batch size {batch.MaxSizeInBytes}");
}

await sender.SendMessagesAsync(batch);
```

cmd

```
D:\emos\SafeBatching>dotnet run --no-build --no-restore SafeBatching.csproj
```

Messages larger than 256KB?

Upgrade to Premium Tier for messages up-to 100MB

Claim Check Pattern

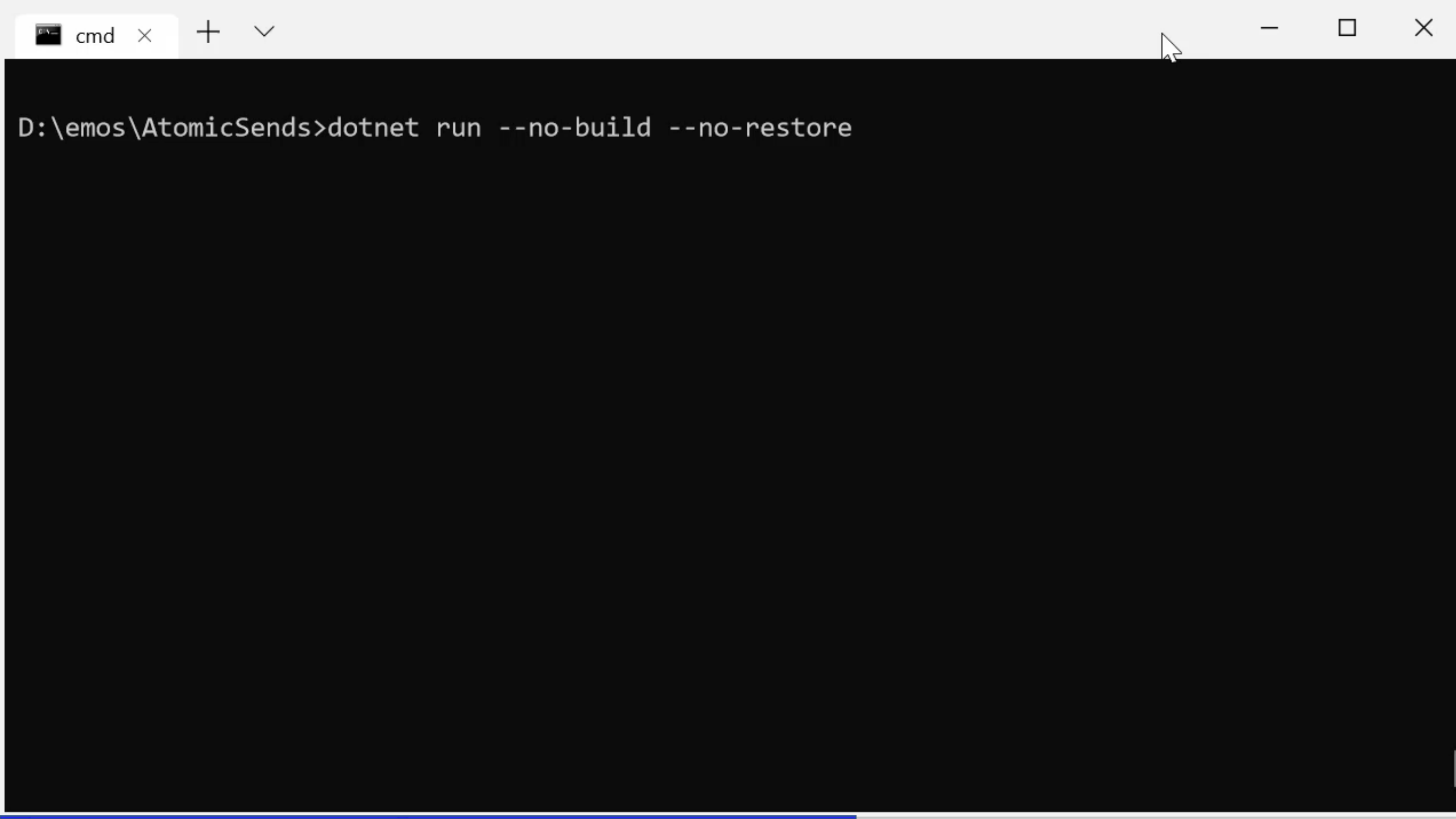
Extend ServiceBus Sender, Receiver, and Processor (SDK samples)

~~ServiceBus.AttachmentPlugin~~ 

Atomic Sends

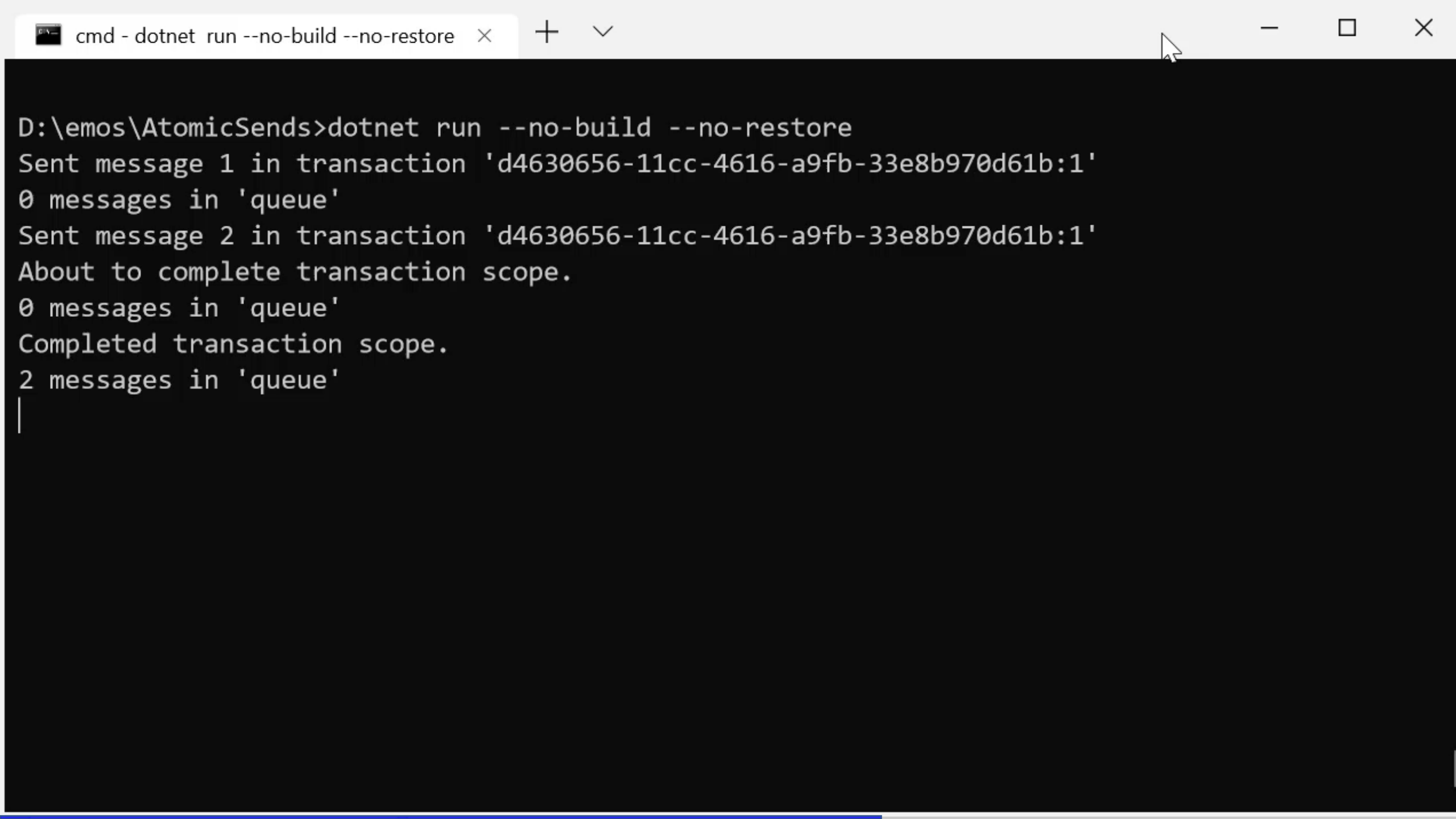
Individual sends

```
using (var scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
{
    var message = new ServiceBusMessage("Deep Dive 1");
    await sender.SendMessageAsync(message);

    message = new ServiceBusMessage("Deep Dive 2");
    await sender.SendMessageAsync(message);

    scope.Complete();
}
```


cmd
D:\emos\AtomicSends>dotnet run --no-build --no-restore

100 limit

```
for (var i = 0; i < 101; i++)
{
    messages.Add(new ServiceBusMessage($"Deep Dive {i}"));
}
```


cmd - dotnet run --no-build --no-restore
D:\emos\AtomicSends>dotnet run --no-build --no-restore
Sent message 1 in transaction 'd4630656-11cc-4616-a9fb-33e8b970d61b:1'
0 messages in 'queue'
Sent message 2 in transaction 'd4630656-11cc-4616-a9fb-33e8b970d61b:1'
About to complete transaction scope.
0 messages in 'queue'
Completed transaction scope.
2 messages in 'queue'

Atomic sends with a batch

```
for (var i = 0; i < 101; i++)
{
    messages.Add(new ServiceBusMessage($"Deep Dive {i}"));
}

using var scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled);
await sender.SendMessagesAsync(messages);
scope.Complete();
```

cmd

```
D:\emos\AtomicSends>dotnet run --no-build --no-restore AtomicSends.csproj
```

Cross-entity Transaction (Send Via)


Incomir
transfer
A
B
c

```
var initiator = client.CreateSender(inputQueue);
await initiator.SendMessageAsync(new ServiceBusMessage("Deep Dive"));
```

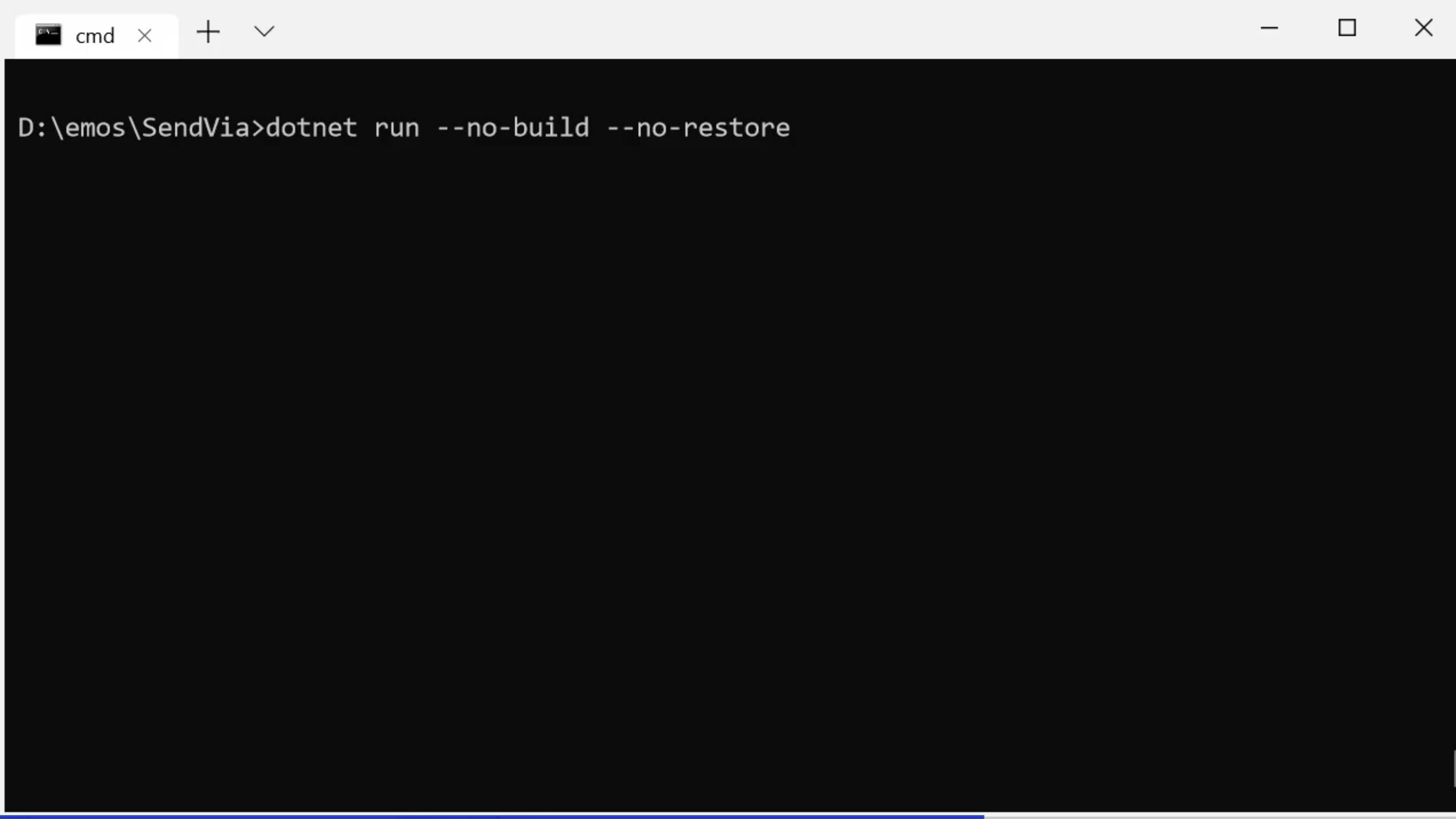
```
var receiver = client.CreateReceiver(inputQueue);
var sender = client.CreateSender(destinationQueue);

var receivedMessage = await receiver.ReceiveMessageAsync();

using (var ts = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
{
    await sender.SendMessageAsync(new ServiceBusMessage("Message for destination"));

    await receiver.CompleteMessageAsync(receivedMessage);

    ts.Complete();
}
```

cmd

```
D:\emos\SendVia>dotnet run --no-build --no-restore
```

When things go wrong

```
using (var ts = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
{
    await sender.SendMessageAsync(new ServiceBusMessage("Message for destination"));

    await receiver.CompleteMessageAsync(receivedMessage);

    throw new Exception("Failure");

    ts.Complete();
}
```

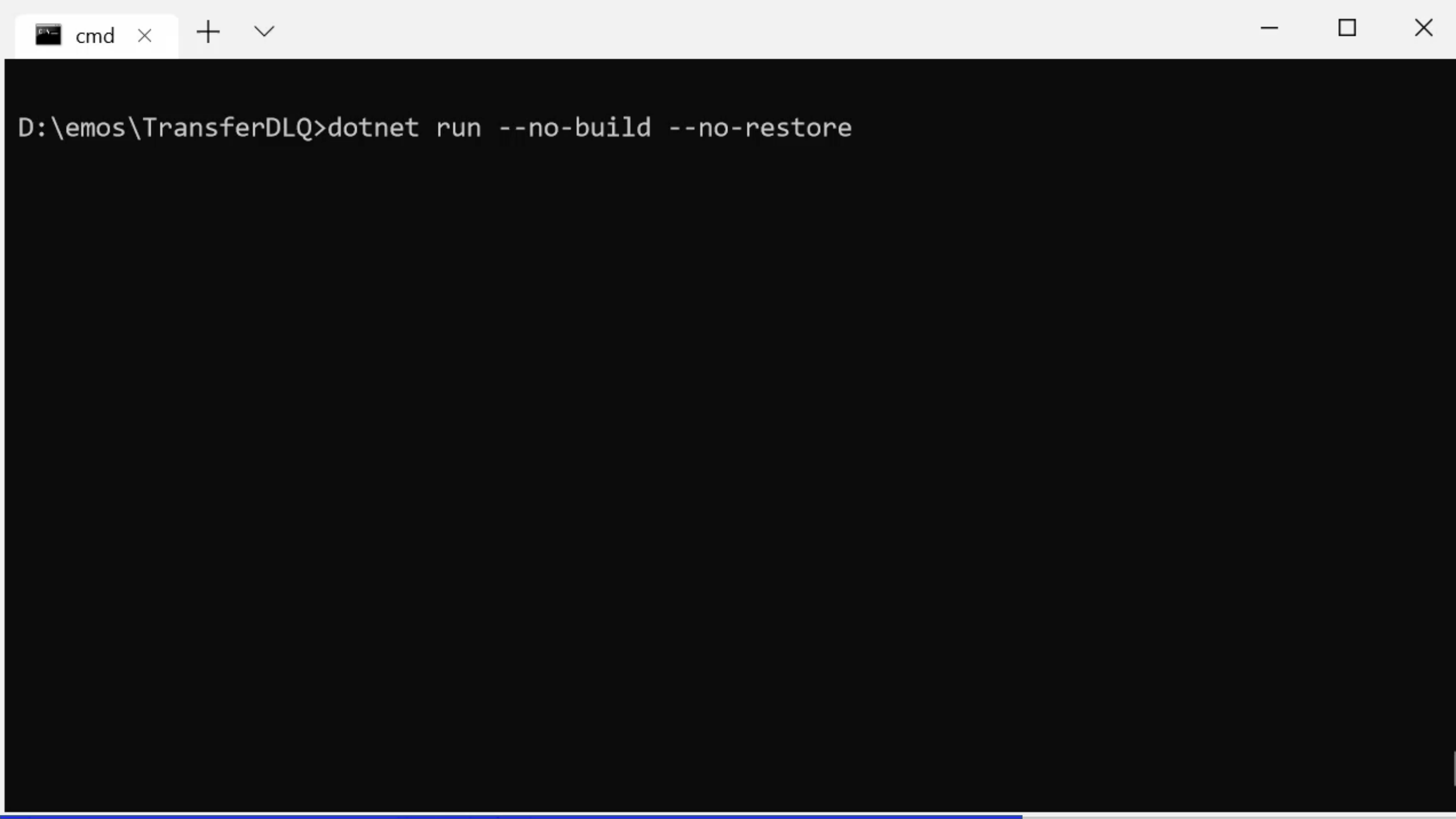

cmd
D:\emos\SendVia>dotnet run --no-build --no-restore

Transfer Dead-lettering

queue/$Transfer/$DeadLetterQueue

```
var initiator = client.CreateSender(inputQueue);
await initiator.SendMessageAsync(new ServiceBusMessage("Deep Dive"));

var receiver = client.CreateReceiver(inputQueue);
var sender = client.CreateSender(destinationQueue);

var receivedMessage = await receiver.ReceiveMessageAsync();

using (var ts = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
{
    await sender.SendMessageAsync(new ServiceBusMessage("Message for destination"));

    await receiver.CompleteMessageAsync(receivedMessage);

    await Prepare.DisableDestination(connectionString, destinationQueue);

    ts.Complete();
}
```

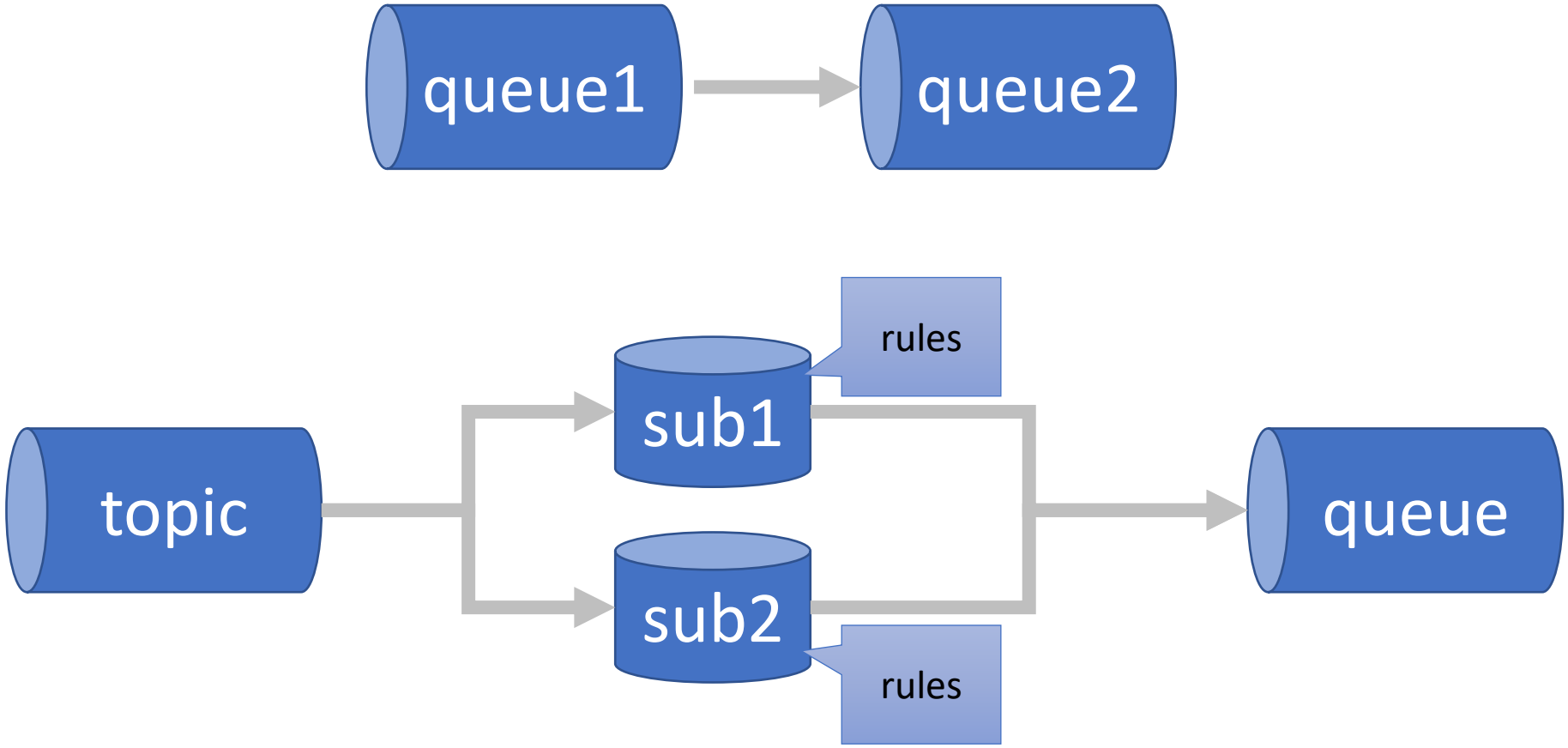
```
QueueProperties properties = await client.GetQueueAsync(destinationQueue);
properties.Status = EntityStatus.SendDisabled;

await client.UpdateQueueAsync(properties);
```


cmd
D:\emos\TransferDLQ>dotnet run --no-build --no-restore

Topologies

What’s a topology?

The way in which parts are interrelated or arranged



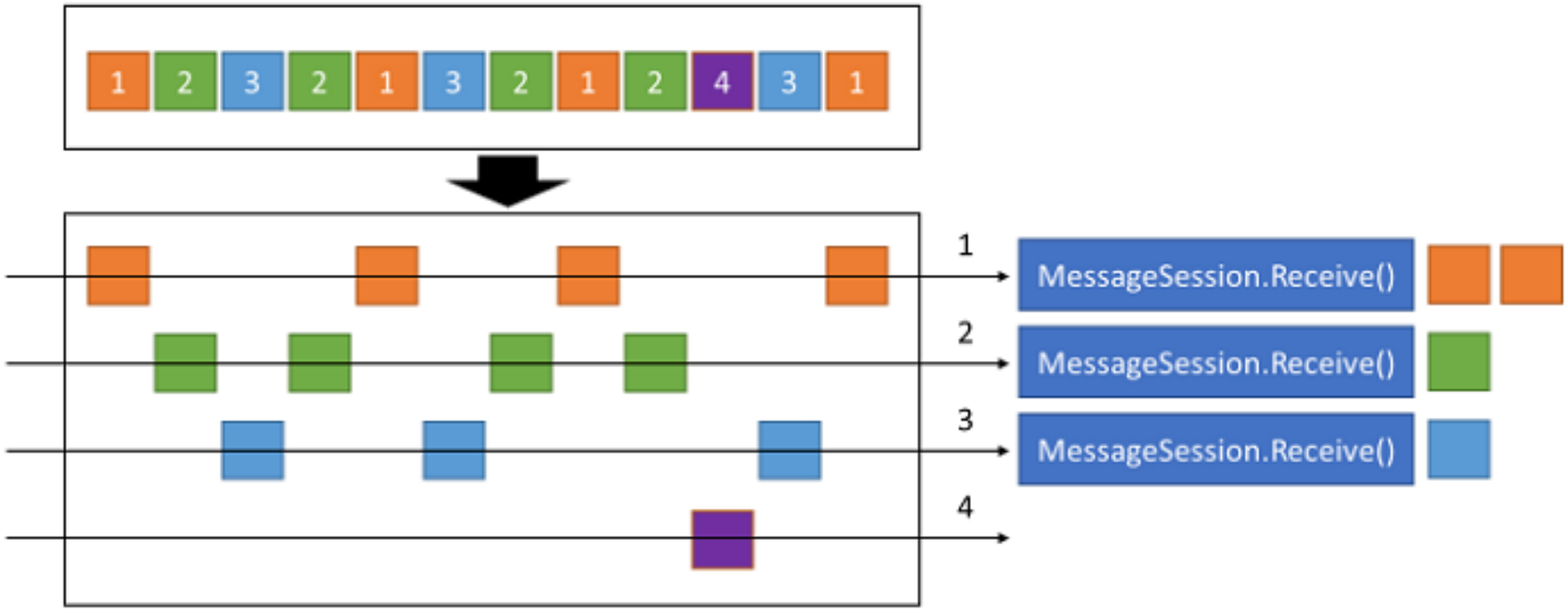
```
var subscriptionOptions = new CreateSubscriptionOptions(topicName, serviceASubscription)
{
    ForwardTo = inputQueue
};
await client.CreateSubscriptionAsync(subscriptionOptions);

subscriptionOptions = new CreateSubscriptionOptions(topicName, serviceBSubscription)
{
    ForwardTo = inputQueue
};
await client.CreateSubscriptionAsync(subscriptionOptions);
```

```
var ruleOptions = new CreateRuleOptions
{
    Name = "MessagesFromServiceA",
    Filter = new CorrelationRuleFilter
    {
        Subject = "rush"
    }
};
await client.CreateRuleAsync(topicName, serviceASubscription, ruleOptions);

ruleOptions = new CreateRuleOptions
{
    Name = "MessagesFromServiceB",
    Filter = new SqlRuleFilter("user.priority in ('high', 'normal', 'low')"),
    Action = new SqlRuleAction("SET sys.Label = user.priority")
};
await client.CreateRuleAsync(topicName, serviceBSubscription, ruleOptions);
```

cmd

```
D:\emos\Topologies>dotnet run --no-build --no-restore
```

Message Sessions (FIFO)


1 2 3 2 1 3 2 1 2 4 3 1
1
2
3
4
MessageSession.Receive()
MessageSession.Receive()
MessageSession.Receive()

```
var queueDescription = new CreateQueueOptions(destination)
{
    RequiresSession = true
};
await client.CreateQueueAsync(queueDescription);
```

```
var messages = new List<ServiceBusMessage>
{
    new("Orange 1") {SessionId = "Orange"},
    new("Green 1") {SessionId = "Green"},
    new("Blue 1") {SessionId = "Blue"},
    new("Green 2") {SessionId = "Green"},
    new("Orange 2") {SessionId = "Orange"},
    new("Blue 2") {SessionId = "Blue"},
    new("Green 3") {SessionId = "Green"},
    new("Orange 3") {SessionId = "Orange"},
    new("Green 4") {SessionId = "Green"},
    new("Purple 1") {SessionId = "Purple"},
    new("Blue 3") {SessionId = "Blue"},
    new("Orange 4") {SessionId = "Orange"}
};
```

```
var options = new ServiceBusSessionProcessorOptions
{
    MaxConcurrentSessions = 1,
    SessionIdleTimeout = TimeSpan.FromSeconds(2),
    //MaxConcurrentCallsPerSession = 10
};
var sessionProcessor = serviceBusClient.CreateSessionProcessor(destination, options);
```

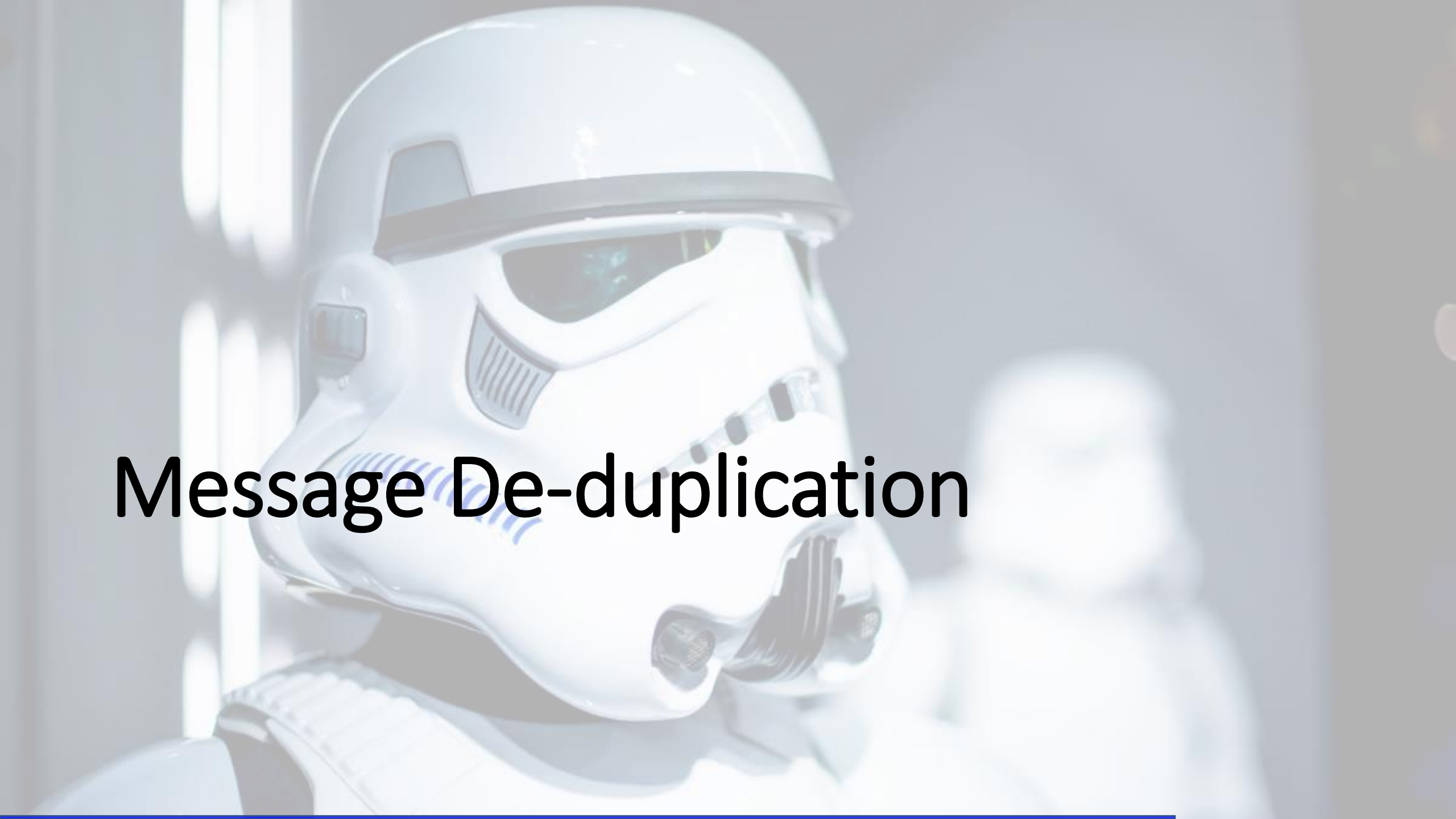
```
sessionProcessor.ProcessMessageAsync += args =>
{
    Console.WriteLine($"Received message for session '{args.SessionId}' ID:'{args.Message.MessageId}' and content:'{args.Message.Body}'");
    return Task.CompletedTask;
};

sessionProcessor.ProcessErrorAsync += args =>
{
    Console.WriteLine($"EntityPath: {args.EntityPath}");
    Console.WriteLine($"FullyQualifiedNamespace: {args.FullyQualifiedNamespace}");
    Console.WriteLine($"ErrorSource: {args.ErrorSource}");
    Console.WriteLine($"Exception: {args.Exception}");
    return Task.CompletedTask;
};

await sessionProcessor.StartProcessingAsync();
```

cmd

```
D:\emos\MessageSessions>dotnet run --no-build --no-restore
```

Session state

- An opaque binary object that can hold data of the size of one message.
- Session state remains as long as it is not cleared up (returning null), even if all messages in a session are consumed.



Message De-duplication

```
var createQueueOptions = new CreateQueueOptions(destination)
{
    RequiresDuplicateDetection = true,
    DuplicateDetectionHistoryTimeWindow = TimeSpan.FromSeconds(20)
};
```

```
var content = "Deep Dive de-duplication";
var messageId = new Guid(content.Take(16).Select(x :char => (byte)x).ToArray()).ToString();

var messages = new List<ServiceBusMessage>
{
    new(content) { MessageId = messageId },
    new(content) { MessageId = messageId },
    new(content) { MessageId = messageId }
};

await sender.SendMessagesAsync(messages);
```

cmd

```
D:\emos\MessageDeduplication>dotnet run --no-build --no-restore
```

Extensibility

Plugins

```
public class PluggableServiceBusSender : ServiceBusSender
{
    private readonly IEnumerable<Func<ServiceBusMessage, Task>> _plugins;

    1 reference
    internal PluggableServiceBusSender(ServiceBusClient client, string queueOrTopicName, IEnumerable<Func<ServiceBusMessage, Task>> plugins)
        : base(client, queueOrTopicName)
    {
        _plugins = plugins;
    }

    1 reference
    public override async Task SendMessageAsync(ServiceBusMessage message, CancellationToken cancellationToken = default)
    {
        foreach (var plugin in _plugins)
        {
            await plugin.Invoke(message);
        }
        await base.SendMessageAsync(message, cancellationToken).ConfigureAwait(false);
    }
}
```

```
public static class ServiceBusClientExtensions
{
    1 reference
    public static PluggableServiceBusSender CreateSender(this ServiceBusClient client, string queueOrTopicName, IEnumerable<Func<ServiceBusMessage, Task>> plugins) =>
        new PluggableServiceBusSender(client, queueOrTopicName, plugins);
```

```
public class Plugins
{
    1 reference
    public static Func<ServiceBusMessage, Task> PrefixPlugin => message =>
    {
        message.Body = new BinaryData($"---PREFIX---{Environment.NewLine}{message.Body}");

        return Task.CompletedTask;
    };
```

```
var serviceBusClient = new ServiceBusClient(connectionString);

var sender = serviceBusClient.CreateSender(queue, new List<Func<ServiceBusMessage, Task>>
{
    Plugins.PrefixPlugin
});

await sender.SendMessageAsync(new ServiceBusMessage("Deep Dive"));

var receiver = serviceBusClient.CreateReceiver(queue, new List<Func<ServiceBusReceivedMessage, Task>>
{
    Plugins.PostfixPlugin
});

var message = await receiver.ReceiveMessageAsync();
```

cmd

```
D:\emos\Plugins>dotnet run --no-build --no-restore
```

BinaryData

```
record Person(string FirstName, string LastName);
```

```
var dataString = new BinaryData("Loki");
Console.WriteLine(dataString);
```

```
var person = new Person("Loki", "Sanekat-Feldman");
var data = new BinaryData(person);
Console.WriteLine(data.ToString());
```

```
var deserialized = data.ToObjectFromJson<Person>();
Console.WriteLine(deserialized.ToString());
```

Functions

In-Proc SDK

Microsoft.NET.Sdk.Functions
Microsoft.Azure.WebJobs.Extensions.ServiceBus

Isolated Worker SDK

Microsoft.Azure.Functions.Work
Microsoft.Azure.Functions.Worker.Extensions.ServiceBuser

```
[FunctionName("InProcFunction")]
0 references
public async Task Run(
    [ServiceBusTrigger("myqueue", Connection = "ASB")]ServiceBusReceivedMessage message,
    ServiceBusMessageActions messageActions)
{
    logger.LogInformation($"Dead-lettering message with ID: {message.MessageId}");

    await messageActions.DeadLetterMessageAsync(message);
}
```


```
[Function("IsolatedWorkerFunction")]
0 references
public async Task Run(
    [ServiceBusTrigger("myqueue", Connection = "ASB")] string body,
    string messageId)
{
    logger.LogInformation($"Cannot dead-letter message with ID: {messageId}");
}
```

Azure CLI

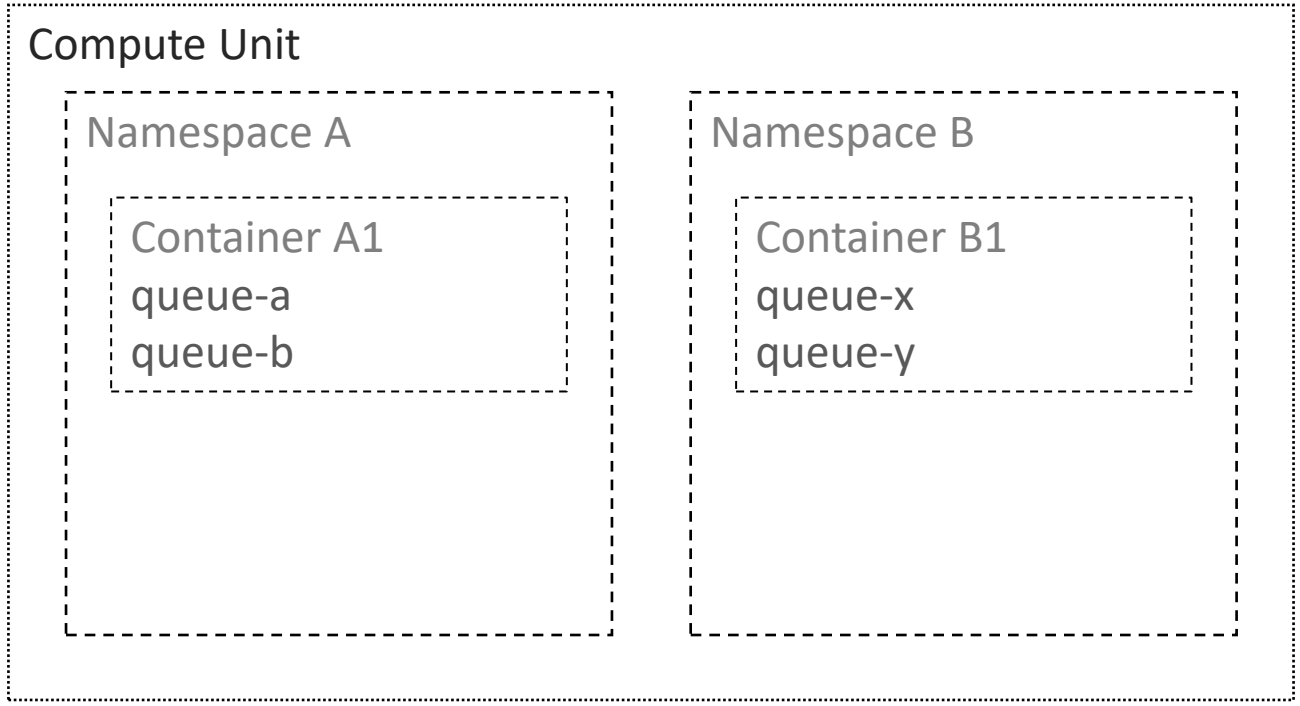
az servicebus --help

https://docs.microsoft.com/en-us/cli/azure/servicebus

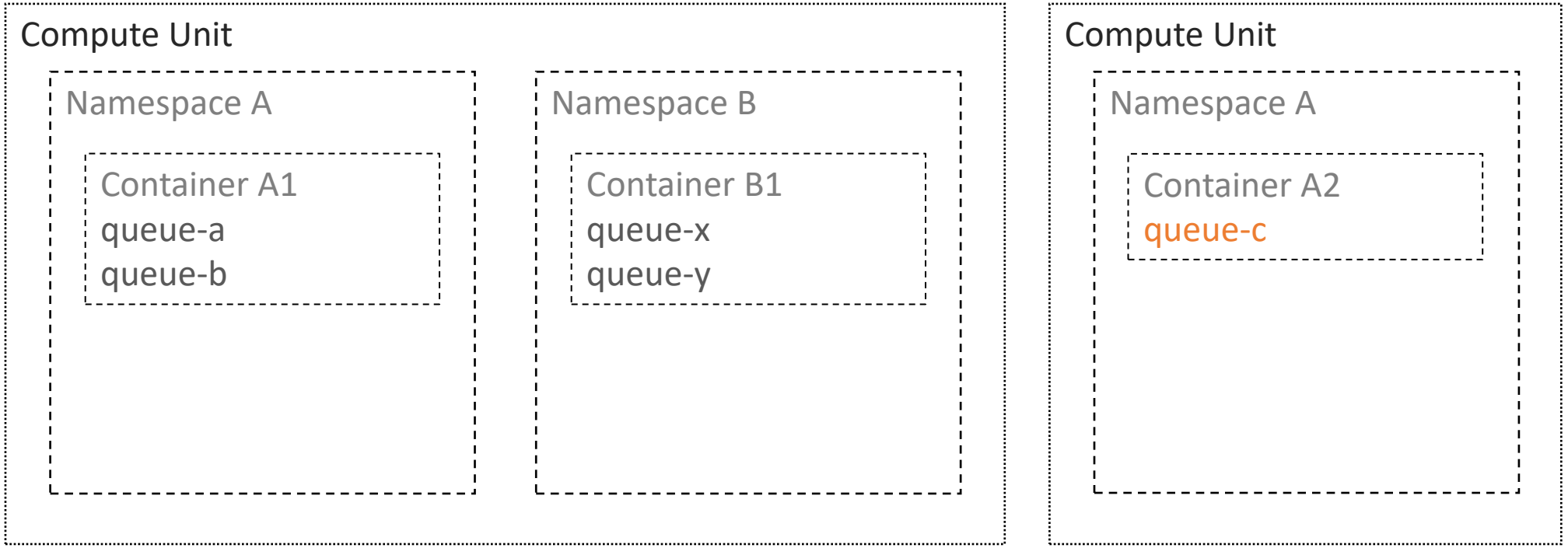
Performance

Standard vs Premium

Standard Namespace Deployment

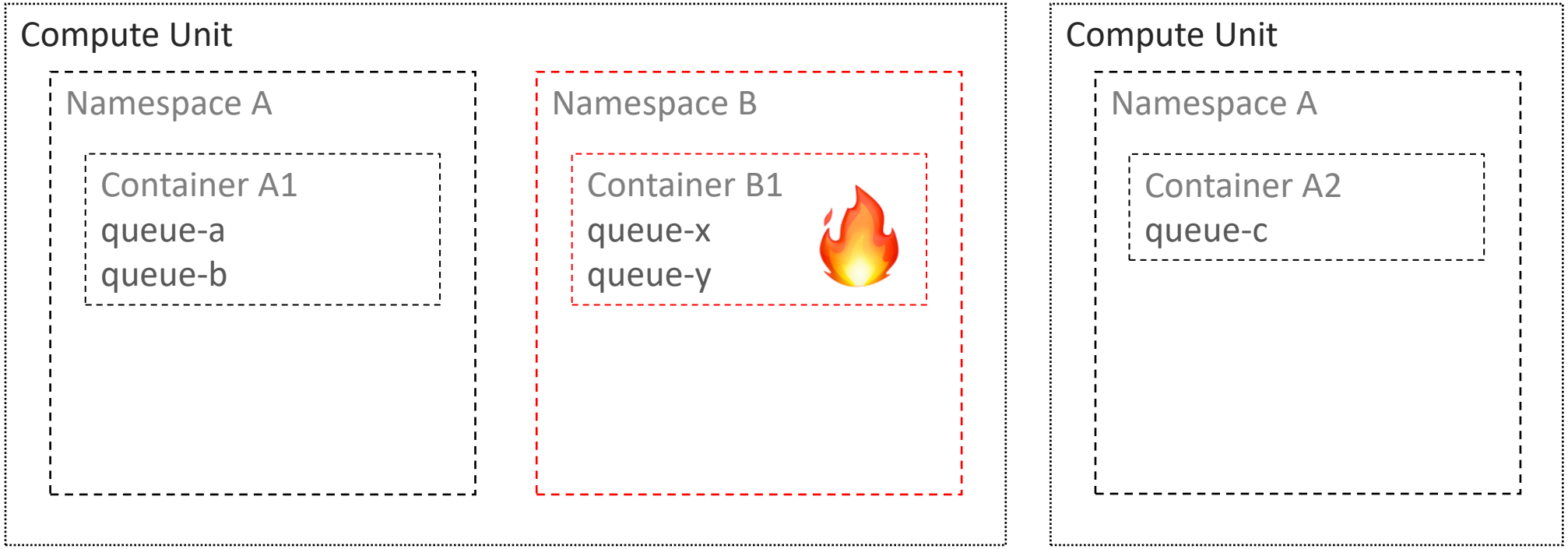


Uneven Performance



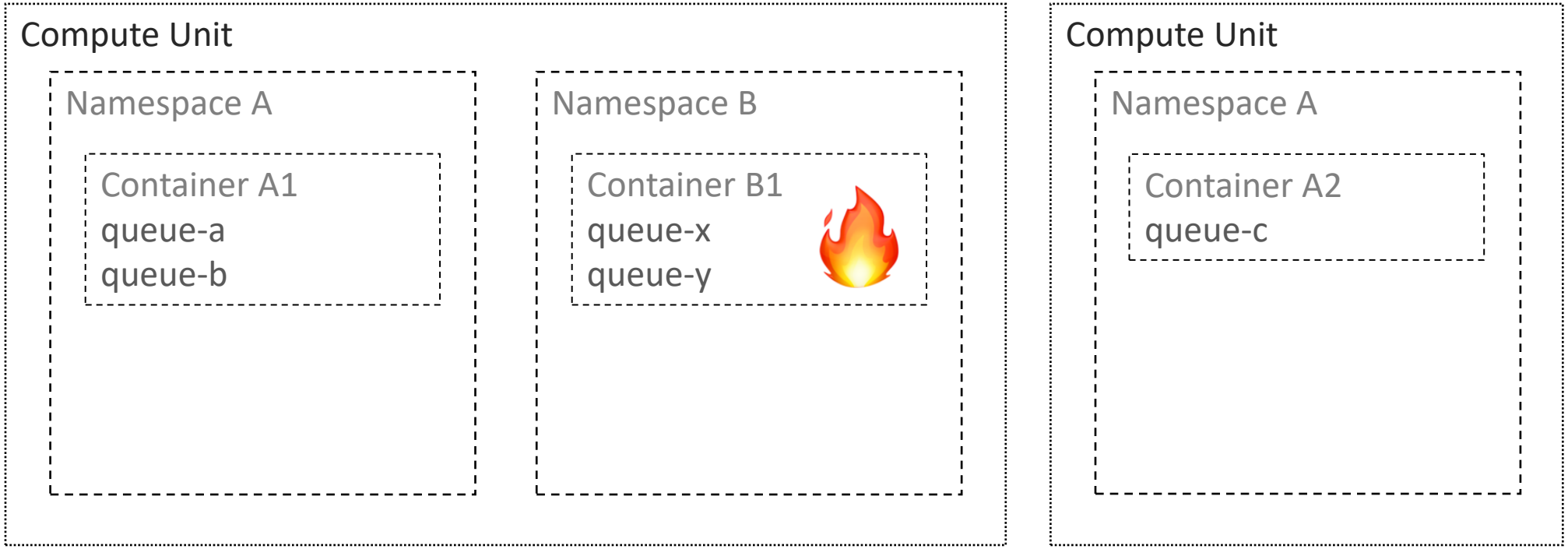
queue-a: 20 msg/sec
queue-c: 200 msg/sec

Noisy Neighbour Effect



ServiceBusFailureReason.ServiceBusy: The request was terminated because the entity is being throttled. Please wait 10 seconds and try again

Credits System



Namespace A: 1000 credits
Namespace B: 1000 credits

Premium or Standard?

- 100MB message size
- Dedicated resources
- Pred
- Scale
- Geo-
- Azur
- VNE
- JMS
- and

- Lower budget requirement
- Multiple namespaces (256KB)

Tooling

workerservice-applicationinsights

Service Bus Namespace

Search (Ctrl+/)

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems

Settings

Shared access policies
Geo-Recovery
Migrate to premium
Encryption (preview)
Properties
Locks
Export template

Entities

Queues

+ Queue + Topic Convert Refresh Delete

1/2

Incoming Requests (Sum) workerservice-applic... 39

Successful Requests ... workerservice-applic... 39

Server Errors. (Sum) workerservice-applic... 0

Incoming Messages (Sum) workerservice-applic... 1

Outgoing Messages (Sum) workerservice-applic... 1

Queues Topics

Search to filter items...

Name	Status	max size	enable partitioning
queue3	Active	1 GB	false
segmented/queue1	Active	1 GB	false
segmented/queue2	Active	1 GB	false

SUMMARY

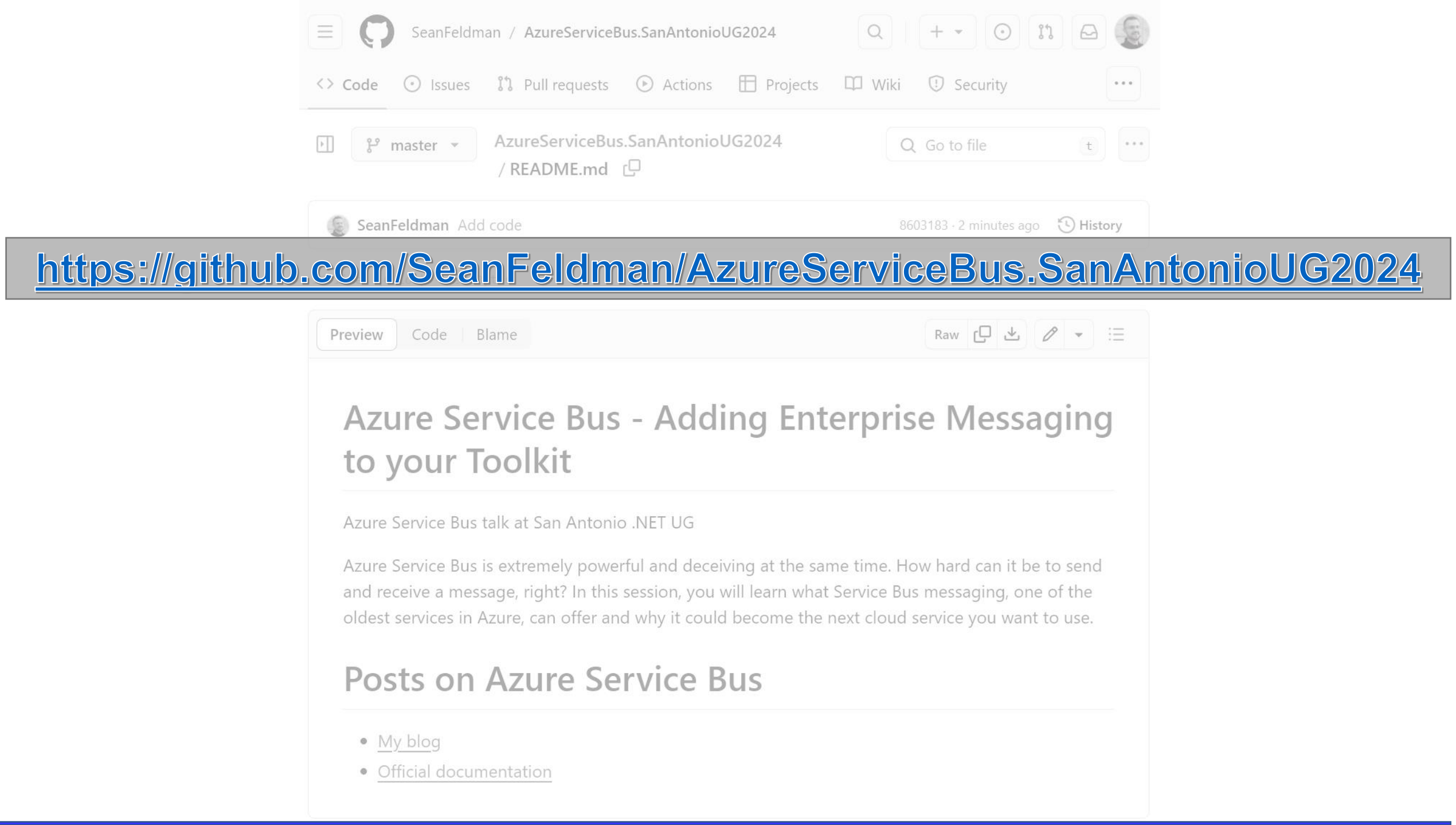
ENTERPRISE MESSAGING FEATURES

MESSAGE TRANSACTIONAL PROCESSING

RELIABILITY

GUARANTEED TYROUGHPUT AND LATENCY

TOTAL: PRICELESS


SeanFeldman / AzureServiceBus.SanAntonioUG2024
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
master
AzureServiceBus.SanAntonioUG2024
/ README.md
Go to file
SeanFeldman Add code
8603183 · 2 minutes ago
History
https://github.com/SeanFeldman/AzureServiceBus.SanAntonioUG2024
Preview
Code
Blame
Raw
Azure Service Bus - Adding Enterprise Messaging to your Toolkit
Azure Service Bus talk at San Antonio .NET UG
Azure Service Bus is extremely powerful and deceiving at the same time. How hard can it be to send and receive a message, right? In this session, you will learn what Service Bus messaging, one of the oldest services in Azure, can offer and why it could become the next cloud service you want to use.
Posts on Azure Service Bus
My blog
Official documentation


THANKS
Q&A
@sfeldman | weblogs.asp.net/sfeldman