Azure Service Bus – Message Sessions

Contents

[Overview 1](#_Toc52092747)

[Setup 2](#_Toc52092748)

[Sdks 3](#_Toc52092749)

[Send Message Session 3](#_Toc52092750)

[Receive Message Session 4](#_Toc52092751)

[Session Options 4](#_Toc52092752)

[Session Handler 5](#_Toc52092753)

[Testing 6](#_Toc52092754)

[Conclusion 6](#_Toc52092755)

# Overview

For scalable applications, the application components are decoupled to scale independently. These application components can use asynchronous messaging for interaction. The general recommendation is to keep these application components stateless, so the message ordering does not matter, and can be processed in parallel.

But there can be scenarios, where the messages need to be processed in order, they were received using FIFO (First-in, first out) pattern. Service Bus supports ordering guarantee using messaging sessions, and can be used for the FIFO pattern.

# Setup

Azure Service Bus supports messaging sessions with both queues and topic subscriptions.

The session flag must be enabled during the creation of the Queue or Topic Subscription. The “Enable Sessions” flag cannot be changed after the Queue/Subscription has been created.

|  |  |
| --- | --- |
| Topic Subscription | Queue |
|  |  |

# Sdks

There are currently 2 SDKs for Service Bus

**Microsoft.Azure.ServiceBus (Recommended)**

* NuGet - <https://www.nuget.org/packages/Microsoft.Azure.ServiceBus/>
* Latest versions are based on .NET Standard
* Older version can be used for .NET Framework
* Enhanced support for Message sessions

**WindowsAzure.ServiceBus**

* NuGet - <https://www.nuget.org/packages/WindowsAzure.ServiceBus/>
* Supports .NET Framework
* Uses the DataContractSerializer for serialzation
* Primitive support for Message sessions

Note: If the message is sent using WindowsAzure.ServiceBus Sdk, and received using Microsoft.Azure.ServiceBus Sdk, the message is not de-serialized correctly and can be null. This requires using InterOp Services to correctly read the message.

# Send Message Session

When sending message, the SessionId must be included. All the messages with the same SessionId will be processed in order, and by the same instance. The message should also have some state to indicate completion of the session.

Ex: For an Order with multiple steps/tasks, the OrderId can be the SessionId, so that all Order Tasks are processed in sequence

|  |
| --- |
| {  "OrderId": "5b255692-9f97-489e-8dd2-3d5b90e4f18b",  "TaskId": "9ba6a9c5-b39e-4048-ad35-9fab2fa3dfa0",  "JobState": "InProgess" //Possible States: In-Progress, Complete, Error  } |

|  |
| --- |
| var topicClient = new TopicClient("--SbConnectionString--", "--TopicName--");  var payload = File.ReadAllText(--PayloadFilePath--");  var orderItem = JsonConvert.DeserializeObject<OrderItem>(payload);  var message = new Message(Encoding.UTF8.GetBytes(payload));  message.SessionId = orderItem.OrderId.ToString();  message.UserProperties["SubscriberName"] = "--SubscriptionName--";  await topicClient.SendAsync(message).ConfigureAwait(false) |

# Receive Message Session

The instance receiving the message, will lock the session, so that all messages for that session will be processed by the same instance. The processor can explicitly close the session based on some status which indicates the end of session. (Ex: If JobState is “Complete” Or “Error”, close the session)

## Session Options

These options can be defined for the session handler when receiving messages

|  |  |
| --- | --- |
| MessageWaitTimeout | Defines when to expire the session. If there is no message with the same SessionId before the MessageWaitTimeout, then the session is closed automatically. Default is 60 seconds. |
| MaxConcurrentSessions | The number of concurrent sessions the instance can process. Each distinct SessionId counts as a session. Default value is 2000. |
| MaxAutoRenewDuration | The duration for which the session lock will be renewed automatically. |
| AutoComplete | Automatically completes the message |

|  |
| --- |
| var sessionHandlerOptions = new SessionHandlerOptions(e => this.LogMessageHandlerException(e))  {  MessageWaitTimeout = TimeSpan.FromSeconds("60"),  MaxAutoRenewDuration = TimeSpan.FromSeconds("300"),  MaxConcurrentSessions = "2000",  AutoComplete = false  };  var subscriptionClient = new SubscriptionClient("--SbConnectionString--", "--TopicName--", "--SubscriptionName--", ReceiveMode.PeekLock);  subscriptionClient.RegisterSessionHandler(this.MessageSessionHandlerAsync, sessionHandlerOptions); |

## Session Handler

The Session Handler is used to process the session messages. Once the message is accepted by the instance, the session is tethered to that instance, and any messages corresponding to that session will be processed by the same instance.

* session.CompleteAsync is used to complete the current message after it is processed
* session.CloseAsync is used to close the session. If the session is not closed before the configured “MessageTimeout” duration, the session is closed automatically. Any new messages arriving for that session, after it is closed, can be locked/processed by any available instance.

|  |
| --- |
| public async Task MessageSessionHandlerAsync(IMessageSession session, Message message, CancellationToken cancellationToken)  {  if (message == null)  {  await session.CompleteAsync(message.SystemProperties.LockToken);  return;  }  string messageBody = null;  if (message.Body != null)  {  messageBody = Encoding.UTF8.GetString(message.Body);  }  if (string.IsNullOrWhiteSpace(messageBody))  {  messageBody = message.GetBody<string>();  }  if (string.IsNullOrWhiteSpace(messageBody))  {  await session.CompleteAsync(message.SystemProperties.LockToken);  return;  }  OrderTask orderTask;  try  {  orderTask = JsonConvert.DeserializeObject<OrderTask>(messageBody);  }  catch (Exception ex)  {  await session.CompleteAsync(message.SystemProperties.LockToken);  return;  }  ///TODO - Process Message    await session.CompleteAsync(message.SystemProperties.LockToken);  var taskStatus = (JobState)Enum.Parse(typeof(JobState), orderTask.JobState);  if (taskStatus == JobState.COMPLETED || taskStatus == JobState.FAILED)  {  if (!session.IsClosedOrClosing)  {  await session.CloseAsync();  }  }  } |

# Testing

The sample code allows to test that the messages are being processed in order

* Sample Repository - <https://github.com/ggais/samples/tree/main/ServiceBusMessageSessions>
* SbMessageSessionSender can be used to send any number of *Numbered* session messages to a topic subscription
* SbMessageSessionReceiver can be used to receive and log session messages to a text file, and confirm they are in order

# Conclusion

Service Bus Message Sessions simplifies scenarios where order guarantee of messages is required.