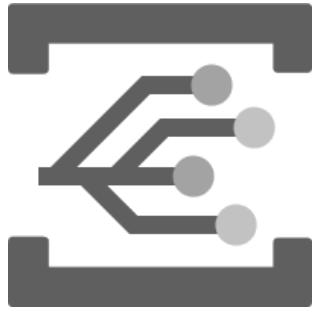




# Deep Dive with Azure Service Bus

@danielmarbach

Events



## Event Grid

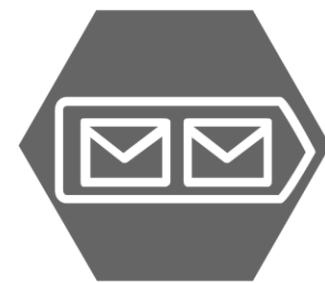
Cross cloud reactive  
eventing



## Event Hubs

Big data streaming

Messages



## Storage Queues

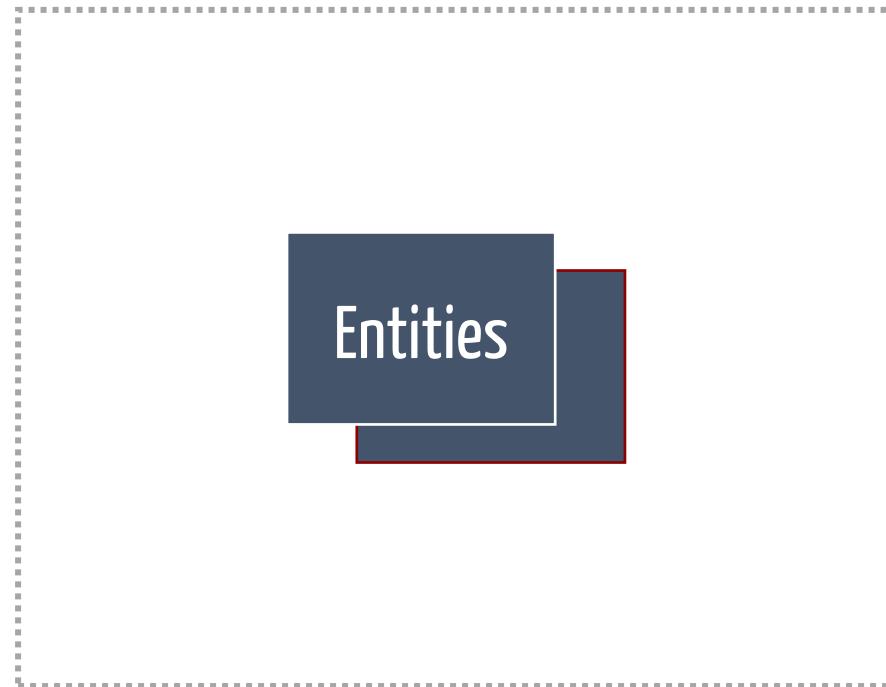
Simple task queues



## Service Bus

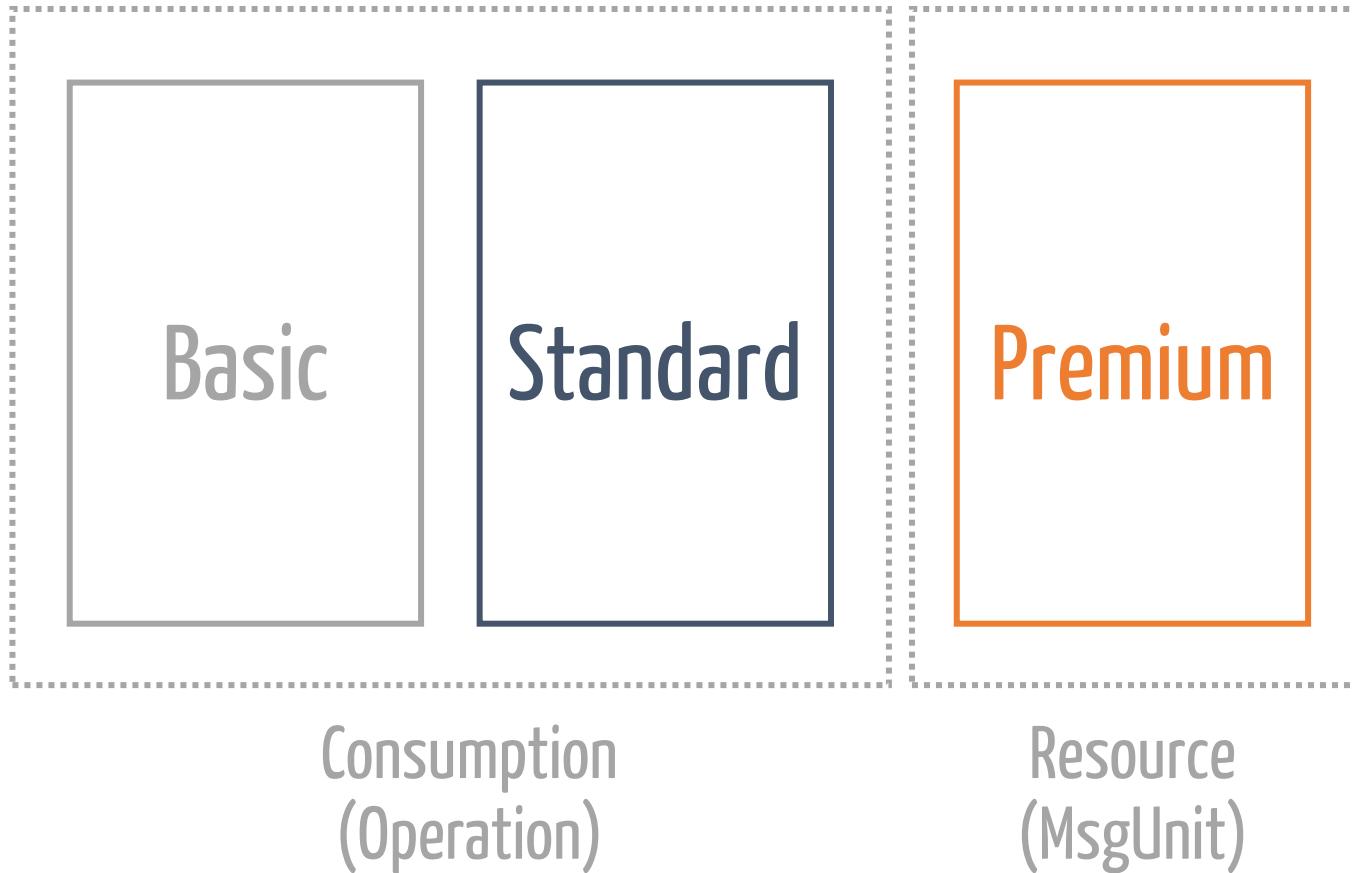
Enterprise messaging

# Namespace



Unit of Isolation

# Tiers





nug**e**t

# Create .NET apps faster with NuGet

**I** Search for packages.



3,055,833  
package versions

87,843,637,885  
package downloads

247,282  
unique packages



Search for packages...



# Azure.Messaging

Azure Service Bus is a fully managed service that allows you to decouple applications and services. Service Bus provides a reliable way to exchange data and state. This client library allows you to interact with the Azure Service Bus.

For more information about Service Bus, see the [Service Bus documentation](#).

[messaging/service-bus-messaging](#)

[Package Manager](#)[.NET CLI](#)[Pack](#)

```
> dotnet add package Azure.Messaging
```



## Release Notes

[https://github.com/Azure/azure-sdk-for-net/blob/Azure.Messaging.ServiceBus\\_7.1.0/sdk/servicebus/Azure.Messaging.ServiceBus/CHANGELOG.md](https://github.com/Azure/azure-sdk-for-net/blob/Azure.Messaging.ServiceBus_7.1.0/sdk/servicebus/Azure.Messaging.ServiceBus/CHANGELOG.md)

[Depend](#)

## Info

- [last updated a month ago](#)
- [Project Site](#)
- [Source repository](#)
- [License: MIT](#)
- [Contact owners](#)
- [Report](#)
- [Download package \(301.06 KB\)](#)
- [Open in FuGet Package Explorer](#)

## Statistics

218,969 total downloads

Send

```
await using var serviceBusClient = new ServiceBusClient(connectionString);
await using var client = serviceBusClient.CreateSender(destination);

var message = new ServiceBusMessage("Deep Dive");
<PackageReference Include="Azure.Messaging.ServiceBus" />
message.ApplicationProperties.Add("TenantId", "MyTenantId");

await client.SendMessageAsync(message);
```

```
message.  
await cl  
Console.
```

-  `GetType`
-  `MessageId`
-  `PartitionKey`
-  `ReplyTo`
-  `ReplyToSessionId`
-  `ScheduledEnqueueTime`
-  `SessionId`
-  `Subject`
-  `TimeToLive`
-  `To`
-  `ToString`
-  `TransactionPartitionKey`

```
c:\p\AzureServiceBus.DeepDive\Send  
> dotnet run  
|
```



 Service Bus Namespace

sb://nservicebustests.servicebus.windows.net/

-  Queues
-  Topics
-  Event Hubs
-  Notification Hubs
-  Relays

 View Queue: queue Log

```
<20:37:38> The application is now connected to the sb://nservicebustests.servicebus.windows.net/ service bus namespace.  
<20:37:38> MessagingFactory successfully created
```

```
c:\p\AzureServiceBus.DeepDive\Send  
> dotnet run
```

### Service Bus Namespace

sb://nservicebustests.servicebus.windows.net/

- Queues
- Topics
- Event Hubs
- Notification Hubs
- Relays

queue (1, 0, 0)

### View Queue: queue

Description Authorization Rules

Path

Relative URI: queue

Auto Delete On Idle

Days: 106751 Hours: 2 Minutes: 48 Seconds: 5 Millisecs: 477

Duplicate Detection History Time Window

Days: 0 Hours: 0 Minutes: 10 Seconds: 0 Millisecs: 0

Default Message Time To Live

Days: 106751 Hours: 2 Minutes: 48 Seconds: 5 Millisecs: 477

Queue Properties

Max Queue Size In GB: 1 GB

Max Delivery Count: 10

User Description:

Forward To:

Forward Dead Lettered Messages To:

Lock Duration

Days: 0 Hours: 0 Minutes: 1 Seconds: 0 Millisecs: 0

Queue Settings

- Enable Batched Operations
- Enable Dead Lettering On Message Expiration
- Enable Partitioning
- Enable Express
- Requires Duplicate Detection
- Requires Session
- Enforce Message Ordering

Queue Information

Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	201
Created At	14.06.2019 18:38:48
Accessed At	14.06.2019 18:40:10
Updated At	14.06.2019 18:38:48
Active Message Count	1
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	1

Purge Purge DLQ Messages Deadletter Transf DLQ Refresh Status Delete Update

### Log

```
<20:37:38> The application is now connected to the sb://nservicebustests.servicebus.windows.net/ service bus namespace.  
<20:37:38> MessagingFactory successfully created.  
<20:38:51> The queue queue has been successfully created.  
<20:38:51> The queue queue has been successfully retrieved.  
<20:40:03> The queue queue has been successfully retrieved.  
<20:40:03> [2] messages have been purged from the [queue] queue in [1602] milliseconds.  
<20:40:20> The queue queue has been successfully retrieved.
```

# Management

```
var client = new ServiceBusAdministrationClient(connectionString);
await client.CreateQueueAsync(destination);
```

# Receive

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

```
var receiverOptions = new ServiceBusReceiverOptions
{
    ReceiveMode = ServiceBusReceiveMode.PeekLock,
    PrefetchCount = 10,
    SubQueue = SubQueue.None
};

await using var receiver = serviceBusClient
    .CreateReceiver(destination, receiverOptions);

await foreach (var message in receiver.ReceiveMessagesAsync())
{
    // consume message
}
```

```
var processorOptions = new ServiceBusProcessorOptions
{
    AutoCompleteMessages = false,
    MaxConcurrentCalls = 1,
    MaxAutoLockRenewalDuration = TimeSpan.FromMinutes(10),
    ReceiveMode = ServiceBusReceiveMode.PeekLock,
    PrefetchCount = 10
};

await using var receiver = serviceBusClient.
    CreateProcessor(destination, processorOptions);
```

```
receiver.ProcessMessageAsync += async messageEventArgs =>
{
    var message = messageEventArgs.Message;

    await Out.WriteLineAsync(
        $"Received message with '{message.MessageId}'
        and content '{UTF8.GetString(message.Body)}'");

    // throw new InvalidOperationException();

    await messageEventArgs.CompleteMessageAsync(message);
};
```

```
receiver.ProcessErrorAsync += async errorEventArgs =>
{
    await Out.WriteLineAsync($"Exception:
{errorEventArgs.Exception}");
    await Out.WriteLineAsync($"FullyQualifiedNamespace:
{errorEventArgs.FullyQualifiedNamespace}");
    await Out.WriteLineAsync($"ErrorSource:
{errorEventArgs.ErrorSource}");
    await Out.WriteLineAsync($"EntityPath:
{errorEventArgs.EntityPath}");
};
```

```
await receiver.StartProcessingAsync();  
  
await untilStopRequired;  
  
await receiver.StopProcessingAsync();
```

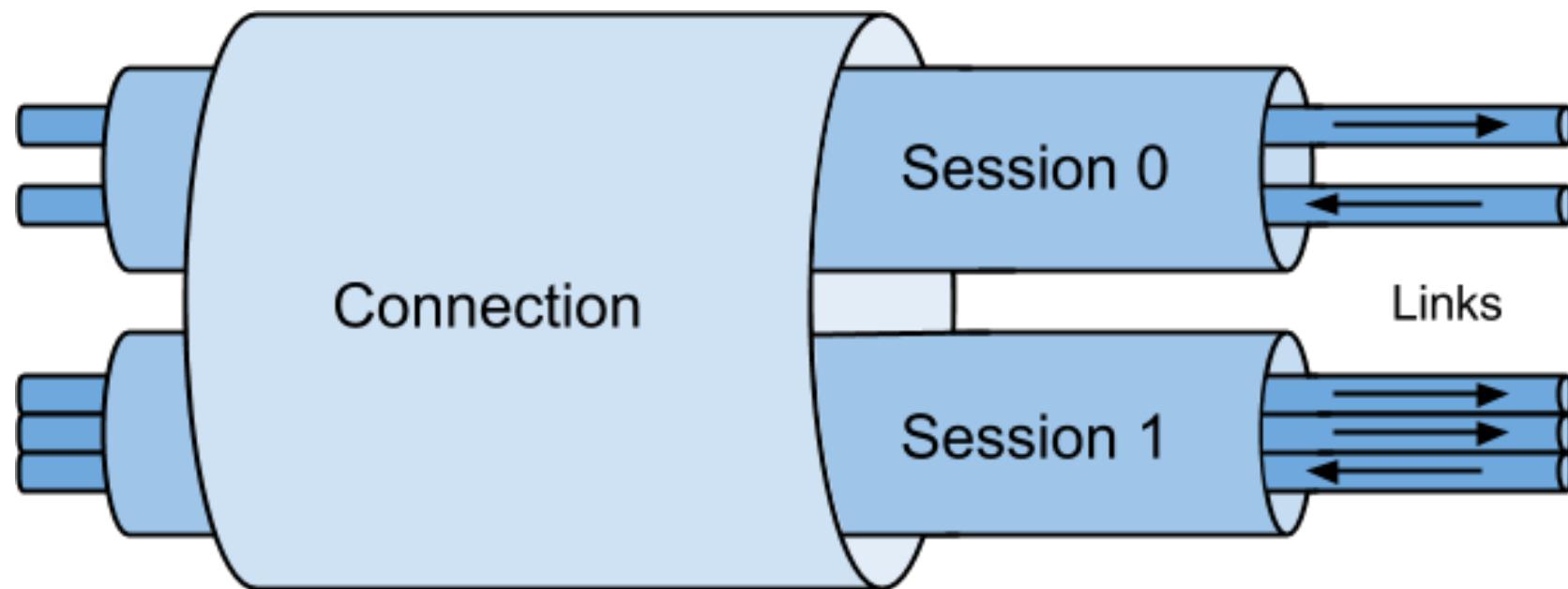
```
c:\p\AzureServiceBus.DeepDive\Receive
```

```
> dotnet run
```

```
c:\p\AzureServiceBus.DeepDive\Receive  
$ dotnet run
```



# Connections



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

await using var connectionSharingSender = serviceBusClient
    .CreateSender(destination);
await connectionSharingSender
    .SendMessageAsync(new ServiceBusMessage("Deep Dive"));

await using var connectionSharingReceiver = serviceBusClient
    .CreateReceiver(destination);
await connectionSharingReceiver.ReceiveMessageAsync();
```

```
C:\p\AzureServiceBus.DeepDive\Connections  
$ dotnet run
```

```
C:\Users\daniel.marbach  
$ netstat -an | find "5671 "|
```

```
await using var senderServiceBusClient =
    new ServiceBusClient(connectionString);
await using var receiverServiceBusClient =
    new ServiceBusClient(connectionString);

await using var senderWithDedicatedConnection =
    senderServiceBusClient.CreateSender(destination);
await using var receiverWithDedicatedConnection =
    receiverServiceBusClient.CreateReceiver(destination);

await senderWithDedicatedConnection
    .SendMessageAsync(new ServiceBusMessage("Deep Dive"));
await receiverWithDedicatedConnection.ReceiveMessageAsync();
```

```
C:\p\AzureServiceBus.DeepDive\Connections  
$ dotnet run  
netstat -na | find "5671 "  
Continue with a dedicated connection
```

Enter to stop  
[ ]

```
C:\Users\daniel.marbach  
$ netstat -an | find "5671 "  
    TCP    192.168.0.14:58459      52.166.127.37:5671      ESTABLISHED  
    TCP    192.168.0.14:58460      52.166.127.37:5671      ESTABLISHED
```

```
C:\Users\daniel.marbach  
$ [ ]
```

# Scheduling

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

var due = DateTimeOffset.UtcNow.AddSeconds(10);
await sender.ScheduleMessageAsync(
    new ServiceBusMessage($"Deep Dive + {due}"), due);
```

```
var sequenceId = await sender.ScheduleMessageAsync(  
    new ServiceBusMessage($"Deep Dive + {due}"), due);  
  
await sender.CancelScheduledMessageAsync(sequenceId);
```

```
c:\p\AzureServiceBus.DeepDive\Scheduling  
> dotnet run
```

# Expiry

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

var message = new ServiceBusMessage("Half life")
{
    TimeToLive = TimeSpan.FromSeconds(10)
};

await sender.SendMessageAsync(message);

await Prepare.SimulateActiveReceiver(serviceBusClient, destination);
```

```
c:\p\AzureServiceBus.DeepDive\Expiry  
> dotnet run
```

Where does the message go?

**Service Bus Namespace**

- sb://nservicebustests.servicebus.windows.net/
  - Queues
    - queue (0, 0)
  - Topics
  - Event Hubs
  - Notification Hubs
  - Relays

**View Queue: queue**

Description | Authorization Rules |

**Path**

Relative URI: queue

**Auto Delete On Idle**

Days:	Hours:	Minutes:	Seconds:	Millisecs:
106751	2	48	5	477

**Duplicate Detection History Time Window**

Days:	Hours:	Minutes:	Seconds:	Millisecs:
0	0	10	0	0

**Default Message Time To Live**

Days:	Hours:	Minutes:	Seconds:	Millisecs:
106751	2	48	5	477

**Queue Properties**

Max Queue Size In GB: 1 GB

Max Delivery Count: 2147483647

User Description:

Forward To:

Forward Dead Lettered Messages To:

**Lock Duration**

Days:	Hours:	Minutes:	Seconds:	Millisecs:
0	0	1	0	0

**Queue Settings**

- Enable Batched Operations
- Enable Dead Lettering On Message Expiration
- Enable Partitioning
- Enable Express
- Requires Duplicate Detection
- Requires Session
- Enforce Message Ordering

**Queue Information**

Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	0
Created At	14.06.2019 20:41:43
Accessed At	14.06.2019 20:41:59
Updated At	14.06.2019 20:41:43
Active Message Count	0
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	0

Purge Purge DLQ Messages Deadletter Transf DLQ Refresh Status Delete Update

**Log**

```
<21:02:00> The queue queue has been successfully deleted.
<22:42:14> The queue queue has been successfully retrieved.
<22:42:15> The queue queue has been successfully retrieved.
<22:42:16> The queue queue has been successfully retrieved.
<22:42:17> The queue queue has been successfully retrieved.
<22:42:17> The queue queue has been successfully retrieved.
<22:42:18> The queue queue has been successfully retrieved.
<22:42:18> The queue queue has been successfully retrieved.
<22:42:18> The queue queue has been successfully retrieved.
<22:42:19> The queue queue has been successfully retrieved.
<22:42:19> The queue queue has been successfully retrieved.
<22:42:19> The queue queue has been successfully retrieved.
```

# Deadlettering

```
var description = new CreateQueueOptions(destination)
{
    DeadLetteringOnMessageExpiration = true,
    MaxDeliveryCount = 1
};
await client.CreateQueueAsync(description);
```

```
var message = new ServiceBusMessage("Half life")
{
    TimeToLive = TimeSpan.FromSeconds(1)
};
await sender.SendMessageAsync(message);
```

```
var message = new ServiceBusMessage("Delivery Count");
await sender.SendMessageAsync(message);
```

```
var message = new ServiceBusMessage("Poor Soul");
message.ApplicationProperties.Add("Yehaa", "Why so happy?");
await sender.SendMessageAsync(message);
```

```
receiver.ProcessMessageAsync += async processMessageEventArgs =>
{
    var message = processMessageEventArgs.Message;
    switch (UTF8.GetString(message.Body)) {
        case "Half life":
            await processMessageEventArgs.AbandonMessageAsync(message);
            await Error.WriteLineAsync("Abandon half life message");
            break;

        case "Delivery Count":
            await Error.WriteLineAsync("Throwing delivery count message");
            throw new InvalidOperationException();

        case "Poor Soul":
            await Error.WriteLineAsync("Dead letter poor soul message");
            await processMessageEventArgs.DeadLetterMessageAsync(message,
                new Dictionary<string, object>
            {
                {"Reason", "Because we can!"},
                {"When", DateTimeOffset.UtcNow}
            });
            break;
    }
};
```

```
c:\p\AzureServiceBus.DeepDive\Deadlettering  
> dotnet run
```



## Service Bus Namespace

- sb://nservicebus-tests.servicebus.windows.net/
  - Queues
    - queue (0, 3, 0)
  - Topics
  - Event Hubs
  - Notification Hubs
  - Relays

## View Queue: queue

[Description](#) [Authorization Rules](#) [Messages](#)

## Message List

MessageId	Seq	Size	Label	EnqueuedTimeUtc	ExpiresAtUtc
-----------	-----	------	-------	-----------------	--------------

## Message Text

1
---

## Message System Properties

## Message Custom Properties

Name	Value

[Purge](#) [Purge DLQ](#) [Messages](#) [Deadletter](#) [Transf DLQ](#) [Refresh](#) [Status](#) [Delete](#) [Up](#)

## Log

```
<22:42:16> The queue queue has been successfully retrieved.  
<22:42:16> The queue queue has been successfully retrieved.  
<22:42:16> The queue queue has been successfully retrieved.  
<22:42:17> The queue queue has been successfully retrieved.  
<22:42:17> The queue queue has been successfully retrieved.  
<22:42:18> The queue queue has been successfully retrieved.  
<22:42:19> The queue queue has been successfully retrieved.  
<22:42:19> The queue queue has been successfully retrieved.  
<22:42:19> The queue queue has been successfully retrieved.  
<22:44:42> The queue queue has been successfully retrieved.  
<22:44:50> [0] messages peeked from the queue [queue].  
<23:11:32> The queue queue has been successfully retrieved.  
<23:12:18> The queue queue has been successfully retrieved.
```

**Service Bus Namespace**

sb://nservicebus.tests.servicebus.windows.net/

- Queues
  - queue (0, 3, 0)
- Topics
- Event Hubs
- Notification Hubs
- Relays

**View Queue: queue**[Description](#) | [Authorization Rules](#) | [Messages](#) | [Deadletter](#) |**Message List**

MessageId	Seq	Size	Label	EnqueuedTimeUtc	ExpiresAtUtc
e11399aea7445249e520e...	1	227		14.06.2019 21:12	14.06.2019 21:12
dfc3c370cba04145b4d89e...	2	270		14.06.2019 21:12	31.12.9999 23:59
cc10d022ccb4944b359e1...	3	207		14.06.2019 21:12	31.12.9999 23:59

**Message Text**

1 Half life

**Message System Properties****Message Custom Properties**

Name	Value
DeadLetterRea...	TTLExpiredException
DeadLetterError...	The message expired and was dead lettered.

[Purge](#) [Purge DLQ](#) [Messages](#) [Deadletter](#) [Transf DLQ](#) [Refresh](#) [Status](#) [Delete](#) [Up](#)**Log**

```
<22:42:16> The queue queue has been successfully retrieved.  
<22:42:17> The queue queue has been successfully retrieved.  
<22:42:17> The queue queue has been successfully retrieved.  
<22:42:18> The queue queue has been successfully retrieved.  
<22:42:19> The queue queue has been successfully retrieved.  
<22:42:19> The queue queue has been successfully retrieved.  
<22:42:19> The queue queue has been successfully retrieved.  
<22:44:42> The queue queue has been successfully retrieved.  
<22:44:50> [0] messages peeked from the queue [queue].  
<23:11:32> The queue queue has been successfully retrieved.  
<23:12:18> The queue queue has been successfully retrieved.  
<23:12:20> The queue queue has been successfully retrieved.  
<23:12:26> [3] messages peeked from the deadletter queue of the queue [queue].
```

**Service Bus Namespace**

- sb://nservicebustests.servicebus.windows.net/
  - Queues
    - queue (0, 3, 0)
  - Topics
  - Event Hubs
  - Notification Hubs
  - Relays

**View Queue: queue**

Description Authorization Rules Messages Deadletter |

Message List

MessageId	Seq	Size	Label	EnqueuedTimeUtc	ExpiresAtUtc
e11399faea7445249e520e...	1	227		14.06.2019 21:12	14.06.2019 21:12
dfc3c370cba04145b4d89e...	2	270		14.06.2019 21:12	31.12.9999 23:59
cc10d022cceba4944b359e1...	3	207		14.06.2019 21:12	31.12.9999 23:59

Message System Properties

Message Custom Properties

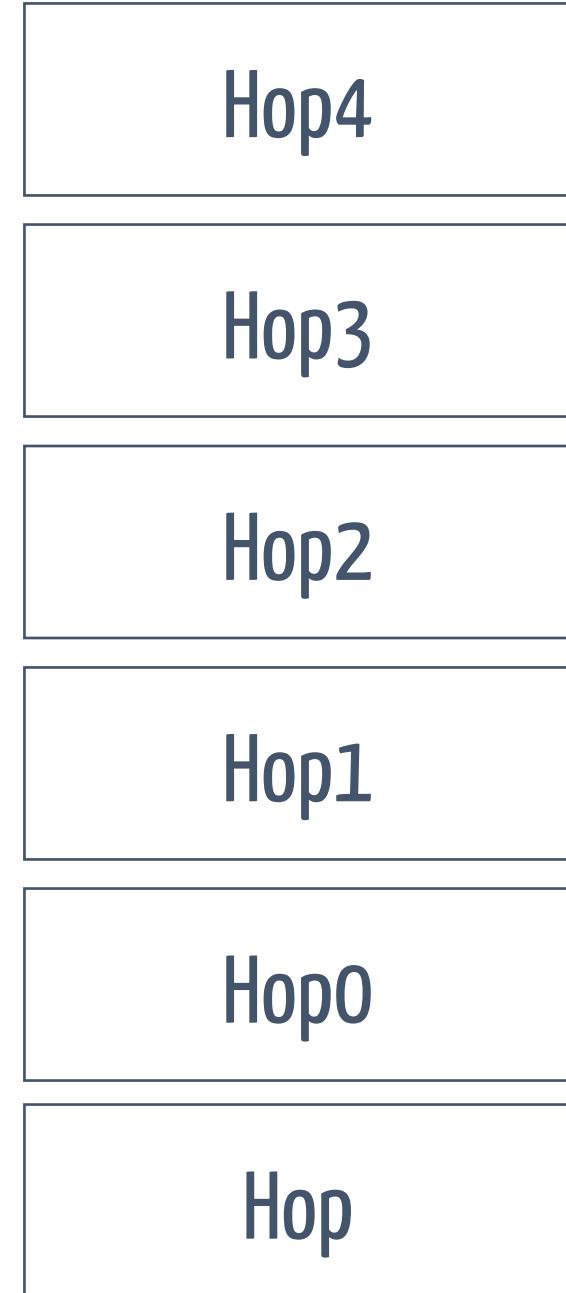
Name	Value
DeadLetterRe...	MaxDeliveryCountExceeded
DeadLetterError...	Message could not be consumed after 1 delivery atte...

Purge Purge DLQ Messages Deadletter Transf DLQ Refresh Status Delete Up

**Log**

```
<22:42:16> The queue queue has been successfully retrieved.
<22:42:17> The queue queue has been successfully retrieved.
<22:42:17> The queue queue has been successfully retrieved.
<22:42:18> The queue queue has been successfully retrieved.
<22:42:19> The queue queue has been successfully retrieved.
<22:42:19> The queue queue has been successfully retrieved.
<22:42:19> The queue queue has been successfully retrieved.
<22:44:42> The queue queue has been successfully retrieved.
<22:44:50> [0] messages peeked from the queue [queue].
<23:11:32> The queue queue has been successfully retrieved.
<23:12:18> The queue queue has been successfully retrieved.
<23:12:20> The queue queue has been successfully retrieved.
<23:12:26> [3] messages peeked from the deadletter queue of the queue [queue].
```

# Forwarding



ForwardTo



```
var description = new CreateQueueOptions("Hop");
await client.CreateQueueAsync(description);

description = new CreateQueueOptions("Hop0");
await client.CreateQueueAsync(description);

description = new CreateQueueOptions("Hop1")
{
    ForwardTo = "Hop0"
};
await client.CreateQueueAsync(description);

description = new CreateQueueOptions("Hop2")
{
    ForwardTo = "Hop1"
};
await client.CreateQueueAsync(description);

description = new CreateQueueOptions("Hop3")
{
    ForwardTo = "Hop2"
};
await client.CreateQueueAsync(description);

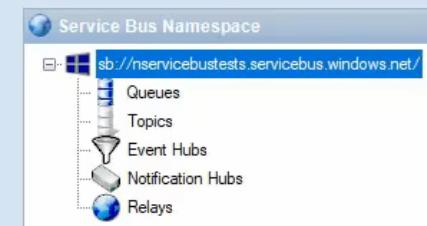
description = new CreateQueueOptions("Hop4")
{
    ForwardTo = "Hop3"
};
await client.CreateQueueAsync(description);
```

```
await using var serviceBusClient =
    new ServiceBusClient(connectionString);
var sender = serviceBusClient.CreateSender("Hop4");

var message = new ServiceBusMessage("Weeeeeeeehhh!");
await sender.SendMessageAsync(message);
```

```
c:\p\AzureServiceBus.DeepDive\Forwarding
```

```
> dotnet run
```



Entity

Service Bus Namespace

- sb://nservicebus-tests.servicebus.windows.net/
  - Queues
  - Topics
  - Event Hubs
  - Notification Hubs
  - Relays



Log

Service Bus Namespace  
sb://nservicebustests.servicebus.windows.net/

- Queues
  - hop (0, 0, 0)
  - hop0 (0, 0, 0)
  - hop1 (0, 0, 0)
  - hop2 (0, 0, 0)
  - hop3 (0, 0, 0)
  - hop4 (0, 0, 0)
- Topics
- Event Hubs
- Notification Hubs
- Relays

### View Queue: hop0

Description Authorization Rules

**Path**

Relative URI: hop0

**Duplicate Detection History Time Window**

Days: 0 Hours: 0 Minutes: 10 Seconds: 0 Millisecs: 0

**Queue Properties**

Max Queue Size In GB:  1 GB

Max Delivery Count: 10

User Description:

Forward To: |

Forward Dead Lettered Messages To:

**Auto Delete On Idle**

Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
--------------	----------	-------------	------------	----------------

**Default Message Time To Live**

Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
--------------	----------	-------------	------------	----------------

**Lock Duration**

Days: 0	Hours: 0	Minutes: 1	Seconds: 0	Millisecs: 0
---------	----------	------------	------------	--------------

**Queue Settings**

- Enable Batched Operations
- Enable Dead Lettering On Message Expiration
- Enable Partitioning
- Enable Express
- Requires Duplicate Detection
- Requires Session
- Enforce Message Ordering

**Queue Information**

Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	0
Created At	14.06.2019 21:33:14
Accessed At	14.06.2019 21:33:20
Updated At	14.06.2019 21:33:14
Active Message Count	0
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	0

Purge Purge DLQ Messages Deadletter Transf DLQ Refresh Status Del

## Log

```
<23:33:35> The queue hop has been successfully retrieved.
<23:33:35> The queue hop0 has been successfully retrieved.
<23:33:35> The queue hop1 has been successfully retrieved.
<23:33:35> The queue hop2 has been successfully retrieved.
<23:33:35> The queue hop3 has been successfully retrieved.
<23:33:35> The queue hop4 has been successfully retrieved.
```

```
c:\p\AzureServiceBus.DeepDive\Forwarding  
> dotnet run  
Sent message  
Got 'Weeeeeeehhh!' on hop 'Hop0'  
Setup forwarding from Hop0 to Hop
```

**Service Bus Namespace**

sb://nservicebustests.servicebus.windows.net/

- Queues
  - hop (0, 0, 0)
  - hop0 (0, 0, 0)
  - hop1 (0, 0, 0)
  - hop2 (0, 0, 0)
  - hop3 (0, 0, 0)
  - hop4 (0, 0, 0)
- Topics
- Event Hubs
- Notification Hubs
- Relays

**View Queue: hop0**

Description | Authorization Rules

**Path**

Relative URI:  
hop0

**Duplicate Detection History Time Window**

Days: 0 Hours: 0 Minutes: 10 Seconds: 0 Millisecs: 0

**Queue Properties**

Max Queue Size In GB: 1 GB

Max Delivery Count: 10

User Description:

Forward To: hop

Forward Dead Lettered Messages To:

**Auto Delete On Idle**

Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
--------------	----------	-------------	------------	----------------

**Default Message Time To Live**

Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
--------------	----------	-------------	------------	----------------

**Lock Duration**

Days: 0	Hours: 0	Minutes: 1	Seconds: 0	Millisecs: 0
---------	----------	------------	------------	--------------

**Queue Settings**

- Enable Batched Operations
- Enable Dead Lettering On Message Expiration
- Enable Partitioning
- Enable Express
- Requires Duplicate Detection
- Requires Session
- Enforce Message Ordering

**Queue Information**

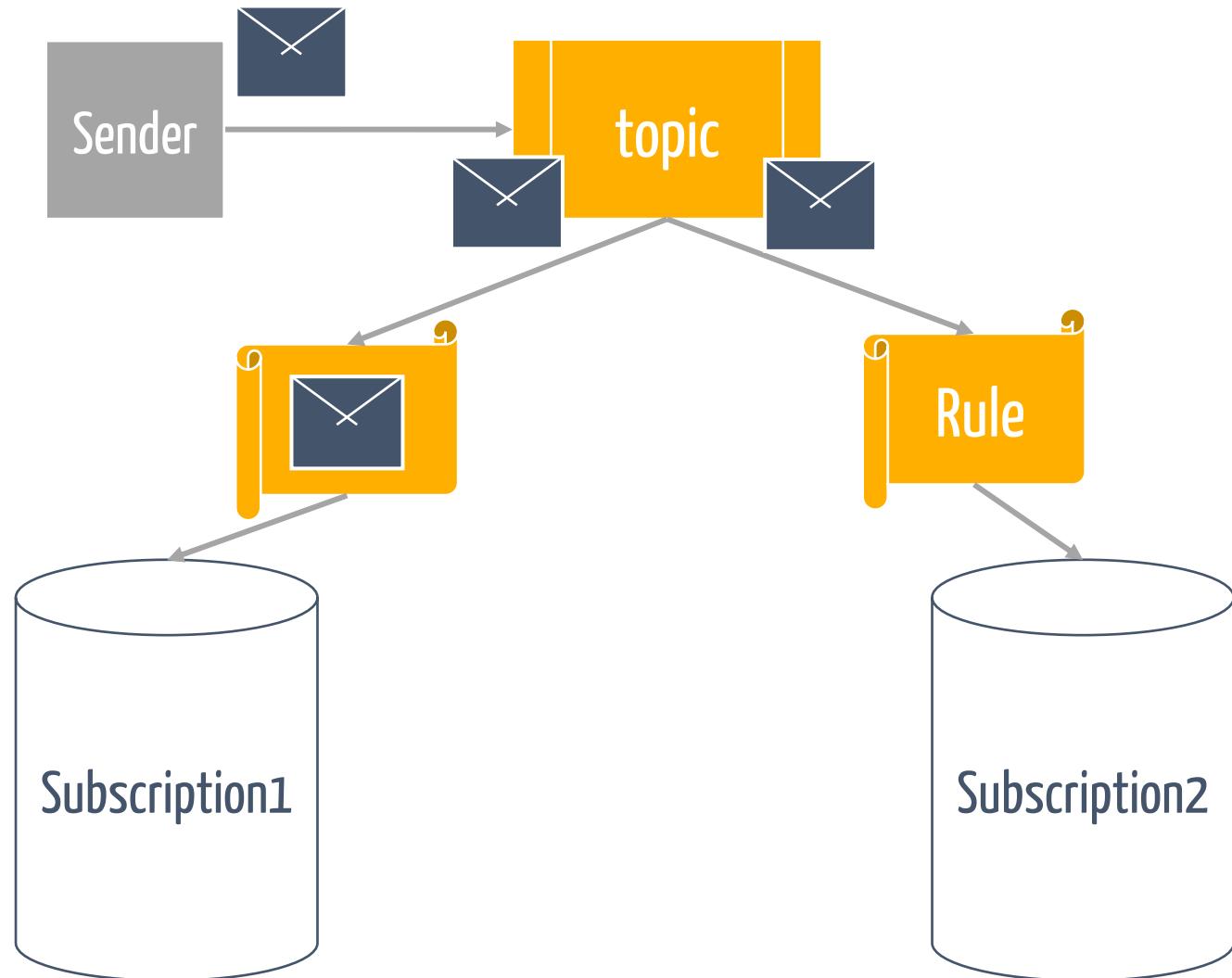
Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	0
Created At	14.06.2019 21:33:14
Accessed At	14.06.2019 21:33:20
Updated At	14.06.2019 21:33:14
Active Message Count	0
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	0

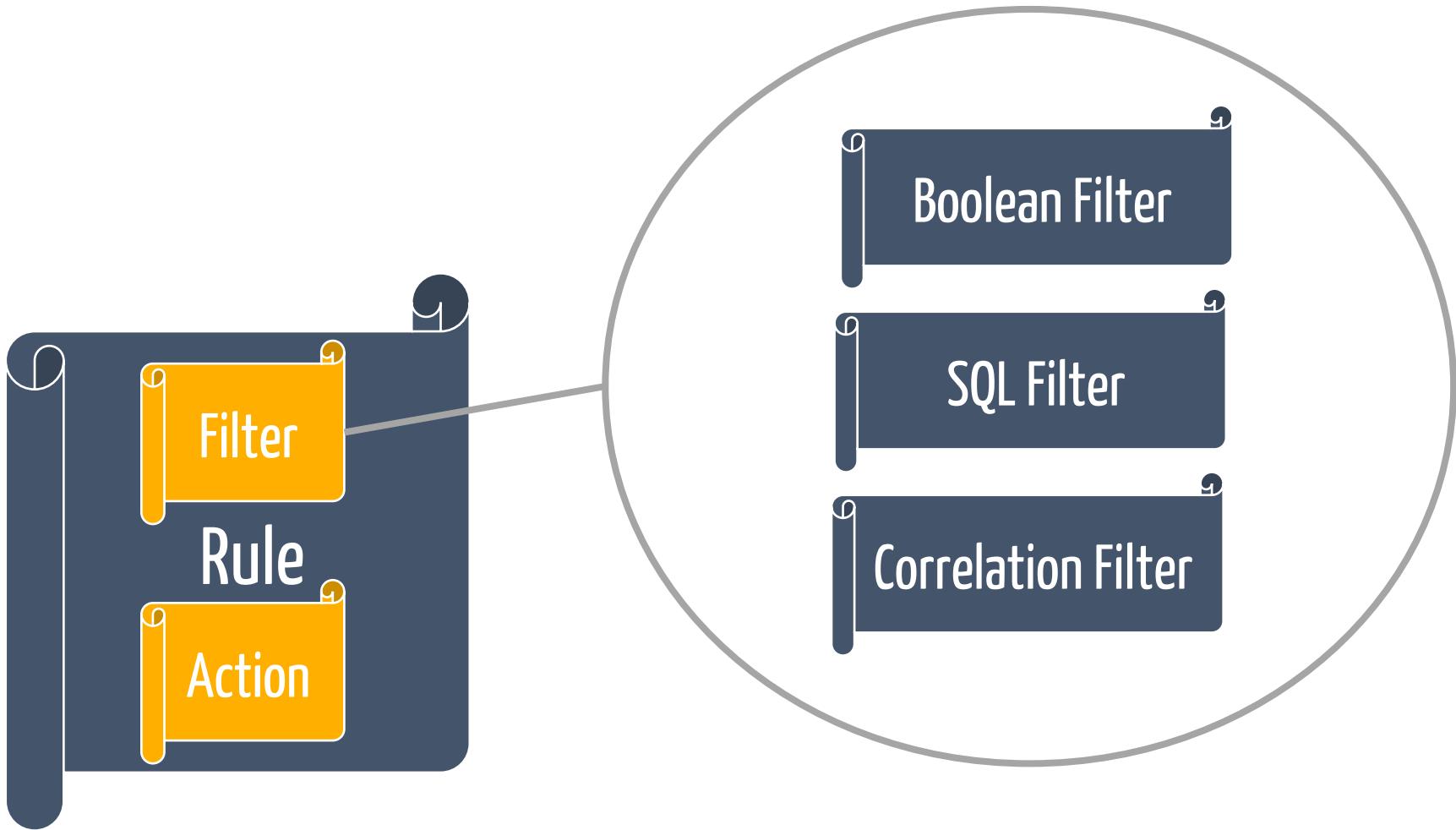
Purge | Purge DLQ | Messages | Deadletter | Transf DLQ | Refresh | Status | Delete

**Log**

```
<23:33:35> The queue hop has been successfully retrieved.
<23:33:35> The queue hop0 has been successfully retrieved.
<23:33:35> The queue hop1 has been successfully retrieved.
<23:33:35> The queue hop2 has been successfully retrieved.
<23:33:35> The queue hop3 has been successfully retrieved.
<23:33:35> The queue hop4 has been successfully retrieved.
<23:33:45> The queue hop0 has been successfully updated.
```

# Pub/Sub





```
var topicDescription = new CreateTopicOptions (topicName);  
await client.CreateTopicAsync(topicDescription);
```

```
var subscriptionDescription =  
    new CreateSubscriptionOptions(topicName, rushSubscription);  
await client.CreateSubscriptionAsync(subscriptionDescription);  
  
subscriptionDescription =  
    new CreateSubscriptionOptions(topicName, currencySubscription);  
await client.CreateSubscriptionAsync(subscriptionDescription);
```

```
var ruleDescription = new CreateRuleOptions
{
    Name = "MessagesWithRushlabel",
    Filter = new CorrelationRuleFilter
    {
        Subject = "rush"
    },
    Action = null
};
await client.
    CreateRuleAsync(topicName, rushSubscription, ruleDescription);
```

```
ruleDescription = new CreateRuleOptions
{
    Name = "MessagesWithCurrencyCHF",
    Filter = new SqlRuleFilter("currency = 'CHF'"),
    Action = new SqlRuleAction("SET currency = 'Złoty'")
};
await client.CreateRuleAsync(topicName,
    currencySubscription, ruleDescription);
```

```
var message = new ServiceBusMessage("Damn I have no time!")
{
    Subject = "rush"
};
await sender.SendMessageAsync(message);

message = new ServiceBusMessage("I'm rich! I have 1000");
message.ApplicationProperties.Add("currency", "CHF");
await sender.SendMessageAsync(message);
```

### Service Bus Namespace

sb://nservicebustests.servicebus.windows.net/

- Queues
- Topics
  - topic
    - + Subscriptions
    - alwaysInRush (1, 0, 0)
    - maybeRich (1, 0, 0)
- Event Hubs
- Notification Hubs
- Relays

### View Topic: topic

Description Authorization Rules

**Path**

Relative URI: topic

**Auto Delete On Idle**

Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
--------------	----------	-------------	------------	----------------

**Default Message Time To Live**

Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
--------------	----------	-------------	------------	----------------

**Topic Properties**

Max Queue Size In GB: 1 GB

User Description:

**Duplicate Detection History Time Window**

Days: 0	Hours: 0	Minutes: 10	Seconds: 0	Millisecs: 0
---------	----------	-------------	------------	--------------

**Topic Settings**

- Enable Batched Operations
- Enable Filtering Messages Before Publishing
- Enable Partitioning
- Enable Express
- Requires Duplicate Detection
- Enforce Message Ordering

**Topic Information**

Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	504
Created At	17.06.2019 10:35:53
Accessed At	17.06.2019 10:35:55
Updated At	17.06.2019 10:35:53
Active Message Count	0
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0

Refresh Disable Delete

### Log

```
<12:36:41> The file C:\p\AzureServiceBus.DeepDive\explorer\ServiceBusExplorer.exe.Config is used for the configuration settings.
<12:36:45> The application is now connected to the sb://nservicebustests.servicebus.windows.net/ service bus namespace.
<12:36:45> MessagingFactory successfully created
<12:36:48> The topic topic has been successfully retrieved.
<12:36:48> The subscription alwaysInRush for the topic topic has been successfully retrieved.
<12:36:48> The subscription maybeRich for the topic topic has been successfully retrieved.
```

### Service Bus Namespace

- sb://nservicebustests.servicebus.windows.net/
  - Queues
  - Topics
    - topic
    - Subscriptions
      - alwaysInRush (1, 0, 0)
      - maybeRich (1, 0, 0)**
  - Event Hubs
  - Notification Hubs
  - Relays

### View Subscription: maybeRich

**Description**

<b>Name</b>	<b>Auto Delete On Idle</b>				
Subscription Name:	Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477
<b>Lock Duration</b>	Days: 0	Hours: 0	Minutes: 1	Seconds: 0	Millisecs: 0
<b>Default Message Time To Live</b>	Days: 106751	Hours: 2	Minutes: 48	Seconds: 5	Millisecs: 477

**Subscription Properties**

Max Delivery Count:	10
User Description:	
Forward To:	
Forward Dead Lettered Messages To:	

**Subscription Settings**

<input checked="" type="checkbox"/> Enable Batched Operations
<input checked="" type="checkbox"/> Enable Dead Lettering On Filter Evaluation Error
<input type="checkbox"/> Enable Dead Lettering On Message Expiration
<input type="checkbox"/> Requires Session

**Subscription Information**

Name	Value
Status	Active
Is ReadOnly	False
Created At	17.06.2019 10:35:53
Accessed At	17.06.2019 10:35:53
Updated At	17.06.2019 10:35:53
Active Message Count	1
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	1

**Log**

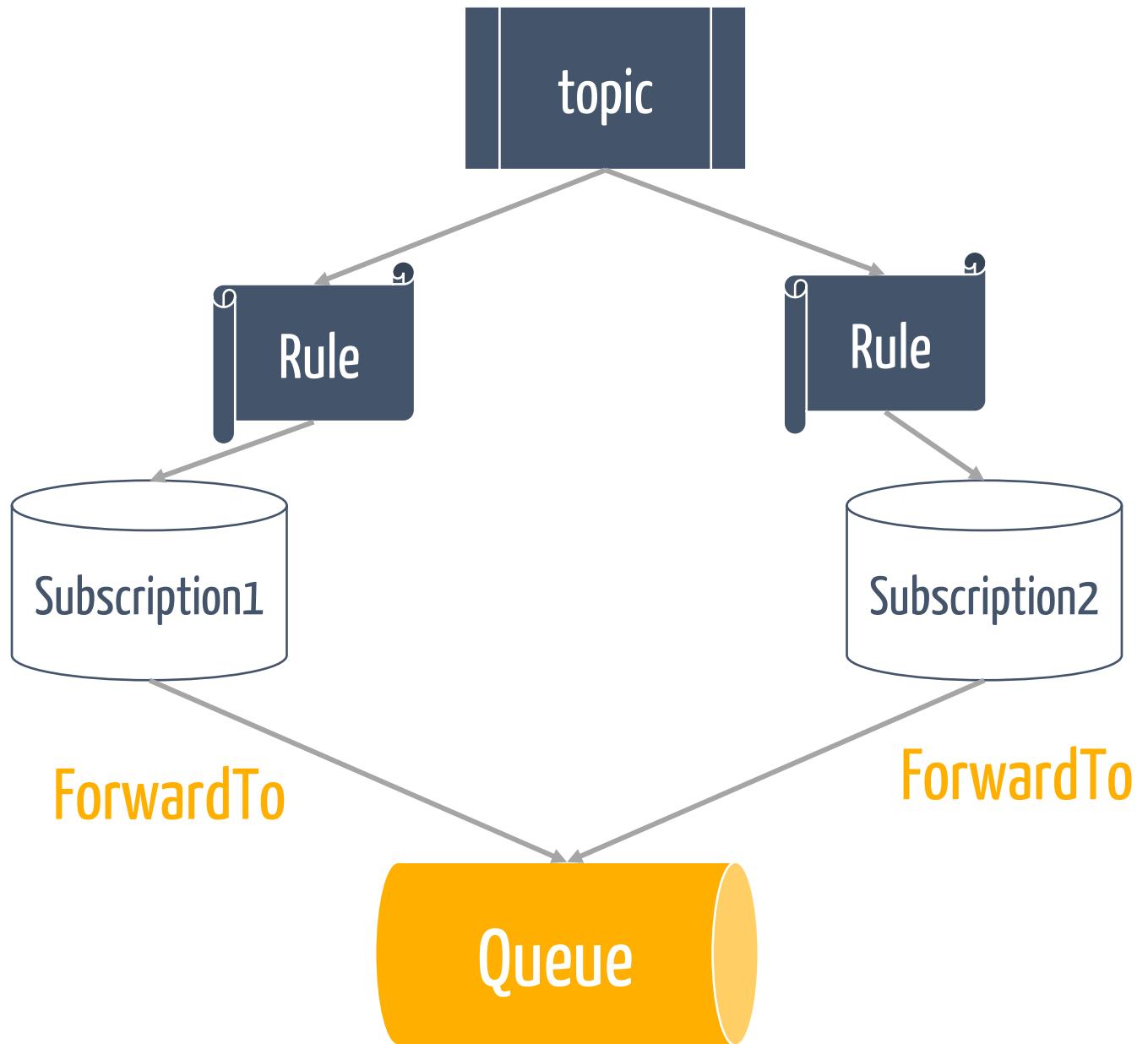
```
<12:36:41> The file C:\p\AzureServiceBus.Deep Dive\explorer\ServiceBusExplorer.exe.Config is used for the configuration settings.
<12:36:45> The application is now connected to the sb://nservicebustests.servicebus.windows.net/ service bus namespace.
<12:36:45> MessagingFactory successfully created
<12:36:48> The topic topic has been successfully retrieved.
<12:36:48> The subscription alwaysInRush for the topic topic has been successfully retrieved.
<12:36:48> The subscription maybeRich for the topic topic has been successfully retrieved.
<12:37:46> The rule MessagesWithRushLabel for the alwaysInRush subscription of the topic topic has been successfully retrieved.
<12:37:54> [1] messages peeked from the subscription [alwaysInRush].
<12:38:01> The rule MessagesWithCurrencyCHF for the maybeRich subscription of the topic topic has been successfully retrieved.
```

Purge Purge DLQ Messages Deadletter Refresh Disable Delete

Favor Correlation filter over SQL filter

Subscriptions are virtual queues and  
subscribers need to receive from them

# Topologies



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

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

```
c:\p\AzureServiceBus.DeepDive\Topologies  
> dotnet run
```



**Service Bus Namespace**

- sb://nservicebustests.servicebus.windows.net/
  - Queues
    - queue (2, 0, 0)
  - Topics
    - topic
  - Event Hubs
  - Notification Hubs
  - Relays

**View Subscription: maybeRich**

**Message List**

MessageId	Seq	Size	Label	EnqueuedTimeUtc	ExpiresAtUtc
e0d9141d27684e3aac0eb6...	1	163		17.06.2019 10:35	31.12.9999 23:59

**Message Text**

```
I'm rich! I have 1000
```

**Message System Properties**

Misc	
Content-Type	
CorrelationId	
DeadLetterSource	
DeliveryCount	1
EnqueuedSequenceNumber	2
EnqueuedTimeUtc	17.06.2019 10:35
ExpiresAtUtc	31.12.9999 23:59
ForcePersistence	False
IsBodyConsumed	False
Label	
LockedUntilUtc	Operation is not valid due to the current lock state.
LockToken	Operation is not valid due to the current lock state.
MessageId	e0d9141d27684e3aac0eb6ea0a

**Message Custom Properties**

Name	Value
currency	Złoty
RuleName	MessagesWithCurrencyCHF

Purge Purge DLQ Messages Deadletter Refresh Disable Delete

**Log**

```
<12:36:41> The file C:\p\AzureServiceBus.Deep Dive\explorer\ServiceBusExplorer.exe.Config is used for the configuration settings.
<12:36:45> The application is now connected to the sb://nservicebustests.servicebus.windows.net/ service bus namespace.
<12:36:45> MessagingFactory successfully created
<12:36:48> The topic topic has been successfully retrieved.
<12:36:48> The subscription alwaysInRush for the topic topic has been successfully retrieved.
<12:36:48> The subscription maybeRich for the topic topic has been successfully retrieved.
<12:37:46> The rule MessagesWithRushLabel for the alwaysInRush subscription of the topic topic has been successfully retrieved.
<12:37:54> [1] messages peeked from the subscription [alwaysInRush].
<12:38:01> The rule MessagesWithCurrencyCHF for the maybeRich subscription of the topic topic has been successfully retrieved.
<12:38:04> [1] messages peeked from the subscription [maybeRich].
<12:53:06> The queue queue has been successfully retrieved.
<12:53:06> The topic topic has been successfully retrieved.
<12:53:07> The subscription alwaysInRush for the topic topic has been successfully retrieved.
<12:53:07> The subscription maybeRich for the topic topic has been successfully retrieved.
```

### Service Bus Namespace

- sb://nservicebustests.servicebus.windows.net/
  - Queues
    - queue (2, 0, 0)
  - Topics
    - topic
      - Subscriptions
        - alwaysInRush (0, 0, 0)
        - maybeRich (0, 0, 0)
  - Event Hubs
  - Notification Hubs
  - Relays

### View Queue: queue

Description | Authorization Rules | Messages |

Message List:

MessageId	Seq	Size	Label	EnqueuedTimeUtc	ExpiresAtUtc
6b1fe11b26be418ab3b349f...	1	228	rush	17.06.2019 10:52	31.12.9999 23:59
ccbbea7401af434890a2f38...	2	273		17.06.2019 10:52	31.12.9999 23:59

Message System Properties

Misc

ContentType	
CorrelationId	
DeadLetterSource	
DeliveryCount	1
EnqueuedSequenceNumber	1
EnqueuedTimeUtc	17.06.2019 10:52
ExpiresAtUtc	31.12.9999 23:59
ForcePersistence	<b>False</b>
IsBodyConsumed	<b>False</b>
Label	<b>rush</b>
LockedUntilUtc	
LockToken	
MessageId	<b>6b1fe11b26be418ab3b349f...</b>

Message Text

```
1 Damn I have not time!
```

Message Custom Properties

Name	Value
------	-------

Purge | Purge DLQ | Messages | Deadletter | Transf DLQ | Refresh | Status | Delete | Up

## Log

```
<12:36:45> The application is now connected to the sb://nservicebustests.servicebus.windows.net/ service bus namespace.
<12:36:45> MessagingFactory successfully created
<12:36:48> The topic topic has been successfully retrieved.
<12:36:48> The subscription alwaysInRush for the topic topic has been successfully retrieved.
<12:36:48> The subscription maybeRich for the topic topic has been successfully retrieved.
<12:37:46> The rule MessagesWithRushlabel for the alwaysInRush subscription of the topic topic has been successfully retrieved.
<12:37:54> [1] messages peeked from the subscription [alwaysInRush].
<12:38:01> The rule MessagesWithCurrencyCHF for the maybeRich subscription of the topic topic has been successfully retrieved.
<12:38:04> [1] messages peeked from the subscription [maybeRich].
<12:53:06> The queue queue has been successfully retrieved.
<12:53:06> The topic topic has been successfully retrieved.
<12:53:07> The subscription alwaysInRush for the topic topic has been successfully retrieved.
<12:53:07> The subscription maybeRich for the topic topic has been successfully retrieved.
<12:53:13> The rule MessagesWithRushlabel for the alwaysInRush subscription of the topic topic has been successfully retrieved.
<12:53:15> The rule MessagesWithCurrencyCHF for the maybeRich subscription of the topic topic has been successfully retrieved.
<12:53:24> [2] messages peeked from the queue [queue].
```

# AtomicSends

```
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();
}
```

```
c:\p\AzureServiceBus.DeepDive\AtomicSend
```

```
> dotnet run
```

Transaction not  
completed

Transaction completed

# Batching

```
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);
```

```
c:\p\AzureServiceBus.DeepDive\Batching  
> dotnet run
```

```
var messagesToSend = new Queue<ServiceBusMessage>();
for (var i = 0; i < 4500; i++)
{
    var message = new ServiceBusMessage($"Deep Dive{i}. Deep Dive{i}. Deep Dive{i}.");
    messagesToSend.Enqueue(message);
}
```

```
var messageCount = messagesToSend.Count;
int batchCount = 1;
while (messagesToSend.Count > 0)
{
    using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

    if (messageBatch.TryAddMessage(messagesToSend.Peek()))
    {
        messagesToSend.Dequeue();
    }
    else
    {
        throw new Exception($"Message {messageCount - messagesToSend.Count} is too large and cannot be sent.");
    }

    while (messagesToSend.Count > 0 && messageBatch.TryAddMessage(messagesToSend.Peek()))
    {
        messagesToSend.Dequeue();
    }

    await sender.SendMessagesAsync(messageBatch);
}
```

Upgrade to premium tier

Use `ServiceBus.AttachmentPlugin`

SendVia

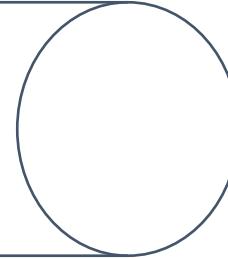
Incoming



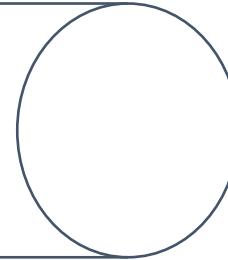
Transfer



Outgoing1



Outgoing2



```
await using var transactionalClient =
    new ServiceBusClient(connectionString,
    new ServiceBusClientOptions
    {
        EnableCrossEntityTransactions = true,
    });

receiver = transactionalClient.CreateProcessor(inputQueue);
sender = transactionalClient.CreateSender(destinationQueue);
```

```
receiver.ProcessMessageAsync += async processMessageEventArgs =>
{
    var message = processMessageEventArgs.Message;
    using (var scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
    {
        await sender.SendMessageAsync(new ServiceBusMessage("Will not leak"));

        if (!message.ApplicationProperties.ContainsKey("Win"))
            throw new InvalidOperationException();

        await sender.SendMessageAsync(new ServiceBusMessage("Will not leak"));

        scope.Complete();
    }
};
```

```
c:\p\AzureServiceBus.DeepDive\SendVia  
> dotnet run
```



```
c:\p\AzureServiceBus.DeepDive\SendVia  
> dotnet run  
#'0' messages in 'destination'  
Received message with 'bdc126e8424549acb9d1c23a5af31799' and content 'Kick off'  
#'1' messages in 'destination'  
Received message with 'bdc126e8424549acb9d1c23a5af31799' and content 'Kick off'  
#'2' messages in 'destination'
```

TransferDLQ

```
receiver.ProcessMessageAsync += async processMessageEventArgs =>
{
    var message = processMessageEventArgs.Message;
    using (var scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
    {
        await sender.SendMessageAsync(new ServiceBusMessage("Will not leak"));

        var client = new ServiceBusAdministrationClient(connectionString);

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

        await client.UpdateQueueAsync(queueProperties);

        scope.Complete();
    }
};
```

```
var client = new ServiceBusAdministrationClient(connectionString);

QueueRuntimeProperties info = await client
    .GetQueueRuntimeInfoAsync(destination);

long activeMessageCount = info.ActiveMessageCount;
long deadLetterMessageCount = info.DeadLetterMessageCount;
long transferDeadLetterMessageCount = info.TransferDeadLetterMessageC
ount;

string destinationDeadLetterPath = EntityNameHelper
    .FormatDeadLetterPath(destination);
string destinationTransferDeadLetterPath = EntityNameHelper
    .FormatTransferDeadLetterPath(destination);
```

```
c:\p\AzureServiceBus.DeepDive\TransferDLQ
```

```
> dotnet run
```

# Dedup

```
var queueDescription = new CreateQueueOptions(destination)
{
    RequiresDuplicateDetection = true,
    DuplicateDetectionHistoryTimeWindow = TimeSpan.FromSeconds(20)
};
await client.CreateQueueAsync(queueDescription);
```

```
var content = Encoding.UTF8.GetBytes("Message1Message1");
var messageId = new Guid(content).ToString();

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

await sender.SendMessageAsync(messages);
```

```
c:\p\AzureServiceBus.DeepDive\Dup
```

```
> dotnet run
```



# SUMMARY

ENTERPRISE MESSAGING FEATURES

MESSAGE TRANSACTIONAL PROCESSING

RELIABILITY

GUARANTEED THROUGHPUT AND LATENCY

---

TOTAL: PRICELESS



## Azure Service Bus Transport



Source | NServiceBus.Transport.AzureServiceBus (1.x)

Target NServiceBus Version: 7.x

The Azure Service Bus transport leverages the .NET Standard Microsoft.Azure.ServiceBus<sup>®</sup> client SDK.

Azure Service Bus<sup>®</sup> is a messaging service hosted on the Azure platform that allows for exchanging messages between various applications in a loosely coupled fashion. The service offers guaranteed message delivery and supports a range of standard protocols (e.g. REST, AMQP, WS\*) and APIs such as native pub/sub, delayed delivery, and more.

Configuring

# go.particular.net/Sample

To use Azure Service Bus as the underlying transport:

```
var transport = endpointConfiguration.UseTransport<AzureServiceBusTransport>();  
transport.ConnectionString("Endpoint=sb://[NAMESPACE].servicebus.windows.net/;SharedAccessKeyName=[KEYNAME];Shared  
AccessKey=[KEY]");
```

Copy

Edit

The Azure Service Bus transport requires a connection string to connect to a namespace.

## Samples

- Azure Service Bus Native Integration  
Consuming messages published by non-NServiceBus endpoints.
- Azure Service Bus Send/Reply Sample  
Demonstrates the send/reply pattern with Azure Service Bus.

Getting Started

NServiceBus

Transports

General

Upgrades Guides

Design Patterns

Azure Service Bus  
Integration

Backwards Compatibility

Transaction Support

Operational Scripting

Azure Service Bus (Legacy)

Azure Storage Queues

SQL Server

MSMQ

RabbitMQ

Amazon SQS



# Thanks

# Slides, Links...

[github.com/danielmarbach/AzureServiceBus.DeepDive](https://github.com/danielmarbach/AzureServiceBus.DeepDive)



# Q & A



Software Engineer  
Microsoft MVP



@danielmarbach  
[particular.net/blog](http://particular.net/blog)  
[planetgeek.ch](http://planetgeek.ch)