<https://learn.microsoft.com/en-us/azure/service-bus-messaging/topic-filters>

<https://sajeetharan.wordpress.com/2020/04/22/best-practices-in-handling-azure-service/>

<https://www.devgem.io/posts/configuring-azure-functions-to-trigger-on-dead-letter-queue-messages-using-terraform>

<https://pmichaels.net/2022/02/19/reading-an-azure-dead-letter-queue-with-azure-messaging-servicebus/>

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-error-pages?tabs=fixed-delay%2Cin-process%2Cnode-v4%2Cpython-v2&pivots=programming-language-csharp#retries>

<https://danyl.hashnode.dev/exponential-retry-with-azure-function-and-service-bus-trigger>

<https://turbo360.com/blog/azure-service-bus-topic-filter>

1. Service Bus Retry

using Azure.Messaging.ServiceBus;  
  
 var retryOptions = new ServiceBusRetryOptions  
 {  
 MaxRetries = 3,  
 RetryDelay = TimeSpan.FromSeconds(5),  
 RetryPolicyType = RetryPolicyType.Exponential  
 };  
  
 var client = new ServiceBusClient(connectionString, retryOptions);

var clientOptions = new ServiceBusClientOptions  
{  
 RetryOptions = new ServiceBusRetryOptions  
 {  
 MaxRetries = 4,  
 Delay = TimeSpan.FromSeconds(4),  
 MaximumDelay = TimeSpan.FromSeconds(30),  
 Mode = ServiceBusRetryMode.Exponential  
 }  
};  
var client = new ServiceBusClient("your-connection-string", clientOptions);

1. Configure Retry Policy in host.json:

* **Exponential Backoff:** This strategy retries with increasing delays between attempts.

{  
 "version": "2",  
 "retry": {  
 "strategy": "ExponentialBackoff",  
 "maxRetryCount": 5,  
 "minimumInterval": "00:00:10",  
 "maximumInterval": "00:01:00"  
 }  
 }

* **Fixed Delay:** This strategy retries with a constant delay between attempts.

{  
 "version": "2",  
 "retry": {  
 "strategy": "FixedDelayRetry",  
 "maxRetryCount": 3,  
 "delayInterval": "00:00:30"  
 }  
 }

Function level retry policy is not supported for service bus.

* **Key Points:**
  + maxRetryCount: The maximum number of retry attempts.
  + minimumInterval/delayInterval: The initial delay before the first retry.
  + maximumInterval: The maximum delay between retries (for exponential backoff).
  + strategy: Choose either "ExponentialBackoff" or "FixedDelayRetry".

2. Handle Message Completion:

* autoCompleteMessages: Set autoCompleteMessages to false in your host.json file to manually control message completion.

Code

{  
 "version": "2",  
 "serviceBus": {  
 "autoCompleteMessages": false  
 }  
 }

* **Manual Completion/Abandonment:** In your function code, after processing a message:
  + If successful, complete the message:

Code

await message.CompleteAsync();

* If the message processing fails, abandon the message to allow Service Bus to re-queue it for retry:

Code

await message.AbandonAsync();

* **Note:** Abandoning a message increments the Service Bus delivery count.

3. Dead-Lettering:

* If a message fails after the maximum number of retries, it will be moved to a dead-letter queue in Service Bus.
* Ensure your Service Bus topic is configured to handle dead-lettering appropriately.

[Function("ServiceBusDeadLetter")]public async Task RunServiceBusDeadLetterAsync( [ServiceBusTrigger("%ServiceBus.TopicName%", "%ServiceBus.SubscriptionName%/$deadletterqueue", Connection = "ServiceBus.Connection")] ServiceBusReceivedMessage message, FunctionContext context, ServiceBusMessageActions messageActions){ // Here you can define the action taken when a message is detected // in the Dead-Letter Queue such as sending an HTTP request to // notify relevant parties.}

Here we can drill down the reasons into two ways:

1. System-level dead-lettering
2. Application-level dead-lettering

**Reasons for System-level dead-lettering**

* Header Size Exceeded — Maximum header size: 64 KB
* Error on processing subscription rule
* Exceeding time to live value
* Exceeding maxDeliveryCount –
* When DeliveryCount exceeds *MaxDeliveryCount*, the message is moved to the DLQ, specifying the ‘*MaxDeliveryCountExceeded’* reason code. This behaviour cannot be disabled, but you can set MaxDeliveryCount to a very large number. You could specify the Maximum Delivery Count between 1 and 2147483647.

Do you want us to help set up and showcase how to monitor and handle dead-letter messages in a smarter way? [Talk to us](https://www.serverless360.com/request-demo)

* When Session id property is set to true (the default is false)

**Reasons for Application-level dead-lettering**

* Messages that cannot be properly processed due to any sort of system issue
* Messages that hold malformed payloads
* Messages that fail authentication when some message-level security scheme is used

<queue path>/$deadletterqueue

<topic path>/Subscriptions/<subscription path>/$deadletterqueue

QueueClient.FormatDeadLetterPath(queuePath)

SubscriptionClient.FormatDeadLetterPath(topicPath, subscriptionName)

**Usage patterns**

* **Broadcast** pattern

The simplest usage scenario for a topic is that every subscription gets a copy of each message sent to a topic, which enables a broadcast pattern.

* **Partitioning** pattern

Partitioning uses filters to distribute messages across several existing topic subscriptions in a predictable and **mutually exclusive** manner. The partitioning pattern is used when a system is scaled out to handle many different contexts in functionally identical compartments that each hold a subset of the overall data; for example, customer profile information. With partitioning, a publisher submits the message into a topic without requiring any knowledge of the partitioning model. The message then is moved to the correct subscription from which it can then be retrieved by the partition's message handler.

* **Routing** pattern

Routing uses filters to distribute messages across topic subscriptions in a predictable fashion, but **not necessarily exclusive**. In conjunction with the [auto forwarding](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-auto-forwarding) feature, topic filters can be used to create complex routing graphs within a Service Bus namespace for message distribution within an Azure region. With Azure Functions or Azure Logic Apps acting as a bridge between Azure Service Bus namespaces, you can create complex global topologies with direct integration into line-of-business applications.