Manual Técnico – Consumo de Exchange RabbitMQ em .NET

Este manual descreve de forma objetiva como um time de desenvolvimento deve consumir mensagens de um Exchange RabbitMQ em aplicações .NET, seguindo boas práticas de arquitetura, resiliência e observabilidade.

1. Arquitetura Recomendada

A aplicação .NET atua como Consumer, conectando-se ao RabbitMQ, declarando a fila, realizando o binding com o Exchange e processando mensagens de forma assíncrona com ACK manual. Recomenda-se o uso de Worker Service ou BackgroundService.

2. Conceitos Essenciais

Componente	Descrição
Exchange	Responsável por rotear mensagens para filas
Queue	Armazena mensagens até que sejam consumidas
Binding	Regra de roteamento entre Exchange e Queue
Routing Key	Chave usada para decisão de roteamento
ACK	Confirmação explícita do processamento

3. Dependências

Adicionar o pacote oficial do RabbitMQ ao projeto:

```
dotnet add package RabbitMQ.Client
```

4. Configuração de Conexão

As configurações devem ser externalizadas no appsettings.json.

```
{
  "RabbitMQ": {
    "Host": "localhost",
    "Username": "guest",
    "Password": "guest",
    "Exchange": "orders.exchange",
    "Queue": "orders.queue",
    "RoutingKey": "orders.created"
  }
}
```

5. Exemplo de Consumer em .NET

Exemplo simplificado de um consumidor utilizando ACK manual e QoS.

```
var factory = new ConnectionFactory
{
```

```
    HostName = "localhost",
    UserName = "guest",
    Password = "guest"
};

using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

channel.ExchangeDeclare("orders.exchange", ExchangeType.Direct, durable: true);
channel.QueueDeclare("orders.queue", durable: true, exclusive: false, autoDelete: false);
channel.QueueBind("orders.queue", "orders.exchange", "orders.created");

channel.BasicQos(0, 1, false);

var consumer = new EventingBasicConsumer(channel);
consumer.Received += (sender, eventArgs) =>
{
    try
    {
        var body = eventArgs.Body.ToArray();
        var message = Encoding.UTF8.GetString(body);

        // Processamento da mensagem
        Console.WriteLine(message);

        channel.BasicAck(eventArgs.DeliveryTag, false);
    }
    catch (Exception)
    {
        channel.BasicNack(eventArgs.DeliveryTag, false, true);
    }
};

channel.BasicConsume("orders.queue", autoAck: false, consumer);
```

6. Boas Práticas Arquiteturais

- Utilizar ACK manual para garantir processamento confiável
- Configurar QoS para evitar sobrecarga do consumidor
- Implementar retry com Dead Letter Queue (DLQ)
- Evitar lógica de negócio diretamente no consumer
- Centralizar logs e métricas

7. Observabilidade e Operação

Utilize o RabbitMQ Management Plugin para monitorar filas, mensagens não processadas e consumidores. Integre logs estruturados e métricas de falha e latência.