

RabbitMQ

O RabbitMQ é uma ferramenta a nível de servidor para envio de mensagens assíncronas através de diversos protocolos.

Instalação:

Existem diferentes formas de instalação do RabbitMQ, a mais fácil é através da imagem do Docker:

```
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.11-management
```

A partir disso vamos poder acessar a aplicação do rabbitmq via web pela porta 15672 e internamente via 5672.

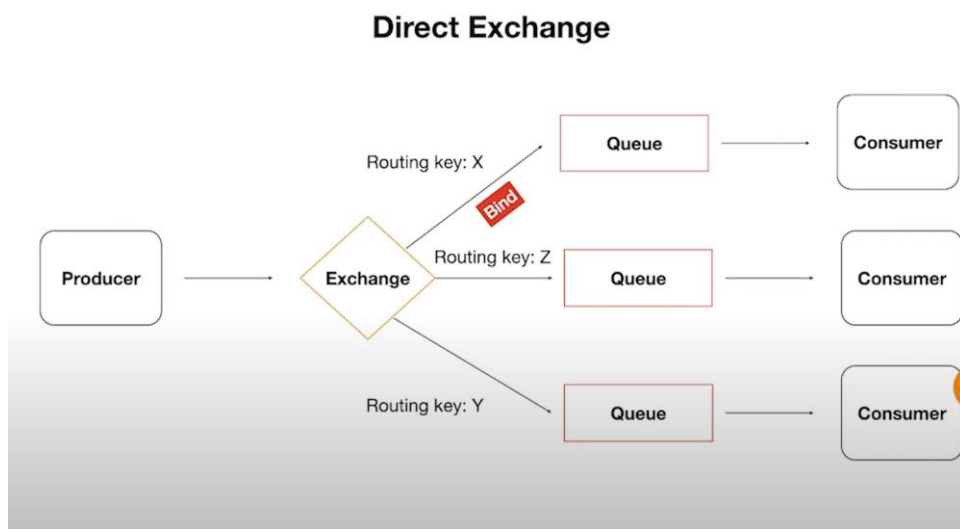
Entendendo o ciclo de vida do RabbitMQ:

É importante entendermos que existem diferentes tipos de Exchange e o ciclo de vida vai depender do tipo de Exchange

Tipos de Exchange:

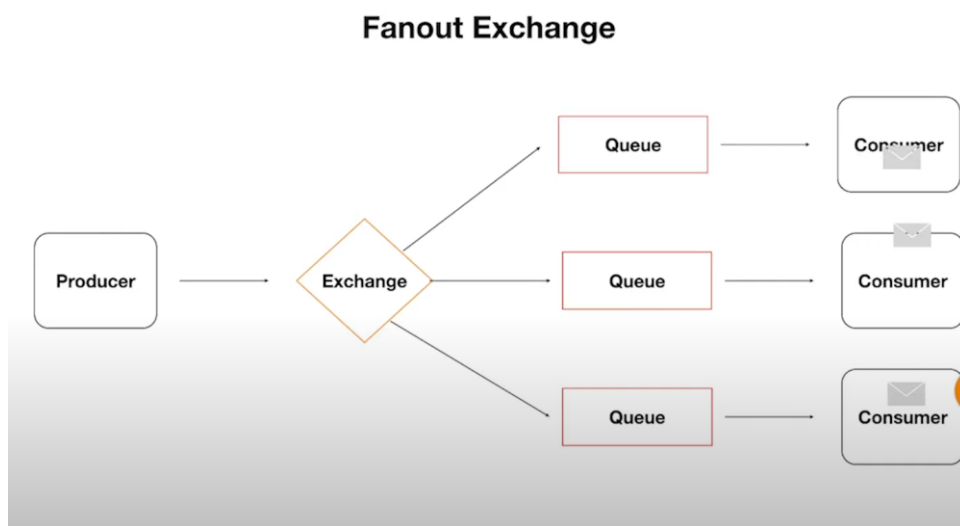
- > Direct
- > Fanout
- > Topic
- > headers

Ciclo de vida Direct Exchange:



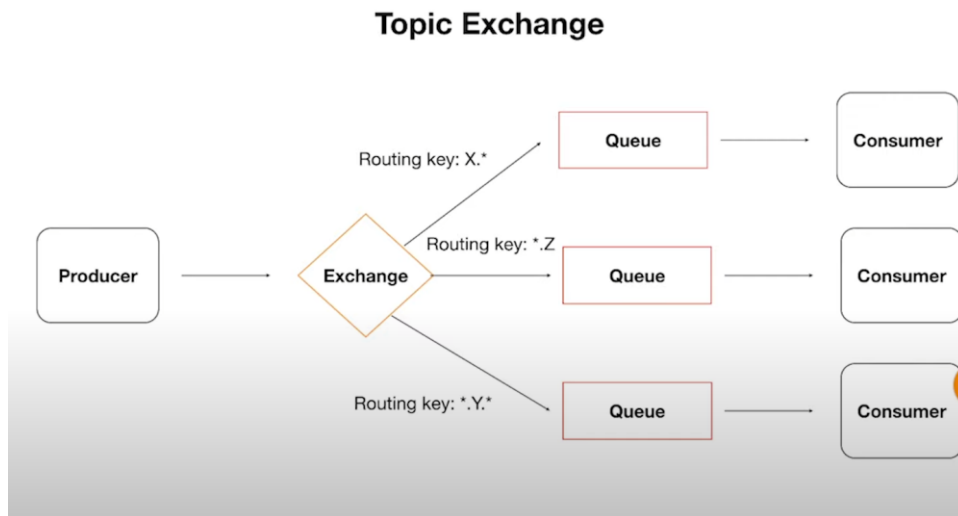
Quando falamos de Direct Exchange estamos falando de forma simplificada que cada fila terá uma bind que é uma conexão entre a fila e a Exchange e nessa bind terá uma Routing key, assim quando enviamos uma mensagem com uma Routing key o Exchange irá direcionar a mensagem apenas para a fila que possui a Routing key especificada.

Ciclo de vida Fanout Exchange:



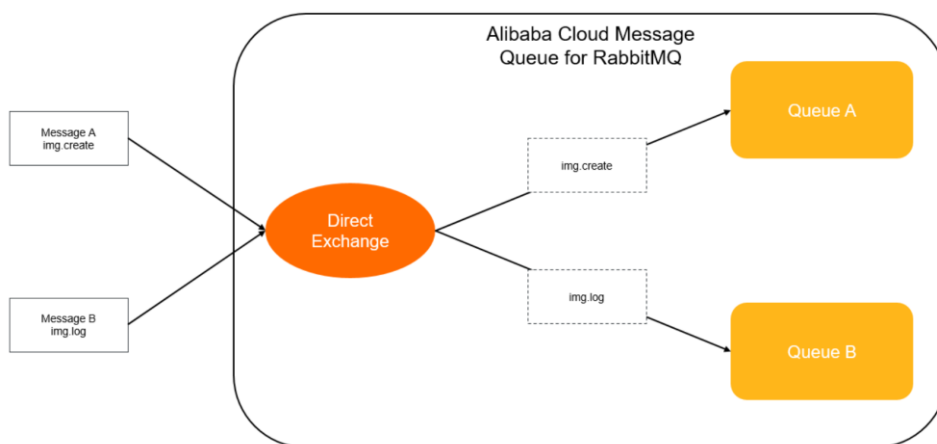
Nesse ciclo de vida quando mandamos uma mensagem para a Exchange ela direciona para todas as filas.

Ciclo de vida Topic Exchange:



No Topic Exchange a Routing key pode ser modificada por exemplo: se temos a Routing key `x.*` então qualquer mensagem com a Routing key começada com `x.qualquerValor` será direcionada para a Routing key `x.*`, a mesma coisa também acontece com o inverso `*.x.*`.

Ciclo de vida Headers Exchange:




Nesse tipo de ciclo de vida o Exchange envia as mensagens para uma fila baseada em um header da mensagem.


Criando uma classe de envio e recebimento simples:


Basicamente vamos criar dois projetos um que irá enviar uma mensagem e outro que irá receber:

Pesquisar em Gerenciador de Soluções (Ctrl+Q)


 Solução 'WebApplication2' (2 de 2 projetos)


◀  **Receiver**

▷  Dependências

▷  Program.cs

◀  **Send**

▷  Dependências

▷  Program.cs

Send:

```
1 using System.Text;
2 using RabbitMQ.Client;
3
4 var factory = new ConnectionFactory { HostName = "localhost" };
5 using var connection = factory.CreateConnection();
6 using var channel = connection.CreateModel();
7
8 channel.QueueDeclare(queue: "hello",
9                     durable: false,
10                    exclusive: false,
11                    autoDelete: false,
12                    arguments: null);
13
14 Console.WriteLine("Escreva uma mensagem");
15 string message = Console.ReadLine();
16 var body = Encoding.UTF8.GetBytes(message);
17
18 channel.BasicPublish(exchange: string.Empty,
19                    routingKey: "hello",
20                    basicProperties: null,
21                    body: body);
22 Console.WriteLine($" [x] Sent {message}");
23
24 Console.WriteLine(" Press [enter] to exit.");
25 Console.ReadLine();
```

Basicamente o send é bem simples é apenas uma conexão com o nosso RabbitMQ onde criamos um canal de comunicação com o nome "hello" codificamos uma mensagem e enviamos ela nessa rota.

Receiver:

```
1 using System.Text;
2 using RabbitMQ.Client;
3 using RabbitMQ.Client.Events;
4
5 var factory = new ConnectionFactory { HostName = "localhost" };
6 using var connection = factory.CreateConnection();
7 using var channel = connection.CreateModel();
8
9 channel.QueueDeclare(queue: "hello",
10                    durable: false,
11                    exclusive: false,
12                    autoDelete: false,
13                    arguments: null);
14
15 Console.WriteLine(" [*] Waiting for messages.");
16
17 var consumer = new EventingBasicConsumer(channel);
18 consumer.Received += (model, ea) =>
19 {
20     var body = ea.Body.ToArray();
21     var message = Encoding.UTF8.GetString(body);
22     Console.WriteLine($" [x] Received {message}");
23 };
24 channel.BasicConsume(queue: "hello",
25                    autoAck: true,
26                    consumer: consumer);
27
28 Console.WriteLine(" Press [enter] to exit.");
29 Console.ReadLine();
```

O nosso receiver também é bem simples, apenas estabelecemos a mesma conexão que o send e criamos a rota “hello” pois ela pode não existir, depois disso criamos um evento consumer que será adicionado ao método BasicConsume e dentro desse método que criamos vamos imprimir a mensagem.

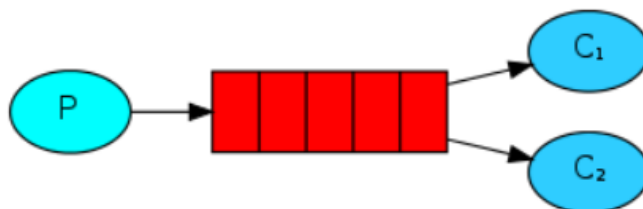
A partir disso criamos a estrutura mais simples possível do RabbitMQ, onde temos apenas um produtor uma rota e um receiver:



Filas de trabalho:

Como forma de economizar recursos e otimizar a entrega de mensagens o RabbitMQ possui a funcionalidade de filas de mensagens que basicamente adicionam diversas mensagens em uma única fila e conforme forem sendo recebidas pelos clientes são removidas das filas.

Nesse tipo de mensageria podemos ter diversos usuários usando uma única fila, porém não existe uma forma de organizar qual dos usuários receberá qual mensagem, basicamente funciona por ordem de chegada o usuário que fizer a solicitação primeiro pega a primeira mensagem e assim sucessivamente:



-> Receiver:

```
var factory = new ConnectionFactory() { HostName = "localhost" };
using (var connection = factory.CreateConnection())
using (var channel = connection.CreateModel())
{
    channel.QueueDeclare(queue: "fila_trabalho",
                        durable: true,
                        exclusive: false,
                        autoDelete: false,
                        arguments: null);

    channel.BasicQos(prefetchSize: 0, prefetchCount: 1, global: false);

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

        Console.WriteLine("Mensagem recebida: {0}", message);

        // Simulando um processamento demorado
        System.Threading.Thread.Sleep(2000);

        Console.WriteLine("Processamento concluído.");

        channel.BasicAck(deliveryTag: ea.DeliveryTag, multiple: false);
    };

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

    Console.WriteLine("Aguardando mensagens...");
    Console.ReadLine();
}
```

Perceba que existem algumas diferenças nesse código em relação ao Receiver de uma única mensagem são elas:

1. Os parâmetros **durable**, **exclusive** e **autoDelete** definem a durabilidade, exclusividade e auto exclusão da fila, respectivamente.
2. Utiliza o **BasicQos** para processar apenas uma mensagem por vez.
3. Confirma o processamento da mensagem enviando uma confirmação (**BasicAck**) para o RabbitMQ.

->Sender:

Exemplo 1:

```
var factory = new ConnectionFactory() { HostName = "localhost" };
using (var connection = factory.CreateConnection())
using (var channel = connection.CreateModel())
{
    channel.QueueDeclare(queue: "fila_trabalho",
                        durable: true,
                        exclusive: false,
                        autoDelete: false,
                        arguments: null);

    string message = "Olá, mundo!";
    var body = Encoding.UTF8.GetBytes(message);

    var properties = channel.CreateBasicProperties();
    properties.Persistent = true;

    channel.BasicPublish(exchange: "",
                        routingKey: "fila_trabalho",
                        basicProperties: properties,
                        body: body);

    Console.WriteLine("Mensagem enviada: {0}", message);
    Console.ReadLine();
}
```

Nesse exemplo a única mudança é que adicionamos a propriedade **properties.Persistent** da mensagem como "true" dessa forma quando consumida a mensagem não será apagada da fila e poderá ser consumida por diversos usuários.

Exemplo 2:

```
var factory = new ConnectionFactory() { HostName = "localhost" };
using (var connection = factory.CreateConnection())
using (var channel = connection.CreateModel())
{
    channel.QueueDeclare(queue: "fila_trabalho",
                        durable: true,
                        exclusive: false,
                        autoDelete: false,
                        arguments: null);

    string[] messages = { "Mensagem 1", "Mensagem 2", "Mensagem 3" };

    foreach (var message in messages)
    {
        var body = Encoding.UTF8.GetBytes(message);

        var properties = channel.CreateBasicProperties();
        properties.Persistent = true;

        channel.BasicPublish(exchange: "",
                            routingKey: "fila_trabalho",
                            basicProperties: properties,
                            body: body);

        Console.WriteLine("Mensagem enviada: {0}", message);
    }

    Console.ReadLine();
}
```

Nesse exemplo não adicionamos uma mensagem que não é apagada quando lida, mas sim uma fila de mensagens, dessa forma diversos usuários podem consumir mensagens diferentes, mas lembrando

não existe uma forma nesse método de garantir que determinado usuário receba determinada mensagem (tudo irá depender de quando o sistema dele consumir a fila).

Publicar/Assinar:

Ate agora vimos exemplos de filas simples, porém quando vemos o conceito de filas no RabbitMQ vemos o conceito de **Exchange** e múltiplas filas, basicamente o Exchange diz o tipo publicação das mensagens e a partir disso a mensagem é publicada em diferentes filas, através dos seus routingKey e depois disso a fila é destruída quando é consumida, vamos entender como cada um funciona em códigos:

1. Fanout:

```
class Sender
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "logs",
                                    type: ExchangeType.Fanout);

            string[] messages = { "Mensagem 1", "Mensagem 2", "Mensagem 3" };

            foreach (var message in messages)
            {
                var body = Encoding.UTF8.GetBytes(message);

                channel.BasicPublish(exchange: "logs",
                                    routingKey: "",
                                    basicProperties: null,
                                    body: body);

                Console.WriteLine("Mensagem enviada: {0}", message);
            }

            Console.ReadLine();
        }
    }
}
```

Nesse exemplo vemos o padrão de publicar uma mensagem do tipo Fanout, ou seja, todas as filas relacionadas ao exchange “logs” receberam as mensagens enviadas a esse Exchange.

```

class Receiver
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "logs",
                                    type: ExchangeType.Fanout);

            var queueName = channel.QueueDeclare().QueueName;
            channel.QueueBind(queue: queueName,
                              exchange: "logs",
                              routingKey: "");

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

                Console.WriteLine("Mensagem recebida: {0}", message);
            };

            channel.BasicConsume(queue: queueName,
                                 autoAck: true,
                                 consumer: consumer);

            Console.WriteLine("Aguardando mensagens...");
            Console.ReadLine();
        }
    }
}

```

A diferença desse Receiver para os demais é que ele busca por um Exchange específico chamado “logs” do tipo Fanout.

2. Direct:

```
class Sender
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "logs",
                                    type: ExchangeType.Direct);

            string[] messages = { "Mensagem 1", "Mensagem 2", "Mensagem 3" };

            foreach (var message in messages)
            {
                var body = Encoding.UTF8.GetBytes(message);

                channel.BasicPublish(exchange: "logs",
                                    routingKey: "info",
                                    basicProperties: null,
                                    body: body);

                Console.WriteLine("Mensagem enviada: {0}", message);
            }

            Console.ReadLine();
        }
    }
}
```

Nesse exemplo temos a criação de um Exchange do tipo Direct e o envio para uma determinada fila através do routingKey "info".

```

class Receiver
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "logs",
                                    type: ExchangeType.Direct);

            var queueName = channel.QueueDeclare().QueueName;

            channel.QueueBind(queue: queueName,
                              exchange: "logs",
                              routingKey: "info");

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

                Console.WriteLine("Mensagem recebida: {0}", message);
            };

            channel.BasicConsume(queue: queueName,
                                 autoAck: true,
                                 consumer: consumer);

            Console.WriteLine("Aguardando mensagens...");
            Console.ReadLine();
        }
    }
}

```

Nesse Receiver ele busca no Exchange específico “logs” e na fila com o routingKey “info”.

3. Topic:

```
class Sender
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "logs",
                                    type: ExchangeType.Topic);

            string[] messages = {
                "error: Erro crítico",
                "info: Informação geral",
                "warning: Atenção!"
            };

            foreach (var message in messages)
            {
                var body = Encoding.UTF8.GetBytes(message);

                channel.BasicPublish(exchange: "logs",
                                    routingKey: "logs",
                                    basicProperties: null,
                                    body: body);

                Console.WriteLine("Mensagem enviada: {0}", message);
            }

            Console.ReadLine();
        }
    }
}
```

Nesse exemplo o sender envia mensagens para as routingKeys tenham o nome "logs".

```

class Receiver
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "logs",
                                    type: ExchangeType.Topic);

            var queueName = channel.QueueDeclare().QueueName;

            channel.QueueBind(queue: queueName,
                              exchange: "logs",
                              routingKey: "logs.*");

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

                Console.WriteLine("Mensagem recebida: {0}", message);
            };

            channel.BasicConsume(queue: queueName,
                                autoAck: true,
                                consumer: consumer);

            Console.WriteLine("Aguardando mensagens...");
            Console.ReadLine();
        }
    }
}

```

Regenerate response

A mágica do Topic ocorre no Receiver pois é nele que teremos as routingKey complexas, pois nesse exemplo todas as filas com o routingKey "logs.*" receberam a mensagem, exemplo "logs.errors", "logs.warnings" etc, é importante destacar que podemos adicionar inversamente também "*.logs.*".

4. Headers:

```
class Sender
{
    0 references
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "headers_exchange",
                                   type: ExchangeType.Headers);

            string message = "Hello, RabbitMQ!";
            var headers = new Dictionary<string, object>
            {
                { "header1", "value1" },
                { "header2", "value2" }
            };

            var body = Encoding.UTF8.GetBytes(message);

            var properties = channel.CreateBasicProperties();
            properties.Headers = headers;

            channel.BasicPublish(exchange: "headers_exchange",
                                routingKey: string.Empty,
                                basicProperties: properties,
                                body: body);

            Console.WriteLine("Mensagem enviada: {0}", message);
        }

        Console.ReadLine();
    }
}
```

Perceba que o que muda nesse tipo de Exchange é que as mensagens devem conter um header, que irá servir para direcionar as mensagens para as filas.

É importante destacar que o header não necessariamente deve ser um Dictionary ele apenas tem que ser um objeto que contém um relacionamento entre chave-valor e a chave deve ser uma String enquanto o valor pode ser qualquer tipo, esse tipo de header serve como forma de mapeamento para as mensagens.

```

0 references
class Receiver
{
    static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.ExchangeDeclare(exchange: "headers_exchange", type: ExchangeType.Headers);

            var queueName = channel.QueueDeclare().QueueName;

            var headers = new Dictionary<string, object>
            {
                { "header1", "value1" }
            };

            channel.QueueBind(queue: queueName,
                              exchange: "headers_exchange",
                              routingKey: string.Empty,
                              arguments: headers);

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

                Console.WriteLine("Mensagem recebida: {0}", message);
            };

            channel.BasicConsume(queue: queueName,
                                autoAck: true,
                                consumer: consumer);

            Console.WriteLine("Aguardando mensagens...");
            Console.ReadLine();
        }
    }
}

```

Perceba que o Receiver também tem que declarar o seu respectivo header, porém ele não precisa ser idêntico ao Sender se houver apenas uma correspondência ele receberá a mensagem.