Documentation

Project Description

In this project, the workflow of RabbitMQ using MassTransit has been demonstrated. To show that, 3 different exchange types are used. Those are, Fanout, Direct and Topic. Those exchange types are explained below;

Fanout: Message is published to all queues that are consuming the message.

Direct: There is a “routing key” within the message. That message can be consumed by any number of queues.

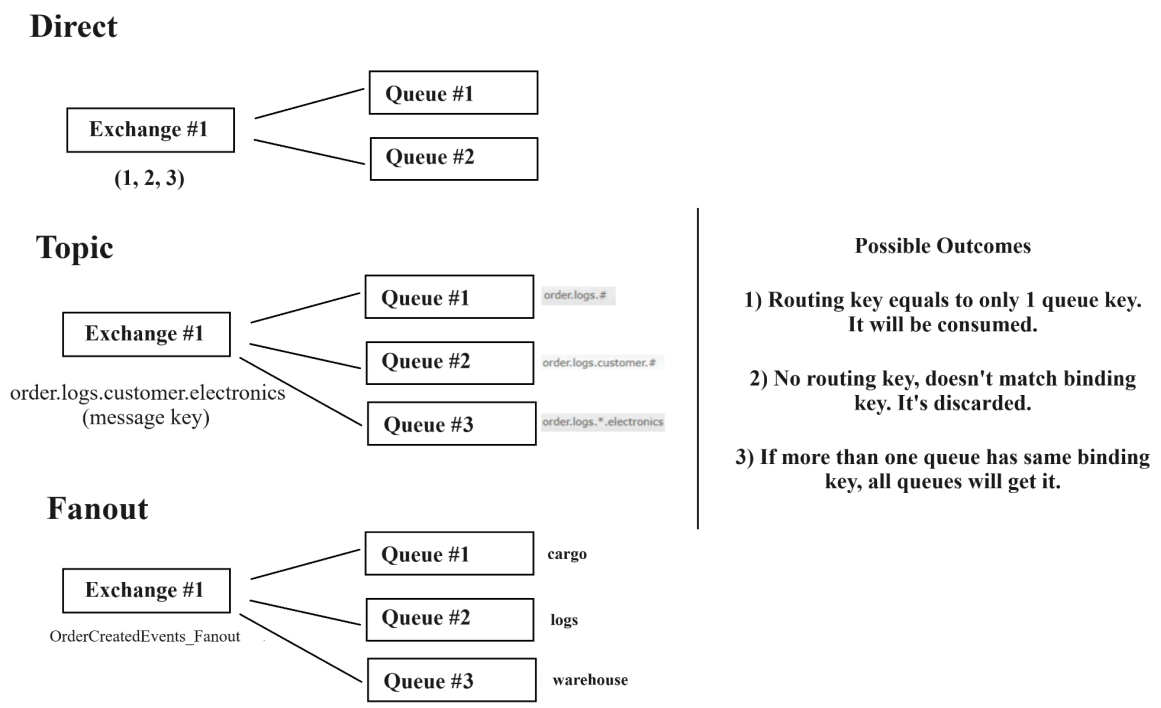
But that message can only be consumed by a queue that has an exact “routing key” which is equal to the “routing key” of the initial message.

Topic: The “routing key” of the message can also include non alphanumeric characters. Such as, “order.test.*.*” or “order.#”.

In this notation “*” represents that there can be any string without “.” escape character. Where “#” notation represents, there can be any string.

As expected, “routing key” of the queues can include non alphanumeric characters too. For example, we’re having two queues with “routing keys” are “order.test.log” and “order.test.prod”.

Therefore, if a message is published with the “routing key” of “order.test.*”. That message will be consumed by both queues respectively.



How to test it?

Project Link

https://github.com/sdsonbay/ExchangeTypes/tree/master

Requirements

Docker

How to run RabbitMQ?

"docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.12-management"

When you run the code above. It is going to host a rabbitmq server.

Default port of that application is 15672, by calling "http://localhost:15672/#/queues" in your browser. You can access the management panel of rabbitmq.

To access the terminal. You can use username as "guest" and password as "guest" too.

You can use another approach for hosting rabbitmq servers as you want. Such as using Docker Desktop.

Test Screenshots (Fanout)

```
await _bus.Publish<IFanoutMessage>(new
{
    Message = message
});
```

```
public class FanoutMessageConsumer : IConsumer<IFanoutMessage>
{
    public Task Consume(ConsumeContext<IFanoutMessage> context)
    {
        Console.WriteLine($"Fanout Message Received. Message: {context.Message.Message}");
        return Task.CompletedTask;
    }
}
```

```
cfg.ReceiveEndpoint("fanout-queue-a", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<FanoutMessageConsumer>(context);
    ep.Bind<IFanoutMessage>(x =>
    {
        x.ExchangeType = ExchangeType.Fanout;
    });
});
cfg.ReceiveEndpoint("fanout-queue-b", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<FanoutMessageConsumer>(context);
    ep.Bind<IFanoutMessage>(x =>
    {
        x.ExchangeType = ExchangeType.Fanout;
    });
});
```

```
Fanout Message Received. Message: string
Fanout Message Received. Message: string
```

fanout-queue-a	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
fanout-queue-b	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s

(Direct)

```
await _bus.Publish<IDirectMessage>(new
{
    Message = message
}, x => x.SetRoutingKey("A"));
```

```
public class DirectMessageConsumer : IConsumer<IDirectMessage>
{
    public Task Consume(ConsumeContext<IDirectMessage> context)
    {
        Console.WriteLine($"Direct Message Received. Message: {context.Message.Message}");
        return Task.CompletedTask;
    }
}
```

```
cfg.ReceiveEndpoint("direct-queue-a", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<DirectMessageConsumer>();
    ep.Bind<IDirectMessage>(z =>
    {
        z.RoutingKey = "A";
        z.ExchangeType = ExchangeType.Direct;
    });
});
cfg.ReceiveEndpoint("direct-queue-b", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<DirectMessageConsumer>();
    ep.Bind<IDirectMessage>(z =>
    {
        z.RoutingKey = "B";
        z.ExchangeType = ExchangeType.Direct;
    });
});
```

```
Direct Message Received. Message: string
```

direct-queue-a	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
direct-queue-b	classic	D	idle	0	0	0			

(Topic)

```
await _bus.Publish<ITopicMessage>(new
{
    Message = message
}, x => x.SetRoutingKey("order.test"));
```

```
public class TopicMessageConsumer : IConsumer<ITopicMessage>
{
    public Task Consume(ConsumeContext<ITopicMessage> context)
    {
        Console.WriteLine($"Topic Message Received. Message: {context.Message.Message}, Source Address: {context.SourceAddress}, Routing Key: {context.R
        return Task.CompletedTask;
    }
}
```

```
cfg.ReceiveEndpoint("topic-queue-a", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<TopicMessageConsumer>(context);
    ep.Bind<ITopicMessage>(x =>
    {
        x.RoutingKey = "order.log.mog";
        x.ExchangeType = ExchangeType.Topic;
    });
});
cfg.ReceiveEndpoint("topic-queue-b", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<TopicMessageConsumer>(context);
    ep.Bind<ITopicMessage>(x =>
    {
        x.RoutingKey = "order.test.*";
        x.ExchangeType = ExchangeType.Topic;
    });
});
cfg.ReceiveEndpoint("topic-queue-c", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<TopicMessageConsumer>(context);
    ep.Bind<ITopicMessage>(x =>
    {
        x.RoutingKey = "order.#";
        x.ExchangeType = ExchangeType.Topic;
    });
});
cfg.ReceiveEndpoint("topic-queue-d", ep =>
{
    ep.ConfigureConsumeTopology = false;
    ep.Consumer<TopicMessageConsumer>(context);
    ep.Bind<ITopicMessage>(x =>
    {
        x.RoutingKey = "order.mog.log";
        x.ExchangeType = ExchangeType.Topic;
    });
});
```

```
Topic Message Received. Message: string, Source Address: rabbitmq://localhost/MUSAHAN_MessagingApp_bus_6bgyyygucksqfuiedqbpquhbz?temporary=true, Routing Key: order.test
```

topic-queue-a	classic	D	idle	0	0	0			
topic-queue-b	classic	D	idle	0	0	0			
topic-queue-c	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
topic-queue-d	classic	D	idle	0	0	0			