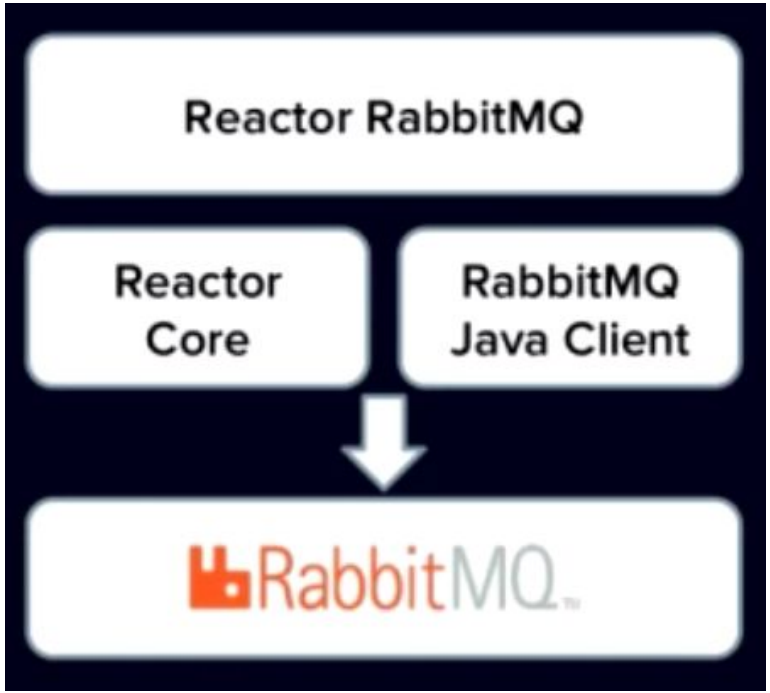
Introduction to Reactor RabbitMQ

Zoltan Altfatter

Reactor RabbitMQ

Use cases:

- Manage exchanges, queues and bindings
- Publish a Flux of messages
- Consume messages as a Flux



Reactor RabbitMQ

```
<dependency>
    <groupId>io.projectreactor.rabbitmq</groupId>
    <artifactId>reactor-rabbitmq</artifactId>
    <version>1.0.0.M3</version>
</dependency>

<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

ReactorRabbitMq

```
public class ReactorRabbitMq {

   public static Receiver createReceiver(ReceiverOptions options) {
       return new Receiver(options);
       // Reactive abstraction to consume messages as a Flux
   }

   public static Receiver createReceiver() {
       return new Receiver();
   }

   public static Sender createSender(SenderOptions options) {
       return new Sender(options);
       // Reactive abstraction to create resources and send messages as a Flux
   }

   public static Sender createSender() {
       return new Sender();
   }
}
```

Sender API

```
Sender sender = ReactorRabbitMq.createSender();

Sender sender = ReactorRabbitMq.createSender(senderOptions);
```

- No connections to RabbitMQ have been made yet
- The underlying Connection is created lazily when creating a resource or sending messages

```
sender.send(flux.map(i -> new OutboundMessage("", QUEUE,
  ("Message_" + i).getBytes())));

Flux<OutboundMessageResult> confirmations = sender.sendWithPublishConfirms(
     Flux.interval(Duration.ofSeconds(1))
                 .map(i -> new OutboundMessage("", QUEUE,
  ("Message_" + i).getBytes())))
```

OutboundMessage and OutboundMessageResult

```
public class OutboundMessage {

     private final String exchange;
     private final String routingKey;
     private final BasicProperties properties;
     private final byte [] body;
}

public class OutboundMessageResult {

     private final OutboundMessage outboundMessage;
     private final boolean ack;
 }
```

Managing resources

```
Mono<AMQP.Exchange.DeclareOk> exchange = sender.declareExchange(
     ExchangeSpecification.exchange("my.exchange")
);
Mono<AMQP.Queue.DeclareOk> queue = sender.declareQueue(
     QueueSpecification.queue("my.queue")
);
Mono<AMQP.Queue.BindOk> binding = sender.bind(
     BindingSpecification.binding().exchange("my.exchange")
               .queue("my.queue").routingKey("a.b")
);

sender.declare(exchange("my.exchange"))
     .then(sender.declare(queue("my.queue")))
     .then(sender.bind(binding("my.exchange", "a.b", "my.queue")))
     .subscribe(r -> System.out.println("Exchange and queue declared and bound"));
```

Receiver API

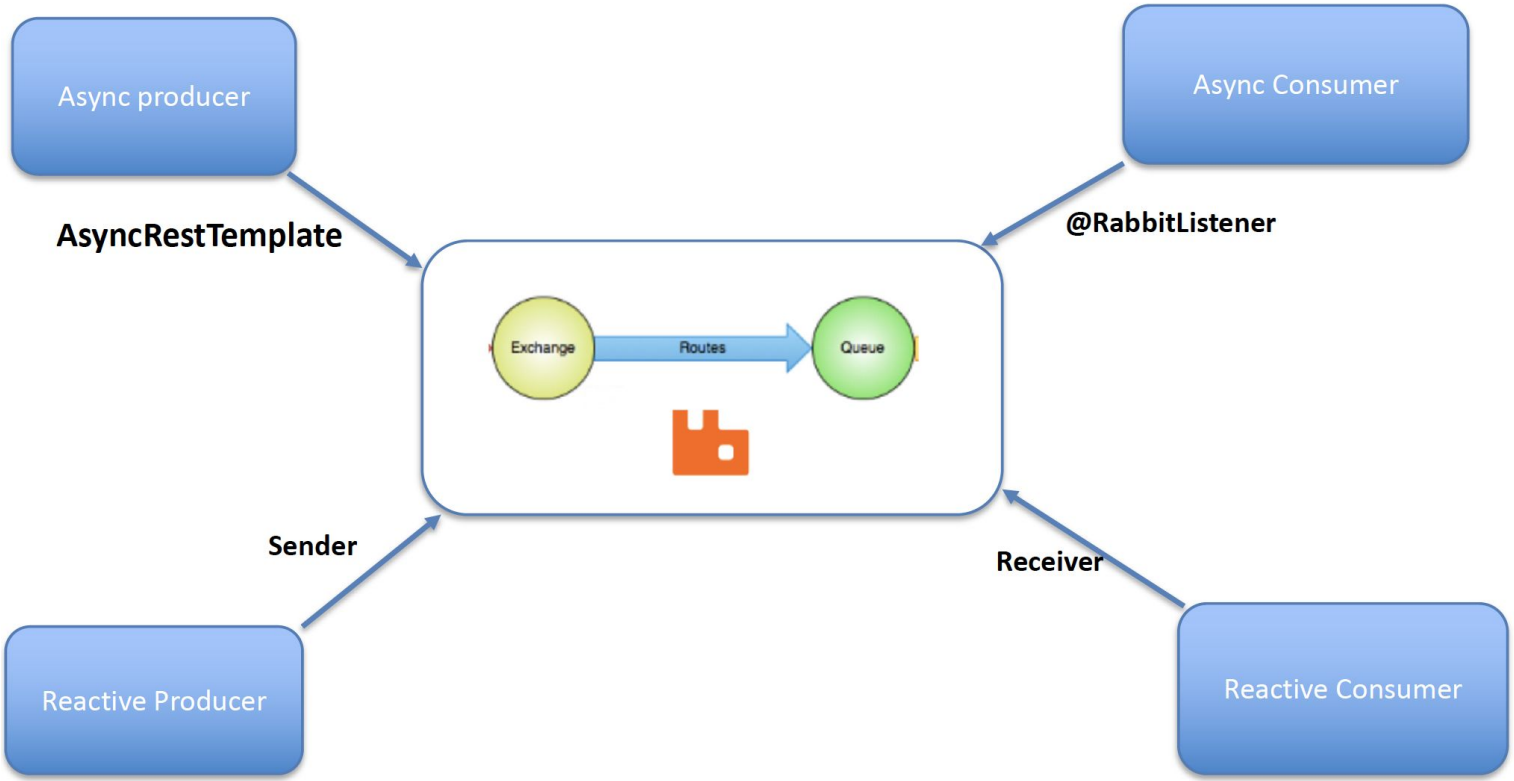
```
Receiver receiver = ReactorRabbitMq.createReceiver();

Flux<Delivery> messages = receiver.consumeAutoAck(QUEUE);

Flux<Delivery> messages = receiver.consumeAutoAck(QUEUE, consumeOptions);

public class Delivery {
   private final Envelope _envelope;
   private final AMQP.BasicProperties _properties;
   private final byte[] _body;
}
```

Demo


Async producer
Async Consumer
AsyncRestTemplate
@RabbitListener
Exchange
Routes
Queue
Sender
Receiver
Reactive Producer
Reactive Consumer