Spring RabbitMQ for High Load

Martin Toshev

MOSCOW bigdata

Who am I

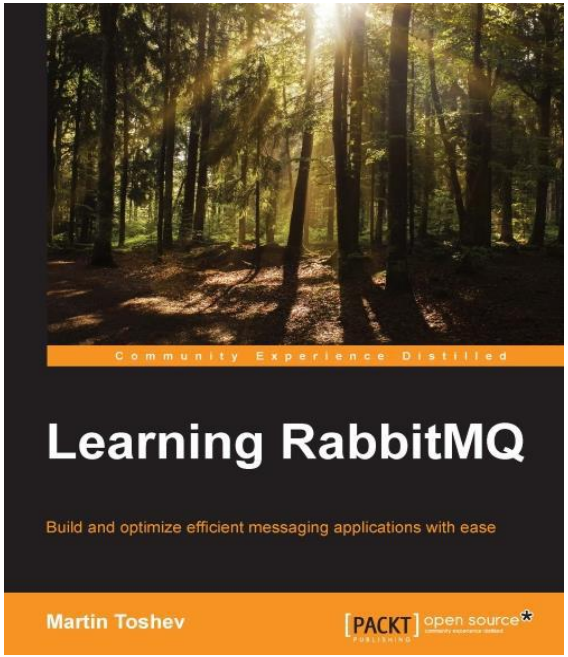
Software consultant (CoffeeCupConsulting)

BG JUG board member (http://jug.bg)

OpenJDK and Oracle RDBMS enthusiast

Twitter: @martin_fmi

MOSCOW bigdata



MOSCOW bigdata

Agenda

- Messaging Basics
- RabbitMQ Overview
- Spring RabbitMQ

MOSCOW bigdata

Messaging Basics

MOSCOW bigdata

Messaging

- Messaging provides a mechanism for loosely-coupled integration of systems

- The central unit of processing in a message is a message which typically contains a **body** and a **header**

MOSCOW bigdata

Use cases

- Offloading long-running tasks to worker nodes
- Distributing and processing high loads of data
- Aggregating logs and propagating events between systems
- Many others …

MOSCOW bigdata

Messaging protocols

- Messaging solutions implement different protocols for transferring of messages such as AMQP, XMPP, MQTT, STOMP, Kafka binary protocol and others

- The variety of protocols imply vendor lock-in

Messaging protocols comparison

	AMQP	MQTT	XMPP	STOMP	Kafka
goal	replacement of proprietary protocols	messaging for resource-constrained devices	instant messaging, adopted for wider use	message-oriented middleware	processing of large real-time data feeds
format	binary	binary	XML-based	text-based	binary
API	divided into classes (> 40 methods in RabbitMQ)	simple (5 basic operations with 2-3 packet types for each)	different XML items with multiple types	~ 10 basic commands	42 request types in latest version (Kafka 2.0.0)
reliability	publisher/subscriber acknowledgements, transactions	acknowledgements	Acknowledgments and resumptions (XEP-198)	Subscriber acknowledgements and transactions	Acknowledgements, transactional replication
security	SASL, TLS/SSL	no built-in TLS/SSL, header authentication	SASL, TLS/SSL	depending on message broker	SASL, TLS/SSL, ACLs
extensibility	extension points	none	extensible	depending on message broker	no extension mechanism defined

Messaging brokers

- A variety of messaging brokers can be a choice for applications …

MOSCOW bigdata

Common broker characteristics

- Secure message transfer, authentication and authorization of messaging endpoints
- Message routing and persistence
- Broker subscriptions

MOSCOW bigdata

RabbitMQ Overview

MOSCOW bigdata

RabbitMQ

- An open source message broker written in Erlang
- Implements the AMQP Protocol (Advanced Message Queueing Protocol)
- Has a pluggable architecture and provides extensions for other protocols such as HTTP, STOMP and MQTT

AMQP

- AMQP is a binary protocol that aims to standardize middleware communication
- Derives its origins from the financial industry
- Defines multiple connection channels inside a single TCP connection

AMQP characteristics

- The AMQP protocol defines:
  - **exchanges** – the message broker endpoints that receive messages
  - **queues** – the message broker endpoints that store messages from exchanges and are used by subscribers for retrieval of messages
  - **bindings** – rules that bind exchanges and queues

- The AMQP protocol is programmable – which means that the above entities can be created/modified/deleted by applications

MOSCOW bigdata

Message handling

- Each message can be published with a **routing key**
- Each binding between an exchange and a queue has a **binding key**
- Routing of messages is determined based on matching between the routing and binding keys

MOSCOW bigdata

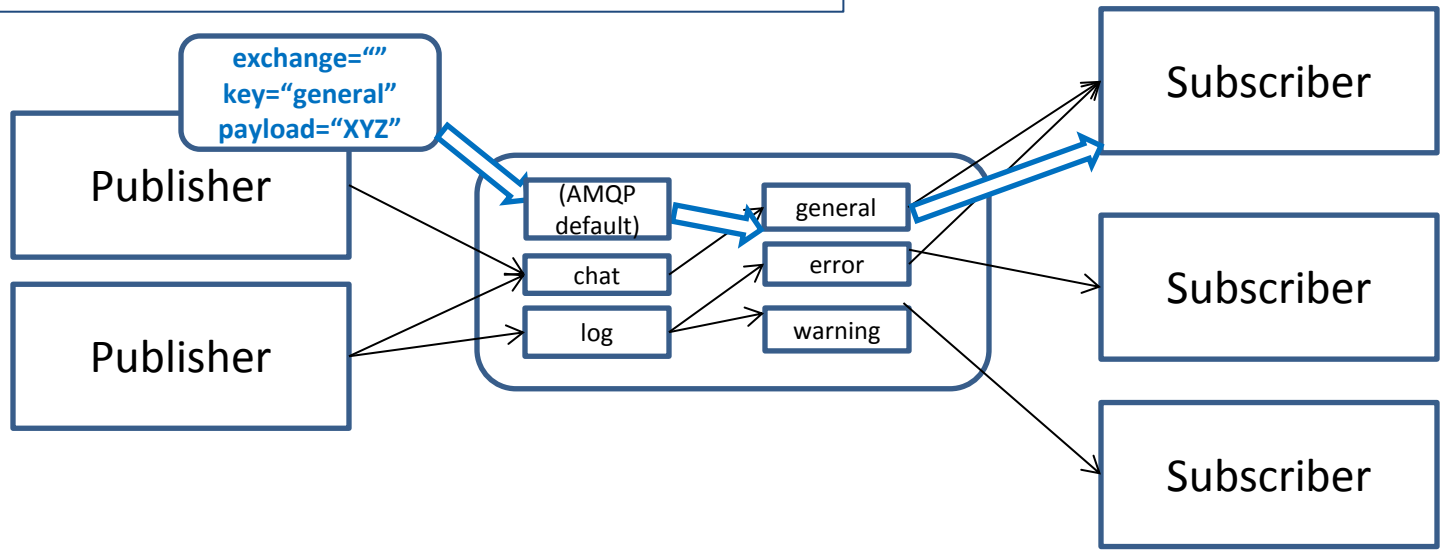
Message routing

- Different types of messaging patterns are implemented by means of different types of exchanges

- RabbitMQ provides the following types of exchanges:
  - direct/default
  - fanout
  - topic
  - headers

Default exchange

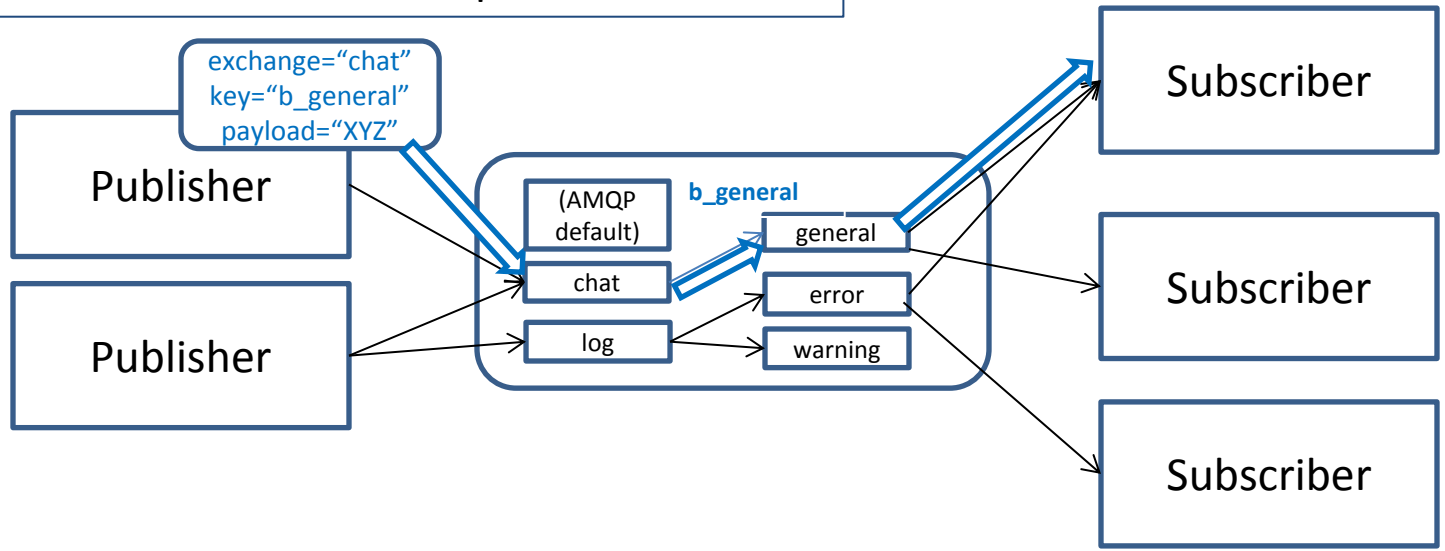
default exchange: suitable for point-to-point communication between endpoints



(AMQP default) *is a system exchange*

Direct exchange

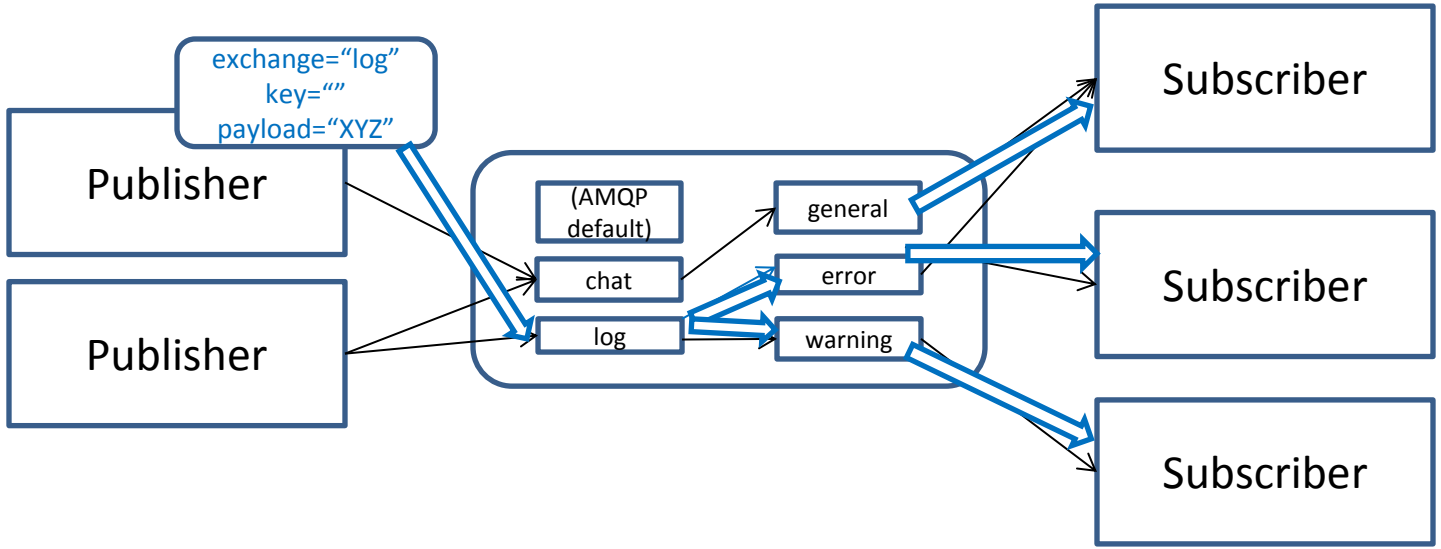
direct exchange: suitable for point-to-point communication between endpoints



***chat** is defined as a direct exchange upon creation*

Fanout exchange

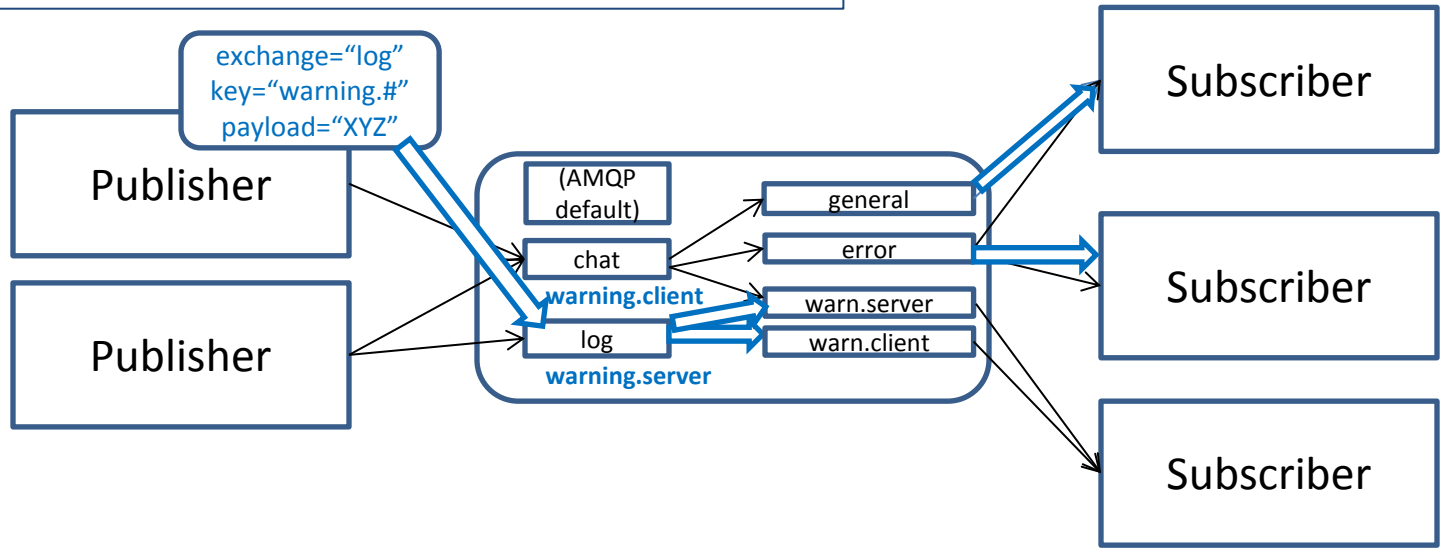
fanout exchange: suitable for broadcast type of communication between endpoints



log *is defined as a fanout exchange upon creation*

Topic exchange

topic exchange: suitable for multicast type of communication between endpoints



***log** is defined as a topic exchange upon creation*

RabbitMQ clustering

- Default clustering mechanism provides scalability in terms of queues rather than high availability

- Mirrored queues are an extension to the default clustering mechanism that can be used to establish high availability at the broker level

MOSCOW bigdata

RabbitMQ Overview
(demo)

MOSCOW bigdata

Spring RabbitMQ

MOSCOW bigdata

Spring RabbitMQ

- The Spring Framework provides support for RabbitMQ by means of:
  - The Spring AMQP framework
  - The Spring Integration framework
  - ~~The Spring XD framework~~ (discontinued as of July 2017)

Spring AMQP

- Provides RabbitMQ utilities such as:
  - the **RabbitAdmin** class for automatically declaring queues, exchanges and bindings
  - Listener containers for asynchronous processing of inbound messages
  - the **RabbitTemplate** class for sending and receiving messages

MOSCOW bigdata

Spring AMQP usage

- Utilities of the Spring AMQP framework can be used either directly in Java or preconfigured in the Spring configuration

MOSCOW bigdata

RabbitAdmin (plain Java)

```
CachingConnectionFactory factory = new
        CachingConnectionFactory("localhost");
Queue queue = new Queue("sample-queue");
TopicExchange exchange =
        new TopicExchange("sample-topic-exchange");
RabbitAdmin admin = new RabbitAdmin(factory);
admin.declareQueue(queue);
admin.declareExchange(exchange);
admin.declareBinding(BindingBuilder.bind(queue).to(exchange)
        .with("sample-key"));
factory.destroy();
```

MOSCOW bigdata

Container listener (plain Java)

```
CachingConnectionFactory factory =
        new CachingConnectionFactory(
"localhost");
SimpleMessageListenerContainer container =
        new SimpleMessageListenerContainer(factory);
Object listener = new Object() {
    public void handleMessage(String message) { … }};
MessageListenerAdapter adapter = new
        MessageListenerAdapter(listener);
container.setMessageListener(adapter);
container.setQueueNames("sample-queue");
container.start();
```

RabbitTemplate (plain Java)

```
CachingConnectionFactory factory =
        new CachingConnectionFactory("localhost");
RabbitTemplate template =
        new RabbitTemplate(factory);
template.convertAndSend("", "sample-queue",
         "sample-queue test message!");
```

MOSCOW bigdata

Spring-based configuration

- All of the above examples can be configured using Spring configuration
- Cleaner and decouples RabbitMQ configuration for the business logic

MOSCOW bigdata

RabbitTemplate (Spring configuration)

```
<rabbit:connection-factory
        id="connectionFactory"
        host="localhost" />

<rabbit:template id="amqpTemplate"
        connection-factory="connectionFactory"
        exchange=""
        routing-key="sample-queue-spring"/>
```

MOSCOW bigdata

Container listener (Spring configuration)

```
<rabbit:listener-container
       connection-factory="connectionFactory">
       <rabbit:listener ref="springListener"
              method="receiveMessage"
              queue-names="sample-queue-spring" />
</rabbit:listener-container>

<bean id="springListener"
class="ua.org.javaday.rabbitmq.spring.ListenerSpringExample"/>
```

MOSCOW bigdata

Container listener (Spring configuration)

```
public class ListenerSpringExample {
	public void receiveMessage(String message) {
		System.out.println("Message received: " +
			message);
	}
}
```

Container listener (Spring annotations)

```
public class ListenerSpringExample {
        @RabbitListener(queues = "sample-queue-spring")
        public void receiveMessage(String message) {
                System.out.println("Message received: " +
                        message);
        }
}
```

RabbitAdmin
(Spring configuration)

```
<rabbit:admin id="amqpAdmin"
        connection-factory="connectionFactory" />
```

Spring Boot

- If you don’t want to use xml-based configuration you can use Spring Boot …

MOSCOW bigdata

Spring Boot

```
@SpringBootApplication
public class AppConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory connectionFactory =
            new CachingConnectionFactory("localhost");
        return connectionFactory;
    }
    @Bean
    public AmqpAdmin amqpAdmin() {
        return new RabbitAdmin(connectionFactory());
    }
}
```

Spring Integration AMQP

- The Spring Integration Framework provides:
  - **Inbound-channel-adapter** for reading messages from a queue
  - **outbound-channel-adapter** for sending messages to an exchange

MOSCOW bigdata

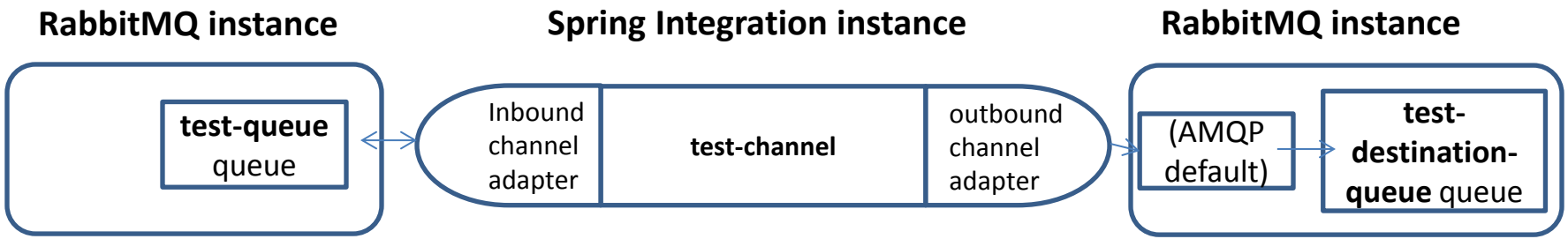
Spring Integration AMQP

- The Spring Integration Framework provides:
  - **Inbound-gateway** for request-reply communication at the publisher
  - **outbound-gateway** for request-reply communication at the receiver

MOSCOW bigdata

Spring Integration AMQP scenario

- Message replication with Spring integration:



Spring Integration AMQP scenario

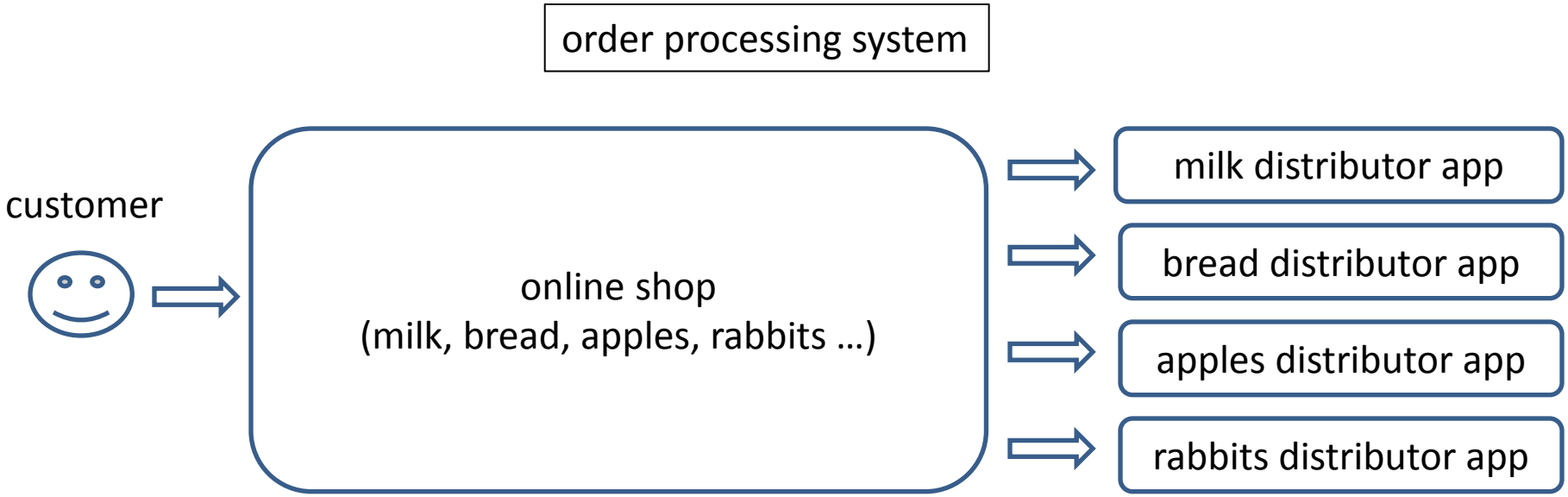
```
<rabbit:connection-factory
        id="connectionFactory"
        host="localhost" />
<channel id="test-channel" />

<rabbit:queue name="test-queue" />
<rabbit:queue name="test-destination-queue" />

<rabbit:template id="amqpTemplate"
        connection-factory="connectionFactory"
        exchange=""
        routing-key="test-queue" />
```

Spring Integration AMQP scenario

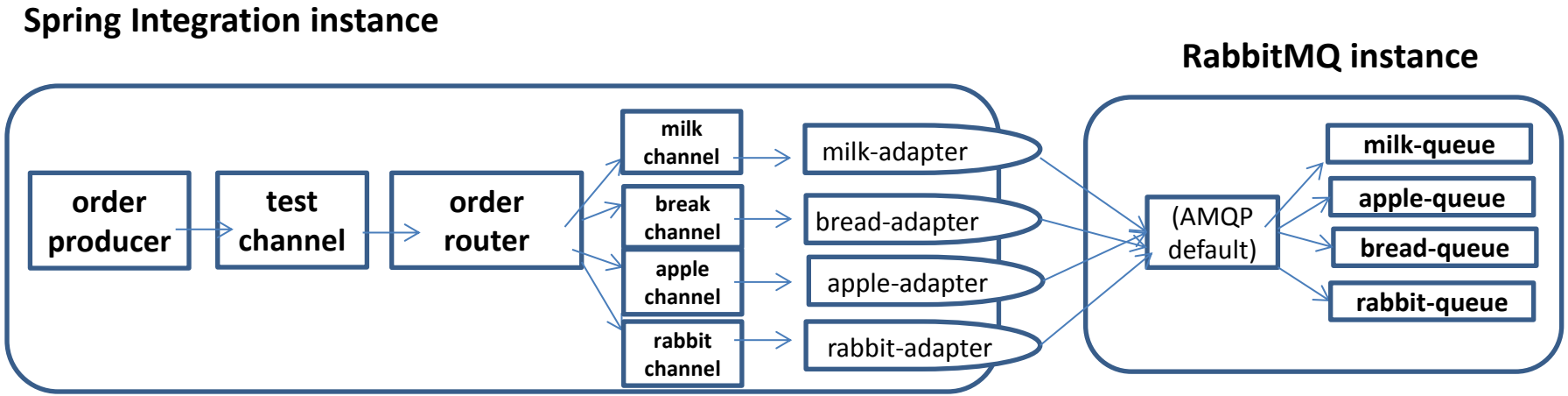
```
<amqp:inbound-channel-adapter
        channel="test-channel"
        queue-names="test-queue"
        connection-factory="connectionFactory" />

<amqp:outbound-channel-adapter
        channel="test-channel"
        exchange-name=""
        routing-key="test-destination-queue"
        amqp-template="amqpTemplate" />
```

MOSCOW bigdata

Yet another scenario



Implementation



Spring RabbitMQ

(demo)

MOSCOW bigdata

Summary

- The Spring Framework provides convenient utilities and adapters for integrating with RabbitMQ
- Favor them over the RabbitMQ Java library in Spring-based applications

MOSCOW bigdata

Thank you !

Q&A

demos: https://github.com/martinfmi/spring_rabbitmq_samples

MOSCOW bigdata

References

AMQP 0.9.1 specification
https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

AMQP list of users
http://www.amqp.org/about/examples

RabbitMQ documentation
http://www.rabbitmq.com/documentation.html

MOSCOW bigdata

References

Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP

http://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html

Spring AMQP reference

http://docs.spring.io/spring-amqp/reference/html/

Spring Integration AMQP

http://docs.spring.io/spring-integration/reference/html/amqp.html

MOSCOW bigdata