Spring Integration: Using Channel Adapters to Integrate with External Systems

INTEGRATING WITH MESSAGE BROKERS

Steven Haines
PRINCIPAL SOFTWARE ARCHITECT

@geekcap www.geekcap.com

Overview

Overview of Channel Adapters

RabbitMQ (AMQP)

ActiveMQ (JMS)

Channel Adapter

A Channel Adapter is a Message Endpoint that enables connecting a single sender or receiver to a Message Channel

Channel Adapter Types

Inbound Channel Adapter

One-way integration inbound

Outbound Channel Adapter

One-way integration outbound

Inbound Gateway

Bidirectional integration inbound

Outbound Gateway

Bidirectional integration outbound

Examples in Docker

The examples in this module will connect to message brokers running in preconfigured publicly available Docker containers

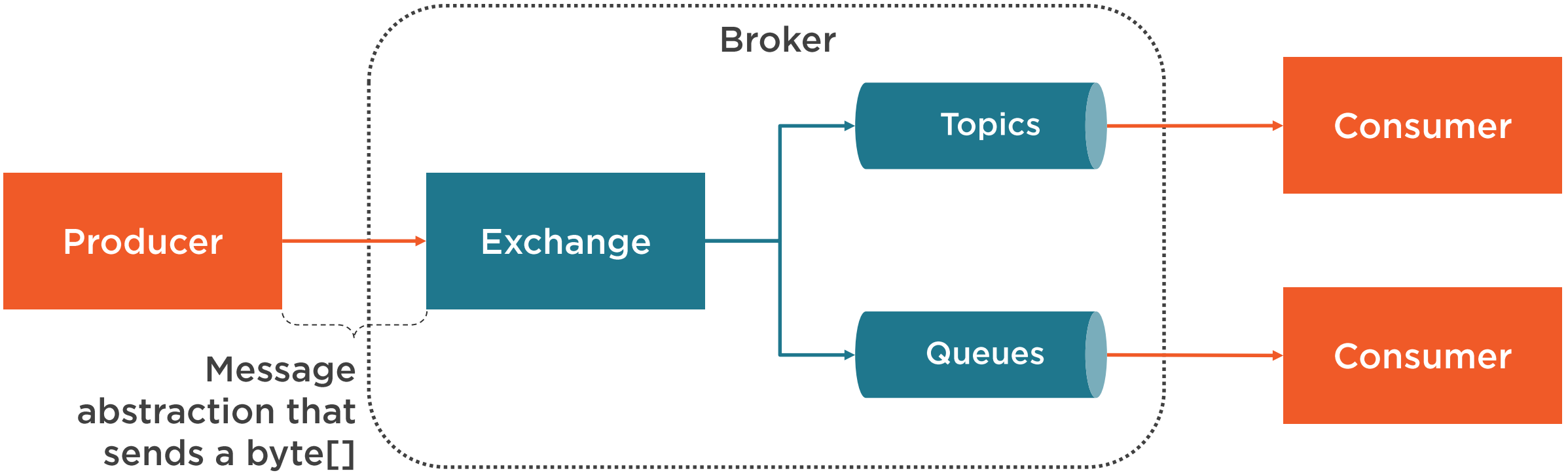
https://www.docker.com/products/docker-desktop

RabbitMQ (AMPQ)

Advanced Message Queuing Protocol (AMQP)

The Advanced Message Queuing Protocol is an open standard for passing business messages between applications or organizations.

AMQP


Broker
Producer
Exchange
Topics
Queues
Consumer
Consumer
Message abstraction that sends a byte[]

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

AmqpInboundChannelAdapter

Outbound Channel Adapter

AmqpOutboundEndpoint

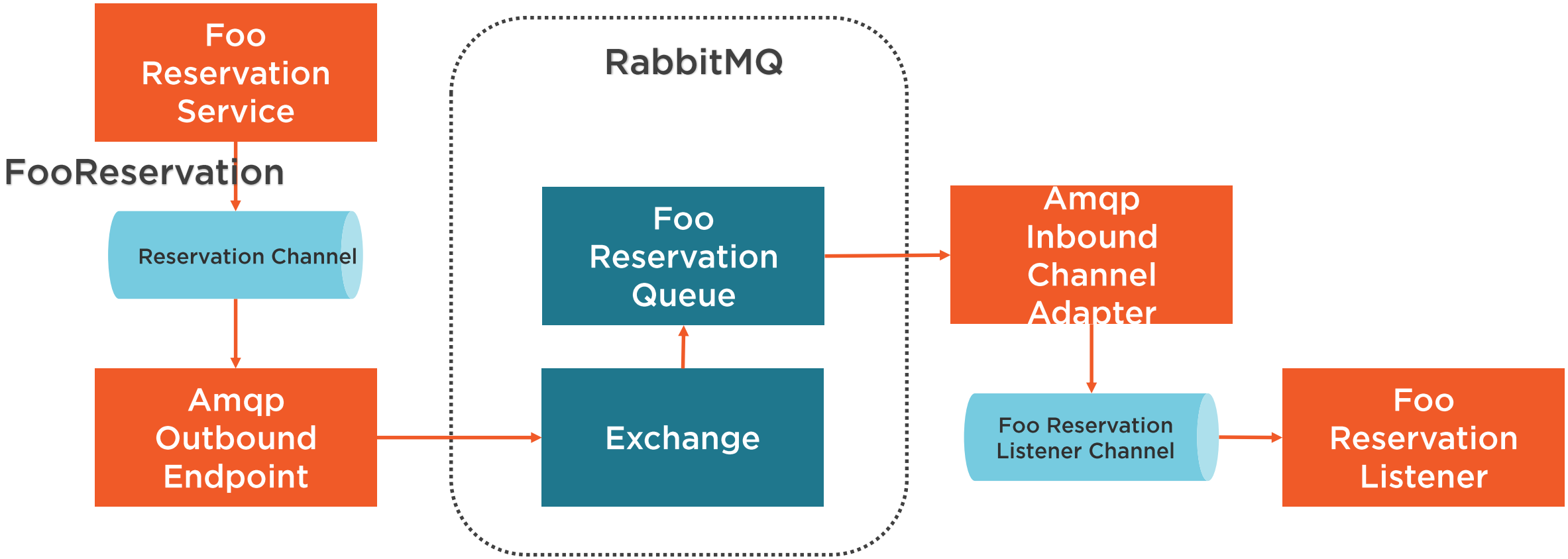
```
@Configuration
public class RabbitMQInboundConfig {
    @Bean
    public MessageChannel amqpInputChannel() {
… }

    @Bean
    SimpleMessageListenerContainer container(

ConnectionFactory connectionFactory) {
        SimpleMessageListenerContainer
container =
                                    new
SimpleMessageListenerContainer();

container.setConnectionFactory(connectionFacto
ry);
        container.setQueueNames(“queueName”);
        return container;
    }

    @Bean
    public AmqpInboundChannelAdapter inbound(
             SimpleMessageListenerContainer
listenerContainer) {
        AmqpInboundChannelAdapter adapter =
```

- Setup a configuration class and enable Spring Integration
- Configure a container that the inbound channel adapter will use to connect to the broker and queue
- Set the name of the queue from which to receive message
- Create an AmpInboundChannelAdapter

```
@Configuration
public class RabbitMQOutboundConfig {
    @Bean
    public MessageChannel
amqpOutboundChannel() { … }

    @Bean
    Queue queue() {
        return new Queue(“queueName”, false);
    }

    @Bean
    @ServiceActivator(inputChannel =

"amqpOutboundChannel")
    public AmqpOutboundEndpoint amqpOutbound(

AmqpTemplate amqpTemplate) {
        AmqpOutboundEndpoint outbound =
                        new
AmqpOutboundEndpoint(amqpTemplate);
        outbound.setRoutingKey(“queueName”);
        return outbound;
    }
}
```

- Setup a configuration class and enable Spring Integration
- Create the queue to which we will publish messages
- Create an outbound endpoint
- Specify the queue name as the routing key

Example: Reservation Service



```
docker run -d --hostname rabbit-host -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

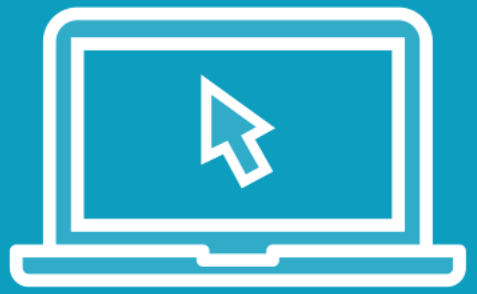
Running RabbitMQ in Docker

Official RabbitMQ Docker Image (with Management UI): rabbitmq:3-management

RabbitMQ DockerHub Link: https://hub.docker.com/_/rabbitmq

Management UI: http://localhost:15672/ , default username/password: guest/guest

Demo



Build our applications

- Foo Reservation Publisher
- Globomantics Registration Service

Publish Foo reservations to RabbitMQ

Validate that we receive those reservation messages

RabbitMQ – Inbound and Outbound Gateways

Inbound and Outbound Gateways

Inbound Gateway

AmqpInboundGateway

Outbound Gateway

AmqpOutboundEndpoint

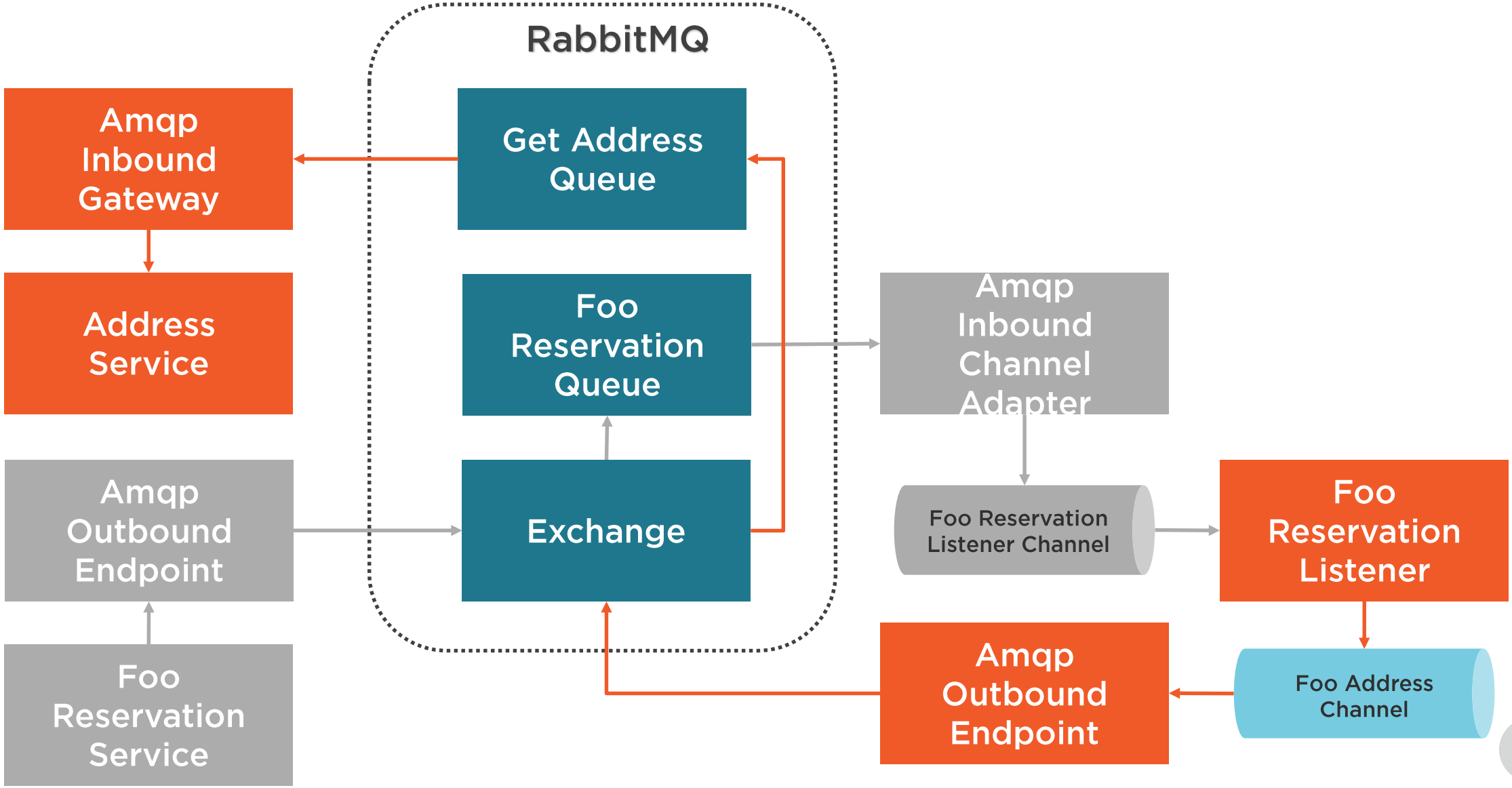
```
@Configuration
public class FooAddressConfig {
    @Bean
    public MessageChannel
getAddressInputChannel() { … }

    @Bean
    public SimpleMessageListenerContainer
container(

ConnectionFactory cf) {
        SimpleMessageListenerContainer
container =
                            new
SimpleMessageListenerContainer(cf);
        container.setQueueNames(“queueName”);
        return container;
    }

    @Bean
    public AmqpInboundGateway inbound(
          SimpleMessageListenerContainer
listenerContainer,
          MessageChannel channel) {
        AmqpInboundGateway gateway =
                        new
```

- Setup a configuration class and enable Spring Integration
- Configure a container that the inbound gateway will use to connect to the broker and queue
- Set the name of the queue from which to receive message
- Create an AmpInboundGateway
- Specify the name of a reply queue

```
@Configuration
public class RabbitMQOutboundConfig {
    @Bean
    public MessageChannel fooAddressChannel()
{ … }

    @Bean
    @ServiceActivator(inputChannel =
"fooAddressChannel")
    public AmqpOutboundEndpoint amqpOutbound(

AmqpTemplate amqpTemplate) {
        AmqpOutboundEndpoint outbound =
                        new
AmqpOutboundEndpoint(amqpTemplate);
        outbound.setExpectReply(true);
        outbound.setRoutingKey(“queueName”);
        return outbound;
    }

    @MessagingGateway(
                    defaultRequestChannel =
"fooAddressChannel")
    public interface AddressGateway {
        Message getAddress(Long userId);
    }
```

- Setup a configuration class and enable Spring Integration
- Create the queue to which we will send messages
- Listen for messages
- Create an AmqpOutboundEndpoint
- Specify that we expect a reply
- Specify the queue name as the routing key
- Create a messaging gateway to publish our request messages

Example: Reservation Service



Summary

AmqpInboundChannelAdapter

AmqpOutboundEndpoint

AmqpInboundGateway

Next up: ActiveMQ (JMS)

ActiveMQ (JMS)

Java Message Service (JMS)

The Java Message Service (JMS) API is a messaging standard that allows application components based on the Java Platform Enterprise Edition (Java EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

Differences Between AMQP and JMS

The main difference between AMQP and JMS is that AMQP is a binary protocol whereas JMS is a Java API

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

JmsMessageDrivenEndpoint

Outbound Channel Adapter

JmsSendingMessageHandler

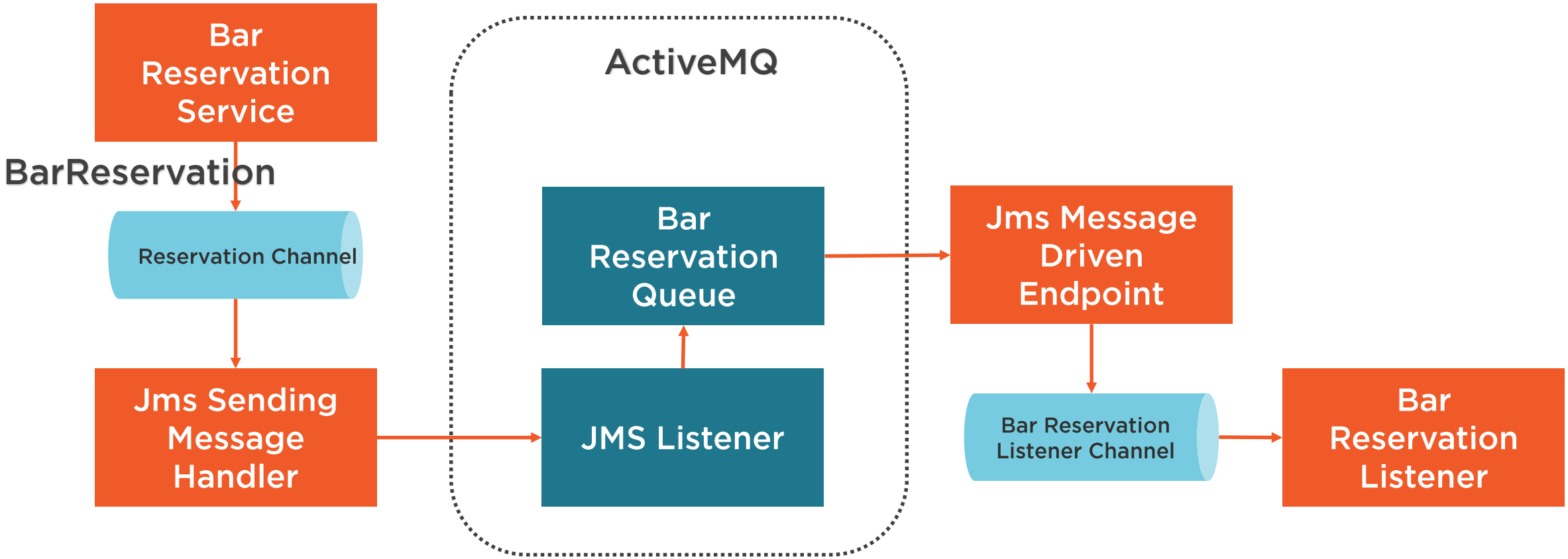
```
@Bean
public SimpleMessageListenerContainer
container(

ConnectionFactory cf) {
    SimpleMessageListenerContainer container =
                                new
SimpleMessageListenerContainer();

container.setConnectionFactory(connectionFacto
ry);

container.setDestinationName(DESTINATION_NAME)
;
    return container;
}

@Bean
public ChannelPublishingJmsMessageListener
listener() {
    return new
ChannelPublishingJmsMessageListener();
}

@Bean
public JmsMessageDrivenEndpoint endpoint(
```

- Create a SimpleMessageListenerContainer to subscribe to an ActiveMQ queue
- Specify the destination queue name
- Create a ChannelPublishingJmsMessageListener to convert JMS messages to Spring Integration messages
- Create a JmsMessageDrivenEndpoint
- Publish messages to our consumer channel

```
@Bean
public MessageChannel publishingChannel() {
    return new DirectChannel();
}

@Bean
@ServiceActivator(inputChannel =
"publishingChannel")
public MessageHandler jmsMessageHandler(

JmsTemplate jmsTemplate) {
    JmsSendingMessageHandler handler =
                    new
JmsSendingMessageHandler(jmsTemplate);

handler.setDestinationName(DESTINATION_NAME);
    return handler;
}

@MessagingGateway(
                defaultRequestChannel =
“publishingChannel")
public interface BarReservationGateway {
    void publishReservation(BarReservation
reservation);
}
```

- Create a channel to publish messages to ActiveMQ
- ServiceActivator to invoke this bean
- Create a JmsSendingMessageHandler
- Set the destination queue name
- Create a MessagingGateway

Example: Reservation Service



```
docker run -p 61616:61616 -p 8161:8161
rmohr/activemq
```

Running ActiveMQ in Docker

ActiveMQ Docker Image: rmohr/activemq

ActiveMQ DockerHub Link: https://hub.docker.com/r/rmohr/activemq

Management UI: http://localhost:8161/

ActiveMQ – Inbound and Outbound Gateways

Inbound and Outbound Gateways

Inbound Gateway

JmsInboundGateway

Outbound Gateway

JmsOutboundGateway

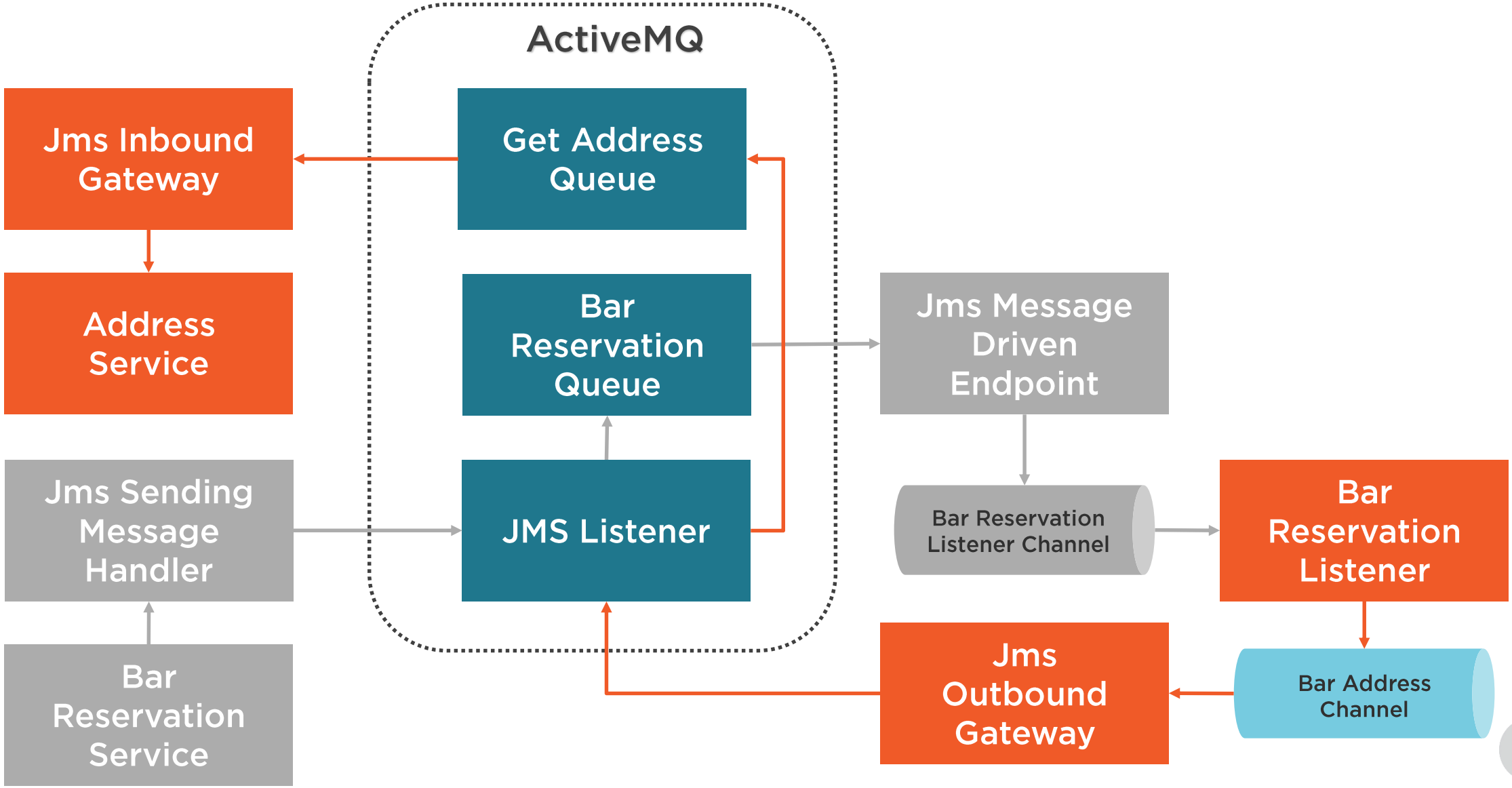
```
public SimpleMessageListenerContainer
container(

ConnectionFactory cf) {
 SimpleMessageListenerContainer container =
                                new
SimpleMessageListenerContainer();

container.setConnectionFactory(connectionFacto
ry);
    container.setDestinationName(QUEUE_NAME);
    return container;
}

public ChannelPublishingJmsMessageListener
listener() {
    ChannelPublishingJmsMessageListener
listener =
                          new
ChannelPublishingJmsMessageListener();
    listener.setExpectReply(true);
    return listener;
}

public JmsInboundGateway inboundGateway(
```

- Create a SimpleMessageListenerContainer to subscribe to an ActiveMQ queue
- Specify the destination queue name
- Create a ChannelPublishingJmsMessageListener to convert JMS messages to Spring Integration messages
- We expect a reply
- Create a JmsInboundGateway
- Messages will be handled by the request channel

```
@Bean
@ServiceActivator(inputChannel =
"getAddressChannel")
public JmsOutboundGateway outboundGateway(

ConnectionFactory connectionFactory) {
    JmsOutboundGateway gateway =

new JmsOutboundGateway();

gateway.setConnectionFactory(connectionFactory
);

gateway.setRequestDestinationName(“requestQueu
e”);

gateway.setReplyDestinationName(“replyQueue”);

gateway.setReplyChannel(getAddressReplyChannel
());
    return gateway;
}

@MessagingGateway(
       defaultRequestChannel =
"getAddressChannel",
```

- Create a JmsOutboundGateway
- Set the request and reply queue names
- Set the channel that will handle the reply message
- Create a messaging gateway to publish our request messages and specify the channel on which we will receive the response

Example: Reservation Service



Summary

JmsMessageDrivenEndpoint

JmsSendingMessageHandler

JmsInboundGateway

JmsOutboundGateway

Next up: Module Wrap-up

Conclusion

Channel Adapter

A Channel Adapter is a Message Endpoint that enables connecting a single sender or receiver to a Message Channel

Channel Adapter Types

Inbound Channel Adapter

One-way integration inbound

Outbound Channel Adapter

One-way integration outbound

Inbound Gateway

Bidirectional integration inbound

Outbound Gateway

Bidirectional integration outbound

Message Brokers

RabbitMQ

AMQP Integration

ActiveMQ

JMS Integration

Summary

You should understand channel adapters and gateways

You should understand how to integrate with RabbitMQ and ActiveMQ

You should understand the Spring Integration model for integrating with AMQP and JMS brokers

Next Module: Integrating with Apache Kafka