



Week 4: Service-to-Service Communication

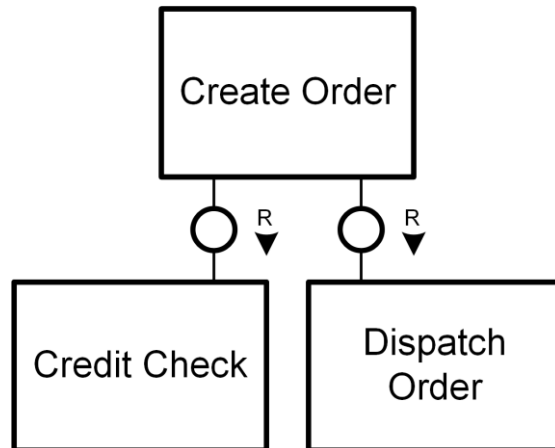
Unit 6: Messaging Using RabbitMQ – Part I

Messaging Using RabbitMQ – Part I

Service Composition Patterns – Orchestration vs. Choreography

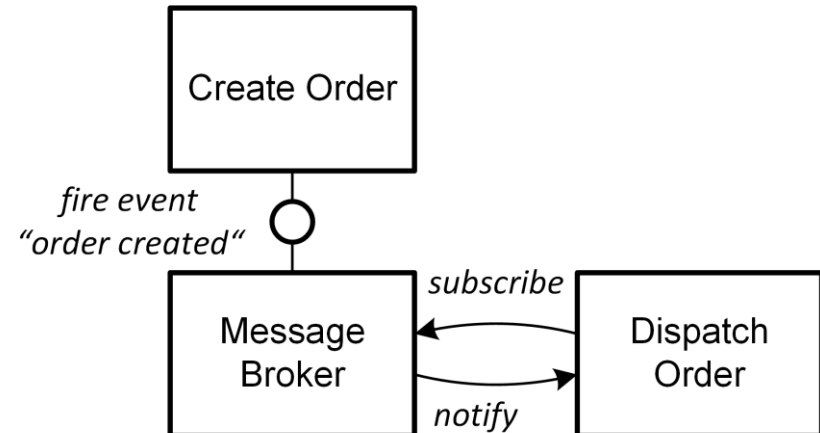
Orchestration (sync)

- Request & response
- Caller calls dep. Service directly
- Use when result is needed to complete operation



Choreography (async)

- Fire & forget, Publish-Subscribe
- Caller publishes event
- Use when multiple consumers want to react asynchronously to events



Message Brokers like **RabbitMQ** offer APIs to publish events and handle subscriptions allowing the consumers to be notified when an event arrives.

Messaging Using RabbitMQ – Part I

RabbitMQ – Message Broker

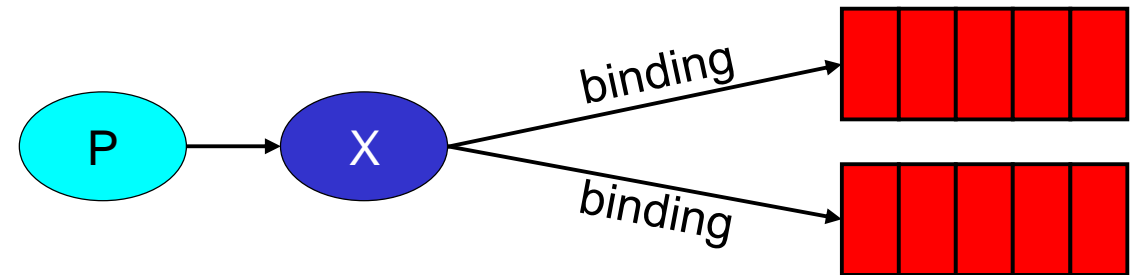
Important to gain high performant and resilient cloud applications!

Supports with AMQP message protocol different Service Qualities

- *fire-and-forget* (often non-durable)
- *at-least-once* and *exactly-once*
(message needs to be persisted till consumer sends acknowledgement back to RabbitMQ)

Queues are bound to “Exchanges” to support

- direct binding using *routing key*
- routing by *topic* e.g. “*.sap.*”
- broadcast message to multiple queues (*fanout*)



[RabbitMQ Tutorials](#)

Messaging Using RabbitMQ – Part I

Simple Message Broadcasting – Setup Exchange / Queue / Binding

Setup using AmqpAdmin

```
public class MessageProvider {  
    @Value("${ROUTING_KEY}")  
    String routingKey;  
  
    @Inject  
    public MessageProvider(AmqpAdmin amqpAdmin, RabbitTemplate rabbitTemplate) {  
        amqpAdmin.declareQueue(new Queue(routingKey)); // creates queue, if not there  
        this.rabbitTemplate = rabbitTemplate;  
    }  
    ...  
}
```

- Declare Queue and bind it directly to Default Exchange (*routing key*)
- Declare Queue on Provider and on Consumer side to make sure that it exists

Messaging Using RabbitMQ – Part I

Simple Message Broadcasting – Provider

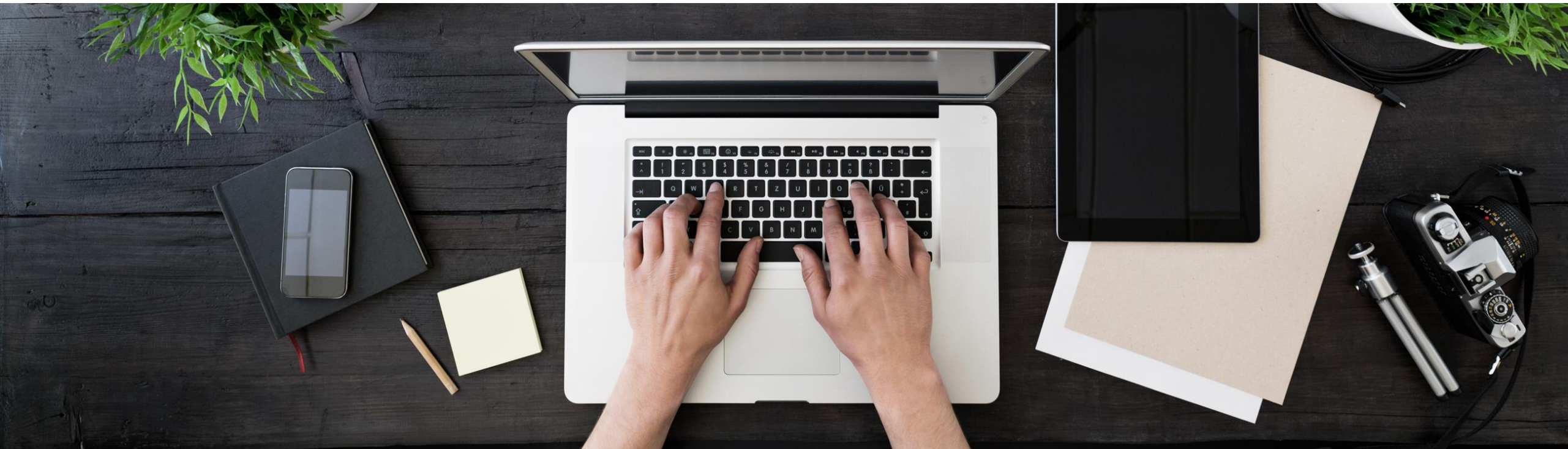
Send Message using RabbitTemplate

```
public void send(Object message) {  
  
    rabbitTemplate.convertAndSend(routingKey, message, new MessagePostProcessor() {  
        public Message postProcessMessage(Message message) {  
            String corrId = LogContext.getCorrelationId();  
            message.getMessageProperties().setCorrelationIdString(corrId);  
            return message;  
        }  
    });  
}
```

- Send Message using RabbitTemplate and SimpleMessageConverter
- Specify properties such as Correlation-Id using MessagePostProcessor
- Use RabbitMQ Admin Console for monitoring message queues (port 15672)

Messaging Using RabbitMQ – Part I

Demo: send message into message queue using Spring AMQP

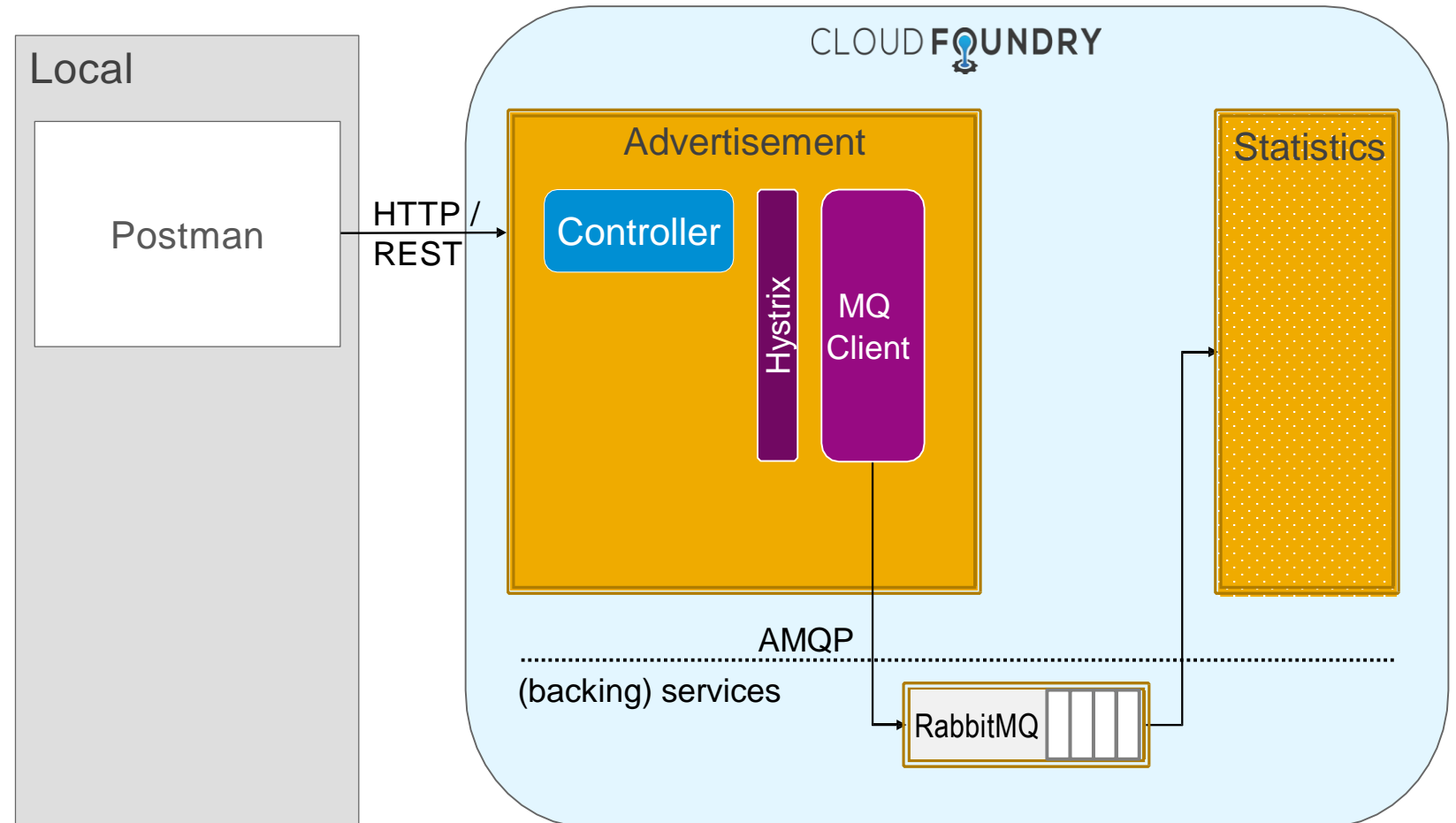


Messaging Using RabbitMQ – Part I

Exercise 20



[Optional] Exercise 20: Use Message Queues



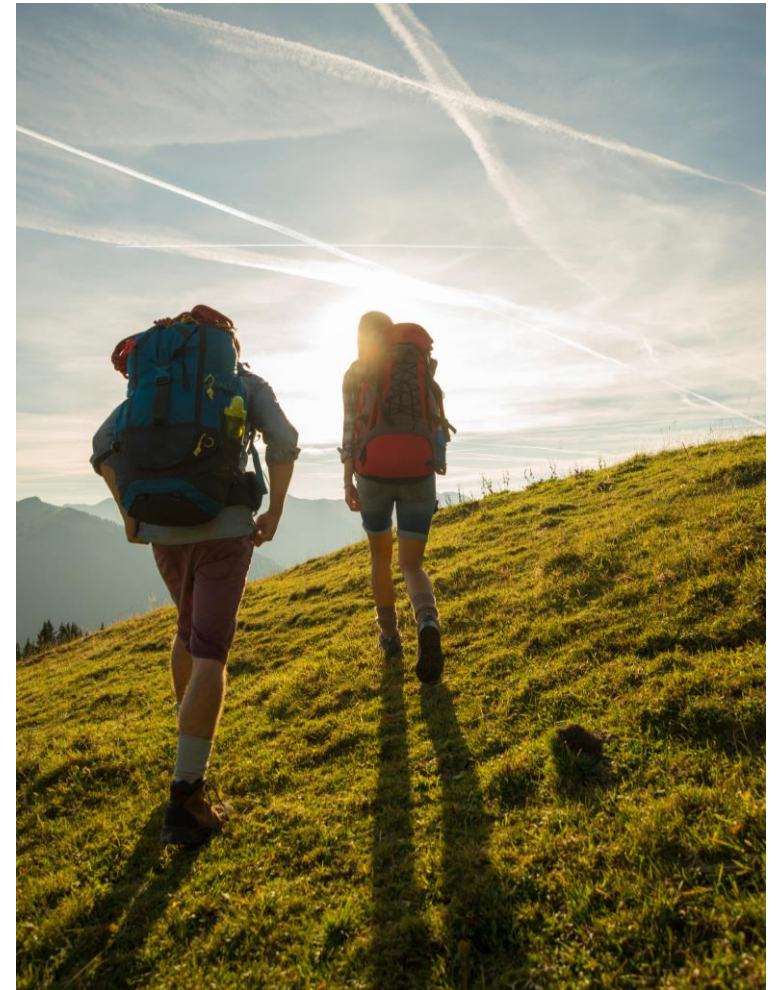
Messaging Using RabbitMQ – Part I

Further reading

- [RabbitMQ Tutorials](#)
- [Spring AMQP](#)



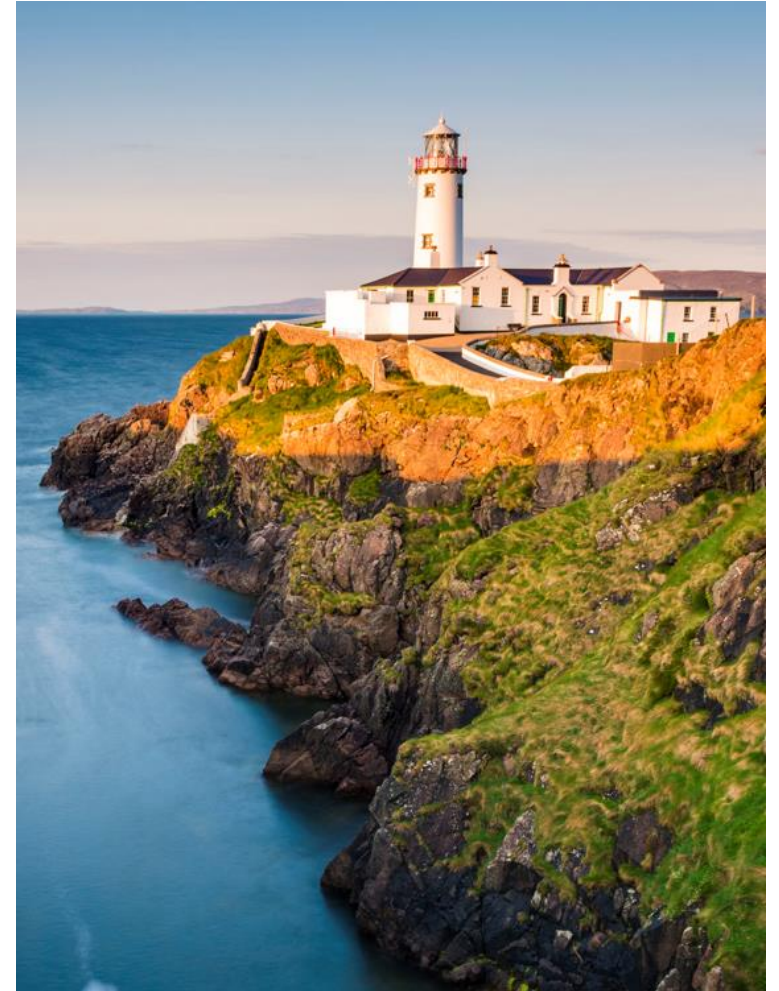
Additional Material



Messaging Using RabbitMQ – Part I

What you've learned in this unit

- What Service Composition Patterns there are
 - Orchestration
 - Choreography
- RabbitMQ
 - What it is
 - Why we use it
 - How to implement a message Provider using *Spring AMQP*



Thank you.

Contact information:

open@sap.com

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

See <http://global.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.