Week 4: Service-to-Service Communication

Unit 6: Messaging Using RabbitMQ – Part I

SAP Run Simple

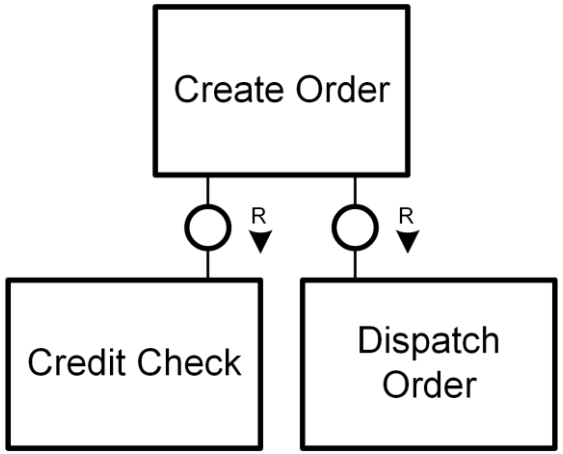
openSAP
open.sap.com

Messaging Using RabbitMQ – Part I

Service Composition Patterns – Orchestration vs. Choreography

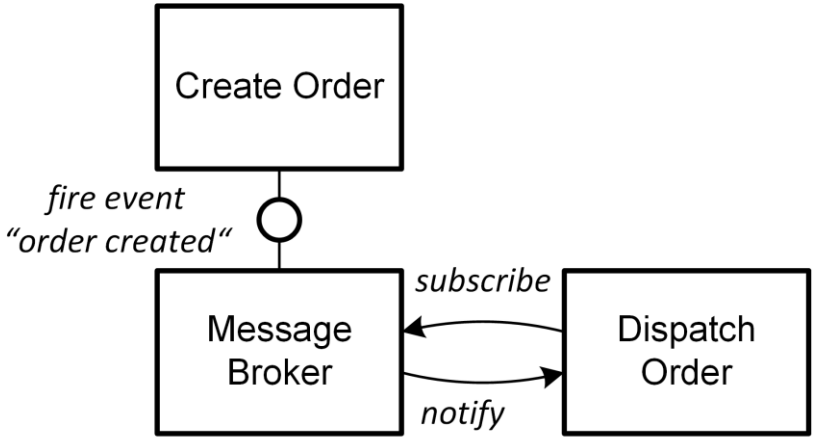
Orchestration (sync)

- 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

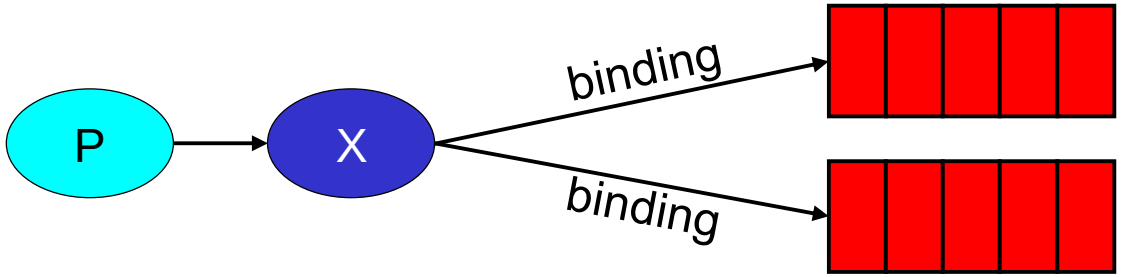
Important to gain high performant and resilient cloud applications!

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

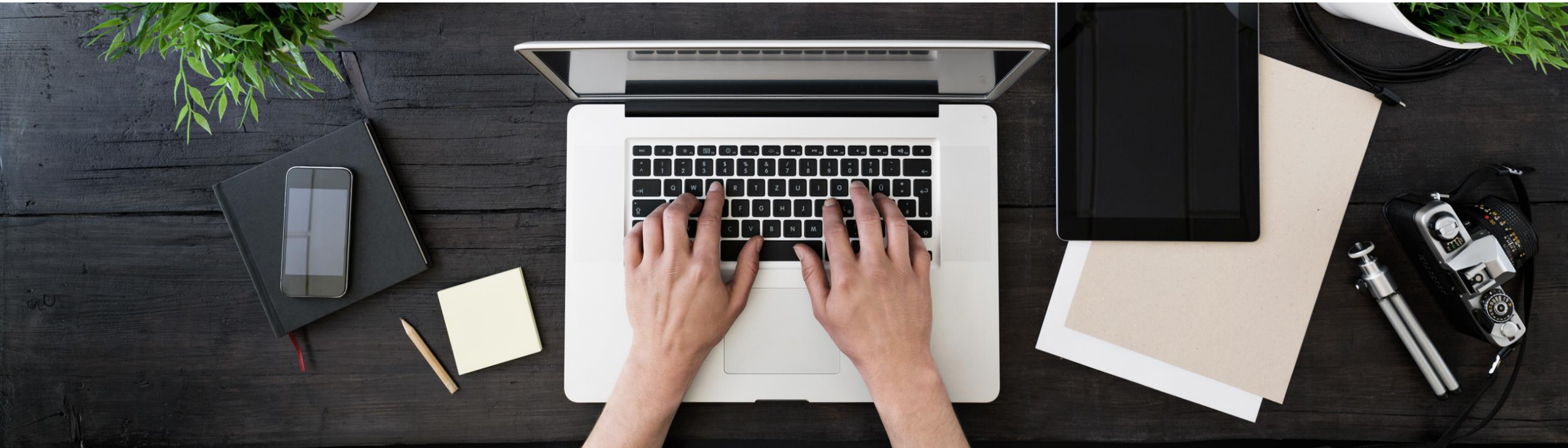
```
public void send(Object message) {

    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

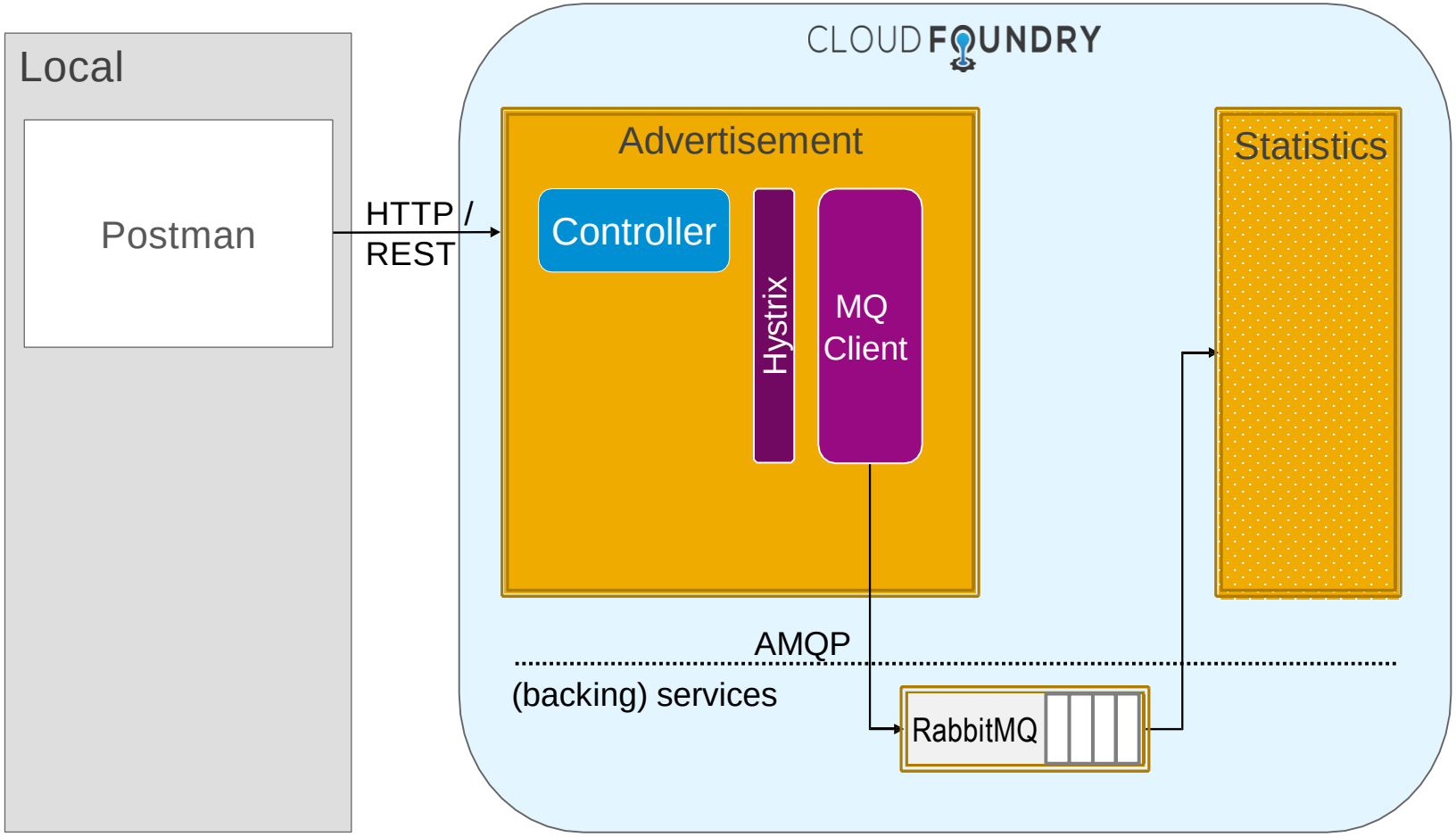
Demo: send message into message queue using Spring AMQP



Messaging Using RabbitMQ – Part I

Exercise 20

[Optional] Exercise 20: Use Message Queues



Messaging Using RabbitMQ – Part I

Further reading

Additional Material

- RabbitMQ Tutorials
- Spring AMQP

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