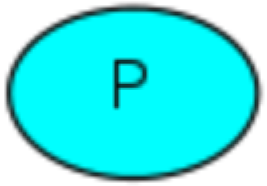
Agenda

1. What is rabbitmq introduction
2. Using rabbitmq with docker
3. AMQP
4. Messaging models

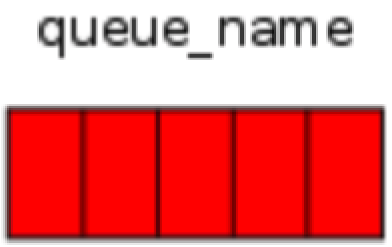
Introduction Rabbitmq

1. RabbitMQ is a message broker: it accepts and forwards messages
2. Analogy: post office:
   - RabbitMQ is a post box, a post office and a postman
3. Messages are binary blobs of data

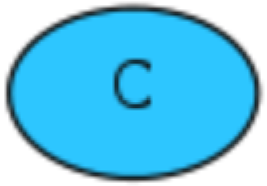
Messaging Jargon



- producer



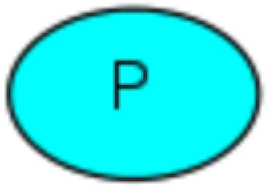
- queue



- consumer

Producer

- Producing means sending
- A program that sends messages is a producer

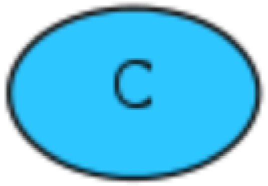


Queue

- Essentially a large message buffer, analogous to a post box which lives inside RabbitMQ
- Bound by the host's memory & disk limits
- Messages flow through RabbitMQ and your applications
  - But they can only be stored inside a queue
- Many producers can send messages to one queue
- Many consumers can try to receive data from one queue

Consumer

- has a similar meaning to receiver
- is a program that mostly waits to receive messages



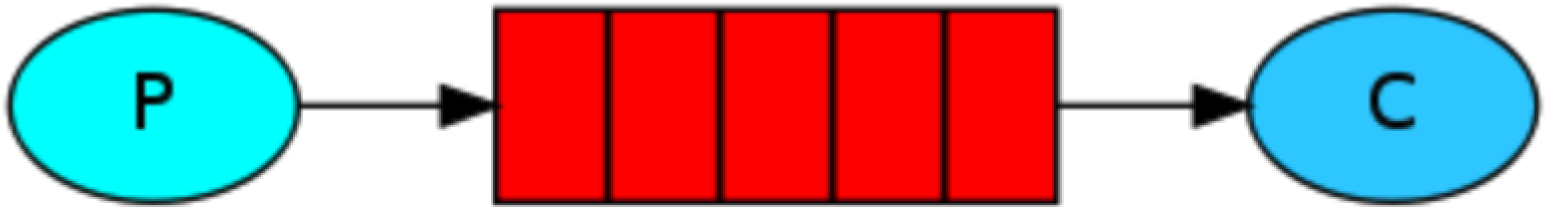
- producer, consumer, and broker usually are on distinct hosts

Hello world of messaging

- Exchange messages via a queue
- Use springboot
- start.spring.io
- SELECT AMQP dependency

What to achieve

- P represents the producer and C the consumer
- The box in the middle is a queue
- P and C will both be implemented as a springBoot application



Docker to the rescue

1. Use rabbitmq docker image
2. Issue:
    1. docker run -d --hostname my-rabbit --name some-rabbit -p 5672:5672 -p 15672:15672 rabbitmq:3-management
3. Connect to the broker:
    1. at port 5672
    2. management console at port 15672

Building the consumer

- a.k.a. receiver

```
@RabbitListener(queues = "hello")
public class Demo1Receiver {

    @RabbitHandler
    public void receive(String in) {
        System.out.println(" [x] Received '" + in + "'");
    }
}
```

Building consumer part 2

- necessary configuration

```
@Configuration
public class Demo1ConsumerConfig {

    @Bean
    public Queue hello() {
        return new Queue("hello");
    }

    @Bean
    public Demo1Receiver receiver() {
        return new Demo1Receiver();
    }

}
```

Setup nuts and bolts

- create a SpringBoot entry point:

```
@SpringBootApplication
public class RabbitAmqpApplication {

    @Bean
    public CommandLineRunner tutorial() {
        return new RabbitAmqpRunner();
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(RabbitAmqpApplication.class, args);
    }
}
```

Setup nuts and bolts part 2

- configuration of project in src/main/resources: application.yml

```
server:
  port: 8000

spring:
  rabbitmq:
    host: 127.0.0.1
    port: 5672
    username: guest
    password: guest

logging:
  level:
    org:  FINE
```

Building the Producer

- The Producer/Sender details:

```
public class Demo1Sender {

    @Autowired
    private RabbitTemplate template;

    @Autowired
    private Queue queue;

    @Scheduled(fixedDelay = 1000, initialDelay = 500)
    public void send() {
        String message = "Hello World!";
        this.template.convertAndSend(queue.getName(), message);
        System.out.println(" [x] Sent '" + message + "'");
    }
}
```

Building the Producer part 2

- Configure the producer part

```
@Configuration
public class Demo1ProducerConfig {

    @Bean
    public Queue hello() {
        return new Queue("hello");
    }

    @Bean
    public Demo1Sender sender() {
        return new Demo1Sender();
    }
}
```

Nuts and bolts

- The nuts and bolts are identical to the consumer the configuration is:

```
server:
  port: 8080

spring:
  rabbitmq:
    host: 127.0.0.1
    port: 5672
    username: guest
    password: guest

logging:
  level:
    org:  FINE
```

The important pieces

- The sender uses the RabbitTemplate

```
@Autowired
private RabbitTemplate template;

@Autowired
private Queue queue;

@Scheduled(fixedDelay = 1000, initialDelay = 500)
public void send() {
    ...
    this.template.convertAndSend(queue.getName(), "my message");
    ...
}
```

The important pieces part 2

- The receiver/consumer

```
@RabbitListener(queues = "hello")
public class Demo1Receiver {

    @RabbitHandler
    public void receive(String in) {
        System.out.println(" [x] Received '" + in + "'");
    }
}
```

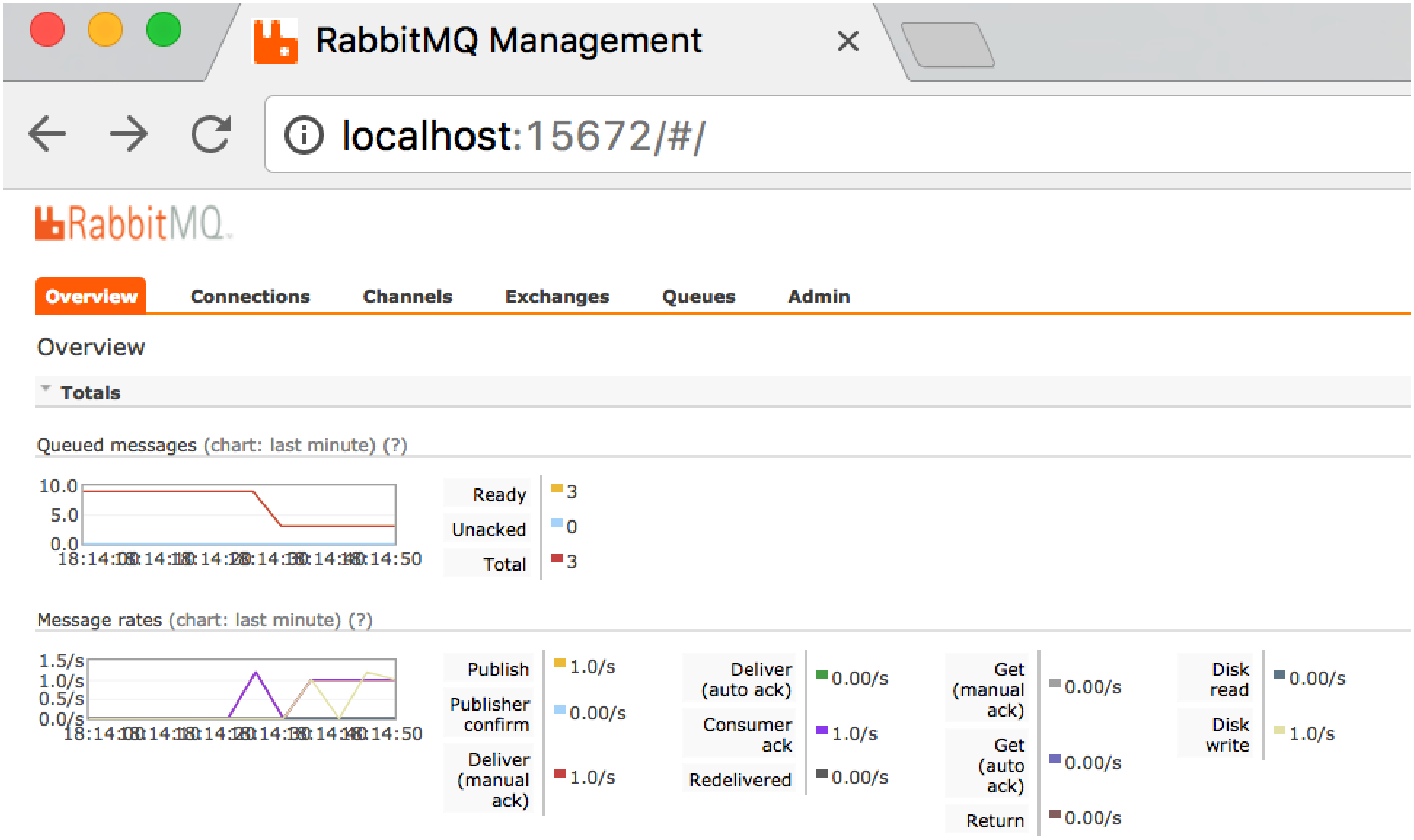
Running the 2 applications

```
Tomcat1 started on port(s): 8080 (http)
 [x] Sent 'Hello World!'
 [x] Sent 'Hello World!'

 Tomcat2 started on port(s): 8000 (http)
 [x] Received 'Hello World!'
 [x] Received 'Hello World!'
```

rabbitmanagement

- Admin webconsole to look at rabbitmq



Overview

Totals

Queued messages (chart: last minute) (?)

Ready	🟨 3
Unacked	🟦 0
Total	🟥 3

Message rates (chart: last minute) (?)

Publish	🟨 1.0/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s	Disk read	⬛ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 1.0/s	Get (auto ack)	🟪 0.00/s	Disk write	🟨 1.0/s
Deliver (manual ack)	🟥 1.0/s	Redelivered	⬛ 0.00/s	Return	🟫 0.00/s		