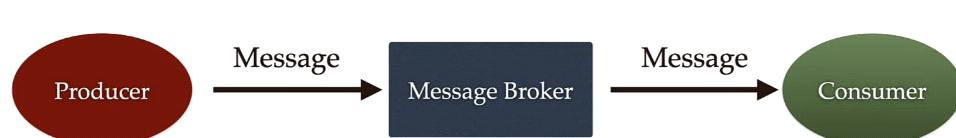
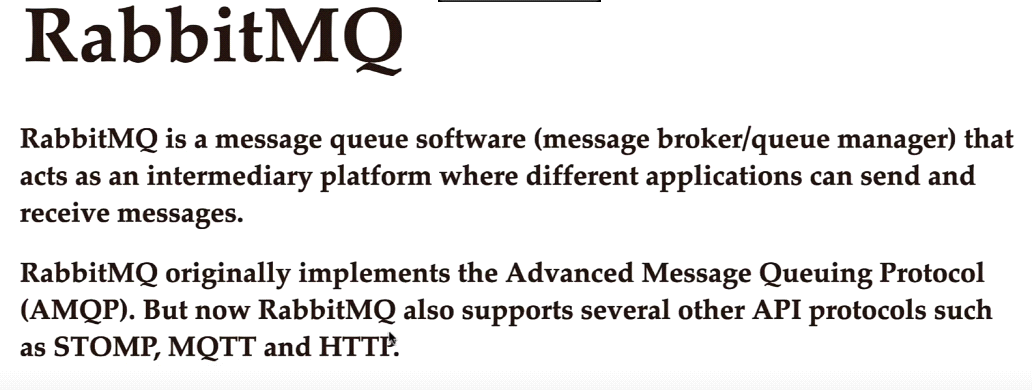
* RabbitMQ poate fi folosit in proeiect cu ajutorul lui Spring AMQP
* Spring AMQP - Spring Advanced Message Queuing Protocol

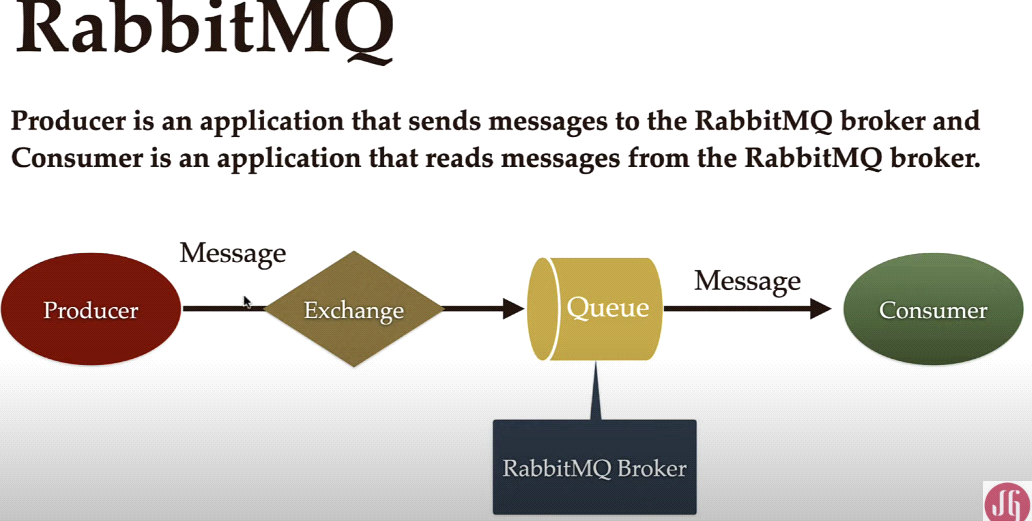
**Message Queue**

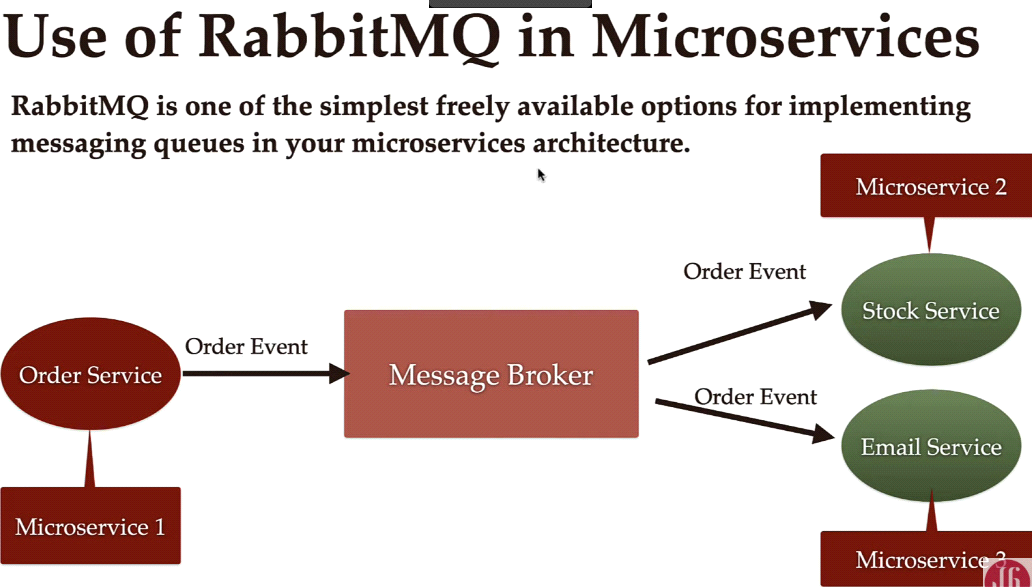
* Message queue permite aplicatiilor sa comunice intre ele prin a trimite mesaje una la alta. Acest message queue ofera temporary message storage cand destination program este ocupat sau neconectat
* Un message queue este creat din producer, broker(message queue sofware) si consumer
* Un message queue ofera comunicare asyncronizata intre aplicatii

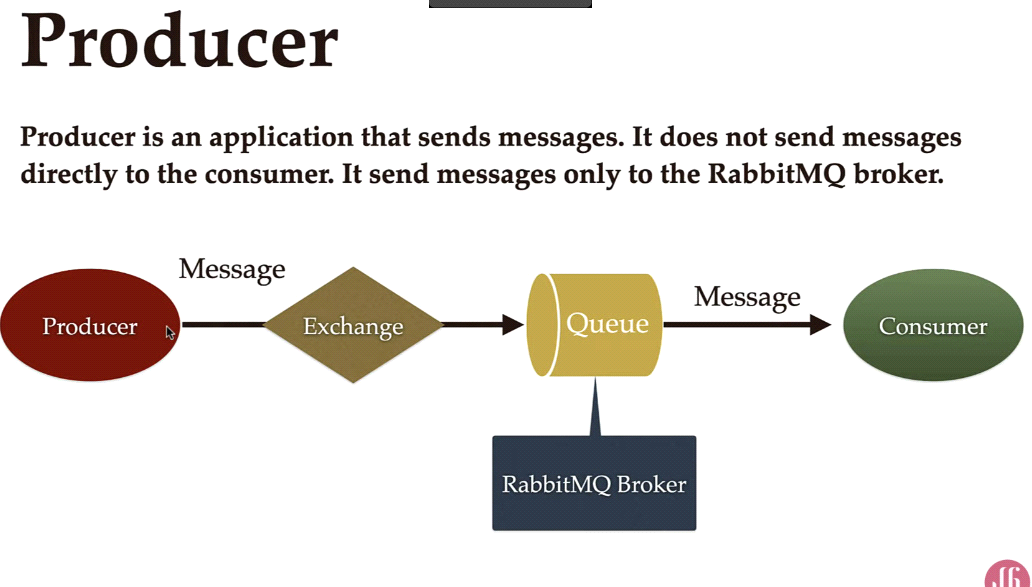


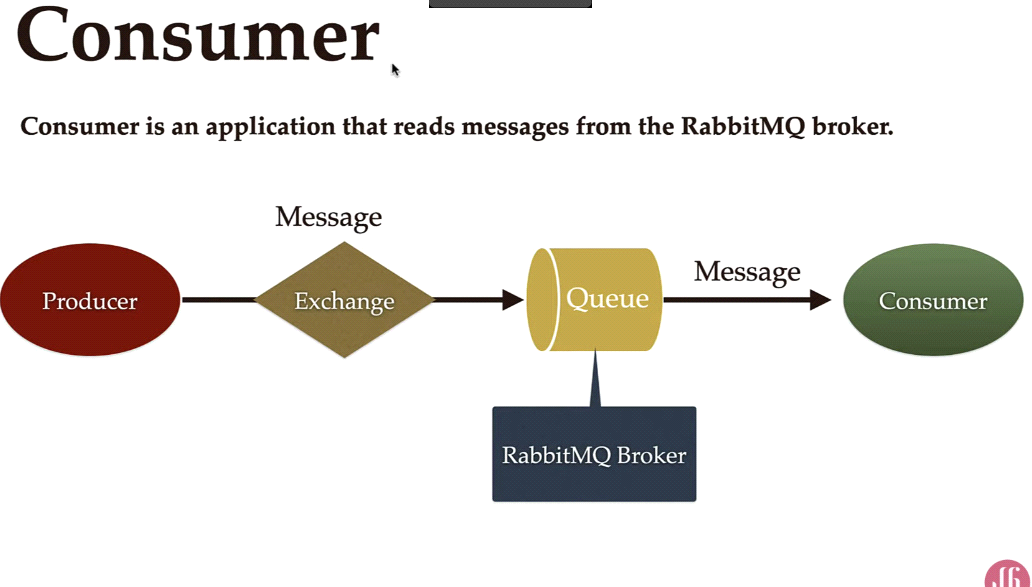
**RabbitMQ**

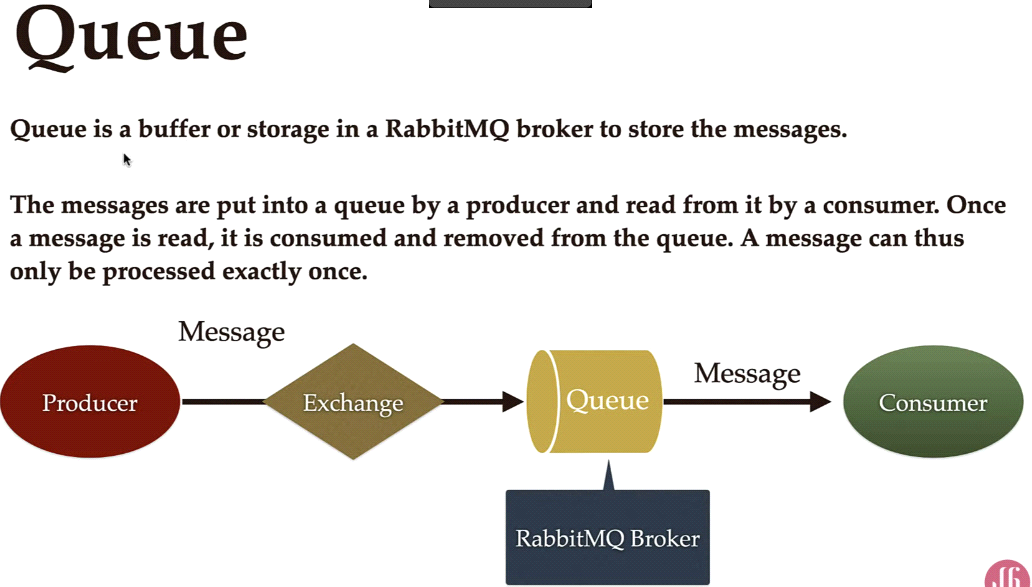


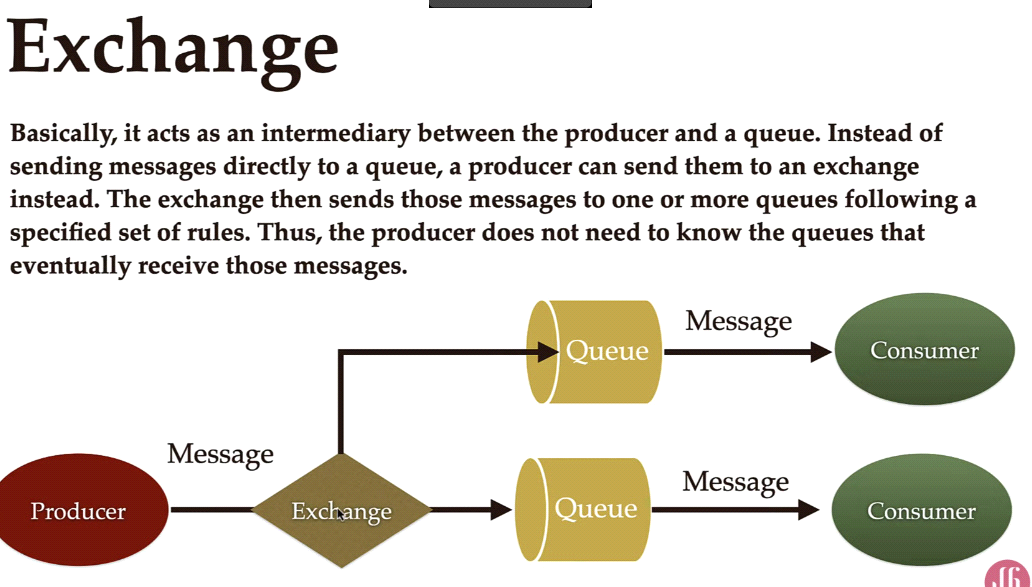




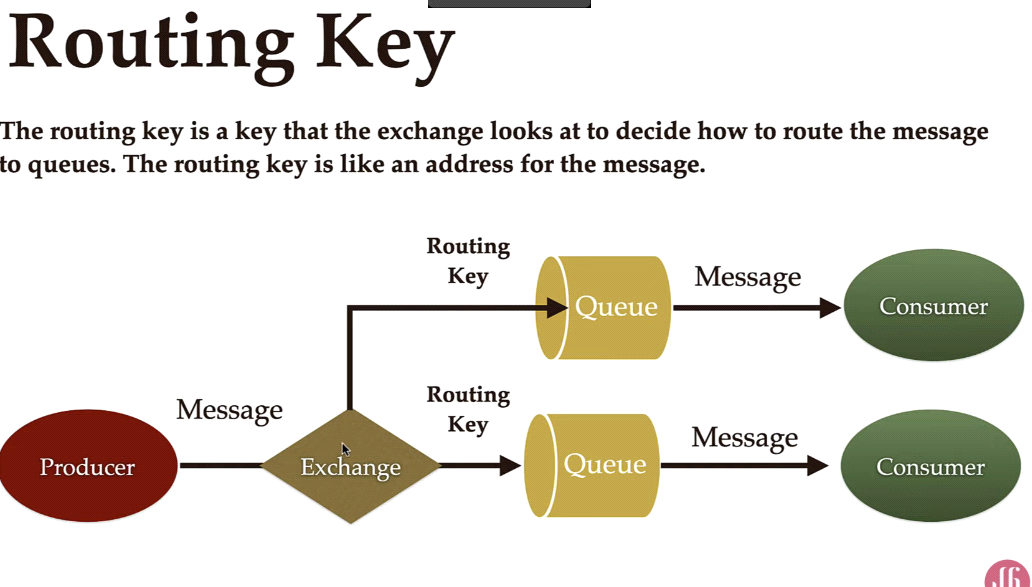




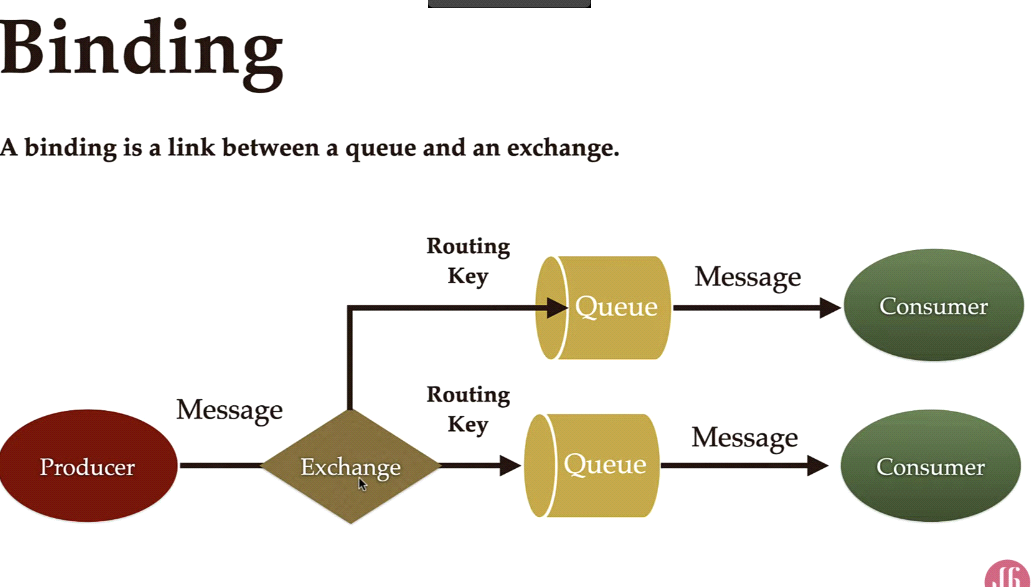




Exchange e cel care asigura transportul mesajului de la Producer la Queue, caci Producer nu stie inca la ce queue urmeaza sa fie trimis mesajul lui, si exchange se va ocupa de asta



Cand Producer trimite un mesaj la Exchange, Exchange inca nu stie la care queue el trebuie trimis. Ecxhange foloseste routing key pentru a scrie mesajul in queue respectiv, si routing queue e trimis tot de producer catre exchange, caci exchange inca nu poate face conexiunea fara routing key



Iar acest binding e facut de Routing Queue

De ex, producer vrea sa trimita un mesaj la Queue2. El va trebui sa trimita mesajul la exchange si routing key pentru queue si exchange va folosi acest routing key ca sa identifice queue2, sa faca conexiunea si sa trimita mesajul, deci va face conexiunea cu queue cerut, adica queue2

**Configuratie**

* Instalam in docker imaginea oficiala, anume 3.12.0-management
* port din web: 15672
* port pentru conexiuni: 5672
* default login and password: guest guest
* Putem folosi si:

**docker run -it -p 5672:5672 -p 15672:15672 -e RABBITMQ\_DEFAULT\_USER=guest -e RABBITMQ\_DEFAULT\_PASS=guest rabbitmq:3-management**

**si ne conectam cu localhost:15672**

**Web interface**

* Connection - conexiunile facute de aplicatiile noastre cu rabbitmq
* Channels - aici apar mesajele intre producer si consumer
* Exchanges - e important sa cream un exchange

Un exchange trebuie legat de un queue

1. Cream un exchange

2. cream un queue Classic

3. Acum, ne ducem la exchange creat si il legam de queue creat

La routing punem ce nume vrem

Exchangee trimite mesajele la Queue, si asta o putem face manual si din exchange din web

4. La queue dam Get Messages si vedem mesajele venite

**Spring Boot**

* Spring AMQP este doar o colectie de interfete ce trebuie implementate. Spring RabbitMQ o implementeaza. Deci Spring AMQP este un API pentru sytemele ce folosesc AMQP protocol
* Ne trebuie dependenta:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-amqp</artifactId>  
</dependency>

* Acum, configuram rabbitmq:

spring.rabbitmq.host = localhost  
spring.rabbitmq.port = 5672  
spring.rabbitmq.username = guest  
spring.rabbitmq.password = guest

* Acum, ne va trebui sa cream beanuri pentru rabbitmq queue:

Dependentele le luam din org.springframework.amqp.core.

* Acum cream un bean si pentru exchange:

@Bean  
public Queue queue(){  
 return new Queue("queue\_demo");  
}  
  
@Bean  
public TopicExchange topicExchange(){  
 return new TopicExchange("exchange\_demo");  
}

* Trebuie sa facem binding la acest queue si exchange folosind routing key setat:

@Bean  
public Binding binding(){  
 return BindingBuilder.*bind*(queue())  
 .to(topicExchange())  
 .with("routing\_key\_demo");  
}

Avem grija sa fie numele identic cu cele setate in Web, caci aici facem un fel de mapare

* In mod normal ne-ar mai trebui beanuri de tip ConnectionFactory, RabbitTemplate, RabbitAdmin, dar Spring Boot deja le configureaza automat

**Trimiterea si primirea mesajelor**

* RabbitTemplate este beanul care ne ofera posibilitatea de a primit si trimite mesaje rapid. El e creat automat de Spring Boot
* Acum putem crea un bean care sa trimita mesajele, gen un @Service

@RequiredArgsConstructor  
@Service  
public class RabbitMqService {  
 private RabbitTemplate rabbitTemplate;  
  
 public void sendMessage(String message){  
 rabbitTemplate.convertAndSend("exchange\_demo","routing\_key\_demo","message");  
  
 System.*out*.println("Message send to queue\_demo");  
 }  
  
}

**convertAndSend("exchange name","routing key name","message")** - mesajul e convertit automat in Message si e trimis la exchange cu routing key.

Acum facem res API:

@RestController  
@RequiredArgsConstructor  
public class Rest {  
 private final RabbitMqService rabbitMqService;  
  
 @GetMapping("/api/{message}")  
 public String get(@PathVariable("message")String msg){  
 rabbitMqService.sendMessage(msg);  
  
 return "Message send to rabbitmq";  
 }  
  
}

**Atentie! Beanurile la Queue, exchange si bind se fac doar cand vrem sa cream un queue, exchenger si routing key dintre ele nou. Daca ele deja exista, nu mai trebuie beanuri.**

**Create Consumer**

@Service  
public class RabbitMqConsumer {  
  
 @RabbitListener(queues = {"queue\_demo"})  
 public void listener(String message){  
 System.*out*.println("Message recceived: "+message);  
 }  
}

Punem doar @RabbitListener cu numele la queues si gata

**JSON Message Configuration**

Un exchange poate avea diferite routing keys, deci un exchange poate crea legaturi cu mai multe queues

* Cream o noua routing key pentru json
* Cream un RabbitTemplate si setam un MessageConvertor in JSON:

@Bean  
public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory){  
 RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);  
 rabbitTemplate.setMessageConverter(new Jackson2JsonMessageConverter());  
   
 return rabbitTemplate;  
}

Acum, modificam metoda de a trimite mesajul, sa trimita un user, desi el tot v-a fi trimis ca String, doar ca scris ca JSON

@RequiredArgsConstructor  
@Service  
public class RabbitMqService {  
 private final RabbitTemplate rabbitTemplate;  
  
 public void sendMessage(User user){  
 rabbitTemplate.convertAndSend("exchange\_demo","routing\_key\_demo\_json",user);  
  
 System.*out*.println("User sent to queue");  
 }  
  
}

Si modificam putin Controller:

@GetMapping("/api/{message}")  
public String get(@PathVariable("message")String msg){  
 User user = new User(1,"Test","User");  
 rabbitMqService.sendMessage(user);  
  
 return "Message send to rabbitmq";  
}

Si, daca scoatem acest listener, vedem ca s-a inregistrat useru in Rabbit

* Acum, pentru a primi json si a-l converti in object:

@RabbitListener(queues = {"queue\_demo"})  
public void listener(User user){  
 System.*out*.println("User recceived: "+user);  
}

Atentie! RabbitMQ va converti singur din JSON in User, dar el are nevoie de un Bean de tip Jackson2JsonMessageConverter

Noi l-am creat direct in RabbitTemplate bean, dar cand va deserializa, el va cauta asa bean si daca nu il gaseste, avem exceptie,de aceea cream beanul:

@Bean  
public MessageConverter messageConverter(){  
 return new Jackson2JsonMessageConverter();  
}