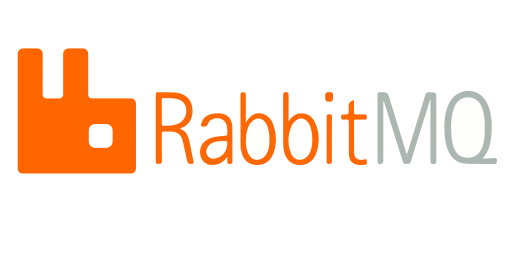
TP :

RabbitMQ (Installation + Interface

+ 2 Mini Projets)



**Table des matières**

[**1)**](#_heading=h.1fob9te) **Installation**  3

[**2)**](#_heading=h.3znysh7) **Manipulation (Interface)**  5

[**3)**](#_heading=h.2et92p0) **Manipulation (Micro services springboot-Messagerie)**  11

[∙](#_heading=h.tyjcwt) **spring-rabbitmq-producer** 11

[∙](#_heading=h.3dy6vkm) **spring-rabbitmq-cosomer** 15

[∙](#_heading=h.1t3h5sf) **Test de la communication** 17

[**4)**](#_heading=h.4d34og8) **Manipulation (Micro services springboot-MySQL)**  18

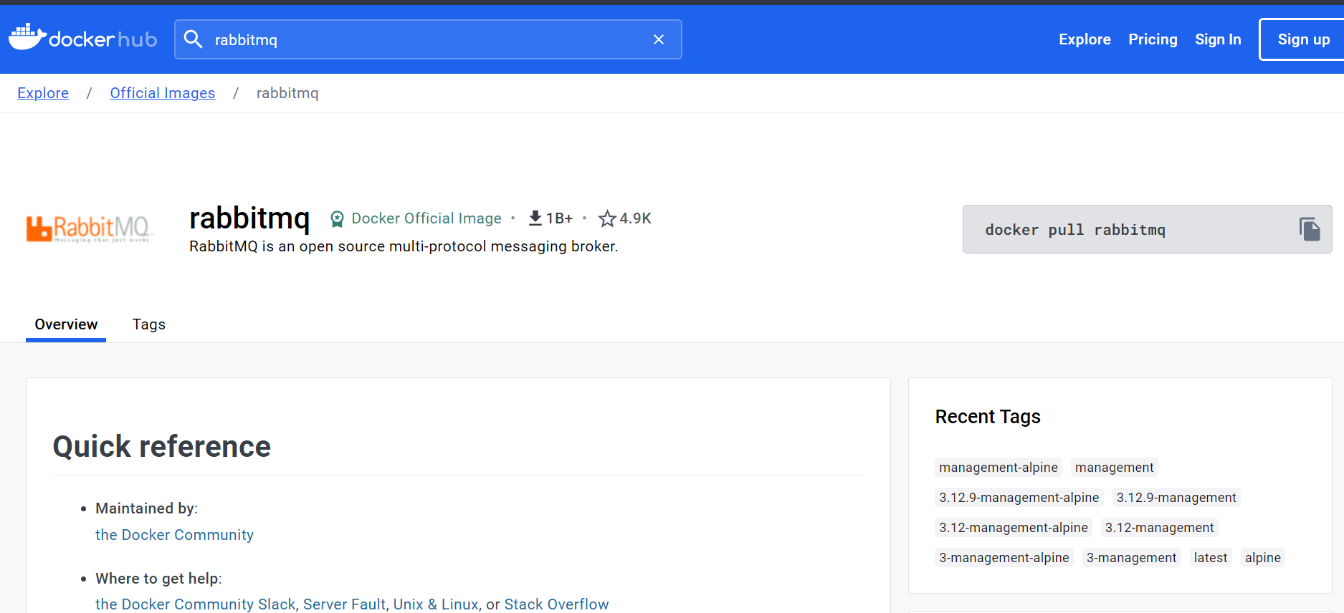
[∙](#_heading=h.2s8eyo1) **spring-rabbitmq-producer** 18

[∙](#_heading=h.17dp8vu) **spring-rabbitmq-cosomer** 22

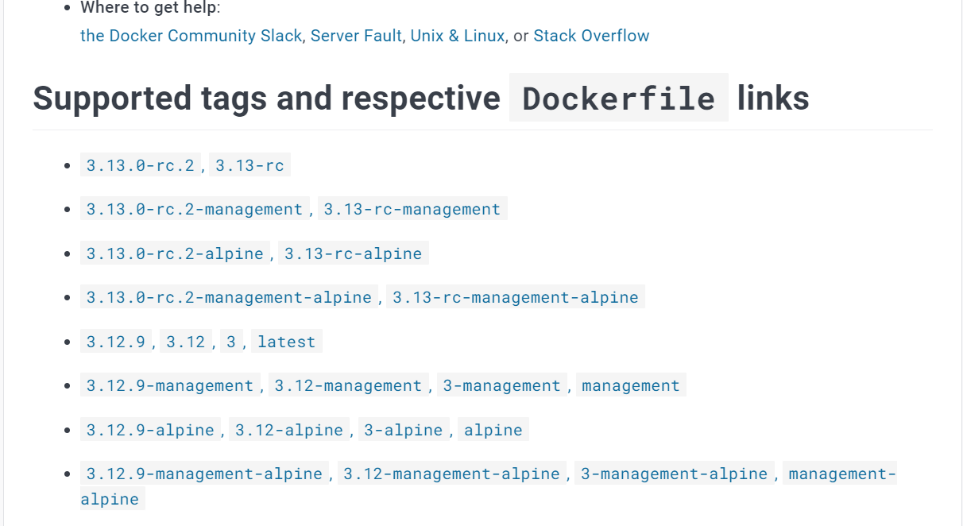
[∙](#_heading=h.3rdcrjn) **Test de la communication** 26

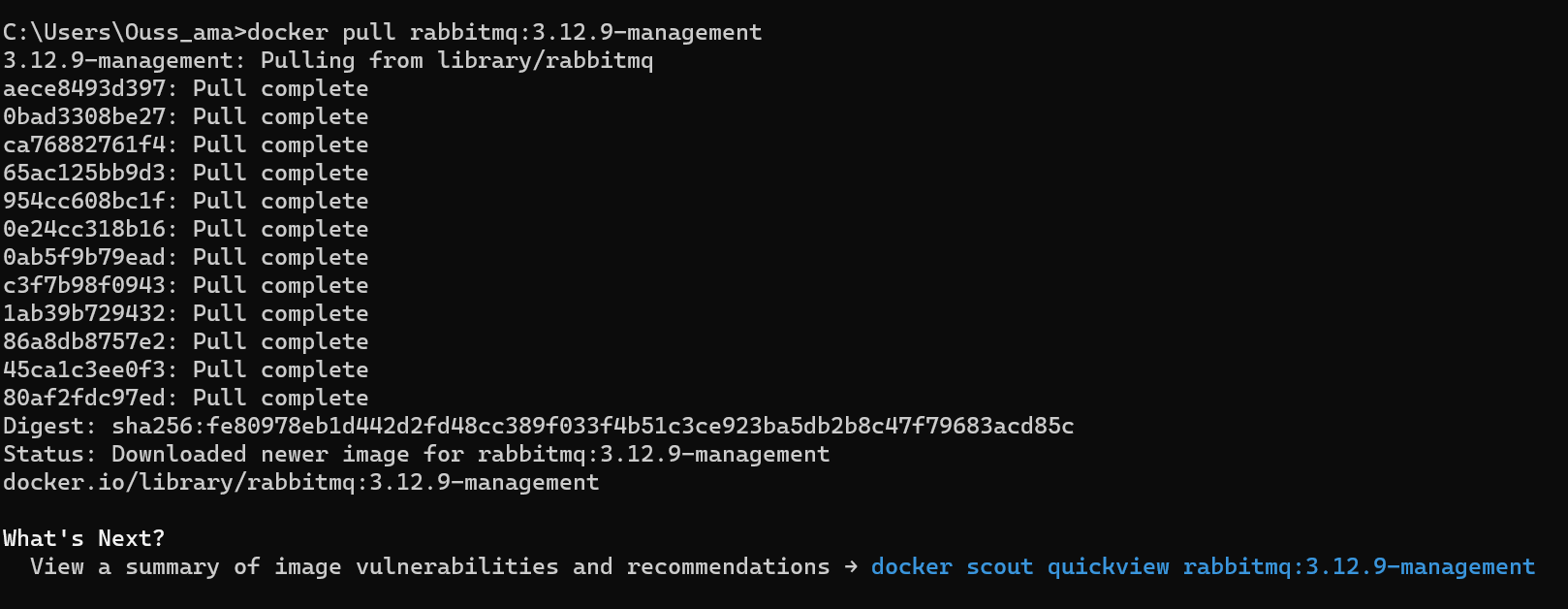
# **Installation**

* Installation de l’image docker

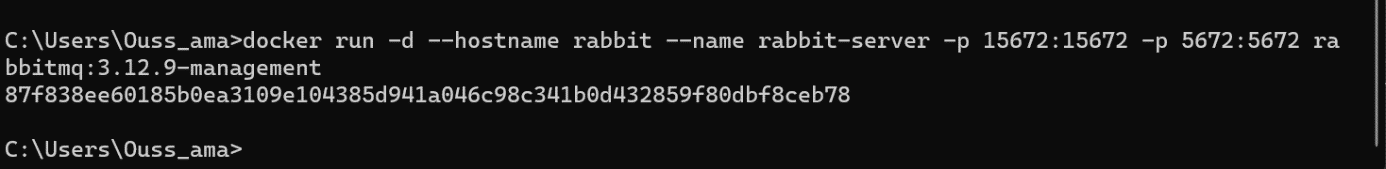


* On va travailler avec la version stable : 3.12.9-management

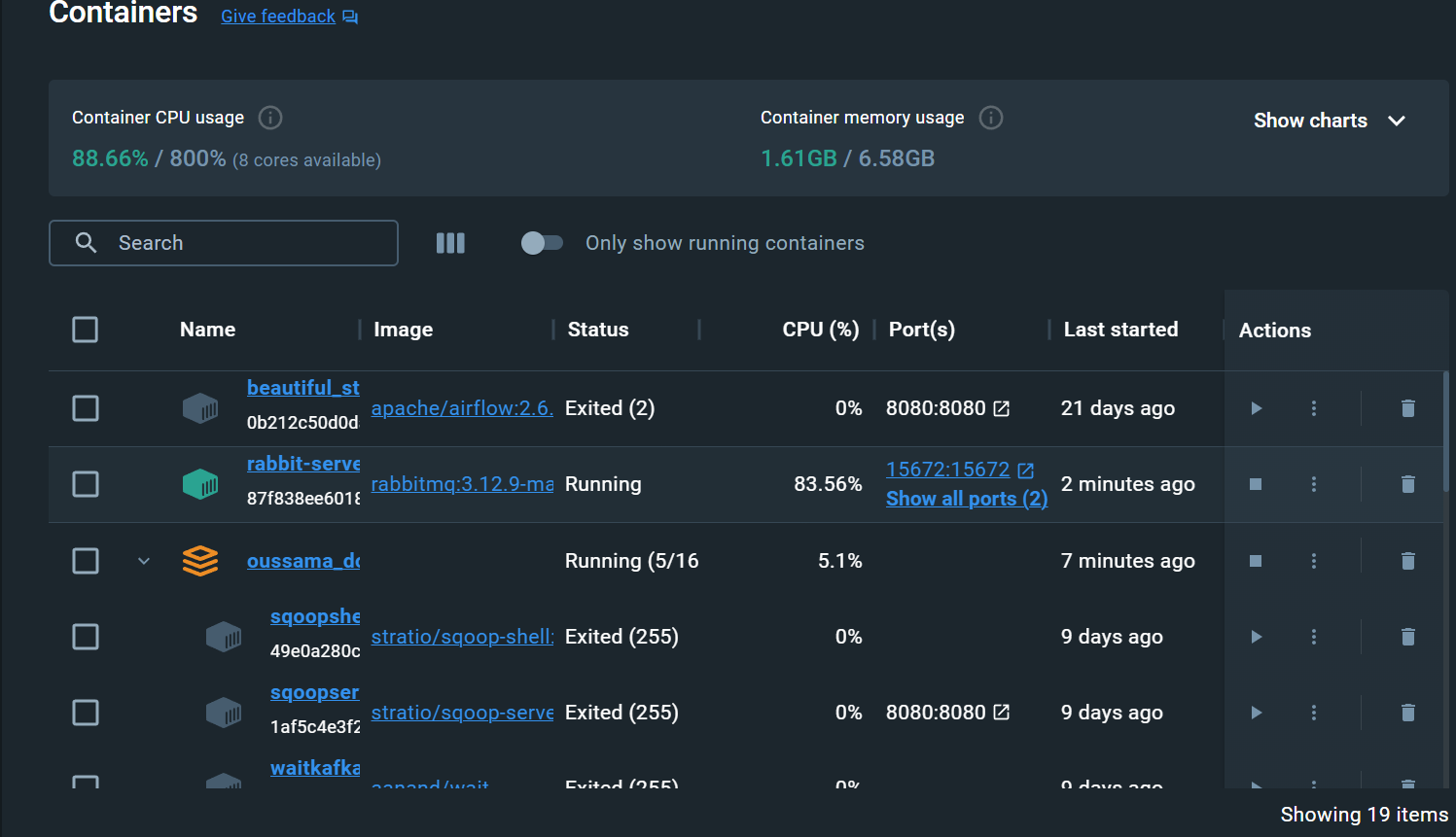




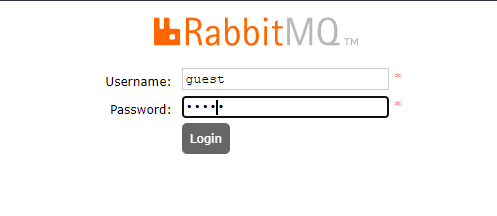
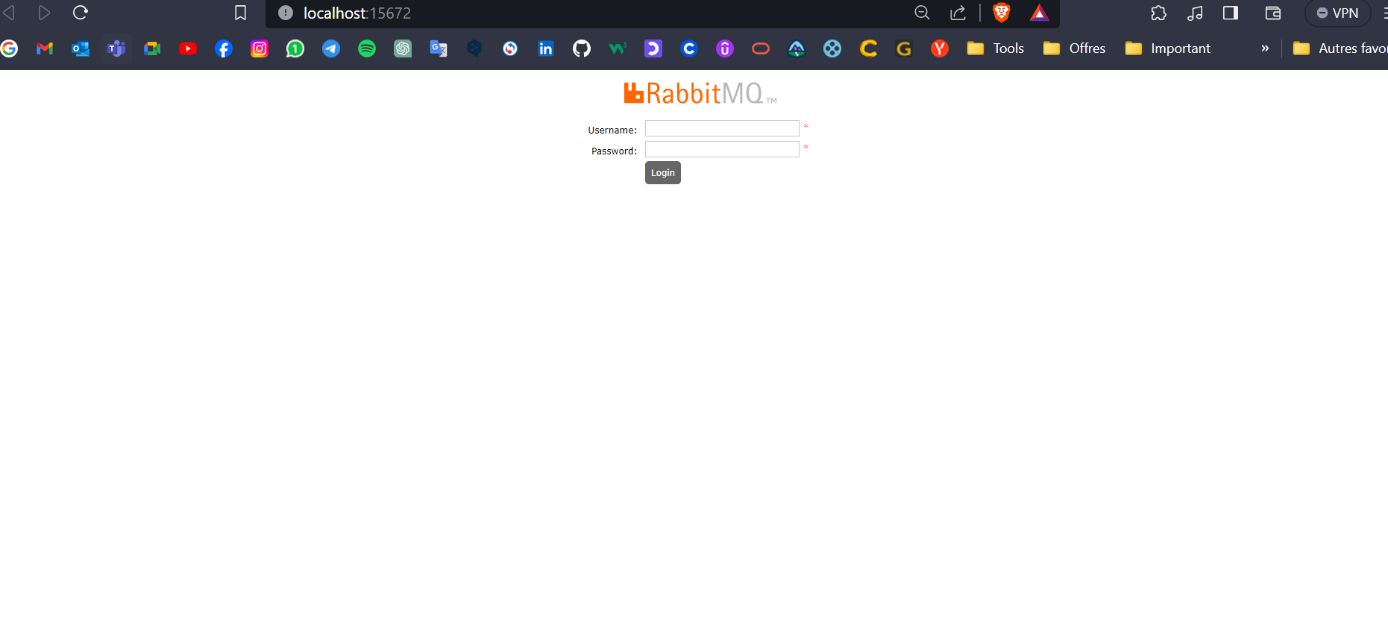
* L’exécution de l’image

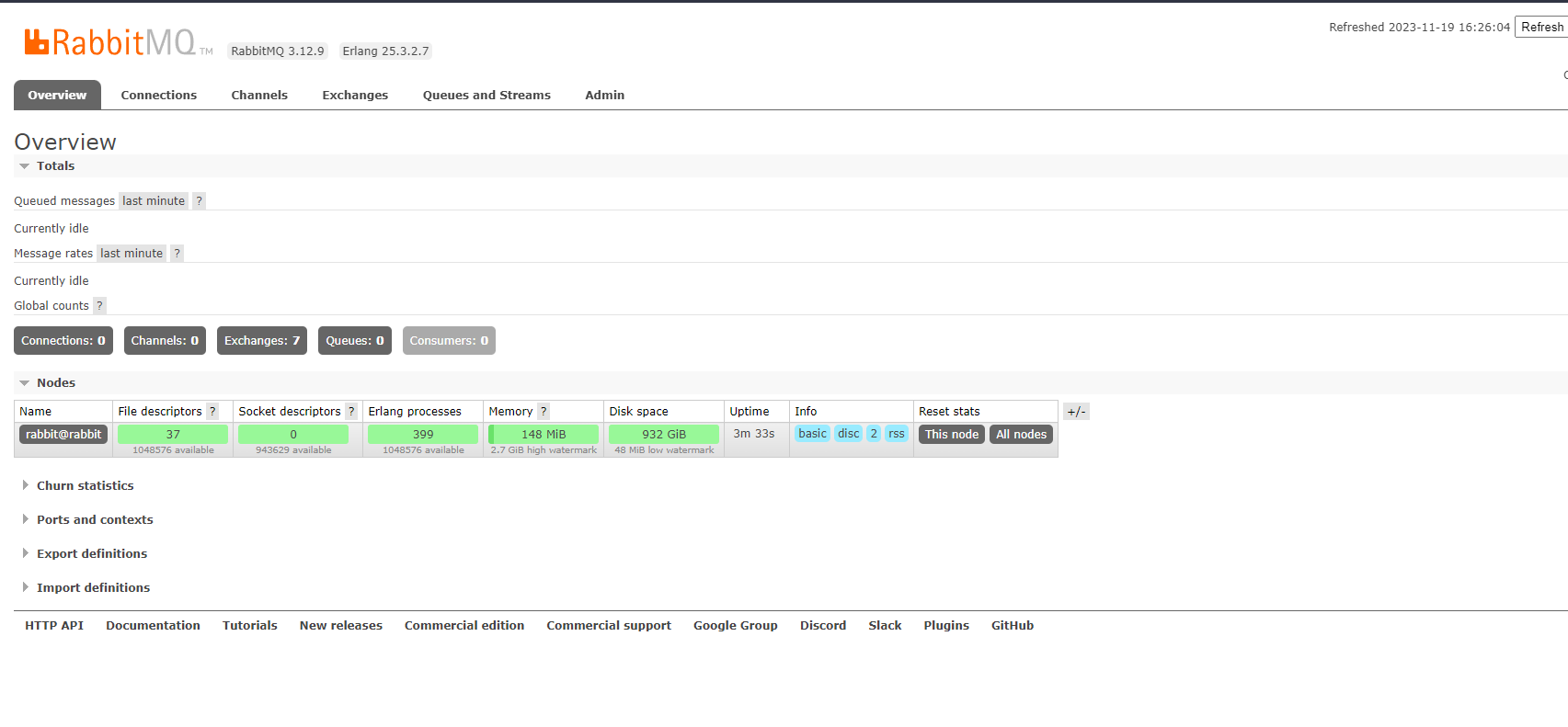


* hostname : is the hostname which we are giving to rabbitMq server
* name: is for the docker UI
* p: is for the port number, so the first port number is your system port number which is mapped with the docker image port number
* 15672 is for the web UI port
* 5672 is for rabbit brocker port

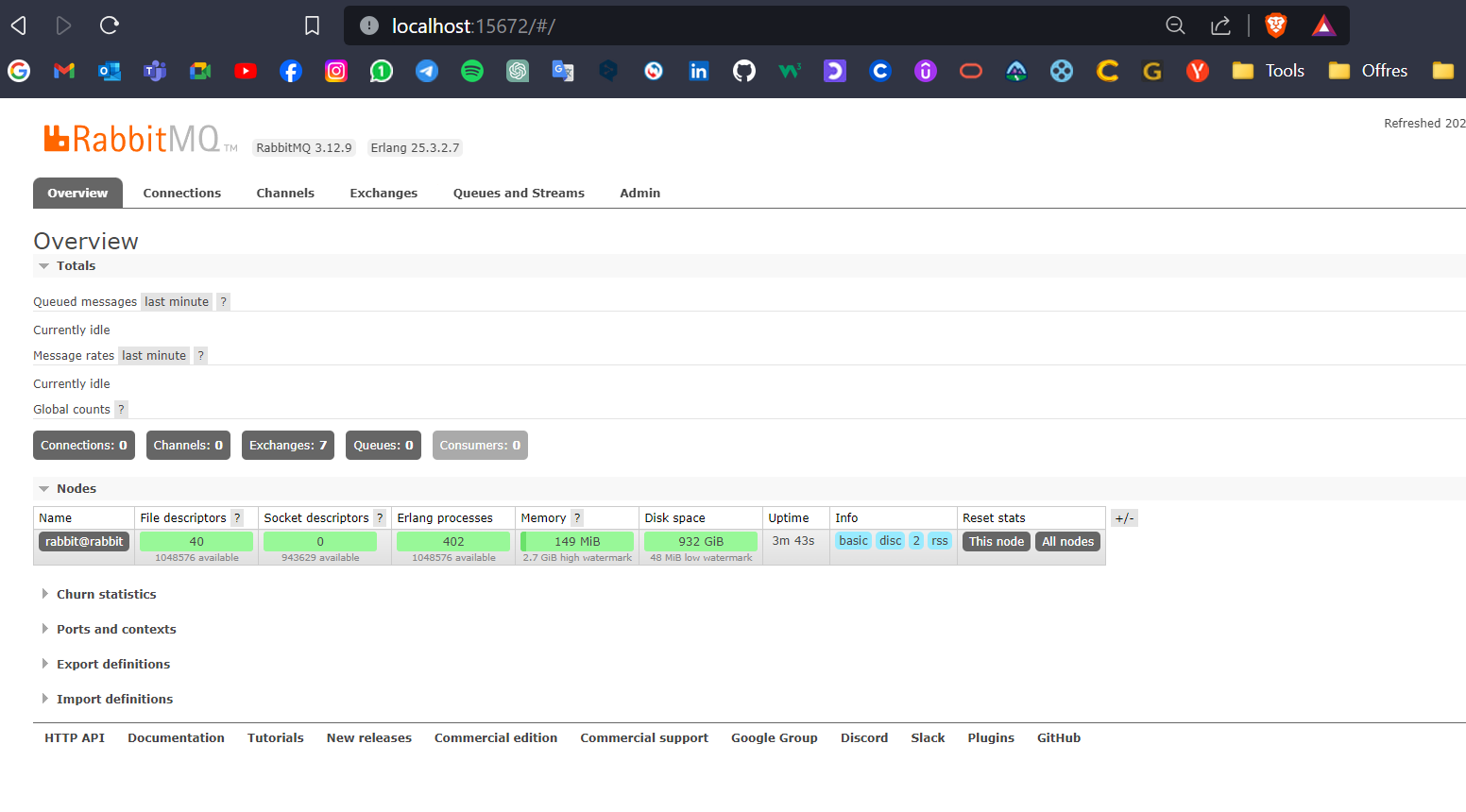


# **Manipulation (Interface)**

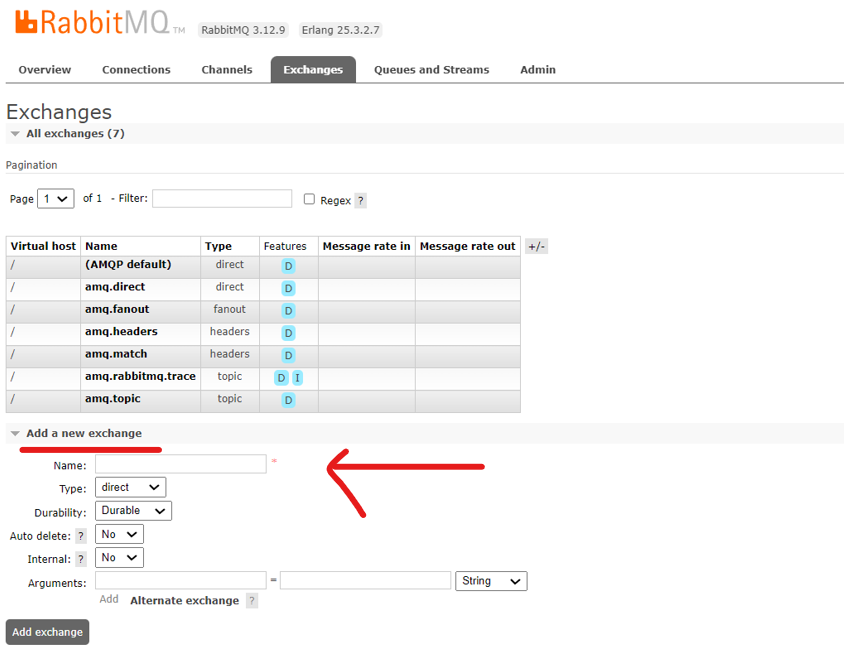
* Interface web
* username: guest
* password: guest

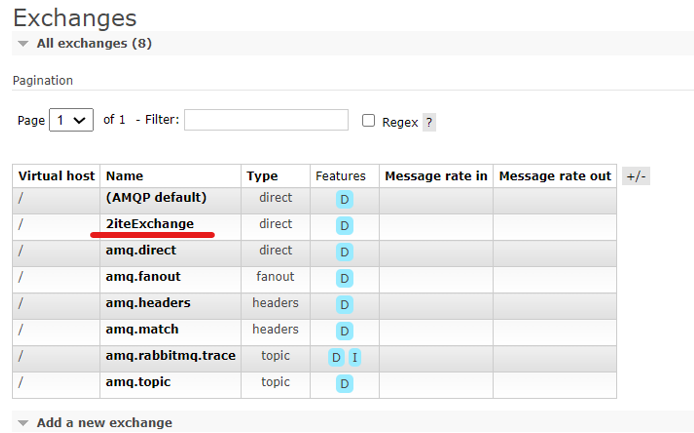


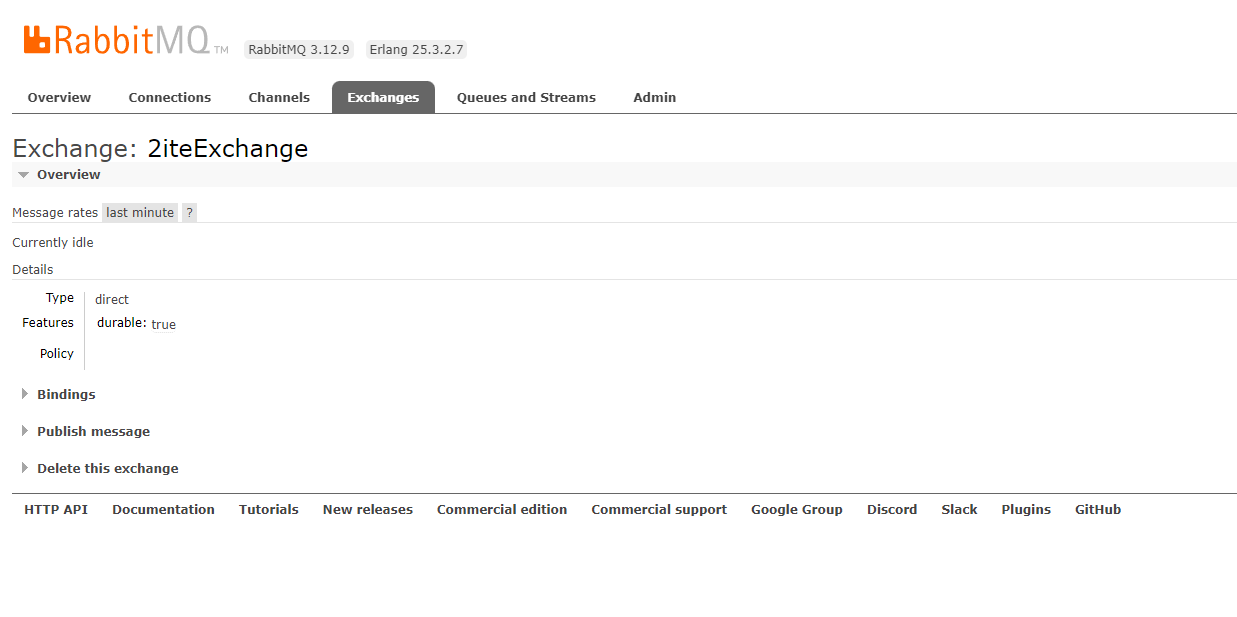
* La page d’accueil

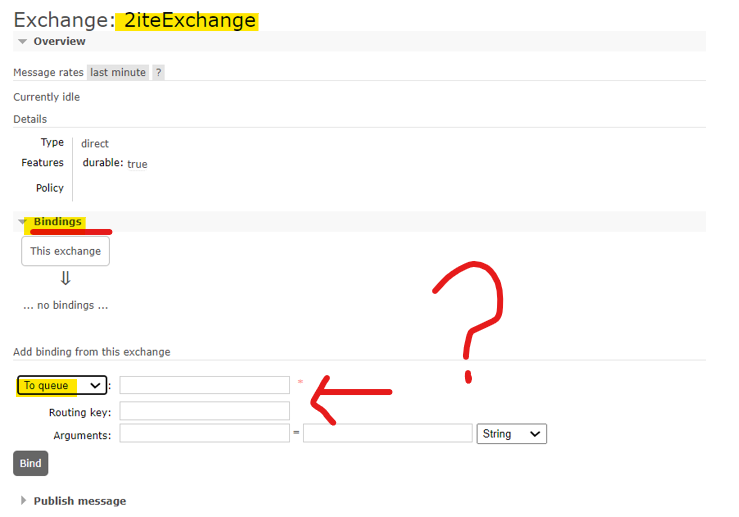


* Ajouter un Exchange : 2iteExchange

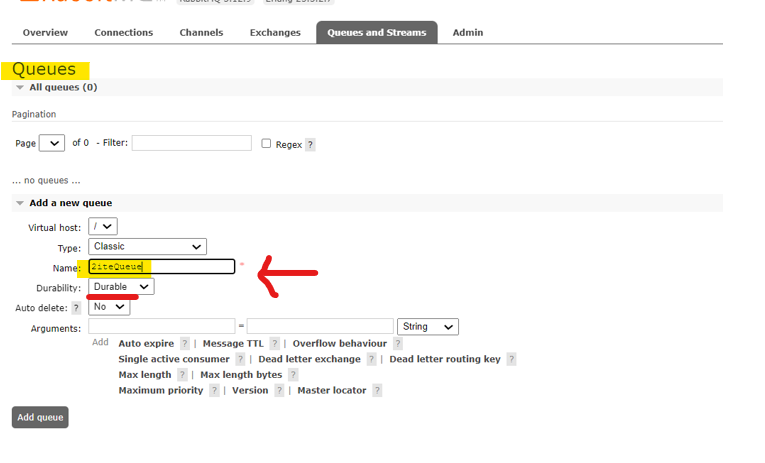


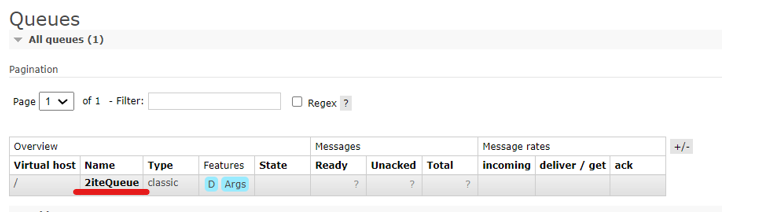




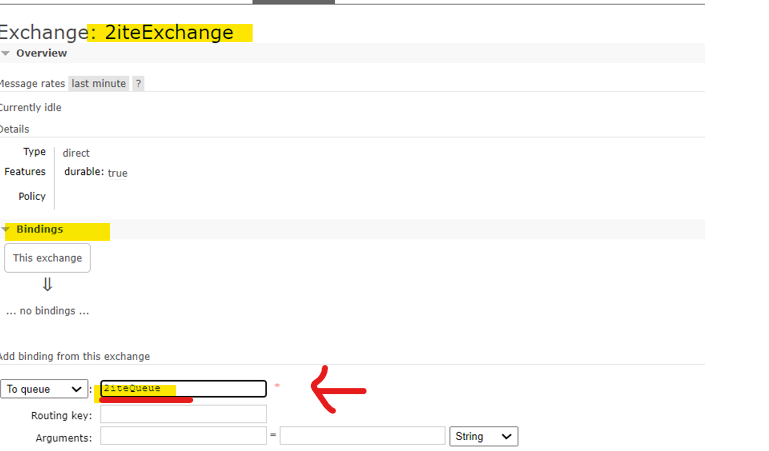


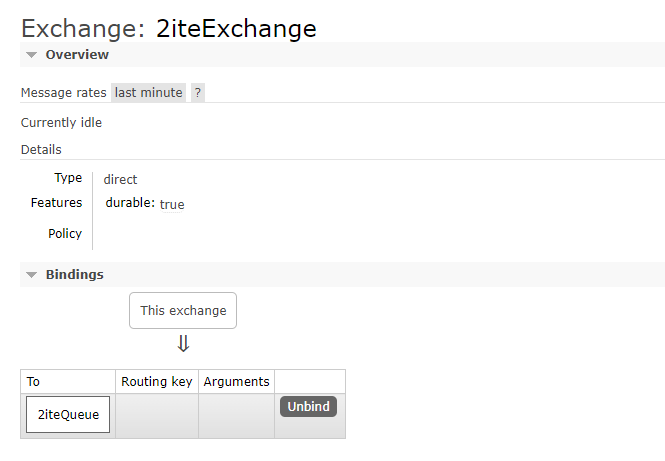
* Ajouter un Queue : 2iteQueue



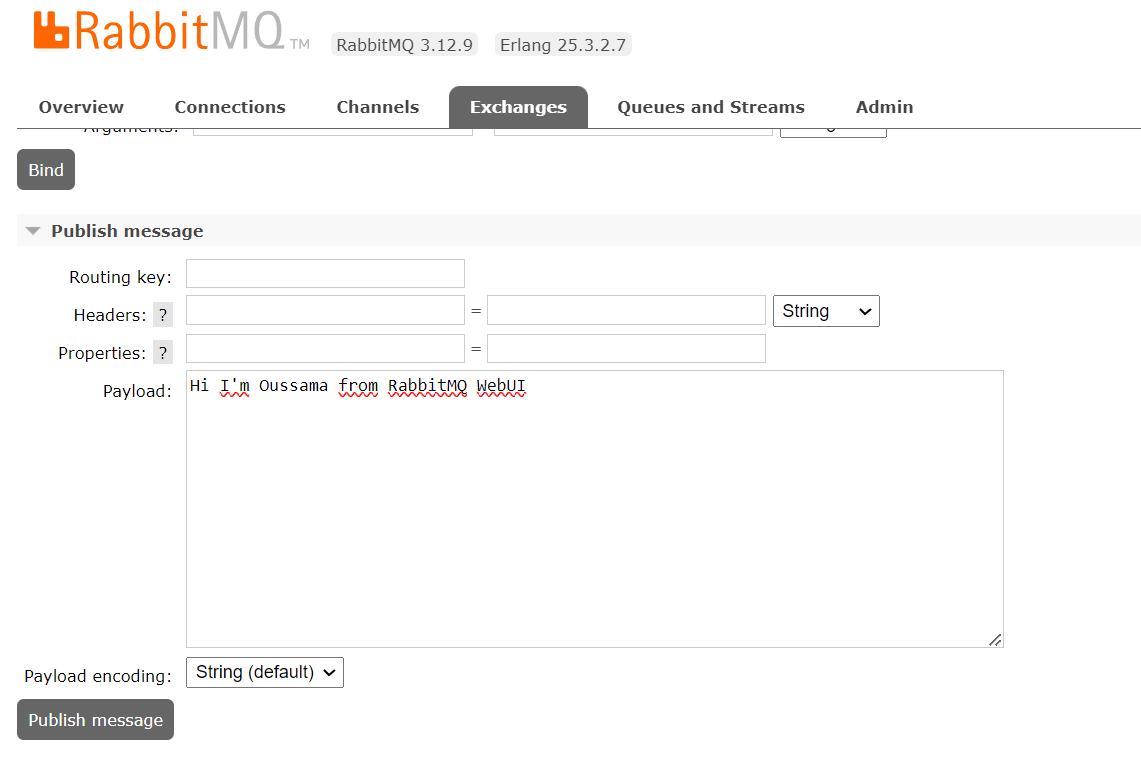


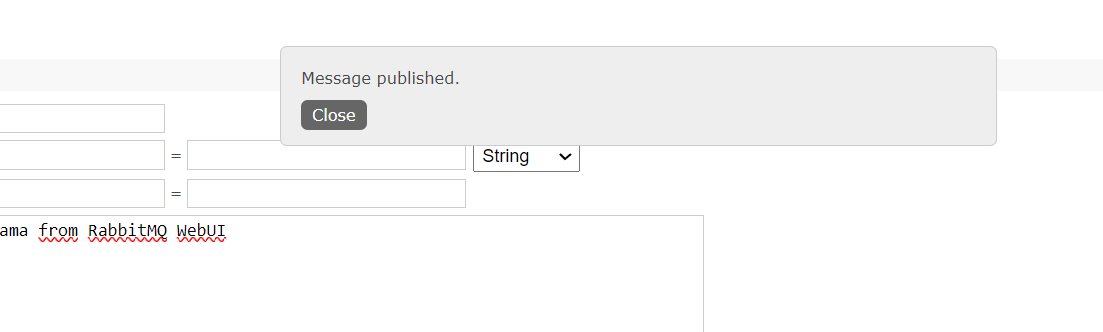
* Sélectionner 2iteQueue dans l’exchange

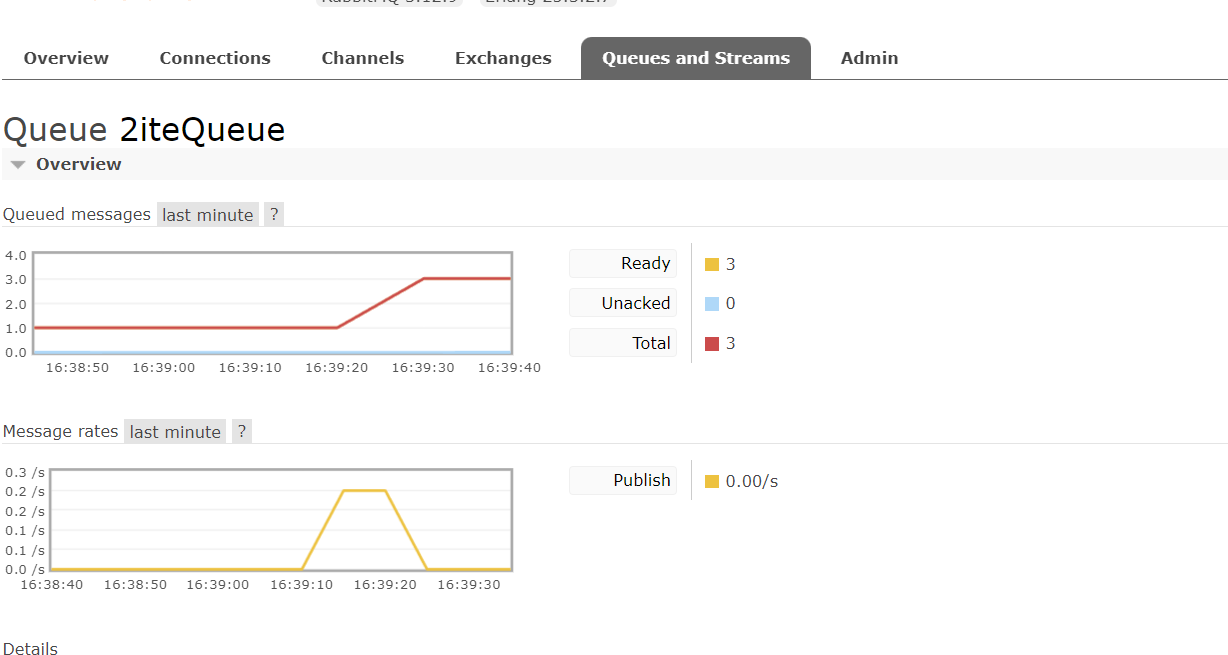




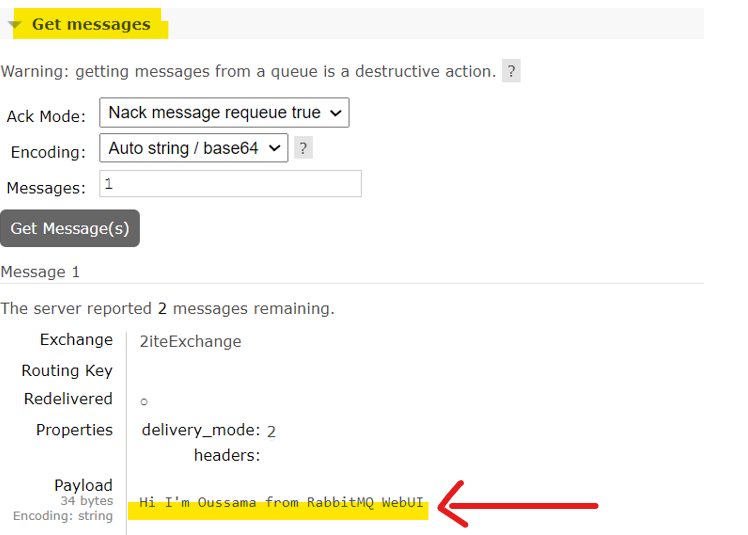
* Publier un message







* Voir les messages



# **Manipulation (Micro services springboot-Messagerie)**

**Dans cette partie, nous allons explorer l'implémentation de RabbitMQ dans Spring Boot en utilisant deux services différents :**

* **spring-rabbitmq-producer**
* **spring-rabbitmq-consumer**

**Au lieu de configurer les paramètres de RabbitMQ dans l'interface comme nous l'avons fait dans l'exemple précédent, cette fois-ci, nous allons les configurer dynamiquement dans notre code.**

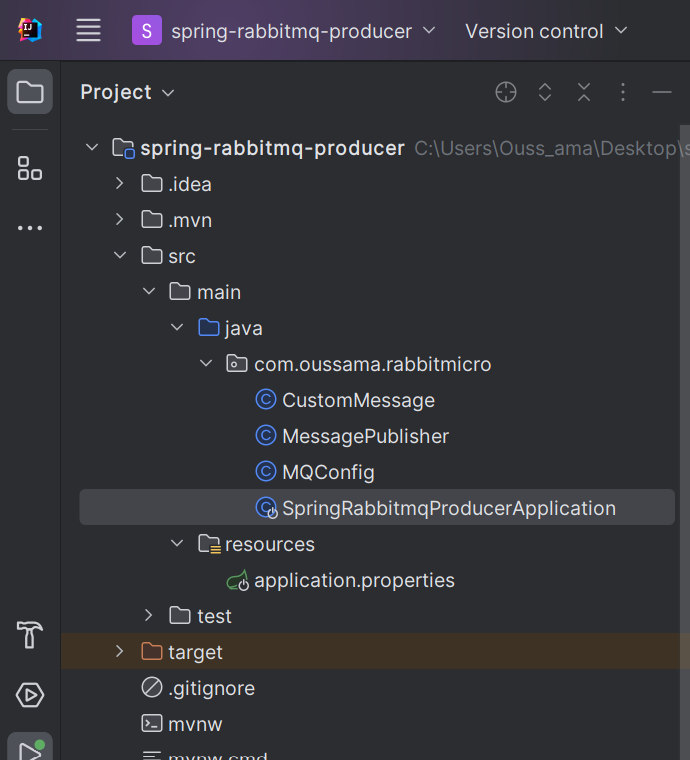
**Ainsi, lorsque nous testerons la communication en utilisant Postman, nous pourrons vérifier que tout fonctionne correctement**

## **spring-rabbitmq-producer**

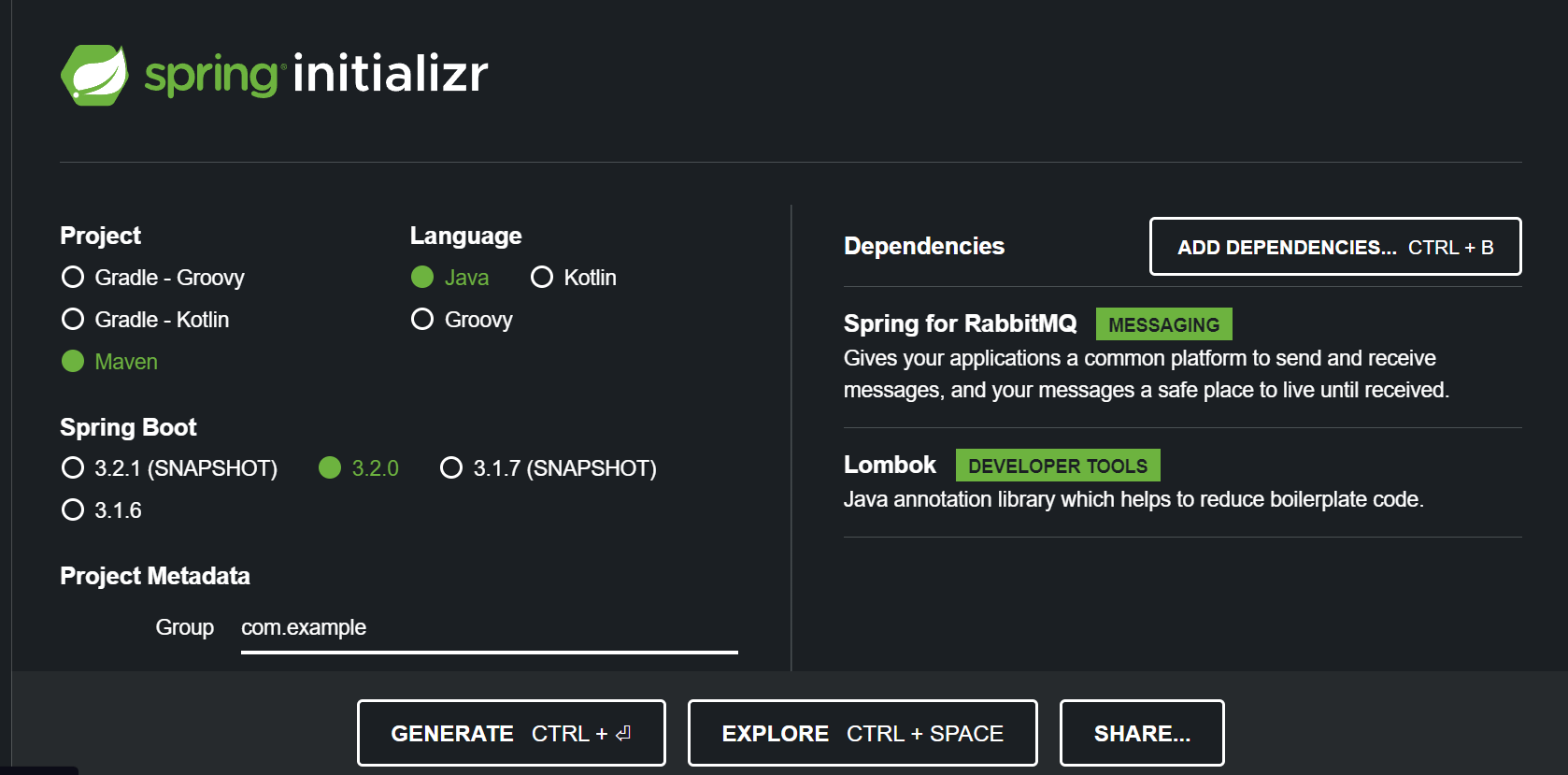
Vous trouver un code source proposer dans mon GitHub :

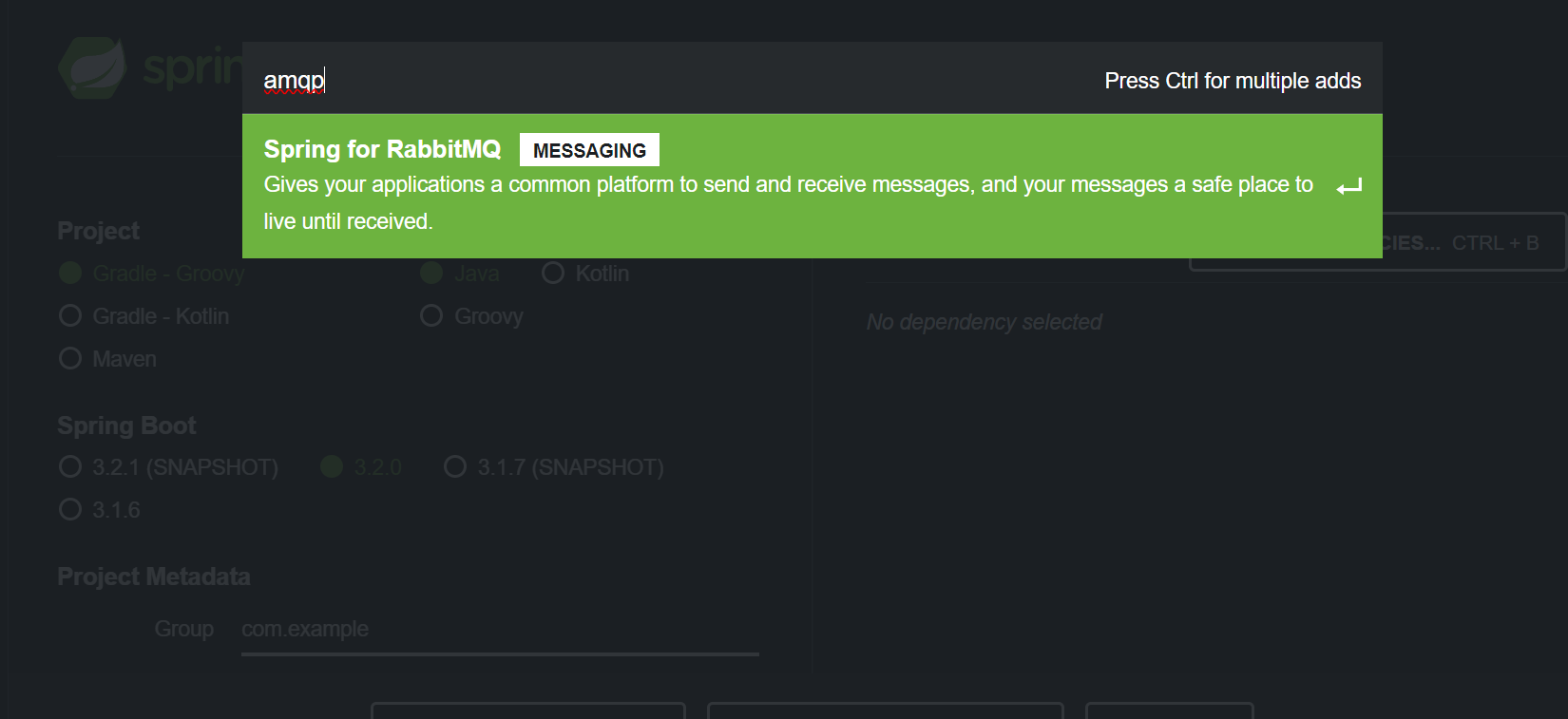
<https://github.com/OUSSAMAOUHA/Microservices_RabbitMq_Messagerie.git>

* Structure du projet :



* Et pour créer un projet de nouveau :

****

****

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

* application.properties

server.port = 8123  
spring.rabbitmq.addresses = localhost:5672

* Configuration RabbitMQ

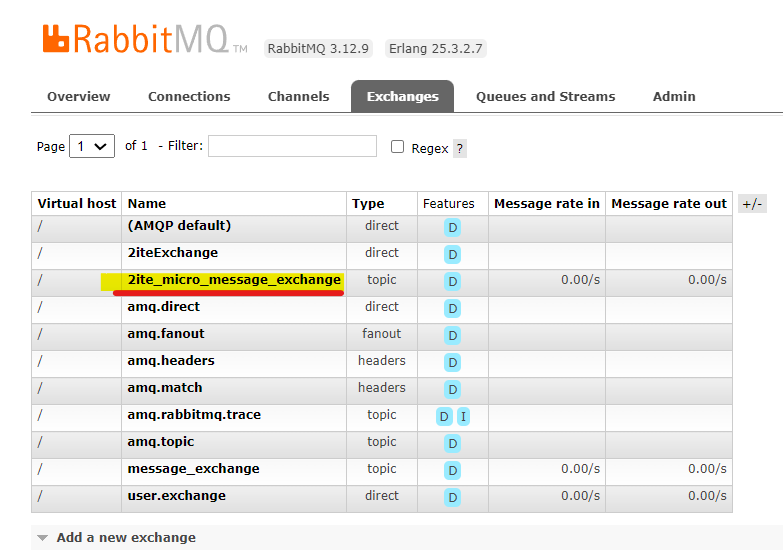
package com.oussama.rabbitmicro;  
  
import org.springframework.amqp.core.\*;  
import org.springframework.amqp.rabbit.connection.ConnectionFactory;  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;  
import org.springframework.amqp.support.converter.MessageConverter;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class MQConfig {  
 public static final String *QUEUE* = "2ite\_micro\_message\_queue";  
 public static final String *EXCHANGE* = "2ite\_micro\_message\_exchange";  
 public static final String *ROUTING\_KEY* = "message\_routingKey";  
  
 @Bean  
 public Queue queue() {  
 return new Queue(*QUEUE*);  
 }  
  
 @Bean  
 public TopicExchange exchange() {  
 return new TopicExchange(*EXCHANGE*);  
 }  
  
 @Bean  
 public Binding binding(Queue queue, TopicExchange exchange) {  
 return BindingBuilder  
 .*bind*(queue)  
 .to(exchange)  
 .with(*ROUTING\_KEY*);  
 }  
  
 @Bean  
 public MessageConverter messageConverter() {  
 return new Jackson2JsonMessageConverter();  
 }  
  
 @Bean  
 public AmqpTemplate template(ConnectionFactory connectionFactory) {  
 RabbitTemplate template = new RabbitTemplate(connectionFactory);  
 template.setMessageConverter(messageConverter());  
 return template;  
 }  
  
}

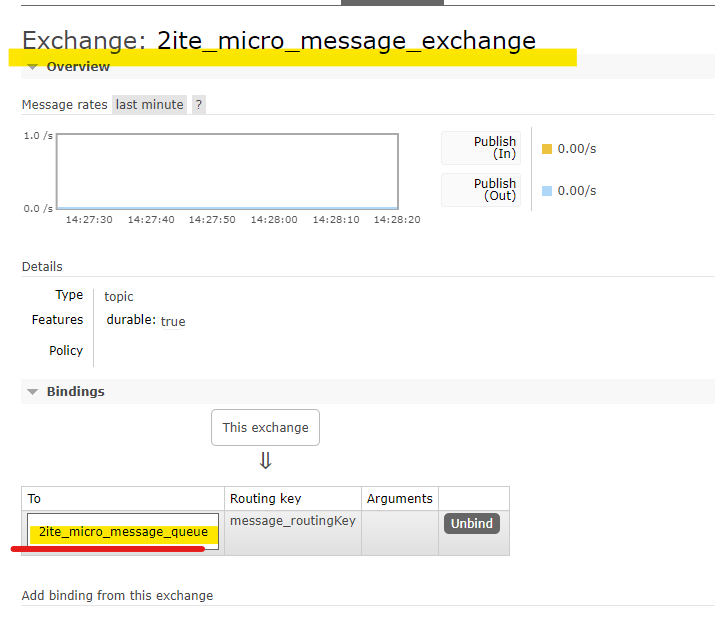
* CustomMessage

package com.oussama.rabbitmicro;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.ToString;  
  
import java.util.Date;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
public class CustomMessage {  
  
 private String messageId;  
 private String message;  
 private Date messageDate;  
  
}

* MessagePublisher

package com.oussama.rabbitmicro;  
  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.Date;  
import java.util.UUID;  
  
@RestController  
public class MessagePublisher {  
  
 @Autowired  
 private RabbitTemplate template;  
  
 @PostMapping("/publish")  
 public String publishMessage(@RequestBody CustomMessage message) {  
 message.setMessageId(UUID.*randomUUID*().toString());  
 message.setMessageDate(new Date());  
 template.convertAndSend(MQConfig.*EXCHANGE*,  
 MQConfig.*ROUTING\_KEY*, message);  
  
 return "Message Published";  
 }  
}

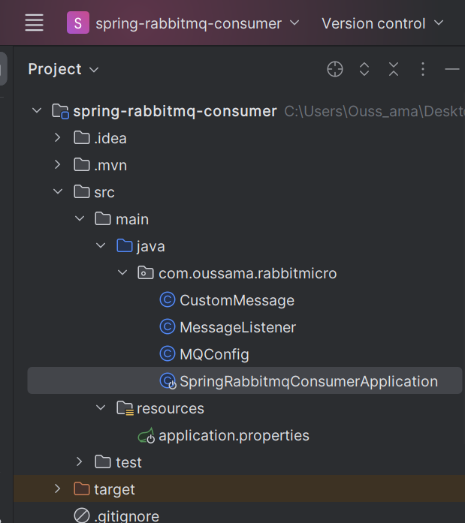
* Vérification dans l’interface



## **spring-rabbitmq-cosomer**

Vous trouver un code source proposer dans mon GitHub :

<https://github.com/OUSSAMAOUHA/Microservices_RabbitMq_Messagerie.git>

* Structure du projet :
* application.properties

server.port = 8223  
spring.rabbitmq.addresses = localhost:5672

* Configuration RabbitMQ

package com.oussama.rabbitmicro;  
  
import org.springframework.amqp.core.\*;  
import org.springframework.amqp.rabbit.connection.ConnectionFactory;  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;  
import org.springframework.amqp.support.converter.MessageConverter;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class MQConfig {  
 public static final String *QUEUE* = "2ite\_micro\_message\_queue";  
 public static final String *EXCHANGE* = "2ite\_micro\_message\_exchange";  
 public static final String *ROUTING\_KEY* = "message\_routingKey";  
  
 @Bean  
 public Queue queue() {  
 return new Queue(*QUEUE*);  
 }  
  
 @Bean  
 public TopicExchange exchange() {  
 return new TopicExchange(*EXCHANGE*);  
 }  
  
 @Bean  
 public Binding binding(Queue queue, TopicExchange exchange) {  
 return BindingBuilder  
 .*bind*(queue)  
 .to(exchange)  
 .with(*ROUTING\_KEY*);  
 }  
  
 @Bean  
 public MessageConverter messageConverter() {  
 return new Jackson2JsonMessageConverter();  
 }  
  
 @Bean  
 public AmqpTemplate template(ConnectionFactory connectionFactory) {  
 RabbitTemplate template = new RabbitTemplate(connectionFactory);  
 template.setMessageConverter(messageConverter());  
 return template;  
 }  
}

* CustomMessage

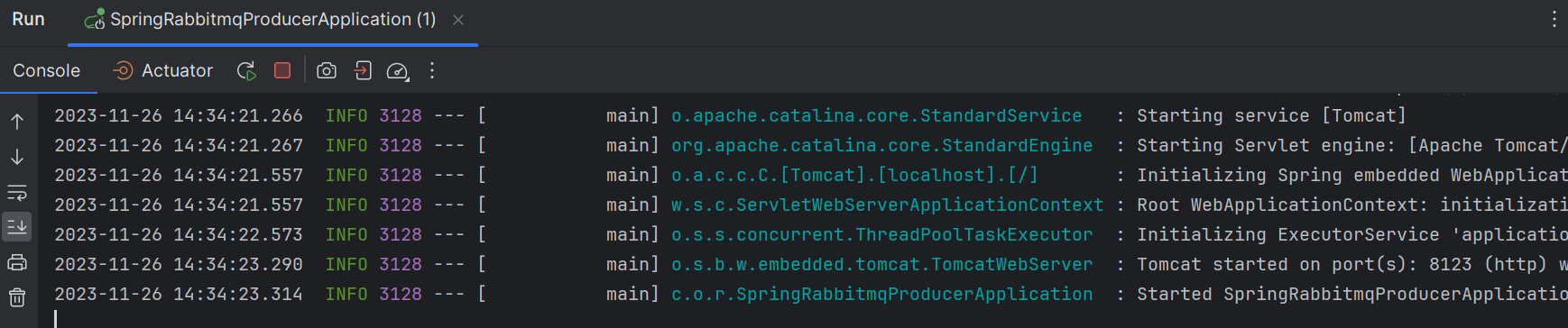
package com.oussama.rabbitmicro;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.ToString;  
  
import java.util.Date;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
public class CustomMessage {  
  
 private String messageId;  
 private String message;  
 private Date messageDate;  
  
}

* MessageListner

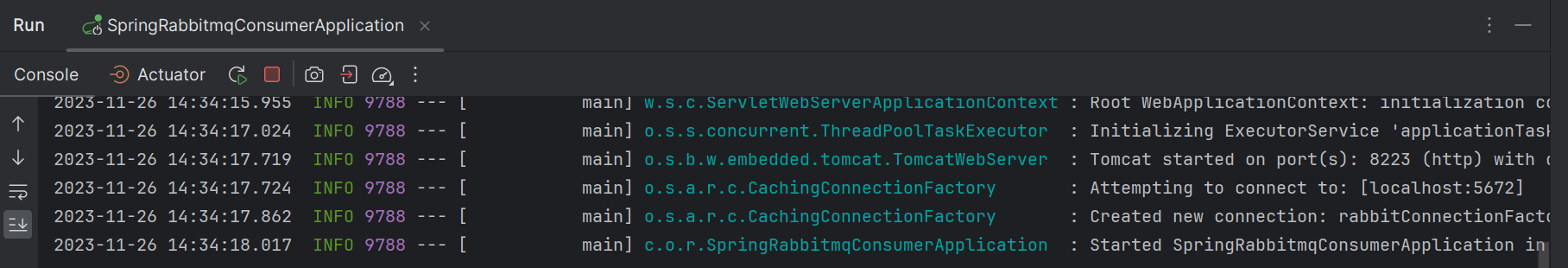
package com.oussama.rabbitmicro;  
  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
public class MessageListener {  
  
 @RabbitListener(queues = MQConfig.*QUEUE*)  
 public void listener(CustomMessage message) {  
 System.*out*.println(message);  
 }  
  
}

## **Test de la communication**

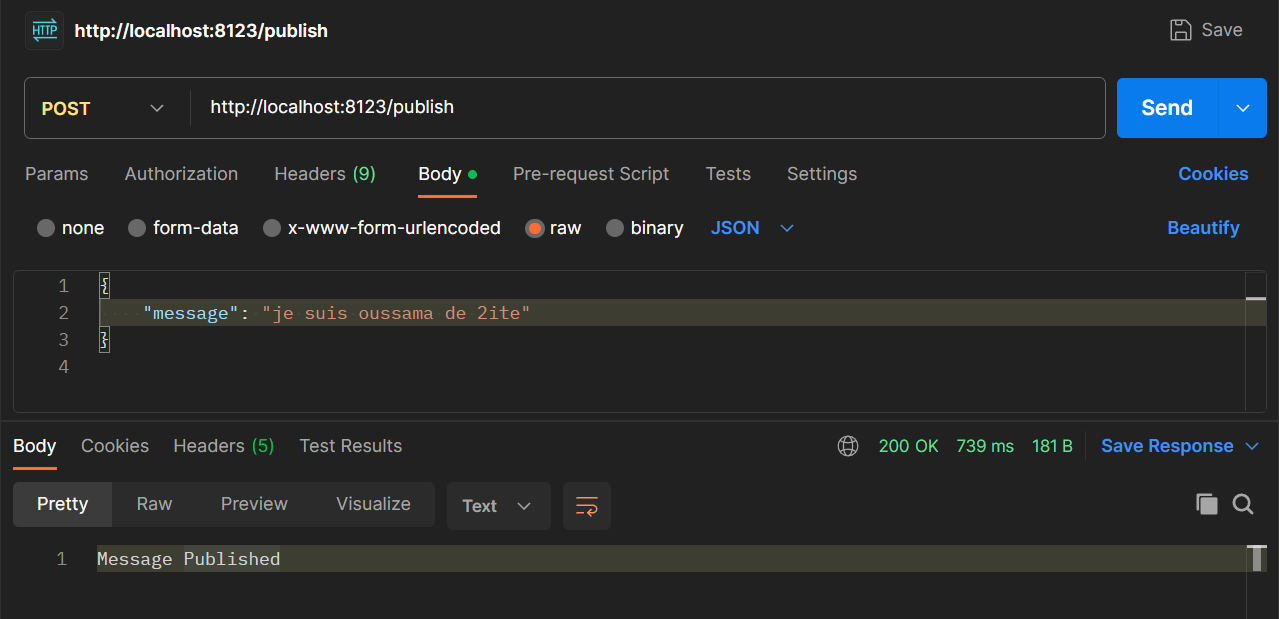
* Start the producer

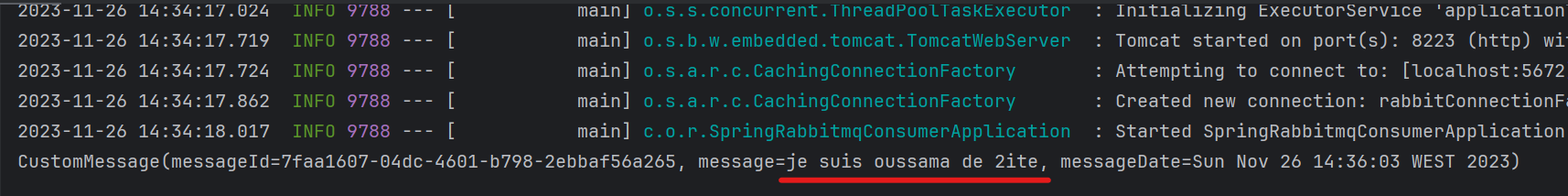


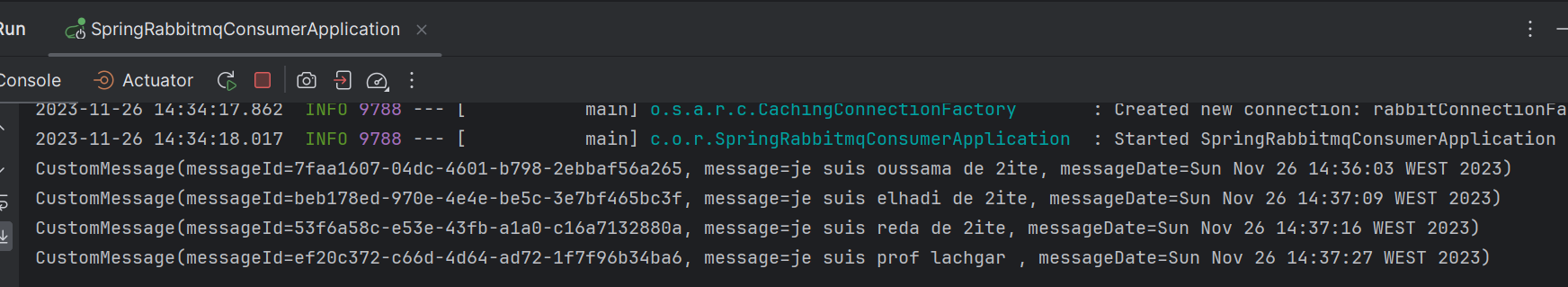
* Start the consumer

****

* Test







# **Manipulation (Micro services springboot-MySQL)**

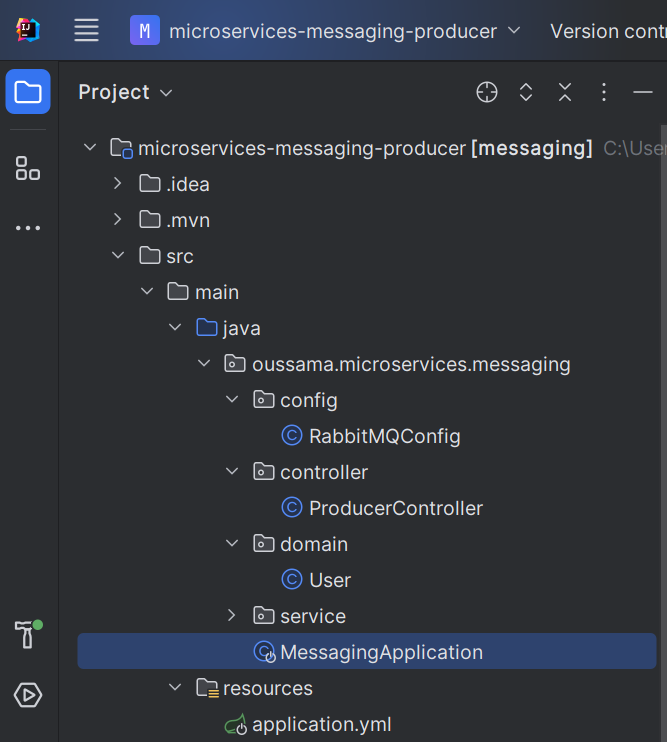
**Dans cette partie, nous allons insérer des données provenant du service producteur dans la base de données du service consommateur en utilisant la file d'attente de RabbitMQ**

Vous trouvez un code source proposer dans mon GitHub :

[**https://github.com/OUSSAMAOUHA/Mivroservices\_SpringBoot\_RabbitMq\_Streaming**](https://github.com/OUSSAMAOUHA/Mivroservices_SpringBoot_RabbitMq_Streaming)

## **spring-rabbitmq-producer**

* Structure du projet :



* Configuration RabbitMQ

package oussama.microservices.messaging.config;  
  
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;  
import org.springframework.amqp.rabbit.connection.ConnectionFactory;  
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;  
import org.springframework.amqp.support.converter.MessageConverter;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class RabbitMQConfig {  
  
 @Value("${spring.rabbitmq.host}")  
 String host;  
  
 @Value("${spring.rabbitmq.username}")  
 String username;  
  
 @Value("${spring.rabbitmq.password}")  
 String password;  
  
 @Bean  
 CachingConnectionFactory connectionFactory() {  
 CachingConnectionFactory cachingConnectionFactory = new CachingConnectionFactory(host);  
 cachingConnectionFactory.setUsername(username);  
 cachingConnectionFactory.setPassword(password);  
 return cachingConnectionFactory;  
 }  
  
 @Bean  
 public MessageConverter jsonMessageConverter() {  
 return new Jackson2JsonMessageConverter();  
 }  
  
 @Bean  
 public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {  
 final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);  
 rabbitTemplate.setMessageConverter(jsonMessageConverter());  
 return rabbitTemplate;  
 }  
}

* Classe User

package oussama.microservices.messaging.domain;  
  
import com.fasterxml.jackson.annotation.JsonIdentityInfo;  
import com.fasterxml.jackson.annotation.ObjectIdGenerators;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.ToString;  
import org.springframework.stereotype.Component;  
  
import java.io.Serializable;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@ToString  
@Component  
public class User implements Serializable {  
 private String userId;  
 private String userName;  
}

* ProducerController

package oussama.microservices.messaging.controller;  
  
import oussama.microservices.messaging.domain.User;  
import oussama.microservices.messaging.service.ProducerService;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
@RequestMapping("/api/")  
public class ProducerController {  
  
 private ProducerService producerService;  
 private static final Logger *logger* = LoggerFactory.*getLogger*(ProducerController.class);  
  
 @Autowired  
 public ProducerController(ProducerService producerService) {  
 this.producerService = producerService;  
 }  
  
 @Value("${app.message}")  
 private String response;  
  
 @PostMapping("/produce")  
 public ResponseEntity<String> sendMessage(@RequestBody User user) {  
 producerService.sendMessage(user);  
 *logger*.info("user sent: " + user);  
 return ResponseEntity.*ok*("user sent: " + user);  
 }  
}

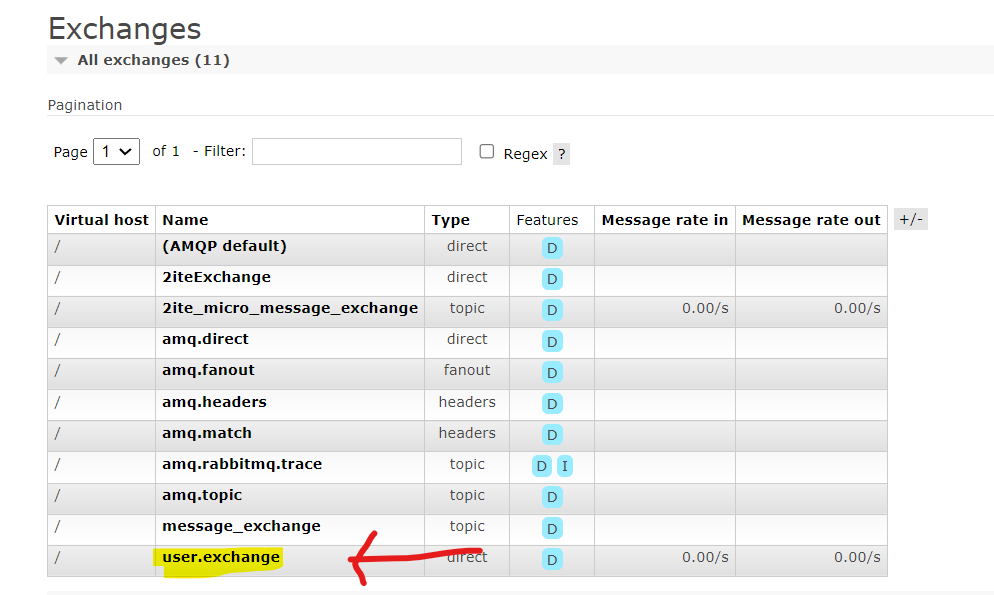
* ProducerService

package oussama.microservices.messaging.service;  
  
import oussama.microservices.messaging.domain.User;  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
@Service  
public class ProducerService {  
  
 private RabbitTemplate rabbitTemplate;  
  
 @Autowired  
 public ProducerService(RabbitTemplate rabbitTemplate) {  
 this.rabbitTemplate = rabbitTemplate;  
 }  
  
 @Value("${spring.rabbitmq.exchange}")  
 private String exchange;  
  
 @Value("${spring.rabbitmq.routingkey}")  
 private String routingkey;  
  
 public void sendMessage(User user) {  
 rabbitTemplate.convertAndSend(exchange,routingkey, user);  
 }  
  
}

* Application.yml

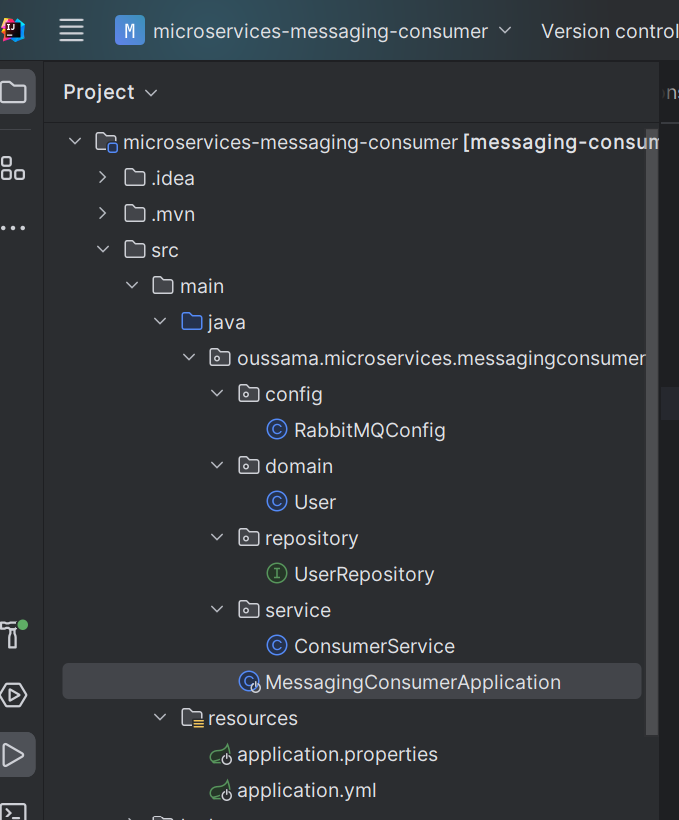
app:  
 message: message succesfuly sent  
spring:  
 rabbitmq:  
 host: localhost  
 password: guest  
 port: 15672  
 username: guest  
 exchange: user.exchange  
 routingkey: user.routingkey  
server:  
 port: 8081

* Vérification dans l’interface



## **spring-rabbitmq-cosomer**

* Structure du projet :



* application.properties

spring:  
 rabbitmq:  
 host: localhost  
 password: guest  
 port: 15672  
 username: guest  
 exchange: user.exchange  
 queue: user.queue  
 routingkey: user.routingkey

* application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/testdb?useSSL=false&serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.jpa.hibernate.ddl-auto=update

* Configuration RabbitMQ

package oussama.microservices.messagingconsumer.config;  
  
import org.springframework.amqp.core.\*;  
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;  
import org.springframework.amqp.rabbit.connection.ConnectionFactory;  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;  
import org.springframework.amqp.support.converter.MessageConverter;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class RabbitMQConfig {  
  
 @Value("${spring.rabbitmq.queue}")  
 private String queue;  
  
 @Value("${spring.rabbitmq.exchange}")  
 private String exchange;  
  
 @Value("${spring.rabbitmq.routingkey}")  
 private String routingKey;  
  
 @Value("${spring.rabbitmq.username}")  
 private String username;  
  
 @Value("${spring.rabbitmq.password}")  
 private String password;  
  
 @Value("${spring.rabbitmq.host}")  
 private String host;  
  
 @Bean  
 Queue queue() {  
 return new Queue(queue, true);  
 }  
  
 @Bean  
 Exchange myExchange() {  
 return ExchangeBuilder.*directExchange*(exchange).durable(true).build();  
 }  
  
 @Bean  
 Binding binding() {  
 return BindingBuilder  
 .*bind*(queue())  
 .to(myExchange())  
 .with(routingKey)  
 .noargs();  
 }  
  
 @Bean  
 public ConnectionFactory connectionFactory() {  
 CachingConnectionFactory cachingConnectionFactory = new CachingConnectionFactory(host);  
 cachingConnectionFactory.setUsername(username);  
 cachingConnectionFactory.setPassword(password);  
 return cachingConnectionFactory;  
 }  
  
 @Bean  
 public MessageConverter jsonMessageConverter() {  
 return new Jackson2JsonMessageConverter();  
 }  
  
 @Bean  
 public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {  
 final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);  
 rabbitTemplate.setMessageConverter(jsonMessageConverter());  
 return rabbitTemplate;  
 }  
}

}

* CustomMessage

package com.oussama.rabbitmicro;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.ToString;  
  
import java.util.Date;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
public class CustomMessage {  
  
 private String messageId;  
 private String message;  
 private Date messageDate;  
  
}

* User Class

package oussama.microservices.messagingconsumer.domain;  
  
import com.fasterxml.jackson.annotation.JsonIdentityInfo;  
import com.fasterxml.jackson.annotation.ObjectIdGenerators;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.ToString;  
import org.springframework.stereotype.Component;  
  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import java.io.Serializable;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@ToString  
@Entity  
public class User implements Serializable {  
 @Id @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String userId;  
 private String userName;  
}

* User Repository

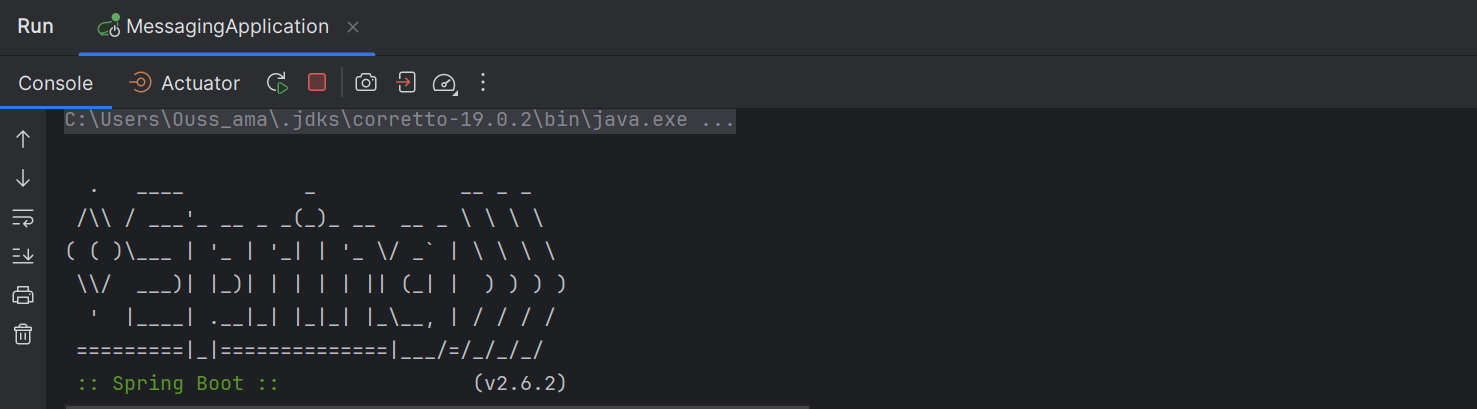
package oussama.microservices.messagingconsumer.repository;  
  
import oussama.microservices.messagingconsumer.domain.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
}

* Consumer Service

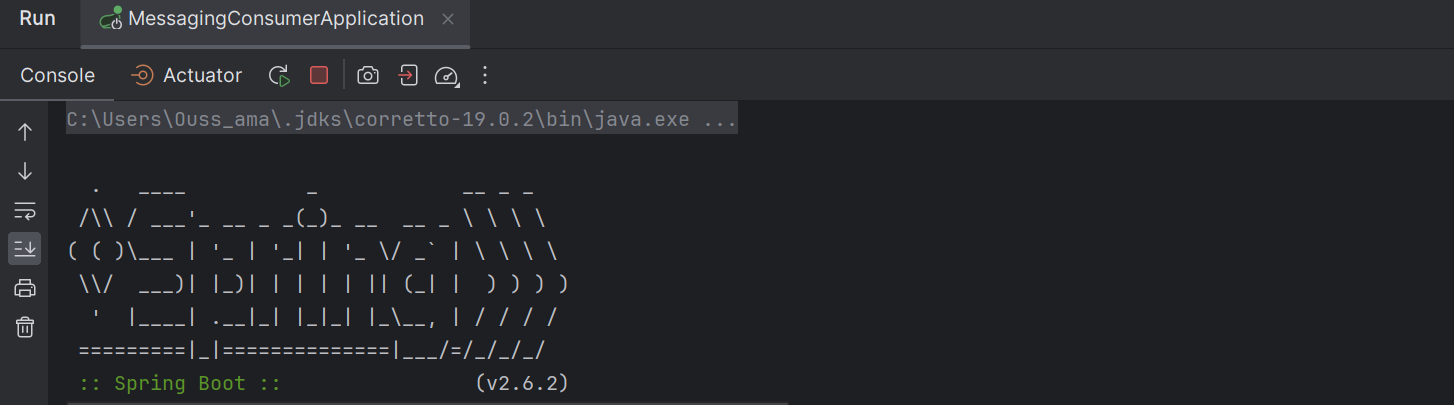
package oussama.microservices.messagingconsumer.service;  
  
import oussama.microservices.messagingconsumer.domain.User;  
import oussama.microservices.messagingconsumer.repository.UserRepository;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Service;  
  
@Service  
public class ConsumerService {  
  
 private final UserRepository userRepository;  
 private static final Logger *logger* = LoggerFactory.*getLogger*(ConsumerService.class);  
  
 @Autowired  
 public ConsumerService(UserRepository userRepository) {  
 this.userRepository = userRepository;  
 }  
  
 @RabbitListener(queues = "${spring.rabbitmq.queue}")  
 public void receivedMessage(User user) {  
 User save = userRepository.save(user);  
 *logger*.info("persisted " + save);  
 *logger*.info("User recieved: " + user);  
 }  
  
}

## **Test de la communication**

* Start the producer



* Start the consumer



* Test

