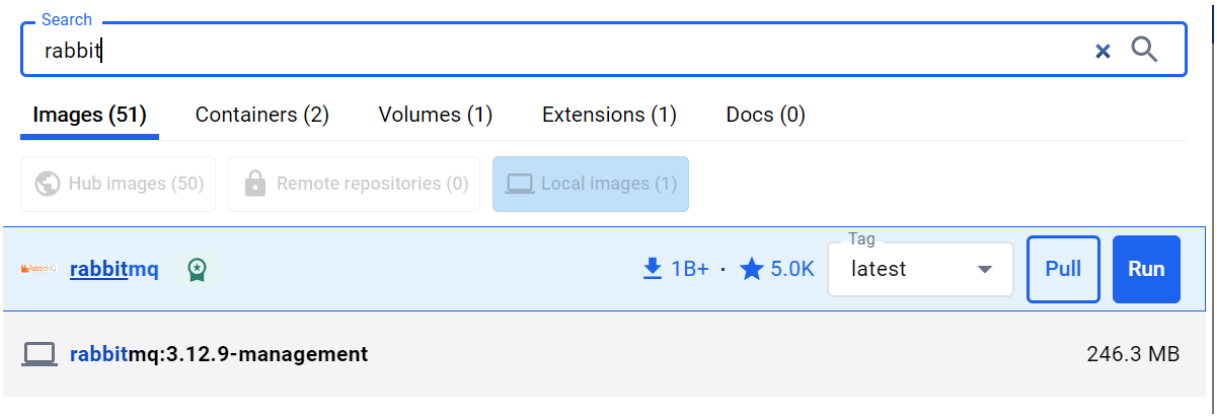
Installation de RabbitMQ

Installation de l'image Docker de RabbitMQ (version stable : 3.12.9-management) sur Docker



Exécution de l'image avec des paramètres tels que le nom d'hôte, le nom de l'interface Docker, et les numéros de port (15672 pour l'interface web et 5672 pour le port de RabbitMQ).

```
C:\Users\Ouss_ama>docker run -d --hostname rabbit --name rabbit-server -p 15672:15672 -p 5672:5672 ra
bbitmq:3.12.9-management
87f838ee60185b0ea3109e104385d941a046c98c341b0d432859f80dbf8ceb78
```

Maintenant dans le web :

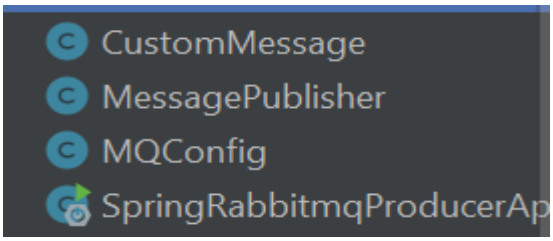
Entrer sur : http://localhost:15672

Entrer : Username : Guest

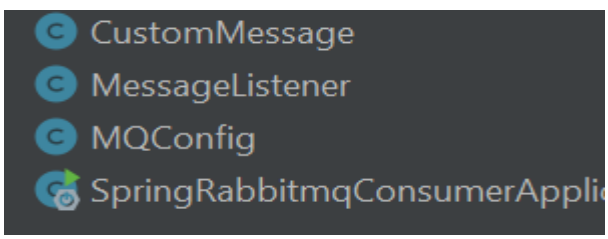
Password : Guest

On aura trois projets :

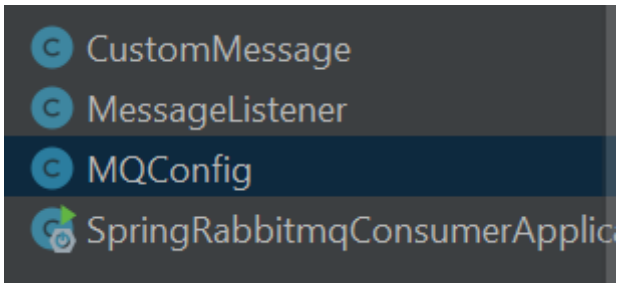
Producer :



Consommer_1 :



Consommer_2 :



Comme vous voyez , on as deux fichiers commun dans toutes les projet :

- **CustomMessage**
- **MessageListner**

Alors maintenant je vous presentre le contenu de chaque fichier :

- **CustomMessage :**

```
1 usage
@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class CustomMessage {
    no usages
    private String messageId;
    no usages
    private String message;
    no usages
    private Date messageDate;

}
```

Le fichier CustomMessage **définit une classe de modèle pour représenter une structure de message personnalisée échangée entre le producteur et les consommateurs dans une application RabbitMQ.**

- **MessageListner :**

```
@Configuration
public class MQConfig {
    public static final String QUEUE_1 = "2ite_micro_message_queue_1";
    public static final String QUEUE_2 = "2ite_micro_message_queue_2";
    public static final String EXCHANGE = "2ite_micro_message_exchange";
    public static final String ROUTING_KEY_1 = "message_routingKey_1";
    public static final String ROUTING_KEY_2 = "message_routingKey_2";

    @Bean
    public Queue queue1() {
        return  new Queue(QUEUE_1);
    }

    @Bean
    public Queue queue2() {
        return new Queue(QUEUE_2);
    }
    @Bean
    public TopicExchange exchange() {
        return new TopicExchange(EXCHANGE);
    }

    @Bean
    public Binding binding1(Queue queue1, TopicExchange exchange) {
        return BindingBuilder
                .bind(queue1)
                .to(exchange)
                .with(ROUTING_KEY_1);
    }

    @Bean
    public Binding binding2(Queue queue2, TopicExchange exchange) {
        return BindingBuilder
                .bind(queue2)
                .to(exchange)
                .with(ROUTING_KEY_2);
    }

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

    @Bean
    public AmqpTemplate template(ConnectionFactory connectionFactory) {
        RabbitTemplate template = new RabbitTemplate(connectionFactory);
        template.setMessageConverter(messageConverter());
        return  template;
    }

}
```

Le fichier MQConfig **joue un rôle central dans la configuration de la communication entre composants d'une application Spring Boot avec RabbitMQ. Il définit deux queues (**QUEUE_1 **et** QUEUE_2**), un échange (**EXCHANGE**), et les bindings correspondants avec des clés de routage distinctes (**ROUTING_KEY_1 **et** ROUTING_KEY_2**). Ces configurations facilitent l'échange de**

messages entre différents modules de l'application. De plus, le fichier déclare des beans pour les queues, l'échange, le convertisseur de message, et le template RabbitMQ, permettant une intégration fluide avec le système de messagerie.

Maintenant , pour chaque projet on vas explique les fichiers qui sont pas commun :

Projet 1 :

- **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_1, message);
        template.convertAndSend(MQConfig.EXCHANGE,
                MQConfig.ROUTING_KEY_2, message);

        return "Message Published";
    }
}
```

Ce fichier MessagePublisher **représente le contrôleur REST responsable de la publication de messages dans le système RabbitMQ. Dans la méthode** publishMessage**, un objet** CustomMessage **est reçu en tant que corps de la requête POST.**

Un identifiant unique (messageId**) est généré, et la date actuelle est assignée au champ** messageDate **de l'objet. Ensuite, le template RabbitMQ est utilisé pour envoyer ce message à deux queues distinctes en utilisant les clés de routage définies dans** MQConfig **(**ROUTING_KEY_1 **et** ROUTING_KEY_2**).**

Ainsi, le message est publié simultanément aux deux queues configurées dans le système, permettant aux consommateurs appropriés de traiter les messages en fonction de la clé de routage spécifiée. La méthode renvoie une chaîne indiquant que le message a été publié avec succès.

Projet 2 :

- **MessageListner1 :**

```
-
    @Component
    public class MessageListener {

        @RabbitListener(queues = MQConfig.QUEUE_2)
        public void handleMessageFromQueue1(CustomMessage message) {

            System.out.println("Consumer 2 - Received message from Queue 2:
    " + message);

        }

    }
```

Ce fichier MessageListener1 **définit un composant Spring géré par conteneur responsable de la consommation des messages de la deuxième queue (**MQConfig.QUEUE_2**) dans le système RabbitMQ. Utilisant l'annotation** @RabbitListener**, la méthode** handleMessageFromQueue2 **est configurée pour écouter et traiter les messages provenant de cette queue spécifique. Lorsqu'un message est reçu, la méthode est invoquée, et le contenu du message est affiché dans la console. En résumé, le fichier** MessageListener **est un consommateur qui réagit à l'arrivée de messages dans la deuxième queue et effectue des actions spécifiques définies dans la méthode** handleMessageFromQueue2**.**

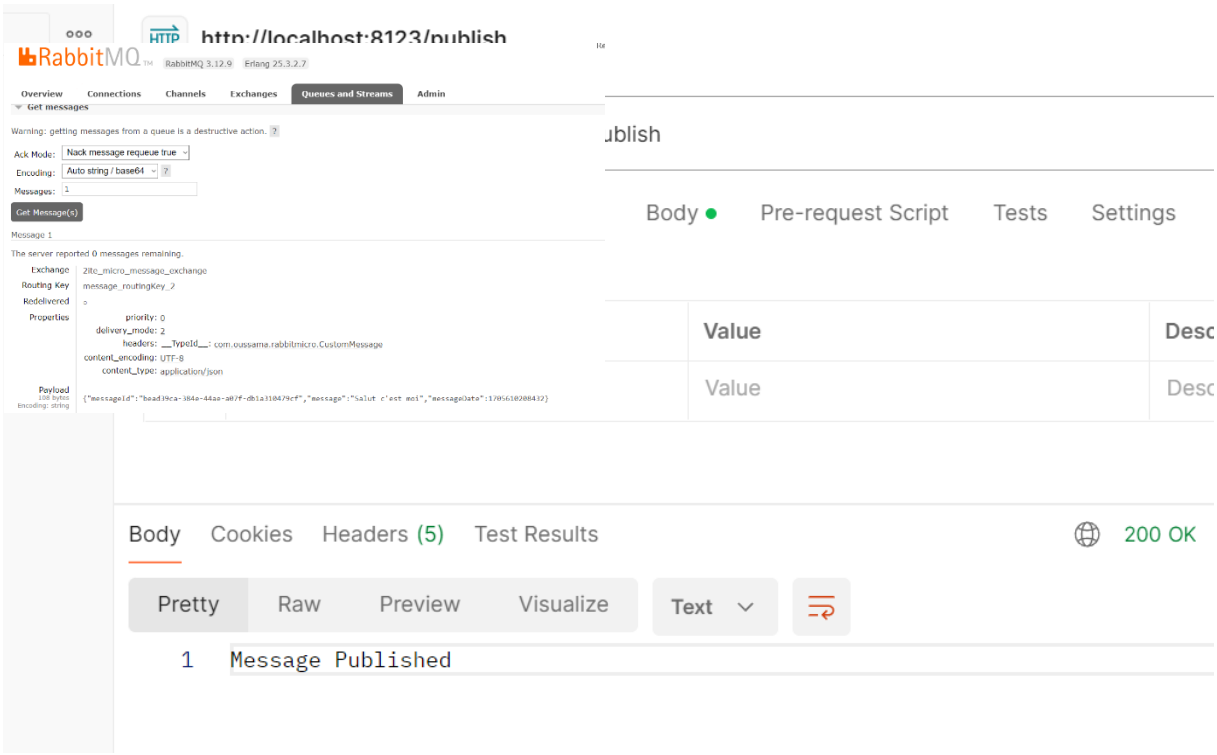
Projet 3 :

- **MessageListner2 :**

```
@Component
public class MessageListener {

    @RabbitListener(queues = MQConfig.QUEUE_1)
    public void handleMessageFromQueue1(CustomMessage message) {

        System.out.println("Consumer 1 - Received message from Queue 1: " +
message);

    }

}
```

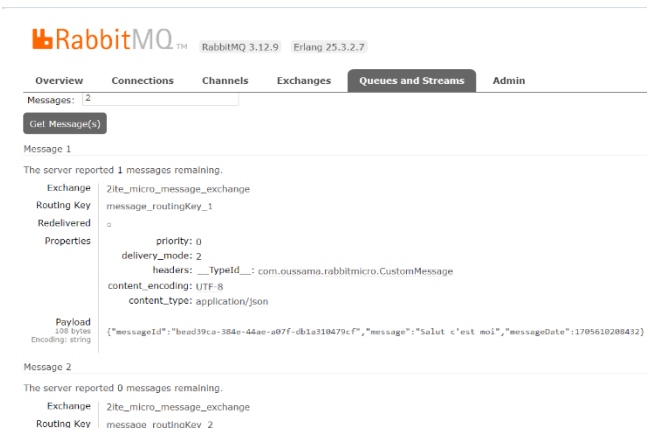
Ce fichier MessageListener **est un composant Spring géré par le conteneur qui agit en tant que consommateur pour la première queue (**MQConfig.QUEUE_1**) dans le système RabbitMQ. Grâce à l'annotation** @RabbitListener**, la méthode** handleMessageFromQueue1 **est configurée pour écouter et traiter les messages de cette queue spécifique. Lorsqu'un message est reçu, la méthode est invoquée, et le contenu du message est affiché dans la console. En résumé, le fichier** MessageListener **agit en tant que consommateur réactif aux messages provenant de la première queue, exécutant des actions spécifiques définies dans la méthode** handleMessageFromQueue1**. Ce modèle de configuration permet à plusieurs consommateurs de réagir à différents événements au sein du système RabbitMQ.**

Execution :

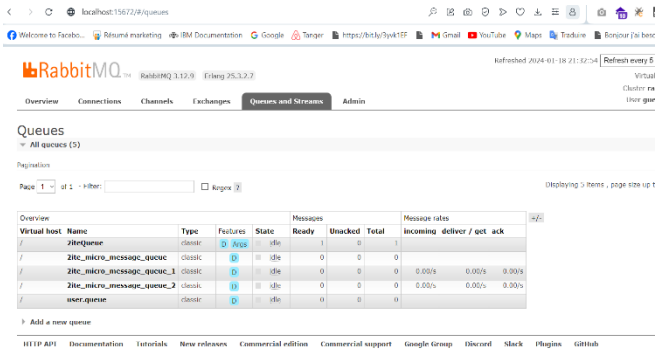


Before running Consumers :

Queue 1 :



Queue 2 :



After running Consumers :

Consummer 1 :

```
2024-01-18 21:29:18.551  INFO 4756 --- [           main] c.o.r.SpringRabbitmqConsumerApplication  : Started SpringRabbitmqConsumerApplication i
Consumer 1 - Received message from Queue 1: CustomMessage(messageId=612a4421-c5a8-40ad-829e-afd11809ad3f, message=Salut c'est moi, messageDate=
Consumer 1 - Received message from Queue 1: CustomMessage(messageId=612a4421-c5a8-40ad-829e-afd11809ad3f, message=Salut c'est moi, messageDate=
```

Consummer 2 :

```
2024-01-18 21:29:18.401  INFO 4756 --- [           main] o.s.a.r.c.CachingConnectionFactory       : Created new connection: rabbitConnectionFact
2024-01-18 21:29:18.551  INFO 4756 --- [           main] c.o.r.SpringRabbitmqConsumerApplication  : Started SpringRabbitmqConsumerApplication i
Consumer 1 - Received message from Queue 1: CustomMessage(messageId=612a4421-c5a8-40ad-829e-afd11809ad3f, message=Salut c'est moi, messageDate=
Consumer 1 - Received message from Queue 1: CustomMessage(messageId=612a4421-c5a8-40ad-829e-afd11809ad3f, message=Salut c'est moi, messageDate=
```