

1 - Projeto de envio de email

Foi criado um projeto de API Rest para envio de e-mail, no seguinte repositório:

<https://github.com/eduardowmu/email-microservice>

2 - Assincronismo com RabbitMQ

Crie uma conta no cloudamqp: <https://customer.cloudamqp.com/>

Crie um team, com um nome qualquer e aceite o termo de uso. A outra opção é opcional. Em seguida, crie uma instância, com plano gratuito, seguindo as orientações a seguir.

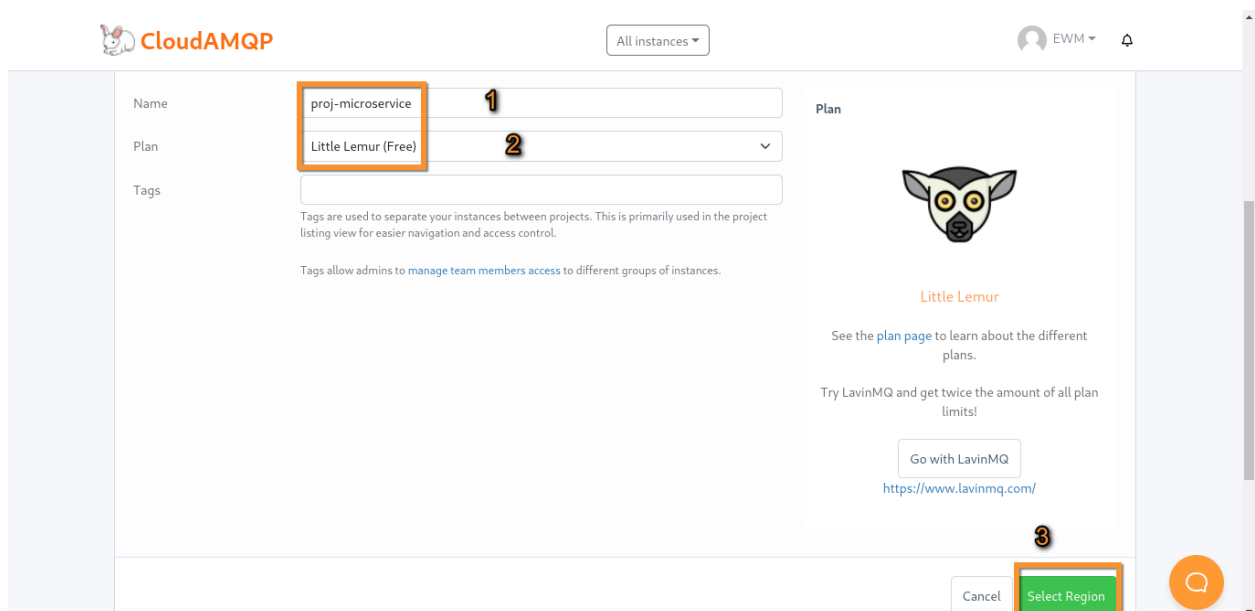
The image is a screenshot of the CloudAMQP web interface. At the top, the CloudAMQP logo is on the left, and a user profile 'EWM' with a bell icon is on the right. Below the header, there's a section for creating a new instance. On the left, there's a sidebar with 'Name', 'Plan', and 'Tags' sections. The 'Name' field contains 'proj-microservice' and is highlighted with an orange box and a '1'. The 'Plan' dropdown menu is set to 'Little Lemur (Free)' and is also highlighted with an orange box and a '2'. The 'Tags' field is empty. To the right of the form, there's a 'Plan' section featuring a lemur illustration, the text 'Little Lemur', and a link to 'Go with LavinMQ' with the URL 'https://www.lavinmq.com/'. At the bottom right, there are 'Cancel' and 'Select Region' buttons. The 'Select Region' button is highlighted with an orange box and a '3'. A '3' is also placed above the 'Select Region' button. The overall layout is clean and modern, with a light blue and white color scheme.

Figura 1

Selecione a seguinte região.

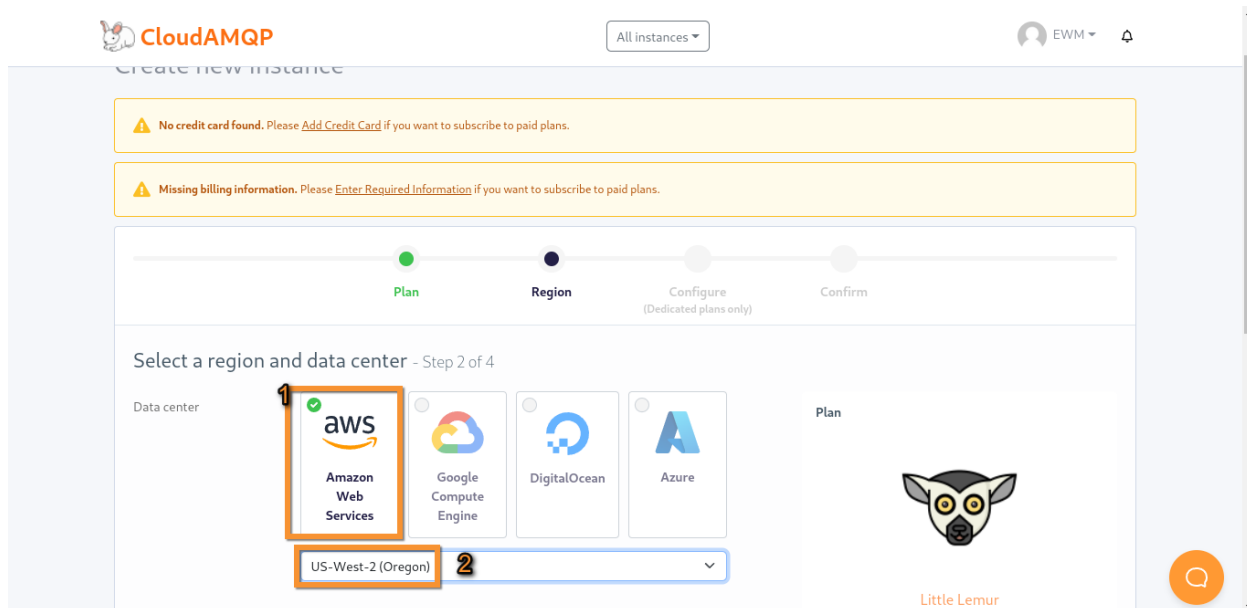


Figura 2

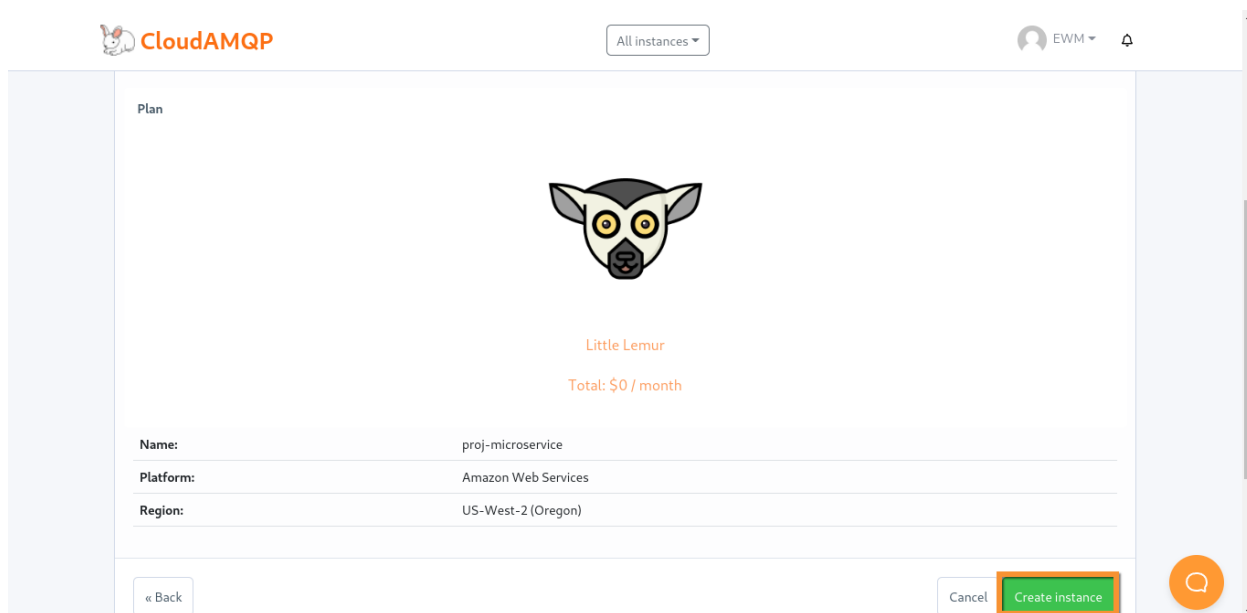


Figura 3

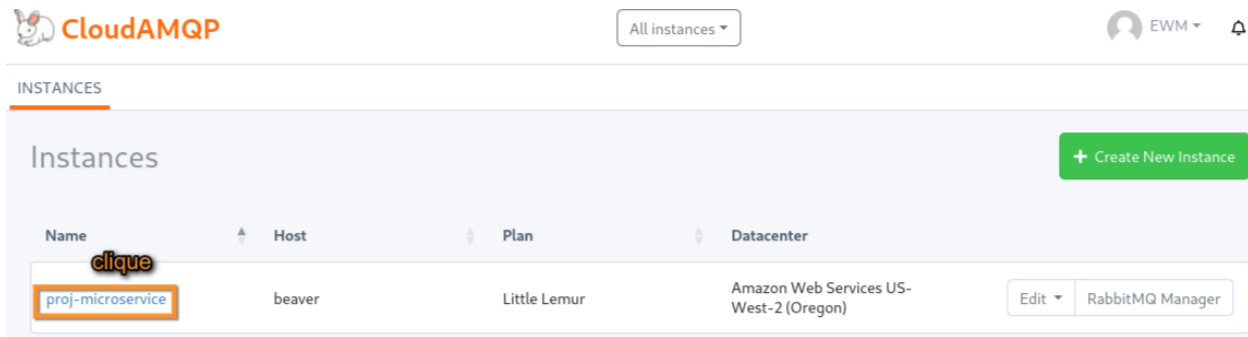


Figura 4

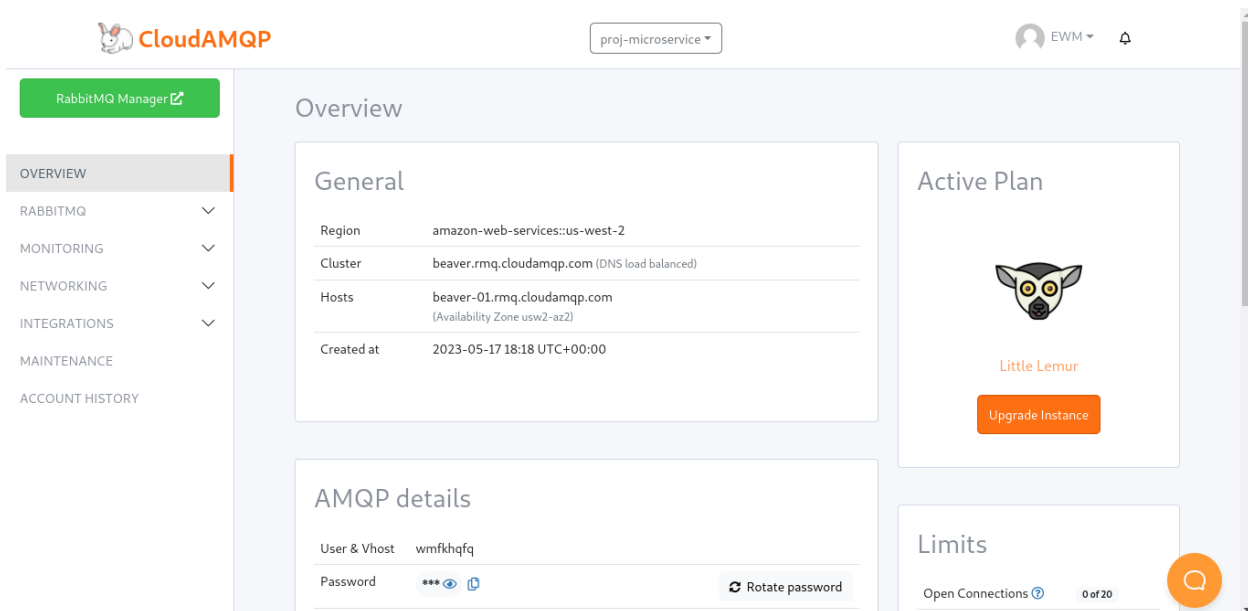


Figura 5 - <https://api.cloudamqp.com/console/11523f8a-3cb6-48ae-aea7-224588a1b15c/details>

Voltemos agora para nosso micro-serviço de envio de e-mail. Antes de iniciarmos nossa aplicação, iremos alterar o tipo do atributo “id” da classe entidade Email. Como iremos trabalhar com vários serviços com transações distribuídas, para não haver conflito, a gente tem que usar um tipo adequado, é para não ter conflitos de recursos entre microserviços.

```
package br.edu.msemail.model;

import br.edu.msemail.statusenum.StatusEmail;

import javax.persistence.*;
import javax.persistence.criteria.From;
import java.io.Serializable;
import java.time.LocalDateTime;
```

```
import java.util.UUID;
```

```
@Entity
```

```
public class Email implements Serializable {  
    private static final long serialVersionUID = 1L;
```

```
    @Id
```

```
    @GeneratedValue(strategy= GenerationType.AUTO)
```

```
    private UUID id;
```

```
    private String ownerRef;
```

```
    private String emailFrom;
```

```
    private String emailTo;
```

```
    private String subject;
```

```
    @Column(columnDefinition = "TEXT", length = 500)
```

```
    private String text;
```

```
    private LocalDateTime sendDateEmail;
```

```
    private StatusEmail statusEmail;
```

```
    public UUID getId() {
```

```
        return id;
```

```
    }
```

```
    public void setId(UUID id) {
```

```
        this.id = id;
```

```
    }
```

```
    public String getOwnerRef() {
```

```
        return ownerRef;
```

```
    }
```

```
    public void setOwnerRef(String ownerRef) {
```

```
        this.ownerRef = ownerRef;
```

```
    }
```

```
    public String getEmailFrom() {
```

```
        return emailFrom;
```

```
    }
```

```
    public void setEmailFrom(String emailFrom) {
```

```
        this.emailFrom = emailFrom;
```

```
    }
```

```
    public String getEmailTo() {
```

```
        return emailTo;
```

```
    }
```

```
    public void setEmailTo(String emailTo) {
```

```
        this.emailTo = emailTo;
```

```
    }
```

```
    public String getSubject() {
```

```
        return subject;
```

```
    }
```

```
    public void setSubject(String subject) {
```

```

    this.subject = subject;
}

public String getText() {
    return text;
}

public void setText(String text) {
    this.text = text;
}

public LocalDateTime getSendDateEmail() {
    return sendDateEmail;
}

public void setSendDateEmail(LocalDateTime sendDateEmail) {
    this.sendDateEmail = sendDateEmail;
}

public StatusEmail getStatusEmail() {
    return statusEmail;
}

public void setStatusEmail(StatusEmail statusEmail) {
    this.statusEmail = statusEmail;
}
}

```

Consequentemente teremos que fazer essa alteração no respectivo repository também.

```

package br.edu.msemail.repository;

import br.edu.msemail.model.Email;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.UUID;

@Repository
public interface EmailRepository extends JpaRepository<Email, UUID> {}

```

Agora iremos preparar nossa aplicação para trabalhar com Spring AMQP, para implementação da mensageria. A primeira coisa é adicionar a dependência.

```

<?xml version="1.0" encoding="UTF-8"?>

```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.11</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>br.edu</groupId>
  <artifactId>ms-email</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ms-email</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

```

</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
          </exclude>
        </excludes>
      </configuration>
    </plugin>
  </plugins>
</build>

</project>

```

Agora iremos adicionar as configurações de conexão do rabbitMQ que geramos na nossa conta que criamos anteriormente, dentro do application.properties. Repare nas duas ultimas linhas, o primeiro é a URL gerada na nossa conta. O segundo é o nome da “fila”, que será consumida pelo serviço de envio de email, que irá ouvir esta fila sempre que chegar mensagens nessa fila, este vai escutar e receber essas mensagens.

```

spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=12345

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

#https://stackoverflow.com/questions/63236701/got-bad-greeting-from-smtp-host-smtp-yandex-ru-port-465-response-eof-wit
#support.google.com/accounts/answer/185833
spring.mail.host=smtp.gmail.com
spring.mail.port=587
#spring.mail.port=465
spring.mail.username=eduwamura@gmail.com
spring.mail.password=uwwkbwjzfijdndtd
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
#spring.mail.properties.mail.smtp.socketFactory.fallback = false
#spring.mail.properties.mail.smtp.ssl.enable = true
#spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.com

```

```
#spring.mail.test-connection= true
```

```
spring.rabbitmq.addresses=amqps://wmfkhqfq:l0R1NYA8xD1yGg2viXOZcrKOI97gQVTe@beaver.rmq.cloudamqp.com/wmfkhqfq  
spring.rabbitmq.queue=ms_email
```

O próximo passo é criar uma classe de configuração para a gente tanto criar a fila e também colocar depois configurações de conversão.

```
package br.edu.msemail.config;
```

```
import org.springframework.amqp.core.Queue;  
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;
```

```
@Bean
```

```
public Queue queue () {  
    /*O parametro booleano "true" serve para que, quando  
    * o rabbitMQ for interrompido, nao iremos perder nem  
    * as mensagens da fila e nem a fila*/  
    return new Queue(queue, true);  
}
```

Agora iremos criar nossa consumer, que será a classe que irá escutar esta fila. Esta classe terá um método que irá escutar a fila de MQ.

```
package br.edu.msemail.consumer;
```

```
import br.edu.msemail.dto.EmailDto;  
import br.edu.msemail.model.Email;  
import br.edu.msemail.service.EmailService;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.beans.BeanUtils;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.messaging.handler.annotation.Payload;  
import org.springframework.stereotype.Component;
```

```
@Component
```

```
public class EmailConsumer {  
    private final EmailService emailService;
```



```

@Autowired
public EmailConsumer(EmailService emailService) {
    this.emailService = emailService;
}

RabbitListener(queues = "${spring.rabbitmq.queue}")
public void listen (@Payload EmailDto emailDto) {
    Email emailModel = new Email();
    BeanUtils.copyProperties(emailDto, emailModel);
    emailService.sendEmail(emailModel);
    System.out.println("Email Status: " + emailModel.getStatusEmail().toString());
}

```

A fila que ele vai estar escutando é a fila que a gente já definiu tanto no *application.properties* quanto no arquivo da classe de RabbitMqConfig.

Além disso, iremos precisar criar um conversor, pois quando o método, que no caso é o ouvinte, de envio de e-mail é chamado, recebe uma mensagem, que no caso é o payload, deve ser convertida. Não iremos usar o conversor padrão do AMQP, pois vamos estar recebendo um tipo EmailDto. Então precisamos configurar este conversor para que consigamos realmente receber este tipo Dto. Faremos isso dentro da classe de configuração

```

package br.edu.msemail.config;

import org.springframework.amqp.core.Queue;
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
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;

    @Bean
    public Queue queue () {
        /*O parametro booleano "true" serve para que, quando
        * o rabbitMQ for interrompido, nao iremos perder nem
        * as mensagens da fila e nem a fila*/
        return new Queue(queue, true);
    }
}

```

```

/*Conversor global*/
@Bean
public Jackson2JsonMessageConverter messageConverter() {
    return new Jackson2JsonMessageConverter();
}

```

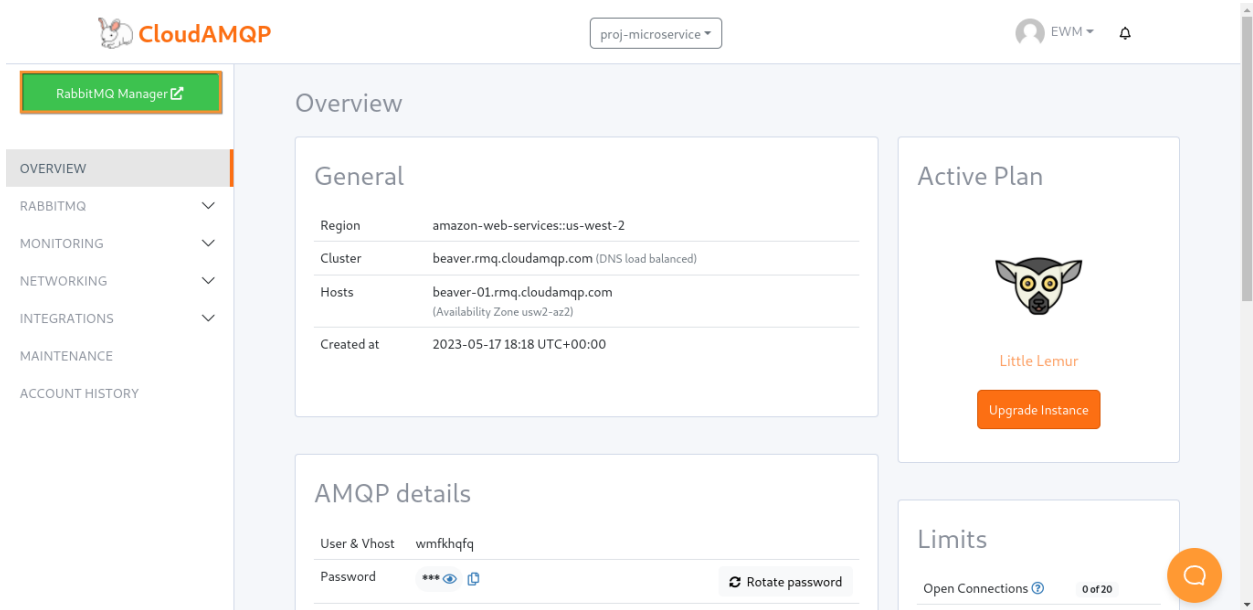


Figura 6

No gerenciador indicado na figura acima, a gente consegue visualizar todas essas métricas, filas e exchanges e tudo mais que foi criado dentro deste broker.

```

2023-05-17 17:17:44.779 INFO 39540 --- [main] o.s.a.r.c.CachingConnectionFactory : Created new connection:
rabbitConnectionFactory#701d2b59:0/SimpleConnection@5341e71a delegate=amqp://wmfkhqfq@35.85.123.167:5671/wmfkhqfq,
localPort= 56572

```

```

2023-05-17 17:17:47.027 INFO 39540 --- [main] br.edu.msemail.MsEmailApplication : Started MsEmailApplication in 19.956
seconds (JVM running for 23.508)

```

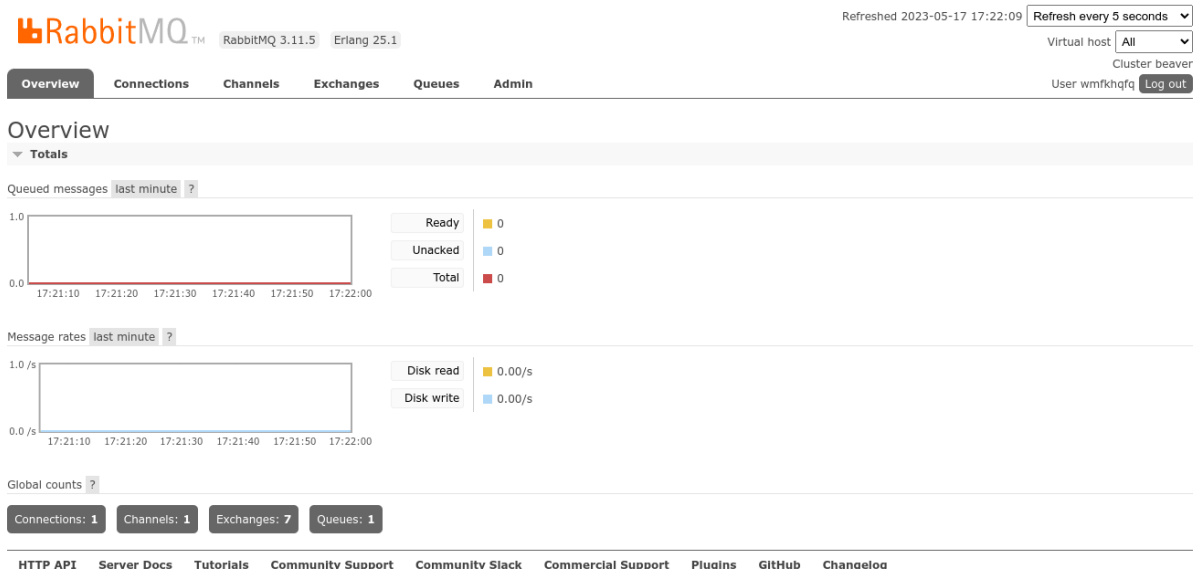


Figura 7

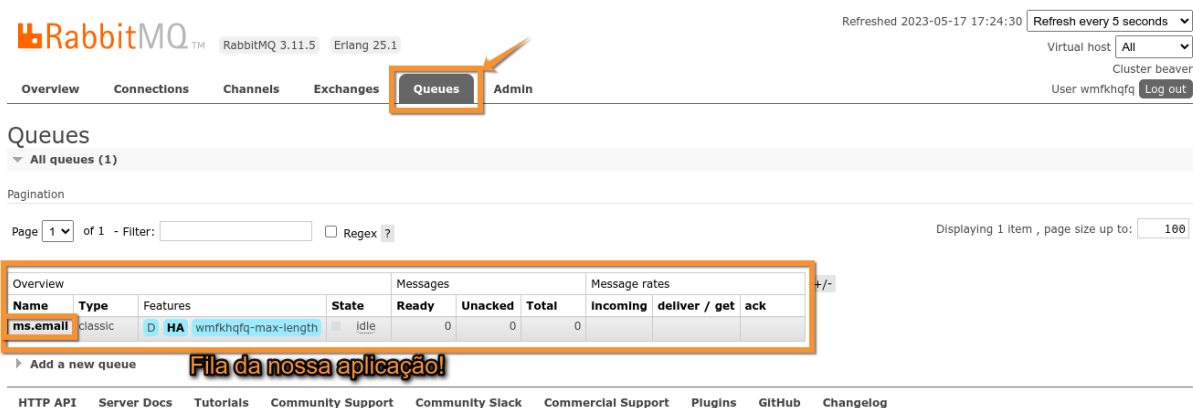


Figura 8

Se clicarmos no ms.email indicado na imagem acima, conseguimos depois ver que temos um consumer conectado e muito mais.

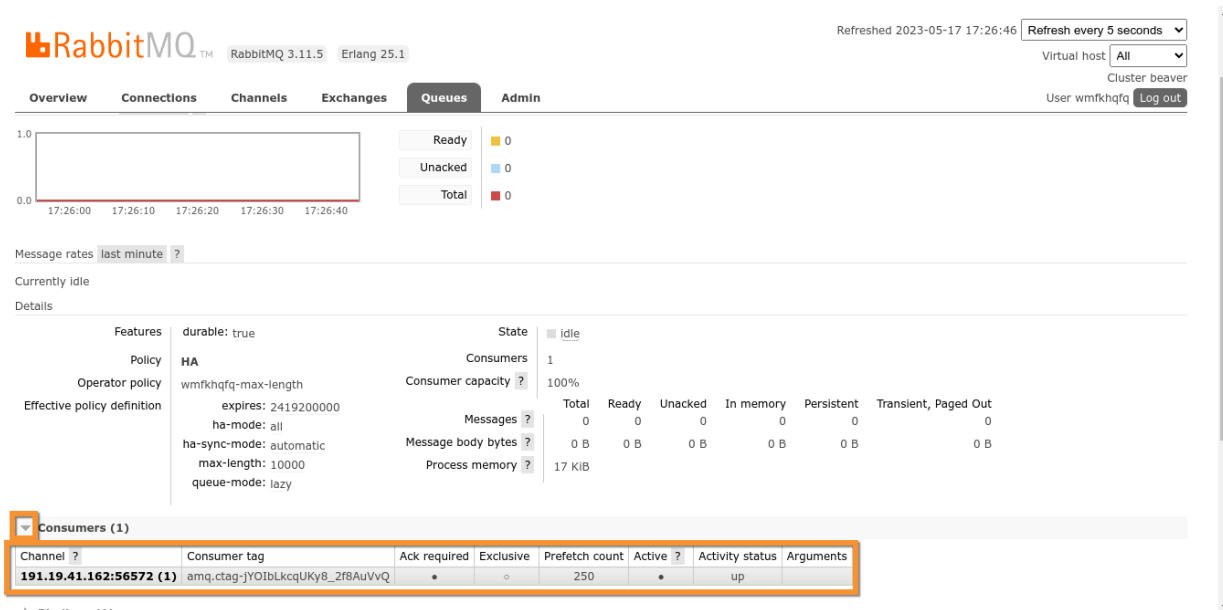


Figura 9

Agora iremos publicar uma mensagem nessa fila, ou seja, simular que a gente é um outro serviço dentro da aplicação que vai enviar uma mensagem para o nosso serviço de email.

The screenshot shows the 'Publish message' form in the RabbitMQ web interface. The 'Payload' field is highlighted with an orange box and contains the following JSON:

```
{
  "ownerRef": "Eduardo",
  "emailFrom": "ewmurakoshi@gmail.com",
  "emailTo": "eduardowmu@yahoo.com.br",
  "subject": "Teste do microserviço de envio de email, via mensageria",
  "text": "Testando o microserviço de email"
}
```

The 'Payload encoding' is set to 'String (default)'. The 'Publish message' button is visible at the bottom of the form.

Figura 10

A mensagem foi o mesmo body que enviamos via post para envio de email, com pequenas alterações no título. Veremos se nosso micro serviço de email vai receber essa mensagem.

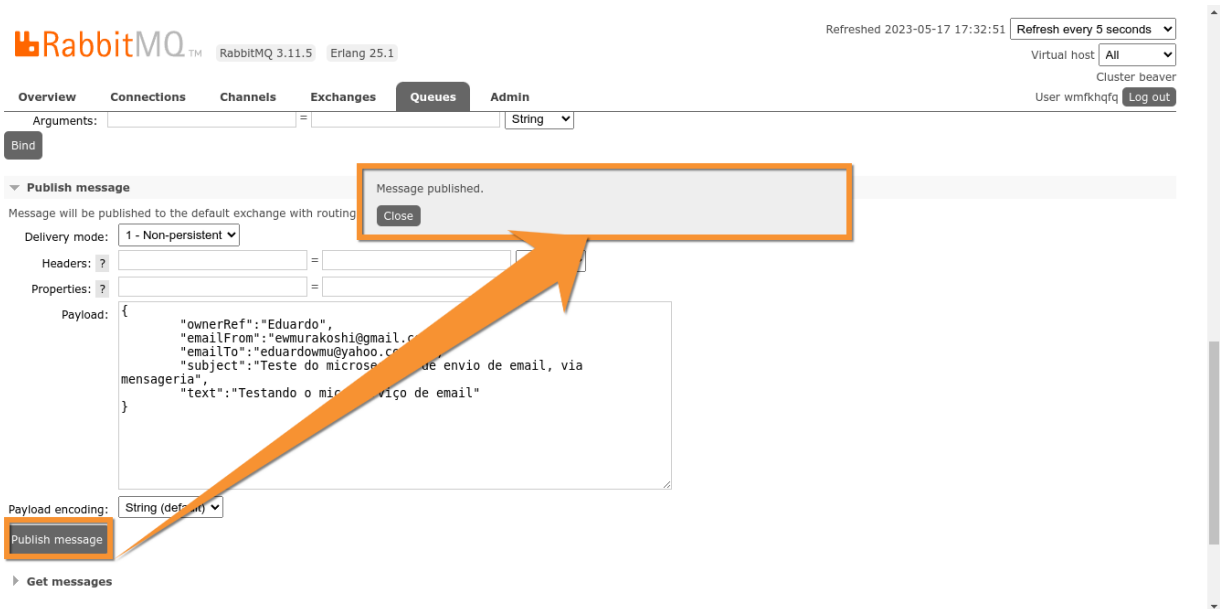


Figura 11

```
2023-05-17 17:17:47.027 INFO 39540 --- [      main] br.edu.msemail.MsEmailApplication      : Started MsEmailApplication in 19.956 seconds (JVM running for 23.508)
```

```
Hibernate: insert into email (email_from, email_to, owner_ref, send_date_email, status_email, subject, text, id) values (?, ?, ?, ?, ?, ?, ?, ?)
```

```
Email Status: SENT
```

Referências:

<https://www.youtube.com/watch?v=V-PqR0BxA8c&list=PL8ilphQOyG-Dp037UnFG0x8aduelvZZWE&index=2>

<https://www.youtube.com/watch?v=V-PqR0BxA8c&list=PL8ilphQOyG-Dp037UnFG0x8aduelvZZWE&index=3>

<https://docs.spring.io/spring-amqp/docs/current/reference/html/>