

Workshop de Práticas

Portaria Digital: Aplicações com microserviços

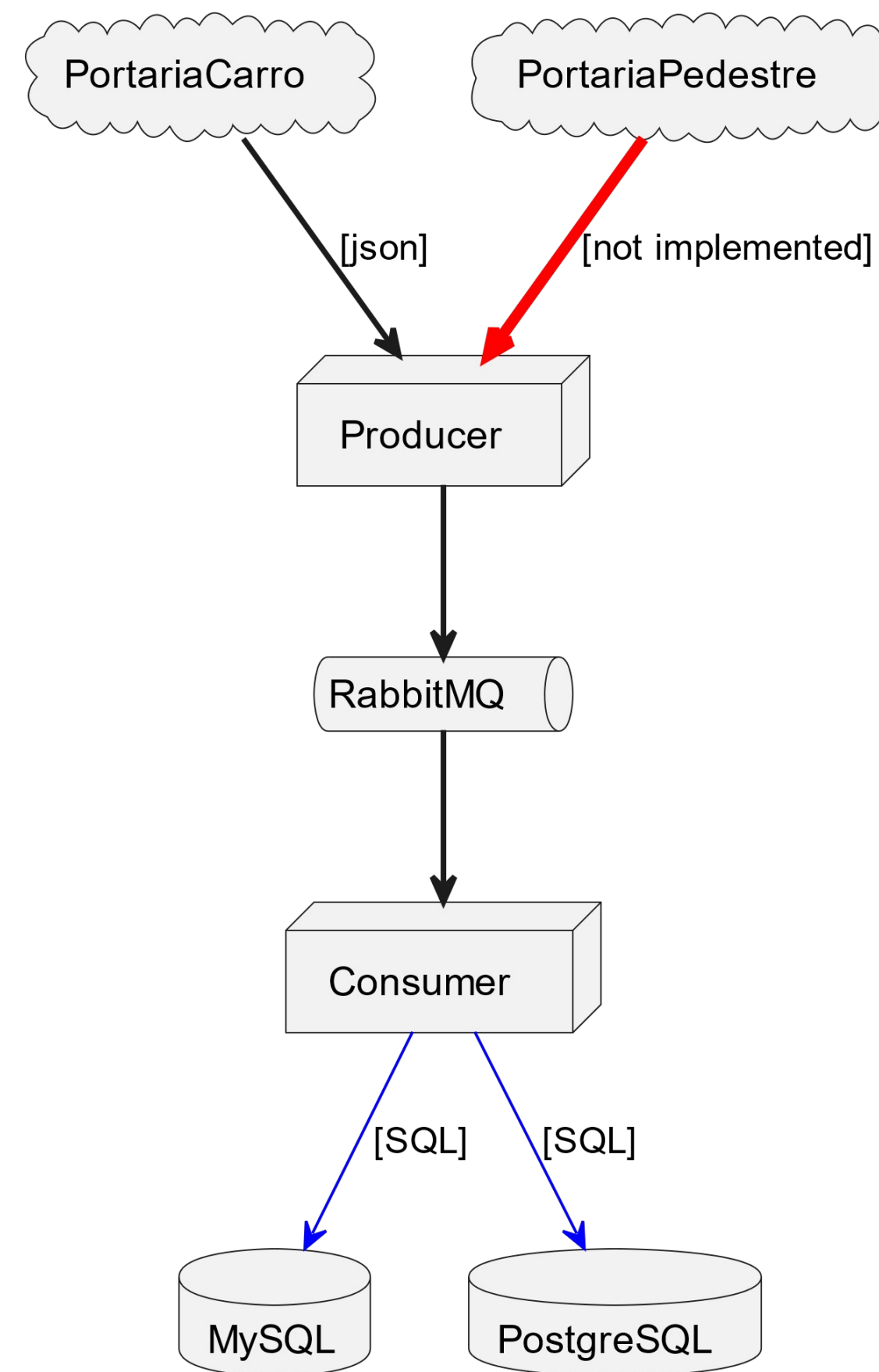
José William Pinto Gomes

Orlando Saraiva do Nascimento Junior

Os microserviços foram introduzidos pela primeira vez em 2014 por Lewis e Fowler em seu famoso blog. Eles definem um estilo arquitetônico para o desenvolvimento de aplicações como suítes de (micro)serviços pequenos e independentes.

Visão Global

- 01 Aplicativo para Portaria: Registro de placas dos carros
- 02 Consumo de API via json e enfileiramento.
- 03 Consumo do enfileiramento e gravação em dois SGBDs diferentes.



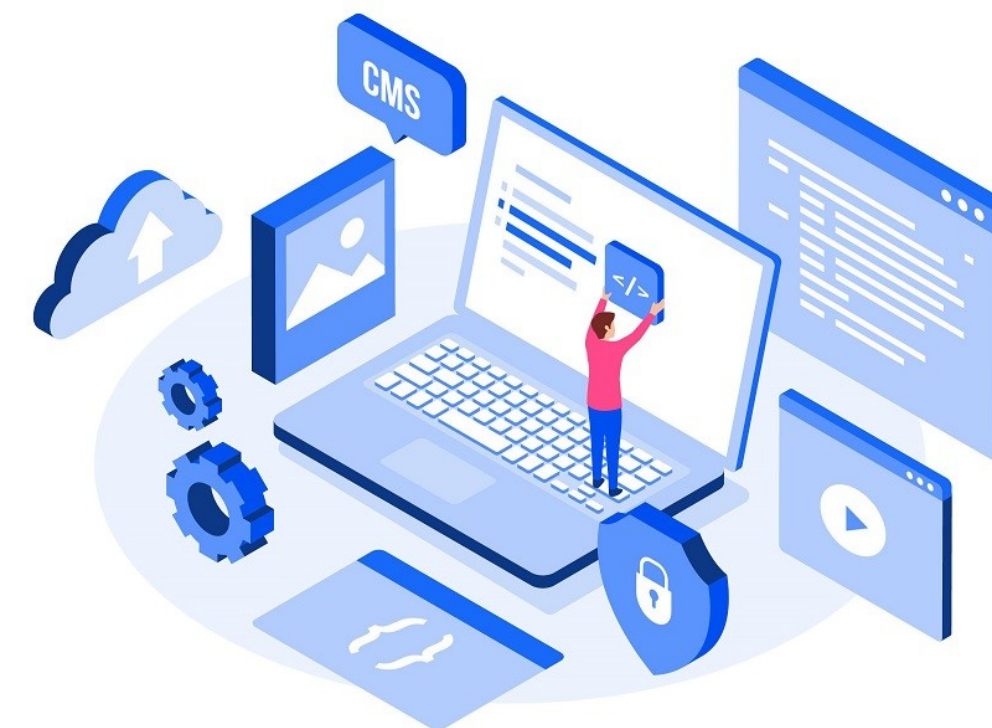
Um microserviço pode ser analisado, compreendido e atualizado sem saber nada sobre os aspectos internos dos outros microserviços em uma arquitetura.

Cada microserviço é construído em torno de uma capacidade de negócios, é executado em seu próprio processo e se comunica com outros microserviços em um aplicativo por meio de mecanismos leves (por exemplo, APIs HTTP).

DESENVOLVIMENTO DA API

API - Registro da entrada de Carros

PRINCIPAIS RECURSOS



Visual Studio Code

IDE de desenvolvimento



Extension Pack for Java

226ms

Popular extensions for Java development...

Microsoft

Pacote da linguagem Java 1



Spring Boot Extension Pack

A collection of extensions for developing...

VMware

Pacote do framework Spring Boot 3.1.3



Docker

Makes it easy to create, manage, and de...

Microsoft

Extensão para gerar a imagem do Docker

API - Registro da entrada de Carros

CLASSE MAIN

```
1 package br.com.portalz.condominio;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class CondominioApplication {  
8  
9     Run | Debug  
10    public static void main(String[] args) {  
11        SpringApplication.run(CondominioApplication.class, args);  
12    }  
13 }
```

APLICAÇÃO EM JAVA

USANDO SPRING BOOT

API - Registro da entrada de Carros

application.properties

```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.datasource.url=jdbc:mysql://192.168.15.199:3306/condominio
3 spring.datasource.username=root
4 spring.datasource.password=condominio
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

**Planejamento para que o
MySQL esteja em um Docker
no servidor**

CONEXÃO COM BANCO DE DADOS

SPRING BOOT / HIBERNATE

COMUNICAÇÃO INTERNA, SEM ACESSO EXTERNO AO BANCO
DE DADOS

API - Registro da entrada de Carros

pom.xml

```
11 <groupId>br.com.portalz</groupId>
12 <artifactId>condominio</artifactId>
13 <version>0.0.1-SNAPSHOT</version>
14 <name>condominio</name>
15 <description>Projeto de API utilizando Spring Boot</description>
16 <properties>
17     <java.version>17</java.version>
18     <start-class>br.com.portalz.condominio.CondominioApplication</start-class>
19 </properties>
```

PACKAGE NAME + VERSÃO DO PROJETO

VERSÃO MINIMA JAVA -> 17

AUTO INICIALIZAÇÃO DA CLASSE PARA O DOCKER

API - Registro da entrada de Carros

entity -> Acesso.java

CRIAÇÃO DA ENTIDADE -> BIBLIOTECA JAKARTA

GETs E SETs USANDO A BIBLIOTECA LOMBOK

ESTRUTURA DOS DADOS -> QUALQUER BANCO DE DADOS

```
1 package br.com.portalz.condominio.entity;
2
3 import java.io.Serializable;
4 import java.sql.Date;
5 import java.sql.Time;
6
7 import jakarta.persistence.Column;
8 import jakarta.persistence.Entity;
9 import jakarta.persistence.GeneratedValue;
10 import jakarta.persistence.GenerationType;
11 import jakarta.persistence.Id;
12 import lombok.Getter;
13 import lombok.Setter;
14
15 @Getter
16 @Setter
17 @Entity(name = "Acesso")
18 public class Acesso implements Serializable{
19     private static final long serialVersionUID = 1L;
20
21     @Id
22     @GeneratedValue(strategy = GenerationType.AUTO)
23     private Long id;
24
25     @Column(nullable = false)
26     private String placa;
27
28     private String portaria;
29     private String tipo;
30     private Date data;
31     private Time hora;
32
33     public Acesso() {
34     }
35 }
```

**Incremento na Chave Primária
utilizando GenerationType.AUTO
para permitir a mudança de
Banco de Dados**

API - Registro da entrada de Carros

Repositório -> AcessoRepository.java

```
1 package br.com.portalz.condominio.repository;  
2  
3 import org.springframework.data.jpa.repository.JpaRepository;  
4  
5 import br.com.portalz.condominio.entity.Acesso;  
6  
7 public interface AcessoRepository extends JpaRepository<Acesso, Long> {  
8  
9     java.util.List<Acesso> findByPlaca(String placa);  
10 }
```

REPOSITÓRIO

IMPLEMENTAÇÃO DO MÉTODO DE PESQUISA ATRAVÉS DE PARÂMETRO INFORMADO

API - Registro da entrada de Carros

Implementação da API -> AcessoREST.java

```
1 package br.com.portalz.condominio;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PathVariable;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestBody;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import br.com.portalz.condominio.repository.AcessoRepository;
12 import br.com.portalz.condominio.entity.Acesso;
13
14 @RestController
15 @RequestMapping("/acesso")
16 public class AcessoREST {
17     @Autowired
18     private AcessoRepository repositorio;
```

PROTOCOLO REST DO
FRAMEWORK SPRING BOOT

MAPEAMENTO PARA
FUNCIONAMENTO DA API

<domínio>/acesso

API - Registro da entrada de Carros

Implementação da API -> AcessoREST.java

```
20 @GetMapping("/{placa}")
21 public java.util.List<Acesso> listar(@PathVariable String placa){
22
23     return repositorio.findByPlaca(placa);
24 }
25
26 @GetMapping("/all")
27 public java.util.List<Acesso> listar(){
28
29     return repositorio.findAll();
30 }
```

Implementação de dois recursos para o consumo de dados, retornando um json

<domínio>/acesso/{placa} -> retorna todos os registros filtrados por placa

<domínio>/acesso/all -> retorna todos os registros

API - Registro da entrada de Carros

Implementação da API -> AcessoREST.java

```
32  @PostMapping
33  public void salvar(@RequestBody Acesso acesso){
34      long millis=System.currentTimeMillis();
35
36      java.sql.Date hoje = new java.sql.Date(millis);
37      java.sql.Time hora = new java.sql.Time(millis);
38
39      acesso.setData( hoje );
40      acesso.setHora( hora );
41
42      repositorio.save(acesso);
43  }
44 }
```

Implementação do recurso de inserção dos dados enviados para a API

O recurso recebe todos os dados do acesso, porém, a data e a hora são alteradas para as respectivas do servidor, não sendo possível manipular data/hora do registro

API - Registro da entrada de Carros

Configurações de criação da imagem do Docker -> Dockerfile

```
1 FROM openjdk:22-slim
2
3 ARG JAR_FILE=target/*.jar
4
5 COPY ${JAR_FILE} /tmp/app.jar
6
7 EXPOSE 8082
8
9 ENTRYPOINT ["java", "-jar", "/tmp/app.jar"]
```

Imagem criada para embarcar todos os recursos necessários para funcionamento a aplicação

Linux + Servidor Web Apache TomCat (Framework Spring Boot)

Java JDK versão 22 slim escolhida para o Docker, visando executar a aplicação em versão posterior à criada (teste de compatibilidade/upgrade)

API - Registro da entrada de Carros

Criação da imagem para o Docker

Através do terminal do VS Code

```
& "c:\Dados\condominio\mvnw.cmd" package -f "c:\Dados\condominio\pom.xml"
```

API - Registro da entrada de Carros

Publicação da imagem do Docker

The screenshot shows the Docker Desktop application window. The left sidebar contains navigation options: Containers, Images, Volumes, Dev Environments (BETA), Docker Scout, and Learning center. The main area is titled 'Images' and shows a list of local images. A table lists the image 'jwpgoes/condominio' with tag 'latest', status 'Unused', created '11 days ago', and size '779.86 MB'. A red box highlights the 'Actions' menu icon (three vertical dots) for this image. To the right, a red box highlights the 'Push to Hub' option in the expanded 'Actions' menu, which also includes 'View packages and CVEs' and 'Pull'.

Name	Tag	Status	Created	Size	Actions
jwpgoes/condominio	latest	Unused	11 days ago	779.86 MB	⋮

Showing 1 item

RAM 2.08 GB CPU 0.00% Signed in v4.23.0 2

Actions

- View packages and CVEs
- Pull
- Push to Hub

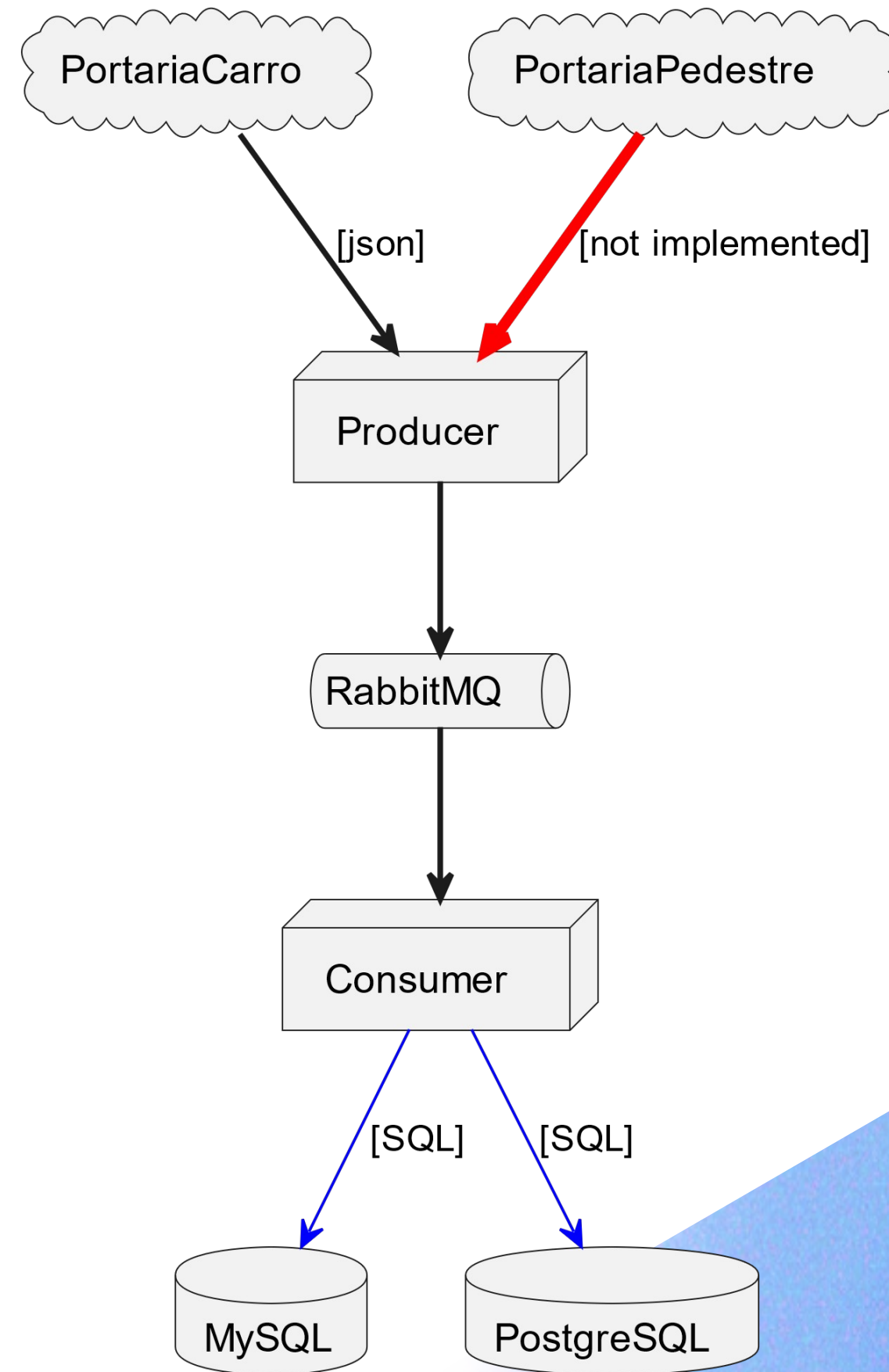
API - Registro da entrada de Carros

Instalação da aplicação no servidor

```
sudo docker pull jwpgomes/condominio
```

```
sudo docker run -d --name app-cond-carro --ip 192.168.15.198 -p  
8082:8082 jwpgomes/condominio
```

CONSUMO DA API



Configurar o **servidor teste**

```
git clone https://github.com/orlandosaraivajr/doutorado_arquitetura_software.git  
cd doutorado_arquitetura_software/micro_django/  
virtualenv -p python3 venv  
source venv/bin/activate  
pip install -r requirements.txt  
python micro_django.py runserver
```


Configurar o **servidor teste**

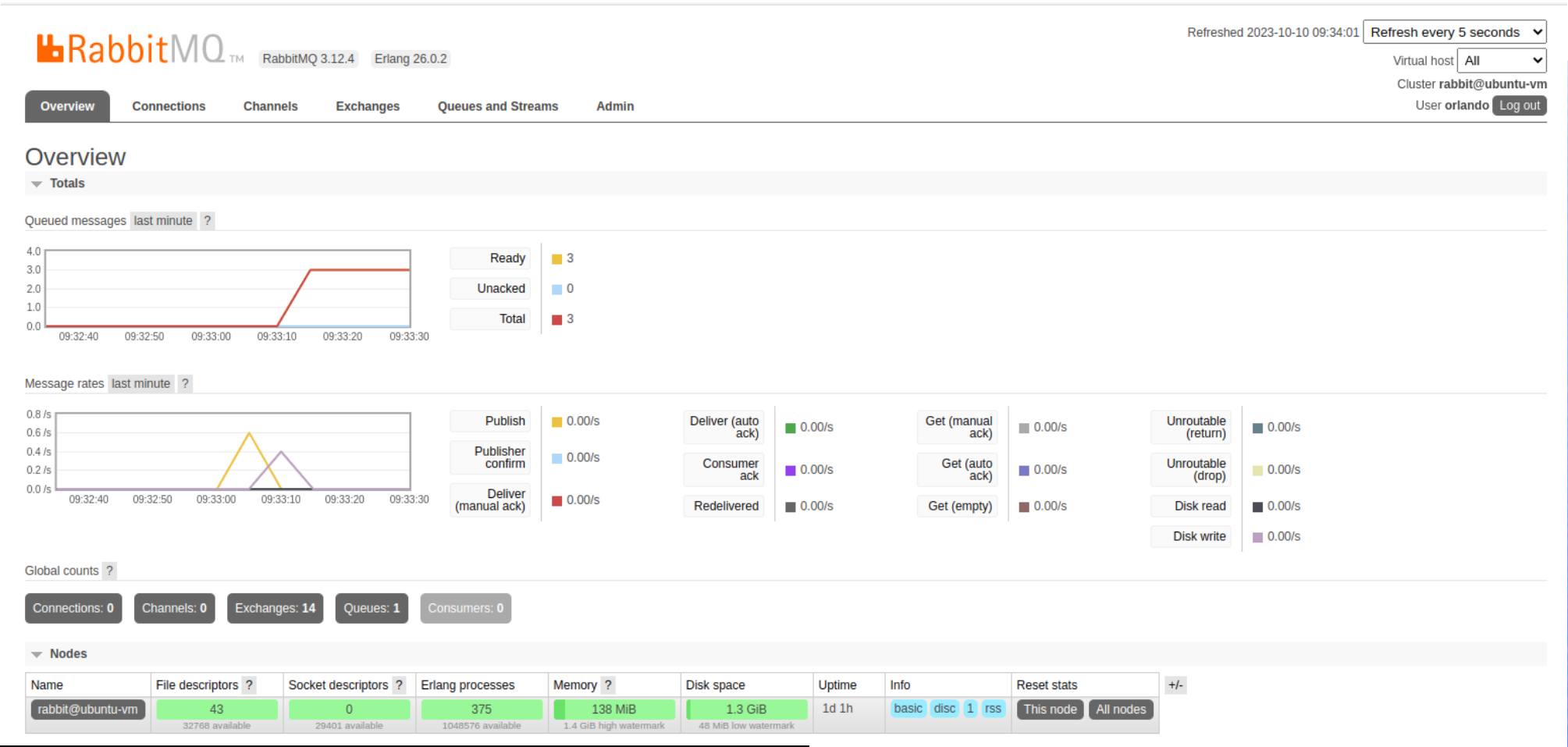
```
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
October 10, 2023 - 09:30:57  
Django version 4.2.5, using settings None  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```


Configurar o **cliente** em um ambiente com acesso a um servidor RabbitMQ, um servidor SGBD MySQL e um servidor SGBD PostgreSQL instalado e configurado:

```
git clone https://github.com/orlandosaraivajr/doutorado_arquitetura_software.git
cd doutorado_arquitetura_software/cliente/
virtualenv -p python3 venv
source venv/bin/activate
pip install -r requirements.txt
cp env .env
nano .env ( não use o editor vi. É sério ! )
python producer.py
python consumer.py
```

Configurar o **cliente** em um ambiente com acesso a um servidor RabbitMQ, um servidor SGBD MySQL e um servidor SGBD PostgreSQL instalado e configurado:

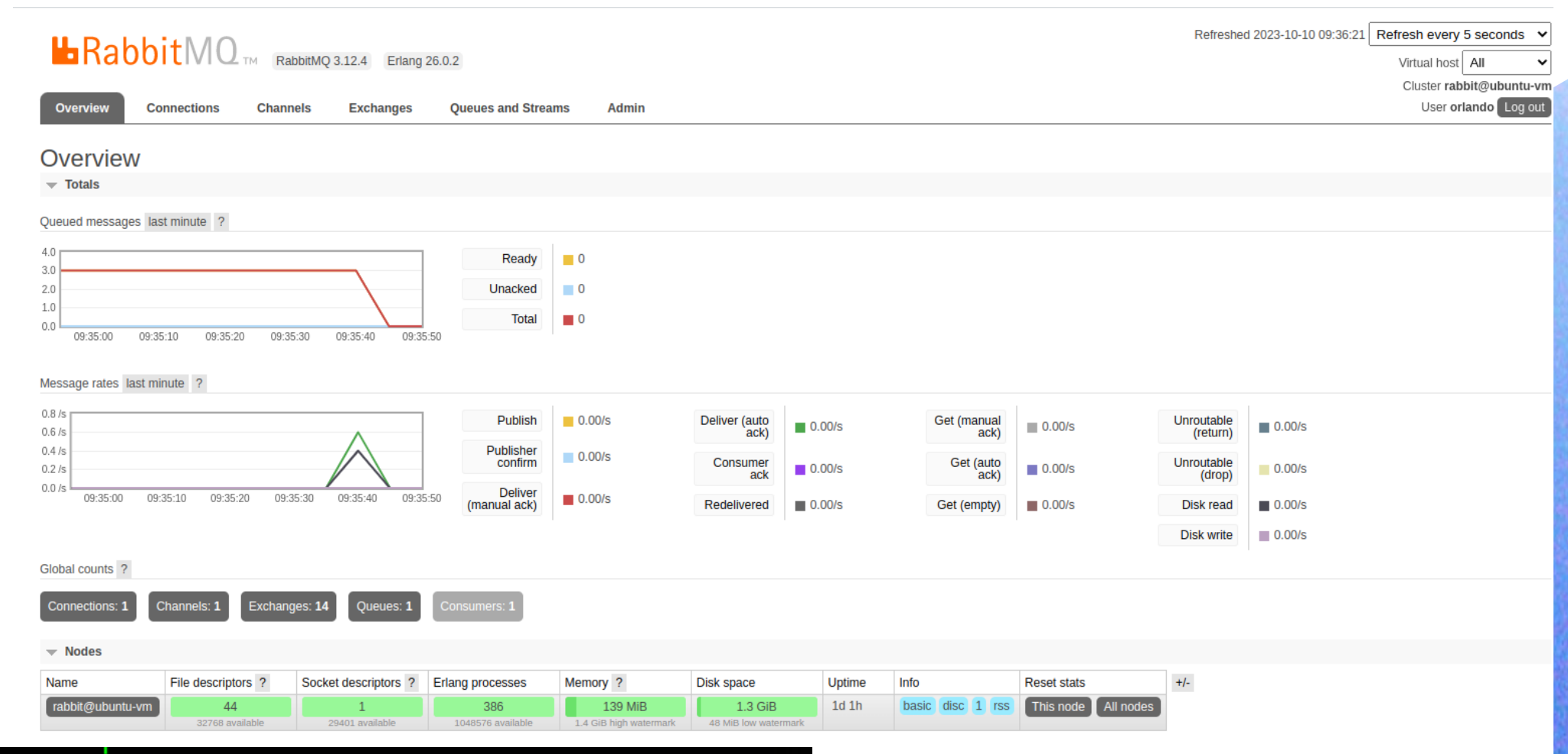
python producer.py



```
(venv) orlando@piracicaba:/tmp/doutorado_arquitetura_software/cliente$ python producer.py
[x] Enfileirado '200|GIN5839|1|S|2023-09-20|00:50:20| '
[x] Enfileirado '201|ABC1234|1|E|2023-09-20|00:45:00| '
[x] Enfileirado '202|ABC1234|1|S|2023-09-20|00:45:45| '
(venv) orlando@piracicaba:/tmp/doutorado_arquitetura_software/cliente$
```


Configurar o **cliente** em um ambiente com acesso a um servidor RabbitMQ, um servidor SGBD MySQL e um servidor SGBD PostgreSQL instalado e configurado:

`python consumer.py`



```
(venv) orlando@piracicaba:/tmp/doutorado_arquitetura_software/cliente$ python consumer.py
Consumindo mensagens. Para sair pressione CTRL+C
[x] Recebido b'200|GIN5839|1|S|2023-09-20|00:50:20|'
[x] Recebido b'201|ABC1234|1|E|2023-09-20|00:45:00|'
[x] Recebido b'202|ABC1234|1|S|2023-09-20|00:45:45|'
```


Conferindo os dados nos SGBD **MySQL** e **PostgreSQL**

The screenshot displays the DBeaver 23.2.1 interface. The top menu bar includes Arquivo, Editar, Navegar, Procurar, Editor SQL, Banco de dados, Janela, and Ajuda. The toolbar contains icons for SQL, Commit, Rollback, and other database operations. The left sidebar shows the 'Navegador de banco de dados' (Database Navigator) with a tree view of databases. The main editor area shows a script with the following SQL commands:

```
SELECT * FROM pg_catalog.pg_tables WHERE schemaname != 'pg_catalog' AND schemaname != 'information_schema';  
  
select * from Carros  
  
delete from Carros;
```

Below the script, a table view for 'carros 1' is shown. The table has the following columns: id local, placa, portaria, tipo, data, and hora. The data is as follows:

id local	placa	portaria	tipo	data	hora
200	GIN5839	1	S	2023-09-20	00:50:20
201	ABC1234	1	E	2023-09-20	00:45:00
202	ABC1234	1	S	2023-09-20	00:45:45

The bottom status bar indicates '3 linha(s) recuperada(s) - 4ms, em 2023-10-10 às 09:39:57'. The bottom right corner shows the connection details: 'Portaria MySQL' and 'pt_BR'.

Configurar o **cliente** para o acessar do **servidor principal**:
Editar o arquivo .env

nano .env

```
GNU nano 4.8                               .env                               Modified
# MySQL credentials
DB_NAME='portaria'
DB_USER='root'
DB_PASSWORD='SENHA'
DB_HOST='10.0.0.151'
# PostgreSQL credentials
PG_NAME='portaria'
PG_USER='orlando'
PG_PASSWORD='SENHA'
PG_HOST='10.0.0.151'
# RabbitMQ credentials
USERNAME='orlando'
PASSWORD='SENHA'
HOSTNAME='10.0.0.151'
VIRTUAL_HOST_SERVER='fila_carros'
QUEUE='hello'
# PORT=5672
# URL credentials
URL='http://masterdev.ddns.net:8082/acesso/all'
# URL='http://127.0.0.1:8000/'

^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify       ^C Cur Pos       M-U Undo
^X Exit          ^R Read File     ^\ Replace       ^U Paste Text    ^T To Spell     ^_ Go To Line     M-E Redo
```


Aplicação dos 12 Fatores

A aplicação doze-fatores é uma metodologia para construir softwares-como-serviço SaaS. A metodologia pode ser aplicada a aplicações escritas em qualquer linguagem de programação, e que utilizem qualquer combinação de serviços de suportes (banco de dados, filas, cache de memória, etc).

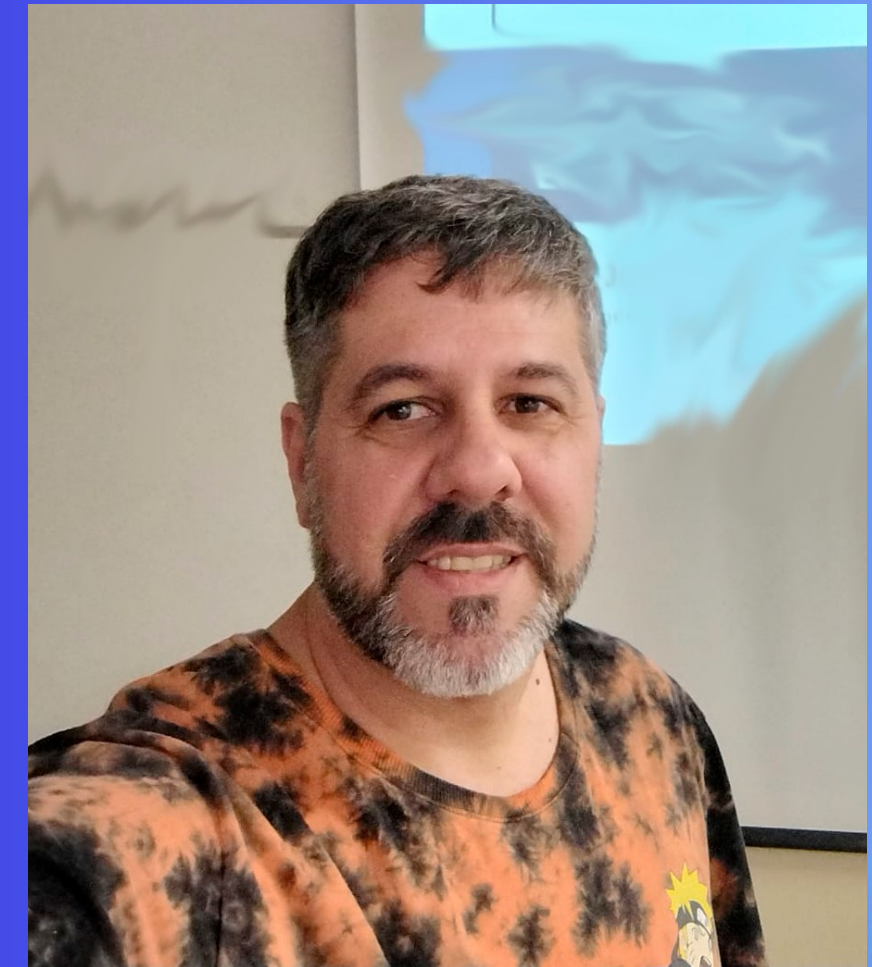
III. Configurações

A configuração de uma aplicação é tudo o que é provável variar entre deploys (homologação, produção, ambientes de desenvolvimento, etc).

A prova de fogo para saber se uma aplicação tem todas as configurações corretamente consignadas fora do código é saber se a base de código poderia ter seu código aberto ao público a qualquer momento, sem comprometer as credenciais.

Workshop de Práticas

José William Pinto Gomes



Orlando Saraiva Jr