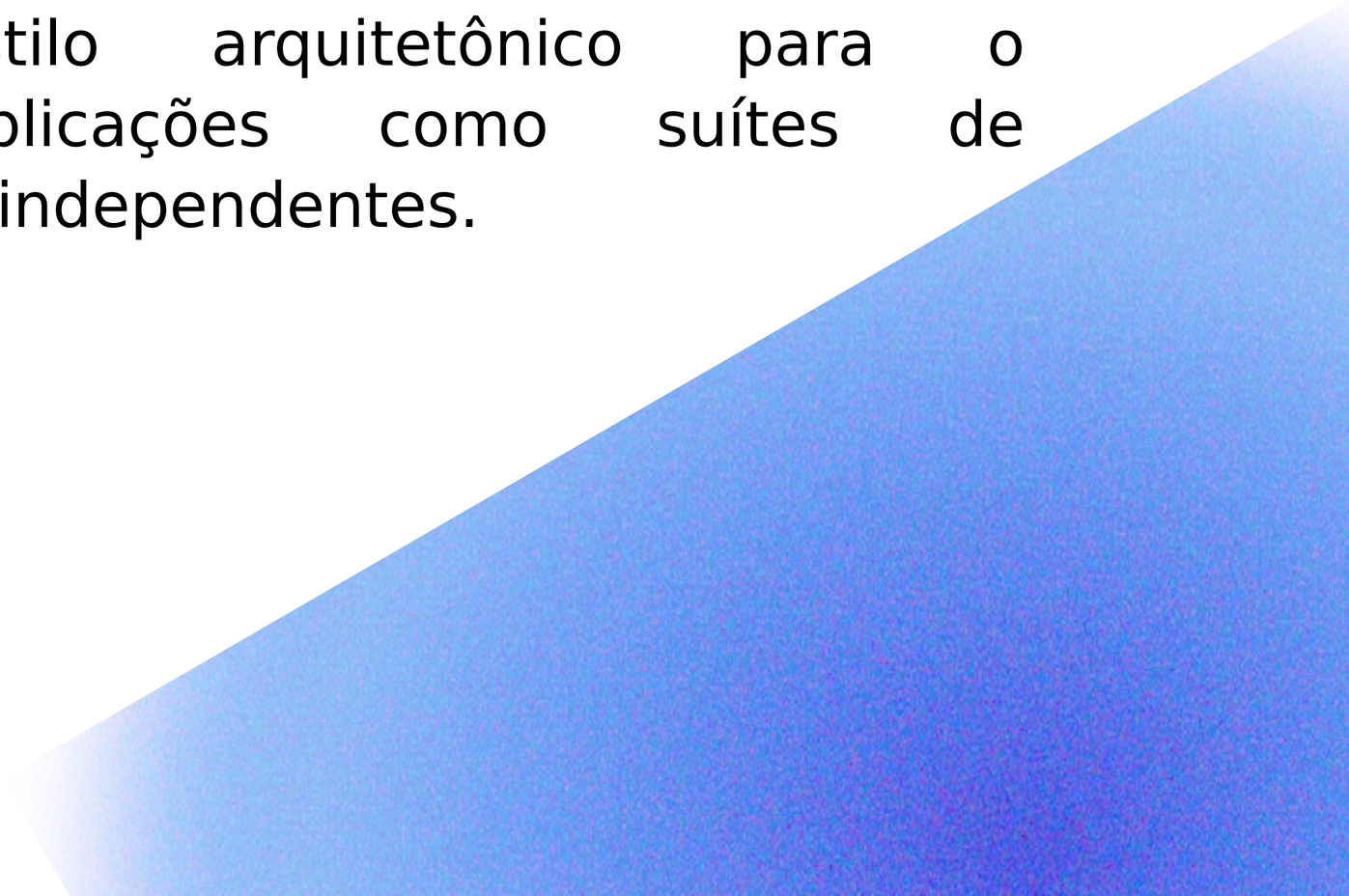
Workshop de Práticas

Portaria Digital: Aplicações com microserviços

José William Pinto Gomes

Orlando Saraiva do Nascimento Junior

Os microsserviços 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.



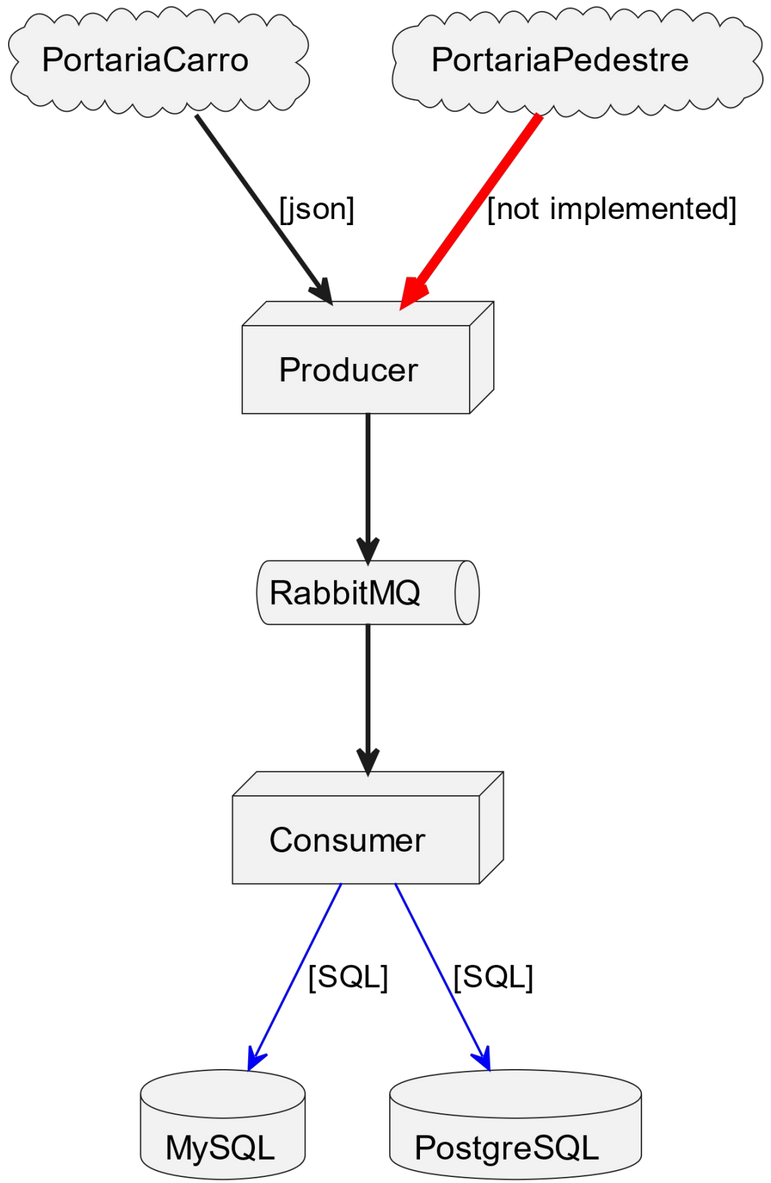
Workshop de Práticas

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.



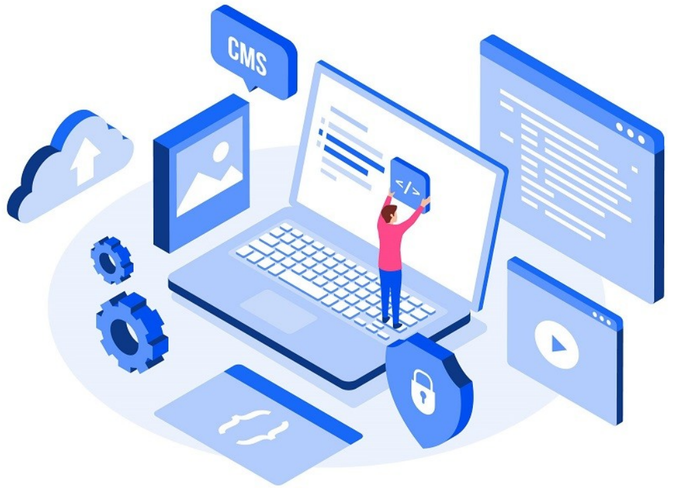
Um microsserviço pode ser analisado, compreendido e atualizado sem saber nada sobre os aspectos internos dos outros microsserviços em uma arquitetura.

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

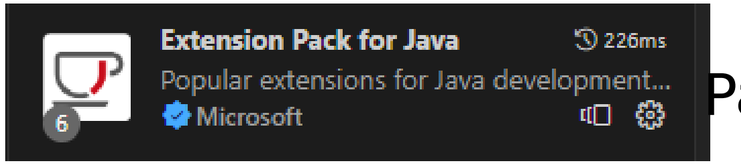
DESENVOLVIMENTO DA API

API - Registro da entrada de Carros

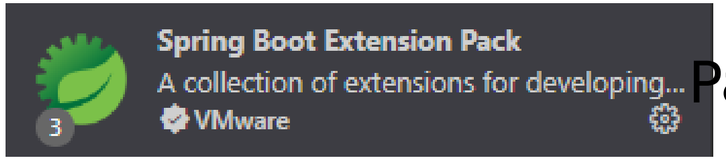
PRINCIPAIS RECURSOS



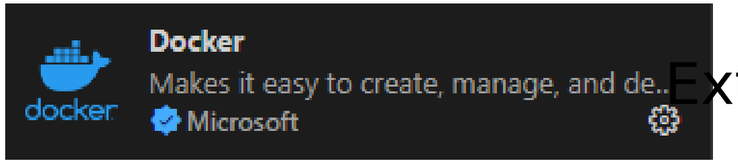
IDE de desenvolvimento



Pacote da linguagem Java 1



Pacote do framework Spring Boot 3.1.3



Extensão para gerar a imagem do Docker

API - Registro da entrada de Carros

CLASSE MAIN

```java
package br.com.portalz.condominio;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CondominioApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(CondominioApplication.class, args);
    }

}
```

APLICAÇÃO EM JAVA

USANDO SPRING BOOT

API - Registro da entrada de Carros

application.properties

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://192.168.15.199:3306/condominio
spring.datasource.username=root
spring.datasource.password=condominio
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

```java
package br.com.portalz.condominio.entity;

import java.io.Serializable;
import java.sql.Date;
import java.sql.Time;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity(name = "Acesso")
public class Acesso implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(nullable = false)
    private String placa;

    private String portaria;
    private String tipo;
    private Date data;
    private Time hora;

    public Acesso() {
    }
}
```

entity -> Acesso.java

CRIAÇÃO DA ENTIDADE -> BIBLIOTECA JAKARTA

GETs E SETs USANDO A BIBLIOTECA LOMBOK

ESTRUTURA DOS DADOS -> QUALQUER BANCO DE DADOS

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

API - Registro da entrada de Carros

Repositório -> AcessoRepository.java

```java
package br.com.portalz.condominio.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import br.com.portalz.condominio.entity.Acesso;

public interface AcessoRepository extends JpaRepository<Acesso, Long> {

    java.util.List<Acesso> findByPlaca(String placa);
}
```

REPOSITÓRIO

IMPLEMENTAÇÃO DO MÉTODO DE PESQUISA ATRAVÉS DE PARÂMETRO INFORMADO

API - Registro da entrada de Carros

Implementação da API -> AcessoREST.java

```java
package br.com.portalz.condominio;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.portalz.condominio.repository.AcessoRepository;
import br.com.portalz.condominio.entity.Acesso;

@RestController
@RequestMapping("/acesso")
public class AcessoREST {
    @Autowired
    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 -> AcessoRESTJava

```
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

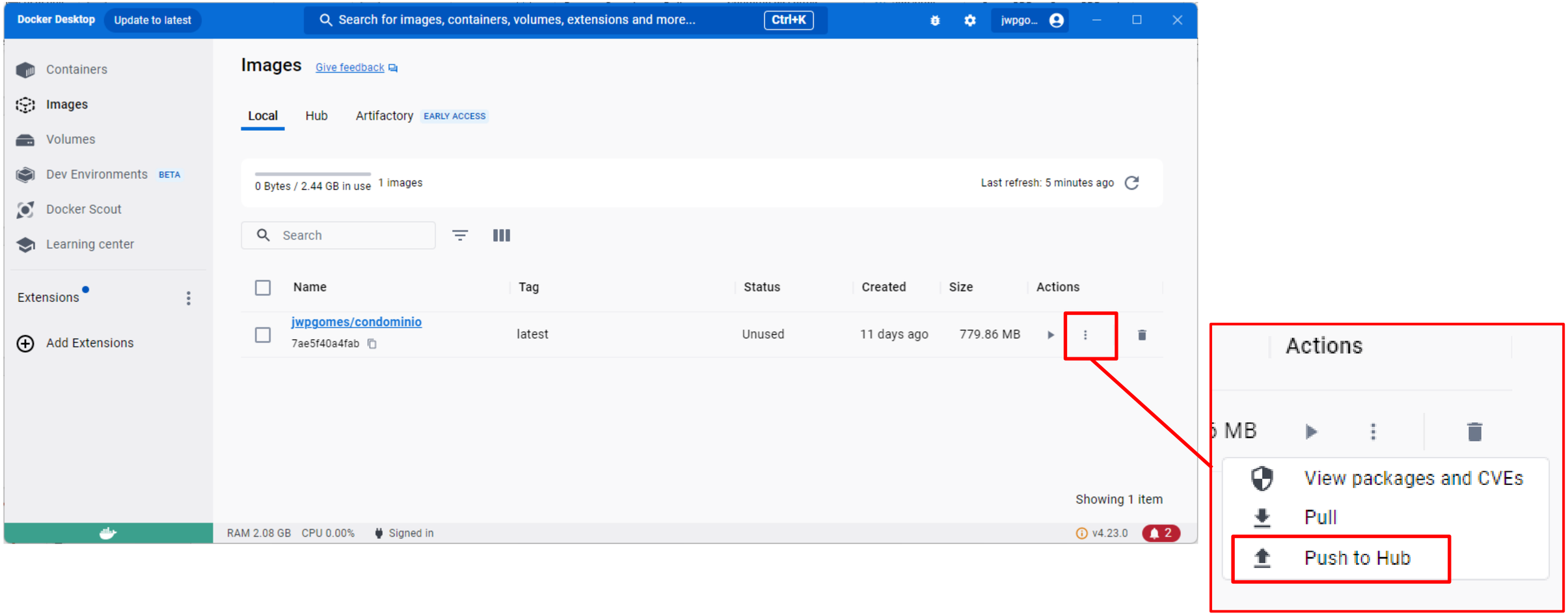
Criação da imagem para o Docker

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



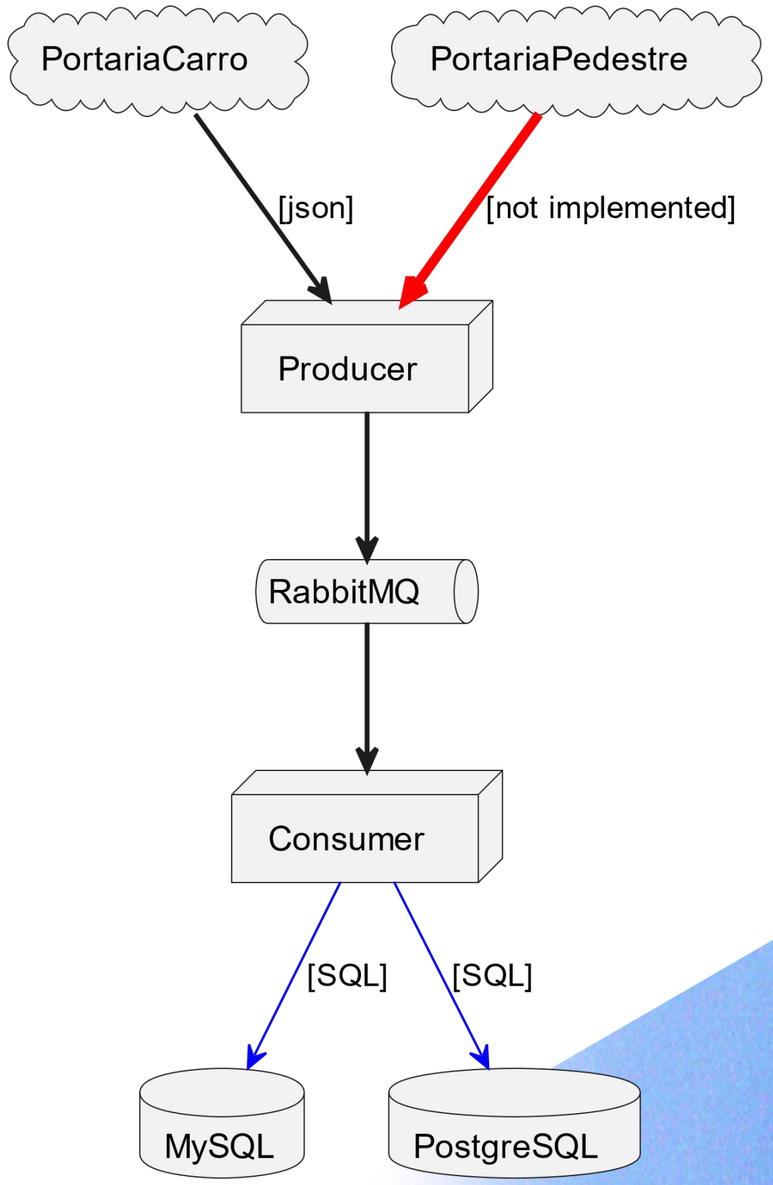
API - Registro da entrada de Carros

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

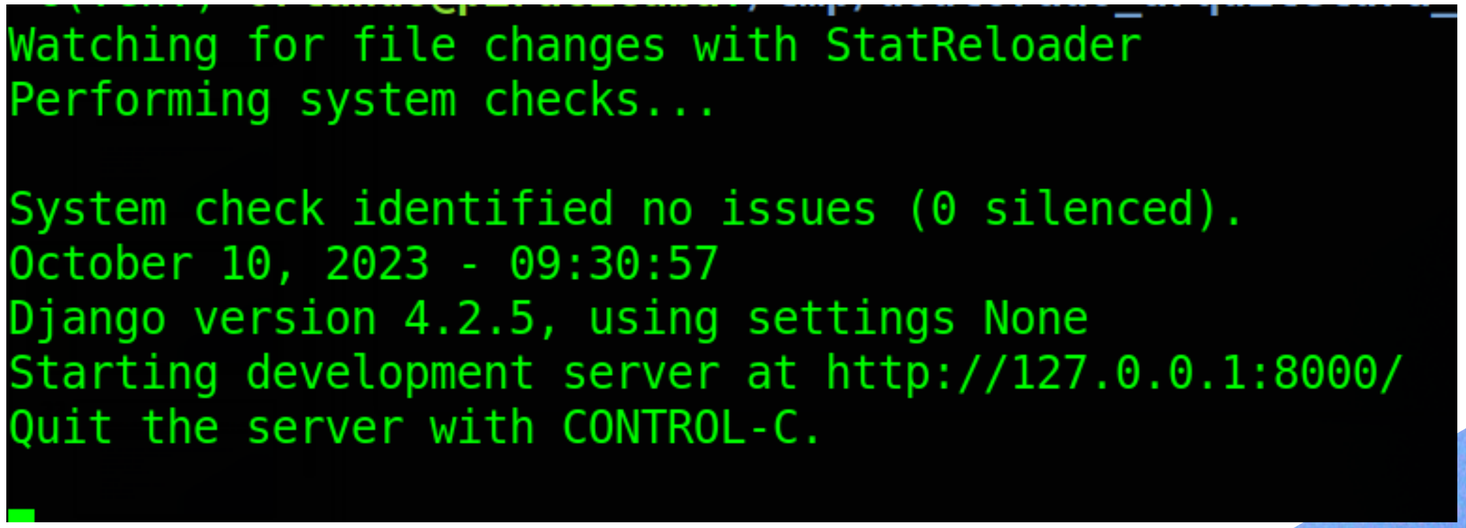
CONSUMO DA API


PortariaCarro
PortariaPedestre
[json]
[not implemented]
Producer
RabbitMQ
Consumer
[SQL]
[SQL]
MySQL
PostgreSQL

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**

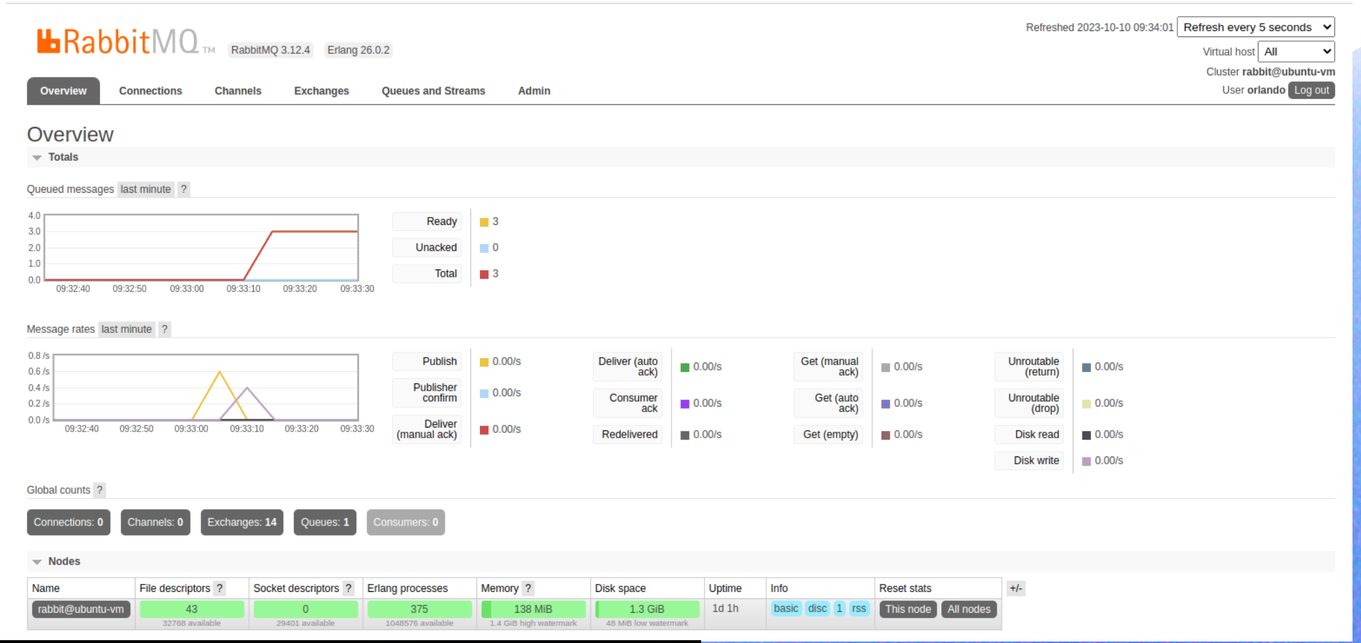


Configurar o **cliente** em um ambiente com acesso a um servidor RabbitMQ, um servidor SGBD MySQL e um servidor SGBD PostgreSQL instalado e configurado:

```
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:

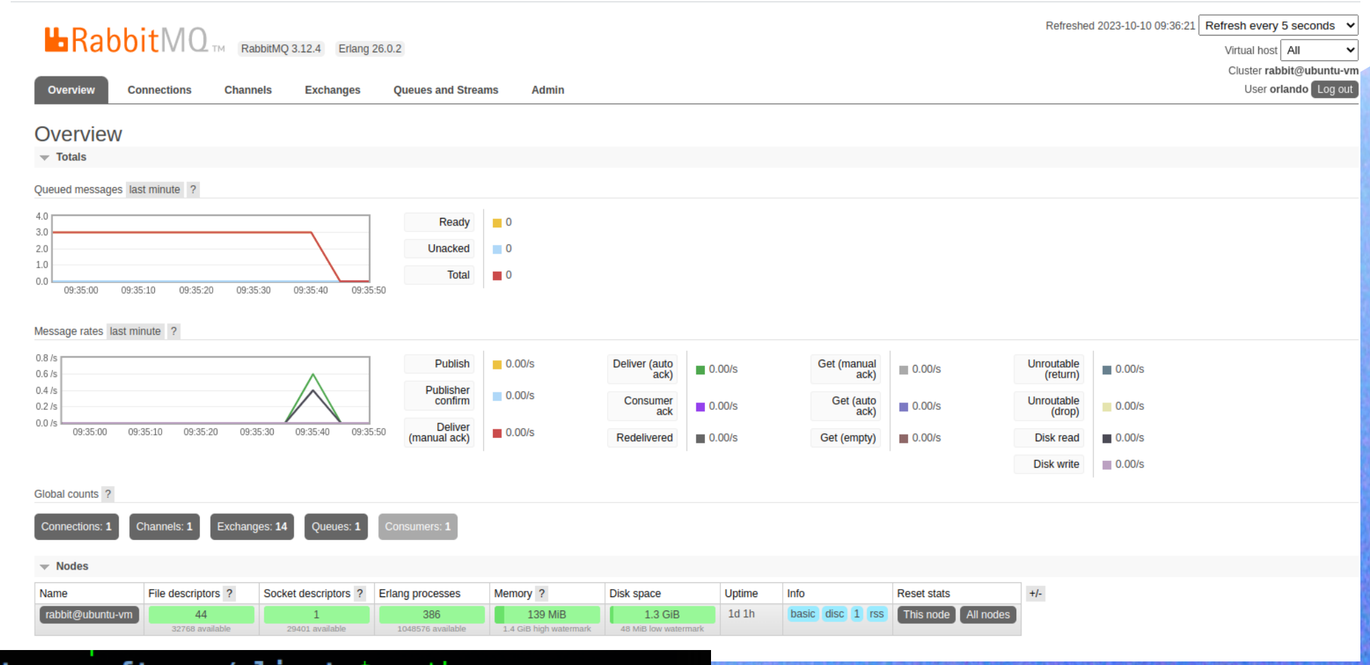
python producer.py




```
(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$
```

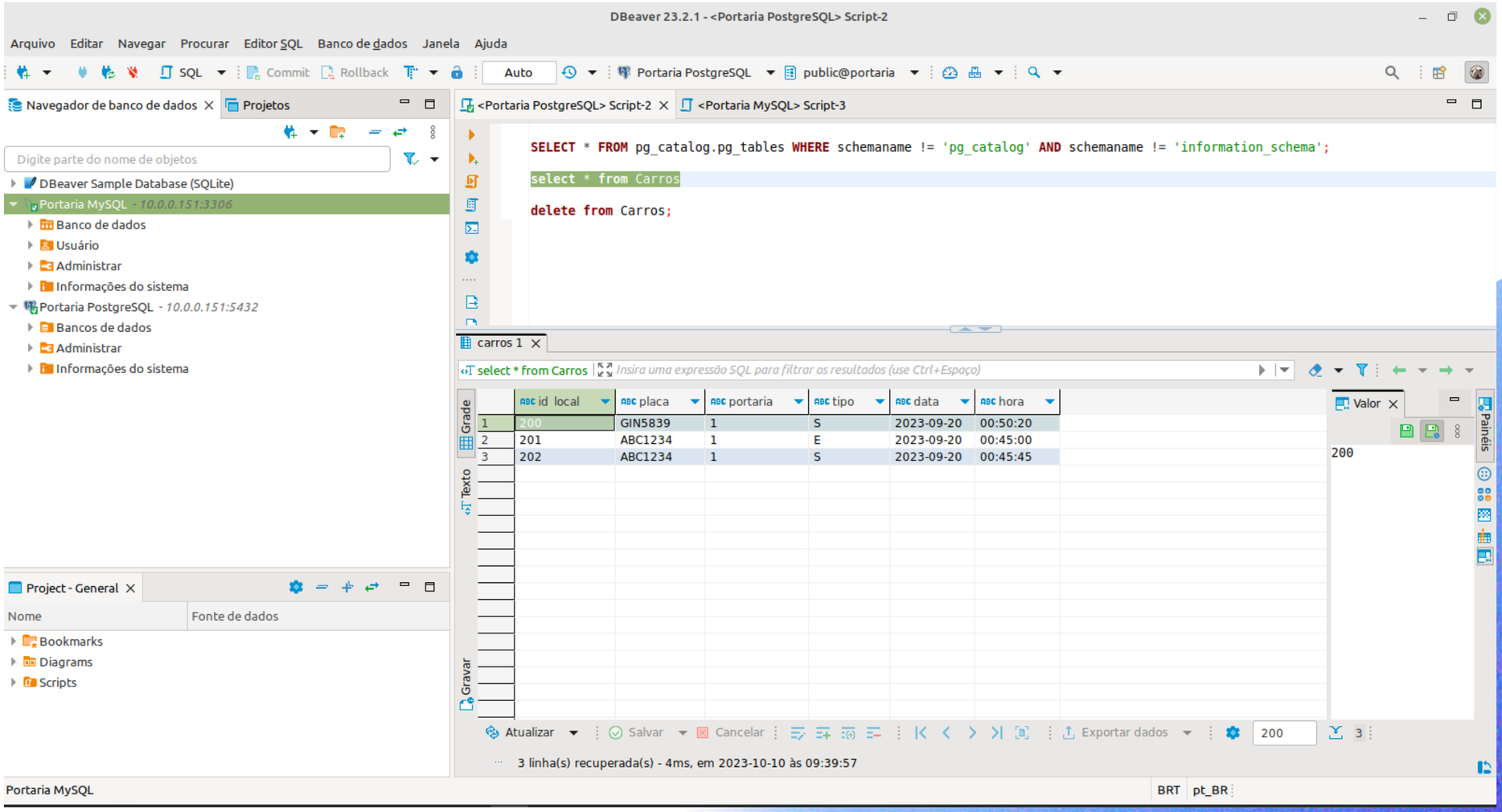
Configurar o **cliente** em um ambiente com acesso a um servidor RabbitMQ, um servidor SGBD MySQL e um servidor SGBD PostgreSQL instalado e configurado:

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**



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

nano .env

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