



INSTITUTO INFNET
FACULDADE EM TECNOLOGIA, INOVAÇÃO E NEGÓCIOS DIGITAIS
MICROSSERVIÇOS E DEVOPS COM SPRING BOOT E SPRING CLOUD
PROFESSOR RICARDO FROHLICH DA SILVA

PEDRO HENRIQUE MACEDO NORA

ASSESSMENT – AT

CASCADEL (PR), 26 DE SETEMBRO DE 2024

VÍDEO DE APRESENTAÇÃO

Link: <https://youtu.be/rqavTHSEv34>

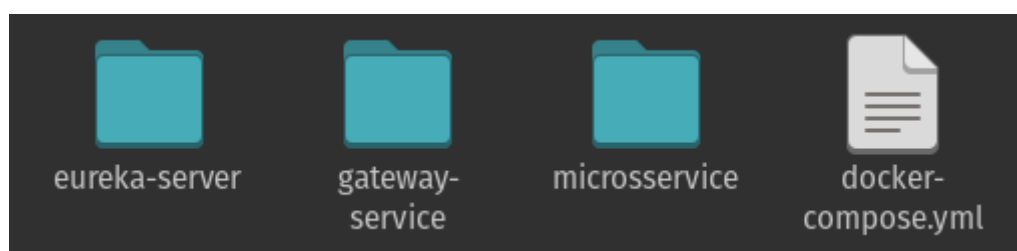
REPOSITÓRIO:

Link: <https://github.com/pedronora/AT-Microservices>

QUESTÃO 1

Implemente um microserviço simples usando Spring Boot Cloud que responda a uma requisição HTTP GET em um endpoint /status e retorne uma mensagem indicando que o serviço está ativo. Certifique-se de usar um Discovery Service como Eureka e testar o balanceamento de carga.

R.: Foram criados três projetos: eureka-server, gateway e o microserviço propriamente dito:



Para facilitar a execução, foi elaborado um docker-compose, que sobe todas as aplicações, sendo que o microservice é replicado por três vezes:

```
version: '3.9'
services:
  eureka-server:
    image: eureka-server
    build:
      context: ./eureka-server
      dockerfile: Dockerfile
    ports:
      - "8761:8761"
    networks:
      - eureka-net

  microservice:
    image: microservice
    build:
      context: ./microservice
      dockerfile: Dockerfile
    environment:
      EUREKA_CLIENT_SERVICE_URL_DEFAULTZONE: http://eureka-server:8761/eureka/
    depends_on:
      - eureka-server
    networks:
      - eureka-net
    deploy:
```

```

    replicas: 3

gateway:
  image: gateway
  build:
    context: ./gateway-service
    dockerfile: Dockerfile
  environment:
    EUREKA_CLIENT_SERVICE_URL_DEFAULTZONE: http://eureka-server:8761/eureka/
  depends_on:
    - eureka-server
    - microservice
  ports:
    - "9999:9999"
  networks:
    - eureka-net
  restart: unless-stopped

networks:
  eureka-net:
    driver: bridge

```

```

pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q1
=> => extracting sha256:44d3aa8d076675d49d85180b0ced9daef210fe4fdff4bdbb 0.9s
=> => extracting sha256:6ce99fdf16e86bd02f6ad66a0e1334878528b5a4b5487850 5.2s
=> [internal] load build context 12.2s
=> => transferring context: 57.10MB 9.6s
=> [2/3] WORKDIR /app 5.1s
=> [3/3] COPY target/gateway-service-0.0.1-SNAPSHOT.jar /app/app.jar 2.8s
=> exporting to image 25.4s
=> => exporting layers 10.4s
=> => exporting manifest sha256:aa3565f3e09766abf33a312de079e103ba580bb3 2.3s
=> => exporting config sha256:eeb1062175d2774d67b39ad37c24fece4546eb163a 1.4s
=> => exporting attestation manifest sha256:11f940fe0e3a5195f71aad6d4404 3.0s
=> => exporting manifest list sha256:ca1faa5262009af164e96e651613f16a961 2.0s
=> => naming to docker.io/library/gateway:latest 0.2s
=> => unpacking to docker.io/library/gateway:latest 4.6s

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview
Creating q1_eureka-server_1 ... done
Creating q1_microservice_1 ... done
Creating q1_microservice_2 ... done
Creating q1_microservice_3 ... done
Creating q1_gateway_1 ... done
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q1$

```

Visualização do Eureka Server e os microserviços devidamente registrados:

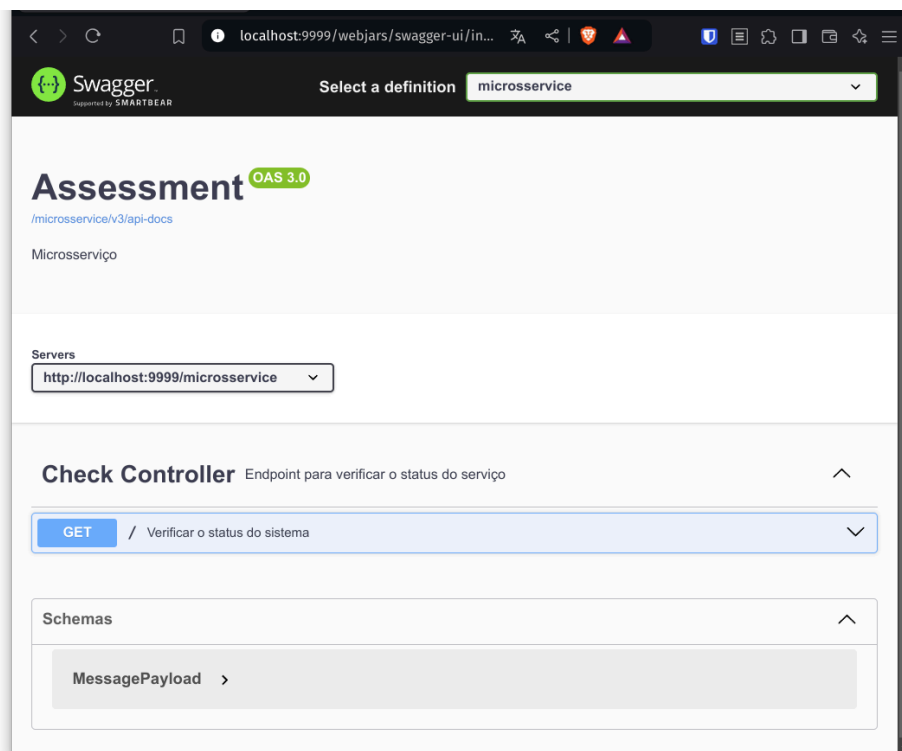
The screenshot displays the Eureka Server's System Status page. The browser address bar shows 'localhost:8761'. The page is divided into several sections:

- System Status:** A table showing environment details and system metrics.

Environment	test	Current time	2024-09-27T14:09:28+0000
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	8
		Renews (last min)	0
- DS Replicas:** A section for distributed storage replicas.
- Instances currently registered with Eureka:** A table listing registered services.

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - gateway-service:e49d307ca64d9e8f75534f308730b973
MICROSSERVICE	n/a (3)	(3)	UP (3) - microservice:164cae7856dfe66342239f05139321e0 , microservice:fded73ce2211a07811636f15f119218a , microservice:6b77070118fad4e48110efae0bbc4ae8
- General Info:** A section for general information.

Visualização da documentação (Swagger) do microserviço:



É possível visualizar o balanceamento de carga em funcionamento, observando o log da aplicação: *docker-compose logs -f*.

```

pedro@pop-os: ~/Documentos/Cursos/Infnet/2024.03/Microserviços/AT/Q1
microservice_3 2024-09-27T14:11:51.319Z INFO 1 --- [microservice] [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'd
microservice_3 dispatcherServlet'
microservice_3 2024-09-27T14:11:51.319Z INFO 1 --- [microservice] [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
microservice_3 2024-09-27T14:11:51.322Z INFO 1 --- [microservice] [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
microservice_3 2024-09-27T14:11:51.361Z INFO 1 --- [microservice] [o-auto-1-exec-1] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:11:55.004Z INFO 1 --- [microservice] [o-auto-1-exec-5] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:11:55.871Z INFO 1 --- [microservice] [o-auto-1-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'd
microservice_2 dispatcherServlet'
microservice_2 2024-09-27T14:11:55.872Z INFO 1 --- [microservice] [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
microservice_2 2024-09-27T14:11:55.874Z INFO 1 --- [microservice] [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
microservice_2 2024-09-27T14:11:55.909Z INFO 1 --- [microservice] [o-auto-1-exec-1] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:11:56.390Z INFO 1 --- [microservice] [o-auto-1-exec-2] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:11:56.962Z INFO 1 --- [microservice] [o-auto-1-exec-4] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:11:57.389Z INFO 1 --- [microservice] [o-auto-1-exec-2] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:11:57.640Z INFO 1 --- [microservice] [o-auto-1-exec-3] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:11:57.873Z INFO 1 --- [microservice] [o-auto-1-exec-2] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:11:58.072Z INFO 1 --- [microservice] [o-auto-1-exec-3] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:11:58.240Z INFO 1 --- [microservice] [o-auto-1-exec-4] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:11:58.429Z INFO 1 --- [microservice] [o-auto-1-exec-6] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:11:58.613Z INFO 1 --- [microservice] [o-auto-1-exec-4] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:11:58.808Z INFO 1 --- [microservice] [o-auto-1-exec-5] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:11:58.991Z INFO 1 --- [microservice] [o-auto-1-exec-7] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:11:59.171Z INFO 1 --- [microservice] [o-auto-1-exec-5] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:11:59.379Z INFO 1 --- [microservice] [o-auto-1-exec-6] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:11:59.600Z INFO 1 --- [microservice] [o-auto-1-exec-8] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:11:59.753Z INFO 1 --- [microservice] [o-auto-1-exec-7] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:12:00.004Z INFO 1 --- [microservice] [o-auto-1-exec-7] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:12:00.160Z INFO 1 --- [microservice] [o-auto-1-exec-9] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:12:00.402Z INFO 1 --- [microservice] [o-auto-1-exec-6] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:12:00.536Z INFO 1 --- [microservice] [o-auto-1-exec-8] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:12:00.746Z INFO 1 --- [microservice] [-auto-1-exec-10] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:12:00.939Z INFO 1 --- [microservice] [o-auto-1-exec-9] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:12:01.136Z INFO 1 --- [microservice] [o-auto-1-exec-9] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:12:01.303Z INFO 1 --- [microservice] [o-auto-1-exec-1] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:12:01.483Z INFO 1 --- [microservice] [-auto-1-exec-10] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:12:01.713Z INFO 1 --- [microservice] [-auto-1-exec-10] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:12:01.897Z INFO 1 --- [microservice] [o-auto-1-exec-3] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:12:02.062Z INFO 1 --- [microservice] [o-auto-1-exec-8] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:12:02.270Z INFO 1 --- [microservice] [o-auto-1-exec-1] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:12:02.728Z INFO 1 --- [microservice] [o-auto-1-exec-5] b.e.i.m.controller.CheckController : Status: Ativo
microservice_2 2024-09-27T14:12:02.963Z INFO 1 --- [microservice] [o-auto-1-exec-1] b.e.i.m.controller.CheckController : Status: Ativo
microservice_3 2024-09-27T14:12:03.152Z INFO 1 --- [microservice] [o-auto-1-exec-2] b.e.i.m.controller.CheckController : Status: Ativo
microservice_1 2024-09-27T14:12:03.327Z INFO 1 --- [microservice] [o-auto-1-exec-4] b.e.i.m.controller.CheckController : Status: Ativo

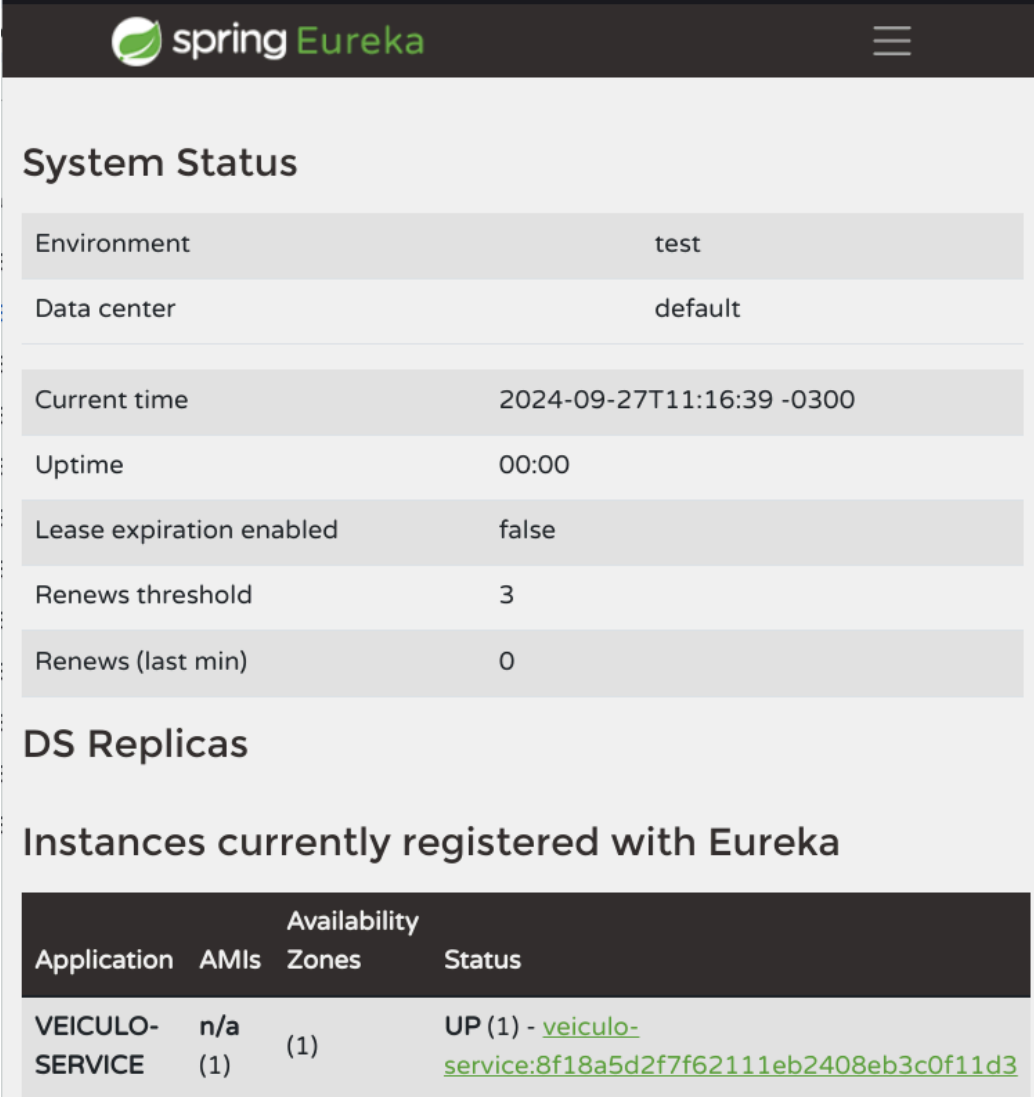
```

A cada solicitação um serviço diferente (cor diferente) é acionado, em razão do balanceamento de carga.

QUESTÃO 2

Desenvolva um microserviço utilizando Spring Boot e Spring Cloud, que será registrado em um servidor Eureka. O microserviço deve fornecer uma API RESTful com operações básicas de gestão de um catálogo de veículos, incluindo endpoints para listar, adicionar, atualizar e remover veículos. Implemente o servidor Eureka para que o microserviço se registre automaticamente, e use um servidor embutido (Tomcat). Teste os endpoints utilizando Spring Test.

R.: Servidor Eureka com o serviço Veículo registrado:



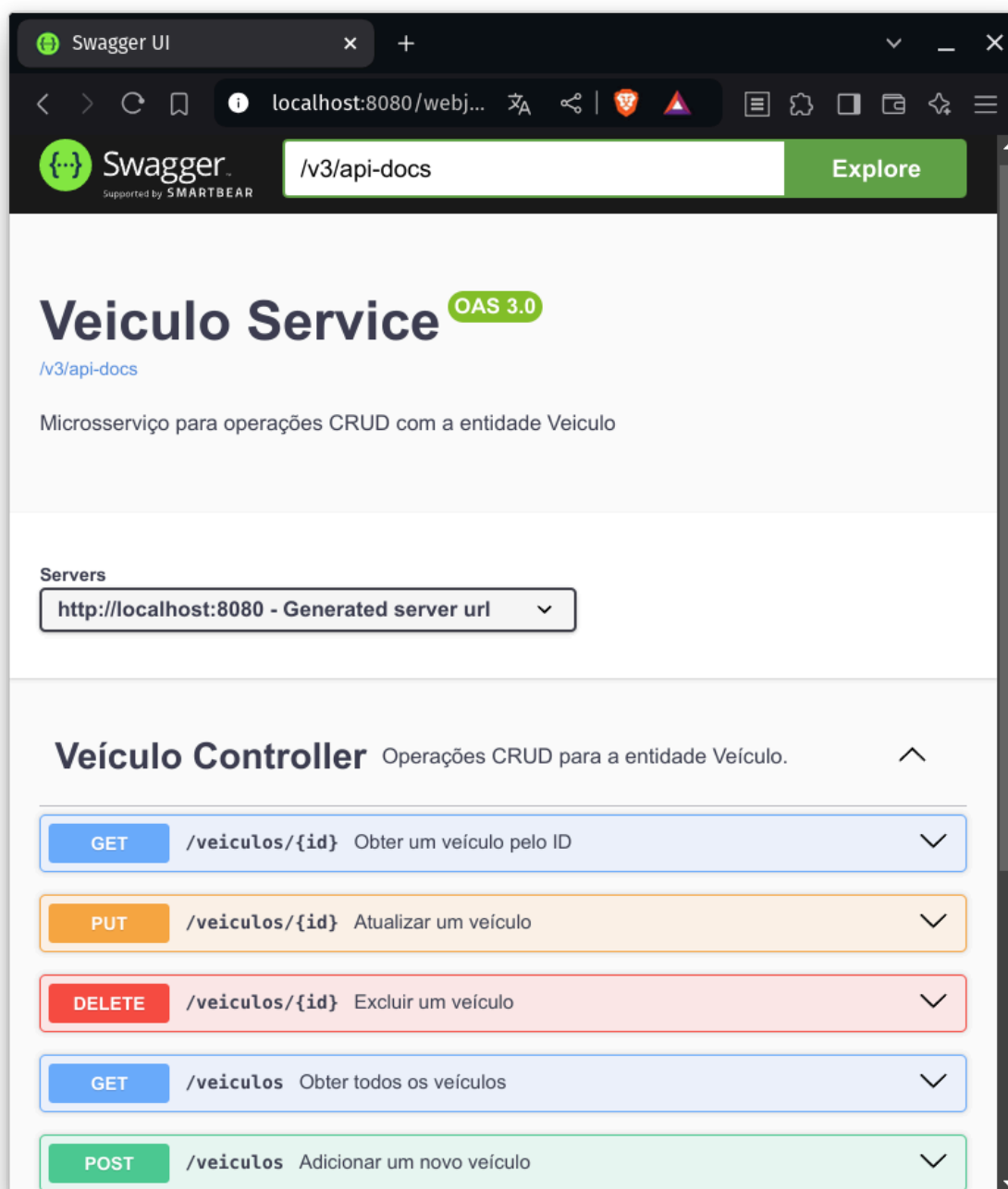
The screenshot shows the Spring Eureka web interface. At the top, there is a header with the 'spring Eureka' logo and a hamburger menu icon. Below the header, the 'System Status' section displays various system metrics in a table-like format. The 'DS Replicas' section is visible but empty. The 'Instances currently registered with Eureka' section shows a table with one registered instance: 'VEICULO-SERVICE'.

System Status	
Environment	test
Data center	default
Current time	2024-09-27T11:16:39 -0300
Uptime	00:00
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	0

DS Replicas	

Instances currently registered with Eureka			
Application	AMIs	Availability	
		Zones	Status
VEICULO-SERVICE	n/a (1)	(1)	UP (1) - veiculo-service:8f18a5d2f7f62111eb2408eb3c0f11d3

Documentação do serviço de Veículo, com as 5 operações CRUD:



E, por fim, o resultado dos testes desenvolvidos para testar todos os métodos contidos no Veículo Controller:

✓	VeiculoControllerTest (br.edu.infnet.\	746 ms
✓	testGetById_NotFound()	479 ms
✓	testGetById_Success()	60 ms
✓	testAddVeiculoBadRequest()	31 ms
✓	testDeleteVeiculoNotFound()	23 ms
✓	testUpdateVeiculo()	28 ms
✓	testGetAll()	82 ms
✓	testAddVeiculo()	14 ms
✓	testUpdateVeiculoNotFound()	18 ms
✓	testDeleteVeiculo()	11 ms

QUESTÃO 3

Descreva os passos necessários para migrar uma aplicação de microsserviços do Docker para o Kubernetes. Quais são os benefícios de realizar essa migração?

R.: A migração de uma aplicação de microsserviços do Docker para o Kubernetes envolve alguns passos importantes, mas o objetivo é garantir uma melhor gestão e escalabilidade da aplicação. Aqui estão os principais passos, explicados de forma clara:

- **Preparar os arquivos de configuração:** Cada microsserviço precisa de um arquivo chamado Dockerfile para ser executado em contêineres. Isso é utilizado no Docker. No Kubernetes, além dos Dockerfiles, é necessário criar arquivos de configuração no formato YAML. Esses arquivos especificam como os contêineres devem ser implantados e escalados no cluster Kubernetes.
- **Configurar o Kubernetes (cluster):** Antes de migrar, é necessário garantir que há um cluster Kubernetes configurado e funcionando. Esse cluster pode estar localmente (em ferramentas como Minikube) ou em um provedor de nuvem, como Google Kubernetes Engine (GKE) ou Amazon EKS.
- **Converter os contêineres para o Kubernetes:** Em vez de executar diretamente os contêineres com Docker, o Kubernetes precisa entender como gerenciá-los. Isso é feito criando objetos como Deployments e Services no Kubernetes. Um Deployment define quantas réplicas do contêiner são necessárias, enquanto o Service garante que os microsserviços possam se comunicar.
- **Configurar persistência e redes:** Muitos microsserviços precisam armazenar dados. No Kubernetes, isso pode ser feito por meio de volumes persistentes. Além disso, é necessário configurar as regras de rede para que os microsserviços possam se comunicar entre si e com o mundo externo.
- **Monitoramento e ajustes:** Após a migração, é essencial monitorar a aplicação no Kubernetes para identificar problemas de desempenho ou falhas. Ajustes podem ser necessários para melhorar a escalabilidade ou corrigir erros.

Benefícios da migração:

- **Escalabilidade automática:** O Kubernetes ajusta automaticamente a quantidade de instâncias de cada microsserviço com base na demanda.
- **Gestão de falhas:** Se um contêiner falhar, o Kubernetes automaticamente cria um novo para manter a aplicação funcionando.
- **Desempenho e eficiência:** Kubernetes otimiza o uso de recursos, garantindo que os microsserviços usem a quantidade certa de CPU e memória.
- **Gerenciamento centralizado:** Ao invés de gerenciar vários contêineres manualmente, o Kubernetes oferece um controle centralizado e organizado.
- **Portabilidade:** Uma vez configurada a aplicação no Kubernetes, ela pode ser facilmente transferida entre diferentes provedores de nuvem, garantindo mais flexibilidade operacional.

A migração para o Kubernetes oferece uma série de benefícios para aplicações de microsserviços. No entanto, é importante planejar cuidadosamente a migração e considerar os desafios e custos envolvidos.

QUESTÃO 4

Utilize Docker para criar e configurar um contêiner que execute um microserviço Spring Boot. O contêiner deve expor a porta correta para acessar o serviço.

R.: Para a tarefa determina, foi criado um Dockerfile, expondo a porta 8080, que é a mesma definida em *application.yml* do serviço:

```
Dockerfile
-----
FROM openjdk:17-jdk-slim

WORKDIR /app

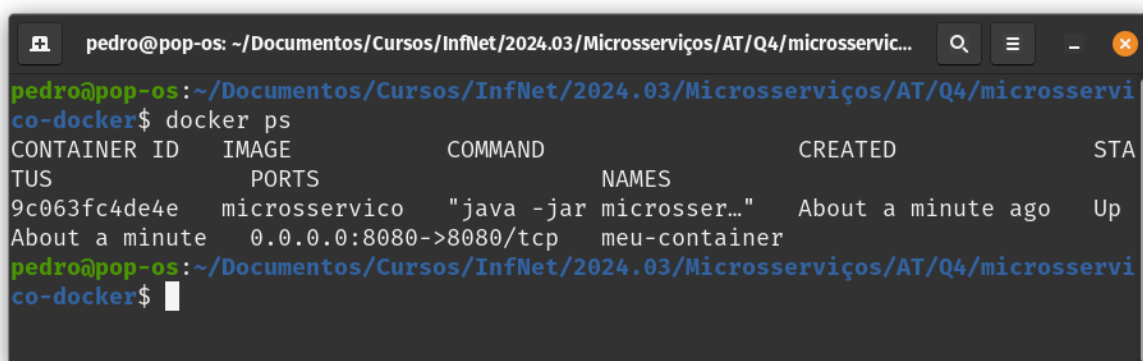
COPY target/microservico-docker-0.0.1-SNAPSHOT.jar /app/microservico.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "microservico.jar"]
```

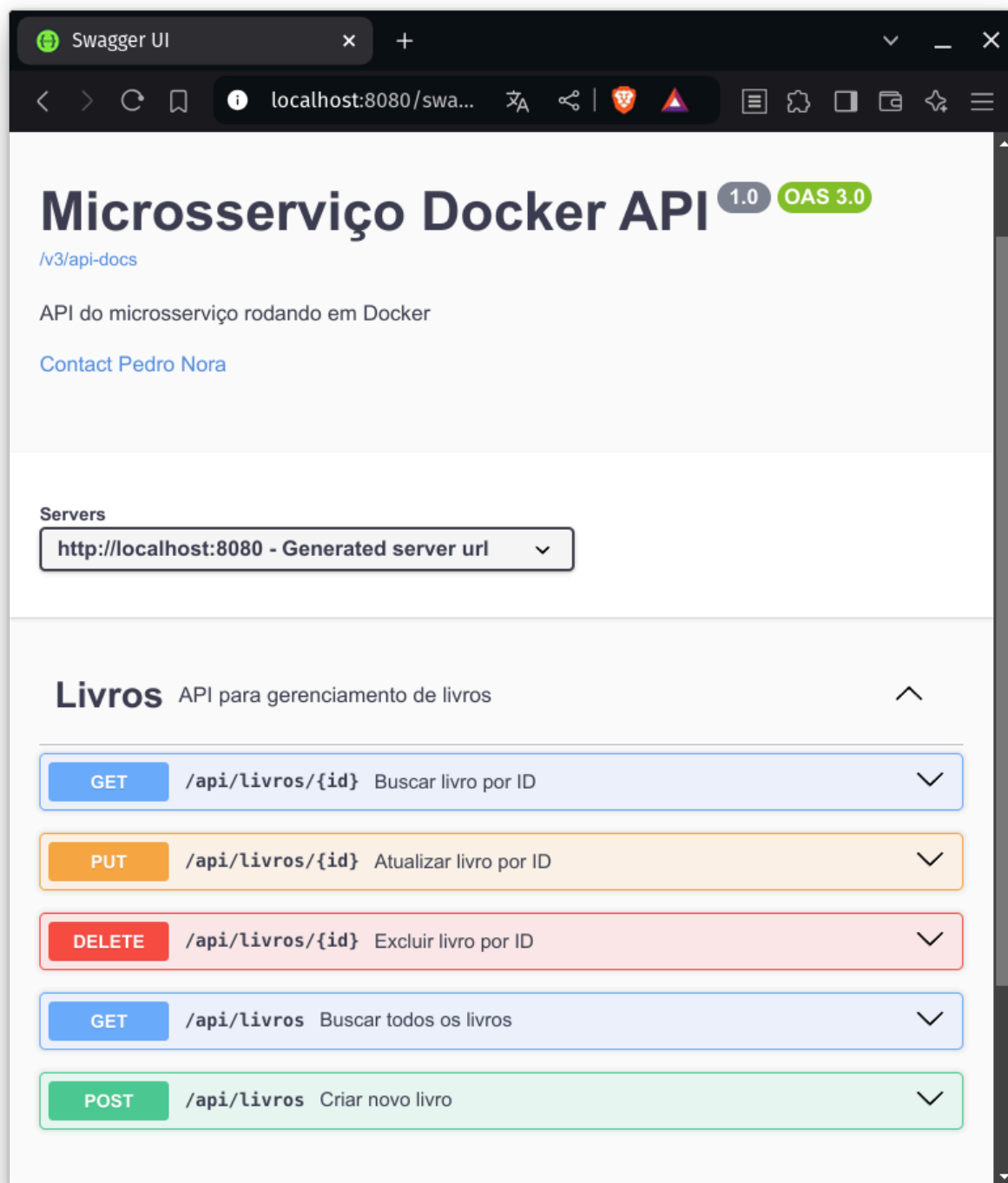
E o comando para criar a imagem Docker a partir do Dockerfile é: “*docker build -t microservico .*”. Após, necessário criar e executar o container a partir dessa imagem: “*docker run -d -p 8080:8080 --name meu-container microservico*”

Demonstração do container “meu-container” em execução:

A terminal window titled 'pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q4/microservic...' displays the command 'docker ps' and its output. The output shows a single container named 'meu-container' with ID '9c063fc4de4e', image 'microservico', and command '"java -jar microsser...". It is running on port 8080 and was created 'About a minute' ago.

```
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q4/microservi
co-docker$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
9c063fc4de4e   microservico   "java -jar microsser..." About a minute Up            0.0.0.0:8080->8080/tcp   meu-container
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q4/microservi
co-docker$
```

Documentação do serviço em execução no referido container:



QUESTÃO 5

Implante um microsserviço Spring Boot no Kubernetes e configure o balanceamento de carga para distribuir as requisições entre várias réplicas do serviço.

R.: Para tanto, foi necessário criar arquivos de configuração da aplicação, que estão localizados no diretório kubernetes:

- deployment.yaml:

```
deployment.yaml
-----
apiVersion: apps/v1
kind: Deployment
metadata:
  name: microservico-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: microservico
  template:
    metadata:
      labels:
        app: microservico
    spec:
      containers:
        - name: microservico
          image: microservico:v1
          ports:
            - containerPort: 8080
```

- service.yaml: destaca-se que foi especificado o tipo de serviço como *LoadBalancer*

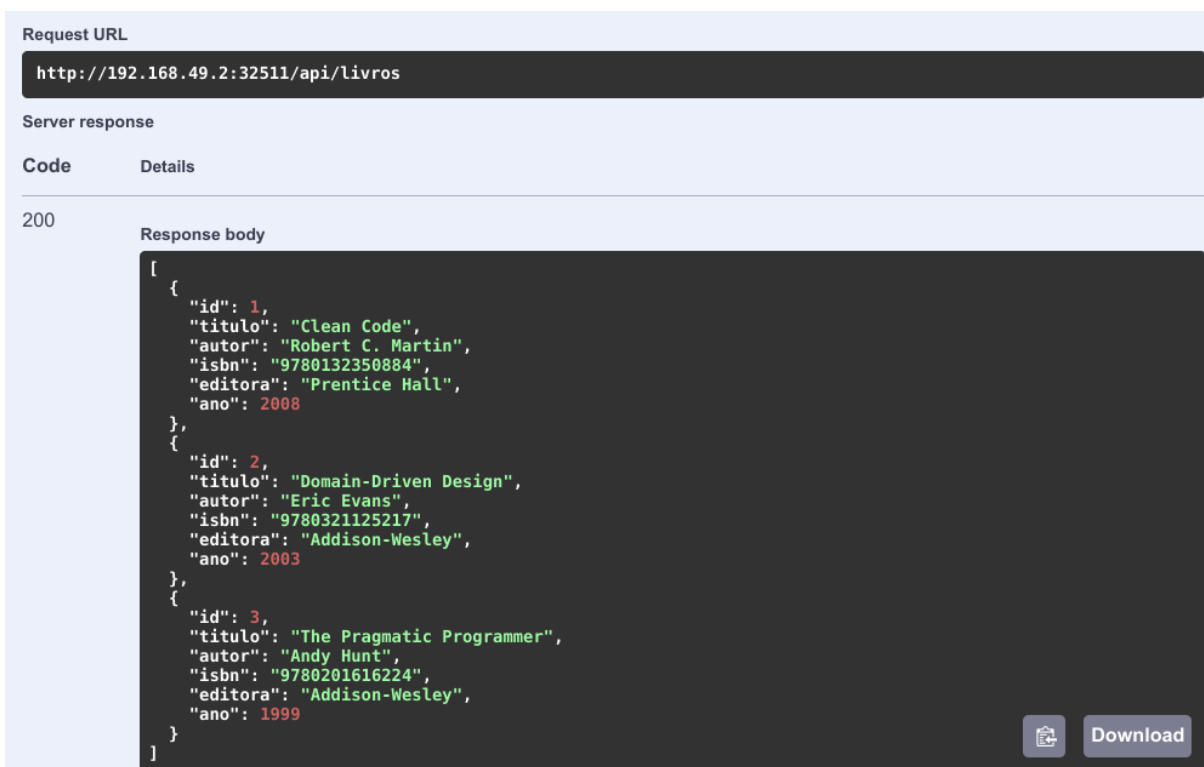
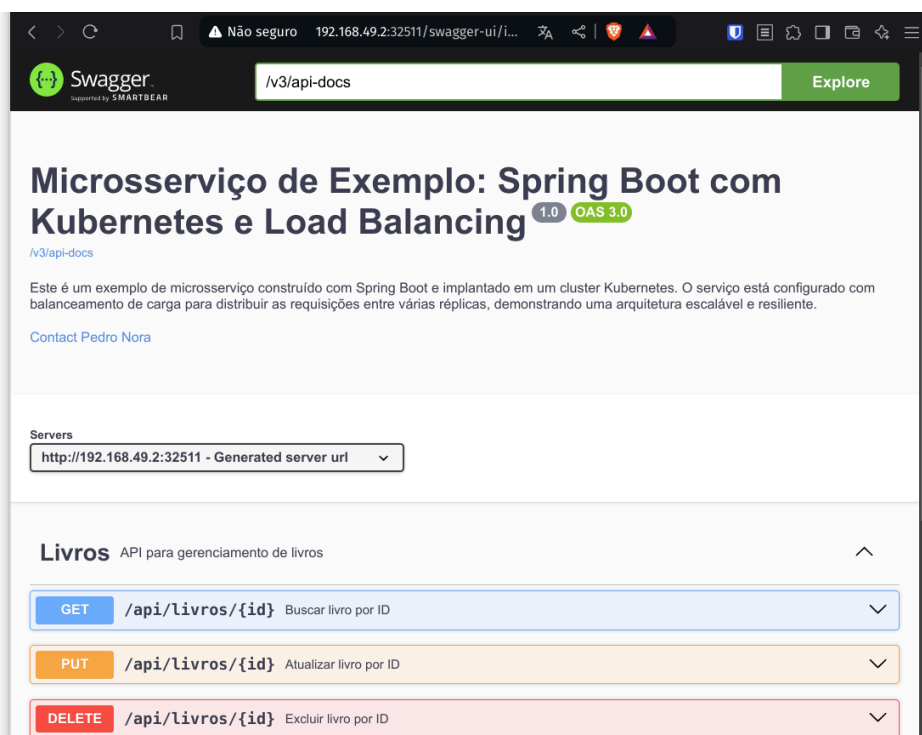
```
service.yaml
-----
apiVersion: v1
kind: Service
metadata:
  name: microservico-service
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: microservico
```

Para fins de teste, foi utilizado o minikube, cujos passos de execução estão discriminados no README.md do repositório da aplicação.

Três instâncias do serviço foram executadas corretamente:

```
pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q5/microservi...
co-docker$ kubectl apply -f kubernetes/service.yaml
service/microservico-service created
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q5/microservi
co-docker$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
microservico-deployment-7fbc58cdd5-445gg  1/1     Running   0          12s
microservico-deployment-7fbc58cdd5-4vxbz  1/1     Running   0          12s
microservico-deployment-7fbc58cdd5-mspdq  1/1     Running   0          12s
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q5/microservi
co-docker$ minikube service microservico-service
|-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | microservico-service | 80 | http://192.168.49.2:32511 |
|-----|-----|-----|-----|
🌐 Opening service default/microservico-service in default browser...
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q5/microservi
co-docker$ Abrindo em uma sessão de navegador existente.
```

E a aplicação está funcionando corretamente:



QUESTÃO 6

Implemente um servidor reativo simples usando Spring WebFlux que se conecte a um banco de dados H2 reativo via Spring Data R2DBC. O servidor deve responder a requisições GET e POST em endpoints para manipulação de dados.

R.: Para cumprir com a proposta apresentada, foram utilizadas estas dependências para a construção do projeto, conforme se verifica no respectivo arquivo pom.xml:

```
pom.xml
-----
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-r2dbc</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>io.r2dbc</groupId>
    <artifactId>r2dbc-h2</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

O banco de dados está assim configurado no *application.yaml*:

```
application.yaml
-----
spring:
```

```
application:
  name: R2DBC-server

r2dbc:
  url: r2dbc:h2:mem:///testdb
  username: sa
  password:

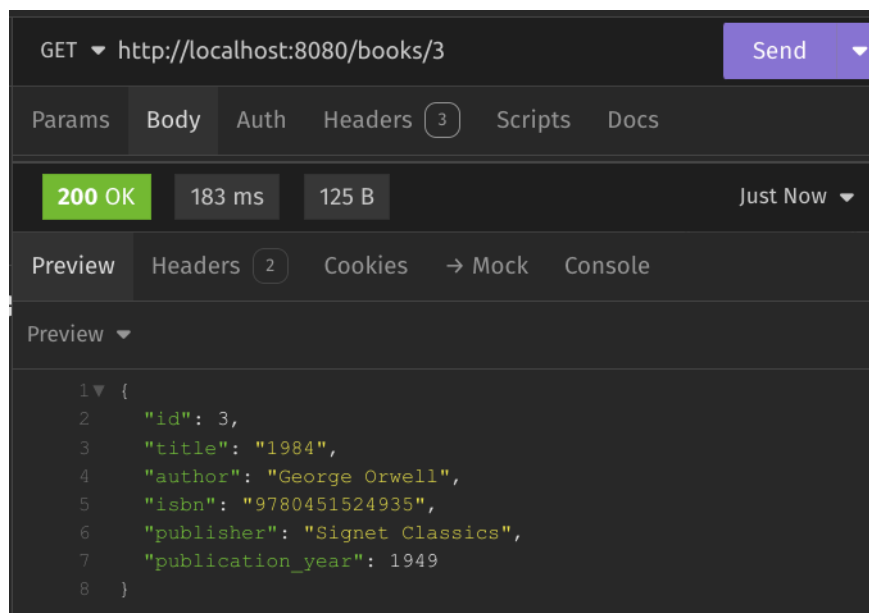
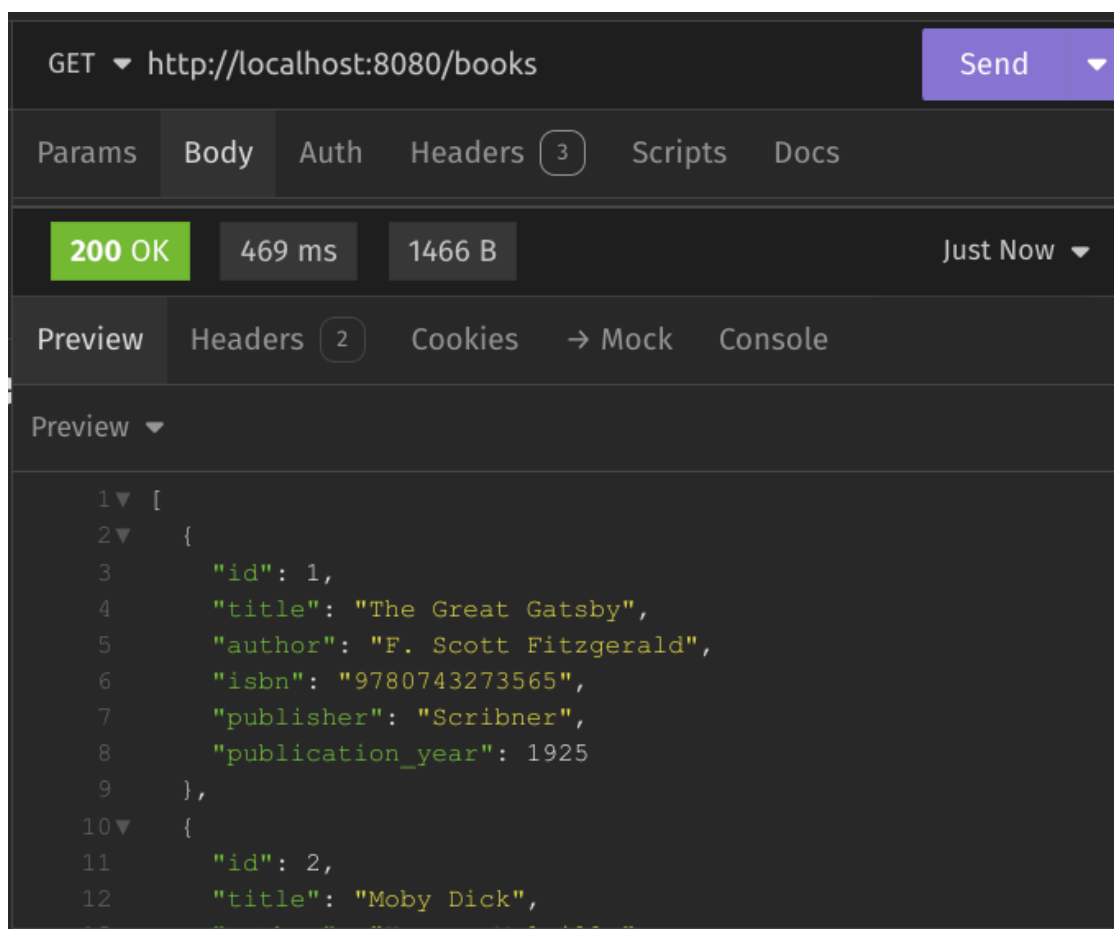
h2:
  console:
    enabled: true

sql:
  init:
    mode: always

logging:
  level:
    org.springframework.r2dbc: DEBUG
```

Para fins de demonstração, foi criada a entidade Book, em que é possível consultá-la, alterá-la ou mesmo criar novos.

Exemplo de requisição GET:



Exemplo de requisição POST:

POST ▼ http://localhost:8080/books Send ▼

Params Body ● Auth Headers 4 Scripts Docs

JSON ▼

```
1 {
2   "title": "Dom Casmurro",
3   "author": "Machado de Assis",
4   "isbn": "9788582850350",
5   "publisher": "Penguin-Companhia",
6   "publication_year": 2016
7 }
```

Beautify JSON

201 Created 262 ms 139 B Just Now ▼

Preview Headers 2 Cookies → Mock Console

Preview ▼

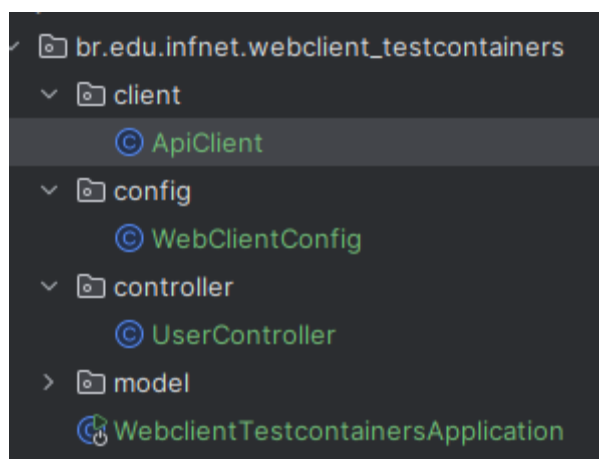
```
1 {
2   "id": 11,
3   "title": "Dom Casmurro",
4   "author": "Machado de Assis",
5   "isbn": "9788582850350",
6   "publisher": "Penguin-Companhia",
7   "publication_year": 2016
8 }
```

QUESTÃO 7

Desenvolva um cliente reativo com WebClient que consuma uma API externa e retorne os dados processados. Teste o cliente utilizando Testcontainers para garantir o comportamento adequado em um ambiente controlado.

R.: Para a execução da tarefa, foi utilizada esta API: <https://jsonplaceholder.typicode.com/>, a qual se denomina como “Free fake and reliable API for testing and prototyping”.

A estrutura do projeto ficou assim desenhada:



Para facilitar a troca da url de requisição entre o ambiente de produção e o de testes, foi criado um método (*setBaseUrl*) em *ApiClient* que tem justamente essa atribuição:

```
ApiClient.java
-----
@Service
public class ApiClient {

    private WebClient webClient;

    public ApiClient(WebClient.Builder webClientBuilder) {
        this.webClient = webClientBuilder.build();
    }

    public void setBaseUrl(String baseUrl) {
        this.webClient = this.webClient.mutate().baseUrl(baseUrl).build();
    }
}
```

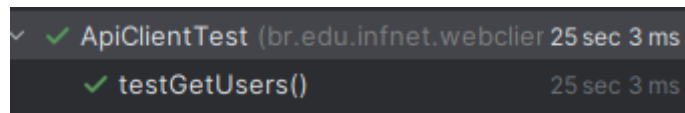
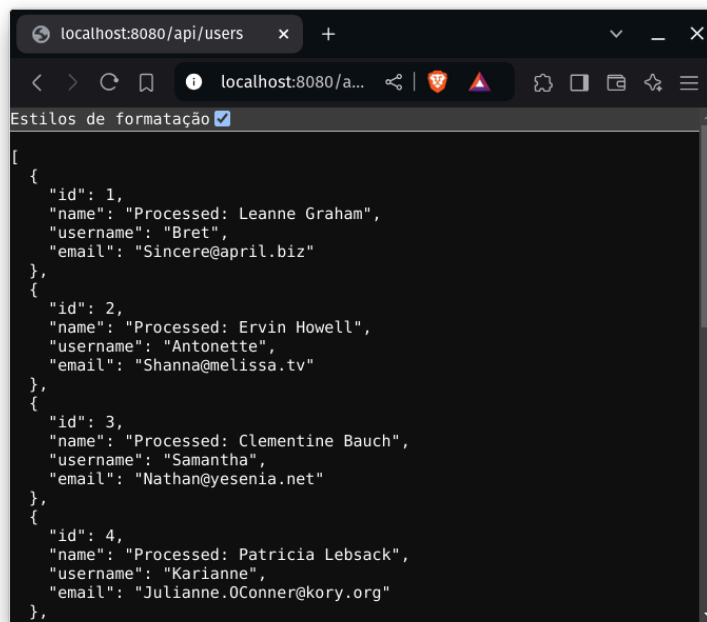
```
}

public Flux<User> getUsers() {
    return webClient
        .get()
        .uri("/users")
        .retrieve()
        .bodyToFlux(User.class)
        .map(
            user ->
                new User(
                    user.getId(),
                    "Processed: " + user.getName(),
                    user.getUsername(),
                    user.getEmail());
        )
}
```

Dessa forma, quando em produção, é determinado a url da API para a requisição, já nos testes, é utilizado o mockServer para tanto:

```
apiClient.setBaseUrl("http://" + mockServer.getHost() + ":" +
    mockServer.getMappedPort(8080));
```

A aplicação, portanto, funciona consumindo diretamente da API, como também testa de forma eficaz o endpoint:



Segue a classe de teste:

```
ApiClientTest.java
-----
@SpringBootTest
public class ApiClientTest {

    private static final String MOCK_SERVER_IMAGE = "wiremock/wiremock:2.31.0";
    private static GenericContainer<?> mockServer;

    @Autowired private ApiClient apiClient;

    @BeforeEach
    void setUp() {
        if (mockServer == null) {
            mockServer =
                new GenericContainer<>(MOCK_SERVER_IMAGE)
                    .withExposedPorts(8080)
                    .withCommand("--port", "8080", "--global-response-templating");
            mockServer.start();

            configureFor(mockServer.getHost(), mockServer.getMappedPort(8080));
            stubFor(
                get(urlEqualTo("/users"))
                    .willReturn(
                        aResponse()
```



```

        .withHeader("Content-Type", "application/json")
        .withBody(
            "[{"id": 1, "name": "John Doe", "username":
\"johndoe\", \"email\": \"john@example.com\"},\"
            + "{\"id\": 2, \"name\": \"Jane Smith\", \"username\":
\"janesmith\", \"email\": \"jane@example.com\"},\"
            + "{\"id\": 3, \"name\": \"Bob Brown\", \"username\":
\"bobb\", \"email\": \"bob@example.com\"}]"
        ).withStatus(200));
    }

    apiClient.setBaseUrl("http://" + mockServer.getHost() + ":" +
mockServer.getMappedPort(8080));
}

@Test
void testGetUsers() {
    Flux<User> usersFlux = apiClient.getUsers();

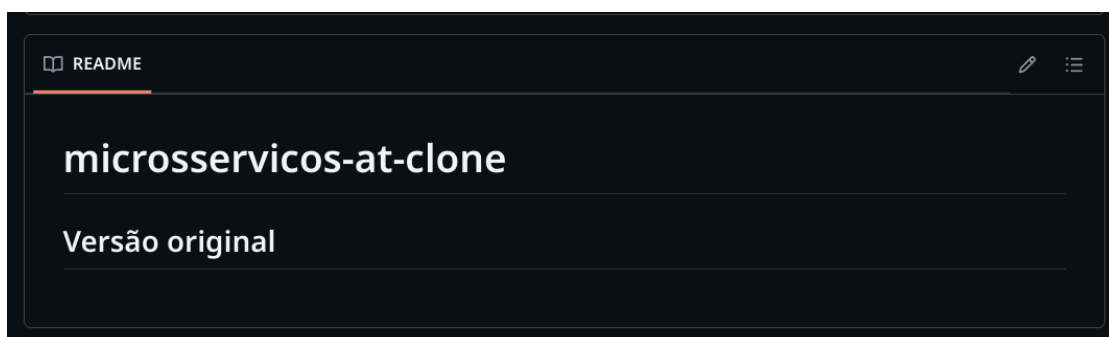
    StepVerifier.create(usersFlux)
        .expectNextMatches(
            user -> {
                return user.getName().equals("Processed: John Doe");
            }
        )
        .expectNextMatches(
            user -> {
                return user.getName().equals("Processed: Jane Smith");
            }
        )
        .expectNextMatches(
            user -> {
                return user.getName().equals("Processed: Bob Brown");
            }
        )
        .verifyComplete();
}
}

```

QUESTÃO 8

Desenvolva um cliente reativo com WebClient que consuma uma API externa e retorne os dados processados. Teste o cliente utilizando Testcontainers para garantir o comportamento adequado em um ambiente controlado.

R.: Para responder a questão, foi criado um repositório no GitHub (<https://github.com/pedronora/microservicos-at-clone>), apenas com um README.md, com este conteúdo:



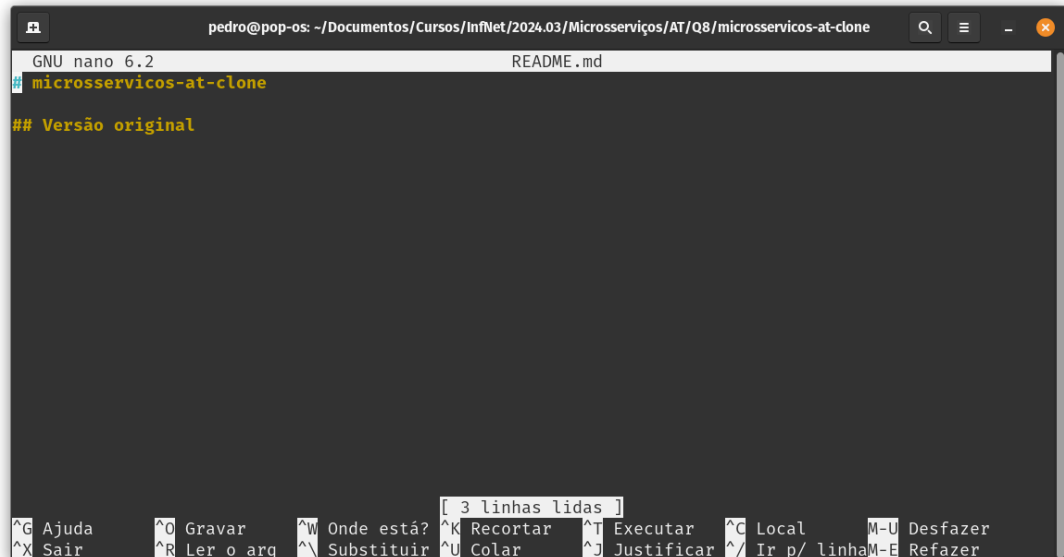
Inicialmente, foi realizado o clone do repositório para uma pasta local, criada como mostrado na imagem a seguir:

```
pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT$ cd Documentos/Cursos/InfNet/2024.03
/Microserviços/AT/
bash: cd: Documentos/Cursos/InfNet/2024.03/Microserviços/AT/: Arquivo ou diretório inexistente
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT$ mkdir Q8
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8$ git clone https://github.com/ped
ronora/microservicos-at-clone.git
Cloning into 'microservicos-at-clone'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8$
```

Em seguida, verificado o conteúdo da pasta e localizado o arquivo README.md que se deseja trabalhar/alterar:

```
pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microservicos-at-clone
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8$ ls
microservicos-at-clone
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8$ cd microservicos-at-clone/
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microservicos-at-clone$ ls
README.md
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microservicos-at-clone$
```

Visualização do arquivo README.md:



The screenshot shows a terminal window with the title bar "pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microservicos-at-clone". The terminal is running GNU nano 6.2, editing the file README.md. The content of the file is:

```
# microservicos-at-clone

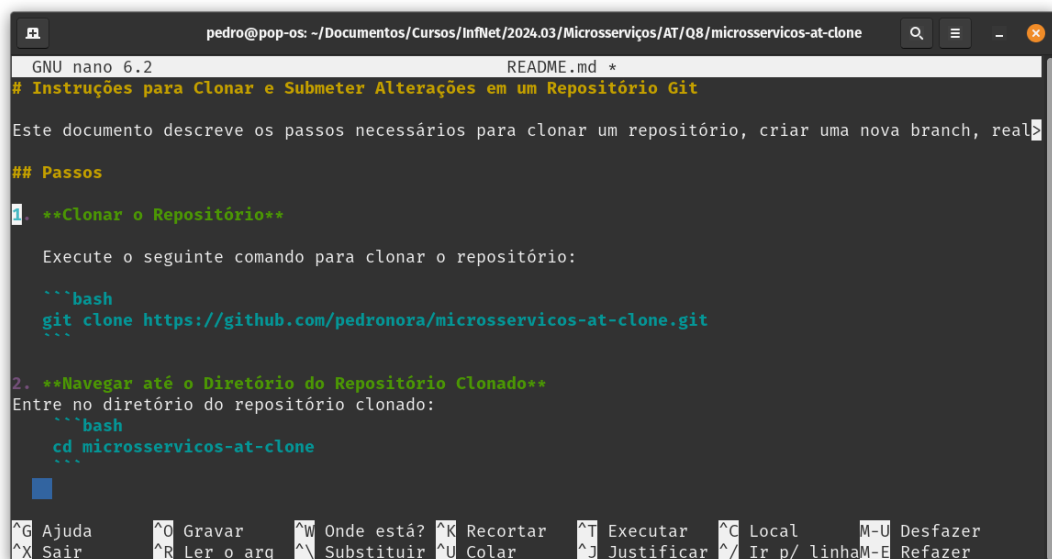
## Versão original
```

 The bottom status bar shows various keyboard shortcuts like ^G Ajuda, ^O Gravar, ^W Onde está?, etc.

Criar nova branch e acessá-la:

```
git checkout -b novas-funcionalidades
```

Realizar as alterações:



The screenshot shows the same terminal window, but the README.md file now contains more text. The content is:

```
# Instruções para Clonar e Submeter Alterações em um Repositório Git

Este documento descreve os passos necessários para clonar um repositório, criar uma nova branch, real>

## Passos

1. **Clonar o Repositório**

Execute o seguinte comando para clonar o repositório:



```
bash
git clone https://github.com/pedronora/microservicos-at-clone.git
```



2. **Navegar até o Diretório do Repositório Clonado**

Entre no diretório do repositório clonado:



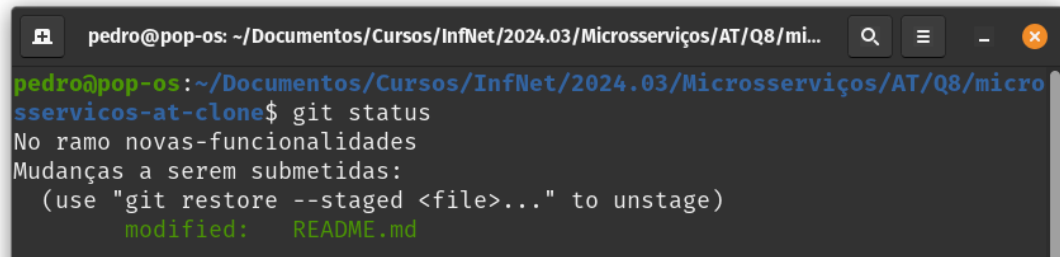
```
bash
cd microservicos-at-clone
```


```

 The bottom status bar is the same as in the previous screenshot.

Na sequência deve adicionar as alterações no *staging*:

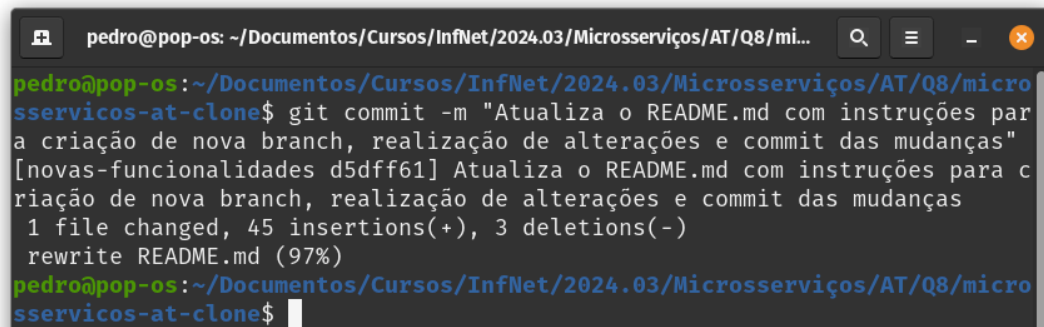
```
git add .
```

A terminal window with a dark background. The title bar shows the user 'pedro@pop-os' and the path '~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/mi...'. The prompt is 'pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microserviços-at-clone\$'. The command 'git status' has been executed, resulting in the following output: 'No ramo novas-funcionalidades', 'Mudanças a serem submetidas:', '(use "git restore --staged <file>..." to unstage)', and 'modified: README.md'.

```
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microserviços-at-clone$ git status
No ramo novas-funcionalidades
Mudanças a serem submetidas:
  (use "git restore --staged <file>..." to unstage)
       modified:   README.md
```

Realizar o commit com a respectiva mensagem:

```
git commit -m "Atualiza o README.md com instruções para criação de nova branch, realização de alterações e commit das mudanças"
```

A terminal window with a dark background. The title bar shows the user 'pedro@pop-os' and the path '~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/mi...'. The prompt is 'pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microserviços-at-clone\$'. The command 'git commit -m "Atualiza o README.md com instruções para criação de nova branch, realização de alterações e commit das mudanças"' has been executed, resulting in the following output: '[novas-funcionalidades d5dff61] Atualiza o README.md com instruções para criação de nova branch, realização de alterações e commit das mudanças', '1 file changed, 45 insertions(+), 3 deletions(-)', 'rewrite README.md (97%)', and a new prompt 'pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microserviços-at-clone\$'.

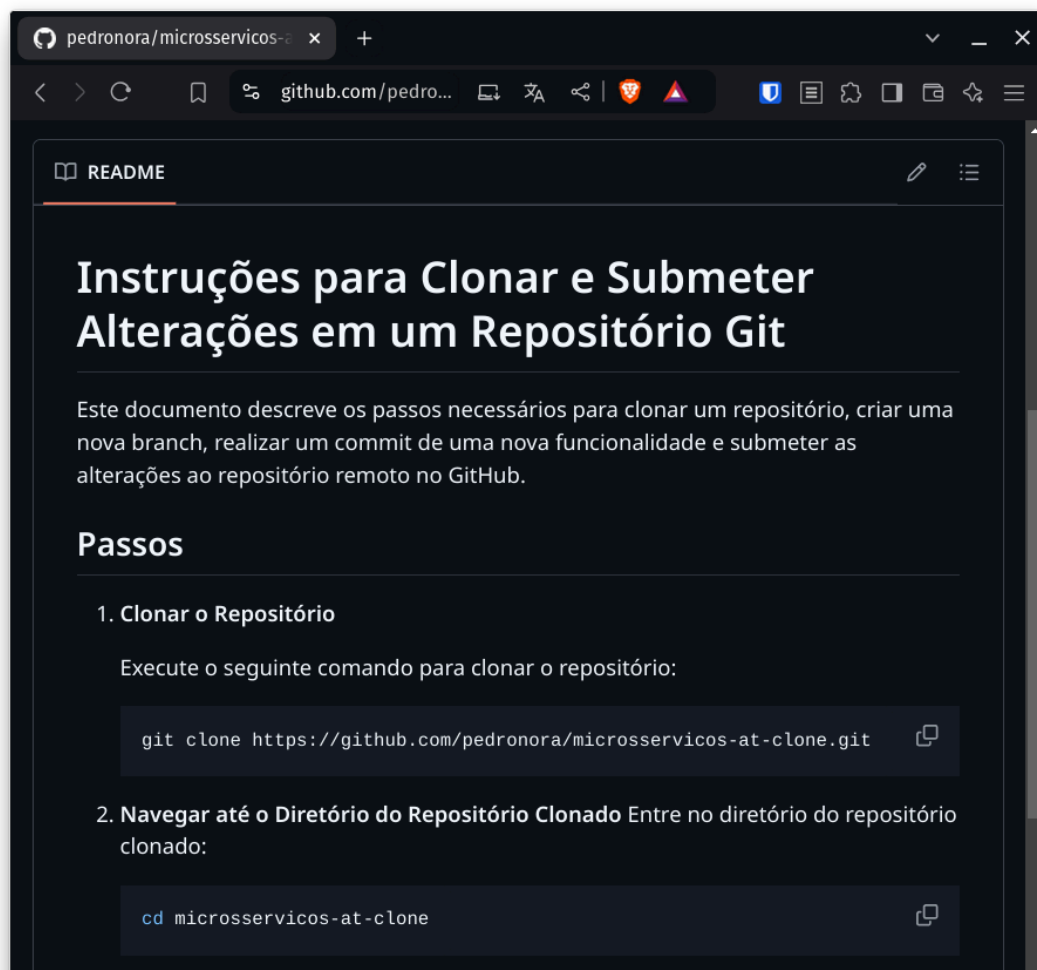
```
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microserviços-at-clone$ git commit -m "Atualiza o README.md com instruções para criação de nova branch, realização de alterações e commit das mudanças"
[novas-funcionalidades d5dff61] Atualiza o README.md com instruções para criação de nova branch, realização de alterações e commit das mudanças
1 file changed, 45 insertions(+), 3 deletions(-)
rewrite README.md (97%)
pedro@pop-os:~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/microserviços-at-clone$
```

E, por fim, submeter as alterações ao repositório remoto:

```
git push origin novas-funcionalidades
```

```
pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/micro
sservicos-at-clone$ git push origin novas-funcionalidades
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 925 bytes | 925.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'novas-funcionalidades' on GitHub by vis
iting:
remote:   https://github.com/pedronora/microservicos-at-clone/pull/new
/novas-funcionalidades
remote:
To https://github.com/pedronora/microservicos-at-clone.git
 * [new branch]      novas-funcionalidades -> novas-funcionalidades
pedro@pop-os: ~/Documentos/Cursos/InfNet/2024.03/Microserviços/AT/Q8/micro
sservicos-at-clone$
```

O resultado pode ser observado no endereço remoto do repositório:



The screenshot shows a web browser displaying the GitHub repository page for 'pedronora/microservicos-at-clone'. The page title is 'Instruções para Clonar e Submeter Alterações em um Repositório Git'. The README content describes the steps for cloning the repository, creating a new branch, committing changes, and pushing them to the remote repository on GitHub. The first step, '1. Clonar o Repositório', includes the command 'git clone https://github.com/pedronora/microservicos-at-clone.git'. The second step, '2. Navegar até o Diretório do Repositório Clonado', includes the command 'cd microservicos-at-clone'.

Instruções para Clonar e Submeter Alterações em um Repositório Git

Este documento descreve os passos necessários para clonar um repositório, criar uma nova branch, realizar um commit de uma nova funcionalidade e submeter as alterações ao repositório remoto no GitHub.

Passos

- 1. Clonar o Repositório**

Execute o seguinte comando para clonar o repositório:

```
git clone https://github.com/pedronora/microservicos-at-clone.git
```
- 2. Navegar até o Diretório do Repositório Clonado**

Entre no diretório do repositório clonado:

```
cd microservicos-at-clone
```

QUESTÃO 9

Explique o que são GitHub Actions e como elas facilitam a automação de tarefas em projetos de software. Quais são os principais benefícios de utilizar GitHub Actions no processo de CI/CD (Integração Contínua e Entrega Contínua)?

R.: O GitHub Actions é uma ferramenta de automação do GitHub que possibilita a elaboração automatizada de fluxos de trabalho para, entre outras coisas, testes, construção e implementação, com os arquivos de configuração sendo descritos no formato YAML e incluídos no repositório.

Simplifica o trabalho de CI/CD ao executar integrações e entregas contínuas, executar testes e implementar alterações automaticamente sempre que ajustes substanciais são feitos ao código.

Os principais benefícios são a automação total, a integração nativa com o GitHub, a execução paralela de tarefas e a escalabilidade – tudo diretamente na plataforma, o que resulta em um desenvolvimento mais veloz e eficaz.

QUESTÃO 10

Descreva como os sistemas Spring Boot podem ser implantados em um ambiente Kubernetes usando GitHub Actions. Quais são as etapas principais envolvidas nesse processo?

R.: O processo para implantar sistemas Spring Boot em Kubernetes usando GitHub Actions, envolve algumas etapas principais.

Inicialmente, o código da aplicação Spring Boot deve estar versionado no GitHub, e um Dockerfile precisa estar configurado para gerar a imagem do contêiner da aplicação.

O GitHub Actions pode ser configurado para criar um workflow que, ao detectar mudanças no código, realiza a build da aplicação, gera a imagem Docker e a envia para um registro de contêineres como Docker Hub ou Google Container Registry.

Em seguida, o workflow também pode aplicar a configuração Kubernetes (manifests YAML) e realizar o deploy no cluster, utilizando ferramentas como kubectl.

O ciclo se repete sempre que há novas atualizações no código, permitindo integração e entrega contínuas.

Nas respostas seguintes, em que é configurada a aplicação SpringBoot para que seja implantada em um ambiente Kubernetes, os requisitos são demonstrados de forma mais prática e clara.

QUESTÃO 11

Implemente um workflow básico no GitHub Actions para compilar e testar um microserviço Spring Boot automaticamente sempre que houver um push no repositório. Inclua código e metadados apropriados no arquivo de configuração.

R.: Para compilar e testar o microserviço, foi deduzido este workflow:

```
name: CI Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up JDK 17
        uses: actions/setup-java@v2
        with:
          java-version: '17'
          distribution: 'adopt'

      - name: Build with Maven
        run: mvn clean package # Compila o projeto e cria o JAR

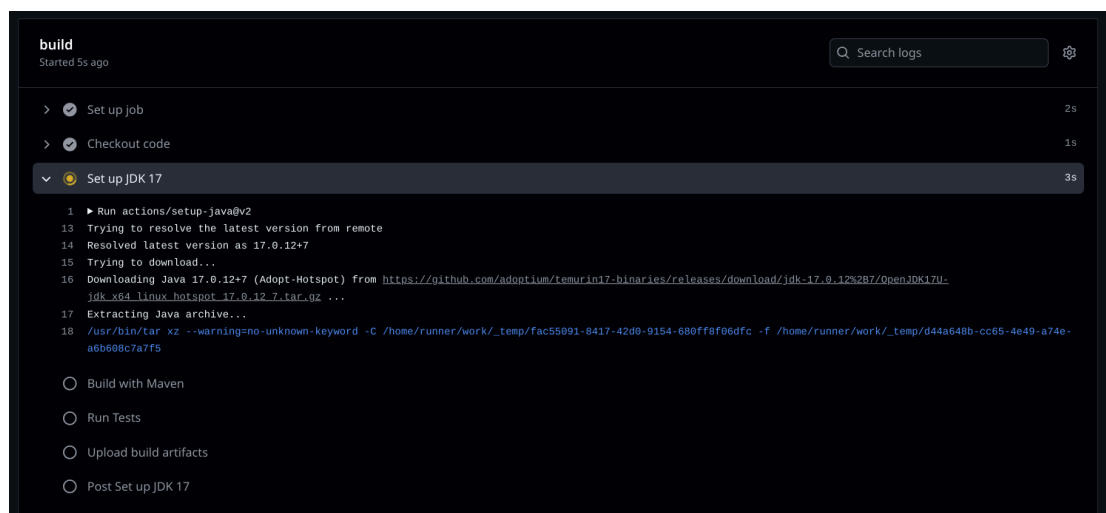
      - name: Run Tests
        run: mvn test # Executa os testes

      - name: Upload build artifacts
        uses: actions/upload-artifact@v3
        with:
          name: maven-build
          path: target/*.jar # Upload do JAR gerado para os artefatos
```


Esse workflow automatiza a compilação e os testes do seu microsserviço Spring Boot sempre que há alterações na branch main, garantindo que o código esteja sempre em um estado funcional. e destacam-se estes pontos:

- **Ativação:** O workflow é acionado em dois casos:
 - Quando há um *push* na branch main; ou
 - Quando há um *pull request* para a branch main.
- **Jobs:**
 - **build:** Esse jobo é executado em um ambiente Ubuntu (distribuição Linux).
- **Etapas:**
 1. **Checkout code:** Baixa o código do repositório;
 2. **Set up JDK 17:** Configura o JDK 17 usando a distribuição AdoptOpenJDK;
 3. **Build with Maven:** Compila o projeto e gera o arquivo JAR;
 4. **Run Tests:** Executa os testes do projeto; e
 5. **Upload build artifacts:** Faz o upload do JAR gerado como artefato do build.

Após o último commit e push, o Github Actions iniciou o processamento do workflow, terminando com sucesso:



build

succeeded 1 minute ago in 41s

- >  Set up job
- >  Checkout code
- >  Set up JDK 17
- >  Build with Maven
- >  Run Tests
- >  Upload build artifacts
- >  Post Set up JDK 17
- >  Post Checkout code
- >  Complete job

QUESTÃO 12

Configure um workflow de exemplo na interface web do GitHub Actions para automatizar o deploy de um microserviço Spring Boot em um cluster Kubernetes. O workflow deve exemplificar o build da aplicação e gerenciar o deploy no Kubernetes.

R.: Esse workflow automatiza o processo de build e deploy de um microserviço Spring Boot em um cluster Kubernetes, desde a construção do projeto até a publicação da imagem Docker e a aplicação do deployment.

```
name: Deploy
-----
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up JDK 17
        uses: actions/setup-java@v1
        with:
          java-version: '17'

      - name: Build with Maven
        run: mvn clean install

      - name: Log in to Docker Hub
        run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u
"${{ secrets.DOCKER_USERNAME }}" --password-stdin
```

```

- name: Build Docker image
  run: docker build . -t ${ secrets.DOCKER_USERNAME }/${ secrets.DOCKER_REPOSITORY }:latest

- name: Push Docker image to Docker Hub
  run: docker push ${ secrets.DOCKER_USERNAME }/${ secrets.DOCKER_REPOSITORY }:latest

- name: Set up Google Cloud SDK
  uses: google-github-actions/setup-gcloud@v0.2.0
  with:
    service_account_key: ${ secrets.GKE_CREDENTIALS }
    project_id: softwares-escalaveis-at

- name: Instala o gke-gcloud-auth-plugin
  run: gcloud components install gke-gcloud-auth-plugin

- name: Deploy
  run: |
    gcloud container clusters get-credentials k8s-at --zone us-central1
    kubectl apply -f kubernetes/deployment.yml

```

1. Definição do Workflow

- **name:** Nome do workflow, neste caso, “Deploy”
- **on:** Define os eventos que acionam o workflow, que, neste caso, são push e pull_request para a branch main.

2. Job: build














- **Ambiente:** Executando em um runner *ubuntu-latest*.
- **Passos (steps):**
 - i. **Checkout code:** Faz o checkout do código fonte do repositório.
 - ii. **Set up JDK 17:** Configura o JDK 17 usando a ação do GitHub.
 - iii. **Build with Maven:** Executa o comando `mvn clean install` para compilar o projeto Java.
 - iv. **Log in to Docker Hub:** Realiza login no Docker Hub usando credenciais armazenadas nos *secrets* do GitHub.

- v. **Build Docker image:** Constrói a imagem Docker a partir do Dockerfile do projeto e a nomeia conforme as variáveis do Docker Hub.
- vi. **Push Docker image to Docker Hub:** Envia a imagem Docker criada para o repositório no Docker Hub.
- vii. **Set up Google Cloud SDK:** Configura o Google Cloud SDK utilizando a conta de serviço armazenada nos segredos do GitHub.
- viii. **Install gke-gcloud-auth-plugin:** Instala o plugin gke-gcloud-auth-plugin, necessário para autenticação com o Kubernetes.
- ix. **Deploy:** Executa os seguintes comandos:
 - 1. Conecta ao cluster GKE usando gcloud container clusters get-credentials.
 - 2. Aplica as configurações de deployment definidas no arquivo kubernetes/deployment.yml usando o kubectl.

O processo foi concluído com sucesso:

build

succeeded 28 minutes ago in 4m 19s

- >  Set up job
- >  Checkout code
- >  Set up JDK 17
- >  Build with Maven
- >  Log in to Docker Hub
- >  Build Docker image
- >  Push Docker image to Docker Hub
- >  Set up Google Cloud SDK
- >  Install gke-gcloud-auth-plugin
- >  Deploy
- >  Post Set up JDK 17
- >  Post Checkout code
- >  Complete job