INSTITUTO INFNET

ESCOLA SUPERIOR DE TECNOLOGIA

GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

Microsserviços e DevOps com Spring Boot e Spring Cloud

AT

Alunos: Daniel Gomes Lipkin

23 de set. de 2024

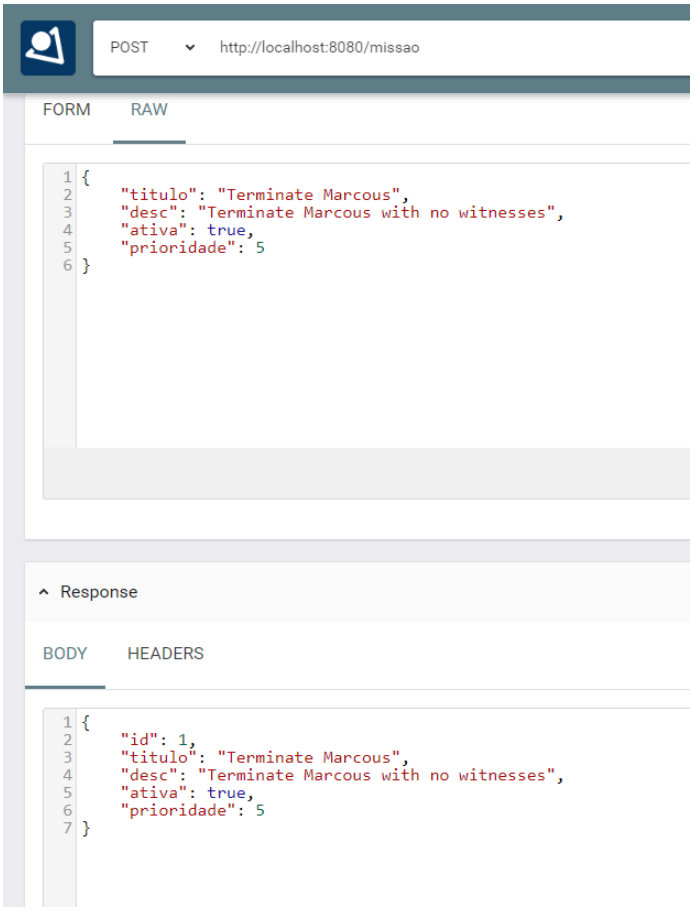
Questões

1 e 2) API Restful com Eureka e Gateway e testes dos endpoints

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
LS-FUNCIONARIO-SERVICE	n/a (1)	(1)	UP (1) - LS-funcionario-service_0:9fb7f4dd12f1986494a7656d36e18c18
LS-GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 16d93945db25:LS-gateway-service:8080
LS-MISSAO-SERVICE	n/a (1)	(1)	UP (1) - LS-missao-service_0:4811447a9a577ed1270e98a7f2a6f6b3

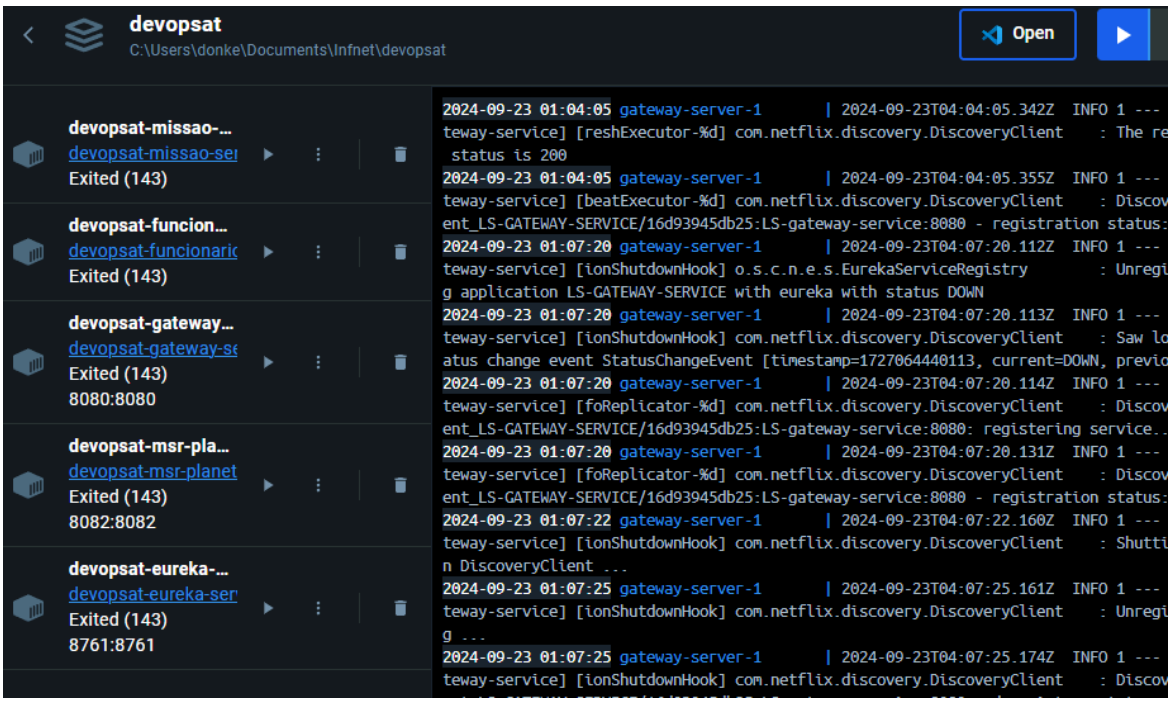
LS-Missao-Service :



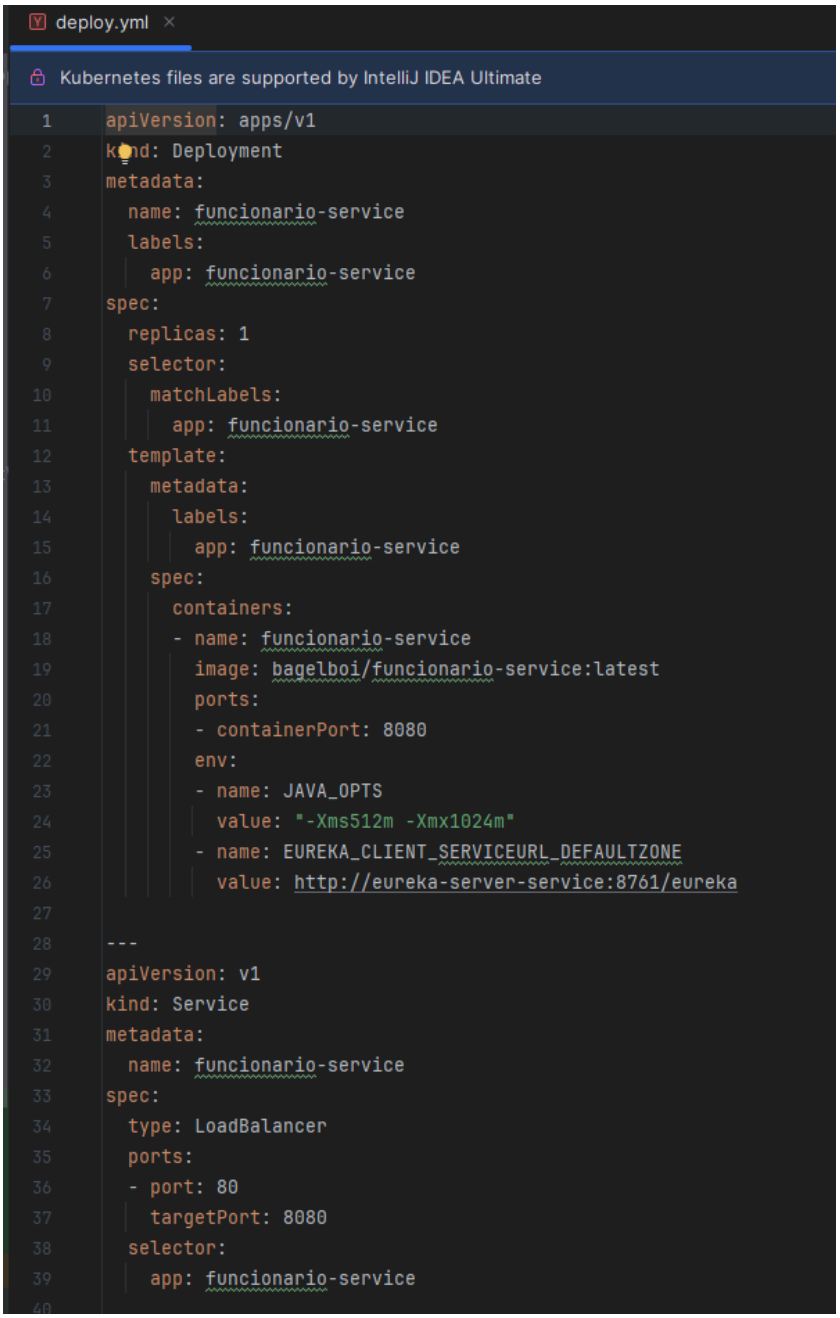
3) Para migrar uma aplicação Docker para Kubernetes é necessario criar para cada imagem dos micro serviços uma configuração de Serviço (para expor a porta á redes externas e gerenciar o endereçamento) e uma configuração de Deployment (para fazer o trabalho de balanceamento de carga e escalonamento). Isso é geralmente feito em um arquivo .yaml que depois é passado para o comando *"kubectl apply -f nome.yaml".* Vera que com *"kubectl get pods"* ou *"kubectl get services"* aparecera os serviços e pods criados para as imagens. Volumes para banco de dados e redes internas tambem podem ser definidos opcionalmente.

Os beneficios estão principalmente no fato de ser nativo a nuvem, com varias empresas (Google, Amazon, etc) fornecendo plataformas para hostear clusters de Kubernetes. Alem disso ha tambem o fato que um container pode ser hosteado varias vezes (pods) em diversos hosts diferentes (nodes), replicas de imagens são ajustadas automaticamente dependendo do trafego, balanceamento de carga automática entre diferentes hosts e containers, reinicialização automática caso caia um dos sistemas, e integração sem dificuldades com workflows de CI/CD quando for atualizado alguma imagem ou uma parte da configuração.

4) Container Docker



5) Implementação Kubernetes
LS-Funcionario-Service :

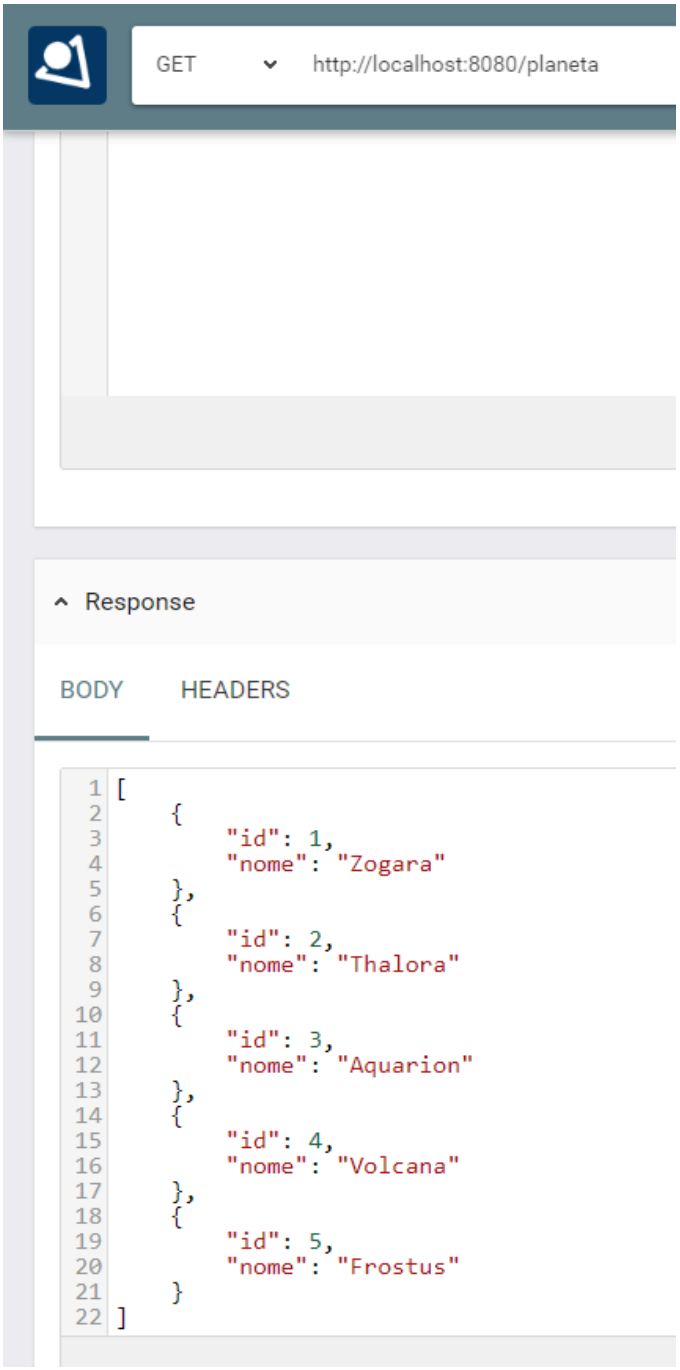


6) Cliente reativo com GET e POST
LS-Funcionario-Service :

7) Consumir API externa e Test Containers

LS-Missao-Service consome da aplicação MSR-Planets

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)  Bagelboi
public class PlanetContainerTest {

    public static GenericContainer<?> planetService = new GenericContainer<>( dockerImageName: "bagelboi/msrplanets:latest")
            .withExposedPorts(8082);

    private static WebClient webClient;  3 usages

    @BeforeAll  Bagelboi
    public static void setUp() {
        String baseUrl = "http://" + planetService.getHost() + ":8082" + "/msr/planeta";
        webClient = WebClient.builder().baseUrl(baseUrl).build();
    }

    @Test  Bagelboi
    public void testGetAllPlanets() {
        Mono<List<PlanetaDTO>> planetListMono = webClient.get() RequestHeadersUriSpec<capture of ?>
                .retrieve() ResponseSpec
                .bodyToFlux(PlanetaDTO.class) Flux<PlanetaDTO>
                .collectList();
    }

    @Test  Bagelboi
    public void testGetPlanetById() {
        Long planetId = 1L; // Assuming a planet with ID 1 exists
        Mono<PlanetaDTO> planetMono = webClient.get() RequestHeadersUriSpec<capture of ?>
                .uri( uri: "/{id}", planetId) capture of ?
                .retrieve() ResponseSpec
                .bodyToMono(PlanetaDTO.class);
    }
}
```


GET
http://localhost:8080/planeta
Response
BODY
HEADERS
[
{
"id": 1,
"nome": "Zogara"
},
{
"id": 2,
"nome": "Thalora"
},
{
"id": 3,
"nome": "Aquarion"
},
{
"id": 4,
"nome": "Volcana"
},
{
"id": 5,
"nome": "Frostus"
}
]

8) Commit no Github

```
C:\Users\donke\Documents\Infnet\devopsat\LS-funcionario-service>git add .

C:\Users\donke\Documents\Infnet\devopsat\LS-funcionario-service>git commit -m "push test"
[main ae61bc2] push test
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 push.txt

C:\Users\donke\Documents\Infnet\devopsat\LS-funcionario-service>git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

C:\Users\donke\Documents\Infnet\devopsat\LS-funcionario-service>git add .

C:\Users\donke\Documents\Infnet\devopsat\LS-funcionario-service>git commit -m "actions fix"
[main b074401] actions fix
 1 file changed, 1 insertion(+)
```

```
C:\Users\donke\Documents\Infnet\devopsat\LS-funcionario-service>git push
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 700 bytes | 350.00 KiB/s, done.
Total 8 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To https://github.com/Bagelboi/ls-funcionario-service.git
   ac4d0df..b074401  main -> main
```

9) O Github Actions automatiza o processo de CI/CD fornecendo maquinas virtuais (Runners) que executam um conjunto de comandos estilo cmd/Powershell (Steps) quando um evento for disparado no repositório (Pull, Push, etc). Esse conjunto de comandos se chama Job e 1 ou mais Jobs podem ser executados em um Workflow definido por um arquivo .yaml em *".github/workflows"*.

Os beneficios são o fato de você poder atualizar seu código-fonte e o repositório sem precisar manualmente testar ou atualizar outros aspectos de produção como a imagem Docker ou a configuração do Kubernetes graças aos Jobs, alem de testar a compatibilidade do programa em outras plataformas com as opções de Runners.

10) Crie um arquivo .yaml de configuração Kubernetes para um dos serviços Spring Boot e depois coloque em um dos Workflows do repositório.

- name: Nome do Step
run: kubectl apply -f ./deploy.yml

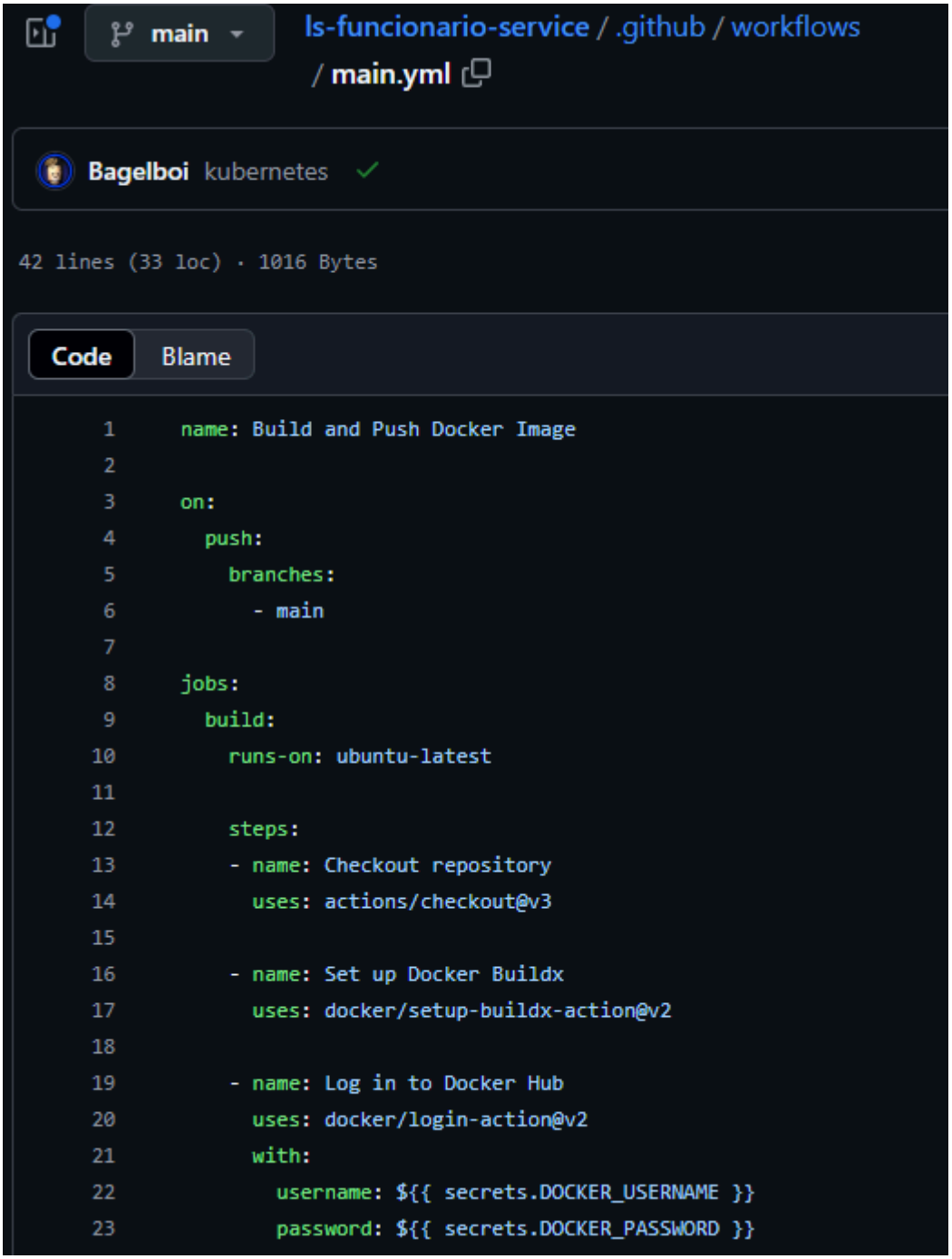
Antes disso podem ser executados Steps para se autenticar e conectar com serviços de nuvem para clusters a fim de utilizá-los, como exemplo o Google Kubernetes Engine usando

- name: Set up Google Cloud

uses: google-github-actions/setup-gcloud@v0.2.0
with:
service_account_key: ${{secrets.GKE_LOGIN}} #Colocar credencias do GKE nos Secrets do Github

- name: Deploy to GKE
run:
gcloud container clusters get-credentials ls-cluster --zone us-central1-a

11) Actions



```yaml
  - name: Build and push Docker image
    uses: docker/build-push-action@v5
    with:
      context: .
      file: ./Dockerfile
      push: true
      tags: ${{ secrets.DOCKER_USERNAME }}/funcionario-service:latest
```

12) GKE

```yaml
# - name: Set up Google Cloud
  # uses: google-github-actions/setup-gcloud@v0.2.0
  # with:
    # service_account_key: ${{secrets.GKE_LOGIN}}

# - name: Deploy to GKE
  # run: |
    # gcloud container clusters get-credentials ls-cluster --zone us-central1-a
    # kubectl apply -f ./deploy.yml
```

Código

Github :

- https://github.com/Bagelboi/ls-funcionario-service
- https://github.com/Bagelboi/ls-missao-service
- https://github.com/Bagelboi/ls-eureka-server
- https://github.com/Bagelboi/ls-gateway-server

Docker :

- bagelboi/funcionario-service
- bagelboi/missao-service
- bagelboi/ls-eureka-server
- bagelboi/ls-gateway-service
- bagelboi/msrplanets