Universidade Federal de Uberlândia

Programa de Pós-Graduação em Engenharia Biomédica

Tarefa 5 – ASA – Arquitetura de Sistemas Aplicado

Uberlândia, julho de 2023

Descrição:

A tarefa consiste em criar uma aplicação rodando em K8s com os seguintes requisitos:

- POD's trocando informações entre si (rede);
- Arquivos de configuração (ConfigMap);
- Persistência de dados.

Conceitos básicos usados no trabalho:

Kubernetes:

Kubernetes é uma plataforma de código aberto que facilita a implantação, o gerenciamento e a escalabilidade de aplicativos em contêiner. Ele fornece um ambiente para executar e coordenar vários contêineres em um cluster de máquinas. O Kubernetes automatiza tarefas, como balanceamento de carga, recuperação de falhas e dimensionamento automático, para garantir a disponibilidade e a confiabilidade dos aplicativos.

Minikube:

Minikube é uma ferramenta que permite executar um cluster Kubernetes em um único nó de máquina local. É útil para desenvolvimento e testes, pois fornece uma maneira fácil de criar um ambiente Kubernetes localmente sem precisar de um cluster completo.

Docker:

Docker é uma plataforma de código aberto que permite empacotar, distribuir e executar aplicativos em contêineres. Ele fornece uma maneira de isolar aplicativos e suas dependências em contêineres leves e portáteis. Os contêineres Docker fornecem consistência entre ambientes de desenvolvimento, teste e produção, facilitando a implantação e o gerenciamento de aplicativos.

Arquivos de Deployment:

Os arquivos de deployment no Kubernetes são usados para descrever como os aplicativos devem ser implantados e executados em um cluster. Esses arquivos geralmente são escritos em YAML e incluem informações sobre os contêineres a serem executados, portas expostas, recursos de computação necessários e outras configurações relevantes. Os arquivos de deployment permitem que o Kubernetes gerencie a criação, atualização e escalabilidade dos aplicativos automaticamente.

ConfigMap:

ConfigMap é um recurso no Kubernetes usado para armazenar configurações não confidenciais, como variáveis de ambiente, arquivos de configuração ou outros dados de configuração. Ele permite que você defina essas configurações separadamente do código do aplicativo, facilitando a modificação e a reutilização dessas configurações sem precisar recriar ou reiniciar o aplicativo.

Secrets:

Secrets são semelhantes aos ConfigMaps, mas são usados para armazenar informações confidenciais, como senhas, chaves de API, certificados e outros dados sensíveis. Os Secrets são criptografados antes de serem armazenados no cluster Kubernetes e podem ser montados como volumes em contêineres ou injetados como variáveis de ambiente para uso seguro pelos aplicativos.

Desenvolvimento:

A arquitetura desenvolvida consiste em um duas imagens dockers: uma para hospedar o banco de dados (postgres); e outra uma API em python usando o framework flask. Foi realizado o deploy de ambas as imagens no ambiente K8s. A Figura 1 mostra uma ilustração da aplicação desenvolvida.

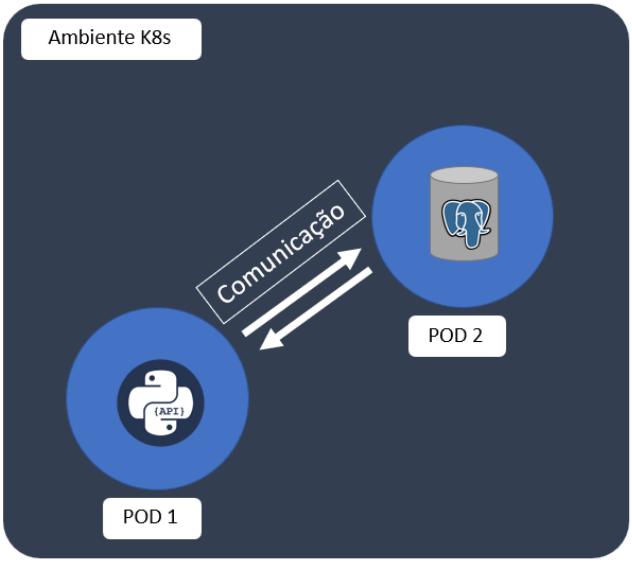

Figura 1 - Arquitetura do sistema desenvolvido. POD 1 representa a API em python e no POD 2 o banco de dados utilizado (postgres). Os PODs trocam informações entre si buscando e enviando dados da aplicação para o banco de dados.

A API consiste no backend desenvolvido em python utilizando o framework Flask responsável por processar os pedidos de um restaurante. O restaurante está dividido em 4 departamentos:

- Departamento de sanduíches
- Departamento de pratos prontos
- Departamento de bebidas
- Departamento de sobremesas

Cada pedido realizado consiste no nome do cliente, o departamento para qual o pedido será realizado e a descrição do prato desejado. A Figura 2 ilustra o Diagrama Entidade Relacionamento (DER) do banco de dados criado onde existe uma relação de n/1 entre a tabela de pedidos e departamentos. A arquitetura do banco de dados é simples visto que a proposta da tarefa é de cunho didático e voltado para a orquestração de toda a arquitetura do sistema e não exclusivamente do banco de dados.

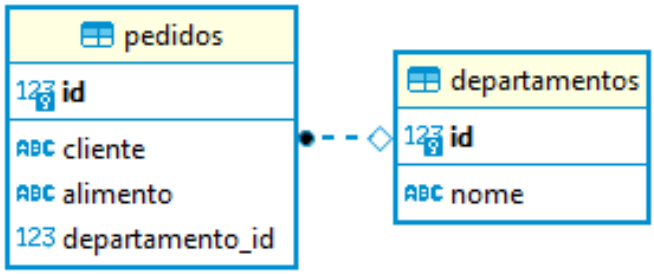

Figura 2 - Diagrama Entidade Relacionamento do banco de dados.

A API possui endpoints para administrar a logística de fazer pedidos e buscar os pedidos realizados. Para tanto, foi desenvolvido um CRUD de pedidos. Abaixo são listados os endpoints possíveis para os pedidos com seus devidos protocolos HTTP:

- Buscar todos os pedidos:
  - [GET] /pedidos

- Buscar pedido por id:
  - [GET] /pedido/<id>

- Adicionar um pedido:
  - [POST] /pedido
  - Exemplo de body da requisição:

```
{
    "cliente": "Eduardo",
    "alimento": "PF - Frango",
    "departamento_id": "2"
}
```

- Editar um pedido:
  - [PUT] /pedido
  - Exemplo de body da requisição:

```
{
    "id": "1",
    "cliente": "Eduardo",
    "alimento": "PF - Frango - sem cebola",
    "departamento_id": "2"
}
```

- Deletar um pedido:

  [DELETE] /pedido/<id>

A imagem do banco de dados necessita de uma senha para que o banco seja configurado e instanciado. Dessa maneira, foi utilizado um arquivo configuração de secrets dentro do ambiente K8s (abaixo). Arquivos de configuração de secrets dentro do K8s necessitam que os valores sejam criptografados em base64, portanto, o valor visto na tag 'POSTGRES_PASSWORD' consiste no valor da senha do banco de dados em base64.

```
apiVersion: v1
kind: Secret
metadata:
    name: postgres-secret
type: Opaque
data:
   POSTGRES_PASSWORD: YWRtaW4xMjM=
```

Outro arquivo de configuração de ambiente do K8s utilizado foi o configMap. Ao instanciar um POD com a imagem da aplicação em questão, o responsável poderá alterar a string de conexão do banco de dados direcionando a aplicação para a base que desejar. No arquivo de configuração (abaixo) é possível notar que a senha do banco de dados encontra-se exposta na string de conexão apresentando uma brecha na segurança do sistema, por expor uma informação confidencial. Trabalhos futuros poderão realizar alterações na arquitetura do sistema para que a senha da base de dados não fique exposta no arquivo de configuração.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-pedidos-configmap
data:
  database_url:
postgresql+psycopg2://postgres:admin123@10.104.19.72:5432/postgres
```

Cada POD instanciado (postgres e API) possui seu próprio serviço e deployment. Para instanciar o POD da API deve-se utilizar as instruções abaixo:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-pedidos
  labels:
    app: app-pedidos
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app-pedidos
  template:
    metadata:
      labels:
        app: app-pedidos
    spec:
      containers:
      - name: app-pedidos
        image: doubleduardo/app-pedidos
        ports:
        - containerPort: 5000
        env:
        - name: DATABASE_URL_ARG
          valueFrom:
            configMapKeyRef:
              name: app-pedidos-configmap
              key: database_url
---
apiVersion: v1
kind: Service
metadata:
  name: app-pedidos-service
spec:
  selector:
    app: app-pedidos
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
      nodePort: 30000
```

Para instanciar o POD da base de dados deve-se utilizar as instruções abaixo:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
  - protocol: "TCP"
    port: 5432
    targetPort: 5432
    nodePort: 32432
  type: LoadBalancer

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-deployment
spec:
  selector:
    matchLabels:
      app: postgres
  replicas: 1
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres
        ports:
        - containerPort: 5432
        env:
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: POSTGRES_PASSWORD
```

Conclusão:

A tarefa consistia em desenvolver a arquitetura de uma aplicação com persistência de dados e comunicação entre PODs utilizando arquivos de configuração. Dessa maneira, a aplicação desenvolvida e orquestrada no ambiente K8s apresenta os requisitos necessários à tarefa. Contendo comunicação entre PODs, arquivos de configuração (configMap) e persistência de dados. A aplicação desenvolvida encontra-se disponível no github, em: https://github.com/eduardoborgesgouveia/k8s_api_postgres_example.

Esses conceitos são fundamentais para entender e trabalhar com arquiteturas baseadas em contêineres e Kubernetes. Eles ajudam a simplificar o processo de implantação, gerenciamento e configuração de aplicativos distribuídos, permitindo uma escalabilidade eficiente e uma infraestrutura altamente disponível.