Centro Federal de Educação Tecnológica de Minas Gerais
Engenharia de Computação

TÓPICOS ESPECIAIS EM FUNDAMENTOS DA ENGENHARIA DE COMPUTAÇÃO:
PROGRAMAÇÃO EM LINUX

Trabalho 1

Eduardo Almeida
João Paulo da Cunha Faria

Divinópolis - MG
Junho de 2025

Sumário

Identificação e Tema

Tema Sorteado

Conforme as especificações do trabalho prático da disciplina de Programação Linux, o tema designado para a equipe é o de número 2: **Deployment com MySQL + Adminer**. O objetivo central deste trabalho é orquestrar uma aplicação de duas camadas, composta por um banco de dados MySQL e uma interface de gerenciamento web Adminer, utilizando os recursos fundamentais do Kubernetes. A tarefa exige a criação de objetos Deployment e Service separados para cada componente. Um requisito crítico do projeto é a utilização de um PersistentVolumeClaim (PVC) para garantir a persistência dos dados do MySQL, desacoplando o ciclo de vida dos dados do ciclo de vida efêmero dos contêineres.

Arquitetura da Solução e Explicação dos Recursos Kubernetes

Visão Geral da Arquitetura

A solução proposta implementa uma arquitetura de microsserviços clássica, onde a interface de gerenciamento (Adminer) e o banco de dados (MySQL) operam como componentes independentes e desacoplados dentro do cluster Kubernetes. A comunicação entre eles é gerenciada pela rede interna do cluster, e o armazenamento de dados críticos é externalizado para um volume persistente. A arquitetura pode ser visualizada da seguinte forma:

- **Componente MySQL**:
  - Gerenciado por um objeto Deployment, que assegura que uma instância (réplica) do contêiner MySQL esteja sempre em execução.
  - Expõe sua porta de comunicação (3306) através de um Service do tipo ClusterIP.
  - Este serviço atribui um nome DNS estável (mysql-service) e um endereço IP interno, tornando o banco de dados acessível apenas por outras aplicações dentro do cluster, como o Adminer.
  - Esta é uma prática de segurança fundamental para proteger o banco de dados contra acessos externos não autorizados.
  - Para persistir os dados, o Pod do MySQL utiliza um PersistentVolumeClaim (PVC) para solicitar e se conectar a um PersistentVolume (PV).
  - Este volume é mapeado para um diretório no sistema de arquivos do nó hospedeiro (host), garantindo que os dados do banco de dados sobrevivam a reinicializações, falhas ou atualizações do Pod.
- **Componente Adminer**:

  - Também gerenciado por um Deployment, que controla o ciclo de vida do contêiner do Adminer.
  - Expõe sua interface web (porta 8080) através de um Service do tipo NodePort.
  - Este tipo de serviço torna a aplicação acessível de fora do cluster, mapeando a porta do serviço para uma porta estática em cada nó do cluster.
  - Isso permite que os usuários acessem a interface do Adminer através do navegador, utilizando o IP de um nó e a porta designada.
  - O Adminer se conecta ao MySQL utilizando o nome DNS do serviço do banco de dados (mysql-service), um exemplo prático do mecanismo de descoberta de serviços (service discovery) nativo do Kubernetes.

- **Componentes de Suporte**:
  - Um objeto Secret é utilizado para armazenar de forma segura a senha de root do MySQL.
  - Esta credencial é injetada no contêiner do MySQL como uma variável de ambiente, evitando a exposição de dados sensíveis diretamente nos manifestos de configuração.

Esta arquitetura não apenas cumpre os requisitos do trabalho, mas também demonstra princípios de design de sistemas robustos e seguros em um ambiente orquestrado, como isolamento de componentes, comunicação controlada e gerenciamento de estado.

O Recurso Deployment

O Deployment é um recurso de alto nível da API do Kubernetes que gerencia de forma declarativa o ciclo de vida de aplicações. Sua principal função é garantir que um número especificado de réplicas de um Pod esteja sempre em execução e disponível. Ao definir um Deployment, o usuário especifica o estado desejado da aplicação — como a imagem do contêiner, o número de réplicas e a estratégia de atualização — e o controlador do Deployment trabalha ativamente para manter o estado real do cluster em conformidade com esse estado desejado.
As principais funcionalidades de um Deployment incluem:

- **Gerenciamento de Réplicas**: Ele utiliza um ReplicaSet subjacente para garantir que o número de Pods especificado no campo `replicas` esteja sempre em execução. Se um Pod falhar, o ReplicaSet automaticamente cria um novo para substituí-lo.

- **Atualizações Declarativas**: Para atualizar uma aplicação, basta modificar o template do Pod no manifesto do Deployment (por exemplo, alterando a tag da imagem do contêiner). O Kubernetes gerenciará a transição da versão antiga para a nova de forma controlada.

- **Estratégias de Atualização**: A estratégia padrão é a `RollingUpdate` (atualização contínua), que substitui gradualmente os Pods antigos pelos novos, garantindo que a aplicação permaneça disponível durante o processo e minimizando o tempo de inatividade (zero downtime). Outra estratégia é a `Recreate`, que desliga todos os Pods antigos antes de criar os novos, resultando em um breve período de indisponibilidade.
- **Rollbacks**: O Deployment mantém um histórico de revisões, permitindo reverter facilmente para uma versão anterior caso uma nova atualização apresente problemas.

Neste projeto, Deployments são utilizados para gerenciar tanto o MySQL quanto o Adminer, aproveitando sua capacidade de autogerenciamento e facilidade de atualização, conforme solicitado nas especificações do trabalho.

O Recurso Service

Um Service no Kubernetes é uma abstração que define um conjunto lógico de Pods e uma política para acessá-los. Como os Pods são efêmeros e seus endereços IP podem mudar sempre que são recriados, o Service fornece um ponto de acesso estável e confiável. Ele atua como um balanceador de carga interno, distribuindo o tráfego de rede entre os Pods que correspondem a um determinado seletor de labels. Para este trabalho, foram utilizados dois tipos de Service:

Service do tipo ClusterIP

O ClusterIP é o tipo de serviço padrão no Kubernetes. Ele expõe o serviço em um endereço IP interno, acessível apenas de dentro do cluster. Este tipo é ideal para comunicação entre microsserviços, como a conexão do Adminer com o banco de dados MySQL. Ao criar um Service ClusterIP para o MySQL, garantimos que o banco de dados não seja exposto à rede externa, uma prática essencial de segurança que limita a superfície de ataque da aplicação.

Service do tipo NodePort

O Service do tipo NodePort expõe a aplicação em uma porta estática em cada nó do cluster. O Kubernetes aloca uma porta de um intervalo predefinido (geralmente 30000-32767), e qualquer tráfego direcionado a essa porta em qualquer nó é encaminhado para o serviço. Este tipo é útil para expor aplicações para acesso externo durante o desenvolvimento e teste, sem a necessidade de configurar um balanceador de carga complexo. Para o Adminer, um Service NodePort permite que a interface web seja acessada diretamente do navegador do desenvolvedor, facilitando a interação e a verificação do banco de dados.

Persistência de Dados com PersistentVolume (PV) e PersistentVolumeClaim (PVC)

O ciclo de vida de um Pod no Kubernetes é, por natureza, efêmero. Quando um Pod é encerrado ou reiniciado, todos os dados armazenados em seu sistema de arquivos são perdidos. Para aplicações que precisam manter estado (stateful), como bancos de dados, é fundamental que os dados persistam independentemente do ciclo de vida do Pod. O Kubernetes aborda essa necessidade através de dois recursos principais: PersistentVolume (PV) e PersistentVolumeClaim (PVC).

- **PersistentVolume (PV)**: É uma abstração para uma peça de armazenamento físico no cluster (como um disco de rede, um diretório local no nó ou um volume em nuvem) que foi provisionada por um administrador. Um PV é um recurso do cluster, assim como um nó é um recurso do cluster. Ele possui um ciclo de vida independente de qualquer Pod que o utilize.
- **PersistentVolumeClaim (PVC)**: É uma solicitação de armazenamento feita por um usuário ou uma aplicação. É análogo a como um Pod consome recursos de CPU e memória de um nó ; um PVC consome recursos de armazenamento de um PV. O PVC especifica o tamanho do armazenamento e os modos de acesso necessários (por exemplo, ReadWriteOnce, que permite que o volume seja montado como leitura e escrita por um único nó).

Neste projeto, que utiliza o Minikube para um ambiente de desenvolvimento local, a abordagem adotada é a criação de um PV do tipo `hostPath`. Este tipo de volume monta um diretório do sistema de arquivos do nó hospedeiro (a máquina onde o Minikube está rodando) dentro do Pod. Embora o `hostPath` não seja recomendado para ambientes de produção de múltiplos nós (pois os dados ficariam presos a um nó específico), ele é perfeitamente adequado para um cluster de nó único como o Minikube, simulando de forma eficaz o armazenamento persistente. O Deployment do MySQL, através de seu template de Pod, especifica um PVC que solicita o armazenamento. O Kubernetes então liga (bind) este PVC ao PV compatível que criamos, garantindo que os dados do MySQL sejam escritos no diretório do host e persistam entre as reinicializações do Pod.

Gerenciamento de Credenciais com Secret

Armazenar informações sensíveis, como senhas de banco de dados, chaves de API ou tokens, diretamente em manifestos YAML ou imagens de contêiner é uma prática de segurança extremamente arriscada. Tais informações se tornariam visíveis a qualquer pessoa com acesso ao repositório de código ou ao cluster. Para mitigar esse risco, o Kubernetes oferece o recurso Secret.

Um Secret é um objeto da API projetado para armazenar pequenas quantidades de dados sensíveis. Os dados dentro de um Secret são armazenados em formato base64. É crucial entender que

base64 é uma codificação, não uma criptografia, e pode ser facilmente decodificada. No entanto, o uso de Secrets oferece várias vantagens:

- **Desacoplamento**: Separa as credenciais da configuração da aplicação.
- **Controle de Acesso**: O acesso a Secrets pode ser controlado através de políticas de RBAC (Role-Based Access Control).
- **Gerenciamento Centralizado**: Facilita a rotação e o gerenciamento de credenciais sem a necessidade de reconstruir imagens de contêiner ou alterar manifestos de Deployment.

Neste projeto, um Secret é criado para armazenar a senha de root do MySQL. O Deployment do MySQL é então configurado para referenciar este Secret e injetar o valor da senha como uma variável de ambiente no contêiner em tempo de execução, utilizando a diretiva `valueFrom.secretKeyRef`. Esta abordagem garante que a senha não seja exposta no manifesto do Deployment e segue as melhores práticas de segurança do Kubernetes.

Análise Crítica: Deployment vs. StatefulSet para Bancos de Dados

O enunciado do trabalho prático especifica o uso de um Deployment para gerenciar o Pod do MySQL. Para um cenário de introdução com uma única instância de banco de dados, esta abordagem é funcional e serve bem ao propósito de ensinar os conceitos básicos de Deployments e PersistentVolumeClaims. No entanto, em um contexto de produção, especialmente para sistemas de banco de dados que exigem alta disponibilidade e replicação, o uso de um Deployment é considerado uma má prática. A ferramenta correta para esta tarefa no ecossistema Kubernetes é o StatefulSet.

A principal distinção reside na forma como esses controladores tratam a identidade e o estado dos Pods que gerenciam. Deployments são projetados para aplicações *stateless* (sem estado), onde cada Pod é idêntico e intercambiável. Se um Pod é substituído, o novo Pod é uma cópia exata, com um novo nome e um novo endereço IP, sem qualquer garantia de ordem. Por outro lado, StatefulSets são projetados especificamente para aplicações *stateful* (com estado), como bancos de dados, que requerem garantias sobre a identidade e o armazenamento.

As principais características que tornam o StatefulSet a escolha superior para bancos de dados são:

- **Identidade de Rede Estável e Única**: Cada Pod em um StatefulSet recebe um nome de host previsível e persistente, seguindo um padrão ordinal (por exemplo, mysql-0, mysql-1, mysql-2). Essa identidade é mantida mesmo que o Pod seja reiniciado ou reagendado para outro nó.
- **Armazenamento Persistente Estável**: Um StatefulSet pode usar um `volumeClaimTemplates` para criar um PersistentVolumeClaim único para cada Pod. Isso significa que `mysql-0` sempre estará associado ao seu volume de dados específico, e `mysql-1` ao seu, e assim por

diante. Isso é fundamental para sistemas de banco de dados replicados, onde cada nó deve manter seu próprio conjunto de dados.

- **Implantação e Escalonamento Ordenados**: StatefulSets garantem que os Pods sejam criados, atualizados e excluídos em uma ordem estrita e previsível. Por exemplo, ao escalar de 2 para 3 réplicas, o Pod `mysql-2` só será criado depois que `mysql-0` e `mysql-1` estiverem em estado Running e Ready. Essa ordenação é vital para a inicialização e manutenção de clusters de banco de dados.

Tentar escalar um Deployment de MySQL para mais de uma réplica usando um único PVC resultaria em um cenário de falha catastrófica: múltiplas instâncias do MySQL tentariam montar e escrever no mesmo sistema de arquivos simultaneamente, levando à corrupção de dados e ao colapso do serviço.

Tabela 1 – Comparativo: Deployment vs. StatefulSet

Característica	**Deployment**	**StatefulSet**
Caso de Uso Principal	Aplicações *stateless* (sem estado), como servidores web e APIs.	Aplicações *stateful* (com estado), como bancos de dados, filas de mensagens e sistemas distribuídos.
Identidade dos Pods	Pods são efêmeros e intercambiáveis, com nomes e IPs aleatórios.	Pods possuem identidades de rede estáveis e únicas (ex: app-0, app-1).
Armazenamento	Geralmente usa um único PVC compartilhado por todas as réplicas ou armazenamento efêmero.	Cada réplica obtém seu próprio PVC, garantindo isolamento e persistência de dados por Pod.
Estratégia de Atualização	RollingUpdate (padrão) ou Recreate. A ordem não é garantida.	RollingUpdate (padrão). As atualizações ocorrem em ordem inversa (ex: app-N até app-0).
Escalonamento	Réplicas são criadas ou destruídas em paralelo, sem ordem definida.	Réplicas são criadas ou destruídas em ordem sequencial (ex: app-0, app-1,...).

Em conclusão, embora este trabalho utilize um Deployment para fins didáticos, é imperativo reconhecer que para qualquer implementação de banco de dados em produção, um StatefulSet é a abordagem tecnicamente correta e recomendada pela indústria.

Configuração do Ambiente de Desenvolvimento

Para realizar este trabalho prático, é necessário configurar um ambiente Kubernetes local. O Minikube foi a ferramenta escolhida, pois permite executar um cluster Kubernetes de nó único dentro de uma máquina virtual ou contêiner Docker, sendo ideal para fins de aprendizado e

desenvolvimento. Os passos a seguir foram executados em um sistema operacional baseado em Ubuntu.

Requisitos de Sistema

- **Sistema Operacional**: Ubuntu 20.04 LTS ou superior (ou outra distribuição Linux compatível).
- **Virtualização**: Suporte à virtualização de hardware habilitado na BIOS.
- **Recursos**: Pelo menos 2 CPUs, 4 GB de RAM e 20 GB de espaço em disco livre.
- **Software**: Docker instalado e em execução, e acesso à internet para download de ferramentas e imagens de contêiner.

Instalação e Inicialização do Minikube

O Minikube simplifica a execução do Kubernetes em uma máquina local.

1. Download do binário do Minikube:

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/
    latest/minikube-linux-amd64

```

2. Instalar o Minikube no sistema:

```
sudo install minikube /usr/local/bin/

```

3. Iniciar o cluster Kubernetes local: O comando a seguir inicia um cluster Minikube utilizando o driver do Docker. Isso significa que o nó do Kubernetes será executado como um contêiner Docker, o que é mais leve e rápido do que usar uma máquina virtual completa.

```
minikube start --driver=docker

```

4. Verificar o status do cluster: Após a inicialização, é crucial verificar se o nó do cluster está pronto para receber cargas de trabalho.

```
kubectl get nodes

```

   A saída esperada deve mostrar um nó com o status `Ready`, confirmando que o ambiente está configurado corretamente e pronto para a implantação da aplicação.

Manifestos YAML Comentados

A abordagem declarativa é um dos pilares do Kubernetes. Em vez de executar comandos imperativos para criar cada recurso, descrevemos o estado final desejado em arquivos de manifesto no formato YAML (YAML Ain't Markup Language). O Kubernetes então trabalha para reconciliar o estado atual do cluster com o estado descrito nesses arquivos. Essa prática de manter a configuração como código em arquivos versionáveis é a base do *GitOps*, um paradigma moderno de DevOps onde um repositório Git serve como a única fonte da verdade para a infraestrutura e as aplicações. Ao criar e gerenciar estes manifestos, não estamos apenas configurando o Kubernetes, mas também adotando uma prática de engenharia robusta que promove automação, rastreabilidade e colaboração.
A seguir, são apresentados todos os manifestos YAML necessários para a implantação da solução MySQL + Adminer, com comentários detalhados explicando cada campo e sua função.

mysql-secret.yaml

Este manifesto define o Secret que armazenará a senha de root do MySQL de forma segura.

```yaml
# apiVersion especifica a versao da API do Kubernetes a ser usada.
# Para o objeto Secret, a versao v1 e estavel.
apiVersion: v1
# kind especifica o tipo de objeto Kubernetes que estamos criando.
kind: Secret
# metadata contem dados que ajudam a identificar unicamente o objeto,
# como o nome e labels.
metadata:
  # O nome do Secret, que sera referenciado pelo Deployment do MySQL.
  name: mysql-secret
# type especifica o tipo de Secret. 'Opaque' e o tipo padrao e pode
# conter pares chave-valor arbitrarios.
type: Opaque
# data contem os dados sensiveis. Os valores devem ser codificados em base64.
data:
  # A chave 'MYSQL_ROOT_PASSWORD' corresponde a variavel de ambiente que a imagem
```

```
17   # oficial do MySQL espera para definir a senha do usuario
      root.
18   # O valor 'bWluaGEtc2VuaGEtc3VwZXItc2VjcmV0YQ==' e a string '
      minha-senha-super-secreta'
19   # codificada em base64.
20   # Para gerar seu proprio valor, use o comando:
21   # echo -n 'sua-senha' | base64
22   MYSQL_ROOT_PASSWORD: bWluaGEtc2VuaGEtc3VwZXItc2VjcmV0YQ==
```

Listing 1 – mysql-secret.yaml

mysql-pv-pvc.yaml

Este arquivo define tanto o PersistentVolume (PV) quanto o PersistentVolumeClaim (PVC) necessários para a persistência dos dados do MySQL.

```
1  # --- PersistentVolume (PV) Definition ---
2  # O PV representa um pedaco de armazenamento fisico no cluster.
3  apiVersion: v1
4  kind: PersistentVolume
5  metadata:
6    # Nome do PersistentVolume.
7    name: mysql-pv
8    # Labels podem ser usados para agrupar e selecionar PVs.
9    labels:
10     type: local
11 spec:
12   # storageClassName 'manual' indica que este PV deve ser
      ligado
13   # manualmente a um PVC que o solicite, em vez de ser
      provisionado dinamicamente.
14   storageClassName: manual
15   # capacity define a quantidade de armazenamento disponivel no
       volume.
16   capacity:
17     storage: 1Gi # 1 Gibibyte de armazenamento.
18   # accessModes define como o volume pode ser montado.
19   # ReadWriteOnce: pode ser montado como leitura e escrita por
      um unico no.
20   # Ideal para um banco de dados de instancia unica.
21   accessModes:
```

```
22      - ReadWriteOnce
23    # hostPath especifica que o volume utiliza um diretorio no no
        hospedeiro.
24    # ATENCAO: Adequado apenas para clusters de no unico como o
       Minikube.
25    hostPath:
26      # O caminho no sistema de arquivos do no onde os dados
       serao armazenados.
27      # Este diretorio deve existir no no do Minikube.
28      path: "/mnt/data/mysql"
29 ---
30 # --- PersistentVolumeClaim (PVC) Definition ---
31 # O PVC e uma solicitacao de armazenamento feita por uma
       aplicacao.
32 apiVersion: v1
33 kind: PersistentVolumeClaim
34 metadata:
35    # Nome do PersistentVolumeClaim, que sera referenciado pelo
       Deployment.
36    name: mysql-pvc
37 spec:
38    # storageClassName 'manual' para garantir que este PVC se
       ligue
39    # ao nosso PV definido manualmente.
40    storageClassName: manual
41    # accessModes deve corresponder aos modos de acesso do PV.
42    accessModes:
43      - ReadWriteOnce
44    # resources.requests especifica a quantidade de armazenamento
        que a aplicacao solicita.
45    # O Kubernetes encontrara um PV que satisfaca esta
       solicitacao.
46    resources:
47      requests:
48        storage: 1Gi # Solicita 1 Gibibyte de armazenamento.
```

Listing 2 – mysql-pv-pvc.yaml

mysql-deployment.yaml

Este manifesto define o Deployment que gerenciará o Pod do banco de dados MySQL.

```yaml
# apiVersion para Deployments e apps/v1, que e a versao estavel.
apiVersion: apps/v1
kind: Deployment
metadata:
  # Nome do Deployment.
  name: mysql-deployment
  # Labels para identificar e agrupar este Deployment.
  labels:
    app: mysql
spec:
  # replicas define o numero de Pods identicos a serem mantidos.
  # Para um banco de dados de instancia unica, usamos 1.
  replicas: 1
  # selector define como o Deployment encontra os Pods que ele deve gerenciar.
  # Deve co              rresponder aos labels definidos no template do Pod.
  selector:
    matchLabels:
      app: mysql
  # template descreve os Pods que serao criados.
  template:
    metadata:
      # Labels aplicados a cada Pod criado por este Deployment.
      labels:
        app: mysql
    spec:
      # containers e uma lista de um ou mais conteineres a serem executados no Pod.
      containers:
      - name: mysql
        # A imagem de conteiner a ser usada. E uma boa pratica fixar uma versao
        # especifica em vez de usar 'latest'.
        image: mysql:8.0
```

```
32        # ports define as portas que o conteiner expoe.
33        ports:
34        - containerPort: 3306 # Porta padrao do MySQL.
35        # env define as variaveis de ambiente para o conteiner.
36        env:
37        - name: MYSQL_ROOT_PASSWORD
38          # valueFrom permite obter o valor da variavel de
    outra fonte,
39          # neste caso, de um Secret.
40          valueFrom:
41            # secretKeyRef seleciona uma chave especifica de um
     Secret.
42            secretKeyRef:
43              # O nome do Secret que criamos anteriormente.
44              name: mysql-secret
45              # A chave dentro do Secret cujo valor queremos
    usar.
46              key: MYSQL_ROOT_PASSWORD
47        # resources define os requisitos e limites de recursos
    (CPU e memoria).
48        resources:
49          # requests sao os recursos minimos garantidos para o
    conteiner.
50          # O Kubernetes so agendara o Pod em um no que possa
    fornecer esses recursos.
51          requests:
52            memory: "256Mi" # 256 Mebibytes de RAM.
53            cpu: "250m"    # 250 millicores (0.25 de um nucleo
    de CPU).
54          # limits sao os recursos maximos que o conteiner pode
     consumir.
55          # Isso evita que um conteiner monopolize os recursos
    do no.
56          limits:
57            memory: "512Mi" # 512 Mebibytes de RAM.
58            cpu: "500m"    # 500 millicores (0.5 de um nucleo
    de CPU).
59        # livenessProbe verifica se o conteiner esta "vivo". Se
     a sonda falhar,
```

```
60        # o Kubernetes reiniciara o conteiner.
61        livenessProbe:
62          exec:
63            # Comando executado dentro do conteiner para
    verificar sua saude.
64            command: ["mysqladmin", "ping", "-h", "localhost"]
65          # Atraso inicial antes da primeira sonda ser
    executada.
66          initialDelaySeconds: 30
67          # Periodo entre as execucoes da sonda.
68          periodSeconds: 10
69          # Tempo limite para a execucao do comando.
70          timeoutSeconds: 5
71        # readinessProbe verifica se o conteiner esta pronto
    para receber trafego.
72        # Se a sonda falhar, o Pod e removido dos endpoints do
    Service.
73        readinessProbe:
74          exec:
75            command: ["mysqladmin", "ping", "-h", "localhost"]
76          initialDelaySeconds: 5
77          periodSeconds: 5
78          timeoutSeconds: 1
79        # volumeMounts define onde montar os volumes dentro do
    conteiner.
80        volumeMounts:
81        - name: mysql-persistent-storage
82          # O caminho dentro do conteiner onde o volume sera
    montado.
83          # /var/lib/mysql e o diretorio de dados padrao do
    MySQL.
84          mountPath: /var/lib/mysql
85      # volumes define os volumes que estao disponiveis para o
    Pod.
86      volumes:
87      - name: mysql-persistent-storage
88        # persistentVolumeClaim especifica que este volume e um
     PVC.
89        persistentVolumeClaim:
```

```
90            # O nome do PVC que criamos anteriormente.
91            claimName: mysql-pvc
```

Listing 3 – mysql-deployment.yaml

mysql-service.yaml

Este manifesto cria o Service do tipo ClusterIP para expor o MySQL internamente no cluster.

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    # Nome do Service. Este nome sera usado como o hostname DNS
     para
5    # outras aplicacoes (como o Adminer) se conectarem ao MySQL.
6    name: mysql-service
7  spec:
8    # type define o tipo de servico. ClusterIP e o padrao e expoe
       o
9    # servico em um IP interno ao cluster.
10   type: ClusterIP
11   # selector define o conjunto de Pods que este servico ira
     direcionar.
12   # Ele deve corresponder aos labels dos Pods do MySQL.
13   selector:
14     app: mysql
15   # ports define as portas do servico.
16   ports:
17   - protocol: TCP
18     # A porta que o servico expoe.
19     port: 3306
20     # A porta nos Pods para a qual o trafego sera encaminhado.
21     targetPort: 3306
```

Listing 4 – mysql-service.yaml

adminer-deployment.yaml

Este manifesto define o Deployment para a interface de gerenciamento Adminer.

```
1  apiVersion: apps/v1
2  kind: Deployment
```

```yaml
metadata:
  name: adminer-deployment
  labels:
    app: adminer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: adminer
  template:
    metadata:
      labels:
        app: adminer
    spec:
      containers:
      - name: adminer
        # Imagem oficial do Adminer.
        image: adminer
        # Variaveis de ambiente para o conteiner do Adminer.
        env:
        - name: ADMINER_DEFAULT_SERVER
          # Define o servidor padrao para o Adminer se conectar.
          # Usamos o nome do servico do MySQL, que o DNS do Kubernetes
          # resolvera para o IP interno do servico.
          value: mysql-service
        ports:
        - containerPort: 8080 # Adminer escuta na porta 8080.
        resources:
          requests:
            memory: "64Mi"
            cpu: "100m"
          limits:
            memory: "128Mi"
            cpu: "250m"
        # Sondas de saude para o Adminer.
        livenessProbe:
          httpGet:
```

```
            path: /
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 20
        readinessProbe:
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 10
```

Listing 5 – adminer-deployment.yaml

adminer-service.yaml

Este manifesto cria o Service do tipo NodePort para expor a interface web do Adminer externamente.

```
apiVersion: v1
kind: Service
metadata:
  name: adminer-service
spec:
  # type 'NodePort' expoe o servico em uma porta estatica em cada no.
  type: NodePort
  selector:
    # Corresponde aos labels dos Pods do Adminer.
    app: adminer
  ports:
  - protocol: TCP
    # A porta interna do servico.
    port: 8080
    # A porta no conteiner para a qual o trafego e direcionado.
    targetPort: 8080
    # O Kubernetes alocara automaticamente uma porta no intervalo 30000-32767
    # se nao for especificada. Esta sera a porta usada para acesso externo.
```

Listing 6 – adminer-service.yaml

Comandos para Implantação e Verificação

Com os manifestos YAML definidos, o próximo passo é aplicá-los ao cluster Kubernetes e, em seguida, verificar se todos os recursos foram criados e estão funcionando como esperado. A ordem de aplicação é importante para garantir que as dependências, como Secrets e PVCs, existam antes dos Deployments que as utilizam.

Sequência de Comandos kubectl apply

O comando `kubectl apply -f <nome-do-arquivo.yaml>` instrui o Kubernetes a criar ou atualizar os recursos definidos no arquivo especificado. A flag `-f` significa "filename".

1. Aplicar o Secret do MySQL:

```
kubectl apply -f mysql-secret.yaml

```

2. Aplicar o PersistentVolume e o PersistentVolumeClaim:

```
kubectl apply -f mysql-pv-pvc.yaml

```

3. Aplicar o Deployment e o Service do MySQL:

```
kubectl apply -f mysql-deployment.yaml
kubectl apply -f mysql-service.yaml

```

4. Aplicar o Deployment e o Service do Adminer:

```
kubectl apply -f adminer-deployment.yaml
kubectl apply -f adminer-service.yaml

```

Comandos de Verificação

Após aplicar os manifestos, é crucial inspecionar o estado dos recursos para garantir que a implantação foi bem-sucedida.

- **Verificar todos os recursos criados**: O comando `kubectl get all` é uma forma rápida de listar os principais recursos (Pods, Services, Deployments, ReplicaSets) no namespace padrão.

```
kubectl get all

```

O que procurar: Verifique se os Pods `mysql-deployment-...` e `adminer-deployment-...` estão com o status `Running`. Os Services `mysql-service` e `adminer-service` devem estar listados com seus respectivos tipos e portas. O Deployment e o ReplicaSet de cada aplicação devem mostrar o número desejado de réplicas como `READY`.

- **Inspecionar o PersistentVolumeClaim**: Verifique se o PVC foi corretamente ligado (Bound) ao PV.

```
kubectl get pvc mysql-pvc

```

O que procurar: O status deve ser `Bound`, indicando uma ligação bem-sucedida com o `mysql-pv`.

- **Detalhar um Pod para Diagnóstico**: O comando `kubectl describe pod <nome-do-pod>` fornece informações detalhadas, incluindo o estado atual, os volumes montados e, mais importante, a seção de Events no final, que registra as ações do Kubernetes relacionadas ao Pod.

```
# Substitua <nome-do-pod-mysql> pelo nome real do Pod do MySQL
kubectl describe pod <nome-do-pod-mysql>

```

O que procurar: Na seção Events, procure por mensagens como `Successfully assigned...`, `Pulling image...`, `Successfully pulled image...`, e `Created container...`. A ausência de eventos de erro é um bom sinal. Verifique também se o volume `mysql-persistent-stor` foi montado corretamente a partir do PVC `mysql-pvc`.

- **Verificar os logs de um contêiner**: Se um Pod estiver em um estado de erro (como `CrashLoopBackOff`), os logs são a principal ferramenta para entender o que está acontecendo dentro do contêiner.

```
# Substitua <nome-do-pod-mysql> pelo nome real do Pod do MySQL
kubectl logs <nome-do-pod-mysql>

```

O que procurar: Para o MySQL, procure por mensagens indicando que o servidor foi iniciado e está pronto para conexões. A ausência de mensagens de erro relacionadas a permissões ou configuração é fundamental.

Acessando a Aplicação

Para acessar a interface web do Adminer, que foi exposta através de um Service NodePort, o Minikube oferece um comando de atalho conveniente.

```
minikube service adminer-service
```

Este comando determina automaticamente o endereço IP do cluster Minikube e a porta alocada para o `adminer-service`, abrindo a URL correta diretamente no navegador padrão. Na página do Adminer, utilize as seguintes credenciais para se conectar ao banco de dados:

- **Sistema**: MySQL
- **Servidor**: mysql-service (o nome do Service do MySQL)
- **Usuário**: root
- **Senha**: A senha definida no `mysql-secret.yaml` (neste caso, minha-senha-super-secreta)

Evidências de Execução (Prints)

Esta seção apresenta as capturas de tela que comprovam o funcionamento correto da aplicação implantada no cluster Kubernetes, conforme solicitado nos itens obrigatórios do trabalho.

Status dos Recursos no Terminal

A imagem a seguir exibe a saída do comando `kubectl get all`, demonstrando que todos os recursos (Deployments, Pods, Services e ReplicaSets) para MySQL e Adminer estão ativos e no estado esperado (Running e Available).

Interface do Adminer

A primeira captura de tela abaixo mostra a página de login do Adminer, acessada através do Service NodePort. A segunda captura de tela confirma a conexão bem-sucedida ao banco de dados MySQL, exibindo a interface de gerenciamento com a lista de bancos de dados padrão.

(Opcional) Minikube Dashboard

Como um complemento, a imagem a seguir mostra o Minikube Dashboard, uma interface gráfica que oferece uma visão geral dos recursos do cluster. Esta visualização confirma graficamente o estado saudável dos Deployments e Pods implantados.

Dificuldades Encontradas e Soluções Aplicadas

A depuração de problemas em Kubernetes pode ser desafiadora para iniciantes, pois os erros podem se manifestar em diferentes estágios do ciclo de vida de um Pod. A chave para uma depuração eficaz é seguir um processo sistemático, começando pela análise do estado do Pod e aprofundando a investigação nos eventos e logs para identificar a causa raiz. Este processo lógico espelha a sequência de ações que o Kubernetes executa: agendamento do Pod, download

da imagem, configuração do contêiner e, finalmente, a execução da aplicação. A seguir, são detalhados os problemas mais comuns que podem ocorrer durante este trabalho prático, juntamente com as estratégias de diagnóstico e as soluções aplicadas.

Pod Preso em Pending

- **Sintoma**: Ao executar `kubectl get pods`, o status do Pod permanece em `Pending` por um longo período.
- **Causa Provável**: O estado `Pending` indica que o Pod foi aceito pelo cluster, mas o scheduler do Kubernetes não conseguiu atribuí-lo a um nó. A causa mais comum em um ambiente como o Minikube é a insuficiência de recursos (CPU ou memória) no nó para atender aos *requests* definidos no manifesto do Deployment.
- **Diagnóstico e Solução**:
  - Utilizar o comando `kubectl describe pod <nome-do-pod>` para obter detalhes.
  - Na seção Events, o scheduler geralmente registra uma mensagem clara, como `0/1 nodes are available: 1 Insufficient cpu`.
  - A solução é revisar a seção `resources.requests` no arquivo YAML do Deployment correspondente e ajustar os valores de cpu e memory para que sejam compatíveis com os recursos disponíveis no nó do Minikube.
  - Após corrigir o manifesto, reaplicá-lo com `kubectl apply -f <arquivo.yaml>`.

Erro ImagePullBackOff

- **Sintoma**: O status do Pod alterna entre `ErrImagePull` e `ImagePullBackOff`.
- **Causas Prováveis**: Este erro indica que o Kubelet no nó não conseguiu baixar a imagem do contêiner especificada. As causas incluem:
  - Erro de digitação no nome da imagem ou na tag (por exemplo, `mysql:lates` em vez de `mysql:latest`).
  - A imagem ou tag especificada não existe no registro de contêineres (Docker Hub, por padrão).
  - Problemas de conectividade de rede entre o nó do Minikube e o registro de contêineres.
- **Diagnóstico e Solução**:
  - Executar `kubectl describe pod <nome-do-pod>`. A seção Events mostrará uma mensagem de erro detalhada, como `Failed to pull image...: rpc error: code = Unknown desc = Error response from daemon: manifest for... not found`.

  - Verificar cuidadosamente o campo `image:` no arquivo YAML do Deployment para corrigir quaisquer erros de digitação.
  - Confirmar que a imagem e a tag desejadas existem no Docker Hub (ou no registro configurado).
  - Após a correção, reaplicar o manifesto. O Kubernetes tentará baixar a imagem correta novamente.

Erro CreateContainerConfigError

- **Sintoma**: O Pod falha ao iniciar e exibe o status `CreateContainerConfigError`.
- **Causa Provável**: Este erro ocorre quando o Kubernetes tem a imagem, mas falha ao configurar o contêiner antes de iniciá-lo. A causa mais comum neste projeto é uma referência a um *Secret* ou *ConfigMap* que não existe no cluster. Isso pode acontecer se o *Deployment* do MySQL for aplicado antes do `mysql-secret.yaml`.
- **Diagnóstico e Solução**:
  - O comando `kubectl describe pod <nome-do-pod>` é novamente a principal ferramenta.
  - A mensagem de erro nos Events será explícita, como `Error: secret "mysql-secret"not found`.
  - Verificar se o nome do Secret referenciado no Deployment (em `secretKeyRef.name`) corresponde exatamente ao nome do Secret definido no `mysql-secret.yaml`.
  - Garantir que o Secret foi aplicado ao cluster com `kubectl apply -f mysql-secret.yaml`.
  - Se necessário, aplicar novamente o Secret e, em seguida, excluir e recriar o Pod para que o Kubernetes tente a configuração novamente.

Erro CrashLoopBackOff

- **Sintoma**: O Pod inicia, mas falha imediatamente, entrando em um loop de reinicializações. O status exibido é `CrashLoopBackOff`.
- **Causa Provável**: Este erro indica um problema no nível da aplicação dentro do contêiner. O contêiner consegue iniciar, mas o processo principal termina com um erro, fazendo com que o Kubernetes o reinicie repetidamente. Para o MySQL, uma causa comum é a ausência ou incorreção da variável de ambiente `MYSQL_ROOT_PASSWORD`, que é obrigatória para a inicialização da imagem oficial.
- **Diagnóstico e Solução**:

  - O comando `kubectl describe pod` pode mostrar um Exit Code diferente de 0, mas a informação mais valiosa está nos logs do contêiner.
  - Utilizar `kubectl logs <nome-do-pod>` para visualizar a saída do processo da aplicação.
  - Se o Pod estiver reiniciando muito rápido, o log pode estar vazio.
  - Nesse caso, use a flag `-previous` para ver os logs da última execução antes da falha: `kubectl logs -previous <nome-do-pod>`.
  - Os logs do MySQL provavelmente indicarão um erro como `[Entrypoint]: Database is uninitialized and password option is not specified`.
  - A solução é verificar a configuração do Secret e a referência a ele no Deployment do MySQL, garantindo que a variável de ambiente `MYSQL_ROOT_PASSWORD` esteja sendo injetada corretamente.

Adminer não consegue conectar ao MySQL

- **Sintoma**: A interface do Adminer é acessível, mas ao tentar fazer login, ocorre um erro de conexão com o banco de dados (por exemplo, "Can't connect to MySQL server").
- **Causas Prováveis**:
  - O Service do MySQL (`mysql-service`) não está selecionando corretamente os Pods do MySQL devido a uma incompatibilidade entre os labels do Pod e o selector do Service.
  - O nome do servidor inserido no Adminer ou definido na variável de ambiente `ADMINER_DEFAULT_SERVER` está incorreto.
- **Diagnóstico e Solução**:
  - Verificar se os endpoints do serviço MySQL estão corretos. O comando `kubectl describe svc mysql-service` mostrará uma seção Endpoints com os endereços IP dos Pods que ele está direcionando.
  - Se esta seção estiver vazia, há um problema de correspondência entre o seletor e os labels.
  - Comparar o campo `spec.selector` no `mysql-service.yaml` com o campo `metadata.labels` na seção `template` do `mysql-deployment.yaml`.
  - Ambos devem ter os mesmos pares chave-valor (neste caso, `app: mysql`).
  - Confirmar que o valor da variável de ambiente `ADMINER_DEFAULT_SERVER` no `adminer-deployment.yaml` é exatamente `mysql-service`.
  - Corrigir qualquer inconsistência nos manifestos e reaplicá-los.

Referências

KUBERNETES. **Deployments**. [Online]. Disponível em: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

KUBERNETES. **Service**. [Online]. Disponível em: <https://kubernetes.io/docs/concepts/services-networking/service/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

KUBERNETES. **Persistent Volumes**. [Online]. Disponível em: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

KUBERNETES. **Secret**. [Online]. Disponível em: <https://kubernetes.io/docs/concepts/configuration/secret/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

KUBERNETES. **StatefulSets**. [Online]. Disponível em: <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

MINIKUBE. **Get Started!**. [Online]. Disponível em: <https://minikube.sigs.k8s.io/docs/start/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

KUBERNETES. **Configure Liveness, Readiness and Startup Probes**. [Online]. Disponível em: <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.

KUBERNETES. **Resource Management for Pods and Containers**. [Online]. Disponível em: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>. Acesso em: 23 jul. 2025. Nenhuma citação no texto.