SATC
EDUCAÇÃO E TECNOLOGIA

Engenharia da Computação

DevOps

Gledson Scotti

Kubernetes Avançado

SATC EDUCAÇÃO E TECNOLOGIA | Engenharia da Computação

Engenharia da Computação

Introdução ao Kubernetes

Objetivo da Aula:

- Entender e aplicar conceitos avançados de Kubernetes, incluindo controle de acesso, escalonamento automático e recursos personalizados.
- Aprender a criar, configurar e gerenciar charts Helm para deployments eficientes.
- Implementar boas práticas para gerenciar aplicações complexas em clusters Kubernetes.

Engenharia da Computação

Kubernetes Pré-requisitos

- Um cluster Kubernetes funcional (Minikube, Kind ou um cluster em nuvem como GKE/EKS/AKS).
- kubectl instalado e configurado para interagir com o cluster.
- Helm v3 instalado.
- Conhecimento básico de Kubernetes (Pods, Services, Deployments).
- Editor de texto (VS Code, Vim, etc.).
- Máquina com pelo menos 4GB de RAM e 2 CPUs para rodar Minikube.

Engenharia da Computação

Kubernetes RBAC

O que é RBAC?

RBAC (Role-Based Access Control) é um mecanismo de segurança do Kubernetes que controla quem pode acessar quais recursos e realizar quais ações em um cluster. Ele é essencial em ambientes multi-tenant para garantir que usuários ou aplicações tenham apenas as permissões necessárias.

Engenharia da Computação

Kubernetes RBAC - Exercício

Objetivo: Criar uma política de acesso que permita a um usuário (dev-user) apenas visualizar Pods no namespace dev, sem permissão para criar ou modificar recursos.

Exercício: Crie um namespace chamado dev para isolar os recursos. Defina um Role que permita operações de leitura (get, list, watch) em Pods. Associe o Role ao usuário dev-user usando um RoleBinding.

Engenharia da Computação

Kubernetes RBAC - Exercício

```
# rbac.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev  # Namespace onde o Role será aplicado
  name: pod-viewer  # Nome do Role, obrigatório e não pode estar vazio
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: dev  # Namespace correspondente
  name: pod-viewer-binding  # Nome do RoleBinding
subjects:
- kind: User
  name: dev-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-viewer  # Deve corresponder ao nome do Role acima
  apiGroup: rbac.authorization.k8s.io
```

Kubernetes RBAC - Exercício

Comandos bash:

```
# Aplica o arquivo de configuração
kubectl apply -f rbac.yaml

# Verifica se o usuário 'dev-user' pode listar Pods
kubectl auth can-i get pods --as=dev-user -n dev  # Retorna "yes"

# Verifica se o usuário 'dev-user' pode criar Pods
kubectl auth can-i create pods --as=dev-user -n dev  # Retorna "no"
```

Engenharia da Computação

Kubernetes RBAC

Por que usar RBAC? Evita acesso não autorizado, especialmente em clusters compartilhados.

Como testar outras permissões? Use kubectl auth can-i para verificar permissões em outros recursos, como Deployments ou ConfigMaps.

ClusterRole vs. Role: Um Role é limitado a um namespace, enquanto um ClusterRole aplica-se a todo o cluster.

Engenharia da Computação

Kubernetes HPA

O que é HPA?

O Horizontal Pod Autoscaler ajusta automaticamente o número de réplicas de um Deployment com base em métricas, como uso de CPU ou memória. Ele depende do Metrics Server para coletar dados de utilização de recursos.

Engenharia da Computação

Kubernetes HPA - Exercício

Objetivo: Configurar um HPA para escalar uma aplicação simples entre 1 e 5 réplicas, com base em 70% de uso médio de CPU.

Exercício: Deploy uma aplicação Nginx que consome CPU. Configure um HPA para monitorar o uso de CPU e escalar as réplicas.

Engenharia da Computação

Kubernetes HPA - Exercício

```
# app-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: cpu-consumer    # Nome do Deployment, obrigatório
  namespace: dev      # Namespace da aula (use 'lab' para a atividade)
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cpu-consumer
  template:
    metadata:
      labels:
        app: cpu-consumer
    spec:
      containers:
      - name: cpu-consumer
        image: nginx
        resources:
          requests:
            cpu: "100m"
          limits:
            cpu: "500m"
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-consumer-hpa
  namespace: dev  # Namespace da aula (use 'lab' para a atividade)
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cpu-consumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

Kubernetes HPA - Exercício

Comandos bash:

```
# Aplica o Deployment e o HPA
kubectl apply -f app-deployment.yaml

# Verifica o status do HPA
kubectl get hpa -n dev

# Verificar IP do pod
kubectl get pods -n dev -l app=cpu-consumer -o wide

# Simula carga para aumentar o uso de CPU (em outro terminal)
kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -n dev -- /bin/sh -c "while true; do
wget -q -O- http://<IP_interno_POD>; done"
```

Engenharia da Computação

Kubernetes HPA

Como o HPA calcula a métrica de CPU? O Metrics Server coleta dados de uso de CPU de cada Pod, e o HPA calcula a média para decidir se deve escalar.

Outras métricas? Além de CPU, o HPA suporta memória e métricas customizadas (ex.: requisições por segundo via Prometheus).

Limitações: O HPA depende de limites de recursos bem configurados nos Pods.

Engenharia da Computação

Kubernetes CRDs

O que são CRDs?

CRDs (Custom Resource Definitions) permitem estender o Kubernetes com recursos personalizados, como objetos que representam aplicações ou workflows específicos. Eles são úteis para criar APIs customizadas no cluster.

Engenharia da Computação

Kubernetes CRDs - Exercício

Objetivo: Criar uma CRD chamada CronTab para gerenciar tarefas agendadas e instanciar um recurso personalizado.

Exercício: Defina uma CRD para o recurso CronTab. Crie uma instância do recurso CronTab no namespace dev.

Kubernetes CRDs - Exercício

```yaml
# crd.yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              cronSpec:
                type: string
              image:
                type: string
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
    - ct
---
apiVersion: stable.example.com/v1
kind: CronTab
metadata:
  name: my-crontab
  namespace: dev  # Use 'lab' se estiver seguindo a atividade
spec:
  cronSpec: "* * * * */5"
  image: my-cron-image
```

Kubernetes CRDs - Exercício

Comandos bash:

```
# Aplica a CRD e a instância
kubectl apply -f crd.yaml

# Lista os recursos CronTab criados
kubectl get crontabs -n dev
```

Engenharia da Computação

Kubernetes CRDs

CRDs vs. Recursos Nativos: CRDs são extensões definidas pelo usuário, enquanto recursos nativos (como Pods) são parte do núcleo do Kubernetes.

Controladores: Para automatizar ações em CRDs (ex.: criar Pods com base em um CronTab), é necessário um controlador personalizado, geralmente implementado com o Operator SDK.

Casos de uso: CRDs são comuns em ferramentas como Istio, Prometheus Operator e ArgoCD.

Engenharia da Computação

Kubernetes Helm

O que é Helm?

Helm é um gerenciador de pacotes para Kubernetes que simplifica o deploy, a configuração e o gerenciamento de aplicações. Um "chart" Helm é um pacote que contém templates de recursos Kubernetes e valores configuráveis.

Engenharia da Computação

Kubernetes Helm - Exercício

Objetivo: Criar um chart Helm chamado my-webapp para deploy de uma aplicação web com Nginx.

Exercício: Crie um chart Helm com a estrutura básica. Configure valores padrão para réplicas, imagem e tipo de serviço.

Kubernetes Helm - Exercício

Comandos bash:

```
# instalar o helm
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
helm version

# Cria um chart Helm chamado 'my-webapp'
helm create my-webapp

cd my-webapp
```

Kubernetes Helm - Exercício

Edite values.yaml para definir configurações padrão:

```
# values.yaml
replicaCount: 2 # Número padrão de réplicas
image:
  repository: nginx # Imagem padrão
  tag: "1.21" # Versão da imagem
  pullPolicy: IfNotPresent # Política de pull
service:
  type: ClusterIP # Tipo de serviço
  port: 80 # Porta exposta
serviceAccount:
  create: false
  name: ""
ingress:
  enabled: false
autoscaling:
  enabled: false

# templates/service.yaml (necessário)
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-webapp
  namespace: {{ .Release.Namespace }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: 80
      protocol: TCP
  selector:
    app: {{ .Release.Name }}-webapp
```

Edite templates/deployment.yaml para usar variáveis do values.yaml:

```
# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-webapp # Nome dinâmico baseado no release
spec:
  replicas: {{ .Values.replicaCount }} # Usa o valor de replicaCount
  selector:
    matchLabels:
      app: {{ .Release.Name }}-webapp
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}-webapp
    spec:
      containers:
      - name: webapp
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}" # Usa valores da imagem
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        ports:
        - containerPort: 80
```

Kubernetes Helm - Exercício

Comandos bash:

```
# Instala o chart no namespace 'dev'
helm install my-webapp-release ~/my-webapp --namespace dev

# Lista os releases Helm
helm list -n dev

# Verifica os Pods criados
kubectl get pods -n dev
```

Engenharia da Computação

Kubernetes Helm

Templates Helm: Os arquivos em templates/ são renderizados com base nos valores de values.yaml, permitindo personalização.

helm install vs. helm upgrade: install cria um novo release, enquanto upgrade atualiza um release existente.

Boas práticas: Use nomes dinâmicos (ex.: {{ .Release.Name }}) para evitar conflitos.

Engenharia da Computação

Kubernetes Helm - Repositórios

O que são repositórios Helm?

Repositórios Helm são coleções de charts pré-configurados, como o repositório Bitnami, que oferece aplicações populares (ex.: WordPress, MySQL).

Objetivo: Instalar o WordPress usando o repositório Bitnami no namespace dev.

Exercício: Adicione o repositório Bitnami. Instale o WordPress com credenciais personalizadas.

Kubernetes Helm - Repositórios

Comandos bash:

```
# Adiciona o repositório Bitnami
helm repo add bitnami https://charts.bitnami.com/bitnami

# Atualiza a lista de charts disponíveis
helm repo update

# Instala o WordPress com configurações personalizadas
helm install wordpress bitnami/wordpress --namespace dev --set wordpressUsername=admin,wordpressPassword=secretpassword
```

Engenharia da Computação

Kubernetes Helm - Repositórios

Personalização: Use --set ou um arquivo values.yaml customizado para ajustar configurações (ex.: número de réplicas, recursos).

Upgrades e Rollbacks: Use helm upgrade para atualizar e helm rollback para reverter mudanças.

Repositórios confiáveis: Sempre valide a origem dos charts para evitar vulnerabilidades.

Engenharia da Computação

Kubernetes Atividades

ATIVIDADES NO AVA.