使用 Minikube

部署网站集群的完整指南

Minikube 是一个本地 Kubernetes 开发环境，非常适合在本地开发和测试网站集群。下面我将详细介绍如何使用 Minikube 部署一个完整的网站集群。

1. 准备工作

安装 Minikube 和 kubectl

```
# 安装 kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# 安装 Minikube
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

启动 Minikube

```
minikube start --driver=docker --cpus=4 --memory=8g --disk-size=20g
```

2. 部署网站集群架构

一个典型的网站集群包含以下组件：

- 前端 Web 服务
- 后端 API 服务
- 数据库服务

- 缓存服务
- 入口控制器

3. 部署示例网站集群

3.1 创建命名空间

```
kubectl create namespace website-cluster
```

3.2 部署数据库 (MySQL)

```
# mysql-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: website-cluster
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "password"
        ports:
        - containerPort: 3306
        volumeMounts:
```

```
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
# mysql-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: website-cluster
spec:
  ports:
  - port: 3306
  selector:
    app: mysql
---
# mysql-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  namespace: website-cluster
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

□□□□

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

3.3 □□□ (Redis)

```
# redis-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  namespace: website-cluster
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: redis:alpine
        ports:
        - containerPort: 6379
        resources:
          requests:
            memory: "100Mi"
            cpu: "100m"
          limits:
            memory: "200Mi"
            cpu: "200m"
---
# redis-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
  namespace: website-cluster
spec:
  ports:
  - port: 6379
  selector:
    app: redis
```

□□□□□

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

3.4 □□□□ API □□

```
# backend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: website-cluster
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: your-backend-image:latest
        ports:
        - containerPort: 8080
        env:
        - name: DB_HOST
          value: "mysql"
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
```

```
            key: password
      - name: REDIS_HOST
        value: "redis"
      resources:
        requests:
          memory: "256Mi"
          cpu: "250m"
        limits:
          memory: "512Mi"
          cpu: "500m"
      livenessProbe:
        httpGet:
          path: /health
          port: 8080
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /ready
          port: 8080
        initialDelaySeconds: 5
        periodSeconds: 5
---
# backend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
  namespace: website-cluster
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: backend
```

□□□□□

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

3.5 □□□ Web □□

```
# frontend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: website-cluster
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: your-frontend-image:latest
        ports:
        - containerPort: 80
        env:
        - name: API_URL
          value: "http://backend"
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "200m"
        livenessProbe:
```

```
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 30
        periodSeconds: 10
---
# frontend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: website-cluster
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30080
  selector:
    app: frontend
```

应用配置:

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

3.6 部署 Ingress 控制器

```
# 启用 ingress 插件
minikube addons enable ingress


# 创建 ingress 资源
# frontend-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-ingress
  namespace: website-cluster
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
```

```
spec:
  rules:
  - host: yourwebsite.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
```

应用配置:

```
kubectl apply -f frontend-ingress.yaml
```

4. 访问网站集群

修改本地 hosts 文件

```
echo "$(minikube ip) yourwebsite.local" | sudo tee -a /etc/hosts
```

访问网站

打开浏览器访问 : http://yourwebsite.local

5. 集群监控和管理

启用 Dashboard

```
minikube dashboard
```

监控集群状态

```
# 查看所有资源状态
kubectl get all -n website-cluster

# 查看 Pod 详情
kubectl describe pods -n website-cluster

# 查看日志
kubectl logs -f <pod-name> -n website-cluster
```

6. 扩展集群

水平扩展后端服务

```
kubectl scale deployment backend --replicas=5 -n website-cluster
```

自动扩缩容 (HPA)

```
# 安装 metrics-server
minikube addons enable metrics-server

# 创建 HPA
kubectl autoscale deployment backend -n website-cluster --cpu-percent=50 --min=2 --max=10
```

7. CI/CD 集成 (可选)

可以使用 Tekton 或 ArgoCD 在 Minikube 中设置 CI/CD 流水线:

```
# 安装 Tekton
kubectl apply --filename https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml

# 安装 ArgoCD
```

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

总结

通过以上步骤，你已经在Minikube中成功部署了一个完整的网站集群，包括：

- 前端Web服务
- 后端API服务
- MySQL数据库
- Redis缓存
- Ingress控制器
- 监控和扩展能力

这个集群架构可以满足大多数中小型网站的需求，并且可以在本地进行开发和测试。当需要迁移到生产环境时，可以轻松地将相同的配置应用到云Kubernetes服务上。

Revision #1
Created 4 June 2025 07:24:39 by Admin
Updated 4 June 2025 07:24:55 by Admin