# BÀI TẬP KUBERNESTES

## Lưu ý:

- Sinh viên nên tạo 1 thư mục riêng cho mỗi bài tập và lưu file resource trên thư mục này để tiện cho việc quản lý.

Sau mỗi phần thực hành các HV có thể xóa resouces đã tạo bằng câu lệnh:

-

```
kubectl delete -f <file-resource.yaml>
```

Ví dụ: kubectl delete -f nginx-pod.yaml

## I. Thực hành Pod:

### Bài Tập 1: Tạo một Pod đơn giản

**Mục tiêu:**

Tạo một Pod chạy container **nginx** đơn giản.

**Hướng dẫn:**

1. **Tạo file YAML** cho Pod:

   Tạo một file có tên nginx-pod.yaml với nội dung sau:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

2. **Apply cấu hình Pod**:

   Sử dụng lệnh kubectl apply để tạo Pod:

```
kubectl apply -f nginx-pod.yaml
```

3. **Kiểm tra trạng thái của Pod**:

Sau khi apply cấu hình, bạn có thể kiểm tra trạng thái của Pod bằng lệnh:

```
kubectl get pods
```

4. **Kiểm tra log của Pod**:

Kiểm tra logs để xác nhận Pod đang chạy đúng:

```
kubectl logs nginx-pod
```

**Bài Tập 2: Tạo một Pod với nhiều container**

**Mục tiêu:**

Tạo một Pod chứa nhiều container, một container chạy NGINX và một container chạy **busybox**.

**Hướng dẫn:**

1. **Tạo file YAML** cho Pod:

Tạo file multi-container-pod.yaml với nội dung sau:

```
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
  - name: busybox
```

```
    image: busybox
    command: [ "sleep", "3600" ]
```

2. Apply **cấu hình Pod**:

Apply YAML để tạo Pod:

```
kubectl apply -f multi-container-pod.yaml
```

3. **Kiểm tra trạng thái của Pod**:

   Kiểm tra tất cả các Pod đang chạy:

```
kubectl get pods
```

4. **Kiểm tra log của container NGINX**:

   Xem log từ container NGINX:

```
kubectl logs -container-pod -c nginx
```

5. **Kiểm tra log của container busybox**:

   Xem log từ container busybox:

```
kubectl logs multi-container-pod -c busybox
```

**Bài Tập 3: Tạo Pod với Liveness và Readiness Probe**

**Mục tiêu:**

Cấu hình **Liveness** và **Readiness Probes** để kiểm tra trạng thái hoạt động của container.

**Hướng dẫn:**

1. **Tạo file YAML** cho Pod:

   Tạo file echo-pod-with-probes.yaml với nội dung sau:

```
apiVersion: v1
kind: Pod
metadata:
  name: echo-pod-with-probes
```

```
spec:
  containers:
  - name: echo-server
    image: k8s.gcr.io/echoserver:1.4
    ports:
    - containerPort: 8080
    livenessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 5
    readinessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 5
```

2. **Apply cấu hình Pod**:

   Apply cấu hình Pod với Probes:

```
kubectl describe pod echo-pod-with-probes
```

3. **Kiểm tra trạng thái của Pod**:

   Kiểm tra trạng thái Pod và xem các probes:

```
kubectl describe pod echo-pod-with-probes
```

**Bài Tập 4: Tạo Pod với Environment Variables và ConfigMap**

**Mục tiêu:**

Sử dụng **Environment Variables** và **ConfigMap** để cấu hình cho Pod.

**Hướng dẫn:**

1. **Tạo ConfigMap**:

   Tạo file nginx-configmap.yaml:

```
apiVersion: vl
kind: ConfigMap
metadata:
  name: nginx-config
data:
  nginx.conf: |
    events {
    }
    http {
      server {
        listen       80;
        server_name  ${NGINX_HOST};  # Use environment vari

        location / {
          root   /usr/share/nginx/html;
          index  index.html index.htm;
        }
      }
    }
```

2. **Tạo Pod sử dụng ConfigMap**: Tạo file nginx-pod-with-configmap.yaml:

```
apiVersion: vl
kind: Pod
metadata:
  name: nginx-pod-with-configmap
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    env:
    - name: NGINX_HOST
      value:domain.local"  # Pass environment variable v
    volumeMounts:
```

```
    - name: config-volume
      mountPath: /etc/nginx/nginx.template.conf  # Mount th
      subPath: nginx.conf
    command: ["/bin/sh", "-c"]
    args:
    - |
      envsubst '${NGINX_HOST}' < /etc/nginx/nginx.template.
      nginx -g 'daemon off;';
  volumes:
  - name: config-volume
    configMap:
      name: nginx-config
```

3. **Apply cấu hình**:

Apply ConfigMap và Pod:

```
kubectl apply -f nginx-configmap.yaml
kubectl apply -f nginx-pod-with-configmap.yaml
```

4. **Kiểm tra Pod và ConfigMap**:

Kiểm tra trạng thái của Pod và ConfigMap:

```
kubectl get pod nginx-pod-with-configmap
kubectl get configmap nginx-config
```

**Bài tập 5:** Tạo resource các bài tập trên trên namespace dev và sửa lại tên pod thêm “-p1” ở phía sau:

ví dụ tên pod: nginx-pod sửa thành nginx-pod-p1

**Hướng dẫn:**

- Chạy lệnh tạo namespace:
- Thêm namepace vào cấu hình metadata.

Ví dụ:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: nginx-pod-with-configmap
  namespace: dev #namespace
.............
.............
```

**II. Thực hành Deployment, Replicaset, HPA:**

**Lưu ý:** Trước khi thực hành deployment, sinh viên thực hiện xóa tất cả các resources pod và configmap đã tạo trong phần

**I. Thực hành Pod**

**Bài Tập 1: Tạo một Deployment đơn giản Mục tiêu:**

Tạo một Deployment đơn giản để chạy container **nginx**.

**Hướng dẫn:**

1. **Tạo file YAML** cho Deployment:

Tạo một file có tên nginx-deployment.yaml với nội dung sau:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
```

```
        ports:
        - containerPort: 80
```

2. **Apply cấu hình Deployment**:

Apply file YAML để tạo Deployment:

```
kubectl apply -f nginx-deployment.yaml
```

3. **Kiểm tra trạng thái của Deployment**:

Kiểm tra trạng thái của Deployment và Pods:

```
kubectl get deployments
kubectl get pods
```

4. **Kiểm tra log của Pods**:

Kiểm tra log từ một trong các Pod:

```
kubectl logs <nginx-pod-name>
```

**Bài Tập 2: Cập nhật Deployment Mục tiêu:**

Cập nhật Deployment để sử dụng một phiên bản nginx khác và quan sát quá trình rolling update.

**Hướng dẫn:**

1. **Cập nhật file YAML**:

Chỉnh sửa file nginx-deployment.yaml để sử dụng phiên bản nginx khác (ví dụ: nginx:1.21):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

2. **Apply cấu hình mới**:

   Apply lại file YAML để cập nhật Deployment:

3. **Kiểm tra quá trình rolling update**:

   Xem chi tiết quá trình rolling update:

```
kubectl rollout status deployment/nginx-deployment
```

4. **Kiểm tra pods sau khi cập nhật**:

   Kiểm tra trạng thái của Pods sau khi cập nhật:

```
kubectl get pods
```

**Bài Tập 3: Tạo và cấu hình Deployment với môi trường (Environment Variables)**

**Mục tiêu:**

Tạo một Deployment với các môi trường cấu hình cho container.

**Hướng dẫn:**

1. **Tạo file YAML cho Deployment**:

   Tạo một file nginx-deployment-env.yaml với nội dung như sau:

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: nginx-deployment-env
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-env
  template:
    metadata:
      labels:
        app: nginx-env
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        env:
        - name: NGINX_HOST
          value: "localhost"
        - name: NGINX_PORT
          value: "80"
```

2. **Apply cấu hình Deployment**:

Apply file YAML để tạo Deployment:

```
kubectl apply -f nginx-deployment-env.yaml
```

3. **Kiểm tra trạng thái của Pods**:

Kiểm tra các Pods để xác nhận rằng môi trường đã được thiết lập đúng:

```
kubectl get pods
```

4. **Kiểm tra log**:

Xem log của container để kiểm tra xem môi trường đã được tải đúng chưa:

```
kubectl logs <nginx-pod-name>
```

**Bài Tập 4: Tạo Deployment với Service để expose ứng dụng**

**Mục tiêu:**

Tạo một Deployment và expose nó thông qua một Service để truy cập từ bên ngoài với loại **service type NodePort**

**Hướng dẫn:**

1. **Tạo file YAML cho Deployment**:

Tạo file nginx-deployment-service.yaml với nội dung sau:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

2. **Tạo Service để expose Deployment**:

Tạo file nginx-service.yaml với nội dung sau:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
```

```
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort
```

3. **Apply cấu hình Deployment và Service**:

Apply file YAML để tạo Deployment và Service:

4. **Kiểm tra Service và truy cập ứng dụng**:

Kiểm tra trạng thái của Service:

```
kubectl get services
```

Kết quả sẽ hiển thị Service với cổng NodePort (30007), ví dụ:

```
NAME          TYPE      CLUSTER-IP     EXTERNAL-IP   POR nginx-service   NodePort
10.96.123.45  <none>       80:
```

5. **Kiểm tra ứng dụng thông qua NodePort Tìm địa chỉ IP của**

**Node:**

Dùng lệnh sau để lấy danh sách các Node trong Kubernetes cluster:

**Truy cập ứng dụng:**

Mở trình duyệt hoặc dùng curl để truy cập vào ứng dụng qua địa chỉ **Node IP** và cổng **30007**:

```
curl http://<node-ip>:30007
```

**Bài tập 5:**

Từ bài tập 4 ở trên, Sinh viên thực hiện gán port Nodeport cố định là 30111 vào service nginx-service sau đó kiểm tra lại bằng câu lệnh curl

**Gợi ý:**

Thêm cấu hình như sau vào file nginx-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80          # Cổng mà Service expose bên trong cluster
      targetPort: 80    # Cổng container (của ứng dụng nginx)
      nodePort: 30111   # Cổng cố định để truy cập từ bên ngoài cluster
  type: NodePort        # Loại Service là NodePort
```

**Bài Tập 6: Sử dụng stefanprodan/podinfo:6.3.5 để kiểm tra Deployment trong Kubernetes**

**Mục tiêu:**

- Triển khai ứng dụng stefanprodan/podinfo:6.3.5 bằng **Deployment**.
- Kiểm tra cơ chế **Rolling Update** của Deployment.
- Thực hành Scale số lượng Pod trong Deployment.
- Kiểm tra hành vi của Deployment khi xảy ra lỗi và sử dụng tính năng Rollback.

**Bước 1: Tạo Deployment YAML**

1. Tạo file podinfo-deployment.yaml với nội dung sau:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: podinfo-deployment
  labels:
    app: podinfo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: podinfo
  template:
    metadata:
```

```
      labels:
        app: podinfo
    spec:
      containers:
      - name: podinfo
        image: stefanprodan/podinfo:6.3.5
        ports:
        - containerPort: 9898
        env:
        - name: PODINFO_UI_MESSAGE
          value: "Welcome to Podinfo!"
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
```

**Giải thích:**

- replicas: 3: Triển khai 3 Pod cho ứng dụng Podinfo.
- selector.matchLabels: Deployment quản lý các Pod có nhãn app: podinfo.
- image: stefanprodan/podinfo:6.3.5: Sử dụng image podinfo phiên bản 6.3.5.
- env: Đặt một biến môi trường PODINFO_UI_MESSAGE để hiển thị thông báo tùy chỉnh trên giao diện.
- resources: Cấu hình tài nguyên CPU và RAM để quản lý tài nguyên cluster hiệu quả.

**Bước 2: Apply Deployment**

1. Apply file YAML để tạo Deployment:

```
kubectl apply -f podinfo-deployment.yaml
```

2. Kiểm tra trạng thái của Deployment:

kubectl get deployment

Kết quả:

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
podinfo-deployment   3/3     3            3           10s
```

3. Kiểm tra trạng thái các Pod:

```
kubectl get pods -l app=podinfo
```

Kết quả:

```
NAME                                 READY   STATUS    RESTARTS   AGE
podinfo-deployment-xxxxxxx           1/1     Running   0          10s
podinfo-deployment-yyyyyyy           1/1     Running   0          10s
podinfo-deployment-zzzzzzz           1/1     Running   0          10s
```

**Bước 3: Kiểm Tra Rolling Update**

1. **Cập nhật image trong Deployment:**

Sửa file podinfo-deployment.yaml để cập nhật image từ 6.3.5 thành 6.3.6:

```
containers:
- name: podinfo
  image: stefanprodan/podinfo:6.3.6
```

3. **Quan sát quá trình Rolling Update:**

Dùng lệnh sau để theo dõi quá trình update:

```
kubectl rollout status deployment/podinfo-deployment
```

Kết quả:

```
deployment "podinfo-deployment" successfully rolled out
```

4. **Kiểm tra lại các Pod:**

kubectl get pods -l app=podinfo

5. **Kiểm tra log của một Pod:**

```
kubectl logs <pod-name>
```

Xác nhận rằng image mới (6.3.6) đang hoạt động.

**Bước 4: Thực Hành Scale Deployment**

1. **Thay đổi số lượng replicas:**

Cập nhật file podinfo-deployment.yaml để thay đổi số lượng replicas từ 3 thành 5:

```
replicas: 5
```

2. **Apply cấu hình mới:**

```
kubectl apply -f podinfo-deployment.yaml
```

3. **Kiểm tra trạng thái Deployment:**

```
kubectl get deployment
```

Kết quả:

```
NAME READY UP-TO-DATE AVAILABLE AGE podinfo-deployment 5/5 5 5 1m
```

4. **Kiểm tra số lượng Pod:**

```
kubectl get pods -l app=podinfo
```

Kết quả sẽ hiển thị 5 Pod đang chạy.

**Bước 5: Thực Hành Rollback**

1. **Giả lập lỗi:**

Cập nhật file podinfo-deployment.yaml để sử dụng image không tồn tại:

image: stefanprodan/podinfo:invalid-version

Apply thay đổi:

```
kubectl apply -f podinfo-deployment.yaml
```

2. **Kiểm tra trạng thái Deployment:**

```
kubectl rollout status deployment/podinfo-deployment
```

Bạn sẽ thấy rằng quá trình rollout bị lỗi.

3. **Rollback về phiên bản trước đó:**

```
kubectl rollout undo deployment/podinfo-deployment
```

4. **Xác nhận rollback thành công:**

```
kubectl get pods -l app=podinfo
```

Kiểm tra lại log của Pod để đảm bảo phiên bản trước (6.3.6) đã được phục hồi.

**Bước 6: Expose Deployment qua Service**

1. Tạo một Service kiểu NodePort để expose Deployment ra bên ngoài:

Tạo file podinfo-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: podinfo-service
spec:
  selector:
    app: podinfo
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9898
```

```
    nodePort: 30008
  type: NodePort
```

2. Apply Service:

```
kubectl apply -f podinfo-service.yaml
```

3. Kiểm tra trạng thái Service:

```
kubectl get service podinfo-service
```

Kết quả:

```
NAME              TYPE      CLUSTER-IP      EXTERNAL-IP   podinfo-service   NodePort
10.96.123.45   <none>
```

4. Truy cập ứng dụng:

Dùng trình duyệt hoặc curl để truy cập ứng dụng qua địa chỉ:

```
http://<node-ip>:30008
```

**Bước 7: Kiểm Tra Tính Năng Của Ứng Dụng**

Podinfo cung cấp nhiều endpoint để kiểm tra thông tin và trạng thái của ứng dụng:

1. **Truy cập thông tin Pod:**

```
curl http://<node-ip>:30008
```

2. **Kiểm tra metadata của ứng dụng:**

```
curl http://<node-ip>:30008/metrics
```

3. **Kiểm tra healthcheck:**

```
curl http://<node-ip>:30008/healthz
```

**Kết quả mong đợi:**

- HV đã triển khai thành công ứng dụng stefanprodan/podinfo:6.3.5 với Deployment.
- HV đã thực hành Rolling Update và Rollback để kiểm tra tính năng cập nhật và khôi phục của Deployment.

- HV đã scale số lượng Pod và kiểm tra trạng thái.
- HV đã expose ứng dụng qua NodePort và truy cập từ bên ngoài.

**Bài tập 7: Tích Hợp Horizontal Pod Autoscaler (HPA) vào Bài Tập Deployment Mục tiêu:**

- Mở rộng bài tập 7 đã triển khai ứng dụng stefanprodan/podinfo:6.3.5 để sử dụng **Horizontal Pod Autoscaler (HPA)**.
- Tự động scale số lượng Pod dựa trên tài nguyên CPU và kiểm tra HPA hoạt động khi tải tăng/giảm.

**Hướng dẫn:**

**Phần 1: Tạo HPA dựa trên CPU**

**Bước 1: Tạo File YAML cho HPA**

Tạo file podinfo-hpa.yaml với nội dung sau:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: podinfo-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo-deployment
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
```

```
        type: Utilization
        averageUtilization: 50
```

**Giải thích:**

- scaleTargetRef: Chỉ định Deployment mà HPA sẽ theo dõi và quản lý (podinfo-deployment).

- minReplicas: Tối thiểu là 2 Pod.
- maxReplicas: Tối đa là 5 Pod.
- metrics: HPA sử dụng chỉ số CPU (averageUtilization) làm cơ sở để scale. Nếu mức sử dụng CPU trung bình vượt quá 50%, HPA sẽ scale Pod.

**Bước 2: Deploy HPA** Deploy file HPA:

```
kubectl apply -f podinfo-hpa.yaml
```

Kiểm tra trạng thái của HPA:

```
kubectl get hpa
```

Kết quả:

```
NAME        REFERENCE                  TARGETS   MINPODS   MAX podinfo-hpa
Deployment/podinfo-deployment   10%/50%      2
```

**Phần 2: Kiểm Tra HPA**

**Bước 1: Tăng Tải Lên Ứng Dụng**

Trên ubuntun-01, thực hiện cài đặt gói wrk

```
apt install wrk
```

Sau đó chạy lệnh bên dưới để giả lập gọi nhiều request

```
wrk -t10 -c2000 -d300s http://<IP-node>:30008
```

**Bước 2: Theo Dõi HPA**

Theo dõi trạng thái HPA và xem khi nào số lượng Pod được scale:

```
kubectl get hpa -w
```

Bạn sẽ thấy số lượng Pod tăng dần khi tải tăng.

Kiểm tra số lượng Pod:

```
kubectl get pods -l app=podinfo
```

**Phần 3: Dừng tạo request và Quan Sát Pod Scale Xuống**

1. **Ngừng câu lệnh wrk:**

   Bấm ctrl + c để dừng câu lệnh wrk ngay tức thì

2. **Theo dõi HPA:**

   Khi tải giảm, HPA sẽ tự động giảm số lượng Pod xuống mức minReplicas:

```
kubectl get hpa -w
```

**Bài tập 8:** Dựa vào các kiến thức trên đã học, Sinh viên hãy tự build docker image nginx với file config custom riêng(chẳng hạn custom index.html như các bài thực hành trước), upload lên docker hub sau đó triển khai resource Deployment, HPA, Service với Docker image này.

# III. Thực hành PV và PVC:

Lưu ý: xóa hết các resources đã tạo ở các bài tập trước

Trong bài tập này sẽ thực hành với storageclass là **local-storage** được tạo trên k8s **node1**

**Bài Tập 1: Tạo PV và PVC cơ bản với Local-Storage Mục tiêu:**

- Tạo PersistentVolume (PV) sử dụng local-storage trên **node1**.
- Gắn PersistentVolumeClaim (PVC) vào một Pod.

**Hướng dẫn:**

1. **Tạo PV:**

   Tạo file local-pv.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /mnt/data
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node1
```

2. **Tạo PVC:**

Tạo file local-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: local-storage
```

3. **Tạo Pod sử dụng PVC:**

Tạo file local-pod.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: local-storage-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: local-storage
  volumes:
  - name: local-storage
    persistentVolumeClaim:
      claimName: local-pvc
```

4. Apply **cấu hình:**

```
kubectl apply -f local-pv.yaml
kubectl apply -f local-pvc.yaml
kubectl apply -f local-pod.yaml
```

5. **Kiểm tra trạng thái:**

```
kubectl get pv
kubectl get pvc
kubectl get pod
```

**Bài tập 2: Tạo Deployment MySQL với StorageClass local-storage và PVC Đã Tạo Sẵn**

**Mục tiêu:**

- Tạo **PersistentVolume (PV)** và **PersistentVolumeClaim (PVC)** trước.
- Triển khai Deployment MySQL sử dụng PVC đã được tạo sẵn và StorageClass local-storage.
- Kiểm tra lưu trữ dữ liệu MySQL trên local-storage của **node1**.

**Bước 1: Chuẩn Bị Local Storage Tạo thư mục trên node1**:

SSH vào **node1** và tạo thư mục để sử dụng làm local-storage:

```
mkdir -p /mnt/mysql-data
chmod 777 /mnt/mysql-data
```

**Bước 2: Tạo PersistentVolume (PV)**

1. **Tạo file** local-pv.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  claimRef:
    name: mysql-pvc
    namespace: default  # Đảm bảo đúng namespace của PVC
  local:
    path: /mnt/mysql-data
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node1  # Đảm bảo đúng tên node trong cluster
```

2. **Apply cấu hình PV**:

```
kubectl apply -f local-pv.yaml
```

3. **Kiểm tra PV**:

```
kubectl get pv
```

**Bước 3: Tạo PersistentVolumeClaim (PVC)**

1. **Tạo file** local-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
  namespace: default  # Đảm bảo đúng namespace
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: local-storage  # Khớp với storageClassN
```

2. Apply **cấu hình PVC**:

```
kubectl get pvc
```

3. **Kiểm tra PVC**:

```
kubectl get pvc
```

Kết quả:

```
NAME STATUS VOLUME CAPACITY ACCESS MODES S mysql-pvc Bound mysql-pv 5Gi RWO l
```

**Bước 4: Tạo Deployment MySQL Sử Dụng PVC**

1. **Tạo file** mysql-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: rootpassword
        volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-data
        persistentVolumeClaim:
          claimName: mysql-pvc  # Sử dụng PVC đã tạo trước
```

**Giải thích:**

- persistentVolumeClaim.claimName: mysql-pvc: Gắn PVC mysql-pvc vào Deployment.

2. **Apply Deployment**:

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

3. **Kiểm tra Deployment**:

```
kubectl get deployment
```

Kết quả:

```
NAME    READY   AGE
mysql   1/1     30s
```

4. **Kiểm tra Pod**:

```
kubectl get pods
```

Kết quả:

```
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   1/1     Running   0          30s
```

5. **Kiểm tra PVC và PV**:

```
kubectl get pvc
kubectl get pv
```

**Bước 5: Kết Nối MySQL và Kiểm Tra Dữ Liệu**

1. **Kết nối vào Pod MySQL**:

```
kubectl exec -it mysql-0 -- mysql -u root -p
```

Nhập mật khẩu:

```
admin123
```

2. **Tạo cơ sở dữ liệu và bảng**:

```
CREATE DATABASE testdb;
USE testdb;
CREATE TABLE users (id INT AUTO_INCREMENT PRIMARY KEY, name VA
INSERT INTO users (name) VALUES ('John Doe');
SELECT * FROM users;
```

**Bước 6: Kiểm Tra Lưu Trữ Dữ Liệu**

1. SSH vào node1 và kiểm tra thư mục /mnt/mysql-data:

```
ls /mnt/mysql-data
```

2. HV sẽ thấy các tệp dữ liệu MySQL được lưu trữ trong thư mục.

**Phần Mở Rộng: Tích Hợp Service**:

Tạo Service để expose MySQL ra bên ngoài cluster:

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  ports:
  - port: 3306
    targetPort: 3306
  selector:
    app: mysql
```

Apply Service:

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

**Thử Xóa PVC và Deployment**:

Xóa **Deployment**:

```
kubectl delete deployment mysql
```

Kiểm tra PV sau khi xóa PVC:

```
kubectl get pv
```

PV vẫn ở trạng thái Released do Reclaim Policy là Retain.

**Kết Quả Mong Đợi**

1. **PVC đã được tạo sẵn** (mysql-pvc) được gắn thành công vào Deployment MySQL.
2. Dữ liệu MySQL được lưu trữ trên local storage của **node1**.
3. Dữ liệu được giữ lại trên server dù PVC hoặc Deployment bị xóa (nhờ Reclaim Policy Retain).

## IV. Thực hành Helm chart:

Lưu ý: HV xóa hết các resources đã tạo ở các bài tập trước:

**Bài Tập 1: Tạo Ingress NGINX với Helm và Cấu Hình NodePort cho HTTP và HTTPS**

**Mục tiêu:**

- Sử dụng Helm để triển khai **Ingress NGINX Controller**.
- Mở cổng **NodePort** cố định cho HTTP (3000) và HTTPS (3443).
- Cấu hình chính sách externalTrafficPolicy=Local để tối ưu hóa IP nguồn.

**Hướng dẫn:**

**Bước 1: Cài Đặt Helm Chart Ingress NGINX**

1. **Thêm Helm Repository của NGINX:** Thêm Helm repository chính thức:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingre
helm repo update
```

2. **Triển khai NGINX Ingress Controller:**

Chạy lệnh sau để cài đặt Ingress NGINX Controller vào name-space ingressnginx với các cổng NodePort cố định:

```
helm install nginx-ingress ingress-nginx/ingress-nginx \
  --namespace ingress-nginx \
  --create-namespace \
  --set controller.service.type=NodePort \
  --set controller.service.externalTrafficPolicy=Local \
  --set controller.service.nodePorts.http=30080 \
  --set controller.service.nodePorts.https=30443
```

**Giải thích:**

- controller.service.type=NodePort: Dịch vụ NGINX Controller được mở qua NodePort.
- controller.service.externalTrafficPolicy=Local: Đảm bảo IP nguồn của client không bị thay đổi (giữ nguyên địa chỉ IP ban đầu).
- controller.service.nodePorts.http=3080: Cấu hình NodePort cố định cho HTTP (3080).

- controller.service.nodePorts.https=3443: Cấu hình NodePort cố định cho HTTPS (3443).
- --create-namespace: Tự động tạo namespace ingress-nginx nếu namespace không tồn tại.

3. **Kiểm tra trạng thái Helm Release:**

Kiểm tra danh sách release Helm:

```
helm list -n ingress-nginx
```

4. **Kiểm tra trạng thái của Pod và Service:**

Kiểm tra tài nguyên được tạo trong namespace ingress-nginx:

```
kubectl get all -n ingress-nginx
```

Kết quả Service:

```
NAME                                                 TYP
service/nginx-ingress-ingress-nginx-controller       Nod
```

**Bước 2: Triển Khai Ứng Dụng và Tạo Ingress Resource**

1. **Tạo một Deployment và Service đơn giản:**

Tạo file http-echo-app.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: http-echo-app
  labels:
    app: http-echo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: http-echo
  template:
    metadata:
      labels:
        app: http-echo
    spec:
      containers:
      - name: http-echo
        image: hashicorp/http-echo:latest
        args:
        - "-text=Hello from Ingress NGINX"
        ports:
        - containerPort: 5678
---
apiVersion: v1
kind: Service
metadata:
  name: http-echo-service
spec:
  selector:
    app: http-echo
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5678
```

2. **Apply cấu hình Deployment và Service:**

```
kubectl apply -f http-echo-app.yaml
```

3. **Tạo Ingress Resource:**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: http-echo-ingress
  annotations:
spec:
  ingressClassName: nginx
  rules:
  - host: http-echo.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: http-echo-service
            port:
              number: 80
```

4. **Apply cấu hình Ingress Resource:**

```
kubectl apply -f http-echo-ingress.yaml
```

5. **Kiểm tra trạng thái Ingress:**

```
kubectl get ingress
```

**Bước 3: Kiểm Tra Hoạt Động**

1. **Tìm địa chỉ IP của Node:**

Lấy địa chỉ IP của một Node trong cluster:

```
kubectl get nodes -o wide
```

Địa chỉ IP nằm trong cột INTERNAL-IP.

2. **Cập nhật file hosts trên máy server Ubuntu-2:**

Thêm domain example.local trỏ đến địa chỉ IP Node. Mở file /etc/hosts và thêm dòng:

```
<node-ip> http-echo.local
```

3. **Kiểm tra truy cập HTTP:**

Dùng curl hoặc trình duyệt để kiểm tra HTTP qua NodePort (30080):

```
curl http://http-echo.local:30080
```

Kết quả

```
Hello from Ingress NGINX
```

**Phần Mở Rộng**

1. **Thêm nhiều host cho Ingress:**

Tạo thêm rules để hỗ trợ 2 domain với yêu cầu như sau:

Domain: http-echo.local → hỗ trợ HTTP

Domain: http-echo.local → hỗ trợ HTTP, HTTPS

Ví dụ cấu hình thêm domain http-echo-2.local vào file http-echo-ingress.yaml

```
rules:
- host: http-echo.local
  http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: echo-app-service
          port:
            number: 80
- host: http-echo-2.local
  http:
```

```
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: echo-app-service
          port:
            number: 80
  tls:
  - hosts:
    - echo-app-2.local  # Thay bằng domain của HV
    secretName: echo-app-2-local-tls
```

**Bài Tập 3: Triển Khai Prometheus trên Helm Chart với Ingress NGINX HTTPS và Volume Từ Local-Storage Mục tiêu:**

- Triển khai Prometheus trên namespace monitoring bằng Helm Chart.
- Cấu hình Ingress NGINX với HTTPS
- Sử dụng local-storage làm PersistentVolume Prometheus.

**Yêu cầu chuẩn bị:**

3. Ingress NGINX đã triển khai sẵn.

**Bước 1: Chuẩn Bị Local Storage**

1. **Tạo thư mục trên Node:**

SSH vào Node 1 trong cluster và tạo thư mục để sử dụng làm local storage:

```
mkdir -p /mnt/prometheus-data
chmod 777 /mnt/prometheus-data
```

2. **Tạo PersistentVolume (PV):**

Tạo file local-pv.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: prometheus-pv
```

```
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  claimRef:  # XÓA phần này nếu bạn để Kubernetes tự động liên
    name: prometheus-pvc
    namespace: monitoring  # Đảm bảo đúng namespace của PVC
  local:
    path: /mnt/prometheus-data
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node1  # Thay bằng tên Node thực tế của bạn
```

3. **Tạo PersistentVolumeClaim (PVC):** Tạo file local-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: prometheus-pvc
  namespace: monitoring
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: local-storage
```

5. **Apply cấu hình PV, và PVC:**

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

6. **Kiểm tra trạng thái PV và PVC:**

```
kubectl get pv
kubectl get pvc
```

Kết quả:

```
 NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STA prometheus-pv   5Gi
RWO           Retain          Bo
```

**Bước 2: Cài Đặt Prometheus Với PVC Được Tạo Trước**

1. **Thêm Helm Repository của Prometheus:**

```
helm repo add prometheus-community https://prometheus-communi
helm repo update
```

2. **Tạo file** prometheus-values.yaml**:**

Tùy chỉnh cấu hình Helm Chart để sử dụng PVC prometheus-pvc đã được tạo trước.

```
# prometheus-values.yaml

alertmanager:
  enabled: false  # Tắt AlertManager để đơn giản hóa

server:
  persistentVolume:
    enabled: true
    existingClaim: prometheus-pvc  # Sử dụng PVC đã tạo trước
    mountPath: /data  # Nơi dữ liệu Prometheus được lưu
  ingress:
    enabled: true
    ingressClassName: nginx
```

```
    annotations:
    hosts:
      - prometheus.local  # Domain của bạn
    tls:
      - secretName: prometheus-tls
        hosts:
          - prometheus.local
```

3. **Cài đặt Prometheus với Helm:**

```
helm install prometheus prometheus-community/prometheus \
  --namespace monitoring \
  --create-namespace \
  --values prometheus-values.yaml
```

4. **Kiểm tra trạng thái Helm Release:**

```
helm list -n monitoring
```

**Bước 3: Cấu Hình Ingress và HTTPS**

1. **Kiểm tra trạng thái Ingress Resource:**

```
kubectl get ingress -n monitoring
```

Kết quả:

```
NAME         CLASS   HOSTS              ADDRESS   PORT
prometheus   nginx   prometheus.local   <external>  80,
```

2. **Cập nhật file hosts trên máy local:**

Thêm domain prometheus.local trỏ đến địa chỉ IP của Node 1

3. **Kiểm tra HTTPS**

- Mở trình duyệt và truy cập https://prometheus.local.
- Hoặc dùng curl:

```
curl -k https://prometheus.local
```

Kết quả: HV sẽ thấy giao diện Prometheus hoạt động qua HTTPS.

**Bước 4: Kiểm thử**

**Xóa Pod Prometheus để kiểm tra tính bền vững của dữ liệu:**

```
kubectl delete pod -n monitoring <prometheus-server-pod>
```

**Kiểm tra dữ liệu sau khi Pod khởi động lại:**

Pod Prometheus sẽ tự động khởi động lại, và dữ liệu vẫn được lưu trong /mnt/prometheus-data.

**Bài Tập 4: Triển Khai Grafana Trên Helm Chart Mục tiêu:**

- Triển khai Grafana trên namespace monitoring bằng Helm Chart.
- Cấu hình Ingress NGINX để truy cập Grafana qua HTTPS
- Sử dụng PersistentVolumeClaim (PVC) từ local-storage để lưu dữ liệu Grafana.
- Kết nối Grafana với Prometheus (đã triển khai trong cùng namespace ở bài tập trước).

**Bước 1: Tạo PersistentVolume (PV) và PersistentVolumeClaim (PVC) cho Grafana**

1. **Tạo thư mục trên Node:**

SSH vào **node1** và tạo thư mục để sử dụng làm local storage:

```
mkdir -p /mnt/grafana-data
chmod 777 /mnt/grafana-data
```

2. **Tạo PersistentVolume (PV):** Tạo file grafana-pv.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: grafana-pv
spec:
  capacity:
```

```
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  claimRef:
    name: grafana-pvc
    namespace: monitoring  # Đảm bảo đúng namespace của PVC
  local:
    path: /mnt/grafana-data
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node1  # Thay bằng tên Node thực tế
```

3. **Tạo PersistentVolumeClaim (PVC):** Tạo file grafana-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: grafana-pvc
  namespace: monitoring
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: local-storage
```

4. **Apply cấu hình PV và PVC:**
5. **Kiểm tra trạng thái PV và PVC:**

```
kubectl get pv
kubectl get pvc -n monitoring
```

Kết quả:

```
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATU grafana-pv    5Gi
RWO            Retain           Bound
```

**Bước 2: Cài Đặt Grafana Với Helm**

1. **Thêm Helm Repository của Grafana:**

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
```

2. **Tạo file** grafana-values.yaml**:**

File này sẽ tùy chỉnh cấu hình Grafana, bao gồm sử dụng PVC đã tạo và cấu hình Ingress HTTPS.

```
persistence:
  enabled: true
  existingClaim: grafana-pvc  # Sử dụng PVC đã tạo
  mountPath: /var/lib/grafana

ingress:
  enabled: true
  ingressClassName: nginx
  annotations:
  hosts:
    - grafana.local  # Domain Grafana
  tls:
    - secretName: grafana-tls
      hosts:
        - grafana.local

datasources:
  datasources.yaml:
    apiVersion: 1
```

```
 datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus-server  # Kết nối đến Prometheus
    isDefault: true
# Set admin password (replace with a secure password)
adminPassword: "admin@123"

dashboardProviders:
  dashboardproviders.yaml:
    apiVersion: 1
    providers:
      - name: 'grafana-dashboards-kubernetes'
        orgId: 1
        folder: ''
        type: file
        disableDeletion: false
        editable: true
        options:
          path: /var/lib/grafana/dashboards/default

dashboards:
  default:
    k8s-addons-prometheus:
      url: https://raw.githubusercontent.com/dotdc/grafana-da...
      token: ''
    k8s-system-api-server:
      url: https://raw.githubusercontent.com/dotdc/grafana-da...
      token: ''
    k8s-system-coredns:
      url: https://raw.githubusercontent.com/dotdc/grafana-da...
      token: ''
    k8s-views-global:
      url: https://raw.githubusercontent.com/dotdc/grafana-da...
      token: ''
    k8s-views-namespaces:
      url: https://raw.githubusercontent.com/dotdc/grafana-da...
      token: ''
```

```
    k8s-views-nodes:
      url: https://raw.githubusercontent.com/dotdc/grafana-da
      token: ''
    k8s-views-pods:
      url: https://raw.githubusercontent.com/dotdc/grafana-da
      token: ''
```

3. **Cài đặt Grafana với Helm:**

```
helm install grafana grafana/grafana \
  --namespace monitoring \
  --values grafana-values.yaml
```

4. **Kiểm tra trạng thái Helm Release:**

```
helm list -n monitoring
```

5. **Kiểm tra trạng thái Pod và Service:**

```
kubectl get all -n monitoring
```

Kết quả:

```
NAME READY STATUS RESTAR pod/grafana-xxxxx 1/1 Running 0
```

**Bước 3: Cấu Hình Ingress và HTTPS**

1. **Kiểm tra trạng thái Ingress Resource:**

```
kubectl get ingress -n monitoring
```

Kết quả:

```
NAME CLASS HOSTS ADDRESS PORTS AG grafana nginx grafana.local <external> 80, 443 1m
```

2. **Cập nhật file hosts trên máy local:**

Thêm domain grafana.local trỏ đến địa chỉ IP của Node 1

3. **Kiểm tra HTTPS:**

- Mở trình duyệt và truy cập https://grafana.local.
- Hoặc dùng curl:

```
curl -k https://grafana.local
```

**Bước 4: Đăng Nhập và Kiểm Tra Datasource**

1. **Đăng nhập vào Grafana:**

- Mặc định, tài khoản đăng nhập Grafana:
- Username: admin
- Password: admin@123 (hoặc kiểm tra trong giá trị Helm Chart).

2. **Kiểm tra Datasource Prometheus:**

- Truy cập menu **Configuration > Data Sources**.
- Xác nhận rằng Prometheus đã được cấu hình tự động.

3. **Dashboard:**

Xem các metric trong các Dashboard được tạo trước từ helm value có hiển thị không.