Part 22: Kubernetes Real-Time Troubleshooting

Introduction 🌐

Welcome to the world of Kubernetes troubleshooting, where every challenge is an opportunity to sharpen your skills and emerge victorious. Join us as we embark on a journey through common real-time scenarios, unraveling mysteries, and uncovering solutions along the way.

PART 22 - KUBERNETES REAL-TIME TROUBLESHOOTING

- Manage Persistent Volumes and Claims
- Install and Configure Metrics Server
- Sort Pods by Age and UID
- Manage Kube-Scheduler
- Create ServiceAccount with Role and RoleBinding

Scenario 106: Managing Persistent Volumes and Claims in Kubernetes

Scenario: You need to create a PersistentVolume (PV) and PersistentVolumeClaim (PVC) to provide persistent storage to a deployment.

Solution:

1. **Create the PersistentVolume:**

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

```
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /Volumes/Data
```

2. **Create the PersistentVolumeClaim:**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: safari-pvc
  namespace: project-tiger
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

3. **Create the Deployment Using the PVC:**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: safari
  namespace: project-tiger
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: safari-container
        image: httpd:2.4.41-alpine
        volumeMounts:
        - name: safari-storage
          mountPath: /tmp/safari-data
      volumes:
      - name: safari-storage
        persistentVolumeClaim:
          claimName: safari-pvc
```

Outcome: The deployment successfully uses persistent storage, ensuring data persistence across pod restarts.

Scenario 107: Installing and Configuring Metrics Server in Kubernetes

Scenario: You need to install the Metrics Server to monitor resource usage in your Kubernetes cluster.

Solution:

1. **Install Metrics Server:**
   ```
   kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
   ```

2. **Allow Insecure TLS (If Required):**
   ```
   kubectl patch deployment metrics-server -n kube-system --type='json' -p='[{"op": "add", "path": "/spec/template/spec/containers/0/args/-", "value": "--kubelet-insecure-tls"}]'
   ```

3. **Verify Metrics Server Installation:**
   ```
   kubectl get deployment metrics-server -n kube-system
   kubectl top nodes
   ```

Outcome: The Metrics Server provides valuable insights into resource usage, aiding in cluster management and optimization.

Scenario 108: Sorting Pods by Age and UID

Scenario: You need to list pods sorted by their creation timestamp and UID for better management and monitoring.

Solution:

Script to List Pods Sorted by Age:

```
kubectl get pods --all-namespaces --sort-by=.metadata.creationTimestamp
```

Script to List Pods Sorted by UID:

```
kubectl get pods --all-namespaces --sort-by=.metadata.uid
```

Outcome: These scripts help in quickly identifying the oldest or newest pods, aiding in resource management and troubleshooting.

Scenario 109: Managing Kube-Scheduler & shcduling pod

Scenario: Use context: kubectl config use-context k8s-c2-AC. Ssh into the master node with ssh cluster2-master1. Temporarily stop the kube-scheduler, this means in a way that you can start it again afterwards. Create a single Pod named manual-schedule of image httpd:2.4-alpine, confirm its created but not scheduled on any node. Now you're the scheduler and have all its power, manually schedule that Pod on node cluster2-master1. Make sure it's running. Start the kube-scheduler again and confirm its running correctly by creating a second Pod named manual-schedule2 ofimage httpd:2.4-alpine and check if it's running on cluster2-worker1.

Solution:

Context:

```
kubectl config use-context k8s-c2-AC
```

Steps:

1. **SSH into the Master Node:**
   - Use ssh cluster2-master1 to access the master node.
2. **Temporarily Stop the Kube-Scheduler:**
   - Move the kube-scheduler.yaml manifest to temporarily stop the scheduler:

```
cd /etc/kubernetes/manifests/ && mv kube-scheduler.yaml ../
```

3. **Create a Pod Named** manual-schedule**:**
   - Run the following command to create the pod:

```
kubectl run manual-schedule --image=httpd:2.4-alpine
```

   - Confirm the pod is created but not scheduled:

```
kubectl get pods
```

4. **Manually Schedule the Pod:**
   - Edit the pod's YAML to include nodeName: cluster2-master1 and apply the changes:

```
apiVersion: v1
kind: Pod
metadata:
  name: manual-schedule
spec:
  nodeName: cluster2-master1
  containers:
  - name: manual-schedule
    image: httpd:2.4-alpine
```

   - Apply the manifest:

```
kubectl apply -f <edited-pod-yaml>
```

5. **Restart the Kube-Scheduler:**
   - Move the kube-scheduler.yaml back to its original location:

```
mv ../kube-scheduler.yaml .
```

   - Verify the scheduler is running by creating another pod:

```
kubectl run manual-schedule2 --image=httpd:2.4-alpine
kubectl get pods -o wide
```

Outcome:

- The manual-schedule pod should be running on cluster2-master1.

- The manual-schedule2 pod should be scheduled and running on cluster2-worker1.

Scenario 110: Creating a ServiceAccount with Role and RoleBinding

Scenario: Use context: kubectl config use-context k8s-c1-H Create a new ServiceAccount processor in Namespace project-hamster. Create a Role and RoleBinding, both named processor as well. These should allow the new SA to only create Secrets and ConfigMaps in that Namespace.

Solution:

Context:

```
kubectl config use-context k8s-c1-H
```

Steps:

1. **Create a ServiceAccount:**
   - Create a ServiceAccount named processor in the project-hamster namespace:

```
kubectl create serviceaccount processor -n project-hamster
```

2. **Create a Role:**
   - Define a Role named processor that allows creating Secrets and ConfigMaps:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: project-hamster
  name: processor
rules:
- apiGroups: [""]
  resources: ["secrets", "configmaps"]
  verbs: ["create"]
```

   - Apply the Role definition:

```
kubectl apply -f processor-role.yaml
```

3. **Create a RoleBinding:**
   - Define a RoleBinding to bind the ServiceAccount to the Role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: processor
  namespace: project-hamster
subjects:
- kind: ServiceAccount
  name: processor
  namespace: project-hamster
roleRef:
```

```
 kind: Role
 name: processor
 apiGroup: rbac.authorization.k8s.io
```

- Apply the RoleBinding definition:

```
kubectl apply -f processor-rolebinding.yaml
```

Outcome:

- The processor ServiceAccount in the project-hamster namespace will have permissions to create Secrets and ConfigMaps.

By following these steps, you can create a well-structured Word document that clearly outlines each scenario, its steps, and expected outcomes. If you need further assistance with formatting or specific Word features, let me know!

In the up-coming parts, we will discussion on more troubleshooting steps for the different Kubernetes based scenarios. So, stay tuned for the and follow @Prasad Suman Mohan for more such posts.