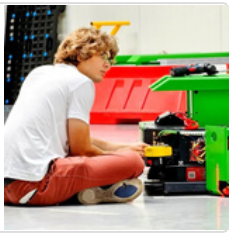
K8s Troubleshoot #03: Resolve Unbound PVC errors in StatefulSets Post-Migration

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



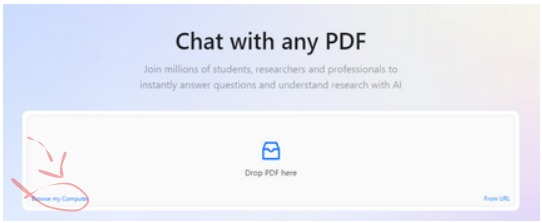
Ask Personal AI Document assistant to learn interactively (FASTER)!

1. Download PDF
2. Go to website
3. Browse file
4. Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/K8sTrouble03.pdf

Click there to go to ChatPdf website 2

3



Ask questions about document! 4

Complety new to Kubernetes?

If you are completely new to this topic, using a document assistant to understand the many definitions can be helpful. However, the best way to start is by watching this video, which I believe provides the best explanation for beginners starting their journey with Kubernetes

Kubernetes Crash Course for Absolute Beginners

Hands-On Kubernetes Tutorial | Learn Kubernetes in 1 Hour - Kubernetes Course for Beginners

https://www.youtube.com/watch?v=s_o8dwzRlu4&ab_channel=TechWorldwithNana

What is Kubernetes Troubleshooting

Kubernetes troubleshooting involves diagnosing and resolving issues within a Kubernetes cluster, such as deployment failures, pod crashes, or network problems. It requires examining logs, monitoring system metrics, and analyzing cluster configurations to identify and fix the root causes of problems.

How Kubernetes troubleshooting is done?

Kubernetes troubleshooting involves using tools like **kubectl** to inspect pod logs, events, and resource statuses to diagnose issues. Additionally, analyzing cluster metrics and leveraging Kubernetes' built-in debugging features, such as **kubectl exec** to access pod shells, can help identify and resolve problems.

When Kubernetes troubleshooting is done?

Kubernetes troubleshooting is performed when issues arise, such as

- ImagePullBackOff,
- CrashLoopBackOff errors,
- unschedulable pods,
- resource sharing conflicts,
- breaches of resource quotas or limits,
- problems with StatefulSets and Persistent Volumes
- security breaches due to faulty network policies.

This PDF will focus on second and third from the list: **Problems with StatefulSets and Persistent Volumes**

System Requirements

- 2 CPUs or more
- 2GB of free memory
- 20GB of free disk space
- Internet connection
- ubuntu 22.04

If you want to install it on a different cloud provider, ask in the comments and I'll provide a solution for you!

Kubernetes: Main components & packages

install kind

```
# For AMD64 / x86_64[ $(uname -m) = x86_64 ] && curl -Lo ./kind
https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-amd64
```

mv binary

```
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
```

Create your kind cluster. Create file **kind-example-config.yaml**

```
# three node (two workers) cluster config
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

Start your first kind cluster:

```
kind create cluster --config kind-example-config.yaml
```

If you've got error: **Failed joining worker nodes**

1) Update Docker

2) updated the ulimit for max_user_watches and max_user_instances to a higher value

```
echo fs.inotify.max_user_watches=655360 | sudo tee -a
/etc/sysctl.conf
echo fs.inotify.max_user_instances=1280 | sudo tee -a
/etc/sysctl.conf
sudo sysctl -p
```

3) If nothing works, reinstall kind and do step 1) and 2)

K8s Troubleshoot #03: StatefulSets and Persistent Volumes

1) What is StatefulSet and how its structure looks

A Kubernetes resource that manages stateful applications, which require stable network identities and persistent storage, using PersistentVolumeClaims (PVCs). It's ideal for apps like databases, where each instance must retain its data and identity across restarts.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  minReadySeconds: 10
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: registry.k8s.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: ebs
      resources:
        requests:
          storage: 1Gi
```

2) What is PVC claim

A PersistentVolumeClaim (PVC) in Kubernetes is a request for storage by a pod, allowing it to claim a specific amount of persistent storage from a PersistentVolume (PV). PVCs ensure that the storage remains available and consistent across pod restarts or rescheduling, making them essential for stateful applications.

Highlited lines in manifest file represent PVC:

```
    volumeMounts:
    - name: www
      mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
   name: www
  spec:
   accessModes: [ "ReadWriteOnce" ]
   storageClassName: ebs
   resources:
    requests:
     storage: 1Gi
```

volumeMounts: Specifies that the www volume is mounted at /usr/share/nginx/html, making it accessible to the application in the container.

volumeClaimTemplates: Creates a PersistentVolumeClaim (PVC) named www, requesting 1Gi of storage with "ReadWriteOnce" access mode, provisioned using the ebs storage class.

3) Deployment: From StatefullSet to Pod

StatefulSet Definition:

- A YAML manifest defines the StatefulSet, specifying replicas, container images, and Persistent Volume Claims (PVCs) for each pod.

PVC Creation:

- Kubernetes generates a PVC for each pod, ensuring each pod has dedicated storage (e.g., pod-0 gets pvc-0). PVCs specify size, access mode, and storage class.

StorageClass:

- A StorageClass defines storage type (e.g., SSD, HDD) and provisioning (dynamic or static). If not specified, the default cluster storage class is used.

Provisioner:

- The Provisioner, integrated with the underlying infrastructure (e.g., AWS, GCE), dynamically creates storage based on the StorageClass instructions.

Persistent Volume (PV):

- The Provisioner creates a PV based on the PVC request. Each pod gets its own bound PV, representing the actual storage.

Pod Creation and Binding:

- After binding to a PV, Kubernetes schedules the pod, mounts the storage, and ensures stable re-attachment on restart.

Summary Flow:

- StatefulSet Manifest → PVC → StorageClass → Provisioner → PV

4) Specific StatefullSet behaviour

Kubernetes initializes StatefulSet pods one by one to ensure that each pod, such as a database instance, gets a stable hostname and its own Persistent Volume in a controlled sequence.

This ordered approach ensures that the primary database is deployed and operational before any backup or replica databases are started.

5) Storage Provisioners

In AWS, the native storage provisioners are EBS (Elastic Block Store) and EFS (Elastic File System). To enable your pods to use different storage services, such as NetApp, you need a CSI (Container Storage Interface) driver.

The CSI driver allows StatefulSets to create Persistent Volume Claims (PVCs) that utilize custom storage provisioners. This integration makes it possible to manage and use non-native storage solutions seamlessly within your Kubernetes environment.

K8s Troubleshoot #03: Unbound PVC error

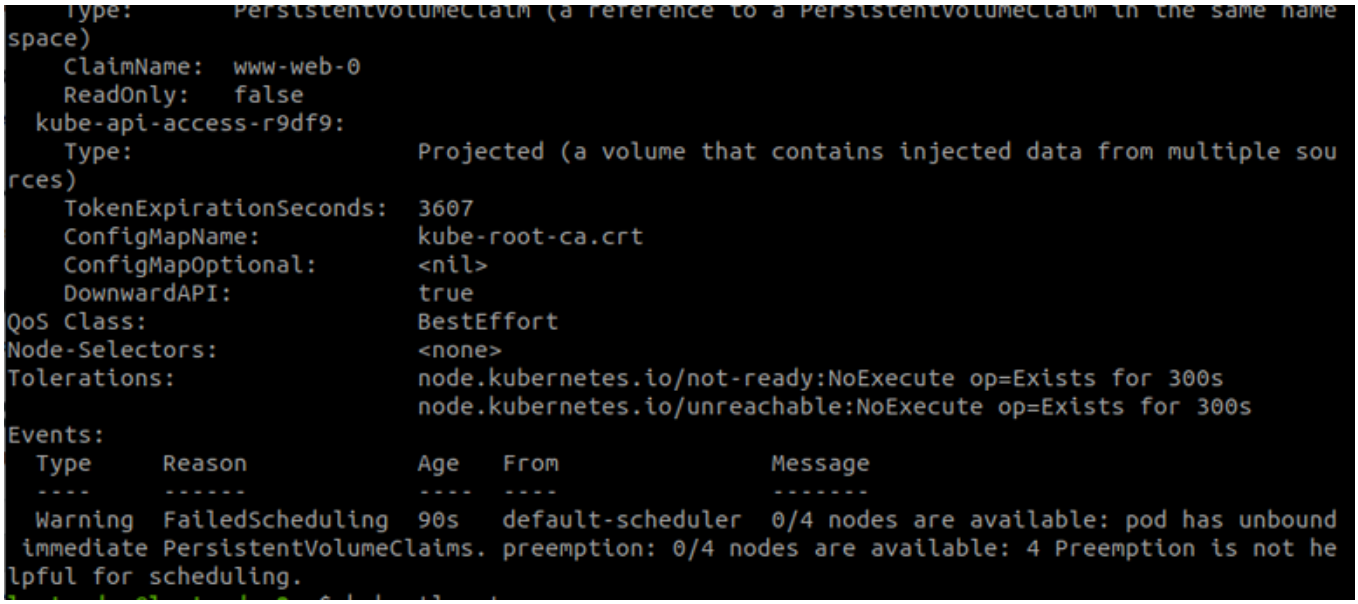
1) Understand popular case study

A developer has successfully deployed a stateful application on Amazon EKS (Elastic Kubernetes Service), utilizing AWS EBS for persistent storage. The application relies on StatefulSets and Persistent Volume Claims (PVCs) to manage its state.

The developer now aims to migrate the application to other Kubernetes environments, specifically Azure AKS (Azure Kubernetes Service) and Google GKE (Google Kubernetes Engine). However, the application encounters issues related to storage provisioning in these new environments.

2) Error explanation

The application works flawlessly on EKS but fails to deploy properly on AKS and GKE due to compatibility issues with the storage provisioners. AWS EBS is not available on AKS or GKE, and the application's PVCs are not being satisfied by the default storage classes in these environments.



3) Solution

For minikube, kind Clusters use Standard standard storage class, modify PVC claim:

```
      volumeMounts:
      - name: my-storage
        mountPath: /data
volumeClaimTemplates:
- metadata:
    name: my-storage
  spec:
    accessModes: [ "ReadWriteOnce" ]
    resources:
      requests:
        storage: 10Gi
    storageClassName: standard
```

For AKS use managed-csi StorageClass, modify PVC claim:

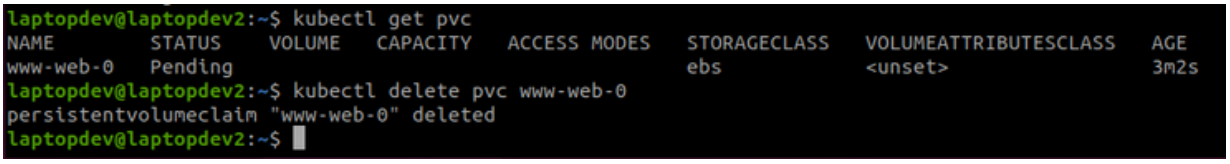
```
      volumeMounts:
      - name: nginx-storage
        mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
    name: nginx-storage
  spec:
    accessModes: ["ReadWriteOnce"]
    storageClassName: "managed-csi"  # Use the managed-csi StorageClass
    resources:
      requests:
        storage: 10Gi  # Size of the persistent volume
```

For AKS use managed-csi StorageClass, modify PVC claim:

```
      volumeMounts:
      - name: nginx-storage
        mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
    name: nginx-storage
  spec:
    accessModes: ["ReadWriteOnce"]
    storageClassName: "standard"  # GKE default StorageClass
    resources:
      requests:
        storage: 10Gi  # Size of the persistent volume
```

4) Solution implementation

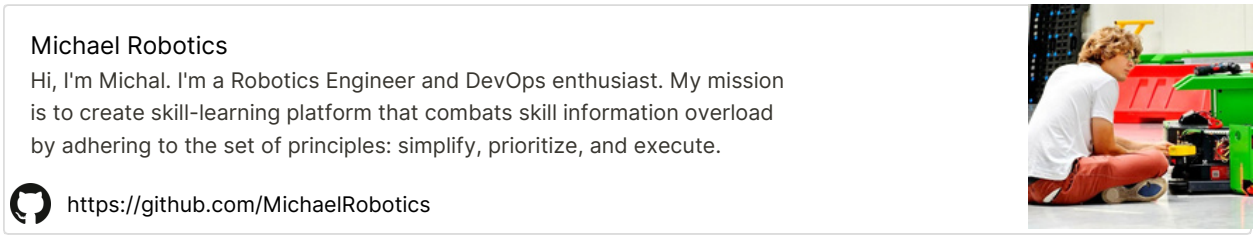
Delete old faulty pvc before creating new one



now everything should work fine. Deploy new statefullset manifest file.

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub



PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!