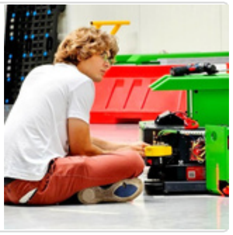
K8s Troubleshoot #04: Resource Sharing, Thread&heap dump, Upgrades

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

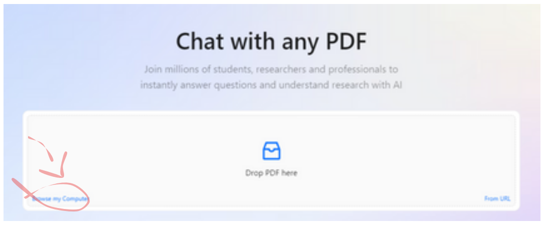
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/K8sTrouble04.pdf

Click there to go to ChatPdf website 2

3



Chat with any PDF

Join millions of students, researchers and professionals to instantly answer questions and understand research with AI

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

TECHWORLD WITH NANA

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.

kubernetes

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.
- Upgrades

This PDF will focus onfifth and last from the list: **breaches of resource quotas or limits** and **Upgrades**

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 #04: Resource Sharing

1) Understand popular case study

A company with dev, QA, and prod environments faced an issue where multiple project teams shared a single Kubernetes cluster in production, organized through namespaces. One of the namespaces began consuming nearly all CPU and RAM resources, causing performance bottlenecks across the cluster.

2) Error explanation

This issue is evident when Out Of Memory (OOM) kills occur, as the system forcibly terminates pods that exceed available memory. These OOM kills indicate that certain workloads are consuming excessive resources, often caused by memory leaks, where applications progressively use more memory without releasing it.

3) Solution

The solution is to implement Kubernetes resource limits and quotas for each namespace to prevent any single project from monopolizing CPU and memory.

Additionally, monitoring tools like Prometheus or Grafana should be set up to track resource usage and trigger alerts if usage approaches critical levels, ensuring balanced resource allocation across all namespaces.

4) Resource Quota

Kubernetes is a policy that limits the amount of resources (such as CPU, memory, and storage) that each namespace can consume within a Kubernetes cluster

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example-quota
  namespace: your-namespace
spec:
  hard:
    pods: "10"                 # Maximum number of pods
    requests.cpu: "4"          # Maximum total CPU requests (in cores)
    requests.memory: "8Gi"     # Maximum total memory requests (in gigabytes)
    limits.cpu: "8"            # Maximum total CPU limit
    limits.memory: "16Gi"      # Maximum total memory limit
```

5) Resource Limit

in Kubernetes is a constraint applied to individual containers within a pod, defining the maximum amount of CPU and memory that the container can use

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  namespace: your-namespace
spec:
  containers:
  - name: example-container
    image: nginx
    resources:
      requests:
        memory: "256Mi"  # Minimum 256 MiB of memory guaranteed
        cpu: "500m"      # Minimum 0.5 CPU cores guaranteed
      limits:
        memory: "512Mi"  # Maximum 512 MiB of memory
        cpu: "1"         # Maximum 1 CPU core
```

K8s Troubleshoot #03: Thread&heap dump

1) Understand popular case study

A DevOps engineer noticed that, even though resource requests and limits were configured, one specific pod continued to get OOMKilled, meaning it exhausted its assigned memory, causing the system to forcibly terminate it. The pod then entered a CrashLoopBackOff state, repeatedly failing and restarting, causing service disruption.

2) Error explanation

The error occurs because the application within the pod has a bug that causes excessive memory or CPU usage, leading it to consistently exceed its assigned memory and CPU limits

3) Solution

the solution involves collecting detailed diagnostic information about CPU and memory usage from the application. In case of Java app, this is achieved by obtaining **heap dumps** and **thread dumps** and then sending them to the development team for analysis.

Java

Heap dump

snapshot of the application's memory that helps developers identify memory leaks and excessive consumption by showing all objects, their references, and sizes.

```
jmap -dump:live,format=b,file=heapdump.hprof <pid>
```

Thread dump

Captures the state and stack traces of all threads at a specific time, helping diagnose issues like contention, deadlocks, and high CPU usage.

```
jstack <pid> > threaddump.txt
```

or

```
kill -3 <pid>
```

K8s Troubleshoot #03: Upgrades

1) Understand popular case study

ABC Tech, a fast-growing SaaS company, relied on Kubernetes for managing its microservices architecture but faced performance, security, and management challenges as its user base grew. To ensure reliability and scalability, the company upgraded its Kubernetes cluster from version 1.18 to 1.22.

2) Error explanation

During ABC Tech's last Kubernetes cluster upgrade, the company encountered several issues that caused temporary disruptions and required additional effort to resolve. Learning from these challenges was critical in planning the most recent upgrade

3) Solution

The key to a successful Kubernetes (K8s) cluster upgrade is to develop a comprehensive upgrade plan that follows specific procedures, adheres to official upgrade manuals, and minimizes the risk of downtime or service disruption.

4) Upgrade steps

First, find desired target version and upgrade Plan. Check official k8s website:

Upgrading kubeadm clusters

This page explains how to upgrade a Kubernetes cluster

https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

Backup your data, use **Velero** or other solution like **Kasten** (Kubernetes native):

Kubernetes Backup and Restore: A Comprehensive Guide

Kubernetes, the popular container orchestration platform, empowers organizations

https://medium.com/@tadbiri2012/kubernetes-backup-and-restore-a-comprehensive-guide

Upgrade controll plane components (tutorial for kubeadm clusters)

Upgrading kubeadm clusters

This page explains how to upgrade a Kubernetes cluster

https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

Upgrade worker nodes one by one, through a rolling upgrade plan

Performing a Rolling Update

Perform a rolling update using kubectl.

https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

Linux node upgrade:

Upgrading Linux nodes

This page explains how to upgrade a Linux Worker Nodes

https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/upgrading-linux-nodes/

Then check if cluster works properly.

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

Check my GitHub

Michael Robotics

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

https://github.com/MichaelRobotics

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!