



Part 11: 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.

The screenshot shows a LinkedIn post titled "PART 11 - KUBERNETES REAL-TIME TROUBLESHOOTING". It features a blue header bar with the title and a profile picture of Prasad Suman Mohan. Below the header, there's a large blue arrow pointing right. On the left side, there's a blue hexagonal icon with a white steering wheel inside, and the word "kubernetes" in a large, semi-transparent font. To the right of the icon, a bulleted list of troubleshooting topics is displayed:

- Kubernetes API Server Rate Limiting
- Pod Anti-affinity rules not respected
- Cluster Autoscaler Failures
- CronJob Failures
- Pod Node Selector Misconfigurations

Scenario 51: Kubernetes API Server Rate Limiting



<https://www.linkedin.com/in/prasad-suman-mohan>



Symptoms: API server returns 429 Too Many Requests errors due to rate limiting, causing client requests to fail or experience high latency.

Diagnosis: Monitor API server metrics and logs (`kubectl get --raw /metrics`, `kubectl logs -n kube-system <api_server_pod>`) to identify rate limiting occurrences and affected clients.

```
},
response: {
  status: 429,
  statusText: 'Too Many Requests',
  headers: {
    date: 'Wed, 24 May 2023 16:15:36 GMT',
    'content-type': 'application/json',
    'content-length': '349',
    connection: 'keep-alive',
    'access-control-allow-origin': '*',
    'openai-model': 'gpt-3.5-turbo-0301',
    'openai-organization': 'niche-mates',
    'openai-processing-ms': '30008',
    'openai-version': '2020-10-01',
    'strict-transport-security': 'max-age=15724800; includeSubDomains',
    'x-ratelimit-limit-requests': '3500',
    'x-ratelimit-limit-tokens': '90000',
    'x-ratelimit-remaining-requests': '3498',
    'x-ratelimit-remaining-tokens': '89914',
    'x-ratelimit-reset-requests': '19ms',
    'x-ratelimit-reset-tokens': '56ms',
    'x-request-id': '0ddf4b34092d830cc9cff11bdaa8ce70',
    'cf-cache-status': 'DYNAMIC',
    server: 'cloudflare',
    'cf-ray': '7cc6d5df9ab90c3b-AMS',
    'alt-svc': 'h3=":443"; ma=86400, h3-29=:443"; ma=86400'
  },
  ...
}
```

Solution:

1. Optimize client applications and controllers to reduce the frequency and volume of API requests, implementing request batching or caching where possible.
2. Adjust API server rate limit configurations (`--max-requests-inflight`, `--max-mutating-requests-inflight`) to accommodate higher request loads.
3. Scale out the API server by adding more instances in a high-availability setup to distribute the request load and reduce rate limiting.
4. Use a dedicated client certificate with higher rate limits for critical or high-priority clients to ensure their requests are not throttled.



Scenario 51: Pod Anti-Affinity Rules Not Respected

```
GNU nano 6.2                                         webapp-pod.yaml *
apiVersion: v1
kind: Pod
metadata:
  name: webapp-pod ← Pod name
spec:
  containers:
    - image: nginx ← Container image
      name: webapp-pod ← Container name
  nodeSelector:
    size: large ← A node selector defined under the 'spec' section of the Pod
```

hashicorp/consul-helm

#243 Pods goes in pending state due to podAntiAffinity rules



17 comments

msidd111 opened on October 8, 2019



Symptoms: Pods are scheduled on the same node despite having pod anti-affinity rules, leading to suboptimal distribution and potential single points of failure.

Diagnosis: Describe the affected pods (`kubectl describe pod <pod_name>`) and inspect the anti-affinity rules and node assignments to verify if the rules are being applied correctly.

Solution:

1. Review and adjust the pod anti-affinity rules in the pod templates to ensure they are correctly defined and feasible within the cluster's node resources.
2. Verify that the scheduler's configuration supports pod anti-affinity rules and is correctly prioritizing them during scheduling decisions.
3. Use taints and tolerations in conjunction with anti-affinity rules to achieve the desired pod distribution and placement across the cluster.
4. Scale out the cluster by adding more nodes to provide additional scheduling options and improve agreement to anti-affinity rules.



Scenario 53: Cluster Autoscaler Failures

kubernetes/autoscaler

#6128 cluster-autoscaler gets stuck with "Failed to fix node..."



43 comments

 com6056 opened on September 22, 2023



Symptoms: Cluster autoscaler fails to add or remove nodes as needed, leading to resource shortages or excessive costs.

Diagnosis: Check the cluster autoscaler logs (`kubectl logs -n kube-system cluster-autoscaler`) and inspect node status and scaling events to identify failures or misconfigurations.

Solution:

1. Verify that the cluster autoscaler configuration matches the cloud provider's scaling policies and resource quotas.
2. Ensure that the autoscaler has the necessary permissions to create and delete nodes within the cluster.
3. Adjust the scaling parameters (e.g., min/max node counts, scale-down delay) to optimize the autoscaler's responsiveness and efficiency.
4. Monitor autoscaler performance and scaling events to identify and address any issues or bottlenecks in the scaling process.

Scenario 54: CronJob Failures

Symptoms: CronJobs fail to execute as scheduled, leading to missed tasks or inconsistent execution.

Diagnosis: Check the CronJob status and events (`kubectl get cronjob`, `kubectl describe cronjob <cronjob_name>`) and inspect the logs of the failed job pods (`kubectl logs <pod_name>`).



kubernetes/kubernetes

#108023 Cronjob fails to be scheduled – reports JobAlreadyActive even...



6 comments



simonjpartridge opened on February 9, 2022



Solution:

1. Ensure that the CronJob schedule syntax is correct and conforms to the expected cron format.
2. Verify that the job template specified in the CronJob is correctly defined and contains valid configurations.
3. Check for resource constraints or quotas that may prevent job pods from being scheduled or running successfully.
4. Review and adjust CronJob concurrency policies (`spec.concurrencyPolicy`) to manage overlapping executions and prevent conflicts.

Scenario 55: Pod Node Selector Misconfiguration

```
surender@kube-srv1:~$ kubectl get nodes kube-srv3.testlab.local --show-labels
NAME           STATUS   ROLES   AGE    VERSION   LABELS
kube-srv3.testlab.local   Ready    <none>   6d    v1.27.3   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=kube-srv3.testlab.local,kubernetes.io/os=linux,size=large ←
surender@kube-srv1:~$ kubectl describe nodes kube-srv3.testlab.local
Name:           kube-srv3.testlab.local
Roles:          <none>
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/arch=amd64
                kubernetes.io/hostname=kube-srv3.testlab.local
                kubernetes.io/os=linux
Annotations:    flannel.alpha.coreos.com/backend-data: {"VNI":1,"VtepMAC":"1a:d0:e9:5d:15:7a"} ← Well-known (default) labels
                size=large ← Manually set label
Annotations:    flannel.alpha.coreos.com/backend-data: {"VNI":1,"VtepMAC":"1a:d0:e9:5d:15:7a"}
```

Symptoms: Pods remain in a pending state due to unsatisfiable node selector requirements, causing deployment delays.



Diagnosis: Describe the affected pods (`kubectl describe pod <pod_name>`) and inspect the node selector and node labels for mismatches.

Solution:

```
surender@kube-srv1:~$ kubectl describe node kube-srv3.testlab.local
Name:           kube-srv3.testlab.local
Roles:          <none>
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/arch=amd64
                kubernetes.io/hostname=kube-srv3.testlab.local
                kubernetes.io/os=linux
Annotations:   flannel.alpha.coreos.com/backend-data: {"VNI":1,"VtepMAC":"be:d3:68:24:f4:26"}
                flannel.alpha.coreos.com/backend-type: vxlan
                flannel.alpha.coreos.com/kube-subnet-manager: true
                flannel.alpha.coreos.com/public-ip: 192.168.0.42
                kubeadm.alpha.kubernetes.io/cri-socket: unix:///var/run/cri-dockerd.sock
                node.alpha.kubernetes.io/ttl: 8
                volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Wed, 05 Jul 2023 14:05:01 +0530
Taints:          <none>
Unschedulable:   false
Lease:
  HolderIdentity: kube-srv3.testlab.local
  AcquireTime:    <unset>
  RenewTime:     Sat, 08 Jul 2023 14:34:36 +0530
Conditions:
  Type        Status  LastHeartbeatTime          LastTransitionTime        Reason
  Message
  ----        -----  -----                    -----                  -----
  NetworkUnavailable  False   Sat, 08 Jul 2023 14:33:08 +0530  Sat, 08 Jul 2023 14:33:08 +0530  FlannelIsUp
  Flannel is running on this node
  MemoryPressure    False   Sat, 08 Jul 2023 14:32:54 +0530  Sat, 08 Jul 2023 14:32:54 +0530  KubeletHasSufficientMemory
  kubelet has sufficient memory available
  DiskPressure      False   Sat, 08 Jul 2023 14:32:54 +0530  Sat, 08 Jul 2023 14:32:54 +0530  KubeletHasNoDiskPressure
  kubelet has no disk pressure
  PIDPressure       False   Sat, 08 Jul 2023 14:32:54 +0530  Sat, 08 Jul 2023 14:32:54 +0530  KubeletHasSufficientPID
  kubelet has sufficient PID available
  Ready             True    Sat, 08 Jul 2023 14:32:54 +0530  Sat, 08 Jul 2023 14:32:54 +0530  KubeletReady
  kubelet is posting ready status. AppArmor enabled
Addresses:
  InternalIP: 192.168.0.42
  Hostname:   kube-srv3.testlab.local
Capacity:
  cpu:           4 ← 4 CPUs
  ephemeral-storage: 30136644Ki
  hugepages-2Mi:  0
  memory:        4833672Ki ← 4 GiB memory
  pods:          110
```

1. Verify that the node selector in the pod specification matches the labels on the intended nodes.
2. Check for any node taints that may prevent pods from being scheduled and apply corresponding tolerations in the pod specification.
3. Adjust the node selector or labels on the nodes to align with the deployment requirements and facilitate pod scheduling.
4. Use node affinity rules to provide more flexible and advanced scheduling constraints based on node characteristics.



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

