Part 9 - 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.

Scenario 41: Resource Quota Exceedance

Symptoms: Pods fail to start or run due to resource quota exceedance, resulting in scheduling failures or evictions and impacting application performance or availability.

Diagnosis: Review resource quota definitions (`kubectl get quota`) and inspect pod resource requests and limits (`kubectl describe pod <pod_name>`) to identify any resource constraints or exceedances.

Solution:

1. Adjust resource quota limits and allocations to accommodate the resource requirements of pods and workloads running within the cluster, based on anticipated resource usage patterns and application requirements.
2. Implement pod resource management policies (e.g., PodDisruptionBudgets, LimitRanges) to enforce resource limits and prevent resource contention or oversubscription within namespaces or deployments.
3. Monitor resource utilization metrics (e.g., CPU, memory, storage) using Kubernetes monitoring tools (e.g., Prometheus, Grafana) to identify resource-intensive workloads or containers and optimize resource allocation accordingly.
4. Use vertical and horizontal pod autoscaling (HPA) to dynamically adjust pod resource allocations based on workload demand and resource utilization metrics, scaling pods up or down to maintain optimal performance and resource efficiency.

Scenario 42: Node Resource Exhaustion

Symptoms: Nodes become unresponsive or experience performance degradation due to resource exhaustion (e.g., CPU, memory, disk), resulting in pod scheduling failures or evictions.

Diagnosis: Monitor node resource utilization metrics (`kubectl top nodes`) and inspect system logs (`kubectl logs -n kube-system <node_pod>`) for any resource-related errors or warnings.

```
Non-terminated Pods:          (8 in total)
  Namespace                   Name                                    CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ---------                   ----                                    ------------  ----------  ---------------  -------------
  default                     devops-deployment-7df96d85c6-xrmbf      600m (31%)    600m (31%)  650Mi (30%)      650Mi (30%)
  kube-system                 coredns-696c4d987c-fpt9l                100m (5%)     0 (0%)      70Mi (3%)        170Mi (7%)
  kube-system                 coredns-696c4d987c-s5ncw                100m (5%)     0 (0%)      70Mi (3%)        170Mi (7%)
  kube-system                 coredns-autoscaler-657d77ffbf-b59hp     20m (1%)      0 (0%)      10Mi (0%)        0 (0%)
  kube-system                 kube-proxy-x26xs                        100m (5%)     0 (0%)      0 (0%)           0 (0%)
  kube-system                 kubernetes-dashboard-6f697bd9f5-hns2v   100m (5%)     100m (5%)   50Mi (2%)        500Mi (23%)
  kube-system                 metrics-server-58699455bc-s467b         0 (0%)        0 (0%)      0 (0%)           0 (0%)
  kube-system                 tunnelfront-7f6484d6b5-xwmsb            10m (0%)      0 (0%)      64Mi (2%)        0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource                    Requests     Limits
  --------                    --------     ------
  cpu                         1030m (53%)  700m (36%)
  memory                      914Mi (42%)  1490Mi (69%)
  attachable-volumes-azure-disk  0            0
Events:                       <none>
```

Solution:

1. Identify resource-intensive workloads or containers running on affected nodes and optimize resource allocations or scale them down to reduce resource usage.
2. Add more nodes to the cluster or increase node capacity (e.g., CPU, memory, disk) to accommodate growing resource demands and alleviate resource contention.
3. Implement node resource management policies (e.g., node allocatable resources, QoS classes) to enforce resource limits and prevent node resource exhaustion or oversubscription.
4. Configure node maintenance and auto-scaling policies to dynamically adjust node resources based on workload demand and resource utilization patterns, scaling nodes up or down as needed to maintain cluster stability and performance.

Scenario 43: Control Plane Component Failures

Symptoms: Kubernetes control plane components (e.g., API server, scheduler, controller manager) become unavailable or fail to respond, causing cluster management operations to fail or stall.

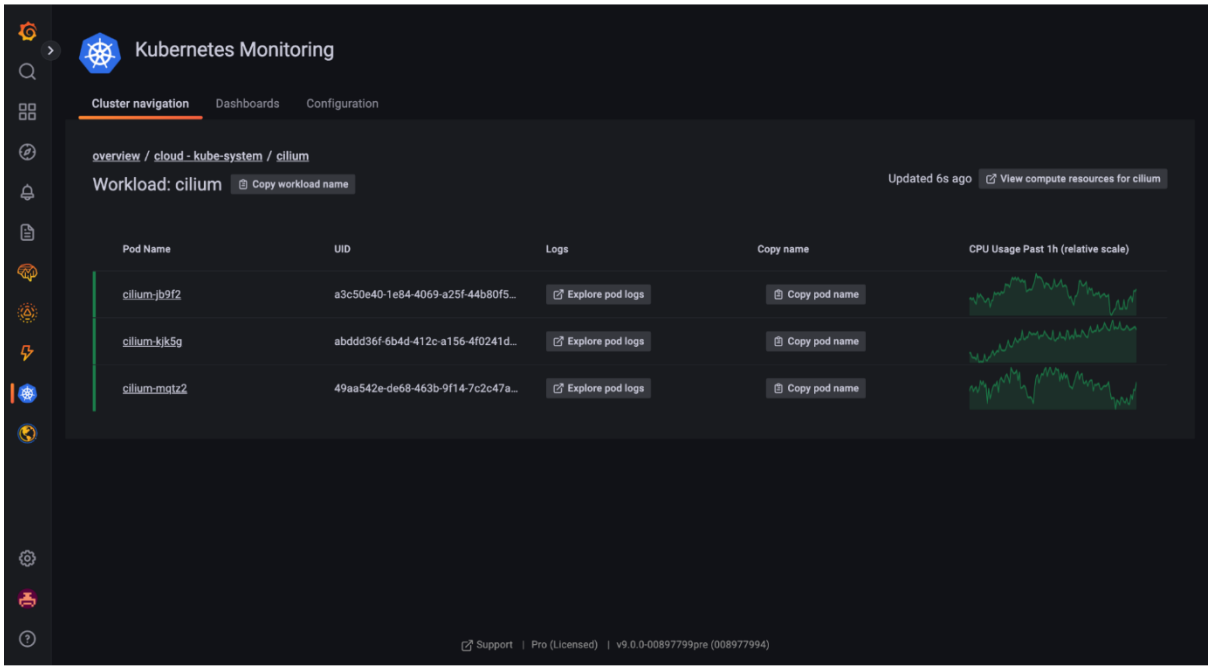
Diagnosis: Check control plane component statuses (`kubectl get componentstatuses`) and review control plane component logs (`kubectl logs -n kube-system <component_pod>`) for any errors or failures.

Solution:

1. Implement control plane component high availability (HA) configurations (e.g., multiple replicas, leader election) to ensure redundancy and fault tolerance of critical Kubernetes control plane services.
2. Monitor control plane component health and performance metrics (e.g., API server latency, scheduler throughput) using Kubernetes monitoring tools (e.g., Prometheus, Alertmanager) to detect and mitigate potential issues proactively.

3. Conduct regular control plane component backups and disaster recovery drills to prepare for and recover from control plane component failures or data corruption events effectively.
4. Update Kubernetes control plane components to the latest stable releases and apply security patches and bug fixes promptly to mitigate known vulnerabilities and improve cluster reliability and resilience.

Scenario 44: External Service Dependency Failures



Symptoms: Applications experience connectivity issues or service disruptions due to failures in external service dependencies (e.g., databases, APIs, third-party services).

Diagnosis: Monitor application logs and metrics (`kubectl logs <pod_name>`, Prometheus metrics) for any errors or timeouts related to external service calls and inspect network traffic flows (`kubectl exec -it <pod_name> -- /bin/sh`) to diagnose connectivity problems.

Solution:

1. Implement circuit breaker and retry mechanisms in application code to handle transient failures and timeouts when making calls to external services and prevent cascading failures and service degradation.
2. Configure service level agreements (SLAs) and health checks for external dependencies to monitor their availability and responsiveness and trigger alerts or failovers in case of service failures or performance degradation.
3. Use Kubernetes service discovery and load balancing features (e.g., Service, Endpoint, Ingress) to route traffic to healthy instances of external services and distribute the workload evenly across multiple endpoints.
4. Establish service level objectives (SLOs) and error budgets for external service dependencies to set performance targets and track service reliability and availability metrics, helping to prioritize and allocate resources for service improvement efforts.

Scenario 45: Persistent Volume Claim (PVC) Binding Issues

Symptoms: Pods remain in a pending state because their Persistent Volume Claims (PVCs) cannot bind to a Persistent Volume (PV).

Diagnosis: Check the PVC and PV status (`kubectl get pvc`, `kubectl get pv`) and describe the PVC (`kubectl describe pvc <pvc_name>`) to find binding errors or unmet storage class requirements.

Solution:

1. Verify that there are available PVs that match the requested storage class, capacity, and access modes specified in the PVC.
2. Ensure that the storage provisioner (e.g., CSI driver) is correctly configured and operational to dynamically provision PVs if necessary.
3. Adjust PVC or PV specifications to resolve mismatches (e.g., increase capacity, change storage class) and facilitate successful binding.
4. Implement monitoring and alerting for storage capacity utilization to proactively manage and expand storage resources as needed.

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.