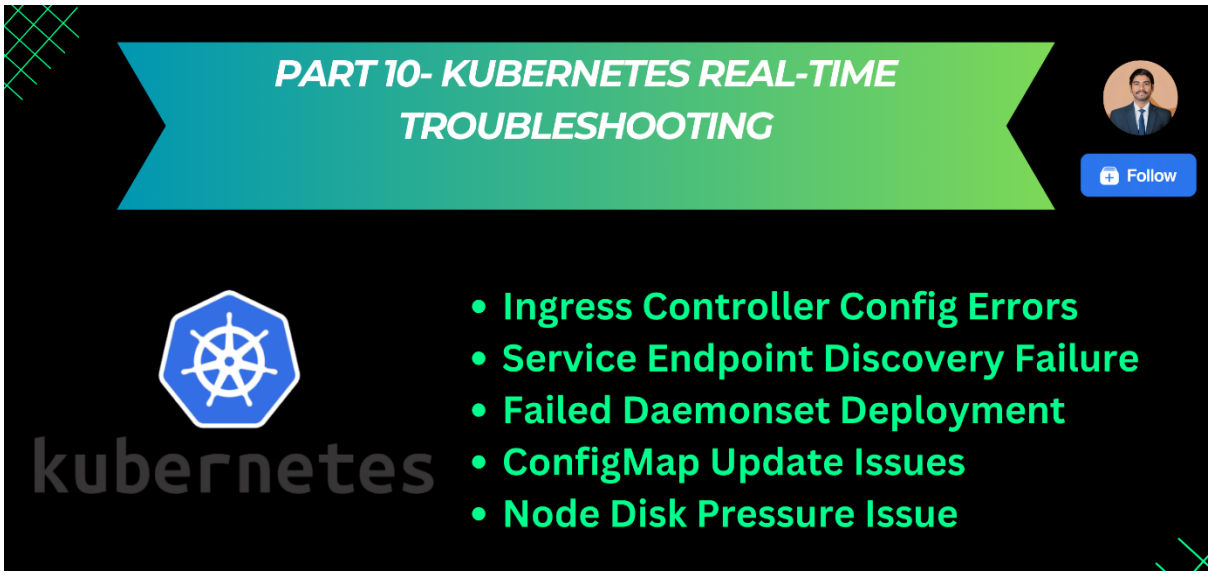
Part 10: Kubernetes Real-Time Troubleshooting

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 46: Ingress Controller Configuration Errors

Symptoms: Ingress resources do not route traffic correctly to backend services, causing HTTP 404 errors or other connectivity issues.

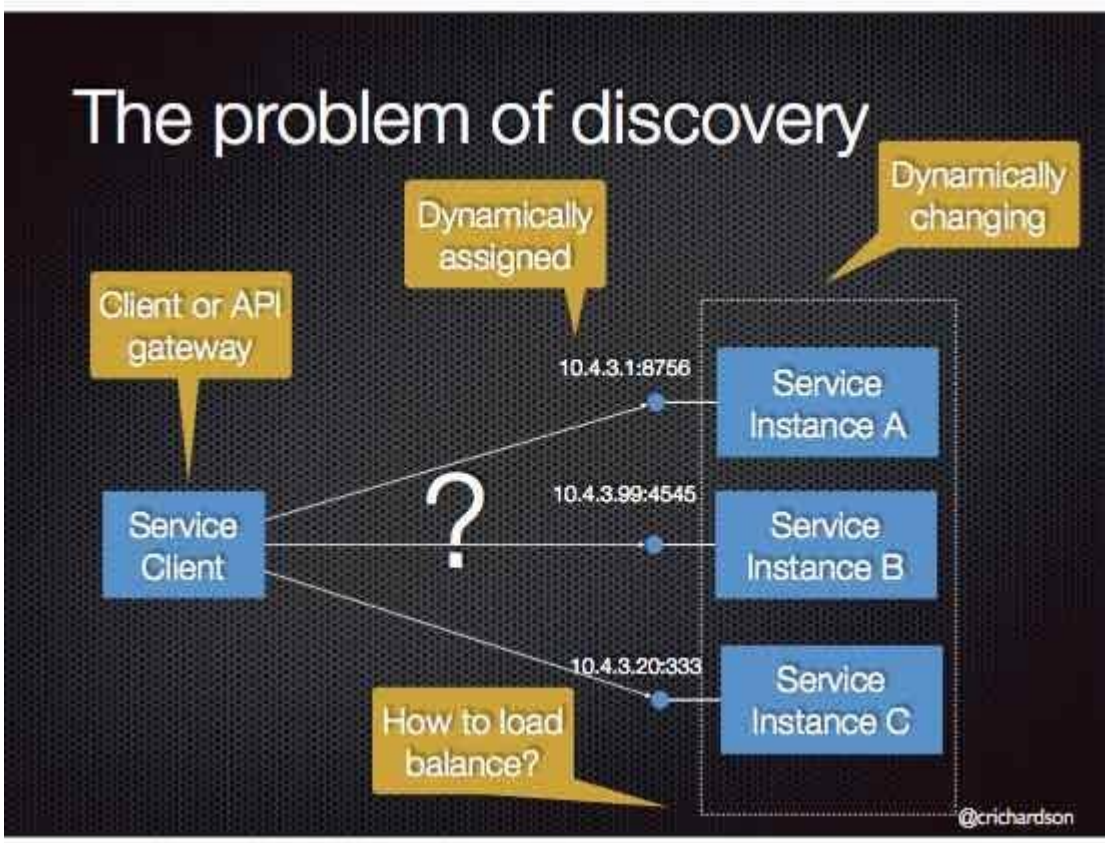
Diagnosis: Check the Ingress resource and Ingress controller logs (`kubectl get ingress`, `kubectl logs -n kube-system <ingress_controller_pod>`) for any configuration errors or routing issues.

Solution:

1. Verify that the Ingress resource specifies the correct backend service names and ports and that the services are correctly defined and operational.
2. Inspect the Ingress controller configuration and ensure it is correctly set up to handle the specified Ingress resources (e.g., correct annotations, SSL certificates).
3. Adjust network policies and firewall rules to allow traffic to and from the Ingress controller.
4. Use a network debugging tool like `kubectl exec -it <pod_name> -- curl <service_url>` to test connectivity and diagnose routing issues from within the cluster.

Scenario 47: Service Endpoint Discovery Failure

Symptoms: Services are not reachable because endpoints are not correctly discovered or updated, resulting in application connectivity issues.



Diagnosis: Inspect the service and endpoint objects (`kubectl get svc`, `kubectl get endpoints`) and describe the service (`kubectl describe svc <service_name>`) to check endpoint assignments.

Solution:

1. Verify that the pods backing the service are running and have the correct labels to match the service's selector.
2. Ensure that network policies are not blocking traffic between services and endpoints, and adjust them if necessary.
3. Investigate kube-proxy logs (`kubectl logs -n kube-system kube-proxy-<node_name>`) for any errors related to service endpoint updates and address any issues found.
4. Restart the affected services or kube-proxy pods to force endpoint reconciliation and update the service endpoint mappings.

Scenario 48: Failed DaemonSet Deployments

Symptoms: DaemonSet pods are not scheduled on all nodes as expected, leading to incomplete or inconsistent deployments across the cluster.

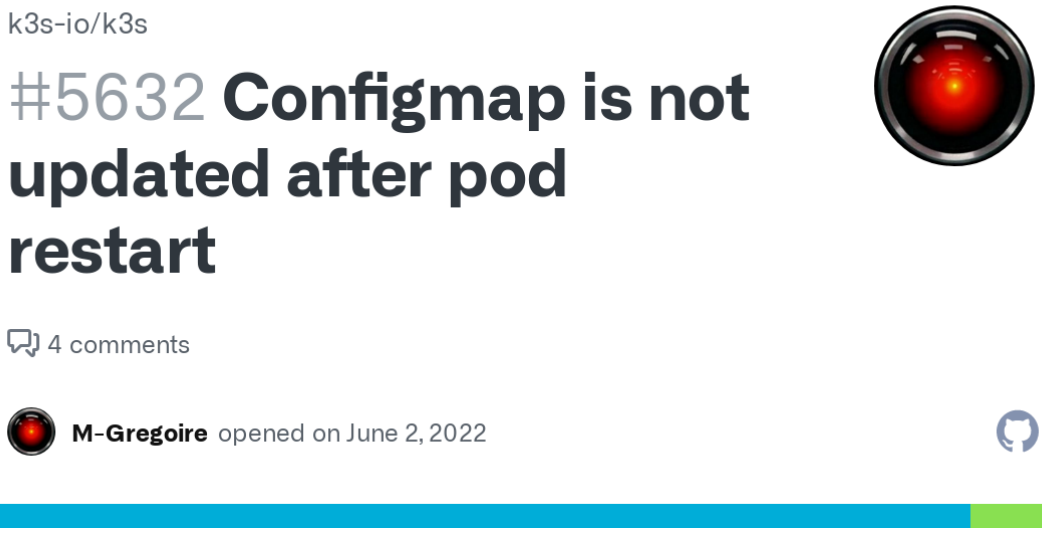
Diagnosis: Check the DaemonSet status and events (`kubectl get daemonset`, `kubectl describe daemonset <daemonset_name>`) to identify scheduling issues or resource constraints.

Solution:

1. Ensure that there are no node selectors or taints that prevent the DaemonSet pods from being scheduled on certain nodes.
2. Verify that there are enough resources (e.g., CPU, memory) available on all nodes to accommodate the DaemonSet pods.

3. Check node labels and adjust the DaemonSet's node selector or affinity rules to match the intended nodes for deployment.
4. Restart the affected DaemonSet to force re-evaluation of pod scheduling and deployment across the cluster.

Scenario 49: ConfigMap Update Issues



Symptoms: Pods do not reflect changes made to ConfigMaps, causing applications to use outdated configurations.

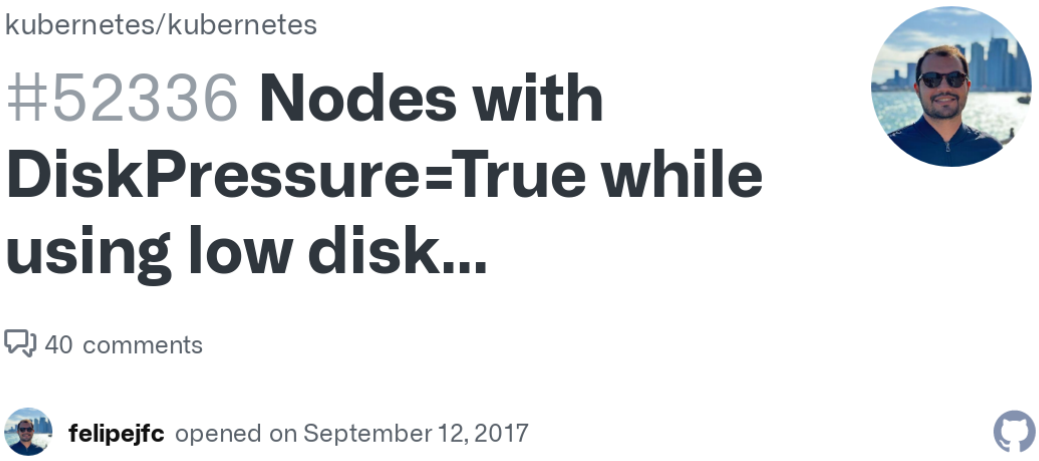
Diagnosis: Describe the ConfigMap and pods (`kubectl describe configmap <configmap_name>`, `kubectl describe pod <pod_name>`) to verify the configurations and check if the pods are using the latest ConfigMap version.

Solution:

1. Ensure that the pods are correctly mounted with the updated ConfigMap and restart the pods if necessary to apply the new configuration.
2. Use ConfigMap checksum annotations in pod templates to trigger rolling updates when ConfigMap changes are detected.
3. Implement automated rolling updates for deployments using `kubectl rollout restart deployment <deployment_name>` to ensure that all pods are updated with the latest ConfigMap changes.
4. Monitor ConfigMap updates and application logs to verify that configuration changes are correctly applied and functioning as expected.

Scenario 50: Node Disk Pressure Issue

Symptoms: Nodes report disk pressure and evict pods to reclaim disk space, causing application disruptions.



Diagnosis: Check node status and conditions (`kubectl describe node <node_name>`) and monitor disk usage metrics (`kubectl top nodes`) to identify nodes under disk pressure.

Solution:

1. Identify and clean up unused or temporary files and data (e.g., logs, caches) on affected nodes to free up disk space.
2. Configure eviction thresholds and policies (`kubelet` configuration) to manage disk pressure more effectively and prevent premature pod evictions.
3. Increase disk capacity on affected nodes or add additional nodes with sufficient disk space to the cluster.
4. Use persistent volumes and storage classes to offload disk usage from nodes to external storage solutions (e.g., network-attached storage, cloud storage).

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.