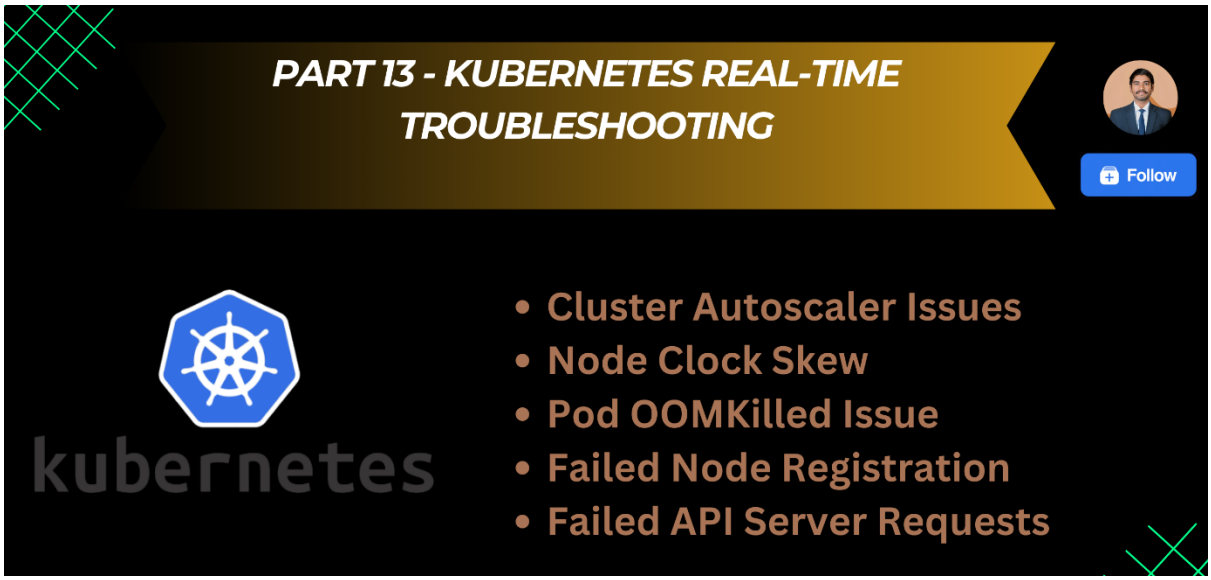
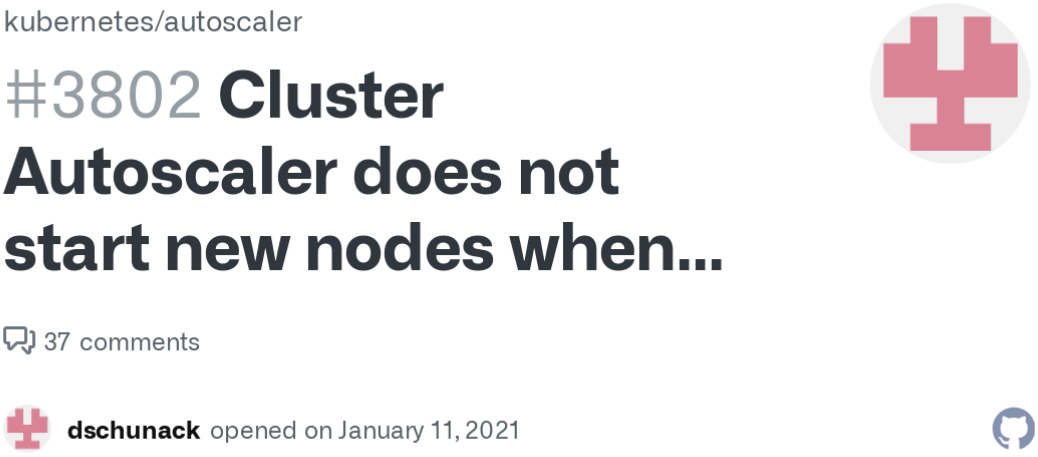
Part 13: Kubernetes Real-Time Troubleshooting

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 61: Cluster Autoscaler Issues



Symptoms: The cluster autoscaler fails to scale nodes up or down as expected.

Diagnosis: Inspect the cluster autoscaler logs (`kubectl logs -n kube-system deployment/cluster-autoscaler`) and node status (`kubectl get nodes`).

Solution:

1. Ensure that the cluster autoscaler configuration matches the node group's scaling policies and resource limits.
2. Verify that the cloud provider's IAM roles and permissions allow scaling actions.
3. Check for any pending pods that cannot be scheduled and ensure sufficient resource requests.
4. Adjust the cluster autoscaler parameters for more aggressive or conservative scaling behaviors.

Scenario 62: Node Clock Skew

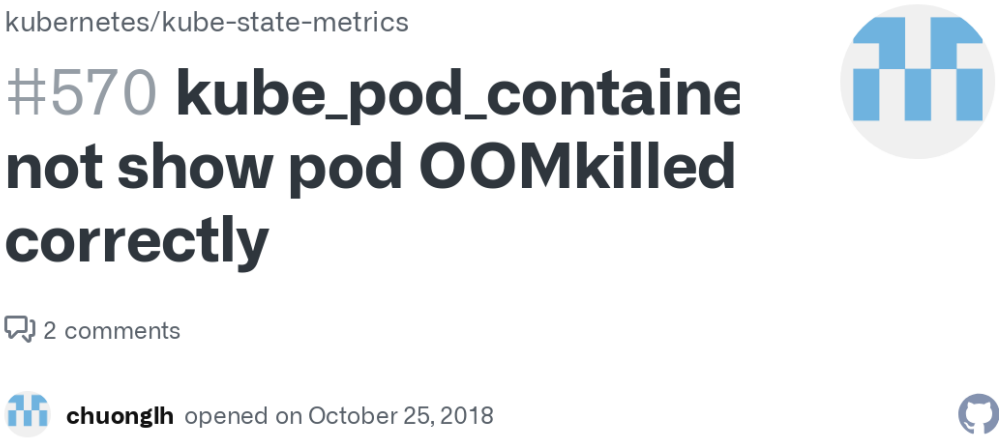
Symptoms: Cluster components experience synchronization issues due to clock skew between nodes.

Diagnosis: Check the system time on the nodes (`date` command) and compare with a reliable time source.

Solution:

1. Ensure that all nodes are synchronized with a Network Time Protocol (NTP) server.
2. Install and configure `chrony` or `ntpd` on the nodes to maintain accurate system time.
3. Monitor clock synchronization status and adjust NTP server settings if necessary.

Scenario 63: Pod OOMKilled Issue



Symptoms: Pods are frequently terminated with an OOMKilled (Out of Memory) error, leading to application instability.

Diagnosis: Inspect the pod status (`kubectl get pods`) and describe the affected pods (`kubectl describe pod <pod_name>`). Check the container logs for memory usage patterns (`kubectl logs <pod_name> -c <container_name>`).

Solution:

1. Increase the memory request and limit in the pod's resource specification to provide more memory for the container.
2. Optimize the application code to reduce memory usage and prevent memory leaks.
3. Use a memory profiling tool to identify and fix memory-intensive processes in the application.
4. Monitor the pod's memory usage over time and adjust the resource limits accordingly to prevent future OOMKilled errors.

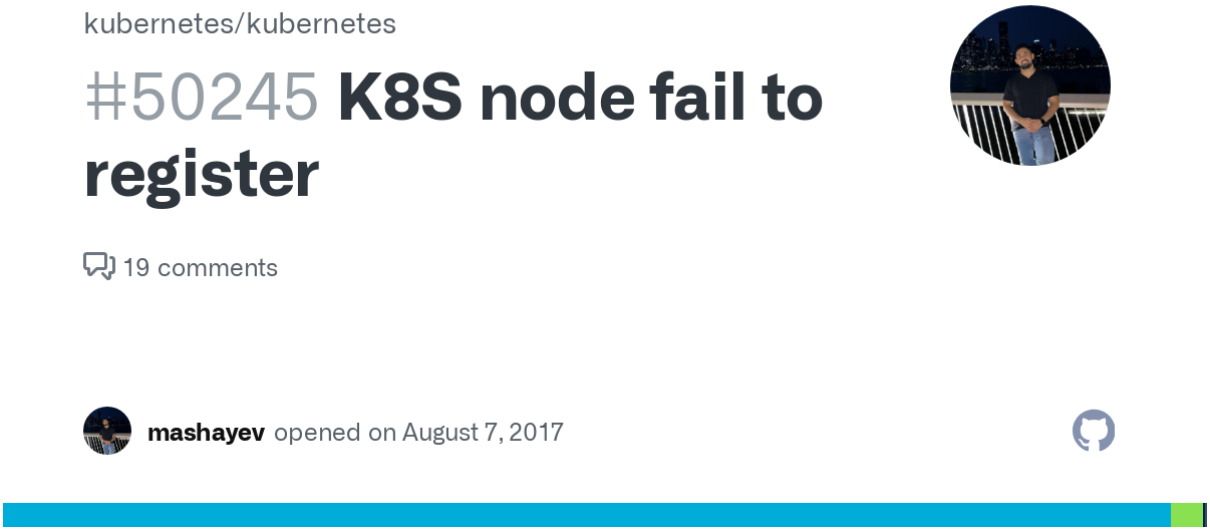
Scenario 64: Failed Node Registration

Symptoms: New nodes fail to register with the Kubernetes cluster, causing them to remain in a not ready state.

Diagnosis: Check the kubelet logs on the affected nodes for registration errors (`journalctl -u kubelet` or `kubectl logs <node_name> -n kube-system kubelet`). Inspect the cluster's API server logs for registration issues.

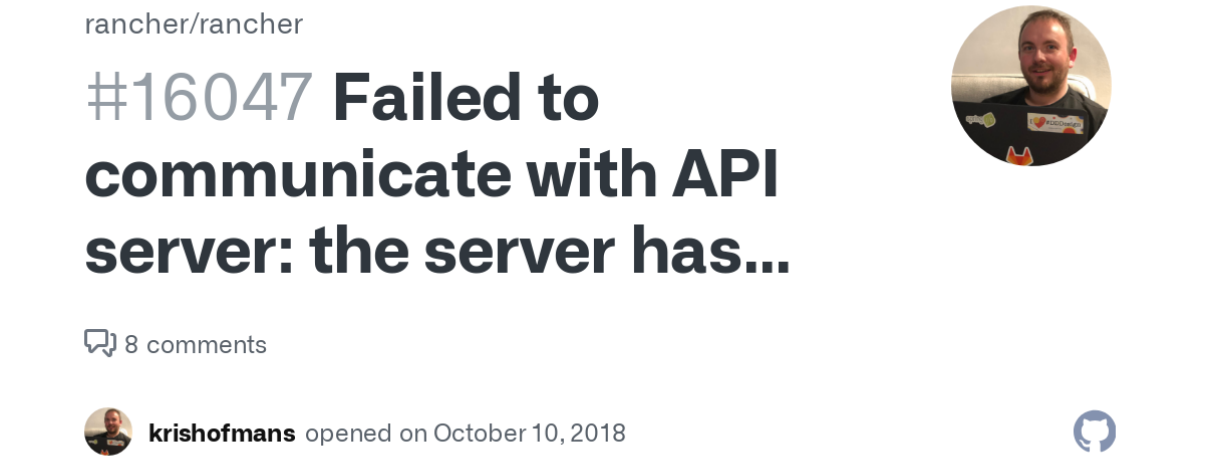
Solution:

1. Ensure that the kubelet configuration on the nodes is correct and includes the appropriate cluster join command.
2. Verify network connectivity between the nodes and the cluster's control plane.



3. Check for authentication or authorization issues that may prevent node registration and resolve them.
4. Restart the kubelet service on the affected nodes to retry the registration process.

Scenario 65: Failed API Server Requests



Symptoms: API server requests intermittently fail, causing disruptions in cluster management and operations.

Diagnosis: Inspect the API server logs (`kubectl logs <apiserver_pod> -n kube-system`) for error messages and check the cluster's control plane components for issues.

Solution:

1. Ensure that the API server is not overloaded and has sufficient resources to handle incoming requests.
2. Check for network connectivity issues between the API server and the cluster nodes.
3. Use API server auditing and logging to trace and diagnose failed requests.
4. Scale out the API server deployment to distribute the load and improve request

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.