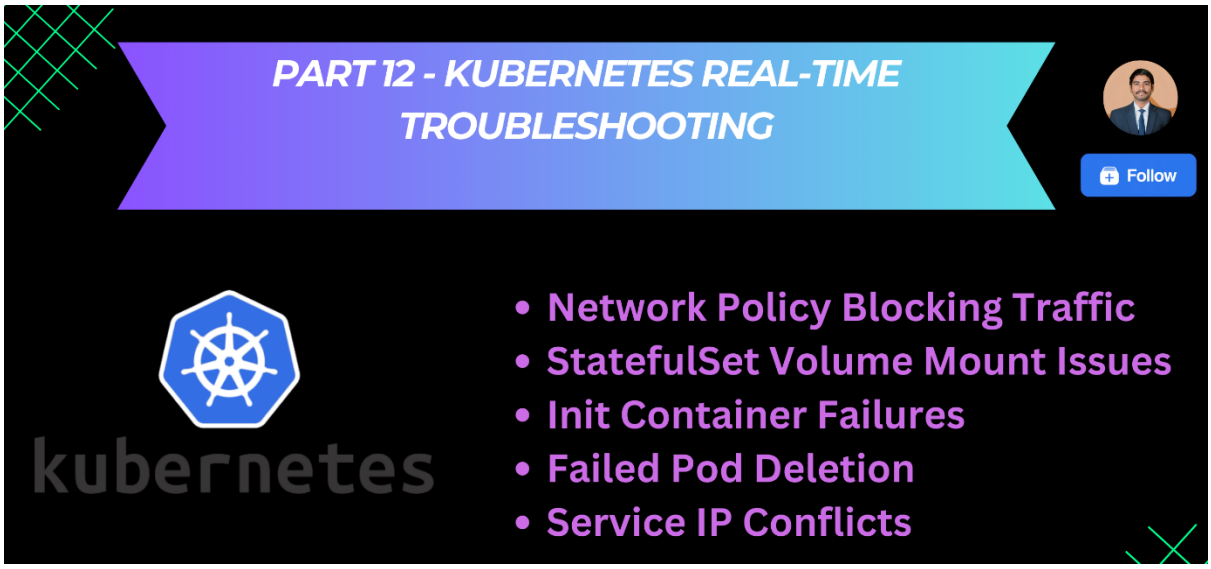
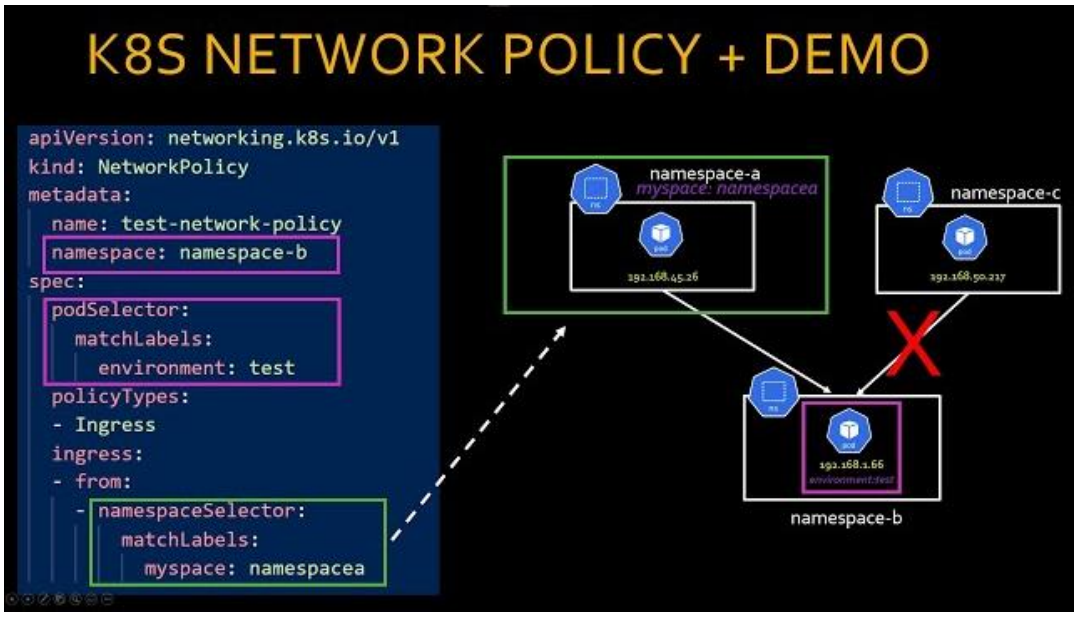
Part 12: Kubernetes Real-Time Troubleshooting

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 56: Network Policy Blocking Traffic



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

Symptoms: Applications cannot communicate with each other or external services due to network policies blocking traffic.

Diagnosis: Review the network policies (`kubectl get networkpolicy`, `kubectl describe networkpolicy <networkpolicy_name>`) and inspect the pod and service configurations for compliance.

Solution:

1. Verify that the network policies are correctly defined to allow the intended traffic flows between pods and services.
2. Adjust the network policy rules to permit necessary ingress and egress traffic while maintaining security constraints.
3. Use network policy logging and monitoring tools to trace and debug blocked traffic flows and identify necessary rule adjustments.
4. Test network connectivity between pods using network tools like `ping`, `curl`, or `netcat` to diagnose and resolve communication issues.

Scenario 57: StatefulSet Volume Mount Issues

Symptoms: Pods in a StatefulSet fail to mount volumes, causing application initialization failures.

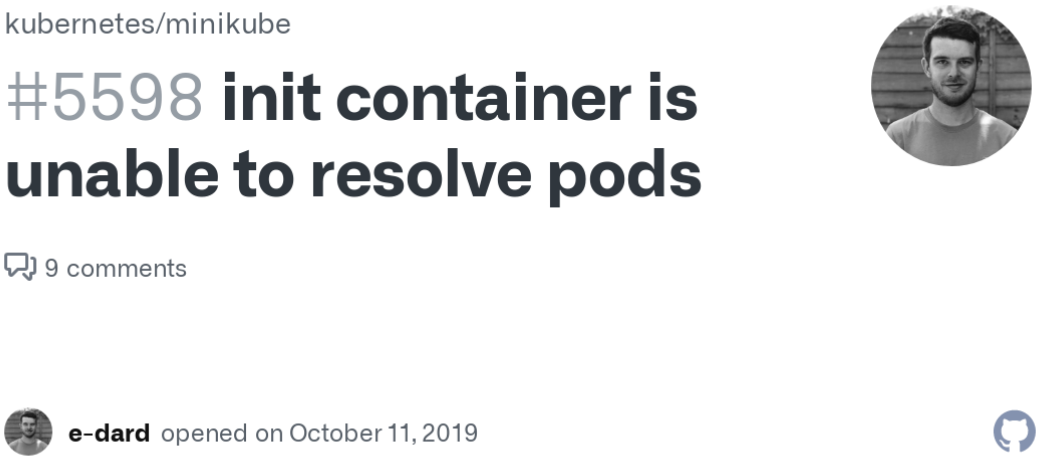
Diagnosis: Check the StatefulSet status and events (`kubectl get statefulset`, `kubectl describe statefulset <statefulset_name>`) and inspect the pod logs for volume mount errors.

Solution:

1. Verify that the Persistent Volume Claims (PVCs) referenced in the StatefulSet are correctly defined and bound to Persistent Volumes (PVs).

2. Ensure that the storage class specified in the PVCs supports the requested volume access modes and capacity.
3. Check for any permission or configuration issues with the underlying storage provider that may prevent volume mounting.
4. Restart the affected StatefulSet pods to trigger re-mounting of volumes and resolve transient issues.

Scenario 58: Init Container Failures



Symptoms: Pods fail to start because init containers fail to complete successfully, causing the main containers to remain in a waiting state.

Diagnosis: Describe the affected pods (`kubectl describe pod <pod_name>`) and inspect the logs of the failed init containers (`kubectl logs <pod_name> -c <init_container_name>`).

Solution:

1. Ensure that the init container image and commands are correctly defined and accessible.
2. Verify that the init containers have the necessary resources and permissions to complete their tasks.
3. Adjust the init container configuration to handle transient errors and implement retries or backoff mechanisms if necessary.
4. Use readiness and liveness probes to monitor the status and health of init containers and ensure they complete successfully before starting the main containers.

Scenario 59: Failed Pod Deletion (stuck in termination)

Symptoms: Pods remain in a terminating state indefinitely, preventing successful deletion and resource cleanup.

Diagnosis: Inspect the pod status and events (`kubectl get pod <pod_name>`, `kubectl describe pod <pod_name>`) to identify any blocking conditions or finalizer issues.

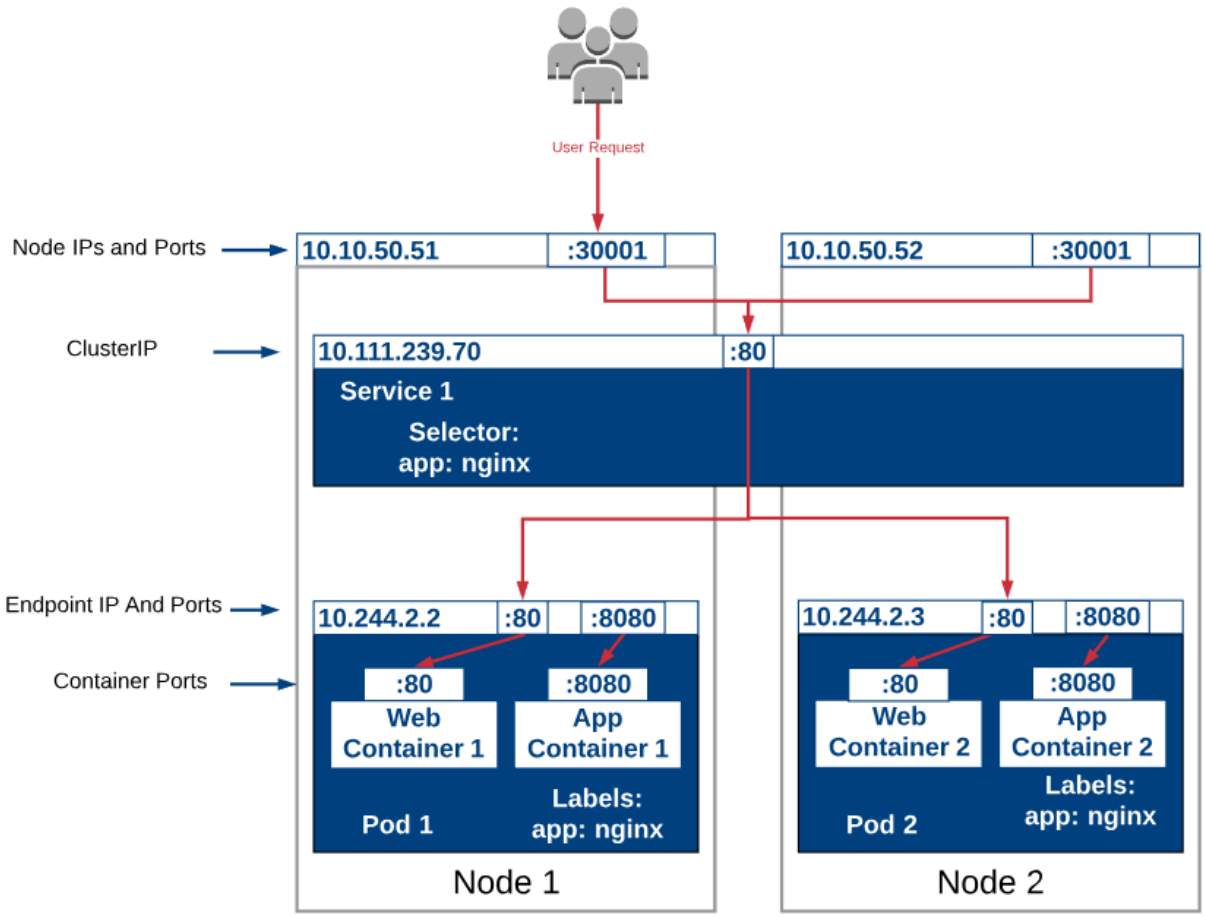
Solution:

1. Check for any finalizers attached to the pod that may prevent deletion and remove them if necessary (`kubectl patch pod <pod_name> -p '{"metadata":{"finalizers":[]}}'`).
2. Investigate any ongoing processes or resources (e.g., volume detachment, network connections) that may delay pod termination and address them.
3. Force delete the pod if necessary to bypass blocking conditions (`kubectl delete pod <pod_name> --force --grace-period=0`).
4. Monitor and manage pod lifecycle events to ensure timely cleanup and prevent resource leaks.

Scenario 60: Service IP Conflicts

Symptoms: Multiple services cannot communicate because they are assigned overlapping IP addresses.

Diagnosis: Inspect the service definitions (`kubectl get svc`) and describe the services (`kubectl describe svc <service_name>`) to identify overlapping IP addresses.



Solution:

1. Ensure that the service CIDR range is properly configured in the cluster's network configuration.
2. Reassign the conflicting services with unique IP addresses or modify the service definitions to use different ports.
3. Check the cluster's DNS settings and ensure there are no conflicts in DNS records.
4. Consider using external DNS or load balancer solutions to avoid internal IP conflicts.

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.