Resolving Network Errors in a Clustered Environment

📌 Summary

Network errors in a clustered environment can be caused by misconfigurations, firewall issues, DNS failures, load balancer errors, or service-to-service communication problems. This document provides **step-by-step troubleshooting** guidelines to identify, diagnose, and resolve such network issues efficiently.

🔍 Steps to Resolve Network Errors

1️⃣ Check Network Connectivity & Node Availability

🔷 **Command to Run:**

```
kubectl get nodes -o wide
ping <node-ip>
```

🔷 **Expected Output:**

- All nodes should be in a `Ready` state.
- Ping should return a response from the destination node.

🔷 **Actions If Issue Exists:**

- If a node is `NotReady`, check `kubectl describe node <node-name>`.
- If ping fails, verify network routes and firewall settings.

2️⃣ Verify Pod & Service Communication

🔹 Command to Check Pods & Services:

```
kubectl get pods -o wide
kubectl get svc -o wide
```

🔹 Expected Output:

- Pods should be running and in `Running` state.
- Services should be exposing the correct ports.

🔹 Actions If Issue Exists:

- Use `kubectl logs <pod-name>` to check for errors.
- Restart affected pods: `kubectl delete pod <pod-name>`.

3️⃣ Test Internal Cluster DNS Resolution

🔹 Command to Run DNS Check:

```
kubectl exec -it <pod-name> -- nslookup <service-name>
```

🔹 Expected Output:

- DNS should resolve to the correct service IP.

🔹 Actions If Issue Exists:

- Restart CoreDNS: `kubectl rollout restart deployment coredns -n kube-system`.
- Check logs for DNS failures: `kubectl logs -n kube-system -l k8s-app=kube-dns`.

4️⃣ Check Network Policies & Firewalls

🔹 **Command to List Network Policies:**

```
kubectl get networkpolicy -A
```

🔹 **Expected Output:**

- Policies should allow traffic between necessary services.

🔹 **Actions If Issue Exists:**

- Modify network policies to allow required communication.
- Check firewall rules on cloud provider (AWS Security Groups, Azure NSG, GCP Firewall).

5️⃣ Validate Load Balancer & Ingress Configuration

🔹 **Command to Check Load Balancer & Ingress:**

```
kubectl get ingress -o wide
kubectl get services -o wide | grep LoadBalancer
```

🔹 **Expected Output:**

- LoadBalancer should have an external IP assigned.
- Ingress should route traffic correctly.

🔹 **Actions If Issue Exists:**

- Verify ingress controller logs: `kubectl logs -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx`.
- Check cloud provider's load balancer status.

6️⃣ Debug with Network Tracing & Logs

🔹 **Commands to Trace Packets:**

```
kubectl exec -it <pod-name> -- tcpdump -i eth0 port <service-port>
kubectl logs <pod-name> --previous
```

🔹 **Expected Output:**

- TCP traffic should flow as expected.
- Logs should not contain `connection refused` or `timeout` errors.

🔹 **Actions If Issue Exists:**

- Use `kubectl port-forward` to test service access.
- Increase logging verbosity for better insights.

✅ Conclusion

By following these steps, you can systematically **identify and resolve network issues in a Kubernetes or clustered environment**. Proper monitoring, logging, and security configurations help prevent recurring issues.

For persistent problems, consider **network observability tools like Istio, Cilium, or Calico** to gain deeper insights into network traffic.