How to fix Kubernetes Node Not Ready error_

https://komodor.com/learn/how-to-fix-kubernetes-node-not-ready-error/#:~:text=Common%20reasons%20for%20a%20Kubernetes,networking%20agent%20on%20the%20node).

What is the Kubernetes Node Not Ready Error?

A Kubernetes node is a physical or virtual machine participating in a Kubernetes cluster, which can be used to run pods. When a node shuts down or crashes, it enters the NotReady state, meaning it cannot be used to run pods. All stateful pods running on the node then become unavailable.

Common reasons for a Kubernetes `node not ready` error include lack of resources on the node, a problem with the kubelet (the agent enabling the Kubernetes control plane to access and control the node), or an error related to kube-proxy (the networking agent on the node).

To identify a Kubernetes node not ready error: run the kubectl get nodes command. Nodes that are not ready will appear like this:

```
NAME                  STATUS    ROLES   AGE      VERSION
master.example.com    Ready     master  5h       v1.17
node1.example.com     NotReady  compute 5h       v1.17
node2.example.com     Ready     compute 5h       v1.17
```

We'll provide best practices for diagnosing simple cases of the `node not ready` error, but more complex cases will require advanced diagnosis and troubleshooting, which is beyond the scope of this article.

The 4 Kubernetes Node States

At any given time, a Kubernetes node can be in one of the following states:

- **Ready**—able to run pods.
- **NotReady**—not operating due to a problem, and cannot run pods.
- **SchedulingDisabled**—the node is healthy but has been marked by the cluster as not schedulable.
- **Unknown**—if the node controller cannot communicate with the node, it waits a default of 40 seconds, and then sets the node status to unknown.

If a note is in the `NodeReady` state, it indicates that the kubelet is installed on the node, but Kubernetes has detected a problem on the node that prevents it from running pods.

Troubleshooting Node Not Ready Error

Common Causes and Diagnosis

Here are some common reasons that a Kubernetes node may enter the `NotRead` state:

Lack of System Resources

Why It Prevents the Node from Running Pods
A node must have enough disk space, memory, and processing power to run Kubernetes workloads.

If non-Kubernetes processes on the node are taking up too many resources, or if there are too many processes running on the node, it can be marked by the control plane as `NotReady`.

How to Diagnose
Run `kubectl describe node` and look in the Conditions section to see if resources are missing on the node:

MemoryPressure—node is running out of memory.
DiskPressure—node is running out of disk space.
PIDPressure—node is running too many processes.

kubelet Issue

Why It Prevents the Node from Running Pods
The kubelet must run on each node to enable it to participate in the cluster. If the kubelet crashes or stops on a node, it cannot communicate with the API server and the node goes into a not ready state.

How to Diagnose
Run `kubectl describe node [name]` and look in the Conditions section—if all the conditions are unknown, this indicates the kubelet is down.

kube-proxy Issue

Why It Prevents the Node from Running Pods
kube-proxy runs on every node and is responsible for regulating network traffic between the node and other entities inside and outside the cluster. If kube-proxy stops running for any reason, the node goes into a not ready state.

How to Diagnose
Run `kubectl get pods -n kube-system` to show pods belonging to the Kubernetes system.

Connectivity Issue

Why It Prevents the Node from Running Pods
Even if a node is configured perfectly, but it has no network connectivity, Kubernetes treats the node as not ready. This could be due to a disconnected network cable, no Internet access, or misconfigured networking on the machine.

How to Diagnose
Run `kubectl describe node [name]` and look in the Conditions section—if the `NetworkUnavailable` flag is `True`, this means the node has a connectivity issue.

Resolving Node Not Ready Issues

Resolving Lack of System Resources

Here are a few ways to resolve a system resource issue on the node:

- Identify which non-Kubernetes processes are running on the node. If there are any, shut them down or reduce them to a minimum to conserve resources.
- Run a malware scan—there may be hidden malicious processes taking up system resources.
- Upgrade the node.
- Check for hardware issues or misconfigurations and resolve them.

Resolving kubelet Issues

To resolve a kubelet issue, SSH into the node and run the command systemctl status kubelet

Look at the value of the Active field:

- `active (running)` means the kubelet is actually operational, look for the problem elsewhere.
- `active (exited)` means the kubelet was exited, probably in error. Restart it.>
- `inactive (dead)` means the kubelet crashed. To identify why, run the command `journalctl -u kubelet` and examine the kubelet logs.

Resolving kube-proxy Issues

Try looking in the following places to identify what is the issue with kube-proxy:

- Run the command `kubectl describe pod` using the name of the kube-proxy pod that failed, and check the Events section in the output.
- Run the command `kubectl logs [pod-name] -n kube-system` to see a full log of the failing kube-proxy pod.
- Run the command `kubectl describe daemonset kube-proxy -n kube-system` to see the status of the kube-proxy daemonset, which is responsible for ensuring there is a kube-proxy running on every Kubernetes node.

Please note that these procedures can help you gather more information about the problem, but additional steps may be needed to resolve the problem. If one of the quick fixes above did not work, you'll need to undertake a more complex, non-linear diagnosis procedure to identify which parts of the Kubernetes environment contribute to the node not ready problem and resolve it.

Solving Kubernetes Node Errors with Komodor

Kubernetes troubleshooting relies on the ability to quickly contextualize the problem with what's happening in the rest of the cluster. More often than not, you will be conducting your investigation during fires in production. The major challenge is correlating service-level incidents with other events happening in the underlying infrastructure.

Komodor can help with our 'Node Status' view, built to pinpoint correlations between service or deployment issues and changes in the underlying node infrastructure. With this view you can rapidly:

- See service-to-node associations
- Correlate service and node health issues
- Gain visibility over node capacity allocations, restrictions, and limitations
- Identify "noisy neighbors" that use up cluster resources
- Keep track of changes in managed clusters
- Get fast access to historical node-level event data

Beyond node error remediations, Komodor can help troubleshoot a variety of Kubernetes errors and issues, acting as a single source of truth (SSOT) for all of your K8s troubleshooting needs. Komodor provides:

- **Change intelligence**: Every issue is a result of a change. Within seconds we can help you understand exactly who did what and when.
- **In-depth visibility**: A complete activity timeline, showing all code and config changes, deployments, alerts, code diffs, pod logs and etc. All within one pane of glass with easy drill-down options.
- **Insights into service dependencies**: An easy way to understand cross-service changes and visualize their ripple effects across your entire system.
- **Seamless notifications**: Direct integration with your existing communication channels (e.g., Slack) so you'll have all the information you need, when you need it.