**TB-1   
A node is broken due to a hardware issue  
  
Situation**

* One of your Kubernetes worker nodes (physical or VM host) suffered a **hardware failure** (e.g., disk crash, CPU failure, RAM issue, NIC problem).
* As a result:
  + Node goes NotReady.
  + Pods on it might go into **Terminating** or **Unknown** state.
  + Scheduler can’t reschedule them properly (if resources are limited).

**✅ Step-by-Step Response**

**1. Identify the Failure**

1. Run:
2. kubectl get nodes

→ Look for the affected node (NotReady).

1. Describe the node for details:
2. kubectl describe node <node-name>

→ Check events for Kubelet stopped posting status, disk pressure, network unavailable, etc.

1. Verify hardware failure in infrastructure monitoring:
   * If bare-metal: check iDRAC/iLO/IPMI logs.
   * If cloud (AWS, GCP, Azure): check cloud provider event logs.
   * If VM-based: check hypervisor logs (VMware/KVM).

**2. Temporary / Immediate Solutions**

Goal → **Restore workloads quickly and reduce downtime**.

**(a) Cordon & Drain Node**

* Prevent new pods from scheduling on broken node:
* kubectl cordon <node-name>
* Evict existing pods safely:
* kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data

This forces pods to move if other nodes are available.

**(b) Reschedule Pods on Healthy Nodes**

* If other nodes have capacity → pods get rescheduled automatically.
* If insufficient capacity → **scale up cluster**:
  + **Cloud**: increase node pool size (eksctl, gcloud, az aks).
  + **On-prem**: manually add a spare node to cluster.

**(c) Temporary Workaround if No Spare Node Available**

* **Run pods manually** on another node:
* kubectl delete pod <pod-name> -n <namespace>

(Deployment/ReplicaSet will recreate it elsewhere if resources allow).

* **Taints & Tolerations**: Temporarily allow pods to tolerate other nodes if they’re restricted.
* **PriorityClass**: Ensure critical workloads have higher priority to reschedule before less critical ones.
* **Use Backup VM/Node**: Quickly spin up a temporary VM in cloud/on-prem and join it to cluster:
* kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

**3. Permanent Solutions**

Goal → **Fix root cause and prevent recurrence**.

**(a) Replace or Repair Hardware**

* Engage **hardware vendor/cloud provider** to replace faulty disk, RAM, or NIC.
* If bare-metal → perform RMA or replace faulty components.
* If VM host → migrate workloads off, repair, then return to pool.

**(b) Rebuild or Replace the Node**

* If node is unrecoverable:
  1. **Delete node from cluster**:
  2. kubectl delete node <node-name>
  3. Provision new node (bare-metal, VM, or cloud instance).
  4. Install dependencies (kubelet, kubeadm, container runtime).
  5. Rejoin node to cluster:
  6. kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

**(c) Improve Resilience (Long-term)**

1. **Cluster Autoscaling**
   * Configure Cluster Autoscaler (EKS/GKE/AKS) or Karpenter to automatically add nodes if capacity is insufficient.
2. **Pod Disruption Budgets (PDBs)**
   * Ensure high availability apps run with multiple replicas so a single node failure doesn’t cause downtime.
3. **Node Affinity / Anti-Affinity**
   * Spread workloads across multiple nodes/zones.
4. **Monitoring & Alerting**
   * Use Prometheus/Grafana or cloud monitoring to get alerts on failing hardware early.
5. **Regular Maintenance**
   * Patch/upgrade nodes in rolling fashion.
   * Use kubectl cordon + drain before maintenance.
6. **Storage Considerations**
   * Use network-attached storage (EBS, CSI drivers) so pods can be re-scheduled without data loss.