**ETCD Backup and Restore in Kubernetes**

**🎯 Step 1: Prep your cluster for backup**

Before touching ETCD, make sure your cluster is up and running.

* **Create some random pods** so you have resources that ETCD will persist.

bash

CopyEdit

kubectl run nginx --image=nginx

kubectl run redis --image=redis

kubectl get pods

This ensures there’s data in ETCD (pod specs, deployments, etc.) to validate backup & restore later.

**📦 Step 2: Install ETCD client**

Install etcdctl for interacting with ETCD.

bash

CopyEdit

apt update

apt install etcd-client -y

Set ETCD API version:

bash

CopyEdit

export ETCDCTL\_API=3

ETCD v3 API is used because Kubernetes 1.13+ clusters use ETCD v3 by default.

**🔍 Step 3: Find important paths**

We need ETCD’s certs & data directories to back up securely.

**a. Find kubelet config path**

bash

CopyEdit

ps -ef | grep kubelet

cd /var/lib/kubelet/

ls

cat config.yaml

Here you’ll see kubelet arguments, but for ETCD specifically, check static pods:

bash

CopyEdit

ps -ef | grep etcd

Look for flags like --data-dir=/var/lib/etcd and certs in /etc/kubernetes/pki/etcd.

**💾 Step 4: Backup ETCD**

Run snapshot save with cert-based authentication:

bash

CopyEdit

etcdctl --endpoints=https://127.0.0.1:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key \

snapshot save /root/myclust.db

✔ **What happens here?**

* Connects to ETCD on https://127.0.0.1:2379
* Authenticates with Kubernetes-generated TLS certs
* Saves a **point-in-time snapshot** as /root/myclust.db.

✅ Verify snapshot:

bash

CopyEdit

etcdctl snapshot status /root/myclust.db

**🔥 Step 5: Simulate disaster**

Delete all pods to simulate data loss:

bash

CopyEdit

kubectl delete pods --all

kubectl get pods

ETCD has lost data if you restart it now.

**🛠 Step 6: Restore ETCD snapshot**

1. First, check original data dir:

bash

CopyEdit

ps -ef | grep etcd

Example:

ini

CopyEdit

--data-dir=/var/lib/etcd

1. **Restore snapshot into a new directory** (DO NOT create it manually):

bash

CopyEdit

etcdctl --data-dir /var/lib/etcd-new snapshot restore /root/myclust.db

✔ This creates /var/lib/etcd-new/member/....

**✍ Step 7: Update ETCD manifest**

Edit ETCD static pod manifest to use new data-dir:

bash

CopyEdit

vi /etc/kubernetes/manifests/etcd.yaml

Find the volumeMounts and volumes sections. Update them:

yaml

CopyEdit

volumeMounts:

- mountPath: /var/lib/etcd

name: etcd-data

volumes:

- hostPath:

path: /var/lib/etcd-new

type: DirectoryOrCreate

name: etcd-data

Save & exit. Kubelet will detect this change and restart ETCD using /var/lib/etcd-new.

**⏳ Step 8: Wait for pods to return**

Give it a minute or two for control plane components to reconcile.

Check pods:

bash

CopyEdit

kubectl get pods --all-namespaces

If successful, your previously deleted pods should reappear because the restored ETCD data has all objects.

**🧠 What’s Happening Behind the Scenes?**

* Kubernetes stores **all cluster state** (pods, services, configmaps, etc.) in ETCD.
* snapshot save creates a consistent copy of ETCD’s data tree.
* When restoring, you spin up ETCD pointing to the restored dataset.
* Kubelet watches /etc/kubernetes/manifests for static pods like ETCD, API server, etc. Any change here triggers a container restart.

**🛡 Best Practices**

✅ Take periodic ETCD backups via cronjob on control plane.  
✅ Always restore in a **new directory** to avoid overwriting existing data.  
✅ Keep certs safe: Without them, you can’t access ETCD.  
✅ Automate snapshot storage to an S3 bucket or external store.

**✅ Cluster Upgrade Process (Kubernetes v1.30 using kubeadm)**

A **kubeadm upgrade** requires you to **first upgrade control plane nodes (master)** and then upgrade the **worker nodes**.

**⚙️ Part 1: Upgrade Master Node**

This is the most critical phase because all cluster state is managed by the control plane.

**🔸 Step 1: Add Kubernetes v1.30 APT Repo**

bash

CopyEdit

sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF

deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /

EOF

* **Why?** Kubernetes v1.30 has switched package distribution to pkgs.k8s.io (earlier it was apt.kubernetes.io).
* **Keyring**: Ensure /etc/apt/keyrings/kubernetes-apt-keyring.gpg exists; else, fetch it.

**🔸 Step 2: Upgrade kubeadm binary**

bash

CopyEdit

sudo apt-mark unhold kubeadm && \

sudo apt-get update && sudo apt-get install -y kubeadm='1.30.0-1.1' && \

sudo apt-mark hold kubeadm

* **Why only kubeadm first?**
  + kubeadm manages the upgrade plan and migration steps.
  + Upgrade kubeadm first before touching the API server or kubelet.

**🔸 Step 3: Plan the Upgrade**

bash

CopyEdit

kubeadm upgrade plan --ignore-preflight-errors=CoreDNSUnsupportedPlugins,CoreDNSMigration

* This validates:
  + The current cluster version.
  + Available upgrade paths.
  + Required CoreDNS and etcd changes.

✅ **Ignore CoreDNS preflight errors** if using unsupported plugins or custom CoreDNS config.

**🔸 Step 4: Apply the Upgrade**

bash

CopyEdit

kubeadm upgrade apply v1.30.0 --ignore-preflight-errors=CoreDNSUnsupportedPlugins,CoreDNSMigration

* **What happens here?**
  + Downloads the new container images for kube-apiserver, kube-controller-manager, kube-scheduler, etcd.
  + Upgrades kube-apiserver and other static pods in /etc/kubernetes/manifests.
  + Updates the cluster state in etcd.

**🔸 Step 5: Upgrade kubelet and kubectl**

bash

CopyEdit

sudo apt-mark unhold kubelet kubectl && \

sudo apt-get update && sudo apt-get install -y kubelet='1.30.0-1.1' kubectl='1.30.0-1.1' && \

sudo apt-mark hold kubelet kubectl

sudo systemctl daemon-reload

sudo systemctl restart kubelet

* Upgrade kubelet (agent running on master) and kubectl.
* Restart kubelet so it picks up the new API version.

**🟢 Validation**

bash

CopyEdit

kubectl get nodes

✅ The master node should now report v1.30.0.

**⚙️ Part 2: Upgrade Worker Nodes**

Each worker is upgraded one at a time to avoid downtime.

**🔸 Step 1: Add Kubernetes v1.30 APT Repo**

On worker node (e.g., w1):

bash

CopyEdit

sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF

deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /

EOF

**🔸 Step 2: Upgrade kubeadm binary**

bash

CopyEdit

sudo apt-mark unhold kubeadm && \

sudo apt-get update && sudo apt-get install -y kubeadm='1.30.0-1.1' && \

sudo apt-mark hold kubeadm

**🔸 Step 3: Drain the Node**

On master node:

bash

CopyEdit

kubectl drain w1 --ignore-daemonsets

* **Why?**
  + Evicts all workloads (Pods) from w1.
  + Ignores DaemonSets so system pods (like kube-proxy, fluentd) keep running.

**🔸 Step 4: Upgrade Node Components**

On worker w1:

bash

CopyEdit

sudo kubeadm upgrade node

* Updates the kubelet configuration to v1.30.

**🔸 Step 5: Upgrade kubelet and kubectl**

bash

CopyEdit

sudo apt-mark unhold kubelet kubectl && \

sudo apt-get update && sudo apt-get install -y kubelet='1.30.0-1.1' kubectl='1.30.0-1.1' && \

sudo apt-mark hold kubelet kubectl

sudo systemctl daemon-reload

sudo systemctl restart kubelet

**🔸 Step 6: Uncordon the Node**

On master node:

bash

CopyEdit

kubectl uncordon w1

* **Why?**
  + Puts w1 back into the scheduler so it can accept workloads.

**🟢 Validation**

bash

CopyEdit

kubectl get nodes

✅ Node w1 should now report v1.30.0.

**📊 Result After Upgrade**

text

CopyEdit

NAME STATUS ROLES AGE VERSION

master Ready control-plane 3d23h v1.30.0

w1 Ready <none> 3d22h v1.30.0

w2 Ready <none> 3d22h v1.29.15

**⚠️ Key Notes & Best Practices**

✅ **1. Upgrade one worker at a time**

* To ensure workloads can reschedule properly.

✅ **2. Backup etcd before control-plane upgrade**

bash

CopyEdit

ETCDCTL\_API=3 etcdctl snapshot save /tmp/etcd-backup.db \

--endpoints=https://127.0.0.1:2379 \

--cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt \

--key=/etc/kubernetes/pki/etcd/server.key

✅ **3. Validate cluster health post-upgrade**

bash

CopyEdit

kubectl get cs # Deprecated in v1.24+, use below

kubectl get componentstatuses

kubectl get pods -A

✅ **4. CoreDNS upgrade**

* If CoreDNS migration fails, manually edit coredns ConfigMap.

✅ **5. Review deprecations**

* Check v1.30 changelog for API removals.

**Prom-Kube-grafana setup :**

**✅ Phase 1: Install Helm**

bash

CopyEdit

curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

chmod 700 get\_helm.sh

./get\_helm.sh

* You download the official Helm v3 install script, make it executable, and run it.
* Validation:

bash

CopyEdit

helm version

Ensure it prints something like version.BuildInfo{Version:"v3.x.x"}.

**✅ Phase 2: Add Prometheus Helm Repo**

bash

CopyEdit

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

* This adds the upstream repo where **kube-prometheus-stack** is maintained.
* Search for available charts and versions:

bash

CopyEdit

helm search repo prometheus-community/kube-prometheus-stack --versions

**✅ Phase 3: Deploy kube-prometheus-stack**

bash

CopyEdit

helm install prometheus prometheus-community/kube-prometheus-stack \

--version 45.7.1 \

--namespace monitoring \

--create-namespace \

--set kubeEtcd.enabled=false

* This installs Prometheus, Alertmanager, Grafana, and exporters as one bundled stack.
* Disabling kubeEtcd.enabled=false skips scraping etcd metrics if etcd is not exposed.

Check deployments:

bash

CopyEdit

helm list -n monitoring

kubectl get pods -n monitoring

**✅ Phase 4: Expose Prometheus UI**

bash

CopyEdit

kubectl edit svc -n monitoring prometheus-operated

* Change the service from ClusterIP to NodePort.
* Remove:

yaml

CopyEdit

clusterIP: None

clusterIPs:

- None

* Save and apply:

bash

CopyEdit

kubectl replace --force -f /tmp/kubectl-edit-\*.yaml

* Get the NodePort assigned:

bash

CopyEdit

kubectl get svc -n monitoring

* Example URL: http://<NodeIP>:<NodePort> → e.g., http://56.125.5.201:31585

**✅ Phase 5: Expose Grafana Dashboard**

bash

CopyEdit

kubectl edit svc -n monitoring prometheus-grafana

* Set type NodePort, get the port:

bash

CopyEdit

kubectl get svc -n monitoring

* Decode Grafana admin password:

bash

CopyEdit

kubectl get secret -n monitoring prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 --decode

* Default credentials:
  + **Username**: admin
  + **Password**: prom-operator (or decoded one)
* Access URL: http://<NodeIP>:<NodePort>

**✅ Phase 6: Test Prometheus Queries**

In Prometheus UI → use **PromQL** queries:

* Running pods:

promql

CopyEdit

kube\_pod\_container\_status\_running

* CPU utilization of a node:

promql

CopyEdit

100 \* (1 - avg(rate(node\_cpu\_seconds\_total{mode="idle", instance="172.31.27.229:9100"}[5m])))

**✅ Phase 7: Enhance Grafana Dashboards**

Install curated dashboards:

* [dotdc Kubernetes dashboards](https://github.com/dotdc/grafana-dashboards-kubernetes)
* Import dashboards into Grafana:
  + Kubernetes Views / Nodes
  + Dashboard ID: **15759** (modern node-level metrics)

**✅ Phase 8: Create Alerts in Grafana**

1. Go to:

nginx

CopyEdit

Dashboards → Kubernetes → Views → Nodes

1. Find the **CPU Utilization** panel, click:

pgsql

CopyEdit

More → Create new alert rule

1. Alert details:
   * **Query (A)**:

promql

CopyEdit

100 \* (1 - avg(rate(node\_cpu\_seconds\_total{mode="idle", instance="<node-ip>:9100"}[5m])))

* + Time range: 5m
  + Pending period: 1m
  + Test with preview.

1. Save & enable alerting.

**🚀 What is a Kubernetes CronJob?**

A **CronJob** in Kubernetes is like the Linux cron utility. It runs **jobs** on a **schedule**. Each **Job** it creates will run a Pod and ensure the task completes successfully.

* 🔥 **Key Use Case**: Repeated tasks (backups, cleanups, report generation) that need to run on a specific schedule.
* ✅ Automatically creates **Jobs**, which run **Pods** to perform the actual work.
* 🕒 Follows **cron schedule syntax**: \* \* \* \* \*

**✅ Real-World Scenario**

Let’s say:  
**"We want to create a CronJob to clean up old log files from a shared storage directory every day at midnight."**

**Directory**: /var/log/myapp  
**Clean-up Rule**: Delete all .log files older than 7 days.

This is a **common pattern** in microservices where Pods dump logs to a shared PersistentVolume, and you need automated cleanup to avoid filling the disk.

**🛠️ Step-by-Step Kubernetes CronJob**

**1️⃣ YAML for CronJob**

yaml

CopyEdit

apiVersion: batch/v1

kind: CronJob

metadata:

name: log-cleanup-cronjob

spec:

schedule: "0 0 \* \* \*" # Run every day at midnight

jobTemplate:

spec:

template:

spec:

containers:

- name: cleanup

image: alpine:3.19

command:

- /bin/sh

- -c

- |

echo "Starting log cleanup..."

find /var/log/myapp -name "\*.log" -type f -mtime +7 -exec rm -f {} \;

echo "Cleanup completed at $(date)"

volumeMounts:

- name: logs-volume

mountPath: /var/log/myapp

restartPolicy: OnFailure

volumes:

- name: logs-volume

persistentVolumeClaim:

claimName: myapp-logs-pvc

**📖 Explanation of the YAML**

| **Field** | **What it Does** |
| --- | --- |
| schedule: "0 0 \* \* \*" | Cron format. Midnight every day. |
| jobTemplate.spec.template | Defines the Pod to be run for the job. |
| containers.image: alpine | Uses a lightweight container (Alpine Linux) to run shell commands. |
| command | Runs a shell script to delete .log files older than 7 days. |
| volumeMounts | Mounts the directory where logs are stored. |
| restartPolicy: OnFailure | If the Pod fails, it will restart (but only for this job run). |
| persistentVolumeClaim | Mounts the PVC (myapp-logs-pvc) so the CronJob can access shared logs. |

**🕵️‍♂️ Real Execution Flow**

1. At midnight, the CronJob **creates a Job**.
2. The Job **spins up a Pod** with the Alpine container.
3. Inside the Pod:
   * The find command cleans up logs.
   * Logs are deleted from the mounted PersistentVolume.
4. The Pod exits successfully, and Kubernetes cleans it up (unless configured to keep history).
5. Kubernetes **keeps track of success/failure**.

**🎯 Use Cases for CronJobs**

✅ **Database Backups** – Export and store DB dumps in S3 or GCS.  
✅ **Log Rotation/Cleanup** – Clean up logs, cache directories.  
✅ **Periodic API Calls** – Call a health-check or metrics endpoint every hour.  
✅ **Sync Tasks** – Sync data to/from external services.  
✅ **Email/Report Generation** – Create and send daily/weekly reports.

**📝 A Real Database Backup Example**

Backup a Postgres database every 6 hours to an S3 bucket:

yaml

CopyEdit

apiVersion: batch/v1

kind: CronJob

metadata:

name: pg-backup

spec:

schedule: "0 \*/6 \* \* \*" # Every 6 hours

jobTemplate:

spec:

template:

spec:

containers:

- name: pg-dump

image: postgres:16

env:

- name: PGPASSWORD

valueFrom:

secretKeyRef:

name: pg-secret

key: password

command:

- /bin/sh

- -c

- |

pg\_dump -h postgres-service -U myuser mydb > /backup/mydb-$(date +%F-%H%M).sql

aws s3 cp /backup/mydb-$(date +%F-%H%M).sql s3://mybucket/db-backups/

volumeMounts:

- name: backup-volume

mountPath: /backup

restartPolicy: OnFailure

volumes:

- name: backup-volume

emptyDir: {}

**⚠️ Things to Watch Out For**

* **History Limits**:
  + successfulJobsHistoryLimit: How many successful jobs to keep.
  + failedJobsHistoryLimit: How many failed jobs to retain.
* **Overlapping Jobs**:  
  Set concurrencyPolicy: Forbid to prevent overlapping runs.
* **Time Zone Awareness**:  
  Kubernetes uses the cluster’s timezone.

**🔥 Lab Idea for Training (Hands-On)**

1. Create a PVC and deploy an **nginx pod** writing access logs to it.
2. Write a **CronJob** to delete logs older than 2 minutes.
3. Scale it up with concurrencyPolicy: Replace and show how overlapping jobs behave.
4. Query CronJob status:

bash

CopyEdit

kubectl get cronjob

kubectl describe cronjob log-cleanup-cronjob

1. Watch Job history:

bash

CopyEdit

kubectl get jobs --watch

**🟣 1. Troubleshooting ETCD Failure**

**ETCD** is the **key-value store** for Kubernetes cluster state. If it fails, the cluster becomes unusable.

**🛑 Symptoms:**

* kubectl commands hang or return errors like:

pgsql

CopyEdit

The connection to the server <master-ip>:6443 was refused

* Control plane components (API Server, Scheduler, Controller Manager) can't read/write cluster state.
* Logs show etcdserver: request timed out.

**🧑‍🔧 Troubleshooting Steps:**

✅ **Check ETCD pod/service health** (if running as static pod):

bash

CopyEdit

kubectl -n kube-system get pods | grep etcd

kubectl -n kube-system describe pod etcd-<node>

✅ **Check ETCD process directly (if running as systemd service):**

bash

CopyEdit

systemctl status etcd

journalctl -u etcd -f

✅ **Check ETCD endpoints:**

bash

CopyEdit

ETCDCTL\_API=3 etcdctl --endpoints=<endpoint> endpoint health

✅ **Check disk space and data directory:**

bash

CopyEdit

df -h

du -sh /var/lib/etcd

✅ **Check certificate validity (TLS issues):**

bash

CopyEdit

openssl x509 -in /etc/kubernetes/pki/etcd/server.crt -text -noout

**⚙️ Common Root Causes:**

* Disk full on /var/lib/etcd
* ETCD data corruption
* Expired TLS certs
* Network partition between ETCD nodes (in HA)

**🩺 Fix:**

* Restore ETCD from backup:

bash

CopyEdit

ETCDCTL\_API=3 etcdctl snapshot restore <snapshot-file> --data-dir /var/lib/etcd

* Replace expired certs.
* Ensure sufficient disk space and defragment:

bash

CopyEdit

ETCDCTL\_API=3 etcdctl defrag

**🟣 2. Troubleshooting Kubelet Failure**

The **Kubelet** runs on every node and manages pods.

**🛑 Symptoms:**

* Node shows as NotReady in kubectl get nodes
* Pods on that node are stuck in ContainerCreating or Unknown
* API server can't communicate with the node

**🧑‍🔧 Troubleshooting Steps:**

✅ **Check Kubelet service status:**

bash

CopyEdit

systemctl status kubelet

journalctl -u kubelet -f

✅ **Check node status:**

bash

CopyEdit

kubectl describe node <node-name>

✅ **Check kubelet config:**

bash

CopyEdit

cat /var/lib/kubelet/config.yaml

✅ **Validate certificate:**

bash

CopyEdit

openssl x509 -in /var/lib/kubelet/pki/kubelet-client-current.pem -noout -dates

✅ **Check logs for error patterns:**

* "x509: certificate has expired"
* "failed to get node info"

**⚙️ Common Root Causes:**

* Misconfigured kubelet config (config.yaml)
* Expired kubelet client certs
* Incorrect --cgroup-driver mismatch between Docker/Containerd and kubelet
* Disk pressure on node (check df -h, free -m)

**🩺 Fix:**

* Restart kubelet:

bash

CopyEdit

systemctl daemon-reload

systemctl restart kubelet

* Renew certs if expired.
* Fix cgroup driver mismatch.

**🟣 3. Troubleshooting Container Runtime Failure**

The container runtime (like Docker or containerd) runs the actual containers.

**🛑 Symptoms:**

* Pods stuck in ContainerCreating
* kubectl describe pod <pod> shows errors like:

pgsql

CopyEdit

Failed to create sandbox

* API server and kubelet are working, but containers don’t start.

**🧑‍🔧 Troubleshooting Steps:**

✅ **Check container runtime service:**  
For containerd:

bash

CopyEdit

systemctl status containerd

journalctl -u containerd -f

For Docker:

bash

CopyEdit

systemctl status docker

journalctl -u docker -f

✅ **Verify container runtime version:**

bash

CopyEdit

containerd --version

docker --version

✅ **Check runtime connectivity:**

bash

CopyEdit

ctr version

docker info

✅ **Check system resources:**

bash

CopyEdit

df -h

free -m

✅ **Inspect pod events:**

bash

CopyEdit

kubectl describe pod <pod>

**⚙️ Common Root Causes:**

* Container runtime service stopped
* CNI plugin errors
* Corrupted images
* Disk full on /var/lib/containerd or /var/lib/docker

**🩺 Fix:**

* Restart container runtime:

bash

CopyEdit

systemctl restart containerd

# or

systemctl restart docker

* Prune unused images:

bash

CopyEdit

docker system prune -a

* Check/repair CNI config in /etc/cni/net.d

**🟣 4. Troubleshooting Scheduler Failure**

The **kube-scheduler** decides which node a pod should run on.

**🛑 Symptoms:**

* Pods stuck in Pending
* kubectl describe pod shows no node assigned
* API server works fine, but new pods aren’t scheduled

**🧑‍🔧 Troubleshooting Steps:**

✅ **Check kube-scheduler pod (if static pod):**

bash

CopyEdit

kubectl -n kube-system get pods | grep scheduler

kubectl -n kube-system describe pod kube-scheduler-<node>

✅ **Check scheduler process:**

bash

CopyEdit

ps aux | grep kube-scheduler

✅ **Check scheduler logs:**

bash

CopyEdit

journalctl -u kube-scheduler -f

# Or inspect container logs:

kubectl -n kube-system logs kube-scheduler-<node>

✅ **Check taints on nodes:**

bash

CopyEdit

kubectl describe node <node>

✅ **Check pending pods:**

bash

CopyEdit

kubectl get pods --all-namespaces --field-selector=status.phase=Pending

✅ **Simulate scheduling with kubectl describe:**  
Look for reasons like:

* No nodes match pod's resource requests
* Node taints not tolerated

**⚙️ Common Root Causes:**

* Scheduler pod crashed
* NodeSelector/taints preventing scheduling
* Resource constraints (no node can satisfy pod requests)
* Misconfigured kube-scheduler.yaml

**🩺 Fix:**

* Restart kube-scheduler:

bash

CopyEdit

systemctl restart kube-scheduler

* Fix pod spec or node taints.
* Scale up cluster if resource shortage.

**✅ Summary Table**

| **Component** | **Symptoms** | **Root Causes** | **Key Fixes** |
| --- | --- | --- | --- |
| **ETCD** | API unresponsive | Disk full, TLS expiry, data corruption | Restore from snapshot, defrag, renew certs |
| **Kubelet** | Node NotReady | Cert expiry, config error, cgroup mismatch | Restart kubelet, fix config, renew certs |
| **Container Runtime** | Pods stuck ContainerCreating | Runtime crashed, CNI error, disk full | Restart runtime, prune images, check CNI |
| **Scheduler** | Pods stuck Pending | Scheduler down, taints/tolerations issues | Restart scheduler, fix taints, scale nodes |