

Mastering Kubernetes Operations

A comprehensive guide to upgrading clusters, protecting your data, monitoring system health, and troubleshooting like a pro



Chapter 1: Upgrading Your Kubernetes Cluster



Why Cluster Upgrades Matter

Security Patches

Each Kubernetes release includes critical security fixes that protect your cluster from vulnerabilities. Staying current means staying secure against the latest threats discovered by the community.

New Features

Access to cutting-edge capabilities like improved scheduling algorithms, enhanced resource management, and better API functionality that can streamline your operations.

Community Support

The Kubernetes community actively maintains only the three most recent minor versions. Running older versions means losing access to bug fixes and community assistance.

Pre-Upgrade Planning Checklist

Before touching any production cluster, thorough preparation prevents disasters. Think of this like planning a cross-country road trip—you wouldn't leave without checking your car, mapping your route, and having a backup plan.

1

Review Release Notes

Check for breaking changes, deprecated APIs, and new requirements in the target version

2

Test in Non-Production

Run the upgrade process in staging environments to identify issues before production

3

Backup Everything

Create ETCD snapshots and document current configurations for quick rollback

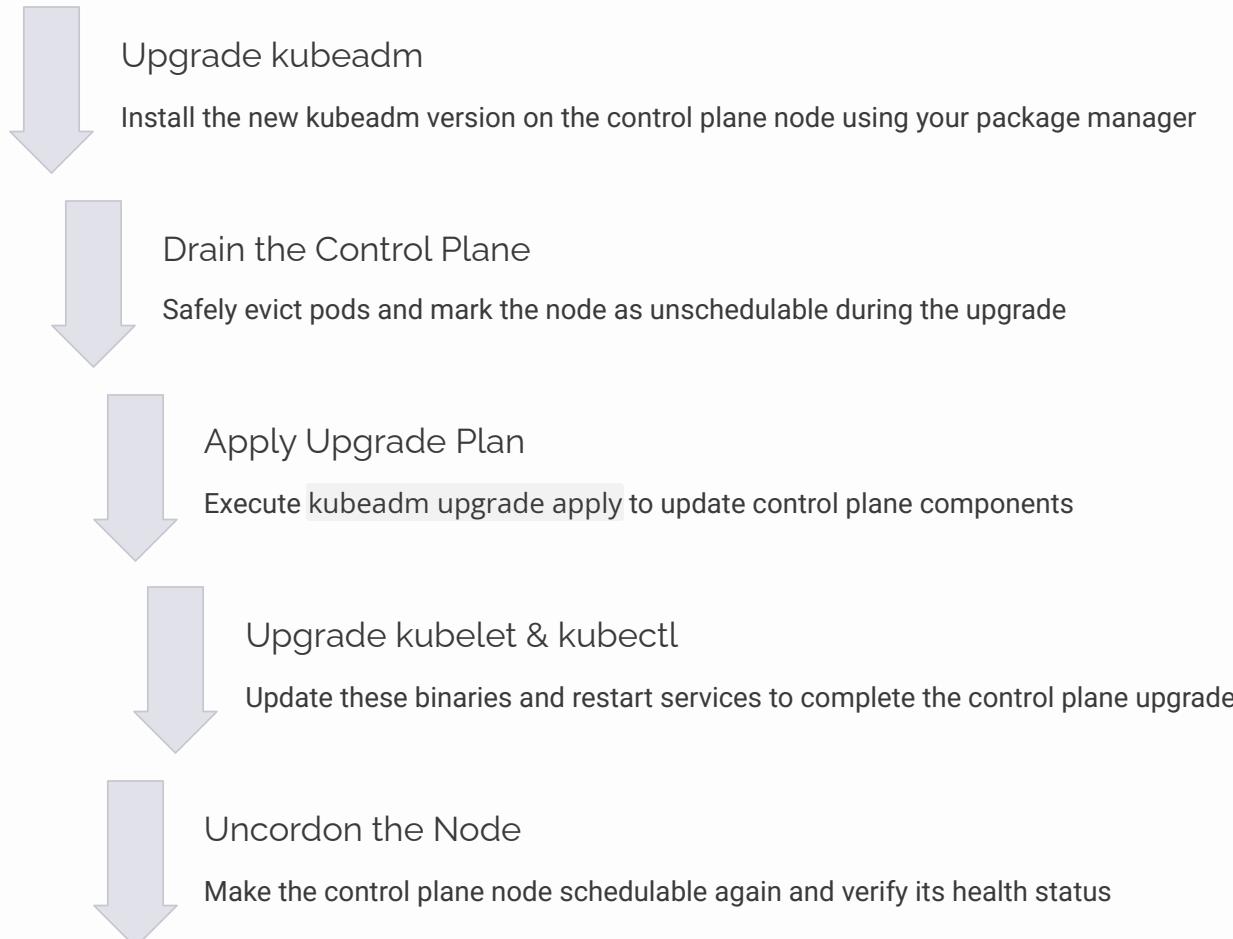
4

Schedule Maintenance Window

Plan the upgrade during low-traffic periods with stakeholder awareness

The Upgrade Process: Control Plane First

Kubernetes follows a specific upgrade order to maintain cluster stability. The control plane components must be upgraded before worker nodes, ensuring compatibility throughout the process.



Real-World Example: Upgrading from 1.27 to 1.28

```
# On control plane node
sudo apt-mark unhold kubeadm
sudo apt-get update
sudo apt-get install -y kubeadm=1.28.0-00
sudo apt-mark hold kubeadm

# Verify the upgrade plan
sudo kubeadm upgrade plan

# Drain the node
kubectl drain control-plane-node --ignore-daemonsets

# Apply the upgrade
sudo kubeadm upgrade apply v1.28.0

# Upgrade kubelet and kubectl
sudo apt-mark unhold kubelet kubectl
sudo apt-get install -y kubelet=1.28.0-00 kubectl=1.28.0-00
sudo apt-mark hold kubelet kubectl
sudo systemctl daemon-reload
sudo systemctl restart kubelet

# Uncordon the node
kubectl uncordon control-plane-node
```

This realistic sequence shows the exact commands you'd run, demonstrating how straightforward the process becomes with proper preparation.



Upgrading Worker Nodes Safely

Worker nodes can be upgraded one at a time or in batches, depending on your cluster size and redundancy. The key principle is maintaining workload availability throughout the process.

- 1 Cordon Node
Prevent new pods from being scheduled while existing ones continue running
- 2 Drain Workloads
Gracefully evict pods to other nodes with respect for PodDisruptionBudgets
- 3 Upgrade Components
Update kubeadm, kubelet, and kubectl using the same version as control plane
- 4 Uncordon & Verify
Make the node available again and confirm it rejoined the cluster successfully

Common Upgrade Pitfalls to Avoid



Skipping Versions

Never jump more than one minor version. Upgrading from 1.25 to 1.27 requires going through 1.26 first to avoid compatibility issues.



Rushing Production

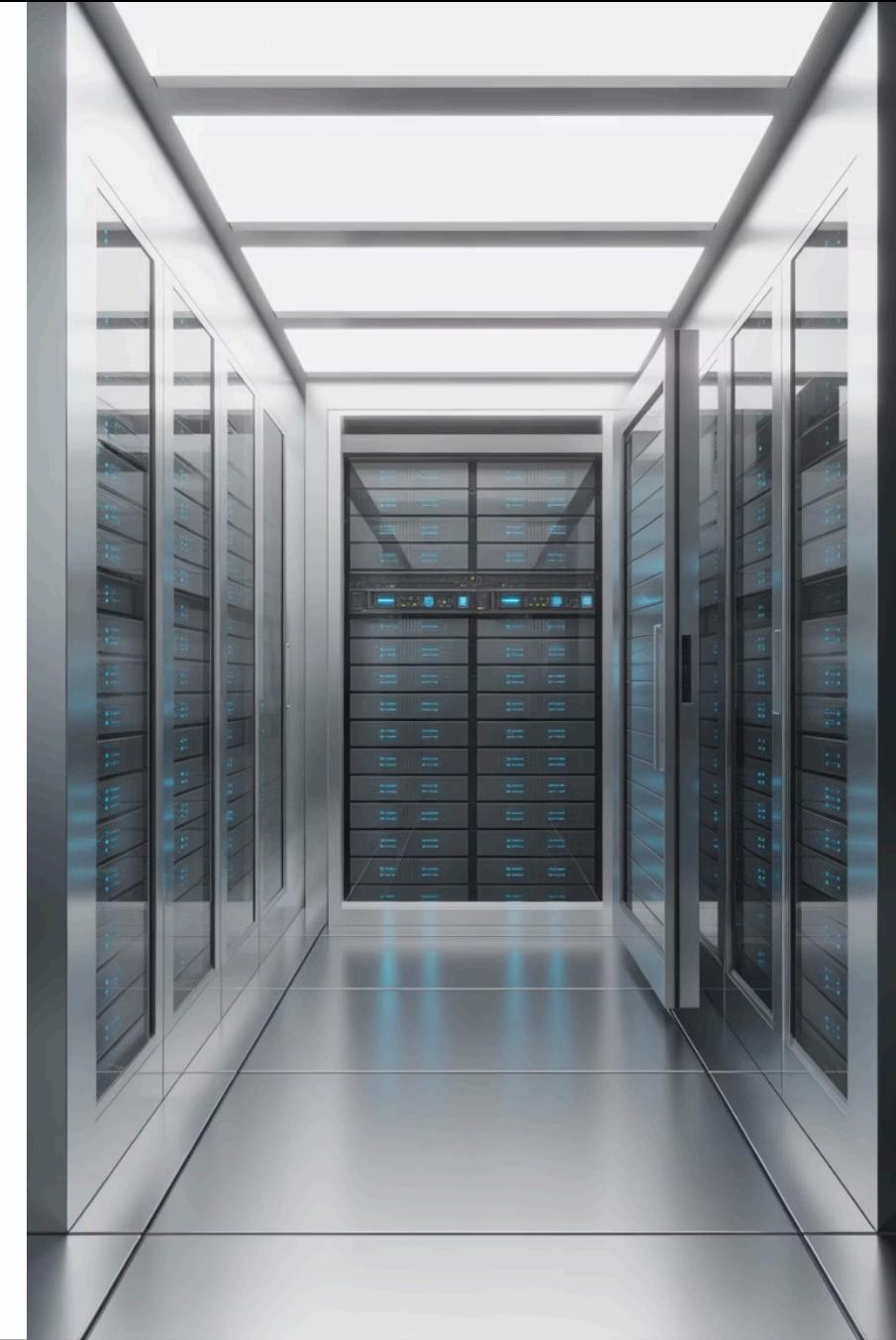
Always test the upgrade in staging environments first. That extra hour of testing can save days of production downtime.



Forgetting Backups

ETCD snapshots are your safety net. Take them before any upgrade, even if you're confident everything will work perfectly.

Chapter 2: ETCD Backup Strategies



What is ETCD and Why It's Critical

ETCD is the distributed key-value store that serves as Kubernetes' source of truth. It stores all cluster data—every deployment, service, secret, and configuration state lives here. If ETCD fails without a backup, your cluster becomes unrecoverable.

Think of ETCD as your cluster's memory. Just as you'd backup important documents before reformatting a computer, you must backup ETCD before any risky operation. A single corrupted ETCD database can render an entire cluster useless, making backups non-negotiable for production environments.

100%

Cluster State

All configuration stored in ETCD

3

Default Replicas

For high availability

Creating an ETCD Snapshot

The ETCD snapshot process captures the entire cluster state in a single file. This operation is quick, consistent, and can be performed without downtime, making it ideal for regular backups.

```
# Set ETCD variables
ETCDCTL_API=3 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 /backup/etcd-snapshot-$(date +%Y%m%d-%H%M%S).db

# Verify the snapshot
ETCDCTL_API=3 etcdctl --write-out=table snapshot status /backup/etcd-snapshot-*.*
```

- **Pro Tip:** Store snapshots on external storage like S3 or NFS. Local storage disappears if the node fails, defeating the purpose of backups.



Automated Backup Strategy

- 1 ── Hourly Snapshots
Automated backups every hour during business hours for rapid recovery
- 2 ── Daily Archives
One snapshot per day retained for 30 days to handle long-term issues
- 3 ── Weekly Long-term
Weekly backups kept for 6 months for compliance and audit requirements

This tiered approach balances recovery point objectives with storage costs. You can restore recent changes quickly while maintaining historical data for compliance.

ETCD Restore Process

Restoring from an ETCD snapshot requires stopping the ETCD cluster, replacing the data directory, and restarting with the restored snapshot. This process is destructive—it overwrites current cluster state entirely.

1

Stop ETCD

Halt all ETCD pods or systemd services across control plane nodes

2

Restore Snapshot

Use `etcdctl snapshot restore` to create a new data directory from the backup file

3

Update Configuration

Modify ETCD manifest to point to the restored data directory location

4

Restart ETCD

Start ETCD services and verify cluster health before resuming normal operations



Real Disaster Recovery Scenario

The Problem

At 2 AM, an engineer accidentally runs `kubectl delete ns production instead of kubectl delete pod`. The entire production namespace—containing 50 microservices, databases, and years of configuration—vanishes instantly.

Panic sets in. Customers are getting errors. The monitoring dashboard is empty. Revenue is bleeding. What do you do?

The Solution

Because you have automated ETCD backups, you restore the snapshot from 30 minutes before the incident. Within 10 minutes, every service, secret, and configmap returns exactly as it was.

Total downtime: 15 minutes. Data loss: zero. Career saved: one.



Chapter 3: Monitoring with Prometheus & Grafana

The Monitoring Stack Architecture

Prometheus collects and stores time-series metrics, while Grafana provides beautiful visualizations and alerting. Together, they form the industry-standard monitoring solution for Kubernetes.



Prometheus

Scrapes metrics from pods, nodes, and services at regular intervals, storing them in a time-series database optimized for queries



Grafana

Connects to Prometheus as a data source, rendering customizable dashboards with real-time visualizations and alert notifications



Alertmanager

Receives alerts from Prometheus, handles deduplication, grouping, and routes notifications to Slack, email, or PagerDuty

Installing Prometheus & Grafana

The Prometheus Operator and kube-prometheus-stack Helm chart provide production-ready monitoring with sensible defaults, saving weeks of manual configuration.

```
# Add Prometheus community Helm repository
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update

# Install kube-prometheus-stack
helm install monitoring prometheus-community/kube-prometheus-stack \
--namespace monitoring --create-namespace \
--set prometheus.prometheusSpec.retention=30d \
--set grafana.adminPassword=your-secure-password

# Access Grafana
kubectl port-forward -n monitoring svc/monitoring-grafana 3000:80
```

Within minutes, you'll have Prometheus scraping metrics and Grafana displaying pre-built dashboards for nodes, pods, and cluster health.



Key Metrics to Monitor

Node Resources

- CPU and memory utilization
- Disk space and I/O pressure
- Network bandwidth usage

Pod Health

- Container restart counts
- Pod scheduling failures
- Resource requests vs limits

Application Metrics

- HTTP request rates and latency
- Error rates and status codes
- Custom business metrics

Creating Effective Dashboards

Great dashboards tell a story at a glance. Start with high-level cluster health, then drill down into specific namespaces, deployments, and pods. Use colors strategically—green for healthy, yellow for warning, red for critical.

Organize panels logically: resource usage at the top, application metrics in the middle, and detailed logs at the bottom. Include meaningful time ranges and comparison periods to spot trends before they become incidents.



Keep It Simple

Show only essential metrics on overview dashboards



Use Variables

Template dashboards with namespace and pod selectors



Set Thresholds

Highlight values that require immediate attention

Setting Up Alerts That Matter

Alert fatigue is real. Too many notifications train your team to ignore them. Focus on actionable alerts that require human intervention, not noise that resolves itself.

Node Down

Trigger: Node unreachable for 5 minutes

Action: Investigate infrastructure issues immediately

High Memory Usage

Trigger: Pod using >90% memory for 10 minutes

Action: Check for memory leaks or scale horizontally

Pod Restart Loop

Trigger: Container restarted >5 times in 10 minutes

Action: Review logs for crashloop causes

Disk Space Critical

Trigger: Node disk >85% full

Action: Clean up logs or expand storage capacity

Real-World Monitoring Example

Your e-commerce application experiences sudden traffic spikes during flash sales. Without monitoring, you'd discover problems when customers complain. With proper monitoring, you see the spike coming.

What You See

- Request rate jumping 400% in 2 minutes
- Response time increasing from 200ms to 2s
- CPU usage hitting 80% across pods
- Memory steadily climbing

What You Do

- Horizontal Pod Autoscaler kicks in automatically
- New pods spin up in 30 seconds
- Load balances across increased capacity
- Response times return to normal



Chapter 4: Troubleshooting Kubernetes Failures



The Troubleshooting Mindset

Effective Kubernetes troubleshooting follows a systematic approach, not random guessing. Start broad, then narrow down based on symptoms. Every error message is a clue, and logs are your detective tools.



Common Pod Failure Scenarios



ImagePullBackOff

Pod can't download container image. Check image name spelling, registry authentication, and network connectivity to the registry.



CrashLoopBackOff

Container starts but immediately crashes. Review application logs with `kubectl logs` and check health probe configurations.



Pending State

Scheduler can't place pod on any node. Investigate resource constraints, node selectors, taints, and pod affinity rules.

Essential Troubleshooting Commands

```
# Check pod status and events
kubectl get pods -n production
kubectl describe pod failing-pod -n production

# View container logs
kubectl logs failing-pod -n production
kubectl logs failing-pod -n production --previous

# Execute commands inside pod
kubectl exec -it failing-pod -n production -- /bin/bash

# Check node status
kubectl get nodes
kubectl describe node worker-node-1

# View cluster events
kubectl get events -n production --sort-by=.lastTimestamp'

# Check resource usage
kubectl top nodes
kubectl top pods -n production
```

These commands form your troubleshooting toolkit. Master them, and you'll diagnose issues in minutes instead of hours.

Storage and PersistentVolume Issues

Problem: Pod Stuck in Pending

Your database pod won't start. Running `kubectl describe pod` shows:

Events:

```
Warning FailedScheduling  
persistentvolumeclaim  
"data-pvc" not found
```

The PVC doesn't exist or isn't bound to a PersistentVolume.

Solution Steps

1. Check if PVC exists: `kubectl get pvc`
2. Verify StorageClass is available
3. Review PV provisioner logs
4. Ensure sufficient storage capacity
5. Check access modes match requirements



Building Your Troubleshooting Playbook

Document every incident you solve. Over time, you'll build a playbook of known issues and solutions, dramatically reducing mean time to recovery.

Symptom What the user reported and initial observations	Investigation Commands run and data gathered during diagnosis	Root Cause The underlying issue discovered after testing
Resolution Exact steps taken to fix the problem	Prevention How to avoid this issue in the future	



Key Takeaways

Upgrade Carefully

Plan thoroughly, test in staging, backup ETCD, and upgrade control plane before worker nodes

Backup Religiously

Automate ETCD snapshots and store them externally—your future self will thank you

Monitor Proactively

Use Prometheus and Grafana to see problems before users do, and set meaningful alerts

Troubleshoot Systematically

Follow a methodical approach, document everything, and build your knowledge base over time

Kubernetes operations mastery comes from experience and preparation. Start with these foundations, practice in safe environments, and gradually build confidence for production deployments.