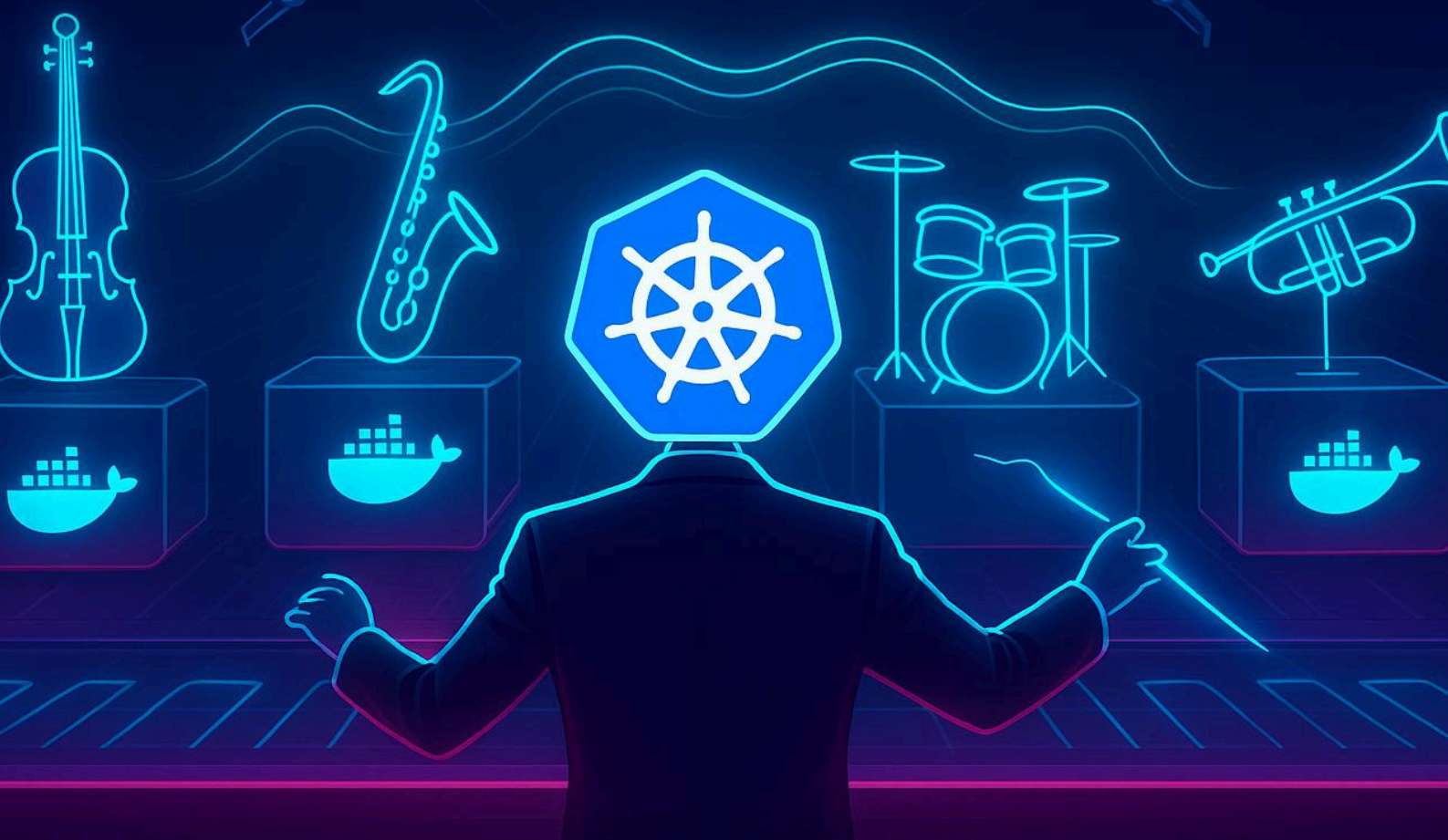
{JS}
MASTERY

Kubernetes Handbook

Created by **JS Mastery**

Visit ***jsmastery.com*** for more



What’s in the guide?

Kubernetes can feel overwhelming at first. There are pods, deployments, services, YAML files, and a command-line tool (kubectl) with what feels like a hundred flags to remember.

The goal of this guide is to make all of that a little less intimidating.

Think of this cheat sheet as your quick reference when you’re working with Kubernetes day to day.

Instead of searching through docs or scrolling Stack Overflow, you’ll find the most common commands, YAML patterns, and explanations right here in one place.

Whether you’re just starting out or you’ve been running clusters for a while, this guide is built to save you time.

Bookmark it, keep it open while you work, and don’t be afraid to experiment with the snippets.

...before you go

While this Kubernetes Cheat Sheet helps you quickly recall commands and YAML patterns, imagine taking the next step by learning how to design & run full production systems.

If you're eager to go beyond the basics — from building APIs and managing databases to handling authentication, scaling, and deploying on AWS — our upcoming **Ultimate Backend Course** has you covered.

It's the perfect blend of strong fundamentals and hands-on projects, guiding you from core backend principles to real-world DevOps and cloud deployment.

Join the waitlist today and be the first to get access.👆

What is Kubernetes?

Kubernetes (K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Instead of manually starting and stopping containers, Kubernetes ensures that your apps are always running, healthy, and scalable.

Key Features:

- **Automatic scaling** - Scale applications up/down based on demand
- **Self-healing** - Restart failed containers and replace unhealthy nodes
- **Load balancing** - Distribute traffic across healthy containers
- **Rolling updates** - Deploy new versions with zero downtime
- **Service discovery** - Automatically expose and discover services

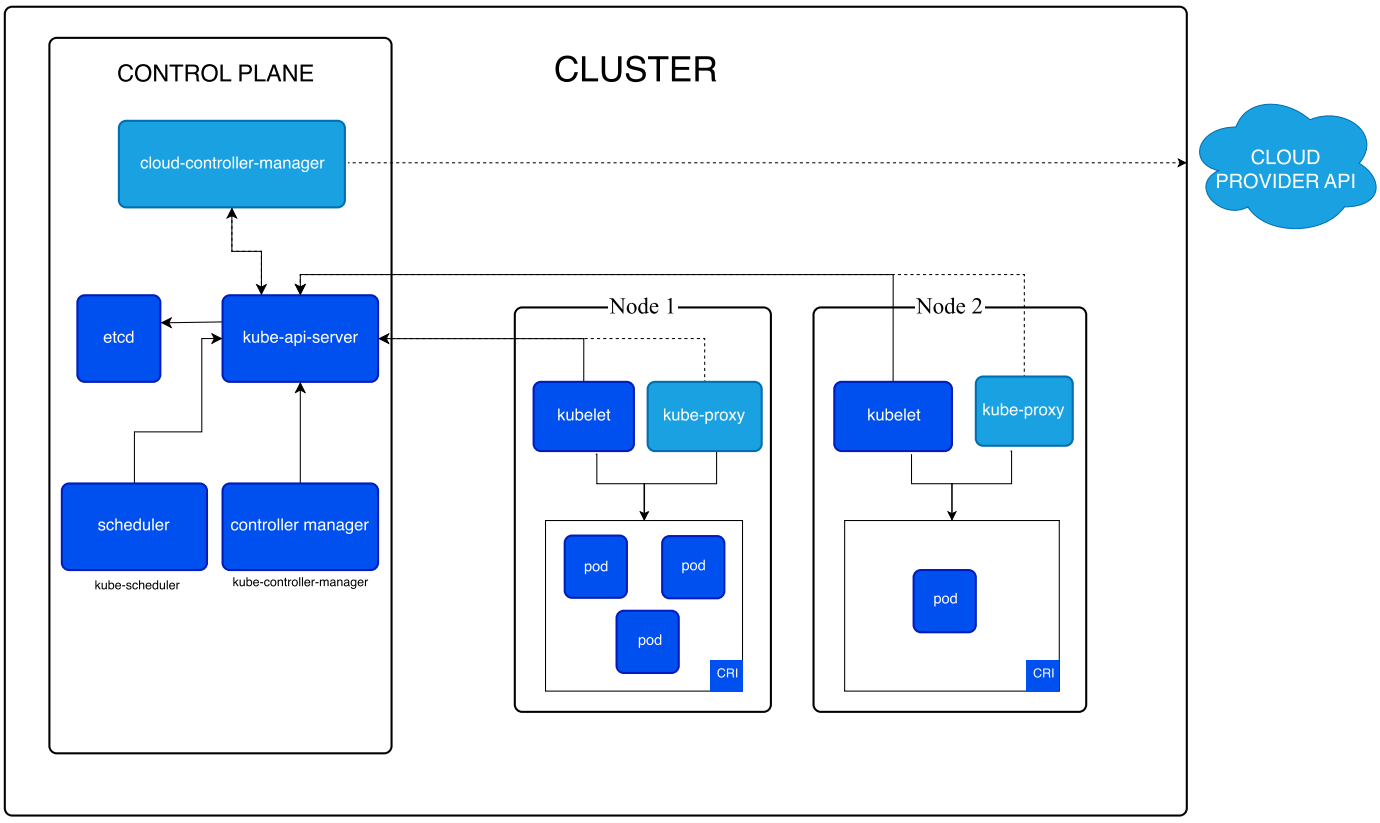
Core Components

Cluster

- A Kubernetes cluster is the full system: a set of nodes (machines) that run your workloads.

Two main parts:

- **Control Plane (brains)** → schedules workloads, maintains desired state.
- **Nodes (workers)** → actually run your apps in containers.



Cluster Components

Control Plane Components

API Server

The front door of the cluster; everything talks to it (kubectl, dashboards, other services).

etcd

A distributed key-value store that holds the cluster state (like the cluster's memory).

Controller Manager

Watches the cluster and makes changes to keep it in the desired state (e.g., if a pod crashes, it spins up a new one).

Scheduler

Decides which node a new pod should run on based on resources and constraints.

Cluster Components

Node Components

Kubelet

Agent running on each node, talks to the API server and ensures pods are running as expected.

Container Runtime

Software that runs containers (Docker, containerd, CRI-O).

Kube-proxy

Handles network rules so services can communicate with pods across nodes.

Core Components

Pod

- Smallest deployable unit in Kubernetes
- Contains one or more containers that share storage and network
- Containers in a pod share the same IP address and port space

Node

- Worker machine (physical or virtual) that runs pods
- Contains kubelet (node agent), container runtime, and kube-proxy

Deployment

- Manages replica sets and provides declarative updates for pods
- Ensures desired number of pod replicas are running
- Handles rolling updates and rollbacks

Core Components

Service

- Stable network endpoint to access pods
- Provides load balancing across pod replicas
- Types: ClusterIP, NodePort, LoadBalancer, ExternalName

ConfigMap

- Stores configuration data as key-value pairs
- Separates configuration from container images
- Can be mounted as volumes or environment variables

Secret

- Stores sensitive data (passwords, tokens, keys)
- Base64 encoded and encrypted at rest
- Similar to ConfigMap but for confidential data

Core Components

Ingress

- Manages external HTTP/HTTPS access to services
- Provides SSL termination, load balancing, and name-based virtual hosting
- Requires an Ingress Controller

Namespace

- Virtual cluster for organizing resources
- Provides scope for names and resource quotas
- Default namespaces: default, kube-system, kube-public

ReplicaSet

- Ensures a fixed number of pod replicas are running.
- Acts as the foundation for higher-level objects like Deployments.
- Automatically replaces failed or deleted pods to maintain the desired count.

Core Components

DaemonSet

Ensures one pod runs on every node (e.g., for logging/monitoring agents).

StatefulSet

Like Deployment but for apps that need stable network IDs and persistent storage (e.g., databases).

Job & CronJob

Run pods that do work and exit. CronJob schedules them like cron.

Horizontal Pod Autoscaler (HPA)

Automatically scales pods up/down based on CPU/memory usage or custom metrics.

Role-Based Access Control (RBAC)

Manages who can do what in a cluster.

YAML Manifests

Kubernetes resources are defined using YAML files with four main sections:

```yaml
apiVersion: v1              # API version
kind: Pod                   # Resource type
metadata:                   # Resource metadata
  name: my-pod
  namespace: default
  labels:
    app: my-app
spec:                       # Resource specification
  containers:
  - name: my-container
    image: nginx:1.21
    ports:
    - containerPort: 80
```

Common apiVersions:

- **v1** - Core API (Pod, Service, ConfigMap)
- **apps/v1** - Deployments, ReplicaSets
- **networking.k8s.io/v1** - Ingress
- **rbac.authorization.k8s.io/v1** - RBAC

kubectl Commands

Get cluster information

Terminal

```
> kubectl cluster-info
> kubectl version
```

Get all resources

Terminal

```
> kubectl get all
> kubectl get all --all-namespaces
```

Get specific resources

Terminal

```
> kubectl get pods
> kubectl get deployments
> kubectl get services
> kubectl get nodes
```

Working with Pods

List pods

Terminal

```
> kubectl get pods
> kubectl get pods -o wide
> kubectl get pods --show-labels
```

Describe pod (detailed info)

Terminal

```
> kubectl describe pod <pod-name>
```

Get pod logs

Terminal

```
> kubectl logs <pod-name>
> kubectl logs <pod-name> -f          # Follow logs
> kubectl logs <pod-name> -c <container> # Multi-container pod
```

Working with Pods

Execute commands in pod

Terminal

```
> kubectl exec <pod-name> -- <command>
> kubectl exec -it <pod-name> -- /bin/bash
```

Delete pod

Terminal

```
> kubectl delete pod <pod-name>
```

Working w/ Deployments

List deployments

Terminal

```
> kubectl get deployments
```

Create deployment

Terminal

```
> kubectl create deployment <name> --image=<image>
> kubectl create deployment nginx --image=nginx:1.21
```

Scale deployment

Terminal

```
> kubectl scale deployment <name> --replicas=5
```

Working w/ Deployments

Update deployment image

Terminal

```
> kubectl set image deployment/<name> <container>=<image>
```

Create deployment

Terminal

```
> kubectl rollout status deployment/<name>
> kubectl rollout history deployment/<name>
> kubectl rollout history deployment/<name>
> kubectl rollout history deployment/<name>
```

Working with Services

List services

Terminal

```
> kubectl get services
> kubectl get svc      # Short form
```

Expose deployment as service

Terminal

```
> kubectl expose deployment <name> --port=80 --target-port=8080
> kubectl expose deployment nginx --port=80 --type=LoadBalancer
```

Port forwarding

Terminal

```
> kubectl port-forward service/<service-name> 8080:80
> kubectl port-forward pod/<pod-name> 8080:80
```

Working with Services

Delete service

Terminal

```
> kubectl delete service <name>
```

Namespace Commands

List namespaces

Terminal

```
> kubectl get namespaces
> kubectl get ns      # Short form
```

Create namespace

Terminal

```
> kubectl create namespace <name>
```

Set default namespace

Terminal

```
> kubectl config set-context --current --namespace=<name>
```

Namespace Commands

Work with specific namespace

Terminal

```
> kubectl get pods -n <namespace>
> kubectl apply -f manifest.yaml -n <namespace>
```

Apply & Delete Resources

Apply configuration

Terminal

```
> kubectl apply -f <file.yaml>
> kubectl apply -f <directory>/
> kubectl apply -f <url>
```

Delete resources

Terminal

```
> kubectl delete -f <file.yaml>
> kubectl delete pod,service <name>
> kubectl delete all --all  #Delete all resources in namespace
```

Resource Management

Creating Deployments

```yaml
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
```

Scaling Applications

Manually scale deployment to specific replica count

Terminal

```
> kubectl scale deployment nginx-deployment --replicas=5
```

Set up horizontal pod autoscaler

Terminal

```
> kubectl autoscale deployment nginx-deployment --cpu-
  percent=50 --min=1 --max=10
```

Check horizontal pod autoscaler status

Terminal

```
> kubectl get hpa
```

Rolling Updates

Update deployment with new image version

Terminal

```
> kubectl set image deployment/nginx-deployment
  nginx=nginx:1.22
```

Check the status of rollout

Terminal

```
> kubectl rollout status deployment/nginx-deployment
```

View rollout history with revisions

Terminal

```
> kubectl rollout history deployment/nginx-deployment
```

Rollbacks

Rollback to previous version

Terminal

```
> kubectl rollout undo deployment/nginx-deployment
```

Rollback to specific revision number

Terminal

```
> kubectl rollout undo deployment/nginx-deployment --to-
  revision=2
```

Perform rolling restart of all pods

Terminal

```
> kubectl rollout restart deployment/nginx-deployment
```

Networking & Storage

SERVICE TYPES

ClusterIP (Default)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
  type: ClusterIP
```

Networking & Storage

NodePort

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30080
  type: NodePort
```

Networking & Storage

LoadBalancer

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-loadbalancer
spec:
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
  type: LoadBalancer
```

Networking & Storage

Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
  tls:
  - hosts:
    - myapp.example.com
    secretName: tls-secret
```

Networking & Storage

PERSISTENT VOLUMES

Persistent Volume (PV)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /data
```

Networking & Storage

Persistent Volume Claim (PVC)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Networking & Storage

Using PVC in Deployment

```yaml
spec:
  template:
    spec:
      containers:
      - name: app
        image: nginx
        volumeMounts:
        - name: storage
          mountPath: /usr/share/nginx/html
      volumes:
      - name: storage
        persistentVolumeClaim:
          claimName: my-pvc
```

Troubleshooting Cmds

Check cluster status

Terminal

```
> kubectl get componentstatuses
> kubectl get nodes
```

Debug pod issues

Terminal

```
> kubectl describe pod <pod-name>
> kubectl logs <pod-name> --previous
> kubectl get events --sort-by=.metadata.creationTimestamp
```

Check resource usage

Terminal

```
> kubectl top nodes
> kubectl top pods
```

Troubleshooting Cmds

Debug network issues

Terminal

```
> kubectl get endpoints
> kubectl describe service <service-name>
> kubectl get ingress
```

Check storage

Terminal

```
> kubectl get pv
> kubectl get pvc
> kubectl describe pvc <pvc-name>
```

Useful Aliases

Add to your shell profile:

Terminal

```
> alias k='kubectl'
> alias kg='kubectl get'
> alias kd='kubectl describe'
> alias kdel='kubectl delete'
> alias kaf='kubectl apply -f'
> alias kex='kubectl exec -it'
> alias klog='kubectl logs -f'
```

YAML Best Practices

✅ DO:

- Use proper indentation (2 spaces)
- Always specify resource limits and requests
- Use meaningful labels and selectors
- Version your container images (avoid `latest`)
- Use ConfigMaps and Secrets for configuration

❌ AVOID:

- Mixing tabs and spaces
- Running containers as root
- Using `latest` tag in production
- Hardcoding values in manifests
- Ignoring resource limits

Common YAML Gotchas

❌ Wrong indentation

```yaml
spec:
containers:
- name: app
```

✅ Correct indentation

```yaml
spec:
  containers:
  - name: app
```

Common YAML Gotchas

❌ Missing quotes for string numbers

```yaml
env:
- name: PORT
  value: 8080
```

✅ Proper quoting

```yaml
env:
- name: PORT
  value: "8080"
```

Common YAML Gotchas

❌ Incorrect selector matching

```yaml
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      name: nginx     # Doesn't match!
```

✅ Matching labels

```yaml
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
```

Quick Reference Cmds

Get everything in current namespace

Terminal

```
> kubectl get all
```

Watch resources (real-time updates)

Terminal

```
> kubectl get pods -w
```

Output to YAML/JSON

Terminal

```
> kubectl top pod <name> -o yaml
> kubectl top service <name> -o json
```

Quick Reference Cmds

Dry run (validate without creating)

Terminal

```
> kubectl apply -f manifest.yaml --dry-run=client

> kubectl create deployment test --image=nginx --dry-run=client
  -o yaml
```

Force delete stuck resources

Terminal

```
> kubectl delete pod <name> --force --grace-period=0
```

Copy files to/from pods

Terminal

```
> kubectl cp <local-path> <pod>:<remote-path>

> kubectl cp <pod>:<remote-path> <local-path>
```

Resource Monitoring

Check cluster status

Terminal

```
> kubectl get componentstatuses
> kubectl get nodes -o wide
```

Debug pod issues

Terminal

```
> kubectl describe pod <pod-name>
> kubectl logs <pod-name> --previous  #Previous container logs
> kubectl get events --sort-by=.metadata.creationTimestamp
```

Debug network issues

Terminal

```
> kubectl get endpoints
> kubectl describe service <service-name>
> kubectl describe ingress <ingress-name>
```

Resource Monitoring

Check storage

Terminal

```
> kubectl get pv
> kubectl get pvc
> kubectl describe pvc <pvc-name>
> kubectl get storageclass
```

RBAC debugging

Terminal

```
> kubectl auth can-i create pods
```

Maintenance Commands

Uncordon node after maintenance

Terminal

```
> kubectl uncordon <node-name>
```

Cordon node (prevent new pods)

Terminal

```
> kubectl cordon <node-name>
```

Taint node

Terminal

```
> kubectl taint nodes <node-name> key=value:NoSchedule
```

Remove taint

Terminal

```
> kubectl taint nodes <node-name> key:NoSchedule-
```

The End

🎯 Remember: Kubernetes is a Marathon, Not a Sprint

Every resilient cluster starts small. Don't try to master every resource at once — begin with pods and deployments, grow into services and ingress, and layer on scaling and security as you go.

Your future self (and your team) will thank you for investing in a solid foundation.

Happy shipping! 🚀

P.S. If this guide sparked your curiosity about running real-world systems, you'll love our upcoming **Ultimate Backend Course**. Join the waitlist today for early access.