

# Kubernetes Intermediate

Visual handwritten study guide - Tier 2

## Understand

Clear, compact explanations

## Visualize

Diagrams that show the moving parts

## Practice

Readable commands and code

## Remember

Highlighted revision points

---

68 topics - each topic begins on a fresh page

A digital asset by Sooraj | [soorajdev.cloud](https://soorajdev.cloud)

# Course Map

Scan the complete learning path before you begin.

## Part 1 - Inside the Cluster

|   |                                                       |   |                                                          |
|---|-------------------------------------------------------|---|----------------------------------------------------------|
| 1 | Course Introduction: From User to Operator            | 2 | The Control Plane in Depth: One Request, Start to Finish |
| 3 | Controllers, Informers and the Reconciliation Pattern | 4 | etcd: The Database Under Everything                      |
| 5 | Build a Real Multi-Node Cluster with kubeadm          | 6 | Server-Side Apply and Field Ownership                    |
| 7 | Ownership, Garbage Collection and Finalizers          | 8 | API Lifecycle, Flow Control and kubectl Power Tools      |

## Part 2 - Advanced Workloads

|    |                                               |    |                                                               |
|----|-----------------------------------------------|----|---------------------------------------------------------------|
| 9  | Deployment Rollouts Under the Microscope      | 10 | Progressive Delivery: Blue/Green, Canary and Traffic Shifting |
| 11 | Graceful Shutdown and Zero-Downtime Deploys   | 12 | Probes, Startup and In-Place Resize                           |
| 13 | Jobs: Parallelism, Indexes and Failure Policy | 14 | CronJobs in Production                                        |
| 15 | StatefulSets in Depth                         | 16 | Multi-Container Patterns and Debugging Running Pods           |

## Part 3 - Scheduling and Scaling

|    |                                                    |    |                                                       |
|----|----------------------------------------------------|----|-------------------------------------------------------|
| 17 | How the Scheduler Decides                          | 18 | Affinity, Anti-Affinity and Topology Spread           |
| 19 | Taints, Tolerations and Node Pools                 | 20 | Priority, Preemption, QoS and Eviction                |
| 21 | Requests, Limits and What the Kernel Actually Does | 22 | Horizontal and Vertical Autoscaling                   |
| 23 | Event-Driven Autoscaling with KEDA                 | 24 | Cluster Autoscaling: Cluster Autoscaler and Karpenter |

## Part 4 - Networking in Depth

|    |                                                      |    |                                                                   |
|----|------------------------------------------------------|----|-------------------------------------------------------------------|
| 25 | CNI Plugins: Overlay, Routing and eBPF               | 26 | Services Internals: kube-proxy, EndpointSlices and Traffic Policy |
| 27 | DNS at Scale: CoreDNS, ndots and NodeLocal DNSCache  | 28 | Gateway API in Depth                                              |
| 29 | TLS and DNS Automation: cert-manager and ExternalDNS | 30 | NetworkPolicy Patterns and Extended Policy Engines                |
| 31 | Service Mesh: When It Earns Its Complexity           | 32 | Egress, Multi-Cluster and Network Debugging                       |

## Part 5 - Storage and Stateful Data

|    |                                                                  |    |                                                 |
|----|------------------------------------------------------------------|----|-------------------------------------------------|
| 33 | CSI Architecture: What Happens When a PVC Is Created             | 34 | StorageClasses, Topology, Expansion and Reclaim |
| 35 | Snapshots, Clones and Volume Populators                          | 36 | Ephemeral and Local Storage                     |
| 37 | Running Databases on Kubernetes: Operators and Honest Trade-offs | 38 | Backup and Disaster Recovery with Velero        |
| 39 | Storage Performance, Shared Filesystems and Data Lifecycle       |    |                                                 |

## Part 6 - Security and Policy

|    |                                                                   |    |                                                              |
|----|-------------------------------------------------------------------|----|--------------------------------------------------------------|
| 40 | RBAC Beyond the Basics: Escalation, Aggregation and Audit         | 41 | Authentication: Humans, Workloads and Workload Identity      |
| 42 | Pod Security in Depth: Contexts, Profiles and Isolation           | 43 | Secrets Management: Encryption, External Stores and Rotation |
| 44 | Policy as Code: ValidatingAdmissionPolicy, Kyverno and Gatekeeper | 45 | Software Supply Chain: Provenance, Signing and Verification  |
| 46 | Auditing, Runtime Detection and Incident Response                 | 47 | Multi-Tenancy: Sharing a Cluster Without Sharing Problems    |
| 48 | Cluster Hardening and Compliance Scanning                         |    |                                                              |

## Part 7 - Observability and Delivery

|    |                                                                     |    |                                                        |
|----|---------------------------------------------------------------------|----|--------------------------------------------------------|
| 49 | Metrics with Prometheus: Scraping, Operators and kube-state-metrics | 50 | Alerting and SLOs That Wake the Right People           |
| 51 | Logging Pipelines: Collect, Parse, Store, Search                    | 52 | Distributed Tracing and OpenTelemetry                  |
| 53 | Helm in Depth: Charts as a Product                                  | 54 | Kustomize in Depth and Configuration Strategy          |
| 55 | GitOps with Argo CD and Flux                                        | 56 | CI for Kubernetes: Validate, Test, Preview and Promote |

## Part 8 - Operating, Projects and Reference

|            |                                                                      |            |                                                       |
|------------|----------------------------------------------------------------------|------------|-------------------------------------------------------|
| 57         | Upgrading Clusters Without Drama                                     | 58         | Node Lifecycle, Maintenance and Failure Handling      |
| 59         | Capacity Planning and Cost Management                                | 60         | Extending Kubernetes: CRDs and the Operator Pattern   |
| 61         | Reliability Engineering: Chaos, Game Days and Post-Incident Learning | PROJECT 1  | Project 1: Build a Production-Grade Platform Baseline |
| PROJECT 2  | Project 2: Upgrade, Break and Recover                                | PROJECT 3  | Project 3: A Self-Service Multi-Tenant Platform       |
| 65         | Review Questions                                                     | APPENDIX A | Operator's Command Reference                          |
| APPENDIX B | Advanced Troubleshooting Guide                                       | APPENDIX C | Glossary and Next Steps                               |

# Course Introduction: From User to Operator

What this volume assumes, what it adds, and how to work through it with a real multi-node cluster.

The beginner volume taught you to **use** Kubernetes: write manifests, run workloads, expose them and keep them healthy. This volume teaches you to **operate** it: understand what the control plane is actually doing, schedule and scale with intent, build secure multi-team clusters, observe them, and deliver changes safely.

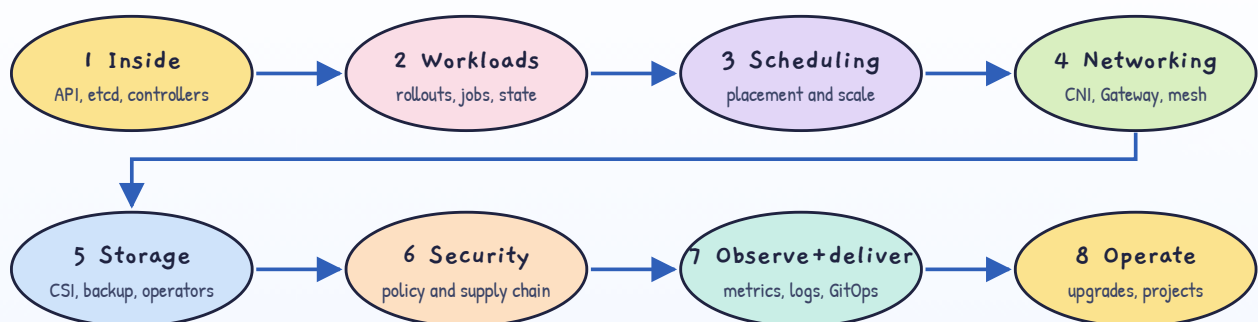
## What you should already be comfortable with

| Skill                | Quick self-test                                                                        |
|----------------------|----------------------------------------------------------------------------------------|
| Core objects         | Write a Deployment, Service, ConfigMap and PVC from memory, then apply them            |
| kubectl              | Find why a pod is Pending using only <code>describe</code> and <code>get events</code> |
| Networking           | Explain what happens between <code>curl http://api</code> and the pod responding       |
| Config and storage   | Explain PV, PVC and StorageClass in one minute                                         |
| RBAC basics          | Grant one ServiceAccount read access to pods in one namespace                          |
| Linux and containers | Explain namespaces and cgroups, and what a container image is                          |

**Watch out:** If two or more of these feel shaky, spend a week on the Beginner volume first. Intermediate material builds on them constantly and does not re-teach them.

## What you will learn

**DIAGRAM** The eight parts of this volume



Each part ends with Labs; Part 8 has three projects.

## How the notes are organised

- ✓ **Concept first, then the mechanism, then a lab.** You should be able to predict what a command does before you run it.
- ✓ **Failure modes** are called out explicitly. Intermediate engineers are paid for what they do when things break.
- ✓ **Version facts** are written for Kubernetes 1.37 (October 2026). Where behaviour changed recently the notes say so.
- ✓ **Illustrative output** is labelled as such; your versions, names and timings will differ.

## The lab environment

Use a **multi-node** cluster. Single-node clusters hide scheduling, networking and failure behaviour. A kind cluster with one control-plane node and three workers is enough for most labs. Topic 5 builds a **kubeadm** cluster on VMs for the control-plane and upgrade work that kind cannot show.

### YAML SNIPPET

15 lines

```
# kind-lab.yaml (4 nodes, zones simulated with labels)
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
networking:
  disableDefaultCNI: true           # install Cilium or Calico yourself in Part 4
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/16
nodes:
- role: control-plane
- role: worker
  labels: { topology.kubernetes.io/zone: zone-a }
- role: worker
  labels: { topology.kubernetes.io/zone: zone-b }
- role: worker
  labels: { topology.kubernetes.io/zone: zone-c }
```

### BASH SNIPPET

2 lines

```
$ kind create cluster --name lab --config kind-lab.yaml
$ kubectl get nodes -L topology.kubernetes.io/zone # nodes stay NotReady until a CNI is installed
```

Labs that simulate failures (stopping nodes, partitioning networks, killing control-plane components) are safe in kind and VMs. Never run them on a shared or production cluster.

## How to study

- 1 Read a topic, then close the notes and write down the mechanism in your own words.
- 2 Run the lab, and **change one thing** to see what breaks.
- 3 Keep a personal runbook: every incident-style lab becomes a page of symptoms, commands and fixes.
- 4 Finish with the review questions and the projects; they are designed to expose gaps.

### Key takeaways

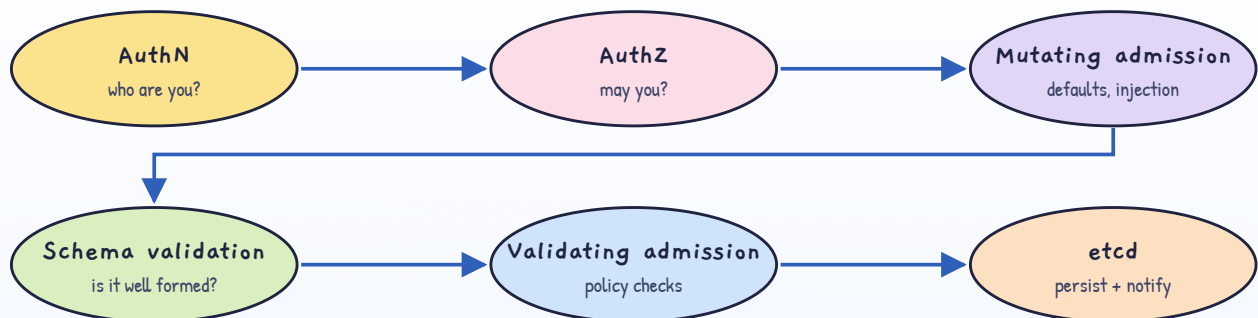
- Operating Kubernetes means understanding the mechanisms underneath the objects.
- Use a multi-node lab; failure and placement behaviour only appear with several nodes.
- Concept, mechanism, lab, then break it on purpose.

# The Control Plane in Depth: One Request, Start to Finish

Follow a `kubectl apply` through authentication, authorisation, admission, storage and the controllers that act on it.

Everything in Kubernetes is a **record in a database** (etcd) that components watch and act on. The API server is the only component that reads and writes that database. Understanding what happens to a request on the way in tells you where policies run, where errors originate and where latency hides.

**DIAGRAM** A write request through the API server



A request is rejected at the first stage it fails; later stages never see it.

## Stage by stage

| Stage                    | What happens                                                                                               | Typical failure message                                      |
|--------------------------|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| Authentication           | Client certificates, bearer tokens (ServiceAccount, OIDC), webhook; produces a user and groups             | <code>Unauthorized</code> (401)                              |
| Authorisation            | RBAC (and optionally Node, ABAC, webhook) decides allow or not; purely additive                            | <code>Forbidden: ... cannot create resource ...</code> (403) |
| Mutating admission       | Built-in plugins (DefaultStorageClass, ServiceAccount) and mutating webhooks or policies change the object | Webhook timeout or <code>failed calling webhook</code>       |
| Object schema validation | The object is checked against the API schema for its version                                               | <code>unknown field</code> , <code>Invalid value</code>      |
| Validating admission     | Policies (PSA, ValidatingAdmissionPolicy, webhooks like Kyverno or Gatekeeper) can reject                  | <code>admission webhook ... denied the request</code>        |
| Persistence              | Written to etcd with a new <code>resourceVersion</code> ; watchers are notified                            | <code>etcdserver: request timed out</code> (rare, serious)   |

Mutating admission runs **before** validating admission, so validating policies see the final, mutated object. Mutating webhooks can be called more than once (re-invocation) and must be idempotent.

## What the API server also does

- ✓ **Serves the watch API:** clients open a long-lived HTTP stream and receive events when objects change, instead of polling.
- ✓ **Converts between API versions:** objects are stored in one version and served in others (this is how `v1beta1` and `v1` coexist).

- ✓ Defaults fields you left out (`restartPolicy`, `dnsPolicy`, `terminationGracePeriodSeconds` ...).
- ✓ Rate limits and prioritises requests with API Priority and Fairness (Topic 8).
- ✓ Aggregates extension API servers (like metrics-server) under `/apis/...`.

## Watch the API yourself

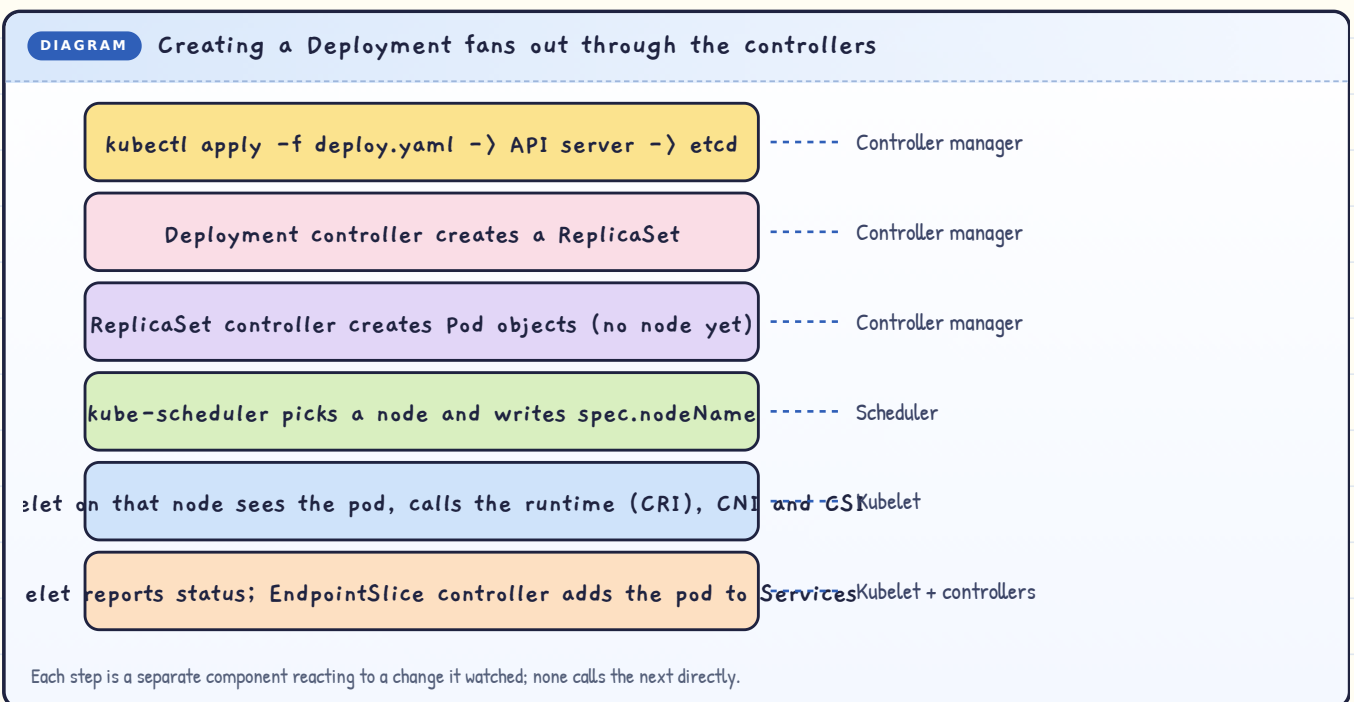
```

BASH SNIPPET 11 lines

$ kubectl get --raw /readyz?verbose | head -8
[+]ping ok
[+]log ok
[+]etcd ok
[+]poststarthook/start-apiserver-admission-initializer ok
...
$ kubectl get pods -v=8 2>&1 | grep -E 'GET|Response Status' | head -3    # shows the real HTTP request
I1025 11:02:01.114 round-trippers.go:632] "Response" verb="GET"
url="https://127.0.0.1:41893/api/v1/namespaces/default/pods?limit=500" status="200 OK"
$ kubectl proxy &
$ curl -s localhost:8001/api/v1/namespaces/kube-system/pods?limit=1 | head -c 300
$ kubectl get --raw '/api/v1/namespaces/default/pods?watch=true' --request-timeout=10s    # a raw watch stream

```

## After the API server: the controllers



## See the chain in the events

```

BASH SNIPPET 10 lines

$ kubectl create deployment web --image=nginx:1.29 --replicas=2
$ kubectl get events --sort-by=.lastTimestamp
LAST SEEN   TYPE      REASON          OBJECT          MESSAGE
9s          Normal    ScalingReplicaSet deployment/web    Scaled up replica set web-7c5d6b9f4 to 2
9s          Normal    SuccessfulCreate replicaset/web-7c5d6b9f4 Created pod: web-7c5d6b9f4-x2k8p
9s          Normal    Scheduled        pod/web-7c5d6b9f4-x2k8p Successfully assigned default/web-... to lab-worker2
8s          Normal    Pulling           pod/web-7c5d6b9f4-x2k8p Pulling image "nginx:1.29"
3s          Normal    Pulled            pod/web-7c5d6b9f4-x2k8p Successfully pulled image ...
3s          Normal    Created           pod/web-7c5d6b9f4-x2k8p Created container nginx
3s          Normal    Started           pod/web-7c5d6b9f4-x2k8p Started container nginx

```

Read the **REASON** column as a trace. The component that emitted each event is in `kubectl get events -o wide` (the **source** column), and it tells you which log to read.

## Where to look when it goes wrong

| Symptom                                 | Component to inspect              | How                                                                                        |
|-----------------------------------------|-----------------------------------|--------------------------------------------------------------------------------------------|
| Object accepted but nothing happens     | kube-controller-manager           | Logs in <code>kube-system</code> ; the object's events; owner references                   |
| Pod stays Pending without a node        | kube-scheduler                    | <code>describe pod</code> events; scheduler logs at higher verbosity                       |
| Pod is assigned a node but never starts | kubelet and runtime               | <code>describe pod</code> ; <code>journalctl -u kubelet</code> ; <code>crictl ps -a</code> |
| Request is rejected                     | API server and admission webhooks | The error text names the stage; audit logs say who and why                                 |
| Everything slow                         | API server / etcd                 | API server metrics ( <code>apiserver_request_duration_seconds</code> ), etcd latency       |

### BASH SNIPPET

7 lines

```
# On a kubeadm control-plane node the components run as static pods
$ ls /etc/kubernetes/manifests
etcd.yaml kube-apiserver.yaml kube-controller-manager.yaml kube-scheduler.yaml
$ sudo crictl ps --name kube-apiserver
$ kubectl -n kube-system logs kube-scheduler-lab-control-plane --tail=20
# kind: nodes are containers
$ docker exec -it lab-control-plane crictl ps
```

### Key takeaways

- A write passes authn, authz, mutating admission, validation, validating admission, then etcd.
- Components never call each other; they watch the API and react to changes.
- The event REASON column is a trace of which controller did what.