

Kubernetes For Beginners

Visual handwritten study guide

Understand

Clear, compact explanations

Visualize

Diagrams that show the moving parts

Practice

Readable commands and code

Remember

Highlighted revision points

63 topics – each topic begins on a fresh page
A digital asset by Sooraj | soorajdev.cloud

Course Map

Scan the complete learning path before you begin.

Part 1 - The Big Picture

1	Course Introduction	2	The Problem Kubernetes Solves
3	Kubernetes Architecture	4	Core Concepts: The Vocabulary Map
5	Build a Lab Cluster on Your Laptop	6	kubectl and kubeconfig
7	YAML and the Object Model	8	The API: Groups, Versions and What Your Cluster Serves

Part 2 - Running Workloads

9	Pods: The Smallest Deployable Unit	10	Multi-Container Pods and Init Containers
11	Pod Lifecycle, Restarts and Probes	12	Labels, Selectors and Annotations
13	ReplicaSets: Keeping N Pods Alive	14	Deployments: Declarative Apps
15	Rolling Updates, Rollbacks and Strategies	16	Jobs and CronJobs
17	DaemonSets and StatefulSets: First Look		

Part 3 - Networking

18	The Kubernetes Networking Model	19	Services: Stable Addresses for Pods
20	NodePort, LoadBalancer, ExternalName and Headless Services	21	DNS and Service Discovery
22	Getting Traffic In: Ingress and Gateway API	23	Gateway API Hands-On: Routes, TLS and Canaries
24	NetworkPolicy: A Firewall for Pods	25	Debugging Network Problems

Part 4 - Configuration and Storage

26	ConfigMaps: Configuration Outside the Image	27	Secrets: Sensitive Data, Handled Carefully
28	Using Config in Pods: Env Vars, Files and Updates	29	Volumes: Giving Pods Storage
30	PersistentVolumes and PersistentVolumeClaims	31	StorageClasses, Provisioning, Expansion and Snapshots
32	Resource Requests, Limits and QoS	33	Stateful Apps and Data Safety

Part 5 - Organising and Securing the Cluster

34	Namespaces: Dividing a Cluster	35	ResourceQuota and LimitRange: Fair Sharing
36	RBAC: Who Can Do What	37	ServiceAccounts: Identities for Pods
38	Security Contexts and Pod Security Standards	39	Scheduling: Where Do Pods Run?
40	Availability: Spread, Disruption Budgets and Priority	41	Admission Control and Policy Guardrails

Part 6 - Packaging and Operating

42	Kustomize: Environments Without Copy-Paste	43	Helm: The Package Manager
44	Debugging Method: Describe, Events and Logs	45	Status Decoder: CrashLoopBackOff, ImagePullBackOff, Pending and Friends
46	kubectl debug, Ephemeral Containers and Node Debugging	47	Logs, Metrics and Basic Observability
48	Autoscaling: HPA, VPA and Cluster Autoscaler	49	Maintenance: Draining, Upgrades and Backups

Part 7 - Real Applications

50	Walkthrough 1: A Stateless Web App, End to End	51	Walkthrough 2: WordPress with MySQL and Persistent Storage
52	Walkthrough 3: A Three-Tier App with Gateway, Policies and Autoscaling	53	Managed Kubernetes and Choosing Where to Run
54	GitOps: Git as the Source of Truth	55	Production Readiness Checklist

Part 8 - Projects and Reference

PROJECT 1	Project 1: Deploy a Complete App on a Fresh Cluster	PROJECT 2	Project 2: Fix the Broken Cluster
PROJECT 3	Project 3: Build a Secure Namespace Template	59	Review Questions
APPENDIX A	kubectl Cheat Sheet	APPENDIX B	Troubleshooting Decision Guide
APPENDIX C	Manifest Templates	APPENDIX D	Glossary and Where to Go Next

Course Introduction

A hands-on beginner course on Kubernetes: what it is, how it works, and how to run, expose, configure and troubleshoot real applications on it.

You can run one container with `docker run`. But what happens when the container crashes at 3 a.m.? When you need ten copies behind one address? When you want to ship a new version without anyone noticing? **Kubernetes is the system that answers those questions.** You tell it what you want running, and it works continuously to make reality match.

These notes take you from zero to capable. You do not need prior Kubernetes experience. You should be comfortable in a terminal and, ideally, know what a container image is (the Docker Beginner volume covers that). Every topic follows the same rhythm: a short idea, a diagram where one helps, commands and manifests you can run, and a recap to revise from.

Learn by doing. Reading about `kubectl apply` teaches you far less than applying a broken manifest and reading the error. Keep a lab cluster running next to these notes (Topic 5 sets one up in about ten minutes, free, on your laptop).

What this course covers

Part	What you will learn	Outcome
1. Big picture	Why Kubernetes, architecture, vocabulary, a lab cluster, kubectl, YAML	You can explain Kubernetes and talk to a cluster
2. Workloads	Pods, probes, labels, ReplicaSets, Deployments, Jobs, DaemonSets, StatefulSets	You can run and update applications
3. Networking	Cluster networking, Services, DNS, Gateway API and Ingress, NetworkPolicy	Your apps can be reached, safely
4. Config and storage	ConfigMaps, Secrets, volumes, PVCs, resources and QoS	Config is separate from code, data survives restarts
5. Organising and access	Namespaces, quotas, RBAC, ServiceAccounts, security contexts, scheduling	Many apps and people share a cluster safely
6. Operating	Kustomize, Helm, debugging, logs and metrics, autoscaling, maintenance	You can package, observe and fix workloads
7. Real applications	Three end-to-end deployments, managed Kubernetes, production checklist	You can deliver something real
8. Projects and reference	Projects, review questions, cheat sheets, troubleshooting, glossary	You can practise and revise

How to read these notes

- ✓ Topic pages start with a pink summary line, then the explanation, commands and a recap.
- ✓ Code blocks are meant to be typed or pasted. Lines starting with `$` are what you type; other lines are output.
- ✓ Yellow boxes hold ideas worth remembering. Red boxes warn about common mistakes. Green boxes are shortcuts.
- ✓ Labs are guided exercises. Do them in order; later topics reuse earlier results.
- ✓ Manifests (YAML files) are shown in full so you can copy them. Names such as `web` or `demo` are examples; change them freely.

What you need before you start

- ✓ A computer with 8 GB RAM or more (16 GB is comfortable) and about 15 GB of free disk space.
- ✓ A container runtime for the lab cluster: Docker Desktop, Docker Engine, Podman or similar (Topic 5).
- ✓ A terminal and a text editor. VS Code with the YAML and Kubernetes extensions is a good default.
- ✓ Basic command-line comfort: `cd`, `ls`, `cat`, editing a file.

DIAGRAM The path through this course



Each stage builds on the previous one. The labs assume you did the earlier ones.

A note on versions

Kubernetes ships a new minor version about three times a year (1.37 is current as these notes are written, in late 2026). Each minor version is supported with patches for roughly a year. The concepts here are stable: Pods, Deployments and Services have worked the same way for a decade. Specific details that change quickly (for example, which ingress controller to choose) are called out in boxes so you know to check current documentation.

BASH SNIPPET

5 lines

```
$ kubectl version
Client Version: v1.37.0
Kustomize Version: v5.7.1
Server Version: v1.37.0
# your numbers will differ; the commands in these notes are the stable core
```

Watch out: Most examples use **1.37-compatible** API versions such as `apps/v1`. If you copy manifests from old blog posts, you may meet APIs that were removed years ago (`extensions/v1beta1`, `policy/v1beta1` PodSecurityPolicy). Topic 8 shows how to check what your cluster actually serves.

Key takeaways

- Kubernetes keeps your declared desired state true, continuously.
- You will learn by applying manifests and reading what happens.
- The course moves from the big picture to workloads, networking, config, storage, access control and operations.

The Problem Kubernetes Solves

From one container on one laptop to hundreds across many machines, and why a scheduler and reconciliation loop became necessary.

One container is easy

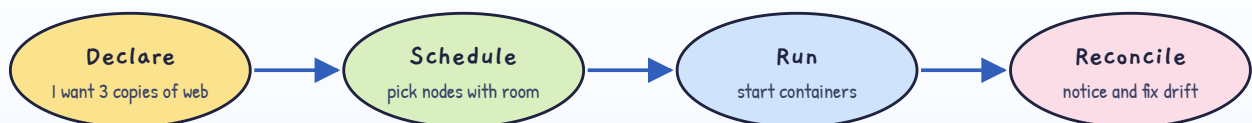
`docker run -d -p 8080:80 nginx` is simple, and for a hobby project on one server it may be all you need. The trouble starts when the stakes and the size grow.

Question	Plain Docker on one host	What you want
The container crashes	Restart policy can restart it; nothing else notices	Detect, restart, and tell me if it keeps failing
The host dies	Everything on it is gone until you fix it	Run those containers somewhere else automatically
Traffic doubles	Start more containers by hand, reconfigure the proxy	Add copies and spread the load automatically
New version	Stop old, start new: downtime, no easy undo	Replace gradually, roll back if it misbehaves
Ten teams, fifty services	A forest of scripts and snowflake servers	A common, declarative way to run everything
Config and secrets	Env files copied around	Managed centrally, injected at run time
Service discovery	Hard-coded addresses	Stable names that follow containers as they move

Orchestration: what is needed

Solving those problems takes a system that knows the **state** of many machines, decides **where** each container should run, watches the result, and **corrects** drift. That is called **container orchestration**. Kubernetes (Greek for 'helmsman', often abbreviated **K8s**: K, eight letters, s) is the one that won.

DIAGRAM Orchestration in four verbs



The fourth verb is the heart of Kubernetes: it never stops comparing what you asked for with what exists.

Declarative, not imperative

With scripts you describe **steps**: 'start container A, then B, then change the proxy'. With Kubernetes you describe a **desired state**: 'there should be three replicas of this image, reachable on port 80'. Kubernetes figures out the steps, and keeps figuring them out whenever something changes.

Imperative (steps)	Declarative (end state)
<code>docker run</code> three times, <code>docker rm</code> one when scaling down	<code>replicas: 3</code> , change the number, apply again
If a step fails halfway, you are in an unknown state	The system keeps trying until the state matches
Hard to review in Git	The file <i>is</i> the documentation and the review artefact
Drift accumulates silently	Drift is corrected automatically

What Kubernetes gives you

- ✓ **Self-healing:** failed containers are restarted; pods on dead nodes are replaced elsewhere.
- ✓ **Scaling:** manual (`replicas`) or automatic by load (Horizontal Pod Autoscaler).
- ✓ **Rolling updates and rollbacks:** change the image, Kubernetes swaps pods gradually and can undo.
- ✓ **Service discovery and load balancing:** stable names and virtual IPs in front of changing pods.
- ✓ **Config and secret management:** injected as env vars or files, changed without rebuilding images.
- ✓ **Storage orchestration:** attach volumes from local disks or cloud providers on demand.
- ✓ **Extensibility:** custom resources and operators let it manage databases, certificates, whatever you teach it.

What Kubernetes does not give you

Watch out: Kubernetes is not a PaaS. It does not build your images, write your CI pipeline, give you logging and metrics dashboards out of the box, make a badly designed app scalable, or remove the need to understand networking. It is a platform for **building** a platform. Many beginners are surprised by how much surrounding tooling a 'complete' cluster needs.

When you probably should not use it

Situation	Simpler answer
One small app on one server	Docker Compose on a VM
Static website	Static hosting/CDN
One HTTP service, pay-per-request, scale to zero	Cloud Run, Azure Container Apps, App Runner
Team with no one to operate a cluster, and no managed option	Managed container platforms (ECS/Fargate)
Learning for a job, or running many services in a team	Kubernetes is the right thing to learn

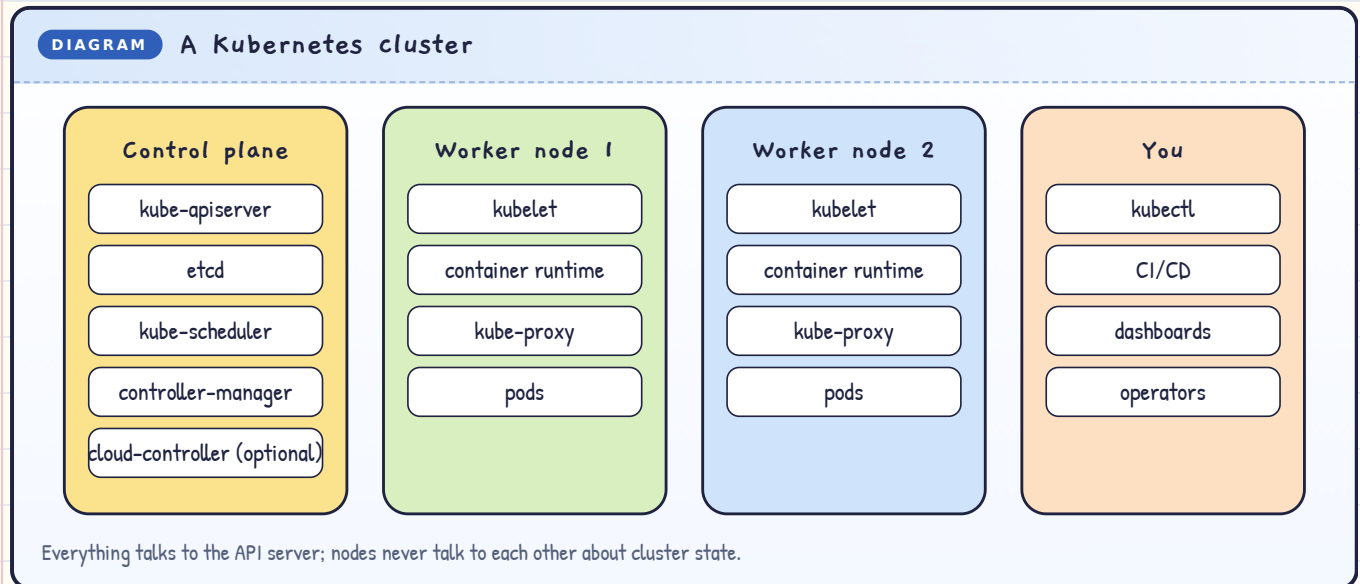
Key takeaways

- Orchestration schedules, runs, watches and repairs containers across machines.
- Declarative desired state plus continuous reconciliation is the core idea.
- Kubernetes is powerful but is a platform to build on, not a finished PaaS.

Kubernetes Architecture

The control plane, the nodes, and how a request becomes a running container.

A Kubernetes **cluster** is a set of machines (physical or virtual) split into two roles: the **control plane**, which makes decisions, and **worker nodes**, which run your containers.



Control plane components

Component	Role	Analogy
kube-apiserver	The front door: REST API for every operation; validates, authenticates, authorises, stores	Reception desk
etcd	Consistent key-value store holding all cluster state	The ledger
kube-scheduler	Assigns new pods to nodes with enough room and matching rules	Seating planner
kube-controller-manager	Runs many controllers (Deployment, ReplicaSet, Node, Job...) that reconcile state	Supervisors
cloud-controller-manager	Integrates with a cloud provider (load balancers, node lifecycle)	Cloud liaison

Node components

Component	Role
kubelet	Agent on each node. Gets pod specs from the API, tells the runtime to start containers, runs probes, reports status
Container runtime	Actually runs containers: containerd or CRI-O, speaking the Container Runtime Interface (CRI). Docker Engine is not needed
kube-proxy	Programs rules on the node so Service IPs reach pods (iptables, IPVS or nftables mode). Some CNIs replace it
CNI plugin	Gives pods network interfaces and IPs (Calico, Cilium, Flannel, cloud CNIs)