

Engineer

Kubernetes

a system that keeps a fleet of containers running



What it is

Kubernetes, often written K8s, is open-source software that runs containerised applications across a group of machines, called a cluster, and holds them to a description you give it: which image each application runs, how many copies exist, how much memory and CPU each copy may use, and how other programs reach it. The project's own documentation calls it "a portable, extensible, open source platform for managing containerized workloads and services". The name is Greek for helmsman or pilot, K8s keeps the K and the s and replaces the eight letters in between with their count, and Google open-sourced the project in 2014.

A cluster has two halves. The control plane holds the description: the API server is the one HTTP interface every change goes through, etcd stores what it is told, the scheduler assigns each new Pod to a suitable node, and the controller manager runs the loops that act on the description. On every worker node, the kubelet makes sure the Pods assigned there are running, kube-proxy maintains the network rules behind Services, and a container runtime such as containerd or CRI-O starts the containers. Docker Engine stopped being a built-in runtime in Kubernetes 1.24, and images built with Docker run unchanged, because they are standard OCI images.

The description is a set of objects, usually written as YAML. A Pod is the smallest unit Kubernetes schedules: one or more containers that share a network address and storage on one machine. A Deployment says how many identical Pods should exist and replaces them step by step when their template changes. A Service gives a changing set of Pods one stable name and address, picking its members by label. A small web application looks like this:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: registry.example.com/web:1.4.2
          resources:
            requests:
              cpu: 250m
              memory: 256Mi
            limits:
              memory: 256Mi
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8080
```

It is usually weighed against three smaller options. Docker Compose describes containers on one machine, and when that machine goes away, so do they. A service manager such as systemd can restart a crashed container and start it at boot, which covers a great many needs with far less machinery, but it has no notion of moving work to another machine. Hosted platforms such as Google Cloud Run or Azure Container Apps run containers without handing you a control plane to operate, in exchange for fewer settings. The documentation is also plain about what Kubernetes is not: a traditional all-inclusive platform as a service. It does not build your application, and it ships no databases or message queues of its own.

Use the concept in a decision

Put “What is Kubernetes?” to work in a real decision. We can help relate the concept to the system you are building or buying. Packaged to behave identically on a laptop, on a test server and in production, and to scale up when a busy week arrives.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Ran 20 Docker apps on Google Cloud and AWS — 20 containerised applications deployed and watched

Ran 20 Docker apps on Google Cloud and AWS Get in touch →

Why it matters

The central idea is that the cluster keeps working towards a written description instead of running a list of commands once. Applying the file above starts nothing directly; it records a wish. Controllers are control loops that watch the cluster and act to move its current state closer to the desired one. If a machine dies and three copies become two, a controller sees the gap, a new Pod is created, and the scheduler places it on a machine with room. Because the description is a set of files, it can live in a repository and go through the same review as the code it runs.

Releases come with the platform. The default strategy is a rolling update: new Pods start, and old ones stop as the new ones become ready, within limits that default to 25% extra and 25% missing at any moment. Each change to the Pod template is kept as a revision, ten by default, and `kubectl rollout undo` returns to the previous one. The script that once drained a server, swapped a binary and brought it back becomes built-in behaviour.

The scheduler also changes the economics of a fleet. It places each Pod by the CPU and memory the Pod requests, so many small services share a set of machines instead of each getting a server sized for its worst day.

The price is real. A cluster is a distributed system in its own right, with versions to upgrade on a schedule, a network layer, a storage layer, certificates and access control, all of which someone has to understand when they misbehave. Managed offerings such as Google Kubernetes Engine, Amazon EKS and Azure Kubernetes Service run the control plane for you, but every manifest, and usually the worker nodes, stay yours. For a handful of services on one or two machines, a plain service manager often does the job with fewer moving parts.

What it looks like in practice

The company's stack page states the rule plainly: Docker and Kubernetes for the workloads that earn them, and plain systemd units for the ones that do not. Kubernetes is chosen per workload, for the size and shape of the job, and not as a default.

Workloads that earned it appear in one engagement. Twenty Docker applications were deployed and maintained across Google Cloud and AWS, among them R-based analytics apps, with consistent configuration and monitoring so that no application became a special case. Troubleshooting went through Podman and the orchestration debugging through Kubernetes. The page reports that problems were diagnosed faster and that deployments stayed stable and reproducible.

The other half of the rule shows on the company's own platform. A container plane built on Podman and Quadlet runs each container as a service of the operating system's own supervisor, described in the same declarative form as everything else on the host. The containers use host networking and publish no ports, so the firewall rules the hardening role installs stay in charge. Where Kubernetes would roll out a new image through its API, a deploy user without root runs one fixed script that builds the image and restarts the service. On one host, that gives every container a written description and the supervisor the host already trusts, with no second process manager. What it leaves out is placement across machines and replacing a dead node, which is where Kubernetes begins to pay for its weight.

Where it goes wrong

Most Kubernetes trouble comes from the gap between what a manifest asks for and what the cluster reports back.

A rollout sits half-finished while the deploy job reports success. When new Pods never become ready, because of a wrong image tag, a crash on start, a failing readiness probe or a missing quota, the Deployment stops making progress. After the progress deadline, 600 seconds by default, it records the reason `ProgressDeadlineExceeded`, and the documentation says Kubernetes takes no other action: the old Pods still running keep serving, and nothing rolls back on its own. `kubectl rollout status` exits non-zero once the deadline passes, so a deploy step can wait on it and undo the release:

```
kubectl apply -f web.yaml
kubectl rollout status deployment/web --timeout=10m \
  || kubectl rollout undo deployment/web
```

A container restarts with `OOMKilled`, or slows down on a node with spare CPU. The kernel enforces limits, and the two resources behave differently. A container over its memory limit can be killed and restarted; one at its CPU limit is throttled, so it runs slower without dying, which looks like an application problem. The scheduler places Pods by what they request, not by what they use, so low requests crowd a node and high ones leave machines idle. A limit given without a request becomes the request. Set memory from measured usage, and treat CPU limits as a deliberate decision.

A slow-starting application is killed before it finishes starting. A failing liveness probe makes the kubelet kill and restart the container, so an application that needs a minute to load, checked after ten seconds, restarts forever. A startup probe holds off the other checks until the application is up. A readiness probe takes a Pod out of its Service without restarting it, which suits a Pod that is busy or warming a cache. A liveness probe that calls the database restarts every replica at once when the database is slow, turning a slow dependency into an outage.

A Pod started by hand disappears, and a fix made by hand comes undone. A Pod created on its own stays on its node until it ends, is deleted or evicted, or the node fails, and no controller recreates it; the documentation advises a Deployment or a Job even for a single copy. A change made with `kubectl edit` lives only in the cluster, and the next apply from the repository puts the old value back, so an emergency fix belongs in the files as soon as the emergency is over.

Further reading

- Kubernetes overview — the project's own account of what Kubernetes is, what it offers and what it leaves out.
- Pods — the smallest unit Kubernetes schedules, its lifecycle and why Pods are rarely created by hand.
- Deployments — rolling updates, rollbacks, revision history and the progress deadline.
- Service — stable names and addresses in front of a changing set of Pods.
- Resource Management for Pods and Containers — requests, limits, and how the kernel enforces each.
- Configure Liveness, Readiness and Startup Probes — the three probes and how to protect a slow-starting container.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Kubernetes?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/kubernetes/