

Kubernetes for Application Developers

Version 1.37 · verified 2026-11-02

What a service author needs on Kubernetes 1.37 — Deployments, Services, Gateway API, probes, requests, HPA, rollouts, RBAC and the kubectl debug loop.

Contents

Mental model 3

Objects and Deployments 3

Services and Gateway API 5

ConfigMaps and Secrets 7

Probes 8

Requests, limits and autoscaling 10

Rollouts and rollbacks 11

Namespaces and RBAC 13

The debugging loop 14
Kustomize and Helm 15
Further reading 16

Mental model

You never run anything on Kubernetes directly: you write objects that declare a desired state, and controllers keep nudging the cluster towards it. A Deployment controller creates ReplicaSets, which create Pods, which the scheduler places and the kubelet runs and probes. When reality does not converge, the reason is recorded as an event on the object that got stuck — so `kubectl describe` is the first question, not the last.

Objects and Deployments

Object	Owns
Pod	One or more containers sharing a network
ReplicaSet	N identical Pods
Deployment	ReplicaSets, rollouts, rollbacks
Labels and selectors	Which Pods an object manages

Never create bare Pods for a service: nothing recreates them when a node dies. A Deployment finds its Pods by

label, and in `apps/v1` its `spec.selector` must be set explicitly and is immutable after creation.

```yaml
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: ghcr.io/acme/web:1.4.2
          ports: [{ containerPort: 8080 }]
```

Services and Gateway API

Kind	Purpose
Service `ClusterIP` (default)	Stable in-cluster name and IP
Service `LoadBalancer`	Cloud load balancer per Service
Gateway	A listener, owned by the platform team
HTTPRoute	Your routing rules to Services
Ingress	Older API, now frozen

A Service load-balances across ready Pods matching its selector. For HTTP from outside, the project recommends Gateway API over Ingress: the Ingress API is frozen (not removed), and the Ingress NGINX controller was retired in March 2026 with no further releases or security fixes. Gateway API kinds are CRDs installed with your chosen implementation.

```yaml
apiVersion: v1
kind: Service
metadata: { name: web }
spec:
  selector: { app: web }
  ports: [{ port: 80, targetPort: 8080 }]
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata: { name: web }
spec:
  parentRefs: [{ name: public-gw }]
  hostnames: ["shop.example.com"]
  rules:
    - backendRefs: [{ name: web, port: 80 }]
```

> **Warning:** Existing Ingress NGINX installs keep working, which is why they go unnoticed. Check with `kubectl get pods -A -l app.kubernetes.io/name=ingress-nginx` and plan a migration.

ConfigMaps and Secrets

Command	Creates
`kubectl create configmap NAME --from-file=…`	ConfigMap
`kubectl create secret generic NAME --from-literal=k=v`	Secret
`envFrom: [{ configMapRef: … }]`	All keys as env vars
`volumes: [{ secret: … }]`	Keys as files

Keep config out of images so one image runs in every environment. ConfigMap values mounted as files are refreshed in running Pods (except `subPath` mounts);

values injected as environment variables are read once, at container start.

```
kubectl create secret generic db \
  --from-literal=password='s3cr3t'
kubectl get secret db \
  -o jsonpath='{.data.password}' | base64 -d
# prints: s3cr3t
```

> **Gotcha:** Base64 is encoding, not encryption, and Secrets are stored unencrypted in etcd by default. Anyone allowed to `get` the Secret, or to create a Pod in its namespace, can read it.

Probes

Probe	On failure
`startupProbe`	Kill and restart; holds the others off
`livenessProbe`	Kill and restart the container

Probe	On failure
`readinessProbe`	Remove Pod from Service endpoints

Defaults: every 10 s (`periodSeconds`), 1 s timeout, 3 failures in a row. Readiness runs for the Pod's whole life, so a Pod can drop out of traffic while overloaded and come back without a restart. Liveness does not wait for readiness to pass.

```
containers:
  - name: web
    startupProbe:
      httpGet: { path: /healthz, port: 8080 }
      failureThreshold: 30   # up to 300 s to
start
    livenessProbe:
      httpGet: { path: /healthz, port: 8080 }
    readinessProbe:
      httpGet: { path: /ready, port: 8080 }
```

> **Gotcha:** A slow-starting app with only a liveness probe is killed before it finishes booting, forever — `CrashLoopBackOff` with no app error. Add a `startupProbe` instead of a large `initialDelaySeconds`.

Requests, limits and autoscaling

Setting	Effect
`requests.cpu` / `.memory`	Scheduling; HPA percentages
`limits.cpu`	Throttled above it
`limits.memory`	Killed (`OOMKilled`) above it
HorizontalPodAutoscaler	Scales `replicas` on metrics

The scheduler places Pods by requests, not actual use; a Pod that fits nowhere stays `Pending` with a `FailedScheduling` event. HPA computes utilisation as a

percentage of the request and needs the Metrics Server add-on. Scale-down waits out a 5-minute stabilisation window by default.

```
resources:
  requests: { cpu: 250m, memory: 256Mi }
  limits: { memory: 512Mi }
```

> **Gotcha:** If a container has no CPU request, CPU utilisation for the Pod is undefined and the HPA silently takes no action on that metric.

Rollouts and rollbacks

Command	Does
`kubectl set image deploy/web web=…:1.4.3`	Start a rollout
`kubectl rollout status deploy/web`	Wait for it

Command	Does
`kubectl rollout history deploy/web`	List revisions
`kubectl rollout undo deploy/web`	Back to previous
`kubectl rollout restart deploy/web`	Re-create Pods

Only a change to `.spec.template` starts a rollout; scaling does not. The default `RollingUpdate` keeps at least 75% of desired Pods available and at most 125% running (25% `maxUnavailable` and `maxSurge`). Ten old ReplicaSets are kept for rollback by default.

```
kubectl set image deploy/web \
  web=ghcr.io/acme/web:1.4.3
kubectl rollout status deploy/web --timeout=5m \
  || kubectl rollout undo deploy/web
```

> **Gotcha:** Changing a ConfigMap does not restart anything, because the Pod template did not change. Use `rollout restart`, or let Kustomize's hash-suffixed generated names change the template.

Namespaces and RBAC

Object	Scope
Namespace	Name scope for most objects
Role / RoleBinding	Permissions in one namespace
ClusterRole / ClusterRoleBinding	Cluster-wide
ServiceAccount	Identity of your Pods

RBAC is additive: there are no deny rules, only grants. Give each workload its own ServiceAccount and a Role with just the verbs it needs, then check the result rather than guessing.

```
kubectl auth can-i list secrets -n shop
kubectl auth can-i get pods -n shop \
  --as=system:serviceaccount:shop:web
```

The debugging loop

Symptom	First command
`Pending`	`kubectl describe pod` (events)
`ImagePullBackOff`	`kubectl describe pod`
`CrashLoopBackOff`	`kubectl logs POD --previous`
Running but wrong	`kubectl exec` or `kubectl debug`
No traffic	Readiness, Service selector, endpoints

Go from the top of the chain down: Deployment, ReplicaSet, Pod, container. `describe` shows recent events; `kubectl events` streams them. Distroless images have no shell, so attach an ephemeral debug container that shares the target's process namespace.

```
kubectl get pods -l app=web -o wide
kubectl describe pod web-7d9c-xk2lp
kubectl events --for pod/web-7d9c-xk2lp
kubectl logs web-7d9c-xk2lp --previous
kubectl debug -it web-7d9c-xk2lp \
  --image=busybox --target=web
kubectl port-forward svc/web 8080:80
```

Kustomize and Helm

Tool	Model
Kustomize	Plain YAML base plus overlays; built into `kubectl -k`
Helm	Templated chart plus values; installs a release

Pick one per repo. Kustomize patches real manifests and needs no extra binary; Helm suits third-party software you install with values (`helm upgrade --install`). Both end as ordinary objects that the debugging loop above applies to.

```
# overlays/prod/kustomization.yaml
resources:
  - ../../base
images:
  - name: ghcr.io/acme/web
    newTag: "1.4.3"
# kubectl apply -k overlays/prod
```

Further reading

- Kubernetes documentation
- Gateway API guides
- Ingress NGINX retirement
- Debug running Pods
- kubectl reference