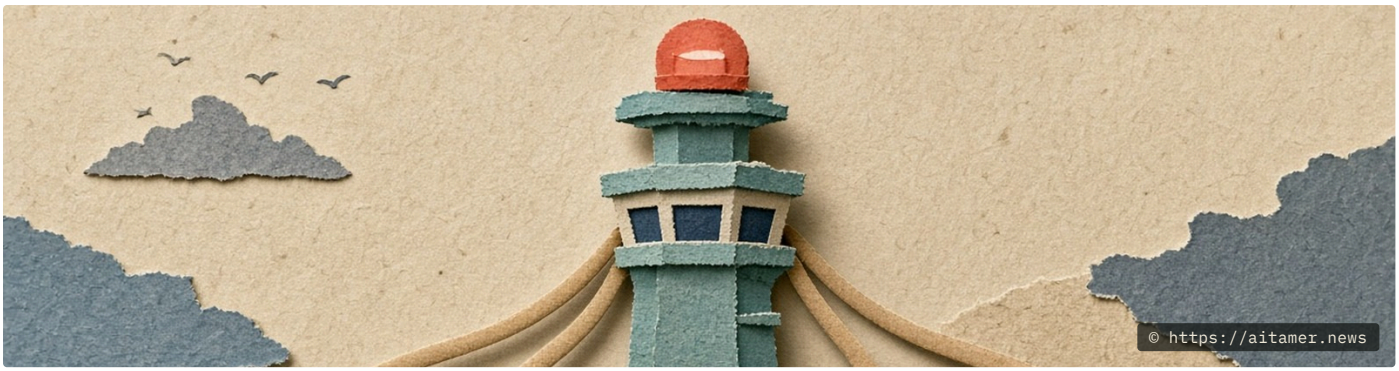


KUBERNETES FROM ZERO · FOUNDATIONS 2 OF 12 · LESSON 2 · LESSON KIT · CHEAT SHEET

Kubernetes from Zero, part 2: a tour of the control plane and the nodes

Everything from Lesson 2 on three pages: what each component does, where it runs, the path of one `kubectl apply`, and the commands from the lab.

Ran every command on kind v0.33.0, Kubernetes v1.37.0 node, kubectl v1.37.2, 10 Nov 2026. Cluster: k8s-from-zero.

THE ONE RULE

Control-plane components and kubelets coordinate through the API server. The kubelet also talks to containerd over CRI, and the API server connects to kubelets for logs, exec and port-forward. Only the API server reads and writes etcd.

Control plane the cluster's brain: global decisions, reacting to events

COMPONENT	ROLE	WHERE IT RUNS
kube-apiserver	The front door. Exposes the HTTP API, checks who you are and whether you are allowed, validates what you send and saves it. You can run several copies behind a load balancer.	Control plane. In kind: static Pod `kube-apiserver-k8s-from-zero-control-plane` in `kube-system`.
etcd	The consistent key-value store for all cluster data. If etcd is lost without a backup, the cluster forgets everything it was told.	Control plane. In kind: static Pod `etcd-k8s-from-zero-control-plane`. Only the API server talks to it.
kube-scheduler	Watches for Pods that have no node yet and picks one for each, then writes `spec.nodeName`. It only decides; it does not start anything.	Control plane. In kind: static Pod `kube-scheduler-k8s-from-zero-control-plane`.
kube-controller-manager	Runs the built-in controllers (Node, Job, EndpointSlice, ServiceAccount, Deployment, ReplicaSet and more), compiled into one binary and run as one process.	Control plane. In kind: static Pod `kube-controller-manager-k8s-from-zero-control-plane`.
cloud-controller-manager (optional)	Links the cluster to a cloud provider's API, for example to create a cloud load balancer.	Control plane, in clouds only. A cluster on your laptop does not have one.

Nodes where your containers run

COMPONENT	ROLE	WHERE IT RUNS
kubelet	The node agent. It makes sure the containers in Pods assigned to its node are running and healthy.	Every node, as a system service. It also runs the static Pod files in `/etc/kubernetes/manifests/`.
Container runtime (containerd)	Actually runs containers. The kubelet talks to it through the Container Runtime Interface (CRI), a gRPC API, so the runtime is swappable.	Every node, as a system service. kind nodes use containerd.
kube-proxy (optional)	Maintains network rules on the node so that Services work. Some network plugins replace it.	Usually every node. In kind: a Pod in `kube-system`.
Add-ons	Cluster DNS (CoreDNS), the network plugin that gives every Pod an IP (kindnet in kind), and in kind a small storage provisioner.	Ordinary Pods. In kind, CoreDNS and kindnet are in `kube-system`.

In kind, one node (`k8s-from-zero-control-plane`) is the control plane and also runs your workloads. A kind node is a Docker container, so `docker exec k8s-from-zero-control-plane ...` gets you inside it. kubeadm runs the four control-plane parts as static Pods: Pod files the kubelet reads straight from a folder on the node, without asking the API server.

SCAN FOR THE LESSON

Lesson: **Kubernetes from Zero, part 2: a tour of the control plane and the nodes**

https://aitamer.news/posts/k8s-from-zero-02-architecture-tour/?utm_source=lessonkit&utm_medium=pdf&utm_campaign=k8s-02&utm_content=cheatsheet

NEXT IN THE SERIES **Next: Lesson 3, your local lab**. Install kubectl and kind properly, compare them with minikube, k3d and Docker Desktop, and learn how kubeconfig and contexts decide which cluster you are talking to. Lesson page

Free to share with credit: © aitamer.news

Life of one `kubectl apply`

Deployment `web`, 2 replicas. Every arrow goes through the API server.

1	**kubectl**	Reads `web.yaml`, asks the API server whether Deployment `web` exists (`404 Not Found`), then sends a `POST` (`201 Created`). The API server authenticates you, validates the object and stores it in etcd.
2	**Deployment controller**	Inside kube-controller-manager. Sees the new Deployment and creates a ReplicaSet. Event: `deployment-controller ScalingReplicaSet`.
3	**ReplicaSet controller**	Sees a ReplicaSet that wants 2 Pods and has 0, so creates two Pod objects: records only, no node, no container. Event: `replicaset-controller SuccessfulCreate`.
4	**kube-scheduler**	Sees Pods with an empty `spec.nodeName`, scores the nodes and binds each Pod by writing a node name into it. Event: `default-scheduler Scheduled`.
5	**kubelet**	Sees Pods assigned to its node and asks containerd to pull the image (or finds it cached), create the container and start it. Events: `kubelet Pulled`, `Created`, `Started`.
6	**kubelet**	Reports the Pod's status, `Running`, back to the API server. `kubectl get pods` now shows it.

Ownership. Each Pod carries an `ownerReference` to its ReplicaSet, and the ReplicaSet one to its Deployment. That chain is how deleting a Deployment cleans up everything below it.

Key commands

commands verbatim from the lesson

Cluster

```
kind create cluster --name k8s-from-zero
```
the cluster from Lesson 1

```
kind delete cluster --name k8s-from-zero
```
remove it when you are done

Tour the cluster

```
kubectl get nodes -o wide
```
`CONTAINER-RUNTIME` column

```
kubectl get pods -n kube-system -o wide
```
static Pod names end in the node name

```
docker exec k8s-from-zero-control-plane ls \
  /etc/kubernetes/manifests
```
the static Pod files

```
docker exec k8s-from-zero-control-plane systemctl is-active \
  kubelet containerd
```
system services on the node

```
docker exec k8s-from-zero-control-plane crictl ps
```
containers as the runtime sees them

```
kubectl get --raw='/readyz?verbose' | tail -3
```
API server health; also `/livez`

```
kubectl api-resources
```
All resource types; shows short names and API groups

```
kubectl get leases -n kube-system
```
who leads the scheduler and controllers

Peek into etcd (learning only)

```
kubectl -n kube-system exec etcd-k8s-from-zero-control-plane \
  -- etcdctl \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  get /registry/deployments/default/web --keys-only
```
Local kind cluster only, on your own machine. This command only reads one key. kubeadm's default certificate paths.

Switch the scheduler off, then on

```
docker exec k8s-from-zero-control-plane mv \
  /etc/kubernetes/manifests/kube-scheduler.yaml /root/
```
Local kind cluster only. The kubelet stops the scheduler. Move the file back into /etc/kubernetes/manifests/ to restore it.

```
kubectl get pods -n kube-system -l component=kube-scheduler
```
is the scheduler running?

```
kubectl scale deployment web --replicas=4
```
new Pods stay `Pending`

```
kubectl get pods -l app.kubernetes.io/name=web
```
the `web` Pods and their `STATUS`

```
docker exec k8s-from-zero-control-plane mv \
  /root/kube-scheduler.yaml /etc/kubernetes/manifests/
```
the scheduler catches up

```
kubectl delete -f web.yaml
```
clean up

When it goes wrong

```
kubectl describe node <name>
```
read `Conditions`

```
kubectl <command> -v=6
```
see kubectl's HTTP calls; `-v=8` adds request contents

```
journalctl -u kubelet
```
on a real node, when it turns `NotReady`

SCAN FOR THE LESSON

Lesson: **Kubernetes from Zero, part 2: a tour of the control plane and the nodes**

https://aitamer.news/posts/k8s-from-zero-02-architecture-tour/?utm_source=lessonkit&utm_medium=pdf&utm_campaign=k8s-02&utm_content=cheatsheet

NEXT IN THE SERIES **Next: Lesson 3, your local lab.** Install kubectl and kind properly, compare them with minikube, k3d and Docker Desktop, and learn how kubeconfig and contexts decide which cluster you are talking to. Lesson page

Free to share with credit: © aitamer.news

Follow one apply

```
kubectl get events --watch-only -o \
  custom-columns=FROM:.source.component,REASON:.reason,KIND:.involvedObject.kind,NAME:.involvedObject.name
```
second terminal, start it first

```
kubectl apply -f web.yaml -v=6 2>&1 | grep 'verb='
```
each API request and its response code

```
POD=$(kubectl get pods -l app.kubernetes.io/name=web -o name | head -1)
```
looks up one Pod from this Deployment. The sample log shows pwjwm and vfjt9. Your suffix will differ.

```
kubectl get "$POD" -o \
  jsonpath='{.spec.nodeName}{"\n"}{.metadata.ownerReferences[0].kind}/{.metadata.ownerReferences[0].name}{"\n"}'
```
the node and the owner; your suffix will differ

```
kubectl get replicaset -l app.kubernetes.io/name=web -o \
  jsonpath='{.items[0].metadata.ownerReferences[0].kind}/{.items[0].metadata.ownerReferences[0].name}{"\n"}'
```
the ReplicaSet's owner

SCAN FOR THE LESSON

Lesson: **Kubernetes from Zero, part 2: a tour of the control plane and the nodes**

https://aitamer.news/posts/k8s-from-zero-02-architecture-tour/?utm_source=lessonkit&utm_medium=pdf&utm_campaign=k8s-02&utm_content=cheatsheet

NEXT IN THE SERIES **Next: Lesson 3, your local lab**. Install kubectl and kind properly, compare them with minikube, k3d and Docker Desktop, and learn how kubeconfig and contexts decide which cluster you are talking to. Lesson page

Free to share with credit: © aitamer.news