

KUBERNETES FROM ZERO · FOUNDATIONS 2 OF 12 · LESSON 2 · LESSON KIT · ANSWER KEY

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

The answers to the homework sheet, matched by exercise number. Every answer comes from the lesson. Names with random suffixes will differ on your cluster.

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

Recall

1. Name the four core control-plane components, and the one optional one.

   kube-apiserver, etcd, kube-scheduler and kube-controller-manager. The optional one is cloud-controller-manager, which links the cluster to a cloud provider's API.

2. Which component is the only one that reads and writes etcd? How do the other components learn about changes?

   kube-apiserver. Control-plane components and kubelets watch the API server for changes and write back what they did. The kubelet also talks to the container runtime over CRI, and the API server connects to kubelets for logs, exec and port-forward.

3. What does every node run? Which node component is optional?

   A kubelet and a container runtime (containerd in kind), and usually kube-proxy. kube-proxy is the optional one; some network plugins replace it.

4. The kubelet talks to the container runtime through an interface. What is it called, what kind of API is it, and why does it matter?

   The Container Runtime Interface (CRI), a gRPC API. Because the kubelet only speaks CRI, the runtime is swappable.

5. What is a static Pod? Which folder holds the static Pod files on a kubeadm node?

   A Pod file that the kubelet reads straight from a folder on the node, without asking the API server. The folder is `/etc/kubernetes/manifests/`.

6. Which field does kube-scheduler write into a Pod? Does the scheduler start the container?

   `spec.nodeName`. No: the scheduler only decides. The kubelet on that node starts the container.

7. Name the add-ons in a kind cluster. How do they run?

   CoreDNS (cluster DNS), kindnet (the network plugin that gives every Pod an IP) and a small storage provisioner. They run as ordinary Pods.

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=answer-key

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

Run this and explain

8 Run `kubectl get nodes -o wide`. Which column names the container runtime, and what does it say?

> `CONTAINER-RUNTIME`. It shows containerd; the lesson's run showed `containerd://2.3.4`. The single node `k8s-from-zero-control-plane` is the control plane and also runs your workloads.

9 Run `kubectl get pods -n kube-system -o wide`. Which Pod names end in the node name, and why?

> `etcd-`, `kube-apiserver-`, `kube-controller-manager-` and `kube-scheduler-k8s-from-zero-control-plane`. They are static Pods, and static Pod names end in the node name. The others (coredns, kindnet, kube-proxy) have random suffixes.

10 Run `docker exec k8s-from-zero-control-plane ls /etc/kubernetes/manifests` and `docker exec k8s-from-zero-control-plane systemctl is-active kubelet containerd`. What do you see, and what does it tell you about how the node is built?

> Four files: `etcd.yaml`, `kube-apiserver.yaml`, `kube-controller-manager.yaml`, `kube-scheduler.yaml`, then `active` twice. The kubelet and containerd are ordinary system services on the node. The four control-plane parts are files in a folder that the kubelet runs as static Pods.

11 Run `docker exec k8s-from-zero-control-plane crictl ps`. What is `crictl`, and what do the rows show?

> A debugging client for any CRI runtime. The rows are the same components (etcd, kube-apiserver, kube-controller-manager, kube-scheduler, coredns and more), seen from below: containers that containerd runs.

12 Run `kubectl get leases -n kube-system`. Which two components hold a Lease, and why do they need one?

> kube-controller-manager and kube-scheduler (you also see an `apiserver-...` lease). In a cluster with several control-plane machines, only one scheduler and one controller manager act at a time; the Lease proves which one is in charge.

13 Start the event watch in a second terminal, then run `kubectl apply -f web.yaml -v=6 2>&1 | grep 'verb='`. In the kubectl output, why is there a `404 Not Found` before the `201 Created`?

> kubectl first asked whether a Deployment called `web` exists. It did not (404), so kubectl sent a `POST` that created it (`201 Created`). Before that, the two `GET .../openapi/v3...` calls fetched the API schema.

14 From the same run, list the event sources (`FROM` column) in the order they appear, and say what each one did.

> `deployment-controller`: ScalingReplicaSet (scaled up a ReplicaSet). `replicaset-controller`: SuccessfulCreate, twice (created two Pods). `default-scheduler`: Scheduled, twice (placed each Pod). `kubelet`: Pulled, Created, Started for each Pod (pulled the image, created and started the container).

15 Run the commands from step 9 of the lesson. What three lines come back from the two jsonpath commands, and which component wrote the first one?

> The node name `k8s-from-zero-control-plane`, then `ReplicaSet/web-<hash>`, then `Deployment/web`. The scheduler wrote `spec.nodeName`. The Pod is owned by the ReplicaSet, which is owned by the Deployment.

16 Run the `etcdctl ... get /registry/deployments/default/web --keys-only` command from step 10. What comes back, and which component wrote it?

> The key `/registry/deployments/default/web`. Your Deployment is a key in etcd, written there by the API server. Run this only on the local kind cluster from the lesson. The command only reads one key. In real life you never touch etcd by hand.

Free to share with credit: © aitamer.news

Predict the outcome

17 You move `kube-scheduler.yaml` out of `/etc/kubernetes/manifests` and run `kubectl scale deployment web --replicas=4`. Predict the `STATUS` of all four `web` Pods, and what `kubectl describe` shows on a new one.

> Two old Pods stay `Running`. Two new Pods exist (the controllers still did their part) but stay `Pending`, because nobody gave them a node. `kubectl describe` shows `Node: <none>` and no events at all.

18 You then move `kube-scheduler.yaml` back into `/etc/kubernetes/manifests`. What happens to the two `Pending` Pods?

> The kubelet starts the scheduler again, the scheduler catches up and binds them, and all four Pods become `Running`.

19 You stop the kubelet with `docker exec k8s-from-zero-control-plane systemctl stop kubelet`. Predict the node's `STATUS` after a short wait, what its conditions say, and whether running containers stop.

> The node turns `NotReady` (47 seconds later in the lesson). `kubectl describe node` shows every condition as `Unknown` with `Kubelet stopped posting node status.` Running containers keep running, which is why kubectl still works. `systemctl start kubelet` brings it back to `Ready`.

20 You run `kubectl delete -f web.yaml`. What happens to the ReplicaSet and the Pods, and what makes that happen?

> They are removed too. Each Pod has an `ownerReference` to its ReplicaSet and the ReplicaSet one to its Deployment; that ownership chain is how deleting a Deployment cleans up everything below it.

21 New Pods stay `Pending` and show no events. Which component do you suspect, and which command checks it?

> The scheduler: nothing scheduled the Pods. Check it with `kubectl get pods -n kube-system -l component=kube-scheduler`. If it runs, `kubectl describe pod` usually shows a `FailedScheduling` event with the reason.

22 kubectl prints `The connection to the server ... was refused`. Give the two likely causes from the lesson.

> kubectl cannot reach the API server: the cluster is stopped, or kubectl points at the wrong cluster.

Free to share with credit: © aitamer.news