



PUBLISHED · OCTOBER 1, 2026

High-Availability Control Plane Architecture and Configuration

Design and build a highly available kubeadm control plane by combining a stable API endpoint, redundant control-plane components, and an etcd quorum.



This is Learn post 21 of the 54-post Certified Kubernetes Administrator (CKA) preparation path. Earlier lessons built, joined, and upgraded a kubeadm cluster. This lesson removes the first control-plane node as the cluster's single point of failure.

High availability (HA) is not produced by copying the API server three times. Clients need a stable route to healthy API servers, control loops need a surviving leader, and etcd must retain a voting majority. Because these are separate failure boundaries, an

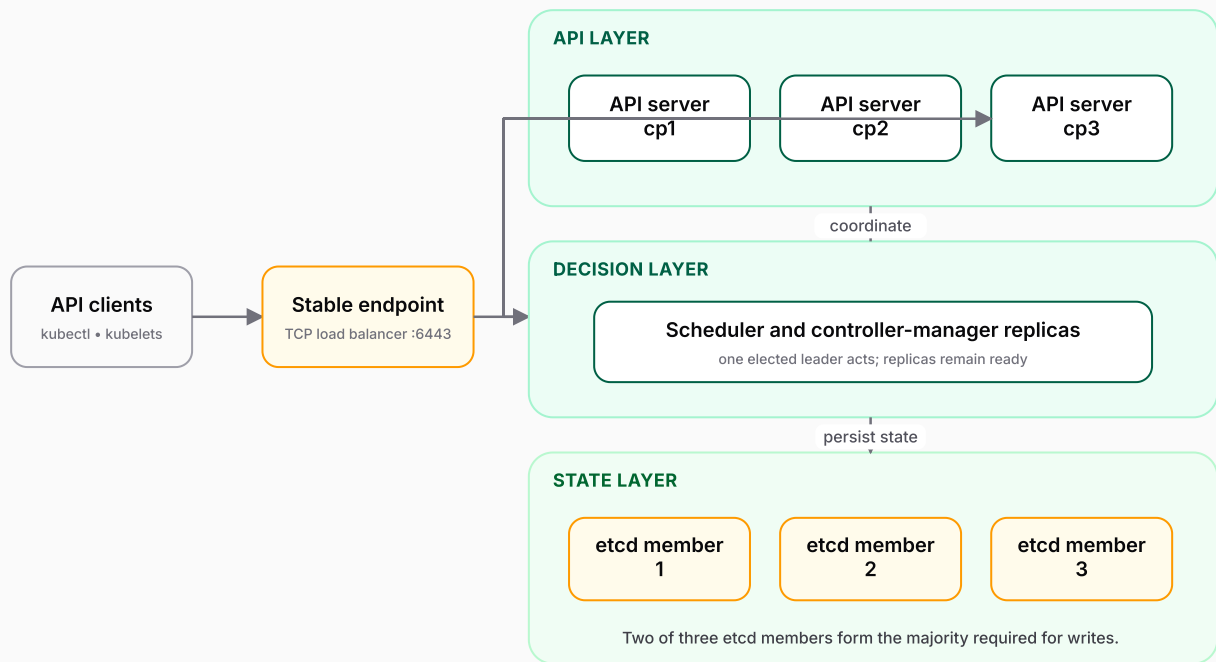
administrator must design and observe all three.

What you'll learn

- Explain how a load balancer, replicated API servers, leader-elected controllers, and an etcd quorum work together.
- Distinguish stacked etcd from external etcd and choose between their failure boundaries.
- Initialize a three-node stacked control plane with a stable kubeadm controlPlaneEndpoint.
- Join additional control-plane nodes securely and understand what kubeadm creates on each node.
- Verify API readiness, control-plane placement, leader election, and etcd membership with administrator commands.

The mental model: one doorway, many API servers, one agreed state

Clients enter through one stable endpoint. Any healthy API server can answer. One scheduler and one controller-manager replica lead each decision loop. A majority of etcd members must agree before cluster state changes.



A highly available control plane has three availability layers

Redundancy at one layer cannot compensate for failure at another: routing, control-loop leadership, and durable state must each remain available.

The API layer is active-active

kubect!, kubelets, controllers, and other clients need one durable server address. A layer-4 load balancer listens on that address and forwards TCP connections to healthy kube-apiserver instances. Because API servers are designed to serve concurrently, the load balancer may route different requests to different replicas.

The stable endpoint and each node's local API endpoint are not interchangeable. `controlPlaneEndpoint` names the shared front door. `localAPIEndpoint.advertiseAddress` identifies one API server instance. Because clients must survive a node failure, their kubeconfigs should use the shared endpoint rather than cp1's address.

The decision layer is active-standby

Every control-plane node runs kube-scheduler and kube-controller-manager, but duplicate schedulers must not bind the same pending Pod independently and duplicate controllers must not race to act. Their replicas therefore compete for Lease objects. The holder performs the control loop; the others remain ready to acquire leadership if the holder stops renewing its Lease.

That distinction matters: API-server capacity is load balanced, while scheduler and controller-manager continuity comes from leader failover. Several running replicas do not mean several replicas are simultaneously making the same decision.

The state layer requires quorum

etcd is the authoritative store behind the Kubernetes API. Its members use consensus to agree on an ordered state. A three-member cluster needs two available members to commit writes; a five-member cluster needs three. This is why odd member counts are useful: a fourth member adds operating cost but does not increase the number of failures the cluster can tolerate.

With three healthy members, losing one leaves a two-member majority. Losing two removes quorum: an API server process may still be reachable, but operations that need consistent state cannot proceed normally. More API servers cannot repair a missing etcd majority.

What HA does—and does not—protect

If one control-plane node fails in a healthy three-node stacked topology, the load balancer removes its API server, another scheduler or controller-manager replica can become leader, and two etcd members retain quorum. Because all three conditions still hold, the control plane continues serving.

HA reduces interruption from component or node failure. It is not a backup, protection from an incorrect cluster-wide change, or proof that workloads are redundant. Existing containers may continue running during a control-plane outage because kubelet and the container runtime are node-local, but scheduling, reconciliation, and API-driven administration are impaired until the control plane returns.

Choose the etcd topology

Stacked etcd

In kubeadm's default HA topology, every control-plane node also hosts one local etcd member. Three machines therefore provide three API servers, three sets of decision components, and three etcd members. The design is economical and easier to operate, but one machine failure removes both a control-plane replica and an etcd vote.

External etcd

An external topology places etcd on a separate three-member host set. A control-plane node failure then does not also remove an etcd member, but the cluster needs at least six machines for three control-plane instances and three etcd members. It also creates a separate etcd lifecycle, network path, certificate set, backup plan, and monitoring responsibility.

For a compact kubeadm installation, three stacked control-plane nodes are the common starting point. Choose external etcd when decoupled failure domains justify the extra infrastructure and operational work.

Worked configuration: a three-node stacked control plane

The demonstration uses three prepared Linux hosts and an existing TCP load balancer. The shared DNS name `api.cka.example.com` resolves to `192.0.2.10`. The load balancer forwards port 6443 to `cp1` at `192.0.2.11`, `cp2` at `192.0.2.12`, and `cp3` at `192.0.2.13`. Replace all example addresses with routable values from your environment.

- All three hosts have unique names, stable addresses, matching Kubernetes versions, a working container runtime, kubelet, and kubeadm.
- Each host can reach the shared endpoint and the required peer ports; the load balancer can reach every API-server backend on 6443.
- The load balancer is itself redundant or provided as a managed service; otherwise the cluster merely moves its single point of failure in front of the API servers.

1. Prove the API path before bootstrap

From `cp1`, test the shared endpoint before an API server exists. This checks name resolution and network routing separately from Kubernetes.

```
cp1 ~$ bash
```

```
getent hosts api.cka.example.com  
nc -zv -w 2 api.cka.example.com 6443
```

A connection refusal can be expected before `kube-apiserver` starts: the packet reached a backend, but nothing listened on 6443. A timeout points to DNS, routing, firewall, load-balancer listener, or backend connectivity and should be fixed before `kubeadm init`.

2. Describe both the node and the cluster endpoint

```
/root/kubeadm-config.yaml on cp1 · yaml
```

```
apiVersion: kubeadm.k8s.io/v1beta4
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.0.2.11
  bindPort: 6443
nodeRegistration:
  criSocket: unix:///run/containerd/containerd.sock
---
apiVersion: kubeadm.k8s.io/v1beta4
kind: ClusterConfiguration
kubernetesVersion: v1.37.0
controlPlaneEndpoint: api.cka.example.com:6443
networking:
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12
```

InitConfiguration is local to cp1, so its advertise address is cp1's address.

ClusterConfiguration is shared cluster intent, so controlPlaneEndpoint is the load-balanced address. kubeadm also includes that shared name in the API server certificate and writes it into generated kubeconfigs.

Plan the shared endpoint before initialization. kubeadm does not support converting a cluster created without controlPlaneEndpoint into an HA cluster later.

Check the configuration with kubeadm's preflight phase. This validates the host and config without attempting the full initialization.

```
cp1 · bash
```

```
sudo kubeadm init phase preflight --config /root/kubeadm-config.yaml
```

3. Initialize cp1 and stage shared certificates

cp1 · bash

```
sudo kubeadm init \  
--config /root/kubeadm-config.yaml \  
--upload-certs
```

The bootstrap sequence is familiar from post 18, with one important extension. `--upload-certs` encrypts the shared control-plane certificate material in the temporary `kubeadm-certs` Secret. `kubeadm` prints a decryption key and a control-plane join command so `cp2` and `cp3` can obtain that material securely.

shape of kubeadm init output · bash

```
kubeadm join api.cka.example.com:6443 \  
--token <bootstrap-token> \  
--discovery-token-ca-cert-hash sha256:<ca-public-key-hash> \  
--control-plane \  
--certificate-key <certificate-key>
```

The token temporarily authenticates bootstrap, the CA hash pins discovery to the intended cluster, `--control-plane` selects the control-plane join workflow, and the certificate key decrypts the uploaded certificate bundle. Treat the complete command as sensitive. The uploaded certificates and key expire after two hours by default.

4. Install networking, then join cp2 and cp3

Install the cluster's chosen Container Network Interface (CNI) add-on using the workflow from post 18. Then run the generated control-plane join command on `cp2`. Wait for it to become healthy before running the same form of command on `cp3`. Sequential joins keep every transition observable and avoid changing `etcd` membership in parallel.

On each joining node, kubeadm downloads the shared certificate material, generates node-specific certificates and kubeconfigs, writes static Pod manifests for the API server, scheduler, and controller-manager, and joins a local etcd member. Kubelet starts those manifests and publishes mirror Pods through the API as described in post 15.

If the two-hour upload window has closed, generate a new encrypted upload and a fresh join command from an existing control-plane node:

```
existing control-plane node · bash
```

```
CERTIFICATE_KEY=$(sudo kubeadm init phase upload-certs --upload-certs | tail -n 1)
```

```
sudo kubeadm token create \  
--print-join-command \  
--certificate-key "$CERTIFICATE_KEY"
```

The first command prints a new decryption key after re-uploading the bundle. The second prints a new control-plane join command. Do not store either value in a public script, shell transcript, or source repository.

Read the finished control plane

Confirm node and static-Pod placement

```
admin workstation or control-plane node · bash
```

```
kubectl get nodes \  
-I node-role.kubernetes.io/control-plane \  
-o wide
```

```
kubectl -n kube-system get pods \  
-I tier=control-plane \  
-o custom-columns='NAME:.metadata.name,NODE:.spec.nodeName,STATUS:.status.phase'
```

representative result · plaintext

NAME	NODE	STATUS
etcd-cp1	cp1	Running
etcd-cp2	cp2	Running
etcd-cp3	cp3	Running
kube-apiserver-cp1	cp1	Running
kube-apiserver-cp2	cp2	Running
kube-apiserver-cp3	cp3	Running
kube-controller-manager-cp1	cp1	Running
kube-controller-manager-cp2	cp2	Running
kube-controller-manager-cp3	cp3	Running
kube-scheduler-cp1	cp1	Running
kube-scheduler-cp2	cp2	Running
kube-scheduler-cp3	cp3	Running

The important pattern is one instance of every component on every control-plane node. Running only proves the local static Pods exist; the next checks verify that clients use the shared route and that the replicated components coordinate correctly.

Confirm the shared API endpoint and readiness

admin workstation or control-plane node · bash

```
kubecttl config view --minify \
-o jsonpath='{.clusters[0].cluster.server}'{"\n"}'

kubecttl get --raw=/readyz?verbose'
```

The server should be `https://api.cka.example.com:6443`, not an individual node. The verbose readyz response should report successful checks, including etcd readiness. Humans use the verbose body for diagnosis; a load balancer should make its routing decision from the health check's HTTP status or an appropriately configured TCP check.

Observe leader election

```
admin workstation or control-plane node · bash
```

```
kubectl -n kube-system get lease \
  kube-controller-manager kube-scheduler \
  -o custom-columns='LEASE:.metadata.name,HOLDER:.spec.holderIdentity,RENEWED:.spec.renewTime'
```

Each Lease has one current holder identity and a recently renewed timestamp. The holder names can differ between components and can change after restart or failure. That is expected: availability depends on a renewable leadership record, not on cp1 remaining permanent leader.

Verify etcd membership and health

Run etcd checks from a stacked control-plane host where the kubeadm-managed client certificates exist. The explicit endpoint list makes the three expected members visible instead of checking only the local member.

```
cp1 · bash
```

```
ETCD_ENDPOINTS='https://192.0.2.11:2379,https://192.0.2.12:2379,https://192.0.2.13:2379'
```

```
sudo ETCDCTL_API=3 etcdctl \
  --endpoints="$ETCD_ENDPOINTS" \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
  member list --write-out=table
```

```
sudo ETCDCTL_API=3 etcdctl \
  --endpoints="$ETCD_ENDPOINTS" \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
  endpoint status --write-out=table
```

member list should show three started members with distinct peer and client URLs. endpoint status should reach every endpoint, show one current Raft leader, and report similar Raft indexes after replication catches up. Leader identity is not a preferred-node setting; etcd may elect another healthy member later.

If etcdctl is not installed on the host, use an approved administration image or install the matching client according to your environment. Do not invent a Pod-based shortcut that depends on the unhealthy API you may be trying to diagnose.

How external etcd changes kubeadm configuration

With external etcd, provision and secure the etcd cluster first. ClusterConfiguration then tells every API server which endpoints and client credentials to use. kubeadm does not create a local etcd static Pod on the control-plane nodes.

external-etcd portion of ClusterConfiguration · yaml

```
apiVersion: kubeadm.k8s.io/v1beta4
kind: ClusterConfiguration
controlPlaneEndpoint: api.cka.example.com:6443
etcd:
  external:
    endpoints:
      - https://192.0.2.21:2379
      - https://192.0.2.22:2379
      - https://192.0.2.23:2379
    caFile: /etc/kubernetes/pki/etcd/ca.crt
    certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt
    keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key
```

Those files authenticate kube-apiserver as an etcd client; they are not generic administrator credentials. The detailed external-etcd bootstrap procedure is infrastructure-specific and longer than the architectural distinction needed here. The invariant remains: every API server must reach a healthy etcd quorum through trusted endpoints.

Operate by preserving one complete path

During maintenance, change one control-plane node at a time and wait for the system to settle. Verify that the node is Ready, its static Pods are Running, the load-balanced readyz endpoint succeeds, control-component Leases keep renewing, and all expected etcd endpoints respond before moving to the next node.

Spread control-plane members across the independent failure domains your infrastructure actually provides. Three virtual machines on one physical host, one power circuit, or one network path are three processes but not three useful failure boundaries. The same reasoning applies to load-balancer replicas and external etcd hosts.

Back up etcd even when it is replicated. Replication keeps members synchronized, including an accidental deletion; a snapshot provides a separate recovery point. The snapshot and restore workflow was introduced with lifecycle preparation in post 20 and remains a necessary companion to HA.

Distinctions worth remembering

- controlPlaneEndpoint is the cluster's stable API address; advertiseAddress belongs to one API-server instance.
- API servers serve concurrently; scheduler and controller-manager replicas use leader election.
- Three etcd replicas provide redundancy; a two-member majority provides write availability after one failure.
- Stacked etcd couples one control-plane replica and one etcd vote to each node; external etcd separates those failure domains at higher cost.
- --upload-certs is a short-lived, encrypted bootstrap mechanism. It does not replace certificate lifecycle management or secret handling.
- HA keeps a service path available; backups recover past state. A production control plane needs both.

Official documentation

[Creating Highly Available Clusters with kubeadm](#) provides the supported stacked and external-etcd bootstrap workflows.

[Options for Highly Available Topology](#) compares the failure boundaries and infrastructure cost of both etcd layouts.

[kubeadm Configuration v1beta4](#) defines controlPlaneEndpoint, localAPIEndpoint, and external etcd fields.

[Kubernetes API health endpoints](#) explains livez, readyz, and their administrator-facing verbose output.

[Set up a High Availability etcd Cluster with kubeadm](#) is the detailed procedure when external etcd is the chosen topology.