Omar Latrach

TECHNICAL ARTICLE

PUBLISHED · SEPTEMBER 29, 2026

Joining and Managing Worker Nodes with kubeadm

Join Linux workers to a kubeadm cluster securely, understand discovery and TLS bootstrap, verify Node readiness, perform maintenance, and remove or rejoin nodes safely.

This is Learn post 19 of the 54-post Certified Kubernetes Administrator (CKA) preparation path. The previous lesson created a working control plane. This lesson adds Linux worker capacity and follows each worker from an untrusted host to a healthy Kubernetes Node.

A join command looks short, but it establishes trust in both directions. The host must prove that it reached the intended cluster, and the kubelet must obtain an identity that the API server accepts. Understanding that exchange makes expired tokens, certificate errors, and NotReady nodes much easier to diagnose.

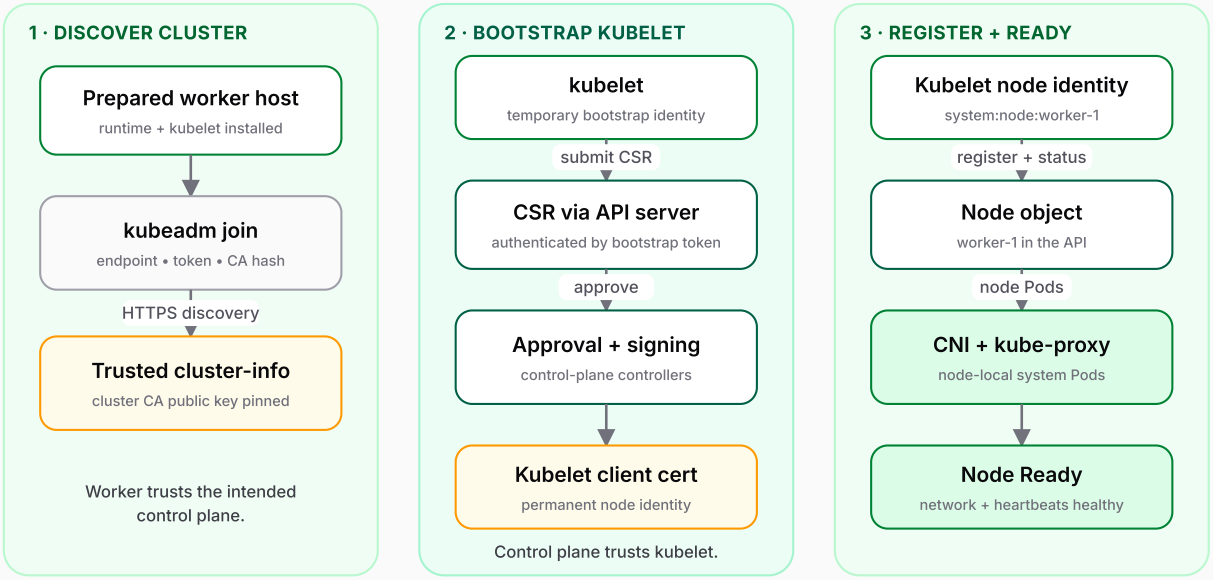
What you'll learn

By the end of this lesson, you will be able to:

- Explain discovery, CA pinning, and kubelet TLS bootstrap as two directions of trust.
- Create a short-lived bootstrap token and join a prepared worker with kubeadm.
- Verify Node registration, readiness, heartbeats, networking, and node-local system Pods.
- Label, cordon, drain, uncordon, remove, and rejoin worker nodes deliberately.

The mental model: join is secure identity bootstrap

> Discovery lets the worker trust the control plane. TLS bootstrap lets the control plane trust the worker. Registration then gives the cluster a Node object to observe and schedule against.



The bootstrap token opens the path; the signed certificate becomes kubelet's lasting identity.

kubeadm join establishes two-way trust before Node readiness

Discovery validates the control plane; TLS bootstrap gives kubelet a signed node identity; registration and node-local networking lead to Ready.

The machine and the Node object are related but different. The machine runs kubelet, the container runtime, and workloads. The Node object is the API representation of that machine: it holds labels, taints, capacity, addresses, and conditions such as `Ready`. Deleting one does not automatically clean the other.

Start with a prepared worker

A worker needs the same host preparation introduced in post 17: a unique hostname, stable addressing, working name resolution, required kernel settings, swap handled according to your kubelet configuration, a Container Runtime Interface (CRI) runtime, kubelet, and kubeadm. It also needs network access to the control-plane API endpoint and the ports used by the chosen CNI plugin.

For this lesson, the control-plane endpoint is `cp1.example.net:6443`, the prepared host is `worker-1`, and containerd uses its standard CRI socket.

```bash
hostnamectl --static
getent hosts cp1.example.net
nc -zv cp1.example.net 6443
sudo systemctl is-active containerd
sudo systemctl is-enabled kubelet
sudo crictl info >/dev/null && echo 'CRI reachable'
kubeadm version -o short
```

The kubelet may restart while it waits for kubeadm to write its configuration; that is expected before the join. The important checks are that containerd is running, the CRI is reachable, kubelet is enabled, and the API endpoint is reachable. For a new join, use the same kubeadm minor version that most recently initialized or upgraded the cluster. Version upgrades belong to the next lesson.

Read the join command as three security inputs

```bash
sudo kubeadm join cp1.example.net:6443 \
  --token 4d7nq2.3rv6xap9m2c8kf5h \
  --discovery-token-ca-cert-hash sha256:0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef \
  --cri-socket unix:///run/containerd/containerd.sock
```

The endpoint tells the host where to find the API server. In a highly available design it normally names a stable load-balancer endpoint, but that architecture is reserved for post 21.

The bootstrap token is a short-lived secret. It temporarily authenticates discovery and, by default, the kubelet's TLS bootstrap request. The token created by `kubeadm init` expires after 24 hours unless configured otherwise.

The CA public-key hash pins the cluster's certificate authority. The worker uses it to reject an API endpoint that cannot prove the expected root of trust. Do not replace this check with `--discovery-token-unsafe-skip-ca-verification` in a normal workflow.

The token is secret; the CA hash is a verifier, not a credential. A join command contains both, so transfer and store the complete command as sensitive material.

Create a short-lived join command

Run token administration from a control-plane host with an administrative kubeconfig. Creating a fresh token for the specific maintenance window is clearer than trying to reuse the token printed days ago by kubeadm init.

```bash
sudo kubeadm token create \
  --ttl 30m \
  --description 'join worker-1' \
  --print-join-command
```

The command prints a complete worker join command with a generated token and the real CA hash. The 30-minute lifetime narrows the window in which that credential can be used. List active tokens when auditing the cluster, and delete an unused token by its six-character ID.

```bash
sudo kubeadm token list
sudo kubeadm token delete 4d7nq2
```

Join worker-1

Connect to worker-1, confirm that you are on the intended host, then run the freshly generated command. The explicit CRI socket removes ambiguity if the machine contains more than one runtime endpoint.

```bash
hostnamectl --static

sudo kubeadm join cp1.example.net:6443 \
  --token 4d7nq2.3rv6xap9m2c8kf5h \
  --discovery-token-ca-cert-hash sha256:0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef \
  --cri-socket unix:///run/containerd/containerd.sock
```

```plaintext
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[kubelet-start] Writing kubelet configuration...
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a successful TLS bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to the API server.
* The Kubelet was informed of the new secure connection details.
```

Exact output can vary by Kubernetes release. The durable signals are the same: preflight passed, cluster configuration was discovered, kubelet configuration was written, TLS bootstrap completed, and kubelet received permanent connection details.

What kubeadm does during a worker join

1. Preflight checks the host for common problems, including privileges, required binaries, ports, runtime access, and conflicting kubeadm-managed files.
2. Discovery downloads cluster information and validates it with the bootstrap token and pinned CA public-key hash.
3. kubeadm writes local kubelet configuration and starts or restarts kubelet with bootstrap credentials.
4. The kubelet creates a local key and submits a certificate signing request (CSR). In a standard kubeadm cluster, the configured bootstrap controllers approve the expected request.
5. The kubelet switches from the temporary token to its signed client certificate and registers the Node object using its node identity.
6. DaemonSets such as the CNI node agent and kube-proxy create node-local Pods. Once networking and kubelet health are established, the Ready condition becomes True.

The bootstrap token is therefore not the kubelet's long-term identity. It opens a controlled path to a client certificate; the kubelet then uses that certificate for ongoing API authentication and rotates it automatically in the normal kubeadm setup.

Verify more than registration

Run verification from an administrative workstation or control-plane host. Seeing a Node name proves registration; seeing Ready plus healthy node-local system Pods proves much more.

```bash
kubectl get nodes -o wide
kubectl describe node worker-1
kubectl get node worker-1 \
  -o jsonpath='{.status.conditions[?(@.type=="Ready")].status}{"\n"}'
```

```plaintext
NAME      STATUS   ROLES          AGE   VERSION
cp1       Ready    control-plane  18m   v1.37.0
worker-1  Ready    <none>         2m    v1.37.0

True
```

A worker normally shows no role because role names are conventional labels, not a value assigned automatically by worker join. Ready is the operational signal. If the Node is briefly NotReady, inspect the CNI Pods before assuming the join failed; the network agent may still be starting on the new host.

```bash
kubectl get pods -n kube-system -o wide \
  --field-selector spec.nodeName=worker-1

kubectl -n kube-node-lease get lease worker-1 \
  -o custom-columns=NAME:.metadata.name,RENEWED:.spec.renewTime
```

The first command should reveal the node's CNI agent and kube-proxy Pod when those add-ons use DaemonSets. The second shows the lightweight `Lease` that kubelet renews as a heartbeat. Node status reports capacity and conditions; the Lease gives the control plane a frequent, inexpensive liveness signal.

Add administrative labels after the join

Labels describe properties that administrators and workloads can use. Apply trusted role and placement labels with privileged kubectl after registration rather than asking kubelet to self-apply restricted label prefixes.

```bash
kubectl label node worker-1 node-role.kubernetes.io/worker=worker
kubectl label node worker-1 workload-tier=batch

kubectl get nodes \
  -L node-role.kubernetes.io/worker,workload-tier
```

The worker role label improves human readability; it does not install worker components or change scheduling by itself. The workload-tier label affects placement only when a Pod's node selector or affinity refers to it, as covered in the earlier scheduling lessons.

Take a worker out of service safely

Use `cordon` when the machine should keep running its current Pods but receive no new scheduler placements. This sets `.spec.unschedulable` on the Node.

```bash
kubectl cordon worker-1
kubectl get node worker-1 \
  -o custom-columns=NAME:.metadata.name,UNSCHEDULABLE:.spec.unschedulable
```

Use `drain` before maintenance or removal. Drain first cordons the Node, then requests safe eviction of eligible Pods. It respects PodDisruptionBudgets and leaves DaemonSet Pods in place when `--ignore-daemonsets` is supplied.

```bash
kubectl drain worker-1 \
  --ignore-daemonsets \
  --delete-emptydir-data

kubectl get pods -A -o wide \
  --field-selector spec.nodeName=worker-1
```

> --delete-emptydir-data permits eviction of Pods that use emptyDir and confirms that their node-local temporary data may be lost. Read a drain error before adding flags; do not turn safety checks into a reflexive list of overrides.

When maintenance is complete and kubelet, the runtime, and networking are healthy, return the Node to scheduler service.

```bash
kubectl uncordon worker-1
kubectl get nodes
```

Remove and rejoin a worker deliberately

Permanent removal has a cluster side and a host side. Drain workloads first. Delete the Node object so the control plane stops tracking the old identity. Then reset kubeadm-managed state on the host.

```control-plane or admin workstation · bash
kubectl drain worker-1 \
  --ignore-daemonsets \
  --delete-emptydir-data

kubectl delete node worker-1
```

```worker-1 · bash
sudo kubeadm reset -f \
  --cri-socket unix:///run/containerd/containerd.sock
```

The reset is best-effort. It removes kubeadm-managed files and stops kubelet-managed containers, but it does not remove CNI configuration in `/etc/cni/net.d`, kube-proxy traffic rules, or a user's `$HOME/.kube` directory. Inspect those separately and clean them only

when the host's next purpose requires it.

```bash
sudo find /etc/cni/net.d -mindepth 1 -maxdepth 1 -print
sudo ls -la /etc/kubernetes /var/lib/kubelet 2>/dev/null
```

To rejoin the same machine, make sure the previous Node object is gone, correct the reason for the reset, create a fresh short-lived join command, and run it on the cleaned host. Kubernetes requires unique Node names and treats reuse of a name as reuse of the same node identity, so stale state should not be carried into the new registration.

Use JoinConfiguration for repeatable joins

Flags are efficient for one node. A kubeadm JoinConfiguration makes node-specific inputs reviewable when provisioning several workers or integrating kubeadm with automation.

join-worker-1.yaml · yaml

```yaml
apiVersion: kubeadm.k8s.io/v1beta4
kind: JoinConfiguration
discovery:
  bootstrapToken:
    apiServerEndpoint: cp1.example.net:6443
    token: 4d7nq2.3rv6xap9m2c8kf5h
    caCertHashes:
      - sha256:0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
nodeRegistration:
  name: worker-1
  criSocket: unix:///run/containerd/containerd.sock
```

```bash
sudo kubeadm join --config join-worker-1.yaml
```

This file contains a live bootstrap token, so protect it as a secret and remove it after use. The configuration does not make the token long-lived; expiry is still enforced by the cluster.

Troubleshoot from the failed trust step

The token is invalid or expired

List tokens on a control-plane host. If the intended token is absent or expired, generate a new join command instead of weakening discovery.

```bash
sudo kubeadm token list
sudo kubeadm token create --ttl 30m --print-join-command
```

The API endpoint is unreachable

```bash
getent hosts cp1.example.net
ip route get 192.0.2.10
nc -zv cp1.example.net 6443
```

Fix name resolution, routing, firewalls, or the endpoint itself before retrying. Token rotation cannot repair a network path that never reaches the API server.

The CA hash does not match

A mismatch is a trust failure, not a warning to bypass. Obtain the hash again from a trusted control-plane host and compare the endpoint in the join command with the cluster you intend to reach.

```bash
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt \
  | openssl rsa -pubin -outform der 2>/dev/null \
  | openssl dgst -sha256 -hex \
  | sed 's/^.* //'
```

Preflight reports existing kubelet state

```bash
sudo ls -l /etc/kubernetes/kubelet.conf \
  /var/lib/kubelet/config.yaml 2>/dev/null
sudo journalctl -u kubelet -n 100 --no-pager
```

Existing files usually mean the host was already joined or a previous attempt progressed far enough to write state. Confirm whether the Node already exists. If the attempt is abandoned, reset the host deliberately before retrying; do not hide an unexplained conflict with ignore-preflight flags.

The Node registers but remains NotReady

```bash
kubectl describe node worker-1
kubectl get pods -n kube-system -o wide \
  --field-selector spec.nodeName=worker-1

sudo systemctl status kubelet containerd --no-pager
sudo journalctl -u kubelet -n 100 --no-pager
```

Read the Ready condition message and kubelet logs together. Common causes include an unavailable runtime, CNI initialization failure, incorrect node networking, and kubelet configuration errors. Because registration succeeded, begin with node health and networking rather than regenerating the bootstrap token.

What to remember

- A worker join is a two-way trust bootstrap followed by Node registration and readiness.
- Use a short-lived token with CA pinning; never treat unsafe discovery as a routine workaround.
- Registration is not the finish line: verify Ready, node-local system Pods, and the Node Lease.
- Cordon stops new scheduler placements; drain also evicts eligible workloads; uncordon returns the Node to service.
- Removing a worker requires both cluster-side cleanup and host-side reset because the Node object and machine state are separate.

Official documentation

Adding Linux worker nodes provides the supported worker prerequisites and join workflow.

kubeadm join documents discovery, TLS bootstrap, join phases, and flags.

kubeadm token covers token creation, lifetime, listing, and deletion.

Nodes explains Node objects, status, conditions, heartbeats, and the node controller.

Safely drain a node describes eviction behavior, DaemonSets, disruption budgets, and maintenance flow.

kubeadm reset defines the best-effort reset scope and the state it intentionally leaves behind.

kubeadm v1beta4 configuration documents JoinConfiguration and nodeRegistration fields.