Nodes

Overview

Overview

Infra Nodes

Infra Nodes

Alauda OS Nodes

Configure a MachineHealthChe Manage Nodes on Bare Metal Manage Dat

Manage Nodes on Huawei DCS Manage Nodes on Huawei Clou Configure F

Manage Nodes on VMware vSphere

Non-Alauda OS Nodes

Add and Remove Nodes

Platform administrators can add nodes to a
run a non-Alauda OS.

Manage Nodes

Update node labels and taints, stop and re

Plan an In-P

Plan and verify
platform create

Overview

Nodes are the fundamental building blocks of a cluster. Nodes can be virtual machines or physical servers, and each node runs the components required to host Pods, including kubelet, kube-proxy, and the container runtime.

Users with platform management permissions can add, manage, and plan nodes under clusters.

> **NOTE**
>
> Adding nodes to imported clusters or deleting nodes from imported clusters is not supported.

TOC

- Node types
- Node operation paths
  - Alauda OS Nodes
  - Non-Alauda OS Nodes
- Linux node availability check
- Supported operating systems and CPU models

Node types

- **Control plane nodes** run cluster components such as kube-apiserver, kube-scheduler, kube-controller-manager, etcd, container networking, and some platform management components.
- **Worker nodes** host business Pods running on the cluster. Plan worker node count based on workload demand.

Node operation paths

The following tasks apply to clusters whose nodes run Alauda OS and to clusters whose nodes run a non-Alauda OS.

Need	Continue with
Plan infra nodes or custom role nodes.	Infra Nodes
Tune CPU, memory, or topology manager policies for performance-sensitive workloads.	Resource manager policies

Alauda OS Nodes

Nodes on Alauda OS clusters are managed through Cluster API resources and are replaced rather than changed in place.

Need	Continue with
Configure machine health checks, or scale and maintain nodes on a specific infrastructure provider.	Alauda OS Nodes

Non-Alauda OS Nodes

Need	Continue with
Add nodes to or delete nodes from a cluster.	Add and Remove Nodes
Manage labels, taints, and node scheduling state.	Manage Nodes

Need	Continue with
Plan in-place operating system upgrades for eligible nodes.	Plan an In-Place OS Upgrade for Nodes

Linux node availability check

Before building a workload cluster on a non-Alauda OS, complete the node checks and confirm that every node meets the requirements. Cluster deployment can fail when prerequisites are not satisfied.

Supported operating systems and CPU models

For supported operating systems and CPU models, see Supported Operating Systems and CPU Models.

Infra Nodes

An infra node is a worker node dedicated to hosting infrastructure components. You create an infra node by giving a worker node the infra role label and a taint, and then move selected components to it.

TOC

- Node Roles
- Create Infra Nodes on Alauda OS Nodes
  - Step 1: Declare the Infra Label and the Taint on the Node Pool
  - Step 2: Verify
- Create Infra Nodes on Non-Alauda OS Nodes
  - Step 1: Add the Infra Role Label to the Node resources
  - Step 2: Add a Taint to the Node resources
  - Step 3: Verify the Label and Taint
- Move Workloads to Infra Nodes
- Custom Role Nodes

Node Roles

A cluster uses Kubernetes node role labels in the format `node-role.kubernetes.io/<role>` to assign different roles to nodes. In this topic, `role label`

refers to this type of label.

By default, a cluster contains two types of nodes: control plane nodes and worker nodes, used to host control plane workloads and application workloads, respectively.

In a cluster, the control plane nodes are labeled with the role label `node-role.kubernetes.io/control-plane`.

> **Note:**
> Prior to Kubernetes v1.24, the community also used the label `node-role.kubernetes.io/master` to mark control plane nodes. For backward compatibility, both labels are considered valid for identifying control plane nodes.

For the nodes or node pools that you plan for infrastructure components, such as logging, monitoring, and ingress, use the role label `node-role.kubernetes.io/infra`. To divide these nodes further, you can define your own roles, for example `node-role.kubernetes.io/log` or `node-role.kubernetes.io/monitor`.

Use role labels and taints to create infra nodes or custom role nodes, then move selected workloads to those nodes when workload isolation requires it.

Create Infra Nodes on Alauda OS Nodes

On an Alauda OS cluster, do not apply role labels or taints to `Node` resources by hand: nodes on this architecture are replaced rather than mutated, so anything applied to an individual `Node` disappears with the next node replacement. Declare the labels and taints on the node-pool resources instead — then every node the pool creates, including replacements, carries them.

The following steps declare the infra role on a node pool. To create a node pool, see the Create a Cluster page of your provider: Bare Metal, Huawei DCS, Huawei Cloud Stack, or VMware vSphere.

Step 1: Declare the Infra Label and the Taint on the Node Pool

Set the label on `.spec.template.metadata.labels` and the taint on `.spec.template.spec.taints` of the pool's `MachineDeployment`. For a control plane pool, set them on `.spec.machineTemplate.metadata.labels` and `.spec.machineTemplate.taints` of its `KubeadmControlPlane` instead. The example shows only the fields that define the infra role; keep the other fields of your `MachineDeployment` as they are.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <infra-md-name>
  namespace: <namespace>
spec:
  clusterName: <cluster-name>
  template:
    metadata:
      labels:
        node-role.kubernetes.io/infra: ""
    spec:
      taints:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
```

Labels and taints that you add or change on the pool's template are applied to the existing nodes of the pool without replacing them.

Step 2: Verify

After the nodes of the pool register, check that the `Node` has the infra role label and the taint `node-role.kubernetes.io/infra=reserved:NoSchedule`, for example with `kubectl describe node <node-name>`.

A single-cluster deployment in which the `global` cluster doubles as the workload cluster is a supported deployment form; its node planning follows the same rules as any other cluster of its size.

Create Infra Nodes on Non-Alauda OS Nodes

By default, a cluster only includes control plane nodes and worker nodes. If you want to designate certain worker nodes as infra nodes dedicated to hosting infrastructure components, you need to manually add the appropriate role label and taint to those nodes.

> **Note:**
> The operations in this section apply only to clusters whose nodes run a non-Alauda OS. Do not use these operations for provider-managed cloud clusters, third-party clusters, or clusters whose nodes run Alauda OS.

Step 1: Add the Infra Role Label to the Node resources

```
kubectl label nodes 192.168.143.133 node-role.kubernetes.io/infra="" --ov
erwrite
```

This command adds the infra role label to the Node 192.168.143.133: `node-role.kubernetes.io/infra: ""`, indicating that the node is an infra node.

Step 2: Add a Taint to the Node resources

Add a taint to prevent other workloads from being scheduled onto the infra node.

```
kubectl taint nodes 192.168.143.133 node-role.kubernetes.io/infra=reserve
d:NoSchedule
```

This command adds the taint `node-role.kubernetes.io/infra=reserved:NoSchedule` to Node 192.168.143.133, indicating that only applications that tolerate this taint can be scheduled onto this node.

Step 3: Verify the Label and Taint

Check whether the node has been assigned the infra role label and taint:

```
# kubectl describe node 192.168.143.133
Name:               192.168.143.133
Roles:              infra
Labels:             node-role.kubernetes.io/infra=""
                    ...
Taints:             node-role.kubernetes.io/infra=reserved:NoSchedule
```

The output indicates that the Node `192.168.143.133` has been configured as an infra node and has the `node-role.kubernetes.io/infra=reserved:NoSchedule` taint.

To set labels and taints for existing nodes in the web console instead, see Manage Nodes.

Move Workloads to Infra Nodes

If you want to schedule a specific Pod onto infra nodes, you need to make the following configurations:

- A nodeSelector targeting the infra role label.
- Corresponding tolerations for the infra node's taint.

Below is an example Pod manifest configured to run on the infra node.

```
apiVersion: v1
kind: Pod
metadata:
  name: infra-pod-demo
  namespace: default
spec:
  ...
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    value: reserved
    operator: Equal
  ...
```

The nodeSelector ensures the Pod is only scheduled on nodes with the label `node-role.kubernetes.io/infra: ""`, and the toleration allows the Pod to tolerate the taint `node-role.kubernetes.io/infra=reserved:NoSchedule`.

With these configurations, the Pod will be scheduled onto the infra node.

> **Note:**
> Moving pods installed through OLM Operators or Cluster Plugins to an infra node is not always possible. The capability to move these pods depends on the configuration of each Operator or Cluster Plugin.

For the following components, the documentation describes how to place them on infra nodes:

- Monitoring: Planning Infra Nodes for Monitoring.
- ALB: Tasks for ALB.
- Envoy Gateway: Tasks for Envoy Gateway.
- Operators: Operator, section Configure Operator Pod Scheduling.

Custom Role Nodes

To divide infra nodes further, you can define custom roles — for example, log nodes for logging components or monitor nodes for monitoring components. You can define any role name for this.

A custom role node works exactly like an infra node, with your role name in place of `infra`:

- Use the role label `node-role.kubernetes.io/<role>` and the taint `node-role.kubernetes.io/<role>=<value>:NoSchedule`. The `<value>` is optional but can help operators understand the scheduling intent, such as `reserved` or `dedicated`.
- On Alauda OS nodes, use **one node pool per role**. Each node pool carries its own `node-role.kubernetes.io/<role>` label and taint, declared on the pool as in Create Infra Nodes on Alauda OS Nodes.
- On non-Alauda OS nodes, add the label and the taint to the `Node` as in Create Infra Nodes on Non-Alauda OS Nodes.

- To schedule a Pod onto these nodes, configure a nodeSelector and tolerations as in Move Workloads to Infra Nodes.

Manage Nodes on Alauda OS Clusters

This section covers day-2 node operations on clusters whose nodes run Alauda OS.

INFO

Nodes of the `global` cluster

Before you scale out or in, replace a control plane or worker node, or rely on a MachineHealthCheck on the `global` cluster itself, make sure the infrastructure provider for the platform that `global` runs on and the Cluster Bootstrap Provider are installed on `global` as cluster plugins; installing `global` does not leave them there. See the provider installation page for Bare Metal, Huawei DCS, Huawei Cloud Stack, or VMware vSphere. In a DR pair, do this on both `global` clusters.

TOC

Tasks for All Providers

Pages in This Section

Related Topics

Tasks for All Providers

- Create Infra Nodes

- Configure a MachineHealthCheck

Pages in This Section

Configure a MachineHealthChe

Overview

Prerequisites

Configure a MachineHealthCheck For a M

Configure a MachineHealthCheck For con

Reference

Manage Nodes on Bare Metal

Prerequisites

Overview

Node Management Operations

Pool and Inventory Observability

Common Failure Modes

Next Steps

Manage Dat

Availability and

Storage Lifecyc

Prerequisites

Step 1: Verify th

Step 2: Define t

Step 3: Render

Step 4: Wait for

Step 5: Allocate

Manage Nodes on Huawei DCS

Manage Nodes on Huawei Cloud Stack

Prerequisites

Overview

Using the Web UI

Using YAML

Verification

Troubleshooting

Configure FQDN Hostname on Existing HCS C

Manage Nodes on VMware vSphere

Prerequisites

Steps

Updating Bootstrap Templates

Runtime Topology Changes

Troubleshooting

Next Steps

Related Topics

- Create Workload Clusters on Alauda OS
- Upgrade Kubernetes on Alauda OS Workload Clusters
- Machine Configuration

Configure a MachineHealthCheck

TOC

Overview

Prerequisites

Configure a MachineHealthCheck For a MachineDeployment

Configure a MachineHealthCheck For control plane nodes

Reference

Overview

A MachineHealthCheck is a resource within the Cluster API which allows users to define conditions under which Machines within a Cluster should be considered unhealthy. A MachineHealthCheck is defined on a management cluster and scoped to a particular workload cluster.

When defining a MachineHealthCheck, users specify a timeout for each of the conditions that they define to check on the Machine's Node. If any of these conditions are met for the duration of the timeout, the Machine will be remediated. By default, the action of remediating a Machine should trigger a new Machine to be created to replace the failed one, but providers are allowed to plug in more sophisticated external remediation solutions.

WARNING

MachineHealthCheck relies on Cluster API's rolling update mechanism. During a rolling update, any previously attached disks are removed and replaced with new disks on newly created machines. Ensure that no cluster functionality or workloads depend on data stored on the original disks.

Prerequisites

Before attempting to configure a MachineHealthCheck, you should have a working management cluster with at least one MachineDeployment or KubeadmControlPlane deployed. To use a MachineHealthCheck on the `global` cluster itself, the infrastructure provider and the Cluster Bootstrap Provider must be installed on `global` first; see Manage Nodes on Alauda OS Clusters.

Configure a MachineHealthCheck For a MachineDeployment

To configure a MachineHealthCheck for a MachineDeployment, you need to create a MachineHealthCheck resource in the management cluster.

```
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: <machinehealthcheck-name>
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  nodeStartupTimeout: 30m
  selector:
    matchExpressions:
    - key: cluster.x-k8s.io/deployment-name
      operator: Exists
  unhealthyConditions:
  - type: Ready
    status: Unknown
    timeout: 600s
  - type: Ready
    status: "False"
    timeout: 600s
```

Configure a MachineHealthCheck For control plane nodes

```yaml
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: <control-plane-machinehealthcheck-name>
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  maxUnhealthy: 1
  selector:
    matchLabels:
      cluster.x-k8s.io/control-plane: ""
  unhealthyConditions:
    - type: Ready
      status: Unknown
      timeout: 600s
    - type: Ready
      status: "False"
      timeout: 600s
```

WARNING

Limitations and Risks for Control Plane Nodes

- **Etcd Leader Failure with Kubelet Downtime:** Automatic remediation may fail if the unhealthy node is the current etcd leader and its `kubelet` service is down. Cluster API attempts to migrate the etcd leadership before deleting the node but relies on the `kubelet` to proxy requests to the local etcd instance. If the `kubelet` is unresponsive, the leadership transfer cannot proceed, blocking the rotation.
- **Risk of Quorum Loss:** Exercise caution when configuring `MachineHealthCheck` for control plane nodes. Aggressive timeouts or misconfiguration can lead to the removal of multiple control plane nodes, potentially breaking the etcd quorum and rendering the cluster's control plane unusable.

Reference

Configure a MachineHealthCheck ↗

Manage Nodes on Bare Metal

This document explains how to scale worker nodes up and down, replace specific machines, and recover failed nodes on the bare-metal provider. Node management uses Cluster API `Machine` resources orchestrated through `MachineDeployment`; the provider binds each `Machine` to one `MachineInventory` from a pool and drives the host through `clean` / `reprovision` plans.

> **INFO**
>
> To upgrade the Kubernetes version of the nodes, see Upgrade Kubernetes on Bare Metal.

TOC

- Prerequisites
- Overview
- Node Management Operations
  - Scaling Worker Nodes
    - Adding Worker Nodes
    - Removing Worker Nodes
  - Replacing a Single Failed Node
  - Moving Workers to a Different Pool
  - Updating Bootstrap Templates
- Pool and Inventory Observability
- Common Failure Modes

Next Steps

Prerequisites

WARNING

Important Prerequisites

- The control plane must already be running. See Create Cluster.
- The worker `MachineInventoryPool` must contain at least as many `Available` inventories as the target replica count, plus the headroom required by the rollout strategy.
- The target `Machine.spec.version` must be a key in the `elemental-image-catalog` ConfigMap.
- Any inventory with non-empty `spec.storage.volumes[]` must already report `status.storage.phase=Prepared` and `StoragePrepared=True` before it can count as `Available`. Follow Manage Data Disks on Bare-Metal Hosts before adding a newly registered storage-managed inventory to the worker pool.
- To manage the nodes of the `global` cluster itself, the provider must be installed on `global` first; see Manage Nodes on Alauda OS Clusters.

INFO

Configuration Guidelines

When working with the configurations in this document:

- Only modify values enclosed in `<>` brackets.
- Replace placeholder values with your environment-specific settings.
- Preserve all other default configurations unless explicitly required.

Overview

The four resources that compose a worker node group:

1. **`MachineInventoryPool`** (`<cluster-name>-worker-pool`) — the allowed set of `MachineInventory` names. Already created in Create Cluster → Step 2.
2. **`BaremetalMachineTemplate`** (`<cluster-name>-worker-template`) — references the worker pool. To move the workers to a different pool, see Moving Workers to a Different Pool.
3. **`KubeadmConfigTemplate`** (`<cluster-name>-worker-bootstrap`) — cloud-init `user-data` for `kubeadm join`. The bare-metal provider normalizes this user-data at reprovision time (hostname, `provider-id` , `criSocket`); operators should not pre-fill those fields.
4. **`MachineDeployment`** — controls replica count, version, rollout strategy.

Managed data disks are not part of `BaremetalMachineTemplate`. Their declaration and retained data belong to each long-lived `MachineInventory` , so two members of the same worker pool may expose different host-local datasets unless operators standardize and verify their storage declarations. Managed storage is changed separately on an unallocated, inactive `MachineInventory` ; see Manage Data Disks on Bare-Metal Hosts.

To add another worker node pool with YAML, follow Step 4: Deploy Worker Nodes on the cluster creation page.

Node Management Operations

Scaling Worker Nodes

Adding Worker Nodes

Use Case: Increase cluster capacity.

Prerequisites:

- The worker pool has enough `Available` inventories. If not, register the additional hosts (boot the SeedImage ISO and confirm the new `MachineInventory` objects), prepare any managed storage, and append them to `MachineInventoryPool.spec.machineInventories[]`.

Procedure:

1. **Check current state**

```
kubectl -n cpaas-system get machines.cluster.x-k8s.io \
  -l cluster.x-k8s.io/deployment-name=<cluster-name>-workers
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.x-k8s.io \
  <cluster-name>-worker-pool
```

   Confirm that `MachineInventoryPool.status.available` is high enough for the planned increment.

2. **Extend the worker pool when more inventories are needed**

```
kubectl -n cpaas-system edit machineinventorypool <cluster-name>-worker-pool
```

   Append the new `MachineInventory` names — they must already be registered and Ready. Inventories with managed volumes must also be Prepared; otherwise they increase `status.total` but remain unavailable for allocation. Save and exit, then confirm the expected increase in `status.available`.

3. **Scale up the `MachineDeployment`**

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers \
  --type='json' -p='[{"op":"replace","path":"/spec/replicas","value":<new-replica-count>}]'
```

4. **Monitor**

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8s.io -w
kubectl get nodes                # workload cluster
```

New `BaremetalMachine` objects advance `Pending → Allocated → Reprovisioning → Running`; the pool's `available` counter decreases as new nodes are bound.

Removing Worker Nodes

Two strategies are supported, identical in shape to the upstream Cluster API contract:

Strategy	When to use
Random removal	Any node can be removed — for example, a temporary capacity reduction.
Targeted removal	A specific physical host should be released (hardware maintenance, replacement, IP recovery).

> **INFO**
>
> **Inventory recycling**
>
> A `clean` plan stops kubelet, deactivates declared storage mounts, clears CRI workload, and stops containerd. It does **not** wipe managed filesystems, Kubernetes persistent directories, or reset the OS — OS cleanup happens later, when the same or a different inventory is picked for a new `BaremetalMachine` and runs the `reprovision` plan. Until then, the inventory returns to `Available` and may be allocated again. Deleting a Machine performs **Deactivate**, not storage Release, so the storage declaration, filesystem UUIDs, and data remain on that inventory.

Random Removal

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers \
  --type='json' -p='[{"op":"replace","path":"/spec/replicas","value":<new-replica-count>}]'
```

CAPI selects machines for deletion in its standard order. Each selected `BaremetalMachine` moves to `Preparing`, the provider writes a `clean` plan, and once the plan reports `Applied`, the inventory returns to `Available`.

Targeted Removal

1. **Identify the machines**

```
kubectl -n cpaas-system get machines.cluster.x-k8s.io \
  -l cluster.x-k8s.io/deployment-name=<cluster-name>-workers
```

2. **Annotate the target machines**

```
kubectl -n cpaas-system patch machine <machine-name> \
  --type='merge' -p='{"metadata":{"annotations":{"cluster.x-k8s.io/dele
te-machine":"true"}}}'
```

   Repeat for every machine you want to remove.

3. **Scale down by exactly the number of annotated machines**

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers
\
  --type='json' -p='[{"op":"replace","path":"/spec/replicas","value":<n
ew-replica-count>}]'
```

   Reducing by fewer leaves annotated machines in place; reducing by more sends random machines through the `clean` plan as well.

4. **Verify cleanup**

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-
k8s.io
kubectl -n cpaas-system get machineinventories.elemental.cattle.io <inv
entory-name> -o yaml
```

   The released inventory must show `baremetal.alauda.io/allocation-state=Available`, the `baremetal.alauda.io/owner-*` annotations must be cleared, and the pool annotation must remain. `MachineInventory` itself is **not** deleted. If it

declares managed volumes, it must return to `status.storage.phase=Prepared` with `StorageActive=False/Inactive`; its business mount paths must be inactive while filesystem UUIDs and data remain unchanged.

Replacing a Single Failed Node

If a `BaremetalMachine` lands in `Failed`, the safest recovery is to delete the failed `Machine`. CAPI immediately creates a replacement `Machine` (because `replicas` did not change), and the bare-metal provider picks an `Available` inventory from the same pool. Most often the replacement is a different `MachineInventory`; the provider does not guarantee that the freshly released inventory will be re-picked. Host-local managed data does not follow the replacement Machine. If the application must reuse a specific inventory's data, use a dedicated or deliberately ordered pool and verify `BaremetalMachine.status.machineInventoryRef` before relying on that data.

```
kubectl -n cpaas-system delete machine <machine-name>
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8s.io -w
```

Investigate the original failure separately: read `BaremetalMachine.status.conditions`, `MachineInventory.status.plan.state`, and the failing plan secret's `failed-output` key on the host that ran the plan. Common root causes are documented in the Common Failure Modes section below.

Moving Workers to a Different Pool

`BaremetalMachineTemplate` carries only the pool reference — there is no CPU, memory, or disk declaration on the template.

Use Case: New hosts have been registered and placed in a new `MachineInventoryPool`, and the nodes should move onto them. Create a new `BaremetalMachineTemplate` that points to the new pool, then point the `MachineDeployment` at it. Nodes are replaced one by one.

Before you start:

- The target pool is in the same namespace as the template.
- The target pool has at least as many `Available` inventories as the `MachineDeployment` has replicas. Each moved node uses one host of the target pool; hosts released by the old nodes return to the old pool and do not join the target pool. Read the count from `status.available`:

```
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluste
r.x-k8s.io <target-pool-name>
```

- The new hosts meet the same prerequisites as the current ones: registered and ready, managed data disks prepared if you use them, the target Kubernetes version and CPU architecture in the image catalog, and the CNI network device present. See Confirm the Worker Pool Capacity.
- Keep the old template and the old pool until the move is finished.

Procedure:

1. Create a new `BaremetalMachineTemplate` with a new `metadata.name` referencing the target pool. Use the format in Configure the Worker `BaremetalMachineTemplate`.
2. Apply the new template.
3. Patch the `MachineDeployment`:

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers
\
  --type='merge' \
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<new-te
mplate-name>"}}}}}'
```

4. Watch the rolling replacement (`maxSurge: 0`, one node at a time) and the target pool's `status.available`:

```
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-
k8s.io -w
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluste
r.x-k8s.io <target-pool-name>
```

What to expect:

- The old node is deleted before its replacement is created. If the target pool runs out of hosts, the node pool stays one node short until you add hosts.
- Node names and IP addresses change.
- Node-local data is not carried over. Data on managed data disks stays on the old host, bound to its `MachineInventory`, and does not appear on the new host. Declare storage on the new hosts separately; see Manage Data Disks on Bare-Metal Hosts.

If the move stalls with no free host (`BaremetalMachine` shows `InventoryAllocated=False / Reason=PoolExhausted`), add `Available` hosts to the target pool; they are picked up automatically. Alternatively, roll back by patching `infrastructureRef.name` back to the old template. Nodes that were already moved are replaced again one by one, so the old pool must still have enough `Available` hosts.

To move control plane nodes, use the same steps and patch `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` instead. Nodes are replaced one at a time. In addition:

- `maxSurge: 0` requires `replicas` of 3 or more.
- During each replacement only two etcd members run. Start only when the control plane is healthy, and keep 3 `Available` hosts ready in the target pool.
- The new hosts must be in the same layer-2 network as the control plane VIP.

After the move is complete, you can delete the old template and the old pool. Hosts released by the old nodes stay in the old pool.

Updating Bootstrap Templates

Modifying an existing `KubeadmConfigTemplate` in place does not roll out existing machines; only newly created machines pick up the change.

To roll out a bootstrap change:

1. Export the existing template:

```
kubectl -n cpaas-system get kubeadmconfigtemplate <cluster-name>-worker
-bootstrap -o yaml \
  > new-worker-bootstrap.yaml
```

2. Change `metadata.name`, remove server-generated fields (`resourceVersion`, `uid`, `creationTimestamp`, `managedFields`, `kubectl.kubernetes.io/last-applied-configuration`) and the entire `status`, and edit the desired fields. For Kubernetes 1.35 or later, add the required kubelet patch settings to `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`.
3. Apply the new template:

```
kubectl apply -f new-worker-bootstrap.yaml
```

4. Patch the `MachineDeployment` to reference the new template:

```
kubectl -n cpaas-system patch machinedeployment <cluster-name>-workers
\
  --type='merge' \
  -p='{"spec":{"template":{"spec":{"bootstrap":{"configRef":{"name":"<n
ew-template-name>"}}}}}}'
```

This triggers a rolling replacement.

Pool and Inventory Observability

The annotations the provider maintains on each `MachineInventory` are the authoritative source for "what is this host being used for right now":

Annotation	Values	Meaning
`baremetal.alauda.io/pool`	Pool name	Which pool owns this inventory.

Annotation	Values	Meaning
`baremetal.alauda.io/allocation-state`	`Available` / `Allocated` / `Preparing` / `Reprovisioning` / `Unavailable`	Lifecycle phase from the provider's point of view.
`baremetal.alauda.io/owner-cluster`	Cluster name	Cluster currently using this inventory (cleared on release).
`baremetal.alauda.io/owner-machine`	`Machine` name	Owning Cluster API `Machine`.
`baremetal.alauda.io/owner-baremetalmachine`	`BaremetalMachine` name	Owning provider machine.

Each inventory's default plan secret also carries `baremetal.alauda.io/plan.type`. Normal node lifecycle uses `clean` or `reprovision`; inventory storage reconciliation additionally uses `storage-prepare` and `storage-release`. Inspect `MachineInventory.status.storage.conditions` and `.operation` together with the plan type when diagnosing storage.

Common Failure Modes

Scenario	Expected condition	What to check
Pool absent	`BaremetalMachine` stays `Pending`, `InventoryAllocated=False / Reason=PoolMissing`	`BaremetalMachineTemplate.s` and namespace
Pool exhausted	`BaremetalMachine` stays `Pending`, `InventoryAllocated=False / Reason=PoolExhausted`	`MachineInventoryPool.statu`

Scenario	Expected condition	What to check
Member missing	`MachineInventoryPool.MembersValid=False / Ready=False`	Pool's `status.unavailable`,
Inventory not Ready	Counted toward `status.unavailable`	`MachineInventory.status.co`
Plan secret missing	Inventory ineligible for allocation	`MachineInventory.status.pl`
Storage not prepared	Inventory does not count as `Available`; `StoragePrepared=False`	`MachineInventory.status.st` `status.observedStorage`, ar
Bootstrap secret missing	`BootstrapReady=False / Reason=BootstrapWaiting`	`Machine.spec.bootstrap.dat`
Catalog miss	`BaremetalMachine` `Failed`, `ImageResolved=False / Reason=ImageCatalogMiss`, no plan written	`elemental-image-catalog` k
Registry annotation missing	`ImageResolved=False / Reason=ImageRegistryMissing`	`Cluster.metadata.annotatio`
Reprovision plan failed	`BaremetalMachine` `Failed`, inventory marked `Unavailable`	Plan secret `failed-output`; h
Managed volume activation failed	`BaremetalMachine` `StorageReady=False`; node readiness is blocked for a required volume	Stable device identity, filesysten plan `failed-output`
Clean or Deactivate failed	Deletion blocked by finalizer; inventory marked `Unavailable`	`clean` plan output, host moun `MachineInventory.status.st`

For the full operator-side state machine reference, see Provider Overview → BaremetalMachine.

Next Steps

- Create Clusters
- Manage Data Disks
- Upgrade Clusters

Manage Data Disks on Bare-Metal Hosts

Use `MachineInventory.spec.storage` to manage data disks that are already attached to a bare-metal host. The storage declaration belongs to the long-lived `MachineInventory`, not to a temporary Cluster API `Machine`, so the provider can retain the filesystem and data when a node is removed and later reuse the same inventory.

This feature does not create, attach, detach, migrate, back up, or replicate physical disks or SAN LUNs. It manages only the devices explicitly listed in `spec.storage.volumes[]`. Every unlisted device remains observed but unmanaged.

DANGER

`InitializeIfBlank` can format a disk

Use `InitializeIfBlank` only after you have verified the exact stable device ID on the target inventory, confirmed that the device is a disposable blank data disk, and backed up any important data independently. Never copy a device ID from another host and never use `/dev/sdX`, `/dev/vdX`, `/dev/nvmeXnY`, or `/dev/mapper/mpathX` as persistent identities.

TOC

Availability and Upgrade Boundary

Storage Lifecycle

Prerequisites

Step 1: Verify the Inventory and Observer

Step 2: Define the Managed Volumes

Step 3: Render and Apply an Atomic Patch

Step 4: Wait for Prepare

Step 5: Allocate and Verify Activation

Step 6: Scale Down and Reuse the Inventory

Step 7: Release Storage Ownership

Failed Operations and Retry

Restrictions

Related Documentation

Availability and Upgrade Boundary

Managed storage requires all three of the following components:

- The Bare Metal provider, including its storage controller and fail-closed `MachineInventory` admission webhooks.
- The matching `elemental-operator` CRD and observer API.
- A host OS image that installs and enables `elemental-storage-observer.service` and the early-boot storage integration.

> **WARNING**
>
> Obtain the matching `base-image` and `base-image-iso` from Alauda technical support before enabling managed storage. Use `base-image` for host reprovisioning and `base-image-iso` for `SeedImage` / live installation. These images are a supported pair for the target release; do not derive one image reference from the other or substitute an independently built image.

Upgrading only the provider and `elemental-operator` on the `global` cluster does not install the observer service on hosts that were created from an older OS image.

- An existing cluster can remain on unmanaged storage after the management-side upgrade when `spec.storage` stays absent or has `volumes: []`. This path does not create storage plans or modify disks.

- To enable managed storage on an older host, first reinstall, reset, or replace that host with an approved observer-capable image. Confirm the observer service and a fresh `status.observedStorage` report before adding any non-empty storage declaration.
- Do not add `spec.storage.volumes[]` to a running or allocated inventory. Storage declarations can change only while the inventory is unallocated and all managed volumes are inactive.

Storage Lifecycle

Stage	Trigger	Expected state	Host
Observe	The host registers and the observer reports devices.	`Unmanaged` or `Observed`	Read- discov device select autom
Prepare	An operator saves a non-empty storage declaration on an available inventory.	`Prepared`, `StoragePrepared=True`	Adopt existir filesys initiali explic autho blank Busin remai unmo
Activate	Cluster API allocates the prepared inventory to a `BaremetalMachine`.	`Active`, `StorageActive=True`	The reprov plan ir activa state; mount start b kubele
Deactivate	The owning Cluster API `Machine` is	`Prepared`, `StorageActive=False`	The cl unmo

Stage	Trigger	Expected state	Host
	deleted.		manag volum Filesy storag owner data r
Release	Volumes are removed from `spec.storage` while the inventory is available and inactive.	`Unmanaged`, `StoragePrepared=True/NoManagedVolumes`	Storag owner activa metac remov withou the file or dat

Deleting a Machine performs **Deactivate**, not Release. A normal scale-down therefore preserves the inventory's storage declaration and data for later reuse.

Prerequisites

1. Install the Infrastructure Provider for Bare Metal on the `global` cluster as described in Install the Bare Metal Provider, and wait for its manager deployment and admission webhooks to become Ready.
2. Obtain the matching observer-capable `base-image` and `base-image-iso` from Alauda technical support. Use the ISO image to install the host and configure the paired base image for later reprovisioning.
3. Keep the target `MachineInventory` outside production pools until Prepare succeeds. It must be `Ready=True`, unallocated, and have a default plan Secret.
4. Back up existing data independently. Storage retention is not an application-consistent backup.
5. Obtain the `storagectl` binary built from the exact same provider revision. `storagectl` reuses the provider's normalization, validation, and hash code; do not calculate the initialization approval hash by hand or use a binary from another release.

6. The identity that submits `InitializeIfBlank` must be authorized for `update` on the virtual `machineinventories/storageinitialize` subresource. Pure `Adopt` declarations do not require this additional permission.

Set operation-specific variables:

```
export BM_NS=cpaas-system
export BM_INVENTORY='<machineinventory-name>'
export BM_STORAGECTL='<path-to-matching-storagectl>'
export BM_STORAGE_FILE=/tmp/baremetal-storage.yaml
export BM_INVENTORY_FILE=/tmp/baremetal-machineinventory.json
export BM_PATCH_FILE=/tmp/baremetal-storage.patch.json
```

Verify the management components and authorization:

```
kubectl -n "$BM_NS" rollout status \
  deployment/cluster-api-provider-baremetal-manager --timeout=5m
kubectl get mutatingwebhookconfiguration \
  cluster-api-provider-baremetal-mutating-webhook-configuration
kubectl get validatingwebhookconfiguration \
  cluster-api-provider-baremetal-validating-webhook-configuration
kubectl auth can-i update machineinventories.elemental.cattle.io \
  --subresource=storageinitialize -n "$BM_NS"
```

The final command must return `yes` before the identity can submit or change an initialization approval. Grant this permission only to the specific administrative user or ServiceAccount that is authorized to format disks.

Step 1: Verify the Inventory and Observer

Confirm that the inventory is available and has no current owner:

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" \
  -o jsonpath='uid={.metadata.uid}{"\n"}allocation={.metadata.annotation
s.baremetal\.alauda\.io/allocation-state}{"\n"}baremetalMachine={.metadat
a.annotations.baremetal\.alauda\.io/owner-baremetalmachine}{"\n"}machine=
{.metadata.annotations.baremetal\.alauda\.io/owner-machine}{"\n"}cluster=
{.metadata.annotations.baremetal\.alauda\.io/owner-cluster}{"\n"}plan={.s
tatus.plan.secretRef.name}{"\n"}'
```

The allocation value must be empty or `Available`; all owner values must be empty. If the inventory is allocated, delete the owning CAPI Machine and wait for the normal clean and Deactivate flow. Do not remove owner annotations or finalizers manually.

On the host, verify the observer:

```
sudo systemctl is-enabled elemental-storage-observer.service
sudo systemctl is-active elemental-storage-observer.service
sudo systemctl cat elemental-storage-observer.service
```

Both state commands must succeed. On the `global` cluster, inspect the observation watermark and device facts:

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" \
  -o jsonpath='{.status.observedStorage.observedAt}{" epoch="}{.status.ob
servedStorage.reportEpoch}{" sequence="}{.status.observedStorage.sequenc
e}{" bootID="}{.status.observedStorage.bootID}{"\n"}'

kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  | jq -r '.status.observedStorage.devices[] |
      [.id,.kind,.systemRole,(.sizeBytes|tostring),(.filesystem.type//"-
"),
       (.filesystem.uuid//"-"),([.mounts[].path]|join(",")),
       ((.health.activePaths//0)|tostring),((.health.totalPaths//0)|tostr
ing)] | @tsv'
```

Use only a device whose report is fresh and whose `systemRole` is `Data`.

Device kind	Stable ID	Supported policy
Direct disk	`wwn:`, `nvme-eui:`, `nvme-nguid:`, or an observer-approved `serial:` ID	`Adopt` or `InitializeIfBlank`
Existing partition	`partuuid:`	`Adopt` only
Multipath aggregate map	`wwid:`	`Adopt`, or `InitializeIfBlank` when the whole map is blank

Do not select a system or `Unknown` device, a multipath member, a read-only or removable device, or a device with an unexpected partition table, filesystem, LVM, RAID, LUKS, swap, consumer, or mount. Do not select both a whole disk and one of its partitions.

Step 2: Define the Managed Volumes

Create a storage declaration. The example intentionally uses placeholders; replace every device ID, UUID, capacity, and mount path with facts from the same target inventory. Remove any volume that does not apply to the host.

`baremetal-storage.yaml`

```yaml
storage:
  retryNonce: 0
  volumes:
    - name: cpaas-data
      source:
        deviceID: <blank-direct-disk-id>
        minimumSize: 100Gi
      filesystem:
        policy: InitializeIfBlank
        type: xfs
      mount:
        path: /var/cpaas
        required: true
        options:
          - noatime

    - name: application-data
      source:
        deviceID: <existing-partition-id>
        minimumSize: 100Gi
      filesystem:
        policy: Adopt
        type: ext4
        expectedUUID: <existing-filesystem-uuid>
      mount:
        path: /data/application
        required: true

    - name: registry-data
      source:
        deviceID: <multipath-map-id>
        multipath:
          minimumActivePaths: 2
      filesystem:
        policy: Adopt
        type: xfs
        expectedUUID: <existing-filesystem-uuid>
      mount:
        path: /srv/registry
        required: true
        options:
          - noatime
```

Storage v2 supports at most 32 volumes per inventory. The first version supports whole-device XFS or ext4, existing XFS/ext4 partitions through `Adopt`, and whole multipath maps. It does not create partitions or manage LVM, mdraid, LUKS, swap, shared-cluster filesystems, or multi-host read/write access.

`required` defaults to `true`. An optional volume is still prepared and activated; only its failure-to-Ready behavior changes. `/var/cpaas` must always be required. The provider adds `nodev,nosuid` to managed mounts and validates every extra mount option against an allowlist.

Validate the declaration locally:

```
"$BM_STORAGECTL" hash --storage "$BM_STORAGE_FILE"
```

Step 3: Render and Apply an Atomic Patch

Always start from a current live object so the patch contains the latest UID and `resourceVersion`:

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  >"$BM_INVENTORY_FILE"
```

For a declaration that contains `InitializeIfBlank`, choose a positive counter strictly greater than `status.storage.lastConsumedInitializationApprovalCounter` and render the patch:

```
jq '.status.storage.lastConsumedInitializationApprovalCounter // 0' \
  "$BM_INVENTORY_FILE"

"$BM_STORAGECTL" render-patch \
  --inventory "$BM_INVENTORY_FILE" \
  --storage "$BM_STORAGE_FILE" \
  --counter <next-initialization-counter> \
  >"$BM_PATCH_FILE"
```

The generated merge patch binds the approval to the current Inventory UID, normalized storage hash, counter, and `resourceVersion`. For an `Adopt`-only declaration, omit `--counter`.

Run a server-side dry-run, inspect the resulting `spec.storage`, device IDs, paths, and approval one more time, and only then apply it:

```
kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE" --dry-run=server -o yaml

kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE"
```

If the API reports a `resourceVersion` conflict, fetch the inventory again, regenerate the patch, repeat the server dry-run, and then apply. Never remove `resourceVersion` from the generated patch.

Step 4: Wait for Prepare

Watch the storage state:

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -w

kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  | jq '.status.storage | {
      phase,appliedSpecHash,pendingSpecHash,
      lastConsumedInitializationApprovalCounter,conditions,appliedVolume
s,operation}'
```

Before the inventory can be allocated, the expected result is:

```
phase=Prepared
StoragePrepared=True/AllRequiredVolumesPrepared
StorageActive=False/Inactive
operation=null
```

At this stage, initialized volumes have filesystem UUIDs and adopted volumes retain their original UUIDs, but the business mount paths are not active. Do not add the inventory to a production `MachineInventoryPool` until Prepare has completed.

Step 5: Allocate and Verify Activation

Add the prepared inventory to the intended `MachineInventoryPool`, then create or scale the CAPI machine group as described in Create a Cluster on Bare Metal and Manage Nodes on Bare Metal.

After reprovisioning completes, verify both the machine and storage state:

```
kubectl -n "$BM_NS" get baremetalmachines.infrastructure.cluster.x-k8s.io
-o json \
  | jq -r --arg inventory "$BM_INVENTORY" \
      '.items[] | select(.status.machineInventoryRef.name==$inventory) |
       [.metadata.name,.status.phase,.status.ready,
        ([.status.conditions[]|select(.type=="StorageReady")|(.status
+"/"+.reason)]|join(","))] | @tsv'

kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  | jq '.status.storage | {phase,conditions,appliedVolumes,operation}'
```

The expected result is `BaremetalMachine.status.phase=Running`, `StorageReady=True/AllVolumesReady`, `MachineInventory.status.storage.phase=Active`, and `StorageActive=True/AllRequiredVolumesActive`.

On the host, confirm each source UUID, filesystem type, mount path, and mount options. Required mount units must enter active state before kubelet:

```
sudo findmnt --mountpoint /var/cpaas
sudo systemctl is-active "$(systemd-escape --path --suffix=mount /var/cpa
as)"
sudo systemctl show "$(systemd-escape --path --suffix=mount /var/cpaas)"
\
  -p ActiveEnterTimestamp
sudo systemctl show kubelet.service -p ActiveEnterTimestamp
```

Step 6: Scale Down and Reuse the Inventory

Delete the owning CAPI `Machine` through the normal scale-down or replacement workflow. Do not delete the `MachineInventory` and do not remove its storage finalizer.

After drain, clean, and Deactivate complete, expect:

```
allocation-state=Available
status.storage.phase=Prepared
StoragePrepared=True
StorageActive=False/Inactive
business mount paths are inactive
filesystem UUIDs and data are retained
```

When the same inventory is allocated again, Activate reuses the existing ownership record and filesystem UUID; it does not format the device again. Generic pool allocation does not guarantee that a replacement Machine receives the same inventory. If an application must return to host-local data on a specific inventory, place that inventory first in a controlled ordered pool or use a dedicated pool and verify the selected `BaremetalMachine.status.machineInventoryRef` before relying on the data.

Step 7: Release Storage Ownership

Release a volume only while the inventory is unallocated and inactive. Removing a volume is non-destructive: it removes Storage v2 ownership and activation metadata but preserves the filesystem UUID and data.

To release all volumes, create an explicit empty declaration:

baremetal-storage-empty.yaml

```yaml
storage:
  volumes: []
```

Fetch the live inventory and render, dry-run, and apply a patch without an initialization counter:

```
kubectl -n "$BM_NS" get machineinventory.elemental.cattle.io "$BM_INVENTO
RY" -o json \
  >"$BM_INVENTORY_FILE"

"$BM_STORAGECTL" render-patch \
  --inventory "$BM_INVENTORY_FILE" \
  --storage /tmp/baremetal-storage-empty.yaml \
  >"$BM_PATCH_FILE"

kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE" --dry-run=server -o yaml
kubectl -n "$BM_NS" patch machineinventory.elemental.cattle.io "$BM_INVEN
TORY" \
  --type=merge --patch-file="$BM_PATCH_FILE"
```

Wait for `phase=Unmanaged`, `StoragePrepared=True/NoManagedVolumes`, an empty `appliedVolumes[]`, and `operation=null`. Verify retained data with a read-only mount when required. A storage finalizer may remain on the live inventory so a later object deletion still follows the correct cleanup order; do not remove it manually.

Failed Operations and Retry

Storage operations fail closed. A required device that is missing, stale, busy, too small, degraded, mounted from an unknown source, or no longer matches its declared UUID prevents allocation or Machine readiness. The controller never selects a replacement disk automatically.

After correcting the external cause of an operation whose `status.storage.operation.phase` is `Failed`, increment `spec.storage.retryNonce` while keeping `volumes[]` unchanged and render a new patch with `storagectl`. `retryNonce` creates another execution attempt for the same logical operation. It does not authorize a first format, bypass preflight, or permit a different storage declaration.

Do not use retry as a recovery mechanism for an unknown or partially mutated declaration. When `pendingSpecHash` is non-empty or the failed operation reports `mutationState=DurableStateChanged` or `Unknown`, keep the original declaration and allow the same operation journal to converge. Contact technical support before considering any break-glass finalizer or ownership action.

Restrictions

- `volumes[]` cannot change while the inventory is allocated or any managed volume is active.
- A prepared volume's `deviceID`, filesystem policy/type, or name cannot be changed in place. Release the old volume first, wait for completion, and then add the replacement declaration.
- Storage v2 does not move data between inventories. Inventory-owned data stays with the physical host or LUN identified by that inventory.
- Multipath support manages the aggregate `wwid:` map only. It does not configure multipath, SAN zoning, LUN masking, fencing, or multi-host read/write filesystems.
- `/var/lib/kubelet`, `/var/lib/containerd`, `/var/lib/etcd`, `/etc`, `/usr`, `/boot`, and other system paths are protected. Use business paths such as `/var/cpaas`, `/data/*`, `/srv/*`, or `/mnt/*` that do not overlap protected paths.
- Do not manage the same device or path through both `MachineInventory.spec.storage` and kubeadm/cloud-init `disk_setup`, `fs_setup`, `mounts`, custom systemd units, or host scripts.

Related Documentation

- Bare Metal Provider

- Create a Cluster on Bare Metal
- Manage Nodes on Bare Metal
- Upgrade Kubernetes on Bare Metal
- MachineInventory API

Manage Nodes on Huawei DCS

This document explains how to manage worker nodes using Cluster API `Machine` resources.

> **INFO**
>
> To upgrade the Kubernetes version of the nodes, see Upgrade Kubernetes on Huawei DCS.

TOC

- Prerequisites
- Overview
- Node Management Operations
  - Scaling Worker Nodes
    - Adding Worker Nodes
    - Removing Worker Nodes
  - Changing Machine Infrastructure
  - Updating Bootstrap Templates
- Managing Node Pools Using the Web UI
  - Viewing Node Pools
  - Adding a Worker Node Pool
  - Deleting a Worker Node Pool
  - Viewing Conditions (Control Plane Only)
- Troubleshooting

A new worker `Machine` stays in `Provisioning` and the DCS VM stays in `creating`

Next Steps

Prerequisites

> **WARNING**
>
> **Important Prerequisites**
>
> - The control plane must be deployed before performing node operations. See Create Cluster for setup instructions.
> - Ensure you have proper access to the DCS platform and required permissions.
> - To manage the nodes of the `global` cluster itself, the provider must be installed on `global` first; see Manage Nodes on Alauda OS Clusters.

> **INFO**
>
> **Configuration Guidelines** When working with the configurations in this document:
>
> - Only modify values enclosed in `<>` brackets
> - Replace placeholder values with your environment-specific settings
> - Preserve all other default configurations unless explicitly required

Overview

Worker nodes are managed through Cluster API `Machine` resources, providing declarative and automated node lifecycle management. A worker node pool is built from four resources:

1. **IP-Hostname Pool Configuration** - Network settings for worker nodes
2. **Machine Template Setup** - VM specifications

3. **Bootstrap Configuration** - Node initialization and join settings
4. **Machine Deployment** - Orchestration of node creation and management

To add another worker node pool with YAML, follow Deploy Worker Nodes on the cluster creation page.

Node Management Operations

This section covers common operational tasks for managing worker nodes, including scaling, machine infrastructure changes, and bootstrap template updates.

> **INFO**
>
> **Cluster API Framework** Node management operations are based on the Cluster API framework. For detailed information, refer to the official Cluster API documentation ↗.

Scaling Worker Nodes

Worker node scaling allows you to adjust cluster capacity based on workload demands. The Cluster API manages the node lifecycle automatically through the MachineDeployment resource.

Adding Worker Nodes

Increase the number of worker nodes to handle increased workload or add new capacity.

Use Case: Scale up cluster to add more compute resources

Prerequisites:

- Verify the IP pool has sufficient available IP addresses for new nodes
- Ensure the DCS platform has adequate resources to provision new VMs

Procedure:

1. **Check Current Node Status**

View the current machines in the cluster:

```
# List all machines in the cluster
kubectl get machines -n cpaas-system

# List machines for a specific MachineDeployment
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-nam
e=<cluster-name>-md-0
```

2. **Extend IP Pool**

   Before scaling up, add new IP configurations to the pool for the additional nodes.

   > **INFO**
   >
   > **IP Pool Expansion** The IP pool must contain at least as many entries as the desired replica count. Add new IP entries for each additional worker node you plan to deploy.

   Add IP entries to the pool:

   First, export the current pool configuration to preserve existing entries:

   ```
   kubectl get dcsiphostnamepool <cluster-name>-worker-pool -n cpaas-syste
   m -o yaml
   ```

   Then use the following command to add new IP configurations. The `pool` array must include **all existing entries** plus the new entries:

```
kubectl patch dcsiphostnamepool <cluster-name>-worker-pool -n cpaas-sys
tem \
  --type='merge' -p='
{
  "spec": {
    "pool": [
      {
        "ip": "<existing-ip-1>",
        "mask": "<worker-cidr-prefix-length>",
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<existing-hostname-1>",
        "machineName": "<existing-machine-name-1>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "<existing-ip-2>",
        "mask": "<worker-cidr-prefix-length>",
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<existing-hostname-2>",
        "machineName": "<existing-machine-name-2>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "<new-worker-ip-1>",
        "mask": "<worker-cidr-prefix-length>",
```

```
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<new-worker-hostname-1>",
        "machineName": "<new-worker-machine-name-1>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "<new-worker-ip-2>",
        "mask": "<worker-cidr-prefix-length>",
        "gateway": "<worker-gateway>",
        "dns": "<worker-dns>",
        "hostname": "<new-worker-hostname-2>",
        "machineName": "<new-worker-machine-name-2>",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "<datastore-cluster-name>",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      }
    ]
  }'
```

WARNING

Important Notes

- The `pool` array must include **all existing entries** plus the new entries you want to add
- Copy the existing entries from the exported YAML to avoid data loss
- `kubectl patch --type='merge'` replaces the entire `spec.pool` array, so copy every existing `persistentDisk` block unchanged, including `isThin` if it is present, unless you

are intentionally adding new disks

- Ensure each new entry has unique `ip`, `hostname`, and `machineName` values
- If new worker nodes also need the platform-required `/var/cpaas` disk, declare it in each new entry's `persistentDisk`
- Network parameters (`mask`, `gateway`, `dns`) typically match existing entries

Example: Adding 2 new nodes to an existing pool of 3 nodes

```
# Current pool has 3 entries (10.0.1.11, 10.0.1.12, 10.0.1.13)
# Adding 2 more entries for nodes 4 and 5
kubectl patch dcsiphostnamepool worker-pool-1-ippool -n cpaas-system \
  --type='merge' -p='
{
  "spec": {
    "pool": [
      {
        "ip": "10.0.1.11",
        "mask": "24",
        "gateway": "10.0.1.1",
        "dns": "10.0.0.2",
        "hostname": "worker-node-1",
        "machineName": "worker-vm-1",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "shared-datastore-cluster",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "10.0.1.12",
        "mask": "24",
        "gateway": "10.0.1.1",
        "dns": "10.0.0.2",
        "hostname": "worker-node-2",
        "machineName": "worker-vm-2",
        "persistentDisk": [
          {
            "slot": 0,
            "quantityGB": 40,
            "datastoreClusterName": "shared-datastore-cluster",
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      },
      {
        "ip": "10.0.1.13",
```

```
    "mask": "24",
    "gateway": "10.0.1.1",
    "dns": "10.0.0.2",
    "hostname": "worker-node-3",
    "machineName": "worker-vm-3",
    "persistentDisk": [
      {
        "slot": 0,
        "quantityGB": 40,
        "datastoreClusterName": "shared-datastore-cluster",
        "path": "/var/cpaas",
        "format": "xfs"
      }
    ]
  },
  {
    "ip": "10.0.1.14",
    "mask": "24",
    "gateway": "10.0.1.1",
    "dns": "10.0.0.2",
    "hostname": "worker-node-4",
    "machineName": "worker-vm-4",
    "persistentDisk": [
      {
        "slot": 0,
        "quantityGB": 40,
        "datastoreClusterName": "shared-datastore-cluster",
        "path": "/var/cpaas",
        "format": "xfs"
      }
    ]
  },
  {
    "ip": "10.0.1.15",
    "mask": "24",
    "gateway": "10.0.1.1",
    "dns": "10.0.0.2",
    "hostname": "worker-node-5",
    "machineName": "worker-vm-5",
    "persistentDisk": [
      {
        "slot": 0,
        "quantityGB": 40,
        "datastoreClusterName": "shared-datastore-cluster",
```

```
            "path": "/var/cpaas",
            "format": "xfs"
          }
        ]
      }
    ]
  }'
```

3. **Verify IP Pool Capacity**

   After extending the IP pool, verify it has sufficient entries for the desired replica count:

   ```
   kubectl get dcsiphostnamepool -n cpaas-system <cluster-name>-worker-poo
   l -o yaml
   ```

   Check that the pool contains at least as many entries as the desired replica count.

4. **Scale Up the MachineDeployment**

   Update the `replicas` field to the desired number of nodes:

   ```
   kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": <new-replica-count>}]'
   ```

   Example: Scale from 3 to 5 nodes

   ```
   kubectl patch machinedeployment worker-pool-1 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": 5}]'
   ```

5. **Monitor the Scaling Progress**

   Watch the machine creation process:

   ```
   # Watch machines being created
   kubectl get machines -n cpaas-system -w

   # Check MachineDeployment status
   kubectl get machinedeployment <cluster-name>-md-0 -n cpaas-system
   ```

The Cluster API controller will automatically create new machines based on the MachineDeployment template.

6. **Verify Nodes Joined the Cluster**
   Switch to the target cluster context and verify the new nodes:

```
# Switch to target cluster context
kubectl config use-context <target-cluster-context>

# Check all nodes are Ready
kubectl get nodes
```

The new nodes should appear in the list and transition to `Ready` status.

> **INFO**
>
> **Rolling Update Behavior** When scaling up, new nodes are created immediately without affecting existing nodes. This ensures zero-downtime scaling.

Removing Worker Nodes

Decrease the number of worker nodes to reduce cluster capacity or remove underutilized resources. The Cluster API supports two removal strategies:

1. **Random removal**: Reduce replicas, the platform randomly selects and deletes machines
2. **Targeted removal**: Mark specific machines for deletion, then reduce replicas (recommended for IP recovery)

> **INFO**
>
> **IP Recovery Scenario** When you need to recycle specific machine IPs (e.g., for reassignment or IP pool management), use the targeted removal method. The deletion annotation ensures the platform deletes the marked machines, not random ones.

> **WARNING**
>
> **Data Loss Warning** Scaling down removes nodes and their associated VMs. Ensure:

- Workloads can tolerate node loss through proper replication
- No critical data is stored only on the nodes being removed
- Applications are designed for horizontal scaling

Declared persistent disks in `DCSIpHostnamePool.spec.pool[].persistentDisk` are not deleted just because a Machine is replaced. They remain available for reuse while the corresponding IP slot stays in the pool. Removing the IP slot from the pool, deleting the pool, or deleting the cluster can trigger persistent-volume cleanup.

Random Removal

Use Case: Scale down cluster where any node can be removed (no specific IP requirements)

Procedure:

1. **Identify Current Machine Status**

   View the current machines in the MachineDeployment:

   ```
   kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-nam
   e=<cluster-name>-md-0
   ```

2. **Scale Down the MachineDeployment**

   Update the `replicas` field to reduce the node count:

   ```
   kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": <new-replica-count>}]'
   ```

   Example: Scale from 5 to 3 nodes

   ```
   kubectl patch machinedeployment worker-pool-1 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": 3}]'
   ```

   The Cluster API controller will randomly select and delete machines to match the desired replica count.

3. **Monitor the Removal Progress**

Watch the machine deletion process:

```
kubectl get machines -n cpaas-system -w
```

The Cluster API controller will:

- Drain the selected nodes (evict pods if possible)
- Delete the underlying VMs from the DCS platform
- Remove the machine resources

4. **Verify Nodes Removed**

   Switch to the target cluster context:

   ```
   kubectl config use-context <target-cluster-context>
   kubectl get nodes
   ```

   The removed nodes should no longer appear in the list.

Targeted Removal

Use Case: Remove specific machines (e.g., for IP recovery, replace unhealthy nodes)

Procedure:

1. **Identify Machines to Remove**

   View the current machines:

   ```
   kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-nam
   e=<cluster-name>-md-0
   ```

   Note the `<machine-name>` of the machines you want to remove.

2. **Annotate Machines for Deletion**

   Mark the specific machines for deletion:

   ```
   kubectl patch machine <machine-name> -n cpaas-system \
     --type='merge' -p='{"metadata": {"annotations": {"cluster.x-k8s.io/de
   lete-machine": "true"}}}'
   ```

Repeat for each machine you want to remove.

Example: Remove two specific machines

```
kubectl patch machine worker-pool-1-abc123 -n cpaas-system \
  --type='merge' -p='{"metadata": {"annotations": {"cluster.x-k8s.io/de
lete-machine": "true"}}}'

kubectl patch machine worker-pool-1-def456 -n cpaas-system \
  --type='merge' -p='{"metadata": {"annotations": {"cluster.x-k8s.io/de
lete-machine": "true"}}}'
```

3. **Scale Down the MachineDeployment**

   After annotating the machines, reduce the replica count:

   > **INFO**
   >
   > **Replica Count Must Match Annotated Machines** Reduce replicas by exactly the number of annotated machines.
   >
   > - If you reduce by fewer, not all annotated machines will be removed
   > - If you reduce by more, additional machines will be randomly selected for deletion

   ```
   kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
     --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
   e": <new-replica-count>}]'
   ```

   Example: If you annotated 2 machines, reduce replicas by exactly 2 (e.g., from 5 to 3)

   The platform will delete the **annotated** machines, not randomly selected ones.

4. **Monitor the Removal Progress**

   Watch the machine deletion process:

   ```
   kubectl get machines -n cpaas-system -w
   ```

5. **Verify Nodes Removed**

   Switch to the target cluster context:

```
kubectl config use-context <target-cluster-context>
kubectl get nodes
```

The removed nodes should no longer appear in the list.

Changing Machine Infrastructure

To change the machine specification of worker nodes (CPU, memory, disk, VM template), create a new machine template, then point the `MachineDeployment` at it. The worker nodes are replaced one by one.

1. **Create a New Machine Template**
   Export the `DCSMachineTemplate` referenced by your `MachineDeployment` to a new file:

   ```
   kubectl get dcsmachinetemplate <current-template-name> -n cpaas-system
   -o yaml > new-template.yaml
   ```

   In the new file:

   - Set `metadata.name` to a unique name
   - Modify the required values (`spec.template.spec.vmTemplateName` , `spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity` , `spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity` , `spec.template.spec.vmConfig.dcsMachineDiskSpec`)
   - Remove the server-generated metadata (`resourceVersion` , `uid` , `generation` , `creationTimestamp` , `managedFields` , the `kubectl.kubernetes.io/last-applied-configuration` annotation) and the entire `status` field
   - Leave `spec.template.spec.providerID` unset

   Keep pool-managed persistent disks, including `/var/cpaas` , in `DCSIpHostnamePool.spec.pool[].persistentDisk` and additional NICs in `DCSIpHostnamePool.spec.pool[].additionNic[]` . Use `dcsMachineDiskSpec` only for the system disk and template-local disks.
   Apply the new template:

```
kubectl apply -f new-template.yaml -n cpaas-system
```

2. **Update Machine Deployment**

   Point the `MachineDeployment` at the new template:

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='merge' -p='{"spec":{"template":{"spec":{"infrastructureRef":
{"name":"<new-template-name>"}}}}}'
```

3. **Monitor the Rolling Update**

   - Worker nodes are replaced one by one with the new specification
   - Any disks declared in `DCSIpHostnamePool.spec.pool[].persistentDisk` are detached from the old VM and reattached to the replacement VM. If the pool uses persistent disks, keep `spec.strategy.rollingUpdate.maxSurge` at `0` on the `MachineDeployment`
   - Monitor the progress:

```
kubectl get machinedeployment <cluster-name>-md-0 -n cpaas-system
kubectl get machines -n cpaas-system -w
```

To change the machine specification of control plane nodes, use the same steps and patch `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` instead.

To move the nodes to the Alauda OS image of a new Kubernetes version, follow Upgrade Kubernetes on Huawei DCS instead.

If you are migrating an existing cluster from the old template-disk layout to pool-managed persistent disks, follow Migrate Existing Huawei DCS Clusters to Pool-Managed Persistent Disks before you rely on upgrade-time data preservation.

Updating Bootstrap Templates

Bootstrap templates (KubeadmConfigTemplate) are used by MachineDeployment and MachineSet resources. Changes to existing templates do not automatically trigger rollouts of existing machines; only new machines use the updated template.

Update Process:

1. **Export Existing Template**

```
kubectl get KubeadmConfigTemplate <template-name> -o yaml > new-template.yaml
```

2. **Modify Configuration**

- Update the desired fields in the exported YAML
- For Kubernetes 1.35 or later, add the required kubelet patch settings to `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`
- Change the `metadata.name` to a new unique name
- Remove extraneous metadata fields (`resourceVersion` , `uid` , `creationTimestamp` , etc.)

3. **Create New Template**

```
kubectl apply -f new-template.yaml
```

4. **Update MachineDeployment**

- Modify the MachineDeployment resource
- Update `spec.template.spec.bootstrap.configRef.name` to reference the new template
- Apply the changes to trigger a rolling update

> **INFO**
>
> **Template Rollout Behavior** Existing machines continue using the old bootstrap configuration. Only newly created machines (during scaling or rolling updates) will use the updated template.

Managing Node Pools Using the Web UI

Adding, deleting, and viewing node pools through Fleet Essentials are not affected by the upgrade limitations described in Fleet Essentials Boundary.

Node pools provide a declarative way to manage groups of nodes with identical configurations. You can view, add, and delete worker node pools through the web UI.

Requirement: This workflow requires Fleet Essentials. The `persistentDisk` declaration on `DCSIpHostnamePool` remains YAML-only and is not exposed in the node-pool UI.

If the node pool relies on pool-managed persistent disks, prepare or update the corresponding `DCSIpHostnamePool` entry with YAML before you use the web UI workflow here.

> **INFO**
>
> **Navigation**: Clusters → Clusters → Select cluster → Node Pools Tab

Viewing Node Pools

The Node Pools Tab displays all node pools in the cluster:

Control Plane Node Pool:

- Fixed at 3 replicas for high availability
- Displays Kubernetes version with upgrade indicator if available
- Shows Conditions link for detailed status

Worker Node Pools:

- Customizable replica counts
- Individual Kubernetes version management
- Scale and upgrade operations

Node Pool Card Information:

Field	Description
Type	Control Plane / Worker Node

Field	Description
Name	Resource name
Status	Badge showing pool health
Replicas	Current count (with Max Surge/Max Unavailable for workers)
SSH Authorized Keys	List of SSH public keys
Kubernetes Version	Current version (upgrade indicator if available)
Machine Template	Associated template name
Conditions	Link to conditions list (Control Plane only)

Adding a Worker Node Pool

Navigation: Node Pools Tab → Click **Add Worker Node Pool**

Form Fields:

Field	Type	Required	Description
Pool Name	text	Yes	Unique identifier for the node pool
Machine Template	dropdown	Yes	Filter by Type: Worker Node and compatible Kubernetes version
Replicas	number	Yes	Number of nodes in the pool
Max Surge	number	No	Default: 0, must be ≥ 0. Keep this value at `0` if the node pool relies on persistent disks
Max Unavailable	number	No	Default: 1, must be ≥ 0. When maxSurge = 0, must be > 0 and ≤ Replicas

Field	Type	Required	Description
SSH Authorized Keys	text	No	Add multiple SSH public keys

Validation:

- Pool name must be unique within the cluster
- IP Pool must have sufficient available IP addresses (≥ Replicas)
- maxSurge/maxUnavailable constraints must be satisfied
- If the node pool relies on persistent disks, keep `maxSurge = 0` so Machines are replaced one by one

Tip: Prefix the pool name with the cluster name followed by a hyphen (e.g., `mycluster-worker-1`) to avoid naming conflicts.

After creation, new nodes appear in the **Nodes** Tab. The number of nodes equals the configured Replicas value.

Deleting a Worker Node Pool

Steps:

1. Click the delete icon on the Worker Node Pool card
2. Confirm deletion in the dialog

> **WARNING**
>
> Deleting a worker node pool permanently removes all associated nodes and machines. Ensure workloads can tolerate the loss of these nodes through proper replication.

Viewing Conditions (Control Plane Only)

Click the **Conditions** link on the Control Plane Node Pool card to view detailed status information.

Conditions List:

Type	Status	Last Transition Time	Reason	Message
Condition Type	Status	Timestamp	Reason	Human-readable details

---

Troubleshooting

A new worker `Machine` stays in `Provisioning` and the DCS VM stays in `creating`

The diagnostic flow is identical for control-plane and worker `Machine` objects — when a `DCSMachine` reaches `Provisioning` with `status.cdRomFile` already set (DCS API auth and ISO upload succeeded), but the underlying DCS VM never schedules to a host, the failure is on the DCS platform side and not in the controller.

See Infrastructure → Troubleshooting: VM stuck in `creating` for the full diagnostic ladder, which covers:

- Confirming the symptom is a placement stall (not a slow boot).
- Checking datastore free space.
- Checking host CPU / memory utilisation and overcommit ratios.
- Coordinating with the DCS platform team to rebalance load when a single DCS Host is saturated.
- Escalation path when the deploy task is genuinely deadlocked.

For control-plane high availability across multiple physical hosts, see Infrastructure → Cross-Host High Availability for Control Plane.

The same diagnostic applies whether the stuck node is the first control-plane node of a fresh cluster, a new worker added through `MachineDeployment` scaling, or a replacement worker

created during a rolling update.

Next Steps

- Create Clusters
- Upgrade Clusters

Manage Nodes on Huawei Cloud Stack

This document explains how to manage worker nodes using Cluster API `Machine` resources on the Huawei Cloud Stack platform.

INFO

To upgrade the Kubernetes version of the nodes, see Upgrade Kubernetes on Huawei Cloud Stack.

TOC

- Prerequisites
- Overview
- Using the Web UI
  - Viewing Node Pools
  - Adding a Worker Node Pool
  - Managing Node IP Addresses
  - Deleting a Worker Node Pool
- Using YAML
  - Scaling Worker Nodes
    - Adding Worker Nodes
    - Removing Worker Nodes
  - Changing Machine Infrastructure
  - Updating Bootstrap Templates
- Verification

Troubleshooting

Viewing Controller Logs

Common Issues

Prerequisites

WARNING

Important Prerequisites

- The control plane must be deployed before performing node operations. See Create Cluster for setup instructions.
- Ensure you have proper access to the HCS platform and required permissions.
- To manage the nodes of the `global` cluster itself, the provider must be installed on `global` first; see Manage Nodes on Alauda OS Clusters.

When using the YAML examples in this document, replace only values enclosed in `<>` with environment-specific values. Preserve the remaining fields unless your cluster policy requires a different value.

Overview

Worker nodes are managed through Cluster API `Machine` resources, providing declarative and automated node lifecycle management. A worker node pool is built from four resources:

1. **Machine Configuration Pool** - Network settings for worker nodes
2. **Machine Template** - VM specifications
3. **Bootstrap Configuration** - Node initialization settings
4. **Machine Deployment** - Orchestration of node creation and management

You can manage these resources through the web UI or with YAML manifests. The web UI provides a guided experience and requires Fleet Essentials; YAML works without it.

Using the Web UI

Requirement: This workflow requires Fleet Essentials `1.0.2` or later. Without it, manage node pools with the YAML workflows in the Using YAML section.

The Node Pools tab lets you view the control plane and worker node pools, add or delete worker node pools, and edit the reserved IP addresses of an existing pool.

> **INFO**
>
> **Navigation**: Clusters → Clusters → Select cluster → Node Pools tab

Viewing Node Pools

The Node Pools tab shows the Control Plane Node Pool and each Worker Node Pool as a card. Both card types share the same fields:

Field	Description
Name	Resource name. Links to the underlying `KubeadmControlPlane` (control plane) or `MachineDeployment` (worker). The control plane card carries a **Control Plane** tag.
Status	Badge showing pool health.
Conditions	Count of status conditions; click to view the condition list.
Ready Replicas	Ready node count over the desired count (for example, `3 / 3`).
Kubernetes Version	Current Kubernetes version of the pool.
Machine Template	Associated `HCSMachineTemplate`; click to open it.

Field	Description
Config Pool	Associated `HCSMachineConfigPool`; click to open it.
SSH Authorized Keys	Number of configured public keys.
Update Strategy	For the control plane, nodes are replaced one at a time, each keeping its node IP. For workers, a rolling update governed by the pool's Max Surge and Max Unavailable values.

Adding a Worker Node Pool

Click **Add Worker Node Pool** and complete the dialog. The fields match Step 5 of the creation wizard: pool name (prefixed with `<cluster-name>-`), Flavor, Availability Zone, Machine Configs (hostname, networks, persistent disks), System Disk, Data Volumes, Replicas, Rollout Strategy, and SSH Authorized Keys. New nodes appear in the Nodes tab after the pool is created.

The selected Availability Zone applies to every node in the new worker pool. To place worker nodes in another zone, create a separate worker pool and select that zone; one pool does not automatically spread its replicas across zones. The dropdown disables zones that HCS reports as unavailable. See Availability Zone Placement for automatic selection, persistent disk, and control plane placement constraints.

Managing Node IP Addresses

Each node in an HCS pool uses a reserved static IP address from its subnet. To change which addresses a pool reserves — for example, to add capacity before scaling the pool up — open the pool card's action menu and select **Update Config Pool** to open the **Manage IPs** dialog.

The dialog edits the machine configurations of the pool's `HCSMachineConfigPool`. For each node you can set the hostname, the networks (each an IP address chosen from its subnet, plus the subnet), and the persistent disks. IP selection follows the same rules as the creation wizard: choose the subnet first, then pick an assignable address from the list; addresses already in use are not offered.

> **WARNING**
>
> HCS clusters use static IP addresses with no DHCP. Reserve enough addresses in the pool before you scale a node pool up — a new node cannot start unless the pool provides an address for it.

Deleting a Worker Node Pool

Open the Worker Node Pool card's action menu, select **Delete**, and confirm.

> **WARNING**
>
> Deleting a worker node pool permanently removes all of its nodes and the underlying ECS instances. Ensure workloads can tolerate the loss of these nodes through proper replication. The Control Plane Node Pool cannot be deleted.

Using YAML

Manage worker nodes with Cluster API manifests. To add another worker node pool with YAML, follow Deploy Worker Nodes on the cluster creation page.

Scaling Worker Nodes

Worker node scaling allows you to adjust cluster capacity based on workload demands.

Adding Worker Nodes

Increase the number of worker nodes to handle increased workload.

Procedure:

1. **Check Current Node Status**

```
# List all machines in the cluster
kubectl get machines -n cpaas-system

# List machines for a specific MachineDeployment
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/deployment-nam
e=<cluster-name>-md-0
```

2. **Extend Configuration Pool**

   Add new machine configurations to the pool for the additional nodes. If the new workers need preserved node-local state such as `/var/cpaas`, include the matching `persistentDisks[]` entries in each new configuration.

```
kubectl get hcsmachineconfigpool <cluster-name>-worker-pool -n cpaas-sy
stem -o yaml
```

   Modify the pool to include new IP entries, then apply:

```
kubectl apply -f <updated-pool-config.yaml>
```

   When you edit the pool, keep all existing `configs[]` entries and their accepted `persistentDisks[]` entries unchanged unless you are intentionally appending a new disk slot.

3. **Scale Up the MachineDeployment**

   Update the `replicas` field to the desired number of nodes:

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
e": <new-replica-count>}]'
```

4. **Monitor the Scaling Progress**

```
# Watch machines being created
kubectl get machines -n cpaas-system -w

# Check MachineDeployment status
kubectl get machinedeployment <cluster-name>-md-0 -n cpaas-system
```

Removing Worker Nodes

Decrease the number of worker nodes to reduce cluster capacity.

> **WARNING**
>
> **Data Loss Warning**
>
> Scaling down removes worker nodes and their ECS instances. Template-owned `dataVolumes[]` are not preserved. Pool-managed persistent disks declared in `HCSMachineConfigPool.spec.configs[].persistentDisks[]` remain tracked by the pool and can be reused while the corresponding hostname entry stays in the pool. Ensure:
>
> - Workloads can tolerate node loss through proper replication
> - No critical data is stored only on the nodes being removed
> - Applications are designed for horizontal scaling

Procedure:

1. **Scale Down the MachineDeployment**

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='json' -p='[{"op": "replace", "path": "/spec/replicas", "valu
e": <new-replica-count>}]'
```

2. **Monitor the Removal Progress**

```
kubectl get machines -n cpaas-system -w
```

The Cluster API controller will:

- Drain the selected nodes (evict pods if possible)
- Delete the underlying VMs from the HCS platform
- Remove the machine resources

Changing Machine Infrastructure

To change worker machine specifications (CPU, memory, disk, VM image), create a new machine template and point the `MachineDeployment` at it. The worker nodes are then replaced one by one. Follow these steps:

Note: HCS `dataVolumes[]` are not preserved during replacement. To preserve node-local state such as `/var/cpaas`, declare it in `HCSMachineConfigPool.spec.configs[].persistentDisks[]` before the rollout and keep `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0`. If your cluster still keeps these disks in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`, first complete Migrate Existing Huawei Cloud Stack Clusters to Pool-Managed Persistent Disks.

1. **Create New Machine Template**

   Copy the existing `HCSMachineTemplate` and modify the required values:

   - `imageName` - VM image
   - `flavorName` - Instance type
   - `rootVolume.size` - System disk size
   - `dataVolumes` - Temporary data disk configurations

   If you need to add a new pool-managed persistent disk, append it to the worker `HCSMachineConfigPool` first. The provider creates or claims the EVS disk, but the running ECS does not mount it until this rolling replacement creates a replacement worker.

   ```
   kubectl get hcsmachinetemplate <current-template> -n cpaas-system -o yaml > new-template.yaml
   ```

   Then edit `new-template.yaml` before applying:

   - Change `metadata.name` to `<new-template>`
   - Leave runtime identity fields unset, including `spec.template.spec.providerID` and `spec.template.spec.serverId`
   - Remove server-generated fields such as:
     - `metadata.resourceVersion`
     - `metadata.uid`
     - `metadata.creationTimestamp`

- `metadata.managedFields`
- `status`

2. **Deploy New Template**

```
kubectl apply -f new-template.yaml -n cpaas-system
```

3. **Update Machine Deployment**

   Modify the `MachineDeployment` to reference the new template:

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='merge' -p='{"spec":{"template":{"spec":{"infrastructureRef":
{"name":"<new-template>"}}}}}'
```

4. **Monitor Rolling Update**

```
kubectl get machines -n cpaas-system -w
```

To change the machine specification of control plane nodes, use the same steps and patch `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` instead.

Updating Bootstrap Templates

To change the bootstrap configuration of worker nodes (for example kubelet settings or the commands that run when a node joins), create a new `KubeadmConfigTemplate` and point the `MachineDeployment` at it. Modifying an existing template in place does not roll out existing machines; only newly created machines pick up the change.

1. **Export the Existing Template**

```
kubectl get kubeadmconfigtemplate <cluster-name>-worker-kct -n cpaas-sy
stem -o yaml > new-bootstrap-template.yaml
```

2. **Modify the Template**

   Edit `new-bootstrap-template.yaml`:

- Set `metadata.name` to a new unique name, `<new-bootstrap-template>`

- Remove server-generated metadata (`resourceVersion` , `uid` , `creationTimestamp` , `managedFields` , the `kubectl.kubernetes.io/last-applied-configuration` annotation) and the entire `status` field
- Edit the fields you want to change. Keep the existing entries of the kubelet patch file `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json` , including the certificate paths, and change only what you need
- For Kubernetes 1.35 or later, add the required kubelet patch settings to that file

3. **Apply the New Template**

```
kubectl apply -f new-bootstrap-template.yaml -n cpaas-system
```

4. **Update the MachineDeployment**

```
kubectl patch machinedeployment <cluster-name>-md-0 -n cpaas-system \
  --type='merge' -p='{"spec":{"template":{"spec":{"bootstrap":{"configR
ef":{"name":"<new-bootstrap-template>"}}}}}}'
```

The rolling replacement of the worker nodes then starts:

```
kubectl get machines -n cpaas-system -w
```

Verification

After deploying worker nodes, verify the deployment:

```
# Check machine status
kubectl get machines -n cpaas-system

# Verify nodes are Ready
kubectl get nodes

# Check MachineDeployment status
kubectl get machinedeployment -n cpaas-system
```

Troubleshooting

Viewing Controller Logs

```
# View HCS controller logs
kubectl logs -n cpaas-system deployment/hcs-controller-manager

# View machine details
kubectl describe hcsmachine <machine-name> -n cpaas-system
```

Common Issues

Node fails to join cluster

- Verify the VM template matches the Kubernetes version
- Check network connectivity between nodes
- Ensure the configuration pool has available entries

Machine stuck in provisioning

- Check HCS platform for resource availability
- Verify credentials and permissions
- Review controller logs for error messages

Configure FQDN Hostname on Existing HCS Clusters

Use this guide when an existing Huawei Cloud Stack (HCS) cluster needs nodes whose POSIX `hostname` returns the short label and whose `hostname -f` returns the full FQDN, and the cluster was originally created before `cluster-api-provider-hcs` v1.0.1.

Applications that depend on a working `hostname` / `hostname -f` split — TLS server certificate SAN matching, Kerberos SPN composition, monitoring labels keyed on the short name, and any code that calls `getfqdn()` — require the node to be initialized by `cluster-api-provider-hcs` v1.0.1 or later. Cloud-init only runs on first boot, so existing nodes do not pick up the new behavior automatically; the node has to be replaced through the standard Cluster API rolling-replacement flow with the new controller in place.

INFO

Version

The Infrastructure Provider for Huawei Cloud Stack delivered with this release renders dotted hostnames correctly. Controller versions earlier than `cluster-api-provider-hcs` v1.0.1 render dotted `HCSMachineConfigPool.spec.configs[].hostname` values as the full FQDN string in the POSIX hostname, which breaks `hostname -f` and can also block kubeadm init.

TOC

Overview

Prerequisites

Step-by-step procedure

1. Confirm the controller is upgraded
2. Decide which pools and templates are in scope
3. Extend the control plane pool with a surge slot
4. Create a new `HCSMachineTemplate` for the control plane
5. Patch `KubeadmControlPlane` to reference the new template
6. Repeat for each `MachineDeployment` (workers)
7. Verify hostname behavior on each new node

Storage and data caveats

Rollback

Related

Overview

Two conditions must both be true for a node to expose the supported hostname / FQDN behavior:

1. The node's `HCSMachineConfigPool.spec.configs[].hostname` entry contains a dot (FQDN form, for example `master-1.example.org`).
2. The node was first booted by `cluster-api-provider-hcs` v1.0.1 or later.

Existing nodes booted under the older controller continue to use the cloud-init user-data they received on first boot, so editing manifests or re-applying user-data does not retroactively change their hostname behavior. The supported migration path is to replace those nodes through Cluster API rolling replacement, with the upgraded controller already installed and Ready on the management cluster.

Manual on-node edits to `/etc/hostname` and `/etc/hosts` do not persist on an Alauda OS node. Cloud-init re-renders these files on every boot, including reboots triggered by `transactional-update`, OS patching, or `systemctl reboot`.

Prerequisites

- The Infrastructure Provider for Huawei Cloud Stack on the management cluster is the version delivered with this release (`cluster-api-provider-hcs` v1.0.1 or later), and the controller Deployment in `cpaas-system` is Ready.
- The existing workload cluster is healthy: all current `Machine` objects report `phase=Running`, all `Node` objects report `Ready`, and no rolling replacement is in flight.
- Storage caveats reviewed (see Storage and data caveats).
- The target VM image used by the existing `HCSMachineTemplate` is still present in the HCS environment. (No image change is needed; the migration only renames the template to trigger replacement.)
- The control plane `HCSMachineConfigPool` has at least one unused `(hostname, IP)` slot available so Cluster API can surge a new control plane machine before draining the old one. If the existing pool is sized exactly to the current replica count, extend it with one more entry before starting.

Step-by-step procedure

1. Confirm the controller is upgraded

```
kubectl get deployment -n cpaas-system -l app=cluster-api-provider-hcs-manager -o wide
kubectl rollout status deployment -n cpaas-system cluster-api-provider-hcs-manager
```

The Deployment must show a v1.0.1-or-later image tag and report all replicas Ready. Replacing nodes before the controller is upgraded re-renders the old user-data on the new VMs, leaving the cluster in the same state.

2. Decide which pools and templates are in scope

The migration is only required for pools whose entries contain a dot. List the existing `HCSMachineConfigPool` resources and check each `spec.configs[].hostname`:

```
kubectl get hcsmachineconfigpool -n cpaas-system
kubectl get hcsmachineconfigpool <pool-name> -n cpaas-system \
  -o jsonpath='{range .spec.configs[*]}{.hostname}{"\n"}{end}'
```

Pools whose entries are all short labels (no dot) are not affected by the migration and do not need to be touched. Continue only for pools that contain dotted entries.

3. Extend the control plane pool with a surge slot

Cluster API replaces control plane machines one at a time. The default rollout strategy needs a free `(hostname, IP)` slot in the pool to bring up the replacement machine before terminating the old one. If the existing control plane pool has exactly as many entries as `KubeadmControlPlane.spec.replicas`, add one more entry now:

```
kubectl edit hcsmachineconfigpool <cp-pool-name> -n cpaas-system
```

Add an extra entry under `spec.configs` with a new hostname and a free IP from the same subnet, then save. The new entry is held in reserve and consumed only when the first machine is replaced.

4. Create a new `HCSMachineTemplate` for the control plane

The new template can be a byte-for-byte copy of the existing one with a new `metadata.name`. The rename alone is what triggers Cluster API to roll machines and pick up the new user-data; you do not need to change `imageName`, `flavorName`, or any other field.

```
kubectl get hcsmachinetemplate <current-cp-template> -n cpaas-system -o y
aml > new-cp-template.yaml
```

Edit `new-cp-template.yaml`:

- Set `metadata.name` to a new value (for example, append `-fqdn` or a date suffix).
- Strip server-generated metadata (`resourceVersion`, `uid`, `generation`, `creationTimestamp`, `managedFields`, the `kubectl.kubernetes.io/last-applied-`

`configuration` annotation) and the entire `status` field.

- Leave runtime identity fields unset, including `spec.template.spec.providerID` and `spec.template.spec.serverId`. The HCS provider assigns these values when it creates the new VMs.

Apply the new template:

```
kubectl apply -f new-cp-template.yaml -n cpaas-system
```

Keep the previous template until the rollout is healthy; rollback uses it.

5. Patch `KubeadmControlPlane` to reference the new template

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system --type='merg
e' \
  -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<new-cp-te
mplate>"}}}}'
```

Cluster API now starts a rolling replacement of the control plane. Monitor progress:

```
kubectl get kubeadmcontrolplane <kcp-name> -n cpaas-system -w
kubectl get machines -n cpaas-system -l cluster.x-k8s.io/control-plane
```

Wait until every control plane `Machine` is the new one (the old `Machine` names disappear) and `KubeadmControlPlane.status.readyReplicas` equals `spec.replicas`.

6. Repeat for each `MachineDeployment` (workers)

For each `MachineDeployment` whose nodes need the new hostname behavior, repeat steps 4 and 5 with the worker template:

```
kubectl get hcsmachinetemplate <current-worker-template> -n cpaas-system
-o yaml > new-worker-template.yaml
# Edit metadata.name, strip status/server fields, apply
kubectl apply -f new-worker-template.yaml -n cpaas-system

kubectl patch machinedeployment <md-name> -n cpaas-system --type='merge'
\
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<new-work
er-template>"}}}}}'
```

Wait for each `MachineDeployment` to report `status.updatedReplicas` equal to `spec.replicas` and all worker `Machine` objects on the new template before moving on.

7. Verify hostname behavior on each new node

On each new node:

```
hostname
hostname -f
cat /etc/hosts
```

For a node whose pool config was `hostname: master-1.example.org`, expect:

- `hostname` returns `master-1` (short label, dot and everything after it stripped).
- `hostname -f` returns `master-1.example.org`.
- `/etc/hosts` contains an entry of the form `<loopback-or-node-ip> master-1.example.org master-1`.

If a new node reports the FQDN-style string from `hostname`, or `hostname -f` returns `Name or service not known`, the cloud-init log on the node (`/var/log/cloud-init.log` and `/var/lib/cloud/instance/user-data.txt`) is the first thing to inspect, then confirm the controller is actually at v1.0.1 or later.

Storage and data caveats

Rolling replacement destroys each old VM and builds a fresh replacement from the new template. On HCS:

- The system disk (root volume) is rebuilt from the VM image on every replacement.
- Data volumes declared in `HCSMachineTemplate.spec.template.spec.dataVolumes` are not detached and reattached; volumes on the old VM may be removed with it.
- Workload data must live on external persistent storage (HCS EVS CSI or equivalent) to survive the migration.

Complete backup and migration of any node-local state before starting.

Rollback

If the new rollout is unhealthy — new VMs fail to boot, nodes do not become `Ready`, or applications break in unexpected ways — patch each affected resource back to the previous template name. Cluster API treats the reversion as a new spec drift and rolls the new machines back to the previous template:

```
kubectl patch kubeadmcontrolplane <kcp-name> -n cpaas-system --type='merg
e' \
  -p='{"spec":{"machineTemplate":{"infrastructureRef":{"name":"<previous-
cp-template>"}}}}'

kubectl patch machinedeployment <md-name> -n cpaas-system --type='merge'
\
  -p='{"spec":{"template":{"spec":{"infrastructureRef":{"name":"<previous
-worker-template>"}}}}}'
```

Do not delete the previous `HCSMachineTemplate` until the new rollout is healthy. If it has already been deleted, recreate it from version control or backup before rolling back.

Related

- Upgrade Kubernetes on Huawei Cloud Stack — Kubernetes-version upgrades use the same rolling-replacement mechanism.

- Troubleshoot Huawei Cloud Stack Workload Clusters — diagnose the kubeadm-init-stalled symptom that appears when an unfixed controller boots a dotted-hostname node.
- Troubleshoot a Workload Cluster Stuck in Provisioned — provider-agnostic import-controller diagnostic flow.
- Hostname behavior on the node — what `hostname` / `hostname -f` / `/etc/hosts` look like for short versus dotted pool entries on a freshly created cluster.

Manage Nodes on VMware vSphere

This document explains how to manage worker nodes on VMware vSphere after the baseline cluster is running. Node lifecycle operations are managed through `VSphereMachineConfigPool`, `VSphereMachineTemplate`, `KubeadmConfigTemplate`, and `MachineDeployment` resources.

To add another worker node pool with YAML, follow Create the worker objects on the cluster creation page.

> **INFO**
>
> To upgrade the Kubernetes version of the nodes, see Upgrade Kubernetes on VMware vSphere.

TOC

- Prerequisites
- Steps
  - Scale out worker nodes
  - Roll out updated worker node configuration
  - Verify worker node status
- Updating Bootstrap Templates
- Runtime Topology Changes
- Troubleshooting
- Next Steps

Prerequisites

Before you begin, ensure the following conditions are met:

- The workload cluster was created successfully. See Create a Cluster on VMware vSphere.
- The worker machine config pool has enough available slots.
- The control plane is healthy and reachable.
- You know which manifest files currently define the worker nodes.
- You know the rollout and pool rules in Provider Requirements.
- To manage the nodes of the `global` cluster itself, the provider must be installed on `global` first; see Manage Nodes on Alauda OS Clusters.

WARNING

A Day-2 `apply` can be rejected

The rollout and pool rules apply to every edit this page makes:

- Re-applying an archived `MachineDeployment` manifest re-submits the whole object, including an apply whose only intended change is `replicas`. Before you apply it, confirm the file sets `strategy.rollingUpdate.maxSurge: 0`; the examples use `maxUnavailable: 1`. With the default `maxSurge` of `1`, the rollout needs an extra slot that a full pool does not have, and it stalls. `strategy.type: OnDelete` needs no surge setting.
- A `KubeadmControlPlane` must set `spec.rolloutStrategy.rollingUpdate.maxSurge: 0` and `spec.replicas` of `3` or more, or an edit to it is rejected. See Check the manifests against the admission rules.
- A slot whose `status.configStatuses[].state` is `InUse` or `Released` cannot be removed, cannot have its `network.primary.ip` or `.ipv6` changed, and cannot lose a disk from `persistentDisks[]` or change one's `sizeGiB` or `mountPath`. Adding a slot is accepted. Do not add a disk to such a slot; see Allocated slot restrictions.

Match a rejection message against Admission rejections.

Steps

1 Scale out worker nodes

When you add more worker nodes, update the worker machine config pool before you increase the replica count.

Worker scale-out depends on the relationship between `MachineDeployment.spec.replicas` and the available node slots in `VSphereMachineConfigPool.spec.configs[]`. The number of slots can be greater than `replicas`, and idle slots do not affect a running cluster. If `replicas` exceeds the number of available slots, CAPV cannot assign the new workers correctly.

1. Add one or more new node slots to `17-vspheremachineconfigpool-worker.yaml`.
2. Update `replicas` in `30-workers-md-0.yaml`.
3. Apply the updated manifests.

The following example adds a worker slot with the required data disks:

```
- hostname: "<worker_node_name_2>"
  datacenter: "<worker_02_datacenter>"
  network:
    primary:
      networkName: "<nic1_network_name>"
      ip: "<worker_02_nic1_ip>/<nic1_prefix>"
      gateway: "<nic1_gateway>"
      dns:
      - "<nic1_dns_1>"
  persistentDisks:
  - name: var-cpaas
    sizeGiB: <worker_var_cpaas_size_gib>
    mountPath: /var/cpaas
    fsFormat: ext4
  ephemeralDisks:
  - name: var-lib-kubelet
    sizeGiB: <worker_var_lib_kubelet_size_gib>
    mountPath: /var/lib/kubelet
    fsFormat: ext4
  - name: var-lib-containerd
    sizeGiB: <worker_var_lib_containerd_size_gib>
    mountPath: /var/lib/containerd
    fsFormat: ext4
```

The example uses the disk layout of a cluster created on provider `v1.0.17`. A pool created on an earlier provider has no `ephemeralDisks[]` and declares `/var/cpaas`, `/var/lib/containerd`, and `/var/lib/etcd` in `persistentDisks[]`; match the slots that pool already has instead of introducing `ephemeralDisks[]` into it.

To attach an additional raw disk without formatting or mounting, omit `mountPath` and `fsFormat`:

```
  persistentDisks:
  # ...standard persistent disks...
  - name: app-data
    sizeGiB: 50
```

The raw disk is available inside the guest OS at `/dev/disk/by-capv/app-data`. On rolling updates, the same VMDK is re-attached to the replacement VM and the symlink

is recreated. The application remains responsible for the disk contents and runtime mount.

Then increase the worker replicas:

```
replicas: <worker_replicas>
```

Use the following order:

```
kubectl apply -f 17-vspheremachineconfigpool-worker.yaml
kubectl apply -f 30-workers-md-0.yaml
```

The second command replaces the whole `MachineDeployment` spec, not only `replicas`. Confirm the archived file carries `strategy.rollingUpdate.maxSurge: 0` before you apply it.

Note: Keep the same existing `VSphereMachineConfigPool.metadata.name` when scale-out or replacement must preserve the logical pool. Slots and VMDKs are not reused across differently named pool objects. `releaseDelayHours` is the reclaim grace period for an unused `Released` slot, not a delay before same-pool reuse.

2 Roll out updated worker node configuration

When you need to change worker VM specifications, create a new `VSphereMachineTemplate` and update the `MachineDeployment` to reference it. This triggers a rolling update that replaces worker nodes with the new configuration.

Typical changes include:

- VM template name (`spec.template.spec.template`)
- CPU or memory sizing (`numCPUs` , `memoryMiB`)
- System disk sizing (`diskGiB`)
- VM network devices (`network.devices`)

To move nodes onto a new Alauda OS image while the Kubernetes version stays the same, follow OS Image Update Without a Kubernetes Version Change instead. If the

Kubernetes version changes too, see Upgrade Kubernetes on VMware vSphere.

Persistent disks are declared in `VSphereMachineConfigPool` slots rather than in `VSphereMachineTemplate`, so a new template does not change them. A slot that is `InUse` or `Released` rejects changing an existing disk's `sizeGiB` or `mountPath`, so a different geometry for an existing disk needs a new slot, not an edit of the current one. See Allocated slot restrictions.

1. **Export the existing template**

```
kubectl get vspheremachinetemplate <cluster_name>-worker -n <namespace> -o yaml > new-worker-template.yaml
```

2. **Modify the template**

   Edit `new-worker-template.yaml`:

   - Set `metadata.name` to a new unique name (for example, `<cluster_name>-worker-v2`)
   - Update the desired specification fields
   - Remove server-generated fields: `metadata.resourceVersion`, `metadata.uid`, `metadata.generation`, `metadata.creationTimestamp`, `metadata.managedFields`, `metadata.annotations["kubectl.kubernetes.io/last-applied-configuration"]`, and `status`

3. **Apply the new template**

```
kubectl apply -f new-worker-template.yaml
```

4. **Update the MachineDeployment reference**

```
kubectl patch machinedeployment <cluster_name>-md-0 -n <namespace>
\
  --type='merge' -p='{
    "spec": {
      "template": {
        "spec": {
          "infrastructureRef": {
            "name": "<new-template-name>"
          }
        }
      }
    }
  }'
```

If you also need to change bootstrap settings, see Updating Bootstrap Templates below.

5. **Monitor the rolling update**

```
kubectl -n <namespace> get machinedeployment <cluster_name>-md-0 -
w
kubectl -n <namespace> get machine
```

To change the machine specification of control plane nodes, use the same steps and patch `KubeadmControlPlane.spec.machineTemplate.infrastructureRef.name` instead.

TIP

Rolling back a failed update

If the rolling update fails (for example, new VMs fail to boot), revert the `MachineDeployment` reference back to the previous template name. This works only while the Kubernetes minor version is unchanged. The old template still exists and Cluster API will roll back to it.

3 Verify worker node status

> Run the following commands to verify the management-cluster and workload-cluster status:
>
> ```
> kubectl -n <namespace> get machinedeployment,machine,vspheremachine,vspherevm
> kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
> ```
>
> Confirm the following results:
>
> - The target worker replica count is reached.
> - Every new worker node joins the cluster.
> - The nodes eventually become `Ready`.

Updating Bootstrap Templates

To update bootstrap configuration, create a new `KubeadmConfigTemplate` and update the `MachineDeployment` reference.

1. **Export the existing template**

   ```
   kubectl get kubeadmconfigtemplate <cluster_name>-worker-bootstrap -n <namespace> -o yaml > new-bootstrap-template.yaml
   ```

2. **Modify the template**

   Edit `new-bootstrap-template.yaml`:

   - Set `metadata.name` to a new unique name (for example, `<cluster_name>-worker-bootstrap-v2`)
   - Update the desired bootstrap configuration fields
   - For Kubernetes 1.35 or later, add the required kubelet patch settings to `/etc/kubernetes/patches/kubeletconfiguration0+strategic.json`
   - Remove the same server-generated fields listed in Roll out updated worker node configuration step 2

3. **Apply the new template**

```
kubectl apply -f new-bootstrap-template.yaml
```

4. **Update the MachineDeployment reference**

```
kubectl patch machinedeployment <cluster_name>-md-0 -n <namespace> \
  --type='merge' -p='{
    "spec": {
      "template": {
        "spec": {
          "bootstrap": {
            "configRef": {
              "name": "<new-bootstrap-template-name>"
            }
          }
        }
      }
    }
  }'
```

The Cluster API controller triggers a rolling update. Existing machines continue using the old bootstrap configuration until they are replaced.

Runtime Topology Changes

Networking, placement, and disk topology are part of the machine templates and machine config pool. Treat a change to these fields as a rollout of replacement machines, not as an in-place VM edit:

1. Add the required slot definitions to the existing `VSphereMachineConfigPool` before increasing replicas or starting replacement. An allocated slot rejects some changes; see the warning in Prerequisites.
2. Create a new `VSphereMachineTemplate` when its VM, network, placement, or system-disk fields change.
3. Create a new `KubeadmConfigTemplate` when bootstrap files or node configuration change.

4. Update the `MachineDeployment` references only after the new pool capacity and templates exist.
5. Monitor replacement Machines and confirm that every node becomes `Ready` before removing old templates.

For the exact second-NIC, failure-domain, and persistent-disk fields, use the creation-time variants in Create a Cluster on VMware vSphere. Preserve the same cross-resource relationships during Day-2 changes:

- Keep `VSphereMachineConfigPool.spec.configs[].network.additional`, `VSphereMachineTemplate.spec.template.spec.network.devices`, and `VSphereFailureDomain.spec.topology.networks` consistent when NIC topology changes.
- Configure `VSphereCluster.spec.failureDomainSelector` and the CPI `[Labels]` block together when enabling failure domains.
- Do not remove the required `var-cpaas` persistent disk, or the required `var-lib-kubelet` and `var-lib-containerd` ephemeral disks, or the control-plane `var-lib-etcd` ephemeral disk.
- Keep a new slot's disk lists identical to the slots already in that pool. `ephemeralDisks[]` exists from provider `v1.0.17`, but a pool created on an earlier provider declares `/var/cpaas`, `/var/lib/containerd`, and `/var/lib/etcd` in `persistentDisks[]`; do not convert those slots, and do not mix layouts inside one pool.
- Do not reshape or add disks to an allocated slot. A slot that is `InUse` or `Released` rejects removing a persistent disk or changing its `sizeGiB` or `mountPath`. Resizing means planning a new slot.
- Do not treat `machineNamingStrategy`, `wipeFilesystem`, or `releaseDelayHours` as a switch that moves disks between pools or accelerates datastore cleanup.

When you expand from one NIC to two NICs, update all of the following fields together:

1. Append the second NIC entry to `VSphereMachineConfigPool.spec.configs[].network.additional`; keep `network.primary` unchanged.
2. Add the second device to `VSphereMachineTemplate.spec.template.spec.network.devices` in the new template.

3. Add the second network to `VSphereFailureDomain.spec.topology.networks` when failure domains are enabled.

When you revert from two NICs to one NIC, remove the corresponding second entry from all three fields. Leave `network.additional` empty or remove the key entirely, and roll out a new `VSphereMachineTemplate`; do not edit existing VMs directly.

Troubleshooting

Use the following checks first when worker node management fails:

- Check `VSphereMachine` conditions for `MachineConfigPoolReady`. If `False`, the reason indicates why slot allocation failed:
  - `PoolBoundToOtherConsumer`: the pool is already bound to a different `KubeadmControlPlane` or `MachineDeployment`.
  - `NoAvailableSlots`: no slots match the required datacenter or failure domain.
- Verify that the worker machine config pool still has free slots.
- Verify that the worker IP addresses, gateway, and DNS settings are correct.
- Verify that the worker VM template still matches the required Kubernetes version and guest-tools requirements.
- Check `VSphereVM.status.addresses` when a node is waiting for IP allocation.

Next Steps

For infrastructure preparation and persistent-disk diagnostics, see VMware vSphere Infrastructure Preparation. For cluster creation manifest fields, see Create a Cluster on VMware vSphere.

Non-Alauda OS Nodes

Add and Remove Nodes

Platform administrators can add nodes to a
run a non-Alauda OS.

Manage Nodes

Update node labels and taints, stop and re

Plan an In-P

Plan and verify
platform creates

Add and Remove Nodes on Non-Alauda OS Clusters

When a cluster needs to scale up or when abnormal nodes on the cluster need to be replaced with new nodes, you can add control plane nodes and worker nodes to existing workload clusters whose nodes run a non-Alauda OS (cluster type **On-Premises**). You can also delete nodes from these clusters, for example, to remove failed nodes.

TOC

Constraints and Limitations
Prerequisites
Procedure
Follow-up Operations
View Execution Progress
Re-add Failed Nodes
Delete Nodes
Constraints and Limitations
Procedure

Constraints and Limitations

- Prepare nodes before adding them to the cluster. Use the Node Availability Check Reference to verify the nodes. Cluster deployment can fail when required conditions are not met.
- The hardware architecture of nodes to be added must be consistent with the cluster's hardware architecture.
- To avoid unpredictable errors, the operating system type of nodes to be added should be consistent with other nodes in the cluster.
- SSH ports and authentication information for nodes added in the same **Add Node** dialog must be unified.
- **Cluster planning guideline**: A cluster must have at least 1 control plane node. Setting exactly 2 control plane nodes is not supported. With 3 or more control plane nodes, the cluster becomes a high-availability cluster (for high-availability clusters, it is recommended to use an odd number of nodes, preferably 3 or 5). **Note**: This requirement applies only when adding or changing control plane capacity; you can safely add worker/worker nodes without being forced to add control plane nodes.
- A node can be part of only one cluster. Nodes that already belong to another cluster cannot be added.

Prerequisites

- When the global cluster cannot directly access nodes to be added to the cluster through SSH service and needs to access through a proxy, prepare the proxy service in advance. Currently, only SOCKS5 proxy is supported.

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** of type **On-Premises** where you want to add nodes.
3. Under the **Nodes** tab, click **Add Node**.
4. Configure the relevant parameters. For parameter details, see Node Configuration Parameters.

5. Click **Add** to perform availability check on the nodes.
   After the check passes, node addition begins, and the nodes are in **Adding** state.

Follow-up Operations

View Execution Progress

On the node list page, you can view the list information of added nodes. For nodes in **Adding** state, you can view the execution progress.

Procedure

1. Click **View Execution Progress** on the right side of nodes in **Adding** state.
2. In the pop-up execution progress dialog, you can view the node execution progress (`status.conditions`).
   Tip: When a certain type is executing or has a failed state with a reason, you can view detailed information about the reason (`status.conditions.reason`) by hovering the cursor over the corresponding reason (displayed in blue text).

Re-add Failed Nodes

After adding nodes, if some nodes fail to be added, a prompt will appear above the node list. Click the **Re-add** button in the prompt box to re-add the failed nodes.

Delete Nodes

You can delete nodes from clusters of type **On-Premises**, for example, to remove failed nodes.

Constraints and Limitations

- Nodes in imported clusters are not supported for deletion.

- When there is only one control plane node in the cluster, deleting this control plane node is not supported.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** of type **On-Premises** where the node to be deleted is located.
3. Under the **Nodes** tab, click **Delete** on the right side of the node to be deleted.
   Tip: If you need to clean up resources under the node after deleting a Linux node, click **Download Cleanup Script** at the bottom of the dialog to download the cleanup script to local. After the node is successfully deleted, log in to the node and execute the cleanup script.
4. Enter the node name, then click **Delete**.

Manage Nodes

To create infra nodes or custom role nodes, see Infra Nodes.

TOC

- Update Node Labels
  - Procedure
- Stop/Resume Node Scheduling
  - Procedure
- Evict Pods
  - Procedure
- Set Taints
  - Procedure
- Label and Taint Management
  - Constraints and Limitations
  - Procedure
- Enable/Disable Virtualization Switch

Update Node Labels

Labels ↗ are key-value pairs attached to nodes that can define node attributes. After setting labels for nodes, you can easily filter or select nodes by labels. For example: directing Pods to

be scheduled to specific nodes.

Supports updating node labels for nodes in normal state, adding or removing custom node labels.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** where the node with labels to be updated is located.
3. Under the **Nodes** tab, click **Update Node Labels** on the right side of the node with labels to be updated.
4. Add, modify, or delete node labels.
5. Click **OK**.
   After successfully updating node labels, the number of node labels changes. You can view all label information of the node in the **Node Labels** item in the **Node** information bar.

Stop/Resume Node Scheduling

By setting the scheduling state of nodes, you can control whether newly created Pods in the cluster are allowed to be scheduled to the node.

- **Stop Scheduling**: Newly created Pods are not allowed to be scheduled to the node, but existing Pods running on the node are not affected.
- **Resume Scheduling**: Newly created Pods are allowed to be scheduled to the node.

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** where the node to stop/resume scheduling is located.
3. Under the **Nodes** tab, click **Stop Scheduling/Resume Scheduling** on the right side of the node to set scheduling state.
4. Click **OK**.

Evict Pods

Evict all Pods except those managed by DaemonSet (daemon set) from nodes in normal state to other nodes in the cluster, and set the node to unschedulable state.

Note: Eviction removes data stored locally in Pods. Confirm the data impact before evicting Pods.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** where the node to evict Pods is located.
3. Under the **Nodes** tab, click the ***node name*** to evict Pods.
4. In the upper right corner, click **Actions** > **Evict Pods**.
5. Review the information of Pods to be evicted, then click **Evict**.

Set Taints

Set taint information for nodes in normal state.

Taints are a property of nodes that allow nodes to refuse to run certain types of Pods or even evict Pods. Taints work together with tolerations on Pods to prevent Pods from being assigned to inappropriate nodes. One or more taints can be applied to each node, and Pods that cannot tolerate these taints will not be accepted by the node.

For example: For a node where we find its memory utilization has reached 91%, it is not recommended to continue scheduling new Pods to this node. We can set a taint for it. After setting the taint, Kubernetes will not schedule Pods to this node.

For Kubernetes taint and toleration behavior, see Taints and Tolerations↗.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.

2. Click the ***cluster name*** where the node to set taints is located.
3. Under the **Nodes** tab, click **Set Taints** on the right side of the node to set taints.
4. Set the key, value, and effect of taints. Multiple taints can be added to a node.
   Taint attributes consist of `key=value [effect]`.
   `key=value` is used to match Pod tolerations. The taint indicates that the node has been contaminated by `key=value`, and Pod scheduling is not allowed or should avoid scheduling to this node, unless the Pod can tolerate (Tolerations) the `key=value` taint.
   effect is the effect of the taint, with the following three options:

   - **NoSchedule**: Indicates scheduling is not allowed, and already scheduled resources are not affected.
   - **PreferNoSchedule**: Indicates try not to schedule.
   - **NoExecute**: Indicates scheduling is not allowed, and already scheduled resources will be deleted after `tolerationSeconds`.
5. Click **OK**.

Label and Taint Management

The platform supports batch setting of labels and taints for nodes.

Constraints and Limitations

- Before setting device labels, you need to deploy device plugins on the cluster first, such as NVIDIA GPU MPS device plugin, NVIDIA GPU device plugin, GPU Manager device plugin, etc.
  Tip: Device labels are actually node labels. For your convenience, the platform categorizes node labels that device plugins depend on as device labels for quick configuration.

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** where you want to manage labels and taints.

3. Under the **Nodes** tab, multi-select the nodes you want to manage, and click the **Label and Taint Management** button.
   Tip: You can enter the node labels you care about in the search box on the node list page to quickly filter out the list of nodes you want to manage labels and taints for.
4. In **Batch Operations**, add and fill in the operations you want to perform, then click OK to submit the batch operations to the cluster.

   - **Node Labels**: You can **add/update** specified labels for selected nodes, or **delete** specified labels. When selecting delete, the platform will filter out all label lists on the selected nodes. When the value is set to **Any**, it represents deleting labels on all nodes containing the specified label key.
   - **Taints**: You can **add/update** specified taints for selected nodes, or **delete** specified taints. When selecting delete, the platform will filter out all taint lists on the selected nodes. When the value is set to **Any**, it represents deleting taints on all nodes containing the specified taint key.
   - **Device Labels**: You can set the devices you want to use for selected nodes, where the device list comes from device plugins you have deployed in this cluster.

Enable/Disable Virtualization Switch

When nodes in an on-premises cluster are physical machines, you can control whether Kubernetes is allowed to schedule virtual machines (VMI, VirtualMachineInstance) to the node by enabling/disabling the node virtualization switch.

When the switch is enabled, newly created virtual machines are allowed to be scheduled to the physical machine node; when the switch is disabled, newly created virtual machines are prohibited from being scheduled to the physical machine node, but this does not affect virtual machines already running on the node.

Tip: For related operations and precautions, see Prepare Virtualization Environment.

Plan an In-Place OS Upgrade for Nodes

Cluster administrators can use this checklist to plan an in-place operating system upgrade for nodes in workload clusters that ACP creates on a non-Alauda OS (cluster type **On-Premises**). Complete the ACP checks before and after the operating system changes.

> **Scope and support boundary**
>
> Use this checklist together with the operating system provider's supported upgrade procedure during a maintenance window. Confirm the target operating system and kernel against the ACP support matrix before you upgrade any node. The checklist does not guarantee that every operating system upgrade path is supported.

TOC

- Scope and Limitations
- Before You Upgrade
  - Step 1: Confirm the target operating system and kernel
  - Step 2: Check for conflicting packages
  - Step 3: Record the current node baseline
  - Step 4: Verify cluster capacity and drain the node
- Additional Checks for Control Plane Nodes
- Perform the Operating System Upgrade
- After You Upgrade
  - Step 1: Confirm the operating system and kernel

Step 2: Verify base node configuration

Step 3: Verify runtime and node component versions

Step 4: Verify time synchronization and DNS

Step 5: Restart kubelet and verify node recovery

Step 6: Validate workload recovery

Known High-Risk Scenarios

Operation Checklist

Scope and Limitations

Use this procedure only when all of the following conditions are met:

- The cluster is a workload cluster that ACP creates on a non-Alauda OS.
- You can log in to the target nodes and manage the node operating system.
- You have platform administrator permissions, `kubectl` administrator permissions, and SSH or console access to each target node.
- You can drain workload Pods from the target node before the operating system upgrade.

Do not use this procedure for the following cluster types:

- Clusters whose nodes run Alauda OS. Apply operating system changes by replacing nodes with new images.
- Managed cloud Kubernetes clusters where the cloud provider manages the node or control plane operating system.
- Imported clusters where you cannot log in to nodes or control plane nodes.

Before You Upgrade

Complete the following checks before changing the operating system on any node.

1 Step 1: Confirm the target operating system and kernel

Confirm that the target operating system version, kernel version, kernel source, and CPU architecture are within the supported range. For the current support matrix and known restrictions, see Supported OS and Kernel Versions.

Apply these rules before the maintenance window:

- The operating system major and minor versions must match the supported matrix.
- The core kernel version must match the supported matrix. Only the build suffix can differ.
- The kernel must be the official kernel shipped by the operating system vendor.
- Ubuntu HWE kernels and third-party or custom-compiled kernels are not supported.
- If the target version is not in the supported matrix, contact technical support for compatibility confirmation before the upgrade.

2 Step 2: Check for conflicting packages

During an in-place operating system upgrade, the operating system package manager might install or overwrite container runtime, Kubernetes, or container network binaries. Before the upgrade, check and resolve packages that conflict with ACP components. For the package lists and commands, see Remove Conflicting Packages.

If conflicting packages are found, prepare an application migration plan and back up the affected data before uninstalling them.

3 Step 3: Record the current node baseline

Record the node state before the operating system upgrade. Use the records for comparison after the node is upgraded.

Run the `Base node configuration check` block from Step 2: Verify base node configuration on the node now, and keep its output. It captures the SELinux mode, the AppArmor and host firewall state, and the firewall rules in use. The post-upgrade checks compare against exactly this output, so a node without it cannot be verified.

Record the runtime and node component versions as well:

```
containerd --version
runc --version
crictl --version
kubelet --version
```

Also record critical node configuration files that your operating system upgrade process might update, such as `/etc/resolv.conf`, `/etc/fstab`, systemd configuration files, and container runtime configuration files.

4 Step 4: Verify cluster capacity and drain the node

Confirm that the remaining nodes have enough capacity to run the Pods evicted from the target node. Then drain the node before the operating system upgrade.

You can use the ACP console to evict Pods from the node. For the console operation, see Manage Nodes.

If you use `kubectl`, confirm the command options with your operations team before running it. For example:

```
kubectl cordon <node-name>
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

Pods managed by DaemonSets are not evicted by the drain operation. Workloads that use local storage might lose local data after eviction. Confirm the workload impact before you proceed.

Additional Checks for Control Plane Nodes

Control plane nodes run components such as `kube-apiserver`, `etcd`, `kube-scheduler`, and `kube-controller-manager`. Upgrade control plane nodes one at a time, and verify cluster health after each node is upgraded.

Before upgrading a control plane node:

- Back up etcd data. For the supported backup mechanism and restore considerations, see etcd Backup and Restore.
- Confirm that the etcd cluster is healthy and that quorum can be maintained while one control plane node is unavailable.
- Confirm that the cluster has at least three control plane nodes when you are performing rolling maintenance on control plane nodes.
- If possible, validate the procedure on worker nodes first, and then proceed with control plane nodes.

You can use the following commands as references when checking control plane health:

```
kubectl get nodes
kubectl get pods -n kube-system | grep -E "etcd|apiserver|scheduler|contr
oller"
```

If you use `etcdctl` on a control plane node, use the certificate paths from your environment:

```
ETCDCTL_API=3 etcdctl member list \
  --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 \
  --write-out=table
```

Perform the Operating System Upgrade

Follow the operating system vendor's supported upgrade procedure for the target version. The exact package commands, repository configuration, and reboot requirements are determined by the operating system vendor and your organization's operating system maintenance policy.

During the operating system upgrade:

- Do not intentionally install container runtime, Kubernetes, or container network packages that conflict with ACP components.

- Preserve node network configuration, DNS configuration, time synchronization configuration, and systemd service configuration unless the vendor procedure requires a change.
- Reboot the node when the vendor procedure requires it.
- Keep the node unschedulable until all post-upgrade checks pass.

After You Upgrade

Run the following checks on each node after the operating system upgrade and reboot are complete.

1 Step 1: Confirm the operating system and kernel

Verify that the node is running the expected operating system and kernel versions.

```
cat /etc/os-release
uname -r
```

Compare the output with the supported matrix and your approved maintenance plan.

2 Step 2: Verify base node configuration

Verify that the operating system upgrade did not revert required node settings.

```
echo "=== Base node configuration check ==="

# SELinux must be disabled. Record the value so that you can confirm
the
# operating system upgrade did not turn it back on.
if command -v getenforce >/dev/null; then
  echo "SELinux: $(getenforce)"
else
  echo "SELinux: getenforce not available on this system"
fi

# AppArmor must be disabled. A missing unit means the system has no A
ppArmor at
# all, which is not the same as AppArmor being present but turned of
f.
if systemctl list-unit-files --no-legend apparmor.service 2>/dev/null
| grep -q .; then
  echo "AppArmor: enabled=$(systemctl is-enabled apparmor.service 2>&
1) active=$(systemctl is-active apparmor.service 2>&1)"
else
  echo "AppArmor: unit not present on this system"
fi

# firewalld. Same distinction as AppArmor.
if systemctl list-unit-files --no-legend firewalld.service 2>/dev/nul
l | grep -q .; then
  echo "firewalld: enabled=$(systemctl is-enabled firewalld.service 2
>&1) active=$(systemctl is-active firewalld.service 2>&1)"
  firewall-cmd --list-all 2>/dev/null || true
else
  echo "firewalld: unit not present on this system"
fi

# ufw. The unit reports active (exited) once it has ever run, so the
unit state
# does not show whether the ruleset is enabled. Only ufw status does.
if command -v ufw >/dev/null; then
  ufw status verbose
else
  echo "ufw: not installed"
fi

# Swap must be disabled.
```

```
swapon --show
free -h | grep Swap

# /tmp must not be mounted with noexec.
mount | grep " /tmp "

# DefaultTasksMax must be infinity.
systemctl show --property=DefaultTasksMax
```

If `swap` is enabled after the operating system upgrade, disable it and remove the swap entry from `/etc/fstab` according to your operating system policy.

SELinux and AppArmor must be disabled. If the operating system upgrade re-enabled either of them, disable it again according to your operating system policy. See Why SELinux and AppArmor Must Be Disabled.

The host firewall must return to the state that the cluster was installed with. If the cluster was installed with the host firewall disabled and the operating system upgrade re-enabled it, disable it again. If the cluster runs with the host firewall enabled, confirm that it is still enabled and that its rules are complete after the upgrade. For the required configuration and the port list, see Hardened Node Security Configuration and Communication Matrix.

3 Step 3: Verify runtime and node component versions

Compare the runtime and node component versions with the pre-upgrade records.

```
containerd --version
runc --version
crictl --version
kubelet --version
```

Then verify that `containerd` is running:

```
systemctl status containerd
```

If any binary was overwritten by an operating system package, stop the maintenance and contact technical support before you continue with other nodes.

4 Step 4: Verify time synchronization and DNS

Verify that time synchronization and DNS configuration were not changed by the operating system upgrade.

```
date
timedatectl
cat /etc/resolv.conf
```

The time skew between nodes must be no more than 10 seconds. If the skew is larger than 10 seconds, synchronize time before restarting workloads on the node.

5 Step 5: Restart kubelet and verify node recovery

Restart `kubelet` after the operating system upgrade is complete and the base node checks pass.

```
systemctl daemon-reload
systemctl restart kubelet
systemctl status kubelet
```

Wait for the node to return to the `Ready` state.

```
kubectl get node <node-name> --watch
```

When the node is healthy, resume scheduling.

```
kubectl uncordon <node-name>
```

You can also resume scheduling from the ACP console. For the console operation, see Manage Nodes.

6 Step 6: Validate workload recovery

Verify the node status and workload placement after scheduling is resumed.

```
kubectl get nodes
kubectl get pods -A -o wide | grep <node-name>
```

Then validate the business services that were affected by the node drain. Proceed to the next node only after the current node and related services are healthy.

Known High-Risk Scenarios

Symptom	Possible Cause	How to Check	Recommended Action
`containerd` fails to start	The operating system upgrade overwrote the ACP runtime binary	Compare `containerd --version` with the pre-upgrade record	Stop the maintenance and contact technical support
The node stays `NotReady`	`kubelet`, container runtime, CNI, DNS, or node network configuration changed	Check `systemctl status kubelet`, `systemctl status containerd`, and node events	Restore the changed configuration or contact technical support
Cluster components report TLS errors	Node time drifted during or after the upgrade	Check `timedatectl` and compare `date` across nodes	Synchronize time before continuing
DNS resolution fails	`/etc/resolv.conf` was overwritten	Check `cat /etc/resolv.conf`	Restore the approved DNS configuration
`kubelet` fails with security policy errors	SELinux or AppArmor was re-enabled by the operating system upgrade	Run the `Base node configuration check` block and compare it with the pre-upgrade record	Disable the mechanism again according to your operating system policy, then reboot the node if the operating system requires it. Contact

Symptom	Possible Cause	How to Check	Recommended Action
			technical support if the errors persist
Workloads cannot be scheduled after uncordon	The node is still unhealthy, tainted, or resource-constrained	Check `kubectl describe node <node-name>`	Resolve node conditions before continuing

Operation Checklist

Use this checklist during the maintenance window and keep the completed record after the upgrade.

Phase	Item	Owner	Status
Pre-upgrade	Confirm that the target operating system and kernel are in the support matrix		
Pre-upgrade	Confirm that the kernel is the official vendor kernel		
Pre-upgrade	Check and resolve conflicting packages		
Pre-upgrade	Record `containerd`, `runc`, `crictl`, and `kubelet` versions		
Pre-upgrade	Record DNS, time synchronization, `/etc/fstab`, and runtime configuration		
Pre-upgrade	Record whether SELinux, AppArmor, and the host firewall are enabled, and record the firewall rules in use		

Phase	Item	Owner	Status
Pre-upgrade	Confirm that the cluster has enough capacity for drained workloads		
Pre-upgrade	Drain the target node and confirm workload impact		
Control plane only	Back up etcd data		
Control plane only	Confirm etcd health and quorum		
Upgrade	Run the operating system vendor's supported upgrade procedure		
Upgrade	Reboot the node if required by the vendor procedure		
Post-upgrade	Confirm operating system and kernel versions		
Post-upgrade	Confirm SELinux and AppArmor are disabled		
Post-upgrade	Confirm swap is disabled		
Post-upgrade	Confirm that the host firewall matches the cluster installation configuration		
Post-upgrade	Confirm `/tmp` is not mounted with `noexec`		
Post-upgrade	Confirm `DefaultTasksMax` is `infinity`		
Post-upgrade	Confirm runtime and node component versions were not overwritten		
Post-upgrade	Confirm `containerd` and `kubelet` are healthy		

Phase	Item	Owner	Status
Post-upgrade	Confirm time skew between nodes is no more than 10 seconds		
Post-upgrade	Confirm DNS configuration is correct		
Post-upgrade	Confirm the node is `Ready`		
Post-upgrade	Resume scheduling on the node		
Post-upgrade	Confirm workloads and business services are healthy		