Clusters

Overview

Overview

Create on Alauda OS

Prepare | Bare Metal | Huawei DCS

Huawei Cloud Stack | VMware vSphere | Troublesho

Create on Non-Alauda OS

Create on Non-Alauda OS

Managed Clusters

Managed Clusters | Import Third-Party Clusters | Register Cl

Public Cloud Cluster Initialization | How to

Public cloud cluster initialization.

How to

Access a Cluster with KubeCor | Add External Address for Built- | Updating P

Overview

Start here to choose whether to create a platform-managed cluster or onboard an existing Kubernetes environment. For the platform-level relationship between cluster responsibility, control-plane topology, and onboarding boundaries, see Cluster Management Models.

TOC

- Choose A Cluster Path
- Platform-Managed Cluster Creation
- Installer-Provisioned Infrastructure
- User-Provisioned Infrastructure
- Hosted Control Plane
- Third-Party Cluster Onboarding
- Version Compatibility
  - Upgrade Prerequisite Behavior

Choose A Cluster Path

If you need to	Start with	Key boundary
Create a cluster whose nodes run Alauda OS, provisioned through a supported infrastructure provider.	Create on Alauda OS	Platform-owned lifecycle applies only within the supported provider and workflow scope.
Create a cluster on machines that you prepare and maintain, running a supported non-Alauda OS.	Create on Non-Alauda OS	The platform manages Kubernetes after node preparation; node OS lifecycle remains user-owned.
Onboard an existing Kubernetes environment.	Managed Clusters	The external owner, distribution, or provider usually owns Kubernetes, node, and infrastructure lifecycle.
Choose between direct platform access and reverse-connect onboarding.	Import Third-Party Clusters and Register Cluster	Import and register are onboarding methods, not separate day-2 capability models.

Platform-Managed Cluster Creation

For clusters that ACP creates and lifecycle-manages, choose the creation path by the operating system of the nodes.

Node operating system	Use when	Continue with
Alauda OS	The platform should manage node operating systems through Alauda OS, with nodes provisioned through a supported infrastructure provider.	Create on Alauda OS
Non-Alauda OS	You prepare physical or virtual machines that run a supported operating system, and the platform installs and manages Kubernetes on those nodes.	Create on Non-Alauda OS

The creation path follows the node operating system rather than the infrastructure responsibility model described below. For example, on Bare Metal you supply the physical

servers, but the nodes run Alauda OS, so Bare Metal clusters use the Alauda OS path.

Installer-Provisioned Infrastructure

In Installer-Provisioned Infrastructure (IPI), the platform provisions machines and manages node operating systems through Alauda OS. This model is used when the platform owns the supported cluster provisioning, scaling, and upgrade lifecycle for the selected provider integration.

Currently, the platform supports Alauda OS for immutable node management. For Alauda OS cluster concepts and provider-specific workflows, see Create on Alauda OS.

User-Provisioned Infrastructure

In User-Provisioned Infrastructure, users prepare physical or virtual machines before cluster creation. The platform installs and manages Kubernetes on those nodes, while node operating system management, including provisioning, patching, and replacement, remains under the user's control.

This model is suitable when organizations already have established infrastructure or operating system management procedures. For the cluster creation workflow, see Create on Non-Alauda OS.

To create and manage User-Provisioned Infrastructure clusters through APIs, see User-Provisioned Infrastructure Provider APIs.

Hosted Control Plane

Hosted Control Plane is a control-plane topology. Each hosted cluster has its own control plane, and multiple hosted control planes run as workloads on a management cluster.

In ACP, HCP is implemented through Kamaji (`TenantControlPlane`).

Third-Party Cluster Onboarding

Third-party clusters are existing Kubernetes environments provided outside Alauda. They can include standard Kubernetes environments, external Kubernetes distributions, or public cloud Kubernetes services. ACP can provide centralized governance and operations within documented prerequisites and caveats, but onboarding does not make ACP the owner of the external cluster's Kubernetes lifecycle, node lifecycle, or provider infrastructure lifecycle.

Third-party clusters are onboarded through import or register workflows:

Onboarding method	Connection model	Use when
Import cluster	The `global` cluster connects to the target cluster API server with supplied address, CA, and credentials.	The platform can reach the target cluster API server and the operator can provide the required cluster information.
Register cluster	A reverse proxy service in the target cluster initiates registration and establishes a tunnel to the platform.	The target cluster should initiate the connection, or direct platform access to the target API server is restricted.

Provider-specific caveats can apply to certificates, audit data, control-plane metrics, ingress, storage, node operations, connectivity, and Extension compatibility. Use the import and register workflow pages for detailed prerequisites.

For onboarding workflows, see Third-Party Cluster Onboarding, Import Third-Party Clusters, and Register Cluster.

Version Compatibility

Use the Kubernetes Support Matrix to choose platform-managed cluster creation targets, verify workload-cluster upgrade prerequisites, and validate third-party cluster onboarding versions. These are separate ranges for separate decisions.

Upgrade Prerequisite Behavior

- Before you upgrade the `global` cluster, every workload cluster must run a Kubernetes version within the Compatible Versions of the target release.
- The third-party onboarding range is independent from the Compatible Versions used for this upgrade prerequisite.

Create a Cluster on Alauda OS

On this path, ACP creates workload clusters whose nodes run Alauda OS, an immutable operating system. Unlike clusters whose nodes run a non-Alauda OS, all node configurations are baked into images and remain unchanged after deployment. Cluster upgrades and configuration changes are applied by replacing nodes with new images, ensuring consistency, reliability, and simplified operations throughout the cluster lifecycle.

Clusters on this path are created and managed declaratively through Cluster API. An infrastructure provider for each supported IaaS platform creates the nodes from Alauda OS images. On Bare Metal, you supply the physical servers; they boot an installation ISO built by the provider and install Alauda OS.

This path follows these principles:

- **Declarative Configuration**: All infrastructure is defined as code
- **Version Control**: Infrastructure changes are tracked and versioned
- **Reproducible Deployments**: Consistent environments across different deployments
- **Stage-Aware Recovery**: Reproducible machine replacement for safe template recovery, with backup-based recovery when a Kubernetes minor upgrade has changed control-plane state

TOC

Architecture

Bootstrap Provider (Kubeadm)

Control Plane Provider (Kubeadm)

Infrastructure Provider
Supported Providers
Control Plane Endpoint
Web UI Support (Fleet Essentials)
Install the Providers on the global Cluster
Cluster Creation Workflow
Related Tasks
Prepare
Bare Metal
Huawei DCS
Huawei Cloud Stack
VMware vSphere
Troubleshooting

Architecture

Cluster lifecycle management on Alauda OS is built on the Cluster API (CAPI) framework, which provides a declarative, Kubernetes-native approach:

- **Declarative API**: Kubernetes-native resources for cluster management
- **Provider Interface**: Standardized interfaces for different infrastructure providers
- **Lifecycle Management**: Automated cluster provisioning, scaling, and upgrades
- **GitOps Integration**: Native support for GitOps workflows

Three Cluster API providers work together, delivered as two packages: the **Alauda Container Platform Cluster Bootstrap Provider** handles Kubernetes bootstrap and control plane management, and an infrastructure provider manages the IaaS platform.

Bootstrap Provider (Kubeadm)

The Cluster API bootstrap provider for kubeadm handles the initialization and configuration of Kubernetes nodes:

- **Node Initialization**: Automates the process of turning bare machines into Kubernetes nodes
- **Configuration Management**: Manages kubelet configuration, certificates, and bootstrap tokens
- **Join Process**: Handles the process of joining worker nodes to the cluster
- **Certificate Management**: Automates certificate generation and rotation

Control Plane Provider (Kubeadm)

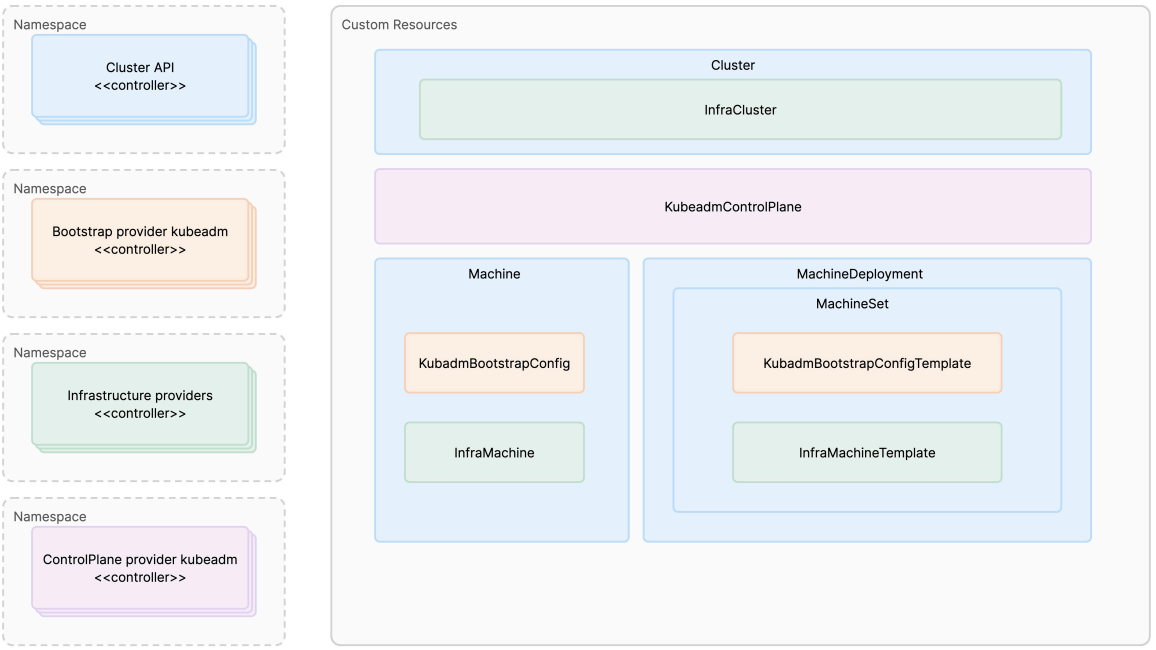
The Cluster API control plane provider for kubeadm manages the Kubernetes control plane:

- **Control Plane Deployment**: Deploys and manages the Kubernetes API server, etcd, and control plane components
- **High Availability**: Supports multi-node control plane deployments with load balancing
- **Upgrade Management**: Handles rolling upgrades of control plane components
- **Health Monitoring**: Monitors control plane health and performs recovery operations

Infrastructure Provider

The infrastructure provider for the target IaaS platform, such as Huawei DCS, manages the underlying infrastructure:

- **Compute Provisioning**: Creates and manages virtual machines or instances on the target platform
- **Network Configuration**: Sets up networking infrastructure
- **Resource Management**: Manages compute, storage, and network resources
- **Provider Integration**: Integrates with the provider's APIs and services



Supported Providers

Each infrastructure provider integrates with one IaaS platform to deliver cluster lifecycle management.

Provider	Description
Bare Metal	Bare-metal servers without virtualization
Huawei DCS	Huawei Datacenter Virtualization Solution
Huawei Cloud Stack	Huawei Cloud Stack
VMware vSphere	VMware vSphere virtualization platform

Control Plane Endpoint

Each cluster's Kubernetes API server (`kube-apiserver`, port `6443`) is reached through a control plane endpoint — a virtual IP (VIP) or load balancer address that fronts the control plane nodes. How that endpoint is provided differs by provider, which determines what you must prepare before cluster creation.

Provider	Control plane endpoint	Mechanism
Bare Metal	The provider creates it (or you provide it)	`BaremetalCluster.spec.controlPlaneLoadBalancer.type=I` deploys the `alive` chart (`keepalived` + IPVS + `kube-lock` arbitration) onto the control-plane nodes; `type=External` dele LB to an existing external appliance
Huawei DCS	You provide it, or the provider manages a Self-built VIP	An external load balancer, a provider-managed `type: interna` built VIP, or a single control plane node reached directly
Huawei Cloud Stack	The provider creates it	An HCS Elastic Load Balance (ELB) instance created automatic during cluster reconciliation
VMware vSphere	The provider creates it (or you provide it)	`VSphereCluster.spec.controlPlaneLoadBalancer.type=int` deploys Alive (`keepalived` + IPVS) onto the control-plane noc `type=external`, and an absent field, delegate the LB to an ex external appliance. The mode is fixed at cluster creation

Huawei Cloud Stack is the only VM provider that creates a platform load balancer object for you. An external load balancer — including its VIP allocation and backend (real-server) maintenance — is prepared and owned outside the cluster. On Bare Metal, Huawei DCS, and VMware vSphere, the provider can instead deploy an in-cluster VIP component (`alive`). For the requirements of each mode, see Plan the Control Plane Endpoint.

Web UI Support (Fleet Essentials)

Fleet Essentials is a cluster plugin that provides a shared web UI framework for cluster management. Infrastructure providers can extend it through UI extension anchors to add platform-specific pages and workflows.

When installed, Fleet Essentials enables:

- **Unified Cluster Management**: Single interface for managing clusters across all platforms

- **Infrastructure Resource Management**: Configure cloud credentials, IP pools, and machine templates
- **Cluster Lifecycle Operations**: Create, view, upgrade, and destroy clusters through guided workflows
- **Node Pool Management**: Declarative node pool management with scaling and upgrade capabilities

Providers that implement these Fleet Essentials extension anchors can add their own cluster creation workflows and management pages:

- **Bare Metal Provider**: Provides YAML-only cluster lifecycle management on physical servers. There is no Fleet Essentials UI for bare-metal clusters; every workflow is driven by `kubectl apply` against `BaremetalCluster`, `BaremetalMachineTemplate`, `MachineInventoryPool`, and the upstream CAPI resources
- **DCS Provider**: Adds DCS-specific UI pages and workflows on Huawei DCS. The `DCSIpHostnamePool.spec.pool[].persistentDisk` configuration for pool-managed persistent disks is YAML-only
- **HCS Provider**: Supports web UI-based cluster management on Huawei Cloud Stack when Fleet Essentials `1.0.2` or later is installed, and YAML-based cluster lifecycle management, including pool-managed persistent disks through `HCSMachineConfigPool.spec.configs[].persistentDisks[]`
- **vSphere Provider**: Provides YAML-based cluster lifecycle management on VMware vSphere. There is no Fleet Essentials UI extension for vSphere cluster creation and management

Install the Providers on the global Cluster

After the `global` cluster is installed, and before you create a workload cluster on Bare Metal, Huawei DCS, Huawei Cloud Stack, or VMware vSphere, upload and install the infrastructure provider for that IaaS platform, together with the Cluster Bootstrap Provider, on the `global` cluster. This applies regardless of which IaaS platform the `global` cluster itself runs on.

To install a provider, download its package from the Customer Portal, where the providers are listed under **Common ACP Extensions**, upload it with `violet push`, and install it as a cluster plugin. See Install the Infrastructure Provider.

Upgrading Kubernetes on an Alauda OS `global` cluster also requires the provider for that IaaS platform to be uploaded and installed on the `global` cluster first.

Cluster Creation Workflow

Create a workload cluster on Alauda OS in the following order. Prepare describes the first three steps in the same order.

1. **Install the Provider**: Install the provider for your IaaS platform, and the Cluster Bootstrap Provider, on the `global` cluster.
2. **Infrastructure Resources**: Configure the infrastructure resources that the provider requires, such as cloud credentials, IP pools, and machine templates. Bare Metal has no separate infrastructure resource page; its host preparation is part of the provider installation and cluster creation pages.
3. **Plan the Cluster**: Read Prepare for the decisions that apply to every provider, such as the control plane endpoint mode and the image registry of the cluster. Also check the following before you create the cluster:
   - **Node Labels and Taints**: If dedicated node pools (monitoring, logging, ingress) are planned, read how to declare role labels and taints on the pool in Infra Nodes before creating the pools.
   - **CPU Baseline**: For x86_64 cluster nodes that use ACP-provided Alauda OS images, the underlying CPUs must support the `x86-64-v2` ISA baseline. For details, see OS Support Matrix.
4. **Create a Cluster**: Create the cluster, including its first worker nodes.

Each provider has its own pages for these steps:

Provider	Overview	Install the Provider	Infrastructure Resources	Create a Cluster
Bare Metal	Bare Metal Provider	Install the Bare Metal Provider	—	Create a Cluster on Bare Metal

Provider	Overview	Install the Provider	Infrastructure Resources	Create a Cluster
Huawei DCS	Huawei DCS Provider	Install the Huawei DCS Provider	Infrastructure Resources for Huawei DCS	Create a Cluster on Huawei DCS
Huawei Cloud Stack	Huawei Cloud Stack Provider	Install the Huawei Cloud Stack Provider	Infrastructure Resources for Huawei Cloud Stack	Create a Cluster on Huawei Cloud Stack
VMware vSphere	VMware vSphere Provider	Install the VMware vSphere Provider	VMware vSphere Infrastructure Preparation	Create a Cluster on VMware vSphere

If cluster creation or deletion stalls, see Troubleshooting.

Related Tasks

Scenario	Documentation
Install the `global` cluster on Alauda OS	Installing the global Cluster
Plan disaster recovery for the `global` cluster	Global Cluster Disaster Recovery
Upgrade the `global` cluster	Upgrade the Distribution Version, then Upgrade Kubernetes on Alauda OS
Upgrade workload clusters	Upgrade the Distribution Version, then Upgrade Kubernetes on Alauda OS
Manage cluster nodes	Manage Nodes on Alauda OS Clusters
Configure machines (OS or kernel level)	Machine Configuration

Scenario	Documentation
Look up Cluster API and provider resources	Cluster API Reference and Provider APIs

Prepare

Plan the Control Plane Endpoin
Choose the Image Registry for a Workload Clu

Bare Metal

Overview
Install the Provider
Create a Clu

Select a Fixed System Disk on Multi-Disk Bare-Metal Hosts

Huawei DCS

Overview
Install the Provider
Infrastructu

Create a Cluster
Troubleshoot a DCSMachine Stuck in Deleting

Huawei Cloud Stack

- Overview
- Install the Provider
- Infrastructu
- Create a Cluster
- Troubleshoot Huawei Cloud Stack Workload C

VMware vSphere

- Overview
- Install the Provider
- Infrastructu
- Create a Cluster

Troubleshooting

- Troubleshoot a Workload Clust
- Troubleshoot a Cluster Stuck in Deleting

Prepare

Complete the following preparation before you create a workload cluster on Alauda OS. It applies to every provider. For the list of supported providers, see Supported Providers.

TOC

Install the Infrastructure Provider

Configure Infrastructure Resources

Plan the Control Plane Endpoint

Choose the Image Registry

Install the Infrastructure Provider

After the `global` cluster is installed, and before you create a workload cluster on Bare Metal, Huawei DCS, Huawei Cloud Stack, or VMware vSphere, upload and install the infrastructure provider for that IaaS platform on the `global` cluster. This applies regardless of which IaaS platform the `global` cluster itself runs on. In a DR pair, do this on both `global` clusters.

In addition to the infrastructure provider, the following component is required for every provider:

- **Alauda Container Platform Cluster Bootstrap Provider**: Handles Kubernetes bootstrap and control plane management

Install both as follows:

1. **Download**: Download the packages from the Customer Portal. For ACP 4.5, the providers are listed under **Common ACP Extensions**. See Download.
2. **Upload**: Upload the packages to the platform with `violet push`. See Upload Packages.
3. **Install**: Install the packages as cluster plugins on the `global` cluster. See Cluster Plugin.

For the packages and verification steps of each provider, see:

- Bare Metal
- Huawei DCS
- Huawei Cloud Stack
- VMware vSphere

Configure Infrastructure Resources

Before creating clusters, configure the infrastructure resources that the provider requires:

- **Cloud Credentials**: Store IaaS platform access information
- **IP Pools**: Define network configurations for cluster nodes
- **Machine Templates**: Specify virtual machine specifications
- **Network Inventory and Load Balancer Plans**: Prepare subnet inventories and API server access addresses when the provider requires them

Provider	Description
Huawei DCS	Huawei Datacenter Virtualization Solution
Huawei Cloud Stack	Huawei Cloud Stack
VMware vSphere	vCenter inventory, credentials, network, template, endpoint, and storage preparation

Bare Metal has no separate infrastructure resource page. Its host preparation is part of Install the Bare Metal Provider and Create a Cluster on Bare Metal.

Plan the Control Plane Endpoint

Every cluster needs a stable endpoint for the Kubernetes API, and the endpoint mode must be selected before the cluster is created. See Plan the Control Plane Endpoint.

Choose the Image Registry

By default, a workload cluster uses the image registry of the `global` cluster. A cluster can instead be bound to a dedicated registry, but only when it is created. See Choose the Image Registry for a Workload Cluster.

Plan the Control Plane Endpoin Choose the Image Registry for a Workload Clu

Plan the Control Plane Endpoint

Every cluster needs a stable endpoint for the Kubernetes API. Select the endpoint mode before you create the cluster, and keep the address stable after kubeadm initializes the control plane.

TOC

Endpoint Modes by Provider

External LoadBalancer Contract

Validate an External LoadBalancer

Self-built VIP Contract

Endpoint Modes by Provider

The endpoint modes available in this release depend on the provider.

Provider	Endpoint modes
Huawei DCS	External LoadBalancer, or `type: internal` Self-built VIP when paired with ACP `v4.4+`. A single control-plane node can use its own IP only for development or PoC.
VMware vSphere	External LoadBalancer, or an `internal` Self-built VIP when paired with ACP `v4.4+`, which deploys Alive (keepalived and IPVS) on the control-plane

Provider	Endpoint modes
	nodes.
Bare Metal	`Internal` Self-built VIP and `External` LoadBalancer.
Huawei Cloud Stack	Provider-managed HCS ELB. Use the endpoint fields and ELB workflow in Create a Cluster on Huawei Cloud Stack.

For DCS, `type: internal` requires ACP `v4.4+`. Plan it before cluster creation. Existing clusters that use an external LoadBalancer keep it unless you have a tested migration plan.

For vSphere, `type: internal` requires vSphere Provider `v1.0.17+` and ACP `v4.4+`, and it is a creation-time choice: only a cluster created that way can use it, and a cluster whose provider was upgraded from an earlier release keeps its external LoadBalancer. It also requires the `alive` cluster plugin to be present and deployable on the `global` cluster, because the provider installs Alive through a `ModuleInfo` rather than shipping it itself. Verify all of these before you plan the mode; see Control Plane Endpoint Modes.

External LoadBalancer Contract

Provision and validate the load balancer before you apply the Cluster API resources.

Requirement	Contract
Listener	Use Layer 4 TCP forwarding with TLS passthrough. The Kubernetes API server, not the load balancer, terminates TLS.
Frontend	Use one stable VIP (an IP address) and TCP port `6443`. Reserve the address before cluster creation.
Backends	Add every control-plane node IP on TCP port `6443`. Do not add worker nodes.
Health check	Prefer an HTTPS check to `https://<control-plane-node-ip>:6443/healthz` and require HTTP `200`. If the device supports only TCP checks, document that the check proves port reachability but not API-server health.

Requirement	Contract
Backend ownership	The load balancer owner must update backend membership when control-plane nodes are added, replaced, or removed. The DCS and vSphere providers do not manage external load balancer backends.
Bootstrap reachability	The bootstrap node or current management cluster must reach the frontend endpoint while Cluster API creates the control plane.
Node reachability	All control-plane nodes must reach the endpoint. Workload-cluster endpoints must also be reachable from the `global` cluster.
Name and certificate	Do not put a DNS name (FQDN) in front of the cluster endpoint — cluster access by domain is not supported. The API server certificate is issued by the cluster's own CA, so a customer-issued certificate cannot be served on the endpoint, and a domain without that certificate defeats its usual purpose.

For ACP port requirements, see the following pages:

- For a `global` cluster, see LoadBalancer Forwarding Rules.
- For a workload cluster, see Connecting global Cluster and Workload Cluster.

> **WARNING**
>
> Do not change the control plane endpoint after the cluster is initialized as a routine load-balancer edit. The endpoint is embedded in API server certificates, kubeconfigs, and management-cluster connections. Changing it requires a planned certificate and connectivity migration.

Validate an External LoadBalancer

Before cluster creation, confirm that the listener exists and that the bootstrap or management environment can open the frontend port:

```
nc -vz <control-plane-endpoint> 6443
```

After at least one API server is running, verify the preferred health check from a network location that represents the load balancer:

```
curl -kfsS https://<control-plane-node-ip>:6443/healthz
```

The expected response is `ok` with HTTP status `200`.

Self-built VIP Contract

For Huawei DCS, use this contract only with ACP `v4.4+`. On VMware vSphere it applies to provider `v1.0.17` and later. For Bare Metal, follow the provider-specific guide.

- Reserve an unused IPv4 address in the same Layer-2 network as the control-plane interface that will hold the VIP.
- Use a VRID that is unique in that Layer-2 domain.
- Allow VRRP traffic and gratuitous ARP updates between the control-plane nodes and the surrounding network.
- Use at least three control-plane nodes for high availability.
- Ensure the node image and the supported `alive` package provide the required VIP runtime and IPVS configuration.
- Use an IP address, not a domain name, for the Self-built VIP.
- Let the provider and Alive reconcile the VIP and backend membership when control-plane nodes are replaced. Do not maintain a second manual backend list for the same VIP.

On VMware vSphere the following additional rules apply:

- The VIP must be IPv4. IPv6 and dual-stack endpoints are not accepted.
- `vrid` must be between `1` and `255`. The range is enforced by the provider's admission webhook rather than by the CRD schema, so an out-of-range value is rejected at apply time and not by `kubectl --dry-run=client`.
- The VIP must not equal any slot address (`network.primary.ip` or `network.additional[].ip`) declared in the cluster's `VSphereMachineConfigPool` objects. It must also fall outside every external IPAM pool range; the provider cannot read those pools and will not detect that conflict.
- `spec.controlPlaneLoadBalancer` is written when the cluster is created or never. An absent field already means an external LoadBalancer, so it cannot be filled in later, and a

field that was written cannot be edited or cleared. Correcting a mistake requires rebuilding the cluster.

- On multi-NIC nodes, set `interface` explicitly. Auto-detection matches the node's primary IP, and a wrong choice cannot be patched afterwards.
- `VSphereCluster.status.conditions[SelfBuiltLoadBalancerReady]` reports the VIP chain only. It does not gate `VSphereCluster.status.ready`, so the infrastructure cluster can report ready while the VIP is still converging.

For DCS manifests and verification, see Create a Cluster on Huawei DCS. For Bare Metal manifests, see Create a Cluster on Bare Metal. For vSphere, see Control Plane Endpoint Modes.

Choose the Image Registry for a Workload Cluster

Alauda Container Platform installs and manages a set of platform components on every workload cluster — Kube-OVN and the platform modules delivered after the cluster is up. Their images come from an image registry, and by default that is the same registry the `global` cluster uses.

You can instead bind a workload cluster to a registry of its own. The common reason is distance: when the `global` cluster and the workload cluster sit in different regions, pulling every platform image across that link is slow and fragile. A registry close to the workload cluster removes that dependency.

Decide before you create the cluster. The binding is read only while the cluster is being created, and an existing cluster cannot be moved to another registry.

TOC

- Options
- What the Binding Covers
- Inherit the Registry of the global Cluster
- Bind a Workload Cluster to Its Own Registry
  - Before You Start
  - Step 1 — Prepare the Registry
  - Step 2 — Create the RegistryCredential Secret
  - Step 3 — Reference the Credential When You Create the Cluster

Step 4 — Verify the Binding
Operate an Existing Binding
Troubleshooting

Options

Option	What you configure	When to use it
Registry of the `global` cluster	Nothing. Every workload cluster inherits it.	The workload cluster reaches the registry the `global` cluster uses with acceptable throughput and availability.
Dedicated registry	A RegistryCredential Secret in `cpaas-system`, referenced from the `Cluster` by the `cpaas.io/registry-reference` label.	The workload cluster is in a different region or network segment and should pull from a registry close to it.

Both options are available on every infrastructure provider. The default needs nothing, so it applies to clusters created through the web UI as well. A dedicated registry is configured through YAML only: the cluster creation wizards do not offer a registry credential to select, so use the YAML workflow of your provider's cluster creation guide when a cluster needs one.

What the Binding Covers

The binding applies to the platform components that Alauda Container Platform installs and manages on the workload cluster, such as Kube-OVN and the platform modules delivered to it.

It does **not** apply to the following. Each is configured elsewhere, and pointing a cluster at a dedicated registry leaves all of them unchanged.

Not covered	Where it comes from instead
Images kubeadm manages: kube-apiserver, kube-controller-manager, kube-scheduler, etcd, CoreDNS, and kube-proxy	`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.imageR` which each provider guide sets independently. On Huawei DCS and Bare Metal it is `cloud.alauda.io/alauda`, matching the images preloaded in the operating systen Change that field if these must come from another registry, and make sure the refere match what the node has locally.
The pause (sandbox) image	The container runtime configuration on the node. The supported operating system im in containerd; it is not resolved through any Kubernetes object.
Bare-metal operating system images	Hosts pull `base-image` and `base-image-iso` from the platform registry during `e install` and every reprovision, before any cluster-level binding exists.
Application workload images	The image references and pull Secrets that each workload declares. A binding confic platform image source only.

Inherit the Registry of the global Cluster

This is the default and needs no configuration. Create the cluster without a `cpaas.io/registry-reference` label and it uses the same registry as the `global` cluster.

How the address and credentials reach the workload cluster depends on whether the platform registry requires authentication:

- **Authenticated registry** — the platform stores the address and credentials in the `public-registry-credential` Secret in `cpaas-system` on the `global` cluster. Every workload cluster created without an explicit reference resolves its registry from that Secret. Despite

its name, this Secret holds the registry the platform itself uses; it is not limited to an internet-facing registry.

- **Anonymous registry** — no credential Secret is populated, and platform components fall back to the platform-wide registry address recorded at installation time.

In both cases the operator does not configure anything per cluster.

If workload cluster nodes cannot reach the address of the `Platform Built-in` registry on the `global` cluster, see Add External Address for Built-in Registry.

Bind a Workload Cluster to Its Own Registry

Before You Start

- You can create Secrets in the `cpaas-system` namespace of the `global` cluster. The credential Secret has to exist before the cluster that references it is created.
- The dedicated registry is reachable from every workload cluster node, and from the `global` cluster if you want to verify it from there.
- The dedicated registry already holds Alauda Container Platform Core for the target version, plus the packages for every platform plugin you intend to install on this cluster. See Prepare the Registry.

Step 1 — Prepare the Registry

The platform never copies images into a dedicated registry. Populate it yourself, before you create the cluster.

Two things decide what it must hold:

- **Alauda Container Platform Core for the target version is always required.** Its images and packages must match the version and CPU architecture of the cluster you are about to create.
- **Everything beyond Core depends on which platform plugins you intend to install on this cluster.** Only you know that set — the platform does not derive it and will not fill the gaps later. Upload the packages for those plugins together with Core.

Use the same Core Package upload procedure that prepares an external platform image registry, with the dedicated registry as the target address. See Prepare the Target-Version Payload.

Then confirm from a node in the target network that the registry answers:

```
export DEDICATED_REGISTRY="<registry-host>:<port>"
curl -sk "https://${DEDICATED_REGISTRY}/v2/" -o /dev/null -w '%{http_cod
e}\n'
```

This proves reachability and TLS only. Confirm the uploaded repositories and tags in your registry's own administration interface or API.

The `/v2/` segment belongs only to Registry HTTP API requests. Do not include it — or a scheme — in the Secret or in any cluster field.

> **WARNING**
>
> Verify the contents before you create the cluster. A cluster bound to an incomplete registry still comes up — kubeadm resolves the control plane from a different registry, so Nodes reach `Ready` and the cluster looks healthy. The gap only surfaces afterwards, as platform Pods stuck in `ImagePullBackOff`. That is considerably harder to diagnose than a registry check that fails up front.

Step 2 — Create the RegistryCredential Secret

Create the Secret in the `cpaas-system` namespace of the `global` cluster. The reference on a cluster carries only a name, so only this namespace is consulted; a Secret with the same name in another namespace is ignored.

Author the credential file with an editor rather than typing the values at a shell prompt. A shell records the whole command, password included, in its history; a file written by an editor leaves no such copy.

```
registry-credential.env
```

```
registry=<registry-host>:<port>
username=<username>
password=<password>
```

For an anonymous registry, keep only the `registry` line and omit the other two.

Restrict the file, create the Secret from it, then remove it:

```
chmod 600 registry-credential.env

REGISTRY_CREDENTIAL_NAME="<registry-credential-name>"

kubectl create secret generic "${REGISTRY_CREDENTIAL_NAME}" \
  --namespace cpaas-system \
  --from-env-file=registry-credential.env

rm -f registry-credential.env
```

WARNING

Do not create a credential Secret with `kubectl apply`. Client-side apply stores the manifest it applied — including every value in it — in the `kubectl.kubernetes.io/last-applied-configuration` annotation on the Secret itself. `kubectl create` writes no such annotation. This applies whether the values were written as `stringData` or as base64 `data`; base64 is an encoding, not protection.

Each line of the file is `key=value`. Do not quote the values — `kubectl` takes quote characters literally, so `password="s3cret"` stores `"s3cret"` with the quotes included. A value may contain `=` and other special characters; only the first `=` on a line separates the key from the value.

The Secret is created with type `Opaque`, which is correct here. The platform recognizes a registry credential by its labels, not by its type, so apply both labels before you create any cluster that references it:

```
kubectl label secret "${REGISTRY_CREDENTIAL_NAME}" --namespace cpaas-system --overwrite \
  capi.cpaas.io/provider=registry-credential \
  cpaas.io/registry-credential-format=direct
```

Key	Required	Notes
Namespace	Yes	Must be `cpaas-system`.
`capi.cpaas.io/provider` label	Yes	Must be `registry-credential`.
`cpaas.io/registry-credential-format` label	Yes, for this procedure	Set it to `direct` for a credential you author. The label itself is optional to the platform, but omitting it selects the legacy encrypted format, which expects an uploaded authentication file instead of the keys below.
`registry`	Yes	`<host>[:<port>]` only. No scheme, no path.
`username`, `password`	Together	Set both or neither. Omit both for an anonymous registry. Setting only one is rejected.

WARNING

Do not skip the `capi.cpaas.io/provider` label. A cluster can still be created against a Secret that lacks it, because the creation path reads the three keys directly. But the rest of the platform then does not recognize the Secret as a credential: later changes to its values are never propagated to the clusters that reference it, and it is not protected against deletion while still in use.

Step 3 — Reference the Credential When You Create the Cluster

Add the `cpaas.io/registry-reference` label to the `Cluster` resource, with the Secret name as its value:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
  labels:
    cpaas.io/registry-reference: <registry-credential-name>
```

Apply the rest of the cluster manifests as described in your provider's cluster creation guide.

The Secret must already exist when you do. The platform resolves the reference while it registers the new cluster, and it reads the Secret by name from `cpaas-system` at that moment. If the Secret is not there, registration does not complete and the cluster makes no progress. It is not a permanent failure — the platform keeps retrying, so creating the Secret afterwards releases the cluster — but nothing happens in the meantime, and the reason is not reported on the `Cluster` resource itself. Create the Secret first.

> **WARNING**
>
> The label is read only when the cluster is first created. Adding, changing, or removing it on an existing `Cluster` has no effect on that cluster. To move an existing cluster to another registry, you must recreate the cluster.

Step 4 — Verify the Binding

Run these checks on the `global` cluster. Confirm the reference reached the platform cluster record:

```
export CLUSTER_NAME="<cluster-name>"
kubectl get cluster.platform.tkestack.io "${CLUSTER_NAME}" \
  -o jsonpath='{.metadata.labels.cpaas\.io/registry-reference}{"\n"}'
```

Confirm the platform resolved the registry address from the credential. The platform writes this annotation; do not set it yourself:

```
kubectl get cluster.platform.tkestack.io "${CLUSTER_NAME}" \
  -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}{"\n"}'
```

The value must be the address from the Secret. If it is empty or still shows another registry, see Troubleshooting.

Finally, confirm on the workload cluster that platform Pods pull from the expected registry:

```
kubectl --kubeconfig <workload-cluster-kubeconfig> get pods -A \
  -o jsonpath='{range .items[*]}{range .spec.containers[*]}{.image}{"\n"}
{end}{end}' \
  | sort -u
```

Platform component images use the dedicated registry. Images kubeadm manages, and the pause image, do not — see What the Binding Covers. Some images still resolving to another registry is expected; every platform component image resolving to the old one is not.

Operate an Existing Binding

Task	Supported	How
Rotate the registry password or switch to another account	Yes	Patch the Secret, then remove the patch file. The platform propagates the change to every cluster that references it, without a cluster restart.
Correct the registry address in a credential	Yes, with care	Patched the same way. It repoints every cluster that references this Secret, so verify that all of them can reach the new address and that it holds the payload for each of their versions.
Move an existing cluster to a different registry	No	The reference is fixed at creation. Recreate the cluster with the intended reference.

Task	Supported	How
Share one credential across clusters	Yes	Reference the same Secret from any number of clusters. A single update reaches all of them. Different Secrets never affect each other.
Delete a credential	Only when unreferenced	The platform rejects deletion while any cluster still references the Secret, and the error names those clusters. Remove that reference first, by deleting those clusters or recreating them against a different credential.

Update a credential with `kubectl patch` rather than `kubectl apply`, for the same reason `kubectl create` was used to make it:

registry-credential-patch.yaml

```yaml
stringData:
  username: <new-username>
  password: <new-password>
```

```
chmod 600 registry-credential-patch.yaml

kubectl patch secret "${REGISTRY_CREDENTIAL_NAME}" --namespace cpaas-syst
em \
  --type=merge --patch-file=registry-credential-patch.yaml

rm -f registry-credential-patch.yaml
```

`stringData` in the patch replaces the matching keys in `data`. Quote the values here if they contain YAML metacharacters — unlike the `key=value` file used at creation, this file is YAML.

Troubleshooting

Symptom	Likely cause	Action
`cpaas.io/registry-address` on the platform cluster record stays empty	The Secret does not exist, is outside `cpaas-system`, or is missing the `capi.cpaas.io/provider: registry-credential` label.	Confirm the name, the namespace, and both labels.
The Secret is rejected on create or update	`registry` is empty, or only one of `username` and `password` is set.	Set `registry`, and set the credential pair together or leave both out.
The Secret is rejected with an unsupported-format message	`cpaas.io/registry-credential-format` carries a value other than `direct` or `encrypted`, including an empty value.	Use `direct` for a credential you author, or remove the label to use the legacy encrypted format.
The cluster reports its control plane as ready, but platform Pods stay in `ImagePullBackOff`	The dedicated registry does not hold the payload for this version, or the nodes cannot authenticate to it.	Repeat Step 1 from a workload cluster node and confirm the credential in the Secret.
Deleting a credential is rejected	One or more clusters still reference it.	Read the cluster names from the error. Delete those clusters, or recreate them against a different credential, before deleting this one.
A new label value on an existing `Cluster` changes nothing	The reference is read only at cluster creation.	Recreate the cluster with the intended reference.
The `Cluster` was applied, but no platform cluster record ever appears and the cluster makes no progress	The referenced Secret does not exist in `cpaas-system`, so the platform cannot finish registering the cluster. The `Cluster` resource itself reports no error for this.	Create the Secret. The platform retries, so the cluster proceeds once it exists.

Symptom	Likely cause	Action
`kubectl apply --dry-run=server` accepted the manifests, but the real apply stalls	A server-side dry run returns before the platform resolves the registry reference, so it cannot detect a missing Secret.	Do not rely on a dry run for this. Confirm the Secret exists first: `kubectl -n cpaas-system get secret <registry-credential-name>`.
Value changes to the Secret never reach the clusters, or the Secret can be deleted while clusters still use it	The Secret is missing the `capi.cpaas.io/provider: registry-credential` label.	Apply the label. See Step 2.

Bare Metal

Overview

Install the Provider

Create a Cl

Select a Fixed System Disk on Multi-Disk Bare-Metal Hosts

Bare Metal Provider

TOC

- Overview
- Status
- Key Features
- Differences from VM-Based Providers
- Concepts and Terminology
  - Object hierarchy
  - Bare-metal concepts
    - `MachineRegistration`
    - `SeedImage`
    - `MachineInventory`
    - `MachineInventoryPool`
    - `BaremetalCluster`
    - `BaremetalMachineTemplate`
    - `BaremetalMachine`
    - Image catalog
    - Host lifecycle plans
    - `alive` (control-plane HA)
    - `MachineInventory.spec.observedNetwork`
- API Group
- Supported Kubernetes Versions

Requirements

Documentation

Overview

The Infrastructure Provider for Bare Metal creates and manages Alauda OS clusters on physical servers, with no virtualization layer in between. It composes two long-running components on the `global` cluster:

- **`elemental-operator`** — registers physical hosts, builds installation ISOs (`SeedImage`), and maintains the long-lived `MachineInventory` object for every host. `elemental-system-agent` runs on each host and executes `MachineInventory` plan secrets.
- **`cluster-api-provider-baremetal`** — the Cluster API infrastructure provider. It groups available `MachineInventory` objects into pools, binds each `Machine` to a `MachineInventory`, and writes `clean` / `reprovision` plans that drive the host through Kubernetes node lifecycle.

Unlike VM providers (DCS, vSphere), the bare-metal provider does **not** create or destroy machines. A host is installed once (live ISO → on-disk OS via `elemental install`), registers itself as a `MachineInventory`, and stays in the inventory across the cluster lifecycle. Node "creation" and "deletion" are realized through `elemental upgrade` driven by plans, followed by reboot and cloud-init re-execution.

Status

The provider currently follows a YAML-only workflow. There is no Fleet Essentials UI for bare-metal clusters yet — every step on this page is driven by `kubectl apply`.

Key Features

- **`MachineInventoryPool` allocation model** — operators pre-declare which `MachineInventory` objects may back a `KubeadmControlPlane` or

`MachineDeployment`. The provider picks an `Available` inventory from that pool when a `Machine` is created; nothing is provisioned outside the declared set.

- **Plan-driven node lifecycle** — node attach uses a `reprovision` plan (write cloud-init, `cloud-init clean`, `elemental upgrade`, reboot, cloud-init re-execute, `kubeadm init/join`). Node detach uses a `clean` plan (stop kubelet, clear CRI workload, stop containerd). `MachineInventory` is never deleted by the provider during scaling, upgrade, or cluster deletion.
- **Cluster API native object tree** — uses upstream `Cluster`, `KubeadmControlPlane`, `MachineDeployment`, `Machine`. The provider only owns the infrastructure tree (`BaremetalCluster`, `BaremetalMachine`, `BaremetalMachineTemplate`, `MachineInventoryPool`). There is no custom control-plane CRD.
- **Image-catalog driven Kubernetes versions** — the provider keeps a cluster-scoped `elemental-image-catalog` ConfigMap that maps `Machine.spec.version` to an `elemental upgrade` image. Upgrades become "patch the version, controller resolves the matching image, reprovisions the node."
- **Two control-plane endpoint modes** — `BaremetalCluster.spec.controlPlaneLoadBalancer.type: Internal` deploys Alive (keepalived + IPVS + `kube-lock` Lease arbitration) and reconciles the Self-built VIP. `External` skips Alive and uses a pre-provisioned Layer 4 TCP LoadBalancer whose backends are maintained outside the cluster. See Plan the Control Plane Endpoint.
- **Network identity preservation** — `elemental-register` reports the live-ISO observed network as `MachineInventory.spec.observedNetwork`. The first `elemental install` and every later `reprovision` plan replay that snapshot as cloud-init `network-config v2`, so a host keeps its address, default route, and DNS across the entire lifecycle.
- **Inventory-owned managed data disks** — an operator can declare up to 32 XFS or ext4 data volumes in `MachineInventory.spec.storage`. The provider prepares them before allocation, activates their mount units before kubelet during reprovision, deactivates them during node removal, and preserves their filesystems and data for reuse of the same inventory. See Manage Data Disks on Bare-Metal Hosts.

Differences from VM-Based Providers

Aspect	VM providers (DCS, vSphere)	Bare-metal provider
Node creation	Clone a VM template from a credential-scoped platform	Reprovision an already-installed physical host via `elemental upgrade`
Node deletion	Power-off and delete the VM	Run `clean` plan; host stays in inventory and returns to pool
Pool model	IP pool / hostname pool sized per replica	`MachineInventoryPool` listing concrete `MachineInventory` names
Data on the node	System and template disks recreated on every replacement; declared persistent disks survive (DCS)	`initramfs` clears Kubernetes-managed system state on every reprovision. Data volumes explicitly declared on the long-lived `MachineInventory` can retain their filesystems and data, but the data stays with that inventory and does not follow a replacement `Machine` automatically.
Recovery time	Minutes per replacement	Minutes per `elemental upgrade` + reboot
First install path	One-off VM template upload by the platform admin	Live ISO built by `SeedImage`, booted on the host, `elemental install` to disk
Control-plane LB	External LB supplied by operator	Internal LB managed by `alive` static pods on the control-plane nodes (External LB supported through `type: External`)
UI support	YAML + Fleet Essentials UI (DCS)	YAML only

Concepts and Terminology

Object hierarchy

Layer	Owned resources	Role in the workflow
`elemental.cattle.io`	`MachineRegistration`, `SeedImage`, `MachineInventory`	Registers physical hosts, builds t ISO, and keeps the long-lived ho
Host agent	`elemental-system-agent` on each host	Polls the default plan secret refe `MachineInventory.status.pla` and reports plan state back to `MachineInventory.status.pla`
`infrastructure.cluster.x-k8s.io`	`BaremetalCluster`, `BaremetalMachineTemplate`, `BaremetalMachine`, `MachineInventoryPool`	Implements the Cluster API infra provider, selects inventories from writes `clean` / `reprovision`

The ownership boundary is deliberate: `elemental-operator` owns host registration and `MachineInventory` lifecycle, while `cluster-api-provider-baremetal` owns the Cluster API infrastructure tree. The provider references `MachineInventory` objects by name and writes plans through the inventory's existing plan secret; it does not create or delete inventories.

Bare-metal concepts

`MachineRegistration`

Declares the registration endpoint and first-install cloud-config that `elemental-register` consumes on the live ISO. Operators set `machineName`, `machineInventoryLabels`, and `machineInventoryAnnotations` (with `${SMBIOS/...}` templating) plus the `elemental.install` and `elemental.registration` blocks. `MachineRegistration` is queried but not modified by the bare-metal provider — it belongs to the elemental layer.

`SeedImage`

Triggers `elemental-operator` to build a bootable ISO that contains the registration URL and the operator's TLS material baked into it. `spec.baseImage` must reference the **ISO** variant of the OS image that matches the target Kubernetes version (the repository name carries the `-iso` suffix; the tag/digest matches the `elemental-image-catalog` entry for the

version you intend to install). The ISO is booted once per physical host; on boot, the host runs `elemental-register` followed by `elemental install` and creates a `MachineInventory`.

`MachineInventory`

The long-lived host identity object. The provider relies on the following parts of its contract:

- `status.plan.secretRef` — the single default plan secret owned by `elemental-operator`.
- `status.plan.state` — `Applied` / `Failed` transitions used to drive the `BaremetalMachine` state machine.
- `status.conditions` — host-side readiness signals.
- `spec.observedNetwork` — fork-only field populated by `elemental-register` from the live-ISO NICs; replayed during install and during every `reprovision` plan.
- `spec.storage` — optional operator-authored declaration of inventory-owned data volumes.
- `status.observedStorage` — objective device, filesystem, mount, and multipath facts reported by `elemental-storage-observer.service` on the host.
- `status.storage` — provider-owned Prepare / Activate / Deactivate / Release state, operation journal, conditions, and applied-volume identities.

The provider never deletes a `MachineInventory`, never uses `MachineInventorySelector`, and does not run a separate `MachineInventoryLifecycleController` — that lifecycle remains with `elemental-operator`.

`MachineInventoryPool`

Operator-authored set of `MachineInventory` names that a given `BaremetalMachineTemplate` is allowed to draw from. Pools are scoped to a single `clusterName`, and each `MachineInventory` belongs to at most one active pool at a time. The pool reconciler aggregates the pool-wide capacity counters used everywhere in the docs:

- `available` — free for allocation

- `allocated` — bound to an active `BaremetalMachine`
- `preparing` — running a `clean` plan
- `reprovisioning` — running a `reprovision` plan
- `unavailable` — Ready=False, plan failed, missing plan secret, or not present in the cluster

`BaremetalCluster`

The Cluster API infrastructure cluster resource. Owns `controlPlaneLoadBalancer` (the VIP and `vrid` consumed by the `alive` chart) and `controlPlaneEndpoint` (backfilled from `controlPlaneLoadBalancer` when only the VIP is set). The reconciler defers cluster-addon deployment (`alive`, `kube-ovn`) until the workload control plane is reachable.

`BaremetalMachineTemplate`

The Cluster API infrastructure template referenced by `KubeadmControlPlane.spec.machineTemplate.infrastructureRef` and by `MachineDeployment.spec.template.spec.infrastructureRef`. Templates only carry `machineInventoryPoolRef` (which pool this machine group draws from). The provider picks the first `Available` inventory in the pool's declaration order.

There is deliberately no `version`, `role`, or `upgradeImage` on the template. Role comes from the owning Cluster API resource; the version comes from `Machine.spec.version`; the upgrade image is resolved at reprovision time from the global image catalog.

`BaremetalMachine`

The Cluster API infrastructure machine. Reconciles a single `Machine` against a single `MachineInventory`:

1. Picks an `Available` inventory from the pool referenced by the owning template.
2. Reads the owning `Machine.spec.bootstrap.dataSecretName` and resolves the elemental upgrade image for `Machine.spec.version` from the image catalog.
3. Normalizes the bootstrap user-data (hostname, kubelet `provider-id`, `criSocket`) and writes the `reprovision` plan into the `MachineInventory` plan secret.

4. Watches `MachineInventory.status.plan.state` until the plan reports `Applied`, then sets `BaremetalMachine.spec.providerID = baremetal:///<inventory-name>`.
5. On deletion, writes a `clean` plan and clears the owner annotations once the plan applies, returning the inventory to the pool.

Phase transitions: `Pending → Allocated → Reprovisioning → Running`; deletion: `Running → Preparing → Deleted`; failure: `* → Failed`.

Image catalog

A cluster-scoped `ConfigMap` (default name `elemental-image-catalog` in `cpaas-system`) that maps `Machine.spec.version` to an `elemental upgrade` image. The bare-metal provider chart renders this ConfigMap from `provider.imageCatalog.images` (`global.registry.address` is prepended to each repository) and from `provider.imageCatalog.data` (for fully-qualified overrides such as digest-pinned images). Both values are empty by default, so the ConfigMap has no entries until you add them; see OS Images Imported and Image Catalog Populated. The reconciler hot-reloads the ConfigMap; a missing key is a terminal `Failed` state, not a fallback to a default image.

Obtain the matching `base-image` and `base-image-iso` references from Alauda technical support. The image catalog uses `base-image` for reprovisioning, while `SeedImage.spec.baseImage` uses `base-image-iso` for live installation. Do not derive one reference from the other or substitute an independently built image.

Host lifecycle plans

The provider serializes all host work through the inventory's single default plan secret. The secret's `baremetal.alauda.io/plan.type` annotation identifies the current lifecycle plan:

- **`storage-prepare`** — runs while an inventory is available and before allocation. It preflights every declared volume, then adopts an existing matching filesystem or initializes an explicitly authorized blank device. It does not mount business paths.
- **`reprovision`** — runs on node attach. It embeds managed-volume activation, writes NoCloud `user-data` / `meta-data` (and, when `MachineInventory.spec.observedNetwork` is non-empty, `network-config` v2), writes a cleanup marker, runs `cloud-init clean --logs --seed`, runs `elemental upgrade --reboot=false --system <image>`, and triggers a delayed reboot. After reboot,

`initramfs` clears `/var/lib/kubelet`, `/var/lib/containerd`, `/var/lib/etcd`, and `/etc/kubernetes`; managed mount units start before kubelet, and cloud-init performs `kubeadm init` / `kubeadm join`.

- **`clean`** — runs on node detach. It stops kubelet, deactivates managed mounts, clears CRI workload, and stops containerd. It explicitly **does not** wipe managed filesystems, run `kubeadm reset`, run `cloud-init clean`, run `elemental upgrade`, or reboot. OS cleanup is deferred to the next `reprovision` plan.
- **`storage-release`** — runs only while an inventory is available and all managed volumes are inactive. It removes provider ownership and activation metadata for volumes removed from `spec.storage` without wiping their filesystems or data.

The provider does not use `MachineInventorySelector` or `FleetBundle`. Storage lifecycle details, safety gates, and examples are documented in Manage Data Disks on Bare-Metal Hosts.

`alive` (control-plane HA)

`alive` is a set of static pods (keepalived + IPVS) plus a `kube-lock` Lease arbitrator that maintain the control-plane VIP described in `BaremetalCluster.spec.controlPlaneLoadBalancer`. The provider deploys `alive` as an `AppRelease` on the workload cluster once the first control-plane Node is Ready, and re-renders it whenever the control-plane membership changes. During the very first `kubeadm init`, the provider prepends a one-off `ip addr add <control-plane-vip>/32 dev eth0` command to the bootstrap so that the first node holds the VIP until `alive` takes over.

The VIP must live in the control-plane nodes' Layer-2 broadcast domain; `vrid` must be unique within that domain.

`MachineInventory.spec.observedNetwork`

Fork-only field populated by `elemental-register` from the live-ISO NIC state and reported back via the `MsgObservedNetworkConfig` registration message. It is consumed in two places:

- During the first `elemental install`, when `MachineInventory.spec.network` is empty, the registration server falls back to translating `observedNetwork` into an `nmconnections` `NetworkConfig` so the on-disk OS keeps the same address that the live ISO had.

- During every `reprovision`, the provider translates `observedNetwork` into cloud-init `network-config` v2 (a netplan subset) and writes it as the third NoCloud seed file. Explicit `spec.network` takes precedence in both cases.

API Group

All bare-metal infrastructure resources belong to `infrastructure.cluster.x-k8s.io/v1beta1`. Elemental resources belong to `elemental.cattle.io/v1beta1`.

Resource	Description	Documentation
`BaremetalCluster`	Cluster-level infrastructure (control-plane VIP, endpoint, network type)	BaremetalCluster
`BaremetalMachine`	Single infra Machine bound to one `MachineInventory`	BaremetalMachine
`BaremetalMachineTemplate`	Template that binds a pool to a `KubeadmControlPlane` or `MachineDeployment`	BaremetalMachineTemplate
`MachineInventoryPool`	Allowed set of `MachineInventory` names for a cluster	MachineInventoryPool

Supported Kubernetes Versions

Use the product support matrix for the Kubernetes versions supported by each ACP release. See OS Support Matrix for the release-approved Kubernetes version and matching component versions.

The bare-metal provider enforces that support decision through `elemental-image-catalog`: every `Machine.spec.version` used by a cluster must exist as a key in that ConfigMap. The catalog wiring and out-of-band override procedure are documented in Installation and Upgrade Kubernetes.

Requirements

- Physical hosts (or PXE-bootable VMs for lab use) with BIOS or UEFI access to mount the `SeedImage` ISO.
- A platform registry reachable from each host (used by both `elemental install` and `elemental upgrade`). Set `global.registry.address` and, when the registry is self-signed, leave `global.registry.tlsVerify=false` (chart default).
- A control-plane VIP, free port (typically `6443`), and a `vrid` unique within the control-plane Layer-2 broadcast domain.
- One `MachineInventoryPool` per role (control plane, worker) sized to at least the target replica count plus headroom for upgrades.
- A host with a real hardware TPM (`/dev/tpm0`) can register with `emulate-tpm: false`. Any host without a hardware TPM — virtual machines and physical servers that ship without a TPM module alike — must set `MachineRegistration.spec.config.elemental.registration.emulate-tpm: true`; the physical-versus-virtual distinction does not apply.
- Managed data disks additionally require an observer-capable host OS image with `elemental-storage-observer.service` enabled. Upgrading only the management-side provider and `elemental-operator` does not add this service to hosts installed from an older image.

Documentation

For detailed instructions on using the bare-metal provider, see:

- Installation
- Create a Cluster
- Manage Nodes
- Managing Data Disks
- Upgrade Kubernetes
- API Reference

Install the Bare Metal Provider

This document describes how to install the Infrastructure Provider for Bare Metal and the Cluster Bootstrap Provider on the `global` cluster, and how to prepare a `global` cluster installed on Bare Metal as a management cluster.

TOC

- When You Need This
- Prerequisites
- Install the Provider Plugins
  - Download the packages
  - Upload the packages
  - Install the cluster plugins
  - Verify the installation
- Prepare the global Cluster as a Management Cluster
  - Point the provider at the platform certificate that actually exists
  - Install the Cluster API providers on the global cluster
  - Upload the base image to the global Registry
  - Create the elemental image catalog
  - Workload registrations use the platform domain
- Next Steps

When You Need This

Complete Install the Provider Plugins after the `global` cluster is installed and before you create the first workload cluster on Bare Metal. It is required regardless of the IaaS platform that the `global` cluster itself runs on. In a DR pair, complete it on **both** `global` clusters, so that the standby can reconcile the same workload clusters after a failover.

If the `global` cluster was installed on Bare Metal, also complete Prepare the global Cluster as a Management Cluster. Every `global` cluster installed on Bare Metal needs it, whether or not you plan Bare Metal workload clusters:

- Complete it before you decommission the bootstrap cluster.
- After handoff, reprovisioning a `global` node pulls `baremetal-base-image` from the `global` Registry, which this section populates.
- Complete it before you create Bare Metal workload clusters.

This is a separate task from the provider steps in Installing the global Cluster. Those steps upload provider packages to the temporary bootstrap cluster to create the `global` cluster itself.

Prerequisites

Before installing the provider, ensure you have:

- Access to the Alauda Container Platform `global` cluster.
- Access to Customer Portal for downloading plugins.

Install the Provider Plugins

1 Download the packages

> The bare-metal workflow requires two cluster plugins. Download both from the Customer Portal. For ACP 4.5, they are listed under **Common ACP Extensions**; see Download.

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Bare Metal** (umbrella chart that ships the `cluster-api-provider-baremetal-manager` and `elemental-operator` subcharts together)

2 Upload the packages

Upload both packages to the platform with `violet push`. For the command and its parameters, refer to Upload Packages.

3 Install the cluster plugins

Install both cluster plugins on the `global` cluster. For the procedure, refer to Cluster Plugin.

If the `global` cluster was installed on Bare Metal, set the following values when you install the Infrastructure Provider for Bare Metal. Determine them before you install:

- `elemental.tls.caCertSecretKey`: the key found in `dex.tls`; see Point the provider at the platform certificate that actually exists.
- In a DR pair, `elemental.systemAgent.splitAuthEnabled: true` on both sides, and `elemental.systemAgent.sharedAuthReadOnly: false` on the active cluster and `true` on the standby. Each `global` cluster keeps its own values; DR synchronization does not copy the provider's `ModuleInfo`.

Install the Infrastructure Provider for Bare Metal with YAML as described in Installing via YAML, and put these values in `spec.valuesOverride` of its `ModuleInfo`; the Web Console install form has no fields for them:

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: ModuleInfo
metadata:
  labels:
    cpaas.io/cluster-name: global
    cpaas.io/module-name: cluster-api-provider-baremetal
    cpaas.io/module-type: plugin
  name: global-temporary-name
spec:
  config: {}
  version: <provider-plugin-version>
  valuesOverride:
    acp/chart-cluster-api-provider-baremetal:
      elemental:
        tls:
          caCertSecretKey: <key-found-in-dex.tls>
        systemAgent:                  # DR pair only
          splitAuthEnabled: true
          sharedAuthReadOnly: false   # true on the standby
```

If the plugin is already installed, add the same `valuesOverride` to its existing `ModuleInfo`: find it with `kubectl get moduleinfo -l cpaas.io/cluster-name=global,cpaas.io/module-name=cluster-api-provider-baremetal`, then run `kubectl edit moduleinfo <name>`. Do not edit the provider's `AppRelease`: the platform regenerates it from the `ModuleInfo`, so the change is lost.

The umbrella chart installs both the bare-metal provider manager and `elemental-operator` in one step. It also creates the `elemental-image-catalog` ConfigMap, but with no entries, and the package contains no Alauda OS image. Before you create a cluster, import the Bare Metal OS images into the platform registry and add every Kubernetes version you deploy to the catalog; see OS Images Imported and Image Catalog Populated.

4 Verify the installation

After installing the plugins, verify that the controllers and CRDs are present:

```
# Provider manager + elemental-operator should both be Running
kubectl -n cpaas-system get pods | grep -E 'baremetal|elemental'

# Bare-metal CRDs
kubectl get crd | grep -E '^(baremetalclusters|baremetalmachines|bare
metalmachinetemplates|machineinventorypools)\.infrastructure\.cluster
\.x-k8s\.io$'

# Elemental CRDs (shipped by the elemental subchart)
kubectl get crd | grep -E '^(machineinventories|machineregistrations|
seedimages)\.elemental\.cattle\.io$'

# Image catalog ConfigMap
kubectl -n cpaas-system get configmap elemental-image-catalog
```

Expected output includes:

- `baremetalclusters.infrastructure.cluster.x-k8s.io`
- `baremetalmachines.infrastructure.cluster.x-k8s.io`
- `baremetalmachinetemplates.infrastructure.cluster.x-k8s.io`
- `machineinventorypools.infrastructure.cluster.x-k8s.io`
- `machineinventories.elemental.cattle.io`
- `machineregistrations.elemental.cattle.io`
- `seedimages.elemental.cattle.io`

Right after installation `elemental-image-catalog` has no `data` entries. After you add the Kubernetes versions you intend to deploy, confirm them:

```
kubectl -n cpaas-system get configmap elemental-image-catalog -o yaml
```

Every `Machine.spec.version` used by a cluster — both at creation time and at upgrade time — must appear as a key in this ConfigMap. A missing key leaves the `BaremetalMachine` in `Failed` with `Reason=ImageCatalogMiss`.

Prepare the global Cluster as a Management Cluster

> **Bare Metal only**
>
> This section applies to a `global` cluster installed on Bare Metal.

Installing the global Cluster creates the `global` cluster itself. Creating Bare Metal workload clusters afterwards is a separate concern: the `global` cluster then acts as the Cluster API management cluster, and it needs provider components and artifacts that the bootstrap preparation did not put there. Complete this section on **both** `global` clusters of a DR pair — after a failover the standby becomes the active management cluster and must be able to reconcile the same workload clusters.

Importing the Cluster API resources during installation only creates the CRDs. It does not install the Bare Metal provider or the Cluster Bootstrap Provider on the `global` cluster.

Set the values this section uses. Run it once per `global` cluster, with `GLOBAL_KUBECONFIG` pointing at the cluster you are preparing. `OS_IMAGE_TAG` and `K8S_VERSION` are the values used in Step 3 and Step 1 of the `global` installation.

```
export GLOBAL_KUBECONFIG="<path-to-this-global-kubeconfig>"
# Tag of the Bare Metal Alauda OS images imported in Step 3 of the global
installation.
export OS_IMAGE_TAG="<os-image-tag>"
# K8S_VERSION from Step 1 of the global installation.
export K8S_VERSION="<target-kubernetes-version>"
```

Read the Registry address and password of the cluster you are preparing from its own `ProductBase` and `cpaas-system/registry-admin` Secret:

```
GLOBAL_REGISTRY_ADDRESS="$(kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" \
  get productbase.product.alauda.io base -o jsonpath='{.spec.registry.add
ress}')"
GLOBAL_REGISTRY_PASSWORD="$(kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -
n cpaas-system \
  get secret registry-admin -o jsonpath='{.data.password}' | base64 -d)"
```

Point the provider at the platform certificate that actually exists

The Bare Metal provider mounts a CA bundle for `platformUrl` from `elemental.tls.caCertSecretName` / `caCertSecretKey`, which default to `dex.tls` and `ca.crt`. Those defaults assume a cert-manager-issued `dex.tls`. On the `global` cluster `dex.tls` comes from the installer's `console.cert` flow instead, and a `thirdParty` certificate produces a Secret with only `tls.crt` and `tls.key`. With the default key the operator never starts:

```
MountVolume.SetUp failed for volume "elemental-ca-cert":
  references non-existent secret key: ca.crt
```

Check which keys the Secret actually has and set `caCertSecretKey` accordingly in the provider's `ModuleInfo`, as described in Install the cluster plugins. For a self-signed platform certificate the certificate is its own CA, so `tls.crt` is the correct key:

```
kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system \
  get secret dex.tls -o jsonpath='{.data}' | jq -r 'keys | join(",")'
```

Install the Cluster API providers on the global cluster

Upload the same Bare Metal and Cluster Bootstrap Provider packages you used for the bootstrap cluster to this `global` cluster, then install both plugins as described in Install the Provider Plugins. For the standby `global` cluster of a DR pair, upload the packages to the standby as described in Upload Packages.

When you install the Bare Metal provider, set `caCertSecretKey` to the key you found in the previous step. In a DR pair, also set the DR fields to the same values as on that side's bootstrap release: `elemental.systemAgent.splitAuthEnabled: true` on both sides, `sharedAuthReadOnly: false` on the active cluster and `true` on the standby. Set only these values, in `spec.valuesOverride` of the provider's `ModuleInfo`, as described in Install the cluster plugins. This `global` cluster does not inherit the bootstrap release values; do not copy the other bootstrap values, such as `handoffHook.*`.

Verify that the manager and `elemental-operator` are running on this `global` cluster before continuing:

```
kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system get deploy \
  cluster-api-provider-baremetal-manager elemental-operator
```

Upload the base image to the global Registry

Workload `SeedImage` builds run on the `global` cluster and pull the base ISO from the Registry recorded in `ProductBase.spec.registry.address`. After handoff, reprovisioning a `global` node also pulls `baremetal-base-image` from that Registry, because the installer rewrites the `global` cluster's `cpaas.io/registry-address` annotation to it. The images imported in Step 3 live in the bootstrap Registry only, so the ISO build fails with `MANIFEST_UNKNOWN` until you push both of them to the `global` Registry. On the bootstrap node, push them from the `bare-metal-os` containerd namespace. `--skip-verify` requires `--local`:

```
for repo in baremetal-base-image baremetal-base-image-iso; do
  ctr -n bare-metal-os images push --local --skip-verify \
    -u "admin:${GLOBAL_REGISTRY_PASSWORD}" \
    "${GLOBAL_REGISTRY_ADDRESS}/tkestack/${repo}:${OS_IMAGE_TAG}" "${rep
o}:${OS_IMAGE_TAG}"
  curl -sk -u "admin:${GLOBAL_REGISTRY_PASSWORD}" \
    "https://${GLOBAL_REGISTRY_ADDRESS}/v2/tkestack/${repo}/tags/list"
done
```

Each response must list `${OS_IMAGE_TAG}`. After that, remove the local copies with `ctr -n bare-metal-os images rm "baremetal-base-image:${OS_IMAGE_TAG}" "baremetal-base-`

`image-iso:${OS_IMAGE_TAG}"`. If the images are no longer on the bootstrap node, import them from the OS archive instead, as described in OS Images Imported and Image Catalog Populated.

Create the elemental image catalog

The provider maps a Kubernetes version to an elemental upgrade image through a ConfigMap located by `--image-catalog-namespace` and `--image-catalog-name`, which default to `cpaas-system` and `elemental-image-catalog`. The provider chart creates this ConfigMap with no version entries, and the `global` cluster does not inherit the entry you set on the bootstrap release. Until you add an entry, the first `BaremetalMachine` of a workload cluster fails immediately:

```
ImageResolved=False  ImageCatalogMiss:
  no elemental upgrade image mapped for Kubernetes version "<version>"
```

Add one entry per Kubernetes version that this `global` cluster manages, including `${K8S_VERSION}` for its own nodes. Patch the existing ConfigMap rather than replacing it, so that entries added earlier are kept:

```
kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system \
  patch configmap elemental-image-catalog --type merge \
  -p "{\"data\":{\"${K8S_VERSION}\":\"${GLOBAL_REGISTRY_ADDRESS}/tkestac
k/baremetal-base-image:${OS_IMAGE_TAG}\"}}"

kubectl --kubeconfig "${GLOBAL_KUBECONFIG}" -n cpaas-system \
  get configmap elemental-image-catalog -o jsonpath='{.data}'
```

ImageCatalogMiss does not recover on its own

A `BaremetalMachine` that hit `ImageCatalogMiss` stays `Failed` after you fix the catalog. Re-reconciling the object and restarting the provider both leave it unchanged; the reconciler deliberately treats a missing mapping as terminal rather than falling back to a default image. Delete the `Machine` so that its owner, `KubeadmControlPlane` or the `MachineSet`, recreates it. Populate the catalog before you apply the first workload cluster manifest to avoid this entirely.

Workload registrations use the platform domain

A workload `MachineRegistration` must not carry the `baremetal.cluster.io/system-agent-auth-scope: global` annotation. Its registration URL resolves to `https://<platform-domain>/...`, and the resulting `MachineInventory` is annotated with `system-agent-auth-scope: shared`. That shared identity, together with the shared ServiceAccount signing key, is what lets a workload host keep reporting through the platform domain after a failover.

Next Steps

After installing the provider, you can proceed to:

- Create a Cluster
- Review Provider Overview for the pool model and plan lifecycle.

Create a Cluster on Bare Metal

This document explains how to create Kubernetes clusters on physical servers using the bare-metal provider. The workflow is YAML-only — there is no Fleet Essentials UI for bare-metal clusters at this time.

TOC

Prerequisites

1. Required Plugin Installation
2. OS Images Imported and Image Catalog Populated
3. Network Connectivity
4. Host Time Synchronization
5. TPM Decision
6. Workload Cluster Image Registry
7. Host Disk and Boot Preparation
8. Node Labels and Taints

Cluster Creation Workflow

Resolving Placeholder Values

Step 1: Build the SeedImage and Register Hosts

Size COS_STATE before the first install

Optional: Prepare Managed Data Disks

Step 2: Create `MachineInventoryPool` Resources

Step 3: Create the Control-Plane Cluster Resources

Step 4: Deploy Worker Nodes

Confirm the Worker Pool Capacity
Configure the Worker `BaremetalMachineTemplate`
Configure the Bootstrap Template
Configure the `MachineDeployment`
Cluster Verification
Using kubectl
Verify the Control Plane Endpoint
Expected Results
Common Failure Modes
Next Steps
Appendix
Complete `KubeadmControlPlane` Configuration

Prerequisites

Before creating clusters, ensure all of the following prerequisites are met.

1. Required Plugin Installation

Before you create the first workload cluster on Bare Metal, upload and install the following plugins on the `global` cluster. This applies regardless of the IaaS platform that the `global` cluster itself runs on:

- **Alauda Container Platform Cluster Bootstrap Provider**
- **Alauda Container Platform Infrastructure Provider for Bare Metal** (umbrella chart that installs both the bare-metal manager and `elemental-operator`)

For the procedure, see Install the Provider. If the `global` cluster was installed on Bare Metal, also complete Prepare the global Cluster as a Management Cluster.

2. OS Images Imported and Image Catalog Populated

Before creating any cluster resources, import **both** bare-metal OS images into the target platform registry, then add the target Kubernetes version to `elemental-image-catalog`. Neither step happens on its own: the provider package contains no OS image, and the provider chart creates the image catalog with no entries.

Image	Used by	Purpose
OS image (`base-image`), repository `baremetal-base-image`	`elemental-image-catalog`	Reprovisions a registered host with the Kubernetes-specific root filesystem.
OS ISO image (`base-image-iso`), repository `baremetal-base-image-iso`	`SeedImage.spec.baseImage`	Builds the live ISO used for initial host installation and registration.

Importing only one image is not sufficient. Adding an image-catalog entry or a `SeedImage` reference does **not** import the corresponding image.

Import the images. The Bare Metal OS download for your ACP release is an archive of one OCI image layout that holds both images under the same tag; see Image Format by Provider. Run the import as root on a host that has containerd and can reach the platform registry, such as a `global` control-plane node; Alauda OS ships `ctr` but not `skopeo`. `/var/lib/containerd` needs free space of at least the archive size. Import the archive into a dedicated containerd namespace and list the image names it creates:

```
ctr -n bare-metal-os images import --all-platforms --no-unpack <bare-metal-os-archive>
ctr -n bare-metal-os images ls -q
```

The output lists `baremetal-base-image:<os-image-tag>` and `baremetal-base-image-iso:<os-image-tag>`. Push both images and confirm that the registry lists the tag for each of them. `--skip-verify` requires `--local`:

```
for repo in baremetal-base-image baremetal-base-image-iso; do
  ctr -n bare-metal-os images push --local --skip-verify \
    -u "<registry-username>:<registry-password>" \
    "<registry-address>/tkestack/${repo}:<os-image-tag>" "${repo}:<os-ima
ge-tag>"
  curl -sk -u "<registry-username>:<registry-password>" \
    "https://<registry-address>/v2/tkestack/${repo}/tags/list"
done

ctr -n bare-metal-os images rm \
  "baremetal-base-image:<os-image-tag>" "baremetal-base-image-iso:<os-ima
ge-tag>"
```

The last command removes the local copies after both pushes succeed. For the registry of the `global` cluster, the `admin` password is stored in the `cpaas-system/registry-admin` Secret. In a disaster recovery pair, import the images into the registry of both `global` clusters; see Prepare the global Cluster as a Management Cluster.

After the import, `<base-image>` is `<registry-address>/tkestack/baremetal-base-image:<os-image-tag>` and `<base-image-iso>` is `<registry-address>/tkestack/baremetal-base-image-iso:<os-image-tag>`. Take both images from the same archive. Do not point cluster resources at a build registry, pair images from different archives, or substitute an independently built image.

Populate the image catalog. The `elemental-image-catalog` ConfigMap in `cpaas-system` maps `Machine.spec.version` to the `elemental upgrade` image used to (re)provision a node. The provider chart creates it with no entries, so no node can be provisioned until you add the target version. Patch the ConfigMap rather than replacing it, so that existing entries are kept, then check the result:

```
kubectl -n cpaas-system patch configmap elemental-image-catalog --type me
rge \
  -p '{"data":{"<kubernetes-version>":"<base-image>"}}'

kubectl -n cpaas-system get configmap elemental-image-catalog -o yaml
```

Every value used as `Machine.spec.version` (for both the control plane and worker `MachineDeployment` resources) must appear as a key in this ConfigMap, with the leading

`v` preserved, and resolve to the imported `base-image`. The provider resolves the image at reprovision time by substituting the platform registry address for the registry portion of the entry. If a target version is missing, no `reprovision` plan is written and `BaremetalMachine` ends up in `Failed / Reason=ImageCatalogMiss`. Adding the entry afterwards does not clear that state; delete the affected `Machine` so that its owner recreates it.

3. Network Connectivity

Bare-metal hosts are registered, installed, and joined entirely over the network, so every path in the table below has to be open before the first host boots the `SeedImage` ISO.

The host firewall on an Alauda OS node is preconfigured and maintained by the platform — you do not manage it, and this table is not a node firewall rule set. See Managing Firewall Ports. What you do have to plan is the network **between** the hosts and the endpoints they reach: site firewalls, security groups, and router ACLs between the host subnet, the platform, and the registry.

Stage	Source	Destination	Protocol / port
First boot	Each host booted from the `SeedImage` ISO	Site DNS servers	UDP and TCP `53`
First boot	Each host	Site NTP servers	UDP `123`
First boot	Each host booted from the `SeedImage` ISO	Platform access address (`global.platformUrl`)	TCP `443`, or the custom p
First boot	Each host	Platform registry (`global.registry.address`)	TCP `11443`

Stage	Source	Destination	Protocol / port
Provisioning and steady state	Each workload-cluster host	Platform access address, path `/kubernetes/global`	TCP `443`, or the custom p
Cluster join	Every node of the new cluster	This cluster's control-plane endpoint	TCP `6443`, or the port set `BaremetalCluster.spec.`
Cluster join	Every node of the new cluster	Platform registry resolved from the `cpaas.io/registry-address` annotation	TCP `11443`
Cluster management	`global` cluster nodes	This cluster's control-plane endpoint	TCP `6443`, or the configu
Cluster runtime	Every node of the new cluster	Every other node of the same cluster	etcd `2379` and `2380` bet Kube-OVN overlay tunnel, U encapsulation; `kubelet`; a not filtered by port

WARNING

`443` alone is not enough for a workload host

`elemental-register` and `elemental-system-agent` both reach the platform through the platform access address, so registration and plan polling need only the platform HTTPS port. That is not the whole requirement. The same host still resolves names through DNS, pulls its operating system and platform images from the registry on `11443`, joins the API server on the control-plane endpoint port, and exchanges cluster traffic with the other nodes. A firewall approval that covers only `443` produces hosts that register successfully and then fail during install or join, where the cause is much harder to see.

Scope of the table. These are the paths for workload-cluster hosts, which is what this page creates. Two neighbouring cases are documented elsewhere and are not covered above:

- **Bootstrap.** While `global` itself is still being installed, its hosts register against the temporary bootstrap endpoint and a different set of ports applies — see Bootstrap node network.
- **`global` hosts after handoff.** A `global` host's system agent uses the Global control-plane endpoint rather than the platform ingress path, and in a DR pair it uses the local control-plane VIP on `6443` directly. See Endpoint and Identity.

A `SeedImage` built during the bootstrap phase points at the bootstrap endpoint and must not be reused after handoff; create new `MachineRegistration` and `SeedImage` objects on the active `global` cluster. A workload `SeedImage` created on the `global` cluster stays valid across a handoff, because its registration URL is the platform domain rather than a bootstrap address.

Control-plane endpoint.

- For `Internal` Self-built VIP, the VIP must live in the same Layer-2 broadcast domain as the control-plane node IPs. The `vrid` must be unique in that domain, the network must allow VRRP and gratuitous ARP updates, and the node image must expose IPVS and allow Alive to set `net.ipv4.conf.all.arp_accept=1` and `net.ipv4.vs.conntrack=1`.
- For `External` LoadBalancer, provision the listener and control-plane backends before cluster creation. Follow Plan the Control Plane Endpoint.

First-boot addressing.

If the target VM or physical host does not receive an address from DHCP while booted into the live ISO, configure the network manually from the host console before waiting for registration. Check the NetworkManager connection name first, then apply the site-specific address, gateway, and DNS values:

```
nmcli connection show
nmcli connection modify "Wired connection 1" \
  ipv4.method manual \
  ipv4.addresses 192.168.254.73/24 \
  ipv4.gateway 192.168.254.1 \
  ipv4.dns 192.168.16.19 \
  ipv4.ignore-auto-dns yes
nmcli connection up "Wired connection 1"
```

4. Host Time Synchronization

Every host that will become a cluster node — control plane and worker alike — must already carry the correct time when it boots the `SeedImage` ISO. The platform's cross-node requirement is a unified time zone on every cluster node and a time synchronization error between nodes of no more than 10 seconds; see Node Checks. That requirement describes the cluster's nodes, so it applies here, but nothing on this path verifies it for you: the quick-configuration script and node checks on that page target non-Alauda OS machines that the platform joins over SSH, and they never run against an Alauda OS host installed by `elemental-operator`.

Treat time synchronization as a gate before Step 1:

- Make the site NTP servers reachable from the host subnet, and open UDP `123` as listed in Network Connectivity. A production cluster should not depend on public NTP.
- Set the clock **from firmware**, not from an operating system. At this point the install disk has been wiped as described in Host Disk and Boot Preparation, so there is no OS on the host to run a command in. Correct the time in the BIOS/UEFI setup or through the BMC, and keep the hardware clock on UTC so that the time zone is applied consistently once Alauda OS is installed.

- Compare the hosts against each other, not only against a reference. Across all hosts intended for the same cluster, the spread must stay inside the 10-second requirement before any of them boots the ISO.
- After the nodes reach `Ready`, confirm the result on each node with `timedatectl` — it reports both the time zone and whether the clock is synchronized — and `chronyc tracking` for the source and current offset.

The whole cluster PKI inherits the clock of the host that runs `kubeadm init`. kubeadm sets the certificate authority's `NotBefore` from that host's current time, every leaf certificate inherits that `NotBefore`, and each certificate's `NotAfter` is derived from it. A host running ahead therefore issues a PKI that the other nodes and any client treat as not yet valid until their own clocks catch up, and a host running behind issues one that expires earlier than the configured validity period implies. A server whose hardware clock has never been synchronized is typically wrong by minutes or hours rather than seconds, so the consequence is a control plane that will not come up at all — and it surfaces at control-plane bring-up, long after the registration step where the wrong clock was introduced.

> **WARNING**
>
> **A Day-2 chrony `MachineConfig` does not replace this**
>
> Configuring the Chrony Time Service writes `/etc/chrony.conf` to nodes that are already running and joined. It is the correct way to point a cluster at the site NTP servers for ongoing operation, and it is a different requirement from this one: it cannot correct the clock of a host that is still installing Alauda OS or running `kubeadm`. Deferring to it leaves the install and join window unprotected.

5. TPM Decision

Set `MachineRegistration.spec.config.elemental.registration.emulate-tpm` from whether the host exposes a **real hardware TPM** (`/dev/tpm0`), not from whether it is physical or virtual:

- `emulate-tpm: false` (or omit the field) — only when the host has a working hardware TPM, so `elemental-register` can use it for `auth: tpm`.
- `emulate-tpm: true` with `emulated-tpm-seed: -1` — for any host **without** a hardware TPM. This includes both virtual machines and physical servers that ship without a TPM

module (for example a Dell R620). On such a host, `auth: tpm` combined with `emulate-tpm: false` makes `elemental-register` fail TPM attestation and never send the registration request, so `elemental-operator` records zero registration POSTs.

6. Workload Cluster Image Registry

Bringing the cluster up needs no registry credential. Bare-metal hosts install the operating system and join the cluster using the platform registry address alone, so you can reach a cluster whose Nodes are all `Ready` without configuring one.

A credential matters at the next step — installing platform components on the new cluster. Those components pull their images from a registry, and a registry that requires authentication needs credentials the cluster can use.

By default the cluster uses the registry of the `global` cluster, backed by the `public-registry-credential` Secret when that registry is authenticated. To have it pull platform component images from a dedicated registry instead, follow Choose the Image Registry for a Workload Cluster.

> **WARNING**
>
> The credential can wait, but the choice of registry cannot. A cluster is bound to its registry when it is created, and an existing cluster cannot be moved to another one. Decide before you apply the manifests below, even if you do not install platform components until later.

Neither choice affects the operating system images described above. Hosts pull `base-image` and `base-image-iso` from the platform registry during `elemental install` and every reprovision, before any cluster-level binding exists.

7. Host Disk and Boot Preparation

Classify **every** disk and virtual disk (VD) on each host before you boot the `SeedImage` ISO. Record which device is the OS install target and which devices, if any, must retain application data:

- Wipe the selected OS install disk and any other obsolete boot disks so that no bootable previous operating system remains. The host must boot the ISO, not an old on-disk OS. Set `MachineRegistration.spec.config.elemental.install.device` to the exact OS disk.
- Do **not** wipe a data disk that you intend to manage with the `Adopt` policy. Back it up independently, record its stable ID and filesystem UUID, and make sure it cannot be selected as the install device.
- A disk intended for `InitializeIfBlank` must be disposable and objectively blank. Formatting still requires a separate initialization approval after the host registers; the registration manifest does not authorize it.
- Remove residual Elemental partition labels — `COS_STATE`, `COS_PERSISTENT`, `COS_OEM`, `COS_RECOVERY` — from **all** disks, not only the intended install disk. Elemental resolves partitions by label (`blkid -L COS_STATE`); a stale label left on a second disk (for example from a previous install or a leftover multipath member) makes it resolve the wrong device and the reprovision snapshotter fails.
- On hosts where the same disk can appear through more than one path (multipath), make sure none of those paths exposes a disk that still carries a `COS_*` label; a residual label on any path can be resolved ahead of the intended install disk. Clean the disk rather than disabling multipath — the image keeps it enabled for network-attached storage boot.
- On a host with multiple disks, do not leave `install.device` empty and do not use `/dev/sda` as a persistent identity. Follow Select a Fixed System Disk on Multi-Disk Bare-Metal Hosts to reuse one ISO while selecting each host's system disk by WWN.
- Set the boot order so the host boots the virtual CD / ISO first.

Use only stable storage identities reported by the registered inventory. Linux paths such as `/dev/sdX`, `/dev/vdX`, `/dev/nvmeXnY`, and `/dev/mapper/mpathX` are runtime paths and must not be placed in `MachineInventory.spec.storage`.

8. Node Labels and Taints

To give the nodes of a pool their own role label and taint — for example to reserve dedicated pools for monitoring, logging, or ingress — declare them on the pool's resources as described in Infra Nodes. Plan them before you create the pools.

Cluster Creation Workflow

When using YAML, import the paired OS images first, then proceed through five required resource steps, with an optional storage-preparation step before pool membership. Every Kubernetes resource must be applied in the `cpaas-system` namespace.

Step	Apply or perform	Result
0	Import `base-image` and `base-image-iso` into the target platform registry and add the target Kubernetes version to `elemental-image-catalog` — see OS Images Imported and Image Catalog Populated	Both OS images are available and the catalog resolves every `Machine.spec.version` before `SeedImage` build or host reprovisioning starts.
1	`MachineRegistration` and `SeedImage`	`elemental-operator` builds the bootable ISO.
2	Boot the ISO on each physical host	The host runs `elemental install` and registers as a `MachineInventory`.
2a (optional)	Declare and prepare `MachineInventory.spec.storage`	Explicit data volumes are validated and prepared while the inventory is unallocated.
3	`MachineInventoryPool` resources, one per role	The allowed control-plane and worker inventories are declared before CAPI creates Machines.
4	`BaremetalCluster`, control-plane `BaremetalMachineTemplate`, `KubeadmControlPlane`, and `Cluster`	Cluster API starts the control-plane rollout using the control-plane pool.
5	Worker `KubeadmConfigTemplate`, worker `BaremetalMachineTemplate`, and `MachineDeployment`	Cluster API starts the worker rollout using the worker pool.

WARNING

Important Namespace Requirement

All bare-metal resources must be applied in the `cpaas-system` namespace. The provider and `elemental-operator` only reconcile objects in that namespace.

WARNING

Workload Cluster Naming

The workload `cluster-name` **must not** be `global`. That name is reserved for the `global` cluster, and reusing it causes the workload cluster's resources to collide with `global` cluster resources in `cpaas-system`. As a convention, keep the CAPI `Cluster` and `BaremetalCluster` named exactly `<cluster-name>`, and prefix dependent resources (`KubeadmControlPlane`, `KubeadmConfigTemplate`, `MachineDeployment`, machine templates, pools, registrations) with `<cluster-name>-`.

Resolving Placeholder Values

The example manifests below use `<placeholder>` syntax for environment-specific values:

Placeholder	Source of truth	How to retrieve
`<cluster-name>`	Operator-supplied workload cluster name. Must not be `global`.	Choose once and use consistently for `C` `BaremetalCluster`, pools, templates, resources.
`<kubernetes-version>`	Kubernetes version of the imported OS image, added as an `elemental-image-catalog` ConfigMap key (with leading `v`, for example `v1.33.7-2`).	OS Support Matrix for the value; `kubec` `get cm elemental-image-catalog -o` key.
`<bare-metal-os-archive>`	Bare Metal OS archive for the target ACP release.	Download page listed in OS Support Ma
`<os-image-tag>`	Tag shared by both images in the archive.	`ctr -n bare-metal-os images ls -q`

Placeholder	Source of truth	How to retrieve
`<registry-username>` / `<registry-password>`	Account that can push to `<registry-address>`.	For the `global` cluster's registry, user password from `kubectl -n cpaas-sys registry-admin -o jsonpath='{.data base64 -d`.
`<base-image>`	Imported OS image used by the image catalog for host reprovisioning.	`<registry-address>/tkestack/barem <os-image-tag>` after the import in OS and Image Catalog Populated.
`<base-image-iso>`	Imported OS ISO image paired with `<base-image>`.	`<registry-address>/tkestack/barem iso:<os-image-tag>`, imported from th `<base-image>`.
`<registry-address>`	Target platform registry into which both OS images are imported (same value as `global.registry.address` on the install chart).	`kubectl get cluster global -n cpa jsonpath='{.metadata.annotations.c address}'` (when one exists).
`<control-plane-load-balancer-type>`	Operator decision. Use `Internal` for provider-managed Alive or `External` for an existing load balancer.	Review Plan the Control Plane Endpoint
`<control-plane-vip>` / `<control-plane-port>`	Operator-supplied stable endpoint. For `Internal`, use an unused IPv4 VIP in the control-plane Layer-2 domain. For `External`, use the load balancer frontend address.	n/a
`<vrid>`	Operator-supplied VRID for `Internal` only. It must be unique in the control-plane Layer-2 domain. Remove the field for `External`.	n/a
`<install-device>`	Whole-disk target for `elemental install`. A direct kernel path is suitable only when its identity is stable and unambiguous. For multi-disk bare-metal hosts that reuse one ISO, use `/dev/elemental-install-target`.	Confirm the disk by WWN from the host a Fixed System Disk on Multi-Disk Bare-

Placeholder	Source of truth	How to retrieve
`<dns-image-tag>` / `<etcd-image-tag>`	Component versions baked into the bare-metal base image for `<kubernetes-version>`.	See the `coredns` and `etcd` columns Matrix.
`<kube-ovn-version>`	Kube-OVN chart version matching the selected ACP and Kubernetes release.	Use the **kube-ovn (chart)** value from th not the Kube-OVN component version, a the installed plugin: `kubectl get moduleplugins.cluster.alauda.io ku jsonpath='{.spec.mainChart}{" "} {.status.latestVersion}'`.
`<dns-server>`	Operator-supplied when the first-boot registration path needs an explicit resolver.	Prefer `MachineRegistration.config.` later `KubeadmControlPlane` bootstrap `cloud-config`). Do not write `/etc/res` `SeedImage.cloud-config`.
`<ssh-authorized-keys>`	Operator-supplied OpenSSH public key — required for any interactive debugging on the node.	n/a
`<inventory-name>`	Allocated by `elemental-operator` after the host boots the ISO and registers. Cannot be predicted before registration.	`kubectl -n cpaas-system get machineinventories.elemental.cattl`
`<pool-name>`	Name of the `MachineInventoryPool` that claims an inventory.	Use `<cluster-name>-control-plane-name>-worker-pool` in the examples be
`<control-plane-inventory-1/2/3>` / `<worker-inventory-1/2/3>`	Exact `MachineInventory` names selected for each role.	Use the names from `kubectl -n cpaa machineinventories.elemental.cattl`
`<control-plane-host-1/2/3>` /	Optional Kubernetes node hostnames applied during reprovision.	Operator-supplied; omit `hostname` to fa inventory name.

Placeholder	Source of truth	How to retrieve
`<worker-host-1>`		
`<pods-cidr>` / `<services-cidr>` / `<kube-ovn-join-cidr>`	Operator-supplied. Must not overlap with the host network, the `global` cluster's CIDRs, or any other CAPI cluster on the same `global`.	n/a
`<workload-kubeconfig>`	Kubeconfig for the workload cluster. Published by Cluster API as a `Secret` once the control plane initializes.	`kubectl -n cpaas-system get secre kubeconfig -o jsonpath='{.data.val`
`<machine-name>`	Existing CAPI `Machine` name used for targeted removal or recovery.	`kubectl -n cpaas-system get machi k8s.io`
`<new-replica-count>`	Desired `MachineDeployment.spec.replicas` value after scaling.	Operator-supplied target worker count.
`<new-template-name>`	Name of the new template for a changed `BaremetalMachineTemplate` or `KubeadmConfigTemplate`.	Operator-supplied; use a new name.
`<new-kubernetes-version>` / `<new-tag>` / `<new-image>`	Target upgrade version and matching bare-metal base-image tag or resolved image.	Use the release support matrix and conf exists in `elemental-image-catalog`.

Step 1: Build the SeedImage and Register Hosts

Create a `MachineRegistration` that describes the registration URL and first-install cloud-config, and a `SeedImage` that points `elemental-operator` at the matching ISO base image.

Set `SeedImage.spec.baseImage` to the final target-platform registry reference of the imported `base-image-iso`. It must be paired with the imported `base-image` used by the image catalog for later reprovisioning: take both images from the same OS archive, never from different archives.

01-machineregistration-seedimage.yaml

```
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: <cluster-name>-registration
  namespace: cpaas-system
  labels:
    app.kubernetes.io/part-of: cluster-api-provider-baremetal
spec:
  # ${...} placeholders are expanded by elemental-register from SMBIOS da
ta.
  # System UUID is preferred over Serial Number because it is more reliab
le
  # across virtualised test environments.
  machineName: "<cluster-name>-${System Information/UUID}"
  machineInventoryLabels:
    app.kubernetes.io/part-of: cluster-api-provider-baremetal
    pool.baremetal.alauda.io/eligible: "true"
    elemental.cattle.io/serial-number: "${System Information/Serial Numbe
r}"
  machineInventoryAnnotations:
    elemental.cattle.io/machine-uuid: "${System Information/UUID}"
  config:
    elemental:
      install:
        device: <install-device>
        eject-cd: true
        reboot: true
        snapshotter:
          type: btrfs
          maxSnaps: 4
      registration:
        # false only when the host has a real hardware TPM (/dev/tpm0). S
et true
        # (with emulated-tpm-seed: -1) on any host without one, including
physical
        # servers that ship without a TPM module.
        emulate-tpm: false
---
apiVersion: elemental.cattle.io/v1beta1
kind: SeedImage
metadata:
  name: <cluster-name>-registration-iso
  namespace: cpaas-system
```

```
  labels:
    app.kubernetes.io/part-of: cluster-api-provider-baremetal
spec:
  type: iso
  baseImage: <base-image-iso>          # Use the imported target-platform
registry reference.
  registrationRef:
    apiVersion: elemental.cattle.io/v1beta1
    kind: MachineRegistration
    name: <cluster-name>-registration
    namespace: cpaas-system
  targetPlatform: linux/amd64
  size: 20Gi
  cleanupAfterMinutes: 120
  cloud-config:
    stages:
      boot:
        - name: "Size COS_STATE for reprovisioning"
          files:
            - path: /etc/elemental/config.d/partitions.yaml
              permissions: 0644
              content: |
                install:
                  partitions:
                    state:
                      size: 20480
```

Size COS_STATE before the first install

The `state` partition holds the running system plus every retained snapshot. The Elemental default of 8192 MiB is not enough: snapshots of one image share extents, but an upgrade to a *different* image has to hold two unrelated systems of roughly 3.5 GiB at once and fails part-way through with `no space left on device`, leaving a partial snapshot behind on every retry. Every cluster upgrade goes through `elemental upgrade`, and partition sizes are fixed at install time, so this only surfaces later — on a cluster already in production that can no longer be repartitioned without a reinstall.

Size it to at least 20480 MiB (20 GiB) at install time.

> **Placement and stage both matter**

The fragment must sit in `SeedImage.spec.cloud-config` under `stages.boot`. Putting it in `MachineRegistration.spec.config.cloud-config`, or under any other stage, leaves an 8 GiB partition and reports no error.

Verify the result on the first installed host before rolling out the rest of the fleet:

```
# Expect ~20G, not 8G
lsblk -o NAME,SIZE,LABEL | grep COS_STATE
df -h /
```

Do not add `/etc/resolv.conf` to `SeedImage.spec.cloud-config`. Keep the ISO generic, and put site-specific resolver configuration in `MachineRegistration.spec.config.cloud-config` only when the first-boot registration path needs it. During the tested Global deployment flow, the ISO did not carry resolver files; node DNS was configured later by the kubeadm bootstrap data.

`install.device`, `install.eject-cd`, and `install.reboot` are intentional. If the target disk is omitted, `elemental install` can select an unintended device. On multi-disk physical hosts, use the fixed system disk workflow instead of a changing `/dev/sdX` name. If `eject-cd` or `reboot` is false, a host can remain in the live environment after the first install and never become usable inventory for Cluster API.

Apply the manifest and wait for the SeedImage build to finish:

```
kubectl apply -f 01-machineregistration-seedimage.yaml
kubectl -n cpaas-system wait \
  --for=condition=SeedImageReady=True \
  seedimage/<cluster-name>-registration-iso \
  --timeout=30m

SEEDIMAGE_REASON="$(kubectl -n cpaas-system \
  get seedimage <cluster-name>-registration-iso \
  -o jsonpath='{.status.conditions[?(@.type=="SeedImageReady")].reaso
n}')"
SEEDIMAGE_DOWNLOAD_URL="$(kubectl -n cpaas-system \
  get seedimage <cluster-name>-registration-iso \
  -o jsonpath='{.status.downloadURL}')"
SEEDIMAGE_CHECKSUM_URL="$(kubectl -n cpaas-system \
  get seedimage <cluster-name>-registration-iso \
  -o jsonpath='{.status.checksumURL}')"

if [ "${SEEDIMAGE_REASON}" != "SeedImageBuildSuccess" ] || \
   [ -z "${SEEDIMAGE_DOWNLOAD_URL}" ] || \
   [ -z "${SEEDIMAGE_CHECKSUM_URL}" ]; then
  echo "SeedImage did not produce a downloadable ISO and checksum" >&2
  exit 1
fi

printf '%s\n%s\n' "${SEEDIMAGE_DOWNLOAD_URL}" "${SEEDIMAGE_CHECKSUM_URL}"
```

Do not use `status.state` as the success gate; the current controller does not populate it. Continue only when `SeedImageReady=True` has reason `SeedImageBuildSuccess` and both `status.downloadURL` and `status.checksumURL` are non-empty. A SeedImage whose download lifetime has expired can still report `SeedImageReady=True`, but its reason changes and its URLs are cleared.

Boot every target host from this ISO. `elemental-register` runs first (creates the `MachineInventory` and uploads `observedNetwork`), then `elemental install` writes the on-disk OS. After install completes, the host stays available for plan execution. If the live ISO environment has no DHCP address, configure NetworkManager manually on the host console as described in Network Connectivity before waiting for the `MachineInventory`.

Confirm registration:

```
kubectl -n cpaas-system get machineinventories.elemental.cattle.io
```

Every inventory you intend to use must:

- Show `Ready=True`.
- Have a non-empty `status.plan.secretRef.name`.
- Have a `spec.observedNetwork` that matches the host's expected NIC (only required when you want the install-time IP to survive across reprovisions).
- When managed data disks are required, run an observer-capable OS image, report a fresh `status.observedStorage`, and reach `status.storage.phase=Prepared` before pool allocation.

Record the exact `MachineInventory` names — they are referenced by name in the next step.

Optional: Prepare Managed Data Disks

Managed storage belongs to the long-lived `MachineInventory`, not to a CAPI `Machine` or `BaremetalMachineTemplate`. Configure it while the inventory is unallocated and before adding that inventory to a production pool.

Follow Manage Data Disks on Bare-Metal Hosts to:

1. Verify `elemental-storage-observer.service` and inspect `status.observedStorage`.
2. Select each device by a stable ID and choose `Adopt` or `InitializeIfBlank`.
3. Use the matching `storagectl` binary to validate and atomically patch the declaration and, when required, its initialization approval.
4. Wait for `StoragePrepared=True/AllRequiredVolumesPrepared` and `status.storage.phase=Prepared`.

If no disks should be managed, leave `spec.storage` absent or set `volumes: []`. The storage controller converges that inventory to `Unmanaged` with `StoragePrepared=True/NoManagedVolumes` and performs no disk operation.

Step 2: Create `MachineInventoryPool` Resources

Create one pool per role. The pool reconciler validates that every member exists, computes capacity counters, and writes the `baremetal.alauda.io/pool=<pool-name>` annotation onto the inventory.

02-machineinventorypool.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: MachineInventoryPool
metadata:
  name: <cluster-name>-control-plane-pool
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  machineInventories:
    - name: <control-plane-inventory-1>
      hostname: <control-plane-host-1>    # Optional; falls back to the i
nventory name.
    - name: <control-plane-inventory-2>
      hostname: <control-plane-host-2>
    - name: <control-plane-inventory-3>
      hostname: <control-plane-host-3>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: MachineInventoryPool
metadata:
  name: <cluster-name>-worker-pool
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  machineInventories:
    - name: <worker-inventory-1>
      hostname: <worker-host-1>
    - name: <worker-inventory-2>
    - name: <worker-inventory-3>
```

Key parameters:

Parameter	Type	Description	Require
`.spec.clusterName`	string	The workload cluster name this pool serves. A `MachineInventory` must not appear in two active pools.	Yes
`.spec.machineInventories[].name`	string	Exact name of a registered `MachineInventory`.	Yes
`.spec.machineInventories[].hostname`	string	Hostname applied during reprovision. Defaults to the inventory name when omitted.	No

Apply and verify:

```
kubectl apply -f 02-machineinventorypool.yaml
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.
x-k8s.io
```

A healthy pool reports `Ready=True`, `total = len(spec.machineInventories)`, and `available = total - allocated - preparing - reprovisioning - unavailable`. Inventories listed in `spec.machineInventories` that fail validation (missing, plan secret missing, Ready=False, or storage not prepared) raise the pool's `unavailable` counter and surface in the `MembersValid` condition. A non-empty storage declaration is an allocation gate: the inventory is not `Available` to CAPI until Prepare succeeds against a fresh observation.

Size the control-plane pool to at least `KubeadmControlPlane.spec.replicas`. Size the worker pool to at least `MachineDeployment.spec.replicas`. For rolling upgrades the pool must hold the entire replica count — the provider uses delete-then-add semantics from the same pool, never both at once.

Step 3: Create the Control-Plane Cluster Resources

Create the `BaremetalCluster` (declares the selected control-plane endpoint mode), the control-plane `BaremetalMachineTemplate` (points at the control-plane pool), the `KubeadmControlPlane` (replicas + kubeadm config), and the CAPI `Cluster`.

The Bare Metal API supports both endpoint modes in the same workflow:

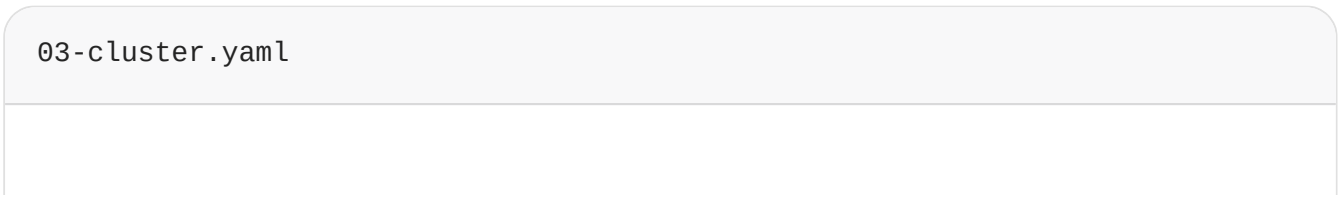
- `Internal`: the provider deploys Alive and reconciles the VIP and control-plane backend membership.
- `External`: the provider skips Alive; the load balancer owner maintains the listener, health check, and backend membership.

Select the mode with `<control-plane-load-balancer-type>` in the manifest below.

> **WARNING**
>
> **Use the Kubernetes images built into the OS image**
>
> The supported bare-metal OS image preloads the kubeadm control-plane, CoreDNS, and etcd images under `cloud.alauda.io/alauda`. Keep that repository in the KCP configuration and use the component tags from the OS Support Matrix. Do not replace it with `<registry-address>/tkestack`: that produces image references that do not match the images preloaded in the OS. The OS configures its built-in pause image through containerd; do not add `pod-infra-container-image` to the KCP kubelet arguments.

> **TIP**
>
> **Full Configuration Reference**
>
> The example below uses a minimal `KubeadmControlPlane`. For the full hardening profile recommended in production — admission, audit, kubelet patches, encryption provider — see Complete KubeadmControlPlane Configuration in the Appendix.

`03-cluster.yaml`



```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalCluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    type: <control-plane-load-balancer-type>
    host: <control-plane-vip>
    port: <control-plane-port>
    # Required only for Internal. Remove this field for External.
    vrid: <vrid>
  networkType: kube-ovn      # Enables provider-managed Kube-OVN AppRelea
se reconciliation.
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalMachineTemplate
metadata:
  name: <cluster-name>-control-plane-template
  namespace: cpaas-system
spec:
  template:
    spec:
      machineInventoryPoolRef:
        name: <cluster-name>-control-plane-pool
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-control-plane
  namespace: cpaas-system
spec:
  replicas: 3
  version: <kubernetes-version>
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0            # Bare-metal does not over-provision; replac
e one at a time.
  machineTemplate:
    nodeDrainTimeout: 5m
    nodeVolumeDetachTimeout: 5m
    infrastructureRef:
```

```
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: BaremetalMachineTemplate
      name: <cluster-name>-control-plane-template
  kubeadmConfigSpec:
    users:
      - name: boot
        sudo: ALL=(ALL) NOPASSWD:ALL
        shell: /bin/bash
        sshAuthorizedKeys:
          - "<ssh-authorized-keys>"
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
          imageTag: <etcd-image-tag>
      apiServer:
        extraArgs:
          profiling: "false"
          tls-min-version: VersionTLS12
      controllerManager:
        extraArgs:
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          profiling: "false"
          tls-min-version: VersionTLS12
    initConfiguration:
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
    joinConfiguration:
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
  annotations:
```

```
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.i
o/v1beta1
    capi.cpaas.io/resource-kind: BaremetalCluster
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>
    cpaas.io/kube-ovn-version: <kube-ovn-version>
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
  labels:
    cluster-type: ProviderBaremetal
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - <pods-cidr>
    services:
      cidrBlocks:
        - <services-cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster-name>-control-plane
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: BaremetalCluster
    name: <cluster-name>
```

Cluster annotations. The bare-metal provider relies on a small set of `Cluster` annotations during reconcile. Authoritative ones the operator must set:

Annotation	Required	Value source	Purpose
`capi.cpaas.io/resource-group-version`	Yes	Literal `infrastructure.cluster.x-k8s.io/v1beta1`	CAPI infrastructure binding.
`capi.cpaas.io/resource-kind`	Yes	Literal `BaremetalCluster`	CAPI infrastructure binding.

Annotation	Required	Value source	Purpose
`cpaas.io/kube-ovn-join-cidr`	Yes	Operator-chosen `/16` CIDR that does not overlap with pods/services or any other cluster.	Kube-OVN inter-node tunnel.
`cpaas.io/kube-ovn-version`	Recommended	**kube-ovn (chart)** value from the OS Support Matrix.	Pins the chart version used when the provider create `cni-kube-ovn`. If omitted the provider falls back to th `kube-ovn` `ModulePlugi` version.
`cpaas.io/sentry-deploy-type`	Yes	Literal `Baremetal`.	Marks the cluster for the bare-metal deploy profile.
`cpaas.io/alb-address-type`	Yes	Literal `ClusterAddress`.	ALB address mode used on bare-metal clusters.

`BaremetalCluster` parameters:

Parameter	Type	Description
`.spec.controlPlaneLoadBalancer.type`	string (`Internal` / `External`)	`Internal` deploys Alive static poc and manages the VIP. `External` skips Alive and requires a pre-provisioned Layer 4 TCP LoadBalancer.
`.spec.controlPlaneLoadBalancer.host`	string	Control-plane endpoint. For `Internal`, use the Self-built VIP. `External`, use the load balancer

Parameter	Type	Description
		frontend address. Backfilled into `.spec.controlPlaneEndpoint.ho` when the endpoint is left empty.
`.spec.controlPlaneLoadBalancer.port`	int (1–65535)	Control-plane port. Typically `6443`
`.spec.controlPlaneLoadBalancer.vrid`	int (0–255)	Keepalived `virtual_router_id`. Must be unique in the control-plane Layer-2 domain. Remove the field f `External`.
`.spec.controlPlaneEndpoint`	object	API server endpoint exposed to CA Once set, must not change. Leave empty to let the reconciler backfill it from `controlPlaneLoadBalancer`
`.spec.networkType`	string	CNI implementation managed by th provider. Set it to `kube-ovn`; any other value causes the provider to s Kube-OVN `AppRelease` reconciliation and node-removal integration.

Apply and watch:

```
kubectl apply -f 03-cluster.yaml
kubectl -n cpaas-system get clusters.cluster.x-k8s.io
kubectl -n cpaas-system get kubeadmcontrolplanes.controlplane.cluster.x-k
8s.io
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io -w
```

Each new control-plane `BaremetalMachine` advances `Pending → Allocated → Reprovisioning → Running`. Watch:

- `BaremetalMachine.status.machineInventoryRef.name` — which inventory was picked.
- `BaremetalMachine.status.planSecretRef.name` — plan secret being driven. The secret carries the annotation `baremetal.alauda.io/plan.type=reprovision`.

- `MachineInventory.status.plan.state` — `Applied` once the host completes `cloud-init clean`, `elemental upgrade`, reboot, and `kubeadm init` / `join`.
- `BaremetalCluster.status.conditions[EndpointReady]` — true once the configured control-plane endpoint is reachable.

> **WARNING**
>
> The bare-metal provider does not support single-node control planes. Provision at least three control-plane replicas (`KubeadmControlPlane.spec.replicas: 3`) so that etcd retains quorum. In `Internal` mode, the same nodes also participate in Alive VIP arbitration.

Step 4: Deploy Worker Nodes

After the control plane is Ready, create the worker `BaremetalMachineTemplate`, the worker `KubeadmConfigTemplate`, and the `MachineDeployment`.

Confirm the Worker Pool Capacity

```
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.x-k8s.io <cluster-name>-worker-pool
```

`status.available` must be at least the target replica count. If you need additional capacity, register more hosts (boot the `SeedImage` ISO on them — see Step 1), prepare any managed storage while each inventory is still unallocated, and then add their names to `spec.machineInventories`.

A host added this way goes through the same first boot as the original nodes, so it has to meet the same prerequisites: the network paths in Network Connectivity and the clock gate in Host Time Synchronization. Confirm both for the new host rather than assuming the approval obtained at delivery time covers its subnet and its hardware clock.

Configure the Worker `BaremetalMachineTemplate`

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: BaremetalMachineTemplate
metadata:
  name: <cluster-name>-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      machineInventoryPoolRef:
        name: <cluster-name>-worker-pool
```

Key parameters:

Parameter	Type	Description
`.spec.template.spec.machineInventoryPoolRef.name`	string	Worker pool name (same namespace). To move the workers to a different pool, see Moving Workers to a Different Pool.

Configure the Bootstrap Template

`imagePullCredentialsVerificationPolicy: NeverVerify` is required only starting with Kubernetes 1.35. Omit this parameter when creating workers with Kubernetes 1.34 or earlier.

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster-name>-worker-bootstrap
  namespace: cpaas-system
spec:
  template:
    spec:
      users:
        - name: boot
          sudo: ALL=(ALL) NOPASSWD:ALL
          shell: /bin/bash
          sshAuthorizedKeys:
            - "<ssh-authorized-keys>"
      files:
        - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.j
son
          owner: "root:root"
          permissions: "0644"
          content: |
            {
              "apiVersion": "kubelet.config.k8s.io/v1beta1",
              "kind": "KubeletConfiguration",
              "imagePullCredentialsVerificationPolicy": "NeverVerify",
              "protectKernelDefaults": true,
              "staticPodPath": null,
              "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
              "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
              "streamingConnectionIdleTimeout": "5m",
              "clientCAFile": "/etc/kubernetes/pki/ca.crt"
            }
      joinConfiguration:
        patches:
          directory: /etc/kubernetes/patches
```

The provider applies a minimal normalization to bootstrap user-data before writing it into the `reprovision` plan, so leave the following fields out of the template:

- **Hostname / FQDN** — set automatically from `MachineInventoryPool.spec.machineInventories[].hostname` (or the inventory name when omitted).

- **`kubeletExtraArgs.provider-id`** — set automatically to `baremetal:///<inventory-name>`.
- **`nodeRegistration.criSocket`** — set automatically to `unix:///var/run/containerd/containerd.sock` when unset.

The normalized user-data is also written back into the bootstrap secret under `data["resolved-value"]` for debugging; the original `data["value"]` is left untouched.

Configure the `MachineDeployment`

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster-name>-workers
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0        # Bare-metal pools cannot over-provision.
      maxUnavailable: 1
  selector:
    matchLabels: {}
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: <cluster-name>
        pool.name: <cluster-name>-workers
    spec:
      clusterName: <cluster-name>
      version: <kubernetes-version>
      nodeDrainTimeout: 5m
      nodeVolumeDetachTimeout: 5m
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster-name>-worker-bootstrap
          namespace: cpaas-system
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: BaremetalMachineTemplate
        name: <cluster-name>-worker-template
        namespace: cpaas-system
```

Key parameters:

Parameter	Type	Description
`.spec.clusterName`	string	Target cluster name.
`.spec.replicas`	int	Number of worker nodes. `replicas ≤ MachineInventoryPool.s + status.allocated` (fo pool).
`.spec.template.spec.version`	string	Worker Kubernetes versio in `elemental-image-cat` from the control-plane ver respecting the standard k policy.
`.spec.strategy.rollingUpdate.maxSurge`	int	Bare-metal does not over- `0`. The provider has no s host to bring up an extra r
`.spec.strategy.rollingUpdate.maxUnavailable`	int	Must be `> 0` when `max` Workers are recycled one this budget.

Apply and watch:

```
kubectl apply -f workers.yaml
kubectl -n cpaas-system get machinedeployments.cluster.x-k8s.io
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io -w
kubectl get nodes -o wide                    # workload cluster
```

Cluster Verification

Using kubectl

```
# Cluster status
kubectl -n cpaas-system get cluster <cluster-name>

# Control plane progress
kubectl -n cpaas-system get kubeadmcontrolplane <cluster-name>-control-pl
ane

# Per-machine state
kubectl -n cpaas-system get machines.cluster.x-k8s.io
kubectl -n cpaas-system get baremetalmachines.infrastructure.cluster.x-k8
s.io

# Pool counters
kubectl -n cpaas-system get machineinventorypools.infrastructure.cluster.
x-k8s.io

# Storage lifecycle for inventories that declare managed volumes
kubectl -n cpaas-system get machineinventories.elemental.cattle.io \
  -o custom-columns='NAME:.metadata.name,PHASE:.status.storage.phase,PREP
ARED:.status.storage.conditions[?(@.type=="StoragePrepared")].status,ACTI
VE:.status.storage.conditions[?(@.type=="StorageActive")].status'

# Workload cluster Nodes (use the workload kubeconfig)
kubectl get nodes -o wide
```

Verify the Control Plane Endpoint

Confirm the configured mode and endpoint:

```
kubectl -n cpaas-system get baremetalcluster <cluster-name> \
  -o jsonpath='{.spec.controlPlaneLoadBalancer.type}{" "}{.spec.controlPl
aneEndpoint.host}{":"}{.spec.controlPlaneEndpoint.port}{"\n"}'
```

For `Internal`, verify Alive and the node prerequisites:

```
kubectl --kubeconfig <workload-kubeconfig> -n kube-system get pods -l app
=alive -o wide
kubectl --kubeconfig <workload-kubeconfig> get --raw=/version
```

Run the following command on each control-plane node, not on the management cluster:

```
sysctl net.ipv4.conf.all.arp_accept net.ipv4.vs.conntrack
```

For `External`, verify the load balancer frontend and each backend according to the External LoadBalancer contract. Confirm that the load balancer owner has registered every current control-plane node and that HTTPS `/healthz` returns HTTP `200` for each backend.

Expected Results

A successfully created cluster shows:

- `Cluster.status.conditions[Ready]=True`.
- `KubeadmControlPlane` replicas all `Ready`.
- Every `BaremetalMachine.status.phase=Running` and `status.ready=true`.
- Every used `MachineInventory.status.plan.state=Applied` with the annotation `baremetal.alauda.io/plan.type=reprovision` on its plan secret (an annotation, not a label — `-o jsonpath='{.metadata.annotations}'`).
- Every used inventory with non-empty `spec.storage.volumes[]` reports `status.storage.phase=Active`, `StoragePrepared=True`, and `StorageActive=True`; the corresponding `BaremetalMachine` reports `StorageReady=True/AllVolumesReady`.
- `MachineInventoryPool.status` satisfies `available + allocated + preparing + reprovisioning + unavailable = total`.
- Kubernetes Nodes Ready.
- For `Internal`, Alive Pods are Ready and the API is reachable through the Self-built VIP.
- For `External`, the load balancer frontend is reachable and its backend list matches the current control-plane nodes.

Common Failure Modes

Symptom	Likely cause
`BaremetalMachine` stuck in `Pending`, `InventoryAllocated=False / Reason=PoolMissing`	`BaremetalMachineTemplate.spec.template.sp` points at a non-existent pool.
`BaremetalMachine` stuck in `Pending`, `InventoryAllocated=False / Reason=PoolExhausted`	Pool has no `Available` inventory left.
`BaremetalMachine` stuck after allocation, `BootstrapReady=False / Reason=BootstrapWaiting`	KubeadmConfig has not produced a bootstrap da
`BaremetalMachine` in `Failed`, `ImageResolved=False / Reason=ImageCatalogMiss`	The target `Machine.spec.version` is not a key chart creates the catalog with no entries.
`ImageResolved=False / Reason=ImageRegistryMissing`	Neither the optional `cpaas.io/registry-addre` registry-credential Secret resolves a registry addr
Inventory is registered but not counted as `Available`	Its managed storage declaration has not reached storage observation is missing or stale.
`BaremetalMachine` waits with `StorageReady=False`	A required managed volume did not activate or n confirms its mount.
Reprovision never completes; `MachineInventory.status.plan.state=Failed`	`elemental upgrade` failed (registry unreachabl
`BaremetalCluster` uses `Internal`, but the VIP is not reachable	`vrid` collides with another cluster; VIP is not in gratuitous ARP is blocked; IPVS or required sysc
`BaremetalCluster` uses `External`, but the endpoint is not reachable	Listener, backend membership, TLS passthrough certificate SAN is incorrect.

For the full operator-side state machine reference (every condition reason and recovery action), see Provider Overview → clean / reprovision plans.

Next Steps

After creating a cluster:

- Manage Nodes
- Manage Data Disks
- Upgrade Cluster

Appendix

Complete `KubeadmControlPlane` Configuration

The hardened configuration recommended for production bare-metal clusters — admission control, audit policy, kubelet patches, encryption provider, and IPv6 bind addresses. Substitute the placeholders from the table in Resolving Placeholder Values.

> **Kubernetes 1.35 kubelet settings**
>
> `imagePullCredentialsVerificationPolicy: NeverVerify` is required only starting with Kubernetes 1.35. Omit this parameter when creating a cluster with Kubernetes 1.34 or earlier.

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-control-plane
  namespace: cpaas-system
spec:
  replicas: 3
  version: <kubernetes-version>
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  machineTemplate:
    nodeDrainTimeout: 5m
    nodeVolumeDetachTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: BaremetalMachineTemplate
      name: <cluster-name>-control-plane-template
  kubeadmConfigSpec:
    users:
      - name: boot
        sudo: ALL=(ALL) NOPASSWD:ALL
        shell: /bin/bash
        sshAuthorizedKeys:
          - "<ssh-authorized-keys>"
    files:
      - path: /etc/kubernetes/admission/psa-config.yaml
        owner: "root:root"
        permissions: "0644"
        content: |
          apiVersion: apiserver.config.k8s.io/v1
          kind: AdmissionConfiguration
          plugins:
          - name: PodSecurity
            configuration:
              apiVersion: pod-security.admission.config.k8s.io/v1
              kind: PodSecurityConfiguration
              defaults:
                enforce: "privileged"
                enforce-version: "latest"
                audit: "baseline"
                audit-version: "latest"
```

```
                warn: "baseline"
                warn-version: "latest"
              exemptions:
                usernames: []
                runtimeClasses: []
                namespaces:
                - kube-system
                - cpaas-system
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
        owner: "root:root"
        permissions: "0644"
        content: |
          {
            "apiVersion": "kubelet.config.k8s.io/v1beta1",
            "kind": "KubeletConfiguration",
            "imagePullCredentialsVerificationPolicy": "NeverVerify",
            "protectKernelDefaults": true,
            "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
            "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
            "streamingConnectionIdleTimeout": "5m",
            "clientCAFile": "/etc/kubernetes/pki/ca.crt"
          }
      - path: /etc/kubernetes/audit/policy.yaml
        owner: "root:root"
        permissions: "0644"
        content: |
          apiVersion: audit.k8s.io/v1
          kind: Policy
          omitStages:
          - "RequestReceived"
          rules:
          - level: None
            users:
            - system:kube-controller-manager
            - system:kube-scheduler
            - system:serviceaccount:kube-system:endpoint-controller
            verbs: ["get", "update"]
            namespaces: ["kube-system"]
            resources:
            - group: ""
              resources: ["endpoints"]
          - level: None
            nonResourceURLs:
```

```
            - /healthz*
            - /version
            - /swagger*
          - level: None
            resources:
            - group: ""
              resources: ["events"]
          - level: None
            verbs: ["get", "list", "watch"]
          - level: None
            resources:
            - group: "coordination.k8s.io"
              resources: ["leases"]
          - level: None
            resources:
            - group: "authorization.k8s.io"
              resources: ["subjectaccessreviews", "selfsubjectaccessrevie
ws"]
            - group: "authentication.k8s.io"
              resources: ["tokenreviews"]
          - level: Metadata
            resources:
            - group: ""
              resources: ["secrets", "configmaps"]
          - level: RequestResponse
            resources:
            - group: ""
            - group: "apps"
            - group: "rbac.authorization.k8s.io"
            - group: "storage.k8s.io"
            - group: "networking.k8s.io"
          - level: Metadata
    postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
          imageTag: <etcd-image-tag>
      apiServer:
        extraArgs:
          audit-log-format: json
```

```
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "200"
          profiling: "false"
          audit-log-mode: batch
          audit-log-path: /etc/kubernetes/audit/audit.log
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          tls-cipher-suites: "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS
_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY130
5,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"
          admission-control-config-file: /etc/kubernetes/admission/psa-co
nfig.yaml
          tls-min-version: VersionTLS12
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
        extraVolumes:
        - name: vol-dir-0
          hostPath: /etc/kubernetes
          mountPath: /etc/kubernetes
          pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          bind-address: "::"
          tls-min-version: VersionTLS12
          profiling: "false"
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
```

Worker bootstrap is symmetric — see Configure the Bootstrap Template for the worker `KubeadmConfigTemplate`.

Select a Fixed System Disk on Multi-Disk Bare-Metal Hosts

Use this guide when one `SeedImage` ISO must be reused across multiple bare-metal hosts and each host has more than one disk. The procedure prevents Elemental from automatically choosing a different disk on each boot and lets an operator select the system disk on each host by its stable WWN.

TOC

- Why a Manual Disk Gate Is Required
- Prerequisites
- Configure the Shared MachineRegistration
- Prepare Each Host
  1. Boot the Expected Live ISO and Block Automatic Installation
  2. Confirm That the Host Is Running from the Live ISO
  3. Configure the Live ISO Network When DHCP Is Unavailable
  4. Identify All Disks and the Intended System Disk
  5. Remove Stale Elemental Labels When Reinstalling a Host
  6. Create the Per-Host Install Target
  7. Run the Final Gate and Start Installation
- Verify the Installed Host
- Troubleshooting
- Limitations

Why a Manual Disk Gate Is Required

Do not leave `MachineRegistration.spec.config.elemental.install.device` empty on a multi-disk host. An empty device and an empty `device-selector` do not make Elemental wait for an operator to choose a disk. Multiple discovered disks can match, and one of them is selected without a uniqueness check.

Do not use `/dev/sda` as a persistent disk identity either. Storage-controller discovery order can change between the Live ISO and the installed system, so the same logical disk can appear as `/dev/sda` during one boot and `/dev/sdb` during another.

This guide uses a fail-closed manual gate:

1. Every reusable ISO contains the same, initially nonexistent install path: `/dev/elemental-install-target`.
2. The install service cannot proceed while that path does not exist, and it does not fall back to automatic disk selection.
3. On each host, an operator identifies the intended whole disk by `/dev/disk/by-id/wwn-*` and creates the gate as a symbolic link to that disk.
4. The operator starts the install service only after the disk, network, mount, and stale-label checks pass.

> **WARNING**
>
> The cleanup procedure in this guide permanently destroys data. Run it only from the Live ISO console, only on disks approved for reinstallation, and only after verifying that none of those disks is mounted.

Prerequisites

- The bare-metal provider and `elemental-operator` are installed on the `global` cluster.

- You can create or edit `MachineRegistration` and `SeedImage` resources in `cpaas-system`.
- You have console access to each host through BMC, iDRAC, iLO, or an equivalent interface.
- Each intended system disk has a stable `/dev/disk/by-id/wwn-*` entry.
- You know which disks may be erased. If the host contains data disks that must be preserved, exclude them from every cleanup command.
- If the Live ISO network has no DHCP service, you have the host IP address, prefix, gateway, DNS server, and Global registration URL.

See Create a Cluster on Bare Metal for the complete provider and cluster creation workflow.

Configure the Shared MachineRegistration

Set the install device to `/dev/elemental-install-target`. Do not configure a `device-selector` for this workflow.

```
MachineRegistration install settings
```

```yaml
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: <cluster-name>-registration
  namespace: cpaas-system
spec:
  machineName: "<cluster-name>-${System Information/UUID}"
  config:
    elemental:
      install:
        device: /dev/elemental-install-target
        eject-cd: true
        reboot: true
        snapshotter:
          type: btrfs
          maxSnaps: 4
      registration:
        auth: tpm
        # Use hardware TPM when available. For hosts without TPM, enable
        # emulation and derive a unique identity for every host.
        emulate-tpm: true
        emulated-tpm-seed: -1
```

The path is intentionally absent when the Live ISO starts. A filesystem label, GPT `PARTLABEL`, or `/dev/disk/by-label/*` path is not an alternative: these identify partitions, while Elemental requires a whole-disk install target.

Create a `SeedImage` that references this MachineRegistration, or rebuild the existing SeedImage after changing the install device. That `SeedImage` must also carry the `state` partition sizing described in Size COS_STATE before the first install — the Elemental default of 8 GiB is too small once a host is upgraded to a different image, and the size cannot be changed without reinstalling the host.

WARNING

The MachineRegistration settings are embedded in the generated ISO. Patching the MachineRegistration does not change an ISO that was already built and downloaded. Rebuild the SeedImage, wait until it is ready, download the new ISO, and verify its checksum before booting hosts from it.

Prepare Each Host

Perform the following steps on one host at a time. Do not prepare multiple hosts in parallel when disk cleanup is required.

1. Boot the Expected Live ISO and Block Automatic Installation

Boot the host from the shared ISO. As soon as the Live ISO console is available, stop and mask the automatic installer:

```
systemctl stop elemental-register-install.service
systemctl mask elemental-register-install.service
systemctl reset-failed elemental-register-install.service
```

Confirm that the ISO contains the manual gate:

```
grep -n -C 3 '/dev/elemental-install-target' \
  /run/initramfs/live/livecd-cloud-config.yaml
```

If the path is not present, the host did not boot the expected ISO. Do not continue.

2. Confirm That the Host Is Running from the Live ISO

```
findmnt -n -o SOURCE,FSTYPE,TARGET /
findmnt /run/initramfs/live
lsblk -e7 -o NAME,PATH,TYPE,SIZE,MODEL,SERIAL,WWN,FSTYPE,LABEL,MOUNTPOINT
S
```

Confirm all of the following:

- `/` is the Live root, such as `LiveOS_rootfs`.
- The virtual CD device, normally `/dev/sr0`, has the `COS_LIVE` label.
- No disk that you plan to clean or use as the install target is mounted.

Check for physical-disk, device-mapper, swap, LVM, and multipath use:

```
findmnt -rn -o SOURCE,TARGET | grep -E '^/dev/(sd|mapper/)' || \
  echo NO_PHYSICAL_DISK_MOUNTS
swapon --show
pvs 2>/dev/null || true
vgs 2>/dev/null || true
multipath -ll 2>/dev/null || true
```

Stop if a target disk is mounted or if you cannot prove that the root filesystem belongs to the Live ISO.

3. Configure the Live ISO Network When DHCP Is Unavailable

First identify the actual NetworkManager connection and interface. Do not assume the connection is always named `Wired connection 1`.

```
nmcli device status
nmcli connection show
ip -br link
```

Configure the selected connection with the values assigned to this host:

```
nmcli connection modify "Wired connection 1" \
  ipv4.method manual \
  ipv4.addresses 192.168.254.76/24 \
  ipv4.gateway 192.168.254.1 \
  ipv4.dns 192.168.16.19 \
  ipv4.ignore-auto-dns yes

nmcli connection up "Wired connection 1"
```

Verify the address, default route, DNS resolution, and the complete registration URL:

```
ip -br addr
ip route
cat /etc/resolv.conf
ping -c 2 -W 2 192.168.254.1
getent hosts <global-domain>
curl -ksS --connect-timeout 10 \
  -o /dev/null -w 'REG_HTTP=%{http_code}\n' \
  'https://<global-domain>/<registration-path-and-token>'
```

Use the site-specific network values instead of copying the example addresses. Keep the full registration URL and token out of source control and shared logs.

This network configuration only connects the Live ISO for registration and initial installation. It does not replace the persistent networking applied to the installed system during Cluster API reprovisioning.

4. Identify All Disks and the Intended System Disk

List stable disk attributes:

```
lsblk -d -e7 -o NAME,PATH,TYPE,SIZE,MODEL,SERIAL,WWN,HCTL

for current_disk in /dev/sda /dev/sdb /dev/sdc /dev/sdd; do
  echo "=== ${current_disk} ==="
  udevadm info -q symlink -n "${current_disk}" | tr ' ' '\n' |
    grep -E 'disk/by-id/(wwn|scsi)-' || true
done
```

Adjust the disk list for the current host. Record and cross-check:

- Current kernel device name.
- Size, model, serial number, and controller address (`HCTL`).
- `/dev/disk/by-id/wwn-*` path.
- Which whole disk is approved as the system disk.

Use the WWN plus the size, model, and controller information to make the decision. Do not select a disk by kernel name or capacity alone.

5. Remove Stale Elemental Labels When Reinstalling a Host

Skip destructive cleanup on disks that must preserve data. When the host is being fully reinstalled, remove old mappings before cleaning the approved disks:

```
swapoff -a
vgchange -an 2>/dev/null || true
systemctl stop multipathd.service multipathd.socket 2>/dev/null || true
systemctl mask multipathd.service multipathd.socket 2>/dev/null || true
multipath -F 2>/dev/null || true
udevadm settle
```

If the host has unrelated LVM, multipath, or shared-storage devices, do not use global deactivation commands. Deactivate only mappings that belong to the approved disks.

For each approved disk, clear its GPT, recognized signatures, and metadata at the start and end of the disk. The example below operates on `/dev/sda`; repeat it only for each disk that you have independently identified and approved.

```
disk_to_wipe=/dev/sda

lsblk -d -o NAME,PATH,SIZE,MODEL,SERIAL,WWN,HCTL "${disk_to_wipe}"
sgdisk --zap-all "${disk_to_wipe}"
wipefs -a -f "${disk_to_wipe}"
dd if=/dev/zero of="${disk_to_wipe}" \
  bs=1M count=64 conv=fsync status=progress

disk_sectors=$(blockdev --getsz "${disk_to_wipe}")
if [ "${disk_sectors}" -le 262144 ]; then
  echo 'REFUSE: disk is too small; stop and recheck the target'
else
  dd if=/dev/zero of="${disk_to_wipe}" bs=512 \
    seek=$((disk_sectors-131072)) count=131072 \
    conv=fsync status=progress
fi
```

Refresh the kernel and udev after all approved disks are cleaned:

```
partprobe /dev/sda /dev/sdb /dev/sdc /dev/sdd 2>/dev/null || true
udevadm trigger --subsystem-match=block
udevadm settle
```

Adjust the `partprobe` list to include only the disks cleaned on this host.

Verify that the cleaned disks have no signatures, partitions, filesystem types, labels, or partition labels:

```
wipefs /dev/sda /dev/sdb /dev/sdc /dev/sdd
lsblk -e7 -o \
NAME,PATH,TYPE,SIZE,MODEL,SERIAL,WWN,FSTYPE,LABEL,PARTLABEL,MOUNTPOINTS

if blkid /dev/sda /dev/sdb /dev/sdc /dev/sdd 2>/dev/null |
  grep -E 'LABEL="(OLD_)?COS_(GRUB|OEM|RECOVERY|STATE|PERSISTENT)"'; then
  echo COS_LABELS_REMAIN
else
  echo NO_COS_LABELS
fi
```

Adjust each device list to match the cleaned disks. Continue only when:

- `wipefs` prints no signature for any cleaned disk.
- `lsblk` shows no old child partitions, `FSTYPE`, `LABEL`, or `PARTLABEL`.
- The check prints `NO_COS_LABELS`.
- The only remaining `COS_LIVE` label belongs to the virtual CD device, not a system disk.

6. Create the Per-Host Install Target

List the stable WWN paths again after cleanup:

```
ls -l /dev/disk/by-id/wwn-*
lsblk -d -e7 -o NAME,PATH,SIZE,MODEL,SERIAL,WWN,HCTL
```

Create the gate only after selecting and verifying the intended whole disk. Replace the example WWN with the value recorded for this host:

```
test ! -e /dev/elemental-install-target
test ! -L /dev/elemental-install-target

ln -s \
  /dev/disk/by-id/wwn-0x0123456789abcdef \
  /dev/elemental-install-target

readlink -f /dev/elemental-install-target
test -b /dev/elemental-install-target && echo TARGET_IS_BLOCK
lsblk -d -o NAME,PATH,SIZE,MODEL,SERIAL,WWN,HCTL \
  "$(readlink -f /dev/elemental-install-target)"
```

If the alias already exists or is a dangling link, do not overwrite it. Inspect it with `readlink`, determine why it exists, and remove it only after confirming it is stale.

The link exists only in the current Live ISO session. It is expected to disappear after reboot; the installed system does not require it for normal startup.

7. Run the Final Gate and Start Installation

Verify the Live root, mounts, target link, network, and DNS one final time:

```
findmnt -n -o SOURCE,FSTYPE,TARGET /
findmnt -rn -o SOURCE,TARGET | grep -E '^/dev/(sd|mapper/)' || \
  echo NO_PHYSICAL_DISK_MOUNTS
readlink -f /dev/elemental-install-target
test -b /dev/elemental-install-target
ip -br addr
ip route show default
getent hosts <global-domain>
```

Start the installer only after every check passes:

```
systemctl unmask elemental-register-install.service
systemctl reset-failed elemental-register-install.service
systemctl start elemental-register-install.service
journalctl -fu elemental-register-install.service
```

In the log, confirm that registration succeeds, `/dev/elemental-install-target` resolves to the intended WWN disk, partitioning and installation complete, and the configured eject/reboot action runs.

Verify the Installed Host

After installation triggers a reboot:

1. Eject the BMC virtual media and restore normal disk-first boot order.
2. Confirm that the host does not return to the Live ISO.
3. Verify the new `MachineInventory` on the Global cluster:

```
kubectl -n cpaas-system get machineinventories.elemental.cattle.io -o w
ide
kubectl -n cpaas-system describe machineinventory <inventory-name>
```

4. Confirm that the inventory is `Ready=True`, reports the expected network, and has only one current Elemental disk layout.
5. Add the inventory to a `MachineInventoryPool` only after it is ready.
6. After Cluster API allocates the host, verify the `BaremetalMachine`, workload Node, control-plane components, and CNI as described in Create a Cluster on Bare Metal.

Troubleshooting

Symptom	Cause and action
The install service fails before you select a disk.	This is the expected fail-closed behavior while `/dev/elemental-install-target` is absent. Stop and mask the service, complete the checks, create the link, then reset and start the service.
`test -b /dev/elemental-install-target` fails.	The WWN path is wrong, missing, or does not resolve to a whole block device. Recheck `lsblk` and `/dev/disk/by-id`; do not start installation.

Symptom	Cause and action
More than one disk still has `COS_STATE` or `COS_PERSISTENT`.	A previous Elemental layout remains. Stop installation and clean every approved reinstall disk until `wipefs`, `blkid`, and `lsblk` all pass.
The same physical disk has a different `/dev/sdX` name after reboot.	This is expected on some storage controllers. Compare WWN, serial number, size, and controller address instead of the kernel name.
The MachineRegistration was patched, but the ISO still uses the old target.	Rebuild the SeedImage and download the newly generated ISO. MachineRegistration changes are not applied retroactively to an existing ISO.
Registration works in the Live ISO, but the installed host has no network after reprovision.	Diagnose installed-system cloud-init and NetworkManager activation separately. The Live ISO `nmcli` profile is temporary and does not prove that persistent reprovision networking succeeded.

Limitations

- This workflow intentionally requires one console action per host. It prevents automatic selection; it does not provide fully unattended heterogeneous disk selection.
- MachineRegistration metadata labels are not interpolated into `install.device`.
- A `device-selector` based only on disk size is safe only if exactly one disk can match, and current selection does not enforce uniqueness. The WWN gate is preferred for hosts with multiple equal-sized disks.
- The target must be a whole disk. Do not point the gate at a partition.

Huawei DCS

Overview

Install the Provider

Infrastructu

Create a Cluster

Troubleshoot a DCSMachine Stuck in Deleting

Huawei DCS Provider

TOC

- Overview
- Key Features
- Cluster Management Capabilities
  - UI-Based Management (Fleet Essentials)
  - YAML-Based Management
- Supported Kubernetes Versions
- Architecture
- Concepts and Terminology
  - DCS Platform Object Hierarchy
  - DCS Platform Concepts
    - DCS Site
    - DCS Cluster
    - DCS Host
    - DCS VM Folder
    - DCS Cluster Folder
    - DCS VM Template
    - DCS Datastore
    - DCS Datastore Cluster
    - DCS Distributed Virtual Switch (DVS)
    - DCS Port Group

DCS Volume
DCS Cloud Credential
Cluster API Resource Mapping
Storage Concepts: Datastore, Datastore Cluster, and Volume
Network Concepts: DVS, Port Group, and IP Identity
Field-to-Concept Reference
`DCSCluster` fields
`DCSMachineTemplate` / `DCSMachine` fields
`DCSIpHostnamePool` fields
Requirements
Documentation

Overview

The Infrastructure Provider for Huawei DCS creates and manages Alauda OS clusters on the Huawei Datacenter Virtualization Solution (DCS) platform.

DCS (Datacenter Virtualization Solution) is Huawei's virtualization platform that provides enterprise-grade virtualization capabilities. The DCS Provider integrates with DCS to manage virtual machines, networks, and storage resources for Kubernetes clusters.

Key Features

- **Virtual Machine Management**: Provision and manage VMs on DCS platform
- **Network Configuration**: Support for multi-NIC configurations with customizable network settings
- **Storage Management**: Flexible disk configuration with template disks and IP-slot-based persistent disks across multiple datastores
- **High Availability**: Built-in support for highly available control plane deployments
- **Self-built VIP**: Provider-managed control plane VIP (`DCSCluster.spec.controlPlaneLoadBalancer.type: internal`) through the `alive`

runtime with ACP `v4.4+`. Reserve an IPv4 VIP in the control-plane Layer 2 network and use the YAML workflow in Create a Cluster on Huawei DCS

- **Rolling Updates**: Automated rolling updates for zero-downtime maintenance
- **Upgrade Data Preservation**: Declared persistent disks can be detached from old VMs and reattached to replacement VMs during rolling upgrades so preserved node-local data is not lost with the system disk

Cluster Management Capabilities

The DCS Provider always supports **YAML-based** cluster management through Cluster API manifests. It also supports **UI-based** cluster management when **Fleet Essentials** is installed. The persistent-disk declaration on `DCSIpHostnamePool` remains YAML-only.

UI-Based Management (Fleet Essentials)

> **INFO**
>
> **Fleet Essentials upgrade boundary**
>
> Fleet Essentials 1.0.4 and later can request the ACP 4.3-and-later Distribution Version upgrade through CVO. It does not perform the DCS Kubernetes and Alauda OS replacement. Use Upgrading Kubernetes on Huawei DCS for that Phase 2 YAML procedure. Cluster creation and routine node-pool management through Fleet Essentials are unaffected by this boundary.

Fleet Essentials provides the UI extension anchors that allow Alauda Container Platform Infrastructure Provider for Huawei DCS to add DCS-specific pages dynamically. When Fleet Essentials is installed, you can manage DCS clusters through the web UI. The `persistentDisk` declaration on `DCSIpHostnamePool` must still be managed through YAML manifests:

Infrastructure Resource Management:

- Cloud Credentials - Store DCS platform access credentials
- IP Pools - Manage IP address allocations and multi-NIC settings. IP-slot persistent disks such as `/var/cpaas` remain YAML-managed

- Machine Templates - Define VM specifications for Control Plane and Worker Nodes, excluding pool-managed persistent disks

Cluster Lifecycle:

- Create Clusters - 5-step wizard workflow for guided cluster creation
- View cluster details with Overview, Nodes, Node Pools, and Aligned Extensions tabs
- Upgrade Clusters - Two-phase upgrade: Distribution Version → Kubernetes
- Destroy clusters with confirmation dialogs

Node Pool Management:

- View Control Plane and Worker Node Pools
- Add Worker Node Pools with configurable replicas and rollout strategies
- Delete Worker Node Pools
- Upgrade Kubernetes version per Node Pool
- View detailed Conditions for Control Plane Node Pool

YAML-Based Management

For automation and GitOps workflows, you can manage DCS clusters using YAML manifests. This workflow does not depend on Fleet Essentials:

- Define infrastructure resources (Secret, DCSIpHostnamePool, DCSMachineTemplate)
- Create cluster resources (KubeadmControlPlane, DCSCluster, Cluster)
- Manage worker nodes (MachineDeployment, KubeadmConfigTemplate)
- Perform rolling updates and upgrades with declared persistent disks preserved across node replacement

See Create a Cluster on Huawei DCS for YAML-based instructions.

Supported Kubernetes Versions

The DCS Provider supports Kubernetes versions as defined in the OS Support Matrix. Each Alauda OS release includes a specific Kubernetes version, and VM templates must be created using the corresponding Alauda OS images.

Architecture

The DCS Provider implements the Cluster API infrastructure provider specification:

- **DCSCluster**: Represents the DCS infrastructure cluster
- **DCSMachine**: Represents individual virtual machines
- **DCSMachineTemplate**: Defines VM templates for machine creation
- **DCSIpHostnamePool**: Manages IP and hostname allocations for machines

Concepts and Terminology

Several Huawei DCS platform concepts have names that closely resemble Cluster API custom resource names. Two of them deserve special attention because the name collision is exact: a `DCS Cluster` (the DCS platform's pool of physical hosts) is not the same object as a `DCSCluster` custom resource (the Cluster API infrastructure resource that represents a Kubernetes cluster on top of DCS). Reading the two as the same thing leads to incorrect `DCSMachineTemplate` and `DCSCluster` configurations.

This section enumerates the DCS platform concepts referenced by the provider, maps each concept to the Cluster API resource or field that references it, and clarifies the network and storage layering that is the most frequent source of configuration errors.

DCS Platform Object Hierarchy

The DCS platform organizes its objects in a hierarchy. The provider works with the following layers (top to bottom):

- **DCS Site** — the top-level scope of a single DCS deployment. Identified by a site ID such as `EC1C108C`.
- Within a site:

  - **DCS Clusters** — pools of physical hosts. A site contains one or more DCS Clusters.
  - **DCS VM Folders** and **DCS Cluster Folders** — organizational labels (logical groupings) for virtual machines and clusters respectively.
  - **DCS Datastores** and **DCS Datastore Clusters** — storage containers and groups of storage containers.
  - **DCS Distributed Virtual Switches (DVS)** — virtual switches that span the site.
- Within a DCS Cluster:
  - **DCS Hosts** — physical servers (for example, `CNA-01`, `CNA-02`).
- Within a DCS DVS:
  - **DCS Port Groups** — network configuration entries that virtual machine NICs attach to.
- Independent of the placement hierarchy:
  - **DCS VM Templates** — virtual machines marked as templates, used as the source image for cloning.
  - **DCS Volumes** — persistent disks created on DCS Datastores.

The provider does not own any of these objects. It references them by name or URN from custom resource fields.

DCS Platform Concepts

DCS Site

The top-level container of a single DCS deployment. The site ID is supplied as `DCSCluster.spec.site` and is also stored in the cloud-credential `Secret`. All DCS API paths used by the provider include the site ID (`/service/sites/{siteId}/...`).

DCS Cluster

A pool of physical hosts on the DCS platform. The DCS scheduler places virtual machines onto hosts that belong to a single DCS Cluster. A site can contain one or more DCS Clusters; a DCS Cluster contains one or more DCS Hosts. DCS Cluster names are visible in the DCS portal and are returned by the `GET /service/sites/{siteId}/clusters` API.

The name `DCS Cluster` collides with the `DCSCluster` Cluster API custom resource. See Cluster API Resource Mapping below.

DCS Host

A single physical server registered in a DCS Cluster. Names follow the local convention on the DCS platform (for example, `CNA-01`). Virtual machines run on hosts; the DCS scheduler chooses which host receives a new virtual machine, based on cluster-level policies (DRS, anti-affinity rules) and host capacity. The provider does not pin virtual machines to a specific DCS Host; the `DCSMachineTemplate` resource does not expose a host-binding field.

DCS VM Folder

An organizational label inside a DCS Site that groups virtual machines. A VM Folder does not affect compute placement; a virtual machine grouped under VM Folder `foo` still runs on a DCS Host belonging to some DCS Cluster. The provider queries VM Folders through `GET /service/sites/{siteId}/folder?type=1`.

VM Folders must be created on the DCS platform before being referenced from `DCSMachineTemplate.spec.template.spec.location` (with `type: folder`).

DCS Cluster Folder

An organizational label that groups DCS Clusters themselves, returned by the same `/folder` endpoint with a different `type` parameter. The provider does not query or reference DCS Cluster Folders. This concept is listed here only to prevent confusion with DCS VM Folder.

DCS VM Template

A virtual machine on the DCS platform that has been marked as a template (`isTemplate=true`). The provider clones from this template to create cluster nodes. A VM Template is identified by name (the value of `DCSMachine.spec.vmTemplateName`); the provider resolves the name to a URN by listing `GET /service/sites/{siteId}/vms?isTemplate=true&name=<name>`.

A VM Template carries its own disk on a DCS Datastore and its own network configuration. The provider overwrites the network configuration (NIC port group, customization) when

cloning; it does not overwrite the disk's storage type. The cloned virtual machine inherits the template's home DCS Cluster by default; see Infrastructure → Advanced: Multi-Cluster Deployment for cross-cluster usage.

DCS Datastore

A storage container on the DCS platform that holds the disks of virtual machines. Datastores can be backed by SAN, IPSAN, NAS, FusionStorage, or local disk. The storage backend determines whether a datastore is visible to one DCS Cluster or to several DCS Clusters: SAN, IPSAN, NAS, and FusionStorage typically support cross-cluster access; local disk does not.

Datastores are referenced from disk specifications by either `datastoreName` or `datastoreUrn`; the provider resolves the name to a URN before sending the clone request.

DCS Datastore Cluster

A group of DCS Datastores. When a disk specification references a Datastore Cluster (via `datastoreClusterName`), the DCS scheduler picks a member datastore that has free capacity. This is the recommended way to declare storage when the DCS administrator has configured a datastore cluster, because it tolerates single-datastore exhaustion.

A DCS Datastore Cluster is **not** a DCS Cluster. The two names overlap because both use the word "cluster" but the underlying concepts are different: a DCS Cluster is a host pool, and a DCS Datastore Cluster is a datastore pool. Watch for this collision when reading platform documentation or talking to DCS administrators.

DCS Distributed Virtual Switch (DVS)

A virtual switch on the DCS platform that spans one or more DCS Hosts. Virtual machine NICs attach to a DCS Port Group, which in turn belongs to a DCS DVS. The provider queries DVSs through `GET /service/sites/{siteId}/dvswitchs`.

DCS Port Group

A network configuration entry on a DCS DVS. Each port group carries a VLAN configuration, security policies, and other network settings. The provider queries port groups through `GET`

`{dvSwitchUri}/portgroups` (using the parent DVS URI) and references them by name (`portGroupName`) or URN (`portGroupUrn`).

A virtual machine NIC must attach to exactly one DCS Port Group. Additional NICs require additional port-group references.

DCS Volume

A persistent disk on the DCS platform. Volumes live on DCS Datastores and are identified by a URN such as `urn:sites:EC1C108C:volumes:9841`. The provider creates volumes through `POST /service/sites/{siteId}/volumes` when `DCSIpHostnamePool.spec.pool[].persistentDisk[]` declares disks that must survive virtual machine replacement; it attaches the resulting volume to the virtual machine through the `attachvol` action.

A DCS Volume is conceptually different from a Kubernetes `PersistentVolume`. A DCS Volume is a DCS-platform storage object attached at the hypervisor level; a Kubernetes `PersistentVolume` is a Kubernetes API object that represents storage available to pods. The provider only manages DCS Volumes; in-cluster `PersistentVolume` resources are managed separately by the workload's CSI driver.

DCS Cloud Credential

The authentication material used by the provider to call the DCS REST API. The provider obtains a session token by sending `X-Auth-User` and `X-Auth-Key` headers to `POST /service/session` and uses the returned `X-Auth-Token` header on subsequent requests. The credentials are stored as a Kubernetes `Secret` referenced from `DCSCluster.spec.credentialSecretRef`. The Secret contains `authUser`, `authKey`, `endpoint`, and `site` keys.

Cluster API Resource Mapping

The following table disambiguates the DCS platform concepts whose names resemble Cluster API custom resources.

DCS platform concept	Cluster API resource with a similar name	Why they are not the same
DCS Cluster	`DCSCluster` (CRD)	A DCS Cluster is a pool of physica platform. `DCSCluster` is a Cluste that represents the infrastructure s cluster running on top of the DCS p Cluster can host the virtual machin Kubernetes clusters.
DCS Host	Kubernetes Node	A DCS Host is a physical server re platform. A Kubernetes Node is a w Kubernetes cluster running `kubel` not expose DCS Hosts to Kuberne between a Kubernetes Node and tl is decided by the DCS scheduler a in any custom resource.
DCS VM Template	`DCSMachineTemplate` (CRD)	A DCS VM Template is an OS imag by a DCS administrator on the DCS `DCSMachineTemplate` is a Cluste references that template by name a specific configuration (CPU, memo `DCSMachineTemplate` does not c points to a DCS VM Template by
DCS Volume	Kubernetes `PersistentVolume`	A DCS Volume is a DCS-platform s at the hypervisor level. A Kubernet `PersistentVolume` is a Kuberne representing storage available to p creates DCS Volumes (driven by `DCSIpHostnamePool.spec.pool[` Kubernetes `PersistentVolume` r independently by the workload's CS
DCS Cloud Credential	Kubernetes `Secret` (referenced by `DCSCluster.spec.credentialSecretRef`)	A DCS Cloud Credential is the use DCS platform administrator. It is de as a Kubernetes `Secret` with the `authKey`, `endpoint`, and `site`

The following concept does not collide with any Cluster API resource but is internally confusable because of the word "cluster":

DCS platform concept	Internally confusable with	Why they are not the same
DCS Datastore Cluster	DCS Cluster	A DCS Datastore Cluster is a group of DCS Datastores (storage). A DCS Cluster is a pool of DCS Hosts (compute). The two share the word "cluster" but belong to different layers of the DCS platform.

Storage Concepts: Datastore, Datastore Cluster, and Volume

The DCS platform exposes three storage-related concepts that the provider references separately:

- A **DCS Datastore** is a single storage container. A virtual machine disk is pinned to one DCS Datastore.
- A **DCS Datastore Cluster** is a group of DCS Datastores. A virtual machine disk that references a DCS Datastore Cluster is placed on one member datastore that has free capacity; the DCS scheduler picks the member.
- A **DCS Volume** is a persistent disk object that lives on a DCS Datastore (or on a member datastore of a DCS Datastore Cluster). Volumes are created and attached to virtual machines through the provider when the `DCSIpHostnamePool` declares persistent disks; they survive virtual machine replacement.

The relationship between the three:

- A `DCSMachineTemplate` declares one or more disks under `vmConfig.dcsMachineDiskSpec[]`. Each disk binds to either a specific `datastoreName` or a `datastoreClusterName`. These template-level disks are created and destroyed together with the virtual machine.
- A `DCSIpHostnamePool` entry can declare one or more `persistentDisk[]` items. Each item binds to either a specific `datastoreName` or a `datastoreClusterName`, and the provider creates a corresponding DCS Volume that survives virtual machine replacement.

Cross-cluster visibility is determined by the underlying storage backend. SAN, IPSAN, NAS, and FusionStorage datastores are typically visible to several DCS Clusters; local-disk

datastores are visible only to a single DCS Cluster. This visibility is what determines whether a single DCS VM Template can be cloned into more than one DCS Cluster.

Network Concepts: DVS, Port Group, and IP Identity

Network configuration on a DCS-backed Kubernetes cluster is split across two layers:

- **DCS-side network plumbing (Layer 2)**: a virtual machine NIC must attach to a DCS Port Group, which belongs to a DCS Distributed Virtual Switch. The DVS and Port Group are DCS platform objects, created and administered on the DCS side. The provider references them by name (`dvSwitchName` , `portGroupName`) or by URN (`portGroupUrn`).
- **Cluster-side network identity (Layer 3)**: the IP address, subnet mask, gateway, DNS server, and hostname assigned to the virtual machine are not DCS platform objects. They are declared in `DCSIpHostnamePool.spec.pool[]` entries (a Cluster API resource owned by the cluster operator) and are written into the virtual machine through cloud-init or Ignition during bootstrap.

This split has two practical consequences:

- The DCS platform team owns the DVS and Port Group definitions. The cluster operator owns the IP and hostname plan. The two are coordinated through the `DCSIpHostnamePool` resource, which references the DCS-side port group while declaring the cluster-side IP identity.
- The provider does not call any DCS API to allocate or reserve IP addresses. Address conflicts must be avoided by the cluster operator when authoring `DCSIpHostnamePool` entries.

The Cluster API resource `DCSIpHostnamePool` and the DCS platform's port group are distinct objects. The DCS platform does not have a native "IP pool" concept; the `DCSIpHostnamePool` resource is the only place where per-virtual-machine network identity is recorded.

Field-to-Concept Reference

The following tables map the most commonly used custom resource fields to the DCS platform concept they reference and to the source of the value.

`DCSCluster` fields

Field	DCS platform concept	Value source
`spec.site`	DCS Site	Site ID supplied by the DCS platform administrator.
`spec.credentialSecretRef`	DCS Cloud Credential	Name of a Kubernetes `Secret` in the same namespace that holds the DCS user, key, endpoint, and site.

`DCSMachineTemplate` / `DCSMachine` fields

Field	DCS platform concept
`spec.template.spec.vmTemplateName`	DCS VM Template
`spec.template.spec.location` (with `type: folder`)	DCS VM Folder
`spec.template.spec.location` (with `type: cluster`)	DCS Cluster

Field	DCS platform concept
`spec.template.spec.vmConfig.dvSwitchName`	DCS DVS
`spec.template.spec.vmConfig.portGroupName`	DCS Port Group
`spec.template.spec.vmConfig.portGroupUrn`	DCS Port Group
`spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreName`	DCS Datastore
`spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreClusterName`	DCS Datastore Cluster

Field	DCS platform concept
`spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreUrn`	DCS Datastore
`spec.template.spec.vmConfig.cdRomDatastoreName`	DCS Datastore
`spec.template.spec.vmConfig.cdRomDatastoreClusterName`	DCS Datastore Cluster

`DCSIpHostnamePool` fields

Field	DCS platform concept	Value source
`spec.pool[].ip` / `mask` / `gateway` / `dns` / `hostname`	Cluster-side network identity (not a DCS platform concept)	Declared by the clu written into the virtu through cloud-init c
`spec.pool[].dvSwitchName`	DCS DVS	Per-IP override of t additional NICs. Fa template-level DVS
`spec.pool[].portGroupName` / `portGroupUrn`	DCS Port Group	Per-IP override of t for additional NICs

Field	DCS platform concept	Value source
`spec.pool[].persistentDisk[].datastoreName`	DCS Datastore	Name of the DCS I hold the persistent
`spec.pool[].persistentDisk[].datastoreClusterName`	DCS Datastore Cluster	Name of the DCS I that the persistent Mutually exclusive `datastoreName`.
`spec.pool[].persistentDisk[]` (the entry itself)	Creates a DCS Volume	The DCS Volume i provider through `/service/sites/{` and attached to the through the `attac`

Requirements

- DCS platform with API access
- A DCS administrator-role user, either:
  - `Interface interconnection user` (default), or
  - `Domain user` (LDAP/AD-backed; requires DCS domain authentication configured by your DCS administrator).

  See Credential User Types for selection criteria and DCS-side requirements.
- Virtual machine templates with Alauda OS images. Use the Alauda OS image published for the ACP release installed on the cluster; see Downloading Alauda OS.
- DCS VM templates `4.2.1` or later when you use pool-managed persistent disks, because safe shutdown and disk detach depend on guest tools
- Shared storage with cross-host access capability

Documentation

For detailed instructions on using the DCS Provider, see:

- Installation
- Infrastructure Resources
- Create a Cluster
- Manage Nodes
- Upgrade Kubernetes
- API Reference

Install the Huawei DCS Provider

This document describes how to install the Infrastructure Provider for Huawei DCS and the Cluster Bootstrap Provider on the `global` cluster.

TOC

When You Need This
Prerequisites
Steps
Download the packages
Upload the packages
Install the cluster plugins
Verify the installation
Next Steps

When You Need This

Complete this page after the `global` cluster is installed and before you create the first workload cluster on Huawei DCS. It is required regardless of the IaaS platform that the `global` cluster itself runs on. In a disaster recovery (DR) pair, complete it on **both** `global` clusters, so that the standby can reconcile the same workload clusters after a failover. Upload the packages to each `global` cluster.

This is a separate task from the provider steps in Installing the global Cluster. Those steps upload provider packages to the temporary bootstrap cluster to create the `global` cluster itself.

Prerequisites

Before installing the provider, ensure you have:

- Access to the Alauda Container Platform `global` cluster
- Access to Customer Portal for downloading plugins

Steps

1 Download the packages

To create clusters on Huawei DCS, you need two cluster plugins. Download both from the Customer Portal. For ACP 4.5, they are listed under **Common ACP Extensions**; see Download.

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Huawei DCS**

2 Upload the packages

Upload both packages to the platform with `violet push`. For the command and its parameters, refer to Upload Packages.

3 Install the cluster plugins

Install both cluster plugins on the `global` cluster. For the procedure, refer to Cluster Plugin.

4 Verify the installation

After installing the provider, verify that it is correctly installed:

```
# Check if the DCS provider pods are running
kubectl get pods -n cpaas-system | grep dcs

# Verify CRDs are registered
kubectl get crd | grep -E '^dcs.*\.infrastructure\.cluster\.x-k8s\.io
$'

# Expected output should include:
# dcsclusters.infrastructure.cluster.x-k8s.io
# dcsmachines.infrastructure.cluster.x-k8s.io
# dcsmachinetemplates.infrastructure.cluster.x-k8s.io
# dcsiphostnamepools.infrastructure.cluster.x-k8s.io
```

Next Steps

After installing the provider, you can proceed to:

- Prepare Infrastructure Resources
- Create a Cluster

Infrastructure Resources for Huawei DCS

TOC

- Overview
- Cloud Credentials
  - Using the Web UI
    - Prerequisites
    - Creating a Cloud Credential
    - Managing Cloud Credentials
  - Using the CLI
  - Credential User Types
- IP Pools
  - Using the Web UI
    - Prerequisites
    - Creating an IP Pool
    - Managing IP Pools
  - Using YAML
  - Additional NICs in the IP Pool
  - Persistent Disks in the IP Pool
- DCS Platform Capacity and Placement
  - Running the diagnostic snippets
  - Choosing a datastore
  - Host placement and high availability

Cross-Host High Availability for Control Plane
Troubleshooting: VM stuck in `creating`
Machine Templates
Using the Web UI
Prerequisites
Creating a Machine Template
Managing Machine Templates
Using YAML
Advanced: Multi-Cluster Deployment with Shared Storage
Resource Relationships
Next Steps

Overview

Before creating clusters on Huawei DCS, you need to configure infrastructure resources including cloud credentials, IP pools, and machine templates.

You can manage infrastructure resources using either the web UI or YAML manifests. The web UI provides a guided interface with validation, while YAML offers automation capabilities.

On Huawei DCS, the IP pool also carries any disks that must survive VM replacement. This includes the platform-required `/var/cpaas` disk.

INFO

Namespace Requirement All infrastructure resources must be deployed in the `cpaas-system` namespace to ensure proper integration with the platform as workload clusters.

INFO

DCS platform concepts If terms such as `DCS Cluster`, `DCS Host`, `DCS VM Folder`, or `DCS VM Template` are unfamiliar, or if it is not clear how they relate to the `DCSCluster` and

`DCSMachineTemplate` custom resources, read Huawei DCS Concepts and Terminology first.

Cloud Credentials

Cloud credentials store the DCS platform access information required for cluster operations.

Using the Web UI

Prerequisites

Before creating a cloud credential, verify the following DCS platform requirements:

User Configuration:

- **User Type**: One of the following:
  - `Interface interconnection user` — default; existing behaviour. Use this when an interconnect-style account is already provisioned in the DCS portal.
  - `Domain user` — LDAP/AD-backed user managed in the directory service. Requires that DCS has **domain authentication configured by your DCS administrator**, and that the directory user is **pre-created in the DCS portal** with the role mapping below.
- **Role**: Must be `administrator`

Password Policy (applies to `Interface interconnection user` only):

Navigate to **System Management** → **Rights Management** → **Rights Management Policy** and verify:

- **Policy**: `Whether to modify the password of an interface interconnection user upon password resetting and first login`
- **Value**: Must be set to **No**

If set to `Yes`, the user's password will be forced to change upon first login, breaking authentication and causing cluster creation failures.

For `Domain user`, password lifecycle is governed by your LDAP/AD directory service, not by DCS portal policy.

Creating a Cloud Credential

Navigation: Clusters → Cloud Credentials → Create Cloud Credential → Select Huawei DCS

Form Fields:

Field	Type	Required	Description
Name	text	Yes	Unique identifier for the credential (1-63 characters, lowercase letters, numbers, and hyphens only)
Display Name	text	No	Custom description for easy identification
DCS Endpoint	URL	Yes	DCS platform API address (must start with `http://` or `https://`)
Username	text	Yes	DCS platform API user login name
Password	password	Yes	DCS platform API user login password
Site	text	Yes	Site where the VM templates are located (all resources must be in the same site)

Validation Rules:

- Name must be 1-63 characters, containing only lowercase letters, numbers, and hyphens, and must start and end with a letter or number
- DCS Endpoint must be a valid URL format starting with `http://` or `https://`

Managing Cloud Credentials

Viewing Credentials: Navigate to Clusters → Cloud Credentials to view all configured credentials with their type, creation time, and creator.

Updating Credentials: Click **Update** on a credential to modify the Display Name. Password updates are not supported in the current version (planned for a future release).

Deleting Credentials: Click **Delete** to remove a credential. Confirm the deletion in the dialog.

> **INFO**
>
> **Domain user support: YAML only in this version**
>
> The web UI currently only creates `Interface interconnection user` credentials. To create a `Domain user` credential, use the CLI path below and set `userType` to `domain`.

Using the CLI

Create a Secret in `cpaas-system` that stores the DCS authentication information.

Author the credential file with an editor rather than typing the values at a shell prompt. A shell records the whole command, password included, in its history; a file written by an editor leaves no such copy.

dcs-credential.env

```
authUser=<auth-user>
authKey=<auth-key>
endpoint=https://<dcs-api-host>:7443
site=<dcs-site-id>
userType=interconnect
```

Restrict the file, create the Secret from it, then remove it:

```
chmod 600 dcs-credential.env

DCS_SECRET_NAME="<auth-secret-name>"

kubectl create secret generic "${DCS_SECRET_NAME}" \
  --namespace cpaas-system \
  --from-env-file=dcs-credential.env

rm -f dcs-credential.env
```

Write the values in plain text — `kubectl` base64-encodes them when it creates the Secret, so there is no separate encoding step. Each line is `key=value`. Do not quote the values: `kubectl` takes quote characters literally, so `authKey="s3cret"` stores `"s3cret"` with the quotes included. A value may contain `=`; only the first `=` on a line separates the key from the value.

Omit the `userType` line for an interface interconnection user, which is the default. Set it to `domain` for a domain user — see Credential User Types.

> **WARNING**
>
> Do not create a credential Secret with `kubectl apply`. Client-side apply stores the manifest it applied — including the DCS password — in the `kubectl.kubernetes.io/last-applied-configuration` annotation on the Secret itself. `kubectl create` writes no such annotation. This applies whether the values were written as `stringData` or as base64 `data`; base64 is an encoding, not protection.

To change a credential later, patch it rather than reapplying a manifest. Author the patch file with an editor, for the same reason:

`dcs-credential-patch.yaml`

```
stringData:
  authKey: <new-auth-key>
```

```
chmod 600 dcs-credential-patch.yaml

kubectl patch secret "${DCS_SECRET_NAME}" --namespace cpaas-system \
  --type=merge --patch-file=dcs-credential-patch.yaml

rm -f dcs-credential-patch.yaml
```

Parameter Descriptions:

Key	Description
`authUser`	DCS platform API user login name.
`authKey`	DCS platform API user login password.
`endpoint`	DCS platform API address, including the `http` or `https` scheme. **Note**: The default API port for the DCS platform is 7443 (not the common 8443). If your environment uses a custom port, confirm with your administrator.
`site`	DCS site ID. The provider reads it from this Secret and writes it to `DCSCluster.spec.site` when that field is left empty, so setting it here is enough. Set `DCSCluster.spec.site` explicitly only to override the Secret for one cluster.
`userType`	Optional. Allowed values are `interconnect` (default; used when the key is absent or empty) or `domain`. Matching is case-insensitive (`Domain`, `DOMAIN`, and `domain` all map to the same value). Any other value, including the legacy DCS magic numbers `1` and `2`, is rejected with a configuration error so a `domain` user is never silently authenticated as `interconnect`.

Credential User Types

DCS Provider supports two credential user types, selected by the optional Secret key `userType`:

`userType` value	Maps to DCS user	When to use	DCS-side requirement	Password lifecycle
`interconnect` (default; equivalent to the key being absent or empty)	`Interface interconnection user`	Existing deployments and the simplest path when an interconnect account is already provisioned in DCS.	DCS portal must keep the policy "Whether to modify the password of an interface interconnection user upon password resetting and first login" set to **No**.	Governed by DCS portal policy.
`domain`	`Domain user` (LDAP/AD-backed)	Customers whose security audit requires LDAP/AD-managed credentials, or deployments that already standardize on directory-service accounts.	DCS must have **domain authentication configured by your DCS administrator**, and the directory user must be **pre-created in the DCS portal with the `administrator` role**.	Governed by your LDAP/AD directory service, not by DCS portal policy.

Compatibility: Existing Secrets without a `userType` key continue to authenticate as before; no migration is required to keep using interconnect-style credentials.

UI coverage: The current web UI form does not expose the `userType` field. To use the `domain` value, configure the Secret through YAML.

IP Pools

IP pools define the network configuration (IP addresses, subnet masks, gateways, DNS) for cluster nodes. Each pool can contain multiple node entries, and each node can have multiple network interface configurations.

For DCS multi-NIC clusters, each IP entry can declare `additionNic` items. These additional NICs are bound to the IP slot and are applied when the provider creates a new VM. Updating `additionNic` on an existing Pool does not hot-add NICs to already-running VMs.

For DCS persistent disks, each IP entry can also declare `persistentDisk` items. These disks are bound to the IP slot instead of the VM lifecycle, so they can be detached from an old VM and reattached to a replacement VM during rolling upgrades. Use this mechanism for the platform-required `/var/cpaas` disk and for any other node-local data that must survive delete-recreate operations. This `persistentDisk` declaration is supported through YAML only.

Using the Web UI

Prerequisites

- Cloud Credential has been created

> **INFO**
>
> The IP Pool web UI covers standard IP, hostname, and network settings only. It does not expose `DCSIpHostnamePool.spec.pool[].persistentDisk`. Configure persistent disks through YAML manifests.

Creating an IP Pool

Navigation: Clusters → Virtual Machine → IP Pools → Create IP Pool → Select Credential

Form Structure:

The IP Pool form consists of a list of **Pools**. Each Pool represents one node and contains:

1. **Node IP** (required, exactly one per Pool)
2. **Additional NIC IPs** (optional, multiple per Pool)

Node IP Fields:

Field	Type	Required	Description
IP	IP address	Yes	IP address for the Kubernetes Node
Subnet Mask	CIDR prefix length	Yes	Subnet mask for the network, entered as a prefix length such as `24`
Gateway	IP address	Yes	Gateway IP address
DNS	IP address	No	DNS server addresses (semicolon-separated for multiple)
Hostname	text	No	Hostname for the virtual machine
Machine Name	text	No	Virtual machine name in the DCS platform
dvSwitch Name	dropdown	No	Virtual switch name (from DCS platform)
Port Group Name	dropdown	No	Port group name (from DCS platform)

Additional NIC IPs Fields:

Field	Type	Required	Description
IP	IP address	Yes	Non-Node IP address (e.g., storage network)
Subnet Mask	CIDR prefix length	Yes	Subnet mask for the network, entered as a prefix length such as `24`
Gateway	IP address	Yes	Gateway IP address
DNS	IP address	No	DNS server addresses (semicolon-separated for multiple)

Field	Type	Required	Description
dvSwitch Name	dropdown	Yes	Virtual switch name (from DCS platform)
Port Group Name	dropdown	Yes	Port group name (from DCS platform)

Validation Rules:

- IP addresses must be unique within the same IP Pool
- IP addresses must be valid IPv4 format
- Subnet mask must be entered as a valid CIDR prefix length, such as `24`. Do not include `/` or use dotted-decimal format such as `255.255.255.0`.
- IP address must be within the configured subnet range
- Gateway must be a valid IPv4 address within the subnet range

Tips:

- At least one node entry is required
- Exactly one Node IP configuration is required per node
- `additionNic[]` entries are optional for multi-NIC scenarios, such as storage network separation

Managing IP Pools

Viewing Pools: Navigate to Clusters → Virtual Machine → IP Pools to view all configured pools with their node IPs and creation time.

Updating Pools: Click **Update** to add or remove node entries and modify network configurations.

Deleting Pools: Click **Delete** to remove a pool. Confirm the deletion in the dialog.

Using YAML

Create a `DCSIpHostnamePool` resource:

dcs-ippool.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <iphostname-pool-name>
  namespace: cpaas-system
spec:
  pool:
  - ip: "<ip-1>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-1>"
    machineName: "<machine-name-1>"
    additionNic:
    - ip: "<additional-ip-1>"
      mask: "<additional-cidr-prefix-length>"
      gateway: "<additional-gateway>"
      dns: "<additional-dns>"
      dvSwitchName: "<additional-dvs>"
      portGroupName: "<additional-port-group>"
  - ip: "<ip-2>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-2>"
    machineName: "<machine-name-2>"
  - ip: "<ip-3>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-3>"
    machineName: "<machine-name-3>"
```

Parameter Descriptions:

Parameter	Type	Description	Require
`.spec.pool[].ip`	string	IP address for the virtual	Yes

Parameter	Type	Description	Required
		machine to be created	
`.spec.pool[].mask`	string	Subnet mask in CIDR prefix-length format without `/`, for example `24`	Yes
`.spec.pool[].gateway`	string	Gateway IP address	Yes
`.spec.pool[].dns`	string	DNS server IPs (use `;` to separate multiple servers)	No
`.spec.pool[].machineName`	string	Name of the virtual machine in the DCS platform	No
`.spec.pool[].hostname`	string	Hostname of the virtual machine	No
`.spec.pool[].additionNic[]`	[]object	Additional NICs for the VM created from this IP slot. The provider applies them only when creating a new VM.	No
`.spec.pool[].additionNic[].ip`	string	Additional NIC IP address for the generated	Recomm

Parameter	Type	Description	Require
		guest network configuration. This is not the Kubernetes Node IP.	
`.spec.pool[].additionNic[].mask`	string	Additional NIC subnet mask in CIDR prefix-length format without `/`, for example `24`.	Recomm
`.spec.pool[].additionNic[].gateway`	string	Gateway for the additional NIC network. Confirm route behavior before setting this value.	No
`.spec.pool[].additionNic[].dns`	string	DNS server IPs for the additional NIC network, separated by semicolons when multiple values are required.	No
`.spec.pool[].additionNic[].dvSwitchName`	string	DCS distributed virtual switch used to resolve the Port Group for the additional NIC.	Yes*
`.spec.pool[].additionNic[].portGroupName`	string	DCS Port Group used to	Yes*

Parameter	Type	Description	Require
		attach the additional NIC.	

*Fields marked `Yes*` are required for the provider to resolve the target network and attach the additional NIC. Fields marked `Recommended*` are needed for the generated static guest network configuration. The CRD does not enforce these fields.

> **WARNING**
>
> You must configure machine information for a number of machines greater than or equal to the number of nodes you plan to deploy. Insufficient entries will prevent node deployment.

Additional NICs in the IP Pool

Declare additional NICs in `DCSIpHostnamePool.spec.pool[].additionNic[]`.

Use this field when a node needs a secondary network, such as a storage, management, or application-isolation network. Keep the primary `.spec.pool[].ip` as the Kubernetes Node IP unless you intentionally plan a Node IP change.

How it takes effect:

- The provider copies `additionNic[]` from the Pool slot into `DCSMachine.status.additionalNic` when a new `DCSMachine` claims that slot.
- The provider adds those NICs when it clones the DCS VM.
- The guest OS configures the interfaces during first boot through the generated NetworkManager configuration.
- Existing VMs are not hot-updated when the Pool is edited later.

Network boundaries:

- The DCS provider does not reserve additional NIC IP addresses on the DCS platform. Avoid IP conflicts before applying the Pool.

- The referenced DVS and Port Group must already exist, and the target DCS hosts must be able to access them.
- Provide both `dvSwitchName` and `portGroupName` for each additional NIC. The provider uses them to resolve the Port Group URN before cloning the VM.
- The provider models IP, mask, gateway, DNS, DVS, and Port Group only. Route metrics, static routes, and NIC roles must be handled through MCP or guest OS configuration.
- If an additional NIC has a gateway, verify that it does not unexpectedly take over the default route from the primary NIC.

Persistent Disks in the IP Pool

Declare upgrade-preserved disks in `DCSIpHostnamePool.spec.pool[].persistentDisk`, not in `DCSMachineTemplate`.

YAML is the only supported way to declare these persistent disks.

- Use the IP entry to bind each persistent disk to a fixed `(ip, slot)` identity.
- Use this model for the platform-required `/var/cpaas` disk.
- Keep `DCSMachineTemplate` focused on the system disk and any template-local disks that may be recreated with the VM.
- When a replacement VM boots, the guest disk setup script checks for an existing filesystem. If one already exists, it skips formatting and mounts the disk directly.
- Pool-managed persistent disks require one-by-one replacement. Keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge = 0` and `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0` when you rely on this feature.

If you plan to use persistent disks, ensure the DCS VM template is `4.2.1` or later because safe shutdown and disk detach depend on guest tools inside the guest OS.

Example:

dcs-ippool-with-persistent-disk.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <iphostname-pool-name>
  namespace: cpaas-system
spec:
  pool:
  - ip: "<ip-1>"
    mask: "<cidr-prefix-length>"
    gateway: "<gateway>"
    dns: "<dns>"
    hostname: "<hostname-1>"
    machineName: "<machine-name-1>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
      mountOptions:
      - defaults
```

Persistent disk field descriptions:

Field	Description
`slot`	Disk position within the IP slot. It determines attach order, guest device naming, and the sequence number used during volume claim and reattach.
`quantityGB`	Disk size in GB.
`datastoreName` / `datastoreClusterName`	Specify exactly one storage target for the persistent disk.
`isThin`	Optional. Controls DCS thin provisioning for newly created persistent volumes. If omitted, the provider does not send `isThin` and DCS uses the platform default.
`path`	Mount path inside the guest OS. `/var/cpaas` is the platform-required path.

Field	Description
`format`	Filesystem used when the disk is first initialized. If the replacement VM sees an existing filesystem, formatting is skipped.
`options`	`mkfs` options applied only when the disk is first formatted.
`mountOptions`	Mount options applied when the disk is mounted.
`pciType`	Optional PCI disk type. If omitted, the controller uses the default value from the implementation.

Update rules:

- You can append new `persistentDisk` entries to an existing IP slot. The controller attaches the newly added disk to the running VM on the DCS side, but it does not format or mount the disk inside the guest OS. Guest formatting and mounting take effect only after the VM is replaced and the replacement VM runs the generated disk setup during bootstrap.
- The controller creates new persistent volumes as independent persistent normal volumes. When it reuses an existing volume, it only requires the volume to be independent and persistent; it does not require a specific DCS volume type.
- Do not delete existing `persistentDisk` entries from the spec. The webhook rejects removal.
- Treat `format`, `options`, and `pciType` as immutable after creation.
- Treat `quantityGB` and datastore changes as rollout-sensitive changes. The webhook performs best-effort platform validation when the pool carries a cluster label. `isThin` is creation-only: changing it does not validate, reconcile, or convert existing volumes, and affects only persistent volumes created later.
- Treat `path` and `mountOptions` changes as guest-side changes that take effect on replacement machines during rolling updates.

DCS Platform Capacity and Placement

Before authoring `DCSMachineTemplate` resources, verify that the target DCS site, datastore, and compute cluster have enough free capacity to host the planned VMs, and that the DCS-side scheduling policy can place new VMs on the available hosts. Skipping these checks is the single most common cause of VMs that appear to hang in `creating` state with no error returned by the DCS API.

Running the diagnostic snippets

The capacity and placement checks below use read-only DCS REST calls. Before running any snippet, set the following environment variables in your shell:

Variable	Meaning	How to obtain
`DCS_BASE`	DCS REST API base URL, including scheme and port. The DCS REST API is published on port 7443 (separate from the 8443 web portal).	Ask the DCS platform administrator for the API endpoint. The same value is stored in the cloud-credential `Secret` under the `endpoint` key.
`SITE`	DCS site ID (uppercase alphanumeric, for example `EC1C108C`).	Stored in the cloud-credential `Secret` under the `site` key, and also visible as `DCSCluster.spec.site`.
`TOKEN`	Short-lived `X-Auth-Token` returned by `POST /service/session`. The token is valid for approximately 30 minutes of idle time.	Run the session command shown below. The DCS user and key are stored in the cloud-credential `Secret` under `authUser` and `authKey`.

Set the three variables once at the start of the diagnostic session:

```
# Replace the placeholders with values appropriate for your environment.
export DCS_BASE='https://<dcs-api-host>:7443'
export SITE='<site-id>'

# Read the user and key from the cloud-credential Secret without printing
them.
DCS_USER="$(kubectl -n cpaas-system get secret <credential-secret-name> -
o jsonpath='{.data.authUser}' | base64 -d)"
DCS_KEY="$(kubectl -n cpaas-system get secret <credential-secret-name> -o
jsonpath='{.data.authKey}' | base64 -d)"

# Acquire a session token. The response header X-AUTH-TOKEN carries the v
alue used by subsequent calls.
HEADERS="$(mktemp)"
curl -k -sS -D "$HEADERS" -o /dev/null -X POST "${DCS_BASE}/service/sessi
on" \
  -H "X-Auth-User: $DCS_USER" \
  -H "X-Auth-Key: $DCS_KEY" \
  -H 'X-ENCRYPT-ALGORITHM: 1' \
  -H 'X-Auth-UserType: 2' \
  -H 'version: 8.1'
export TOKEN="$(grep -i '^X-AUTH-TOKEN:' "$HEADERS" | tail -n1 | cut -d'
' -f2- | tr -d '\r\n ')"
rm -f "$HEADERS"
unset DCS_USER DCS_KEY
```

WARNING

Token handling Do not log the token to a file or to shell history. The session token grants the same privileges as the underlying DCS user, which is typically an administrator.

Reuse `$DCS_BASE`, `$SITE`, and `$TOKEN` in every subsequent diagnostic snippet on this page. Refresh the token by re-running the `POST /service/session` command when the API begins returning `errorCode: 10000002` (session expired or not logged in).

Choosing a datastore

Each `DCSMachineDiskSpec` entry binds a disk to a DCS datastore through one of two mutually exclusive fields:

- `datastoreName` — pins the disk to a specific datastore. The DCS scheduler does **not** look at sibling datastores even if they have free space. This is the strictest binding.
- `datastoreClusterName` — pins the disk to a datastore cluster (a DCS-side group of datastores). The DCS scheduler chooses one specific datastore from the cluster that satisfies capacity. This is the safer default when the platform provides a datastore cluster.

Prefer `datastoreClusterName` whenever the DCS administrator has configured a datastore cluster, so a single near-full datastore does not block deployments while sibling datastores still have free space.

Capacity preflight:

Before applying a new `DCSMachineTemplate`, list the existing DCS datastores and confirm that the target has enough free capacity for the worst case of `system disk + ∑(spec disk size) × replica count`. From an environment with DCS API access, the read-only listing looks like this:

```
curl -k -sS "${DCS_BASE}/service/sites/${SITE}/datastores" \
  -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1' \
  | python3 -c "
import json,sys
for ds in json.load(sys.stdin).get('datastores',[]) or []:
  print(f\"{ds.get('name','?'):24} cap={ds.get('capacityGB','?')}GB used=
{ds.get('usedSizeGB','?')}GB free={ds.get('freeSizeGB','?')}GB state={ds.
get('status','?')}\")"
```

Each `DCSMachineTemplate` cp / worker pair commonly needs the system disk (template-default, often ~80 GB) plus the explicit data disks declared in the spec, so a single control-plane node with `etcd 10G + kubelet 100G + containerd 100G + /var/cpaas 100G` consumes roughly 390 GB before the template's system disk. A datastore used above ~70 % is a strong signal to pick a different datastore — even if the absolute free space looks large, fragmentation and reservations can cause new VM placement to stall.

Host placement and high availability

Host placement on the DCS platform is performed by the DCS scheduler. The DCS Provider does not expose a way to pin a virtual machine to a specific DCS Host through `DCSMachineTemplate`. Virtual machines cloned from the same template can therefore land on the same physical host.

Two consequences follow from this:

- **Capacity hot spots**: If the DCS cluster has uneven host load (for example, one host is severely CPU-over-committed while another is mostly idle), the DCS scheduler may refuse new placements on the over-committed host. The deploy task can stall until the load is rebalanced on the DCS side or DRS is enabled. Use the snippet below to confirm whether host load is the cause.
- **Control plane spread depends on DRS**: Control-plane virtual machines are not pinned to specific hosts through the machine template. For high-availability deployments, enable provider-managed DRS mutual exclusion as described in Cross-Host High Availability for Control Plane below.

To check current host load before applying a new template:

```
curl -k -sS "${DCS_BASE}/service/sites/${SITE}/hosts" \
  -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1' \
  | python3 -c "
import json,sys
for h in json.load(sys.stdin).get('hosts',[]) or []:
  print(f\"{h.get('name','?'):10} status={h.get('status','?'):10} cpuUse
d/total={h.get('cpuUsedCores','?')}/{h.get('cpuTotalCores','?')}c memUsed
={h.get('memUsedSizeMB','?')}MB cpuMuxRatio={h.get('cpuMuxRatio','?')}
\")"
```

A `cpuMuxRatio` (CPU allocation / physical cores) above ~3× on every host in the cluster indicates the cluster is already saturated and will likely refuse new placements until the load is rebalanced on the DCS side.

Cross-Host High Availability for Control Plane

The DCS Provider does not implement host pinning or Cluster API `FailureDomain` spreading. Instead, it can maintain a provider-owned DCS DRS rule for control-plane virtual machines when you enable `DCSCluster.spec.controlPlaneHA`.

The provider creates and updates a DRS `ruleType=2` mutual-exclusion rule after the control-plane VMs exist and their VM URNs are known. The rule lists the current control-plane VMs for one workload cluster and asks the DCS scheduler to keep them on different physical hosts. DCS remains responsible for executing the placement decision and any runtime migration.

The provider does not call the DCS Run DRS API and does not apply DRS recommendations. If `ControlPlaneHAReady` stays `False` with reason `ControlPlaneHAPending` after the rule has been maintained, the DCS scheduler has not converged the placement yet. Wait for DCS-side scheduling, or use the DCS platform operation process to run DRS and apply the generated recommendations.

Before you enable this feature, verify the following DCS-side requirements:

- The target DCS compute cluster has DRS enabled.
- The target DCS compute cluster has enough healthy hosts and capacity. A three-replica control plane normally needs at least three usable hosts.
- The selected datastores are visible from all hosts that may run the control-plane VMs.
- Runtime DRS migration is not blocked by DCS-side constraints such as a mounted CD-ROM device.

Enable the feature in `DCSCluster.spec.controlPlaneHA` when you write the cluster manifest. For the manifest example and the status fields to watch, see Configure DCSCluster and Verify Control Plane HA.

After the cluster is created, inspect the condition and status on `DCSCluster`:

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{range .status.conditions[?(@.type=="ControlPlaneHARead
y")]}{.status}{" "}{.reason}{" "}{.message}{"\n"}{end}'

kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{.status.controlPlaneHA}{"\n"}'
```

`ControlPlaneHAReady=True` means the provider-managed DRS rule exists and the provider has observed the current control-plane VMs on different DCS hosts. `ControlPlaneHAReady=False` with reason `ControlPlaneHAPending` usually means the

controller is waiting for VM URNs, waiting for enough members, or waiting for DCS-side placement to converge. `ControlPlaneHAReady=False` with reason `ControlPlaneHAFailed` means the provider could not query, validate, create, update, delete, or check the DRS rule. Read the condition message before checking controller logs.

Troubleshooting: VM stuck in `creating`

If a `DCSMachine` stays in `Provisioning` with `status.cdRomFile` set (so DCS API auth and ISO upload succeeded) but the DCS-side VM never leaves `status=creating` and never gets a `hostName`, the failure is almost always at the DCS placement layer, not in the controller.

Diagnostic ladder, in order:

1. **Confirm it is a placement stall**, not a slow boot. Query the VM and the matching deploy task on DCS:

   ```
   curl -k -sS "${DCS_BASE}/service/sites/${SITE}/vms?name=${VM_NAME}&limi
   t=10" \
     -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1'
   curl -k -sS "${DCS_BASE}/service/sites/${SITE}/tasks?status=running&lim
   it=50" \
     -H "X-Auth-Token: ${TOKEN}" -H 'version: 8.1'
   ```

   Symptoms of a placement stall: `status=creating`, `hostName=` (empty / unscheduled), `disks` array empty or unchanging, and the deploy task sitting at the same `progress` value with no errors for more than ~5 minutes.

2. **Check datastore free space** and **host load** using the snippets in Choosing a datastore and Host placement requirements. Datastore above 70 % used, or all hosts at high `cpuMuxRatio`, are the leading causes.

3. **Rebalance load on the DCS side**. If a single host is saturated while others have free capacity, ask the DCS platform team to rebalance the load (manual `Migrate` or DRS-driven rebalance) before retrying. The DCS Provider does not expose a per-template host pin, so this remediation is owned by the DCS platform team.

4. **If the DCS-side deploy task is genuinely deadlocked** (cancellable but unresponsive to ACP-driven `DeleteVm`), escalate to a DCS administrator with portal-admin credentials to

cancel the task. The 7443 REST API does not expose a task-cancel endpoint for non-admin users, so the recovery path is owned by the DCS platform team, not by ACP.

5. **Watch out for serialization side-effects**. In environments with a global DeployVM lock, a single stuck task can cause every new deploy to stall at the same progress percentage. Resolving the original stuck task usually unblocks the queue.

A `DCSMachine` whose status conditions show `InfrastructureReady=False reason=WaitingForInfrastructure` for more than 10 minutes, together with the symptoms above, should be treated as a DCS-platform incident and not as a Cluster API or DCS Provider regression.

Machine Templates

Machine templates define the virtual machine specifications (VM template, CPU, memory, disk, network) for cluster nodes. Each machine template has a **Type** that determines its usage:

- **Control Plane**: For control plane nodes
- **Worker Node**: For worker nodes

Using the Web UI

Prerequisites

- IP Pool has been created
- VM Template has been created in the DCS platform using Alauda OS image
- ConfigMap YAML has been applied to the global cluster

VM Template and ConfigMap:

Each Alauda OS release includes a ConfigMap YAML that maps VM templates to Kubernetes versions. Apply this YAML before creating machine templates:

```
apiVersion: v1
data:
  corednsTag: 1.12.4-v4.2.3
  etcdTag: v3.5.21-251117
  kubernetesVersion: v1.33.6
  vmImageVersion: alaudaos-42.m55.202512081125-0.x86_64
kind: ConfigMap
metadata:
  labels:
    cpaas.io/dcs-vm-template: <vm-template-name>
    cpaas.io/distribution-version: v4.2.0
    cpaas.io/kubernetes-version: v1.33
  name: 420-dcs-vm-template
  namespace: cpaas-system
```

Important: The `cpaas.io/dcs-vm-template` label value must match the VM template name in the DCS platform.

Creating a Machine Template

Navigation: Clusters → Virtual Machine → Machine Templates → Create Machine Template → Select Credential

Form Fields:

Field	Type	Required	Description
Name	text	Yes	Unique identifier for the template (1-63 characters, lowercase letters, numbers, and hyphens only)
Type	dropdown	Yes	Control Plane or Worker Node
VM Template Name	dropdown	Yes	From ConfigMap, shows OS Version and Kubernetes Version
Location	dropdown	No	DCS VM Folder used to group the cloned virtual machines on the DCS platform. See Huawei DCS Concepts and Terminology for the VM Folder

Field	Type	Required	Description
			definition. The folder must be created on the DCS platform first.
Specs	-	Yes	CPU and memory specifications
Specs.CPU	number	Yes	CPU cores (integer)
Specs.Mem	number	Yes	Memory size in MB (displayed as GB in list view)
Disk	-	Yes	Disk configuration (see below)
IP Pool	dropdown	Yes	Reference to an existing IP Pool

Disk Configuration:

The disk configuration varies by template type.

Control Plane Required Disks:

Mount Path	Default Size (GB)	Can Delete
System Volume	(template default)	No
`/var/lib/kubelet`	100	No
`/var/lib/containerd`	100	No
`/var/lib/etcd`	10	No

Worker Node Required Disks:

Mount Path	Default Size (GB)	Can Delete
System Volume	(template default)	No
`/var/lib/kubelet`	100	No

Mount Path	Default Size (GB)	Can Delete
`/var/lib/containerd`	100	No

You may add additional disks, but must retain all mandatory disks listed above.

> **INFO**
>
> **Platform-required persistent disk**
>
> `/var/cpaas` is still required by the platform, but it is no longer documented as a `DCSMachineTemplate` disk. Configure it in the matching `DCSIpHostnamePool.spec.pool[].persistentDisk` entry instead so it can survive VM replacement.

Disk Field Descriptions:

Field	Type	Description
Mount Path	text	Directory path for disk mounting
Disk Size	number (GB)	Size of the disk
Datastore	dropdown	Type: ClusterName or Name, then select from DCS platform

VM Template Selection Tip:

> **TIP**
>
> If multiple VM templates have the same Kubernetes version, select the template with the most recent OS version to benefit from the latest security updates and system improvements.

Managing Machine Templates

Viewing Templates: Navigate to Clusters → Virtual Machine → Machine Templates to view all templates with their VM Template Name, Location, Specs, and IP Pool.

Updating Templates: Click **Update** to modify specifications. Note that the Name field cannot be changed after creation.

Deleting Templates: Click **Delete** to remove a template. Confirm the deletion in the dialog.

Using YAML

Create a `DCSMachineTemplate` resource:

dcs-machinetemplate.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSMachineTemplate
metadata:
  name: <machine-template-name>
  namespace: cpaas-system
spec:
  template:
    spec:
      vmTemplateName: <vm-template-name>
      location:
        type: folder
        name: <folder-name>
      vmConfig:
        dvSwitchName: <dv-switch-name> # Optional
        portGroupName: <port-group-name> # Optional
        dcsMachineCpuSpec:
          quantity: <cpu-cores>
        dcsMachineMemorySpec: # MB
          quantity: <memory-mb>
        dcsMachineDiskSpec: # GB
        - quantity: 0
          datastoreClusterName: <datastore-cluster-name>
          systemVolume: true
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/kubelet
          format: xfs
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/containerd
          format: xfs
        - quantity: 10
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/etcd
          format: xfs
      ipHostPoolRef:
        name: <iphostname-pool-name>
```

Parameter Descriptions:

Parameter	Type
`.spec.template.spec.vmTemplateName`	string
`.spec.template.spec.location`	object
`.spec.template.spec.location.type`	string
`.spec.template.spec.location.name`	string
`.spec.template.spec.vmConfig`	object
`.spec.template.spec.vmConfig.dvSwitchName`	string
`.spec.template.spec.vmConfig.portGroupName`	string

Parameter	Type
`.spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec`	[]object
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreClusterName`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].systemVolume`	bool
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].path`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].format`	string
`.spec.template.spec.ipHostPoolRef.name`	string

*Required when parent object is specified

Advanced: Multi-Cluster Deployment with Shared Storage

The default workflow assumes that the DCS VM Template (`vmTemplateName`) and the cloned virtual machines live in the same DCS Cluster. For most deployments, this assumption is

correct and the `spec.template.spec.location` field is set to a VM Folder for organizational grouping only.

Deployments that span multiple DCS Clusters and want to clone from a single shared DCS VM Template into a different target DCS Cluster require the following conditions:

- The DCS Datastore that holds the DCS VM Template must be visible to the target DCS Cluster. This is the case for SAN, IPSAN, NAS, or FusionStorage datastores configured for cross-cluster access; it is not the case for datastores backed by host-local disk.
- The DCS-side network resources (distributed virtual switches and port groups) referenced by the `vmConfig` must also be visible to the target DCS Cluster.

When both conditions are met, the cloned virtual machine can be placed under the target DCS Cluster by setting `spec.template.spec.location.type: cluster` and `spec.template.spec.location.name: <target-cluster-name>` instead of the default `type: folder` form. If the datastore is not shared across DCS Clusters, the clone task fails at the DCS platform side. The recommended alternative is to prepare a separate DCS VM Template inside each target DCS Cluster, so each `DCSMachineTemplate` references a template that already lives in the cluster where the virtual machines will run.

Confirm cross-cluster datastore visibility with the DCS platform administrator before using this configuration.

WARNING

Storage Requirements

Datastore Cross-Host Access The datastore clusters (`datastoreClusterName`) must support cross-host access across all physical machines in the DCS platform. If a datastore is only available on specific hosts, VM creation will fail when the DCS platform attempts to schedule the VM on a different host.

Shared Storage for Ignition If your datastore does not support direct file uploads (required for Ignition configs), you must provide a shared storage solution (e.g., NFS) that supports multi-host mounting.

Disk Configuration Rules You may add custom template disks, but must retain mandatory disks by node role:

- Control plane: `systemVolume`, `/var/lib/kubelet`, `/var/lib/containerd`, `/var/lib/etcd`
- Worker: `systemVolume`, `/var/lib/kubelet`, `/var/lib/containerd` Configure `/var/cpaas` in the IP pool as a persistent disk.

Resource Relationships

Infrastructure resources have the following dependency relationships:

```
Cloud Credential
    ↓
IP Pool
    (network settings + persistent disk declarations)
    ↓
Machine Template → references IP Pool
    ↓
Cluster Creation
```

Resource Reusability:

- One Cloud Credential can be used for multiple clusters
- Multiple IP Pools can be created for different network segments
- Multiple Machine Templates can be created for different node types and specifications

---

Next Steps

After configuring infrastructure resources:

- Create a Cluster
- Manage Node Pools

- Upgrade Clusters

Create a Cluster on Huawei DCS

This document provides instructions for creating Kubernetes clusters on the Huawei DCS platform. YAML-based cluster creation is available through manifests. If Fleet Essentials is installed, you can also create clusters through the web UI. The `persistentDisk` declaration on `DCSIpHostnamePool` is available through YAML only and is not exposed in the web UI. With ACP `v4.4` or later, `Self-built VIP` is also supported through the YAML workflow.

> **INFO**
>
> The web UI provides a guided workflow with validation, while YAML offers more automation flexibility.

TOC

- Prerequisites
  1. Infrastructure Resources
  2. Required Plugin Installation
  3. Virtual Machine Template Preparation
  4. Network Connectivity
  5. LoadBalancer Configuration
  6. Workload Cluster Image Registry
  7. Node Labels and Taints
- Using the Web UI
  - Creation Workflow

- Step 1: Basic Info
- Step 2: Control Plane Node Pool
- Step 3: Worker Node Pools
- Step 4: Networking
- Step 5: Review

Using YAML

- Cluster Creation Workflow
- Configuration Workflow
- Resolving Placeholder Values
- Magic-Token Placeholders
- Network Planning and Load Balancer
- Configure KubeadmControlPlane
- Configure DCSCluster
  - Self-built VIP behavior and limitations
  - Verify Self-built VIP
  - Single-Control-Plane (No External LB) Layout
- Configure Cluster
- Deploy Worker Nodes
  - Step 1: Configure IP-Hostname Pool
  - Step 2: Configure Machine Template
  - Persistent Disks Managed by the IP Pool
  - Additional NICs Managed by the IP Pool
  - Step 3: Configure Bootstrap Template
  - Step 4: Configure Machine Deployment

Cluster Verification

- Using the Console
- Using kubectl
- Verify Control Plane HA
- Expected Results

Appendix

- Complete KubeadmControlPlane Configuration
- How the Encryption Provider Configuration Is Delivered

Next Steps

Troubleshooting

Prerequisites

Before creating clusters, ensure all of the following prerequisites are met:

1. Infrastructure Resources

Configure the following infrastructure resources before creating a cluster:

- Cloud Credential - DCS platform access information
- IP Pool - Network configuration for cluster nodes, additional NICs, and any IP-slot persistent disks such as `/var/cpaas`
- Machine Template - VM specifications for control plane and worker nodes, excluding pool-managed persistent disks

See Infrastructure Resources for Huawei DCS for detailed configuration instructions.

2. Required Plugin Installation

Before you create the first workload cluster on Huawei DCS, upload and install the following plugins on the `global` cluster. This applies regardless of the IaaS platform that the `global` cluster itself runs on:

- **Alauda Container Platform Cluster Bootstrap Provider**
- **Alauda Container Platform Infrastructure Provider for Huawei DCS**

For the procedure, see Install the Provider.

3. Virtual Machine Template Preparation

For Kubernetes installation, you must:

- Upload the Alauda OS image to the DCS platform. Use the Alauda OS image published for the ACP release installed on the cluster; see Downloading Alauda OS.
- Create a virtual machine template based on this image
- Ensure the template includes all necessary Kubernetes components
- Use DCS VM templates `4.2.1` or later if you plan to use persistent disks, because safe shutdown and disk detach depend on guest tools
- Use one-by-one replacement for any cluster that will rely on pool-managed persistent disks. Keep `maxSurge: 0` on the control plane and on worker node pools.

For details on the Kubernetes components included in each VM image, see OS Support Matrix.

4. Network Connectivity

Global cluster nodes must be able to reach the DCS platform at two distinct destinations:

From	To	Port	Purpose
Global cluster node	DCS VRM virtual IP	TCP/7443	DCS REST API. Covers cluster lifecycle calls and the first step of file uploads (`applyUpload`).
Global cluster node	DCS physical host MGMT IP (every host that may receive a clone)	DCS-returned port; typically TCP/8443; confirm with the DCS administrator	File-stream upload of the Ignition ISO. The provider streams the file to the URL returned by the `applyUpload` response.

File upload is a two-step flow: the provider calls `applyUpload` on the VRM virtual IP (TCP/7443); the DCS platform responds with a URL that points to a specific physical host's management IP and port; the provider then streams the file to that URL. Both destinations must be reachable end-to-end before cluster creation.

If global cluster nodes use a multi-NIC layout (for example, one NIC on the ACP cluster network and another NIC on the customer's management network where DCS is deployed),

ensure that **both** destinations — the VRM virtual IP and every DCS physical host MGMT IP — are routable from the appropriate NIC.

Requirement: Connectivity to both destinations is mandatory for cluster creation and management.

5. LoadBalancer Configuration

Configure a stable Kubernetes API endpoint before creating the cluster. The endpoint distributes or directs API server traffic for control plane access.

Mode	Version	Who manages it	When to use
External LoadBalancer	All supported versions	You provision and maintain the load balancer before cluster creation.	Use this when your environment already has a hardware or software load balancer.
`Self-built VIP`	ACP `v4.4+`	The DCS provider creates and reconciles the provider-managed `alive` `ModuleInfo`.	Use this when you need a highly available control plane endpoint but no external load balancer is available.
Single control plane direct endpoint	All supported versions	No load balancer is created. The API endpoint is the single control plane node IP.	Use this only for development or PoC clusters with one control plane replica.

For external LoadBalancer requirements, follow Plan the Control Plane Endpoint. For a single-control-plane deployment that has no load balancer in front of the API server, see Single-Control-Plane (No External LB) Layout.

6. Workload Cluster Image Registry

Decide which registry this cluster pulls its platform components from — Kube-OVN and the platform modules that Alauda Container Platform installs on the cluster. This choice does not affect the images kubeadm manages, which come from

`KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.imageRepository`.

Option	What you configure	When to use it
Registry of the `global` cluster	Nothing. The cluster inherits it.	Workload nodes reach the registry the `global` cluster uses with acceptable throughput.
Dedicated registry	A RegistryCredential Secret in `cpaas-system`, referenced from the `Cluster` by the `cpaas.io/registry-reference` label.	Workload nodes sit in a different region or network segment and should pull from a registry close to them.

Either way, the registry must already hold Alauda Container Platform Core for the target version plus the packages for every platform plugin you intend to install on this cluster, and every workload cluster node must reach its address.

For the dedicated-registry procedure and its constraints, see Choose the Image Registry for a Workload Cluster.

> **WARNING**
>
> The registry binding is read only when the cluster is first created. Changing or removing `cpaas.io/registry-reference` on an existing `Cluster` does not move that cluster to another registry. A dedicated registry is configured through YAML only; the web UI workflow always uses the registry of the `global` cluster.

7. Node Labels and Taints

To give the nodes of a pool their own role label and taint — for example to reserve dedicated pools for monitoring, logging, or ingress — declare them on the pool's resources as described in Infra Nodes. Plan them before you create the pools. This applies to every node pool, whether the cluster is created through the web UI or with YAML; the web UI node pool steps do not set labels or taints.

Using the Web UI

> **INFO**
>
> **Fleet Essentials upgrade boundary**
>
> Fleet Essentials 1.0.4 and later can request the ACP 4.3-and-later Distribution Version upgrade through CVO. It does not perform the DCS Kubernetes and Alauda OS replacement. Use Upgrading Kubernetes on Huawei DCS for that Phase 2 YAML procedure. Cluster creation through Fleet Essentials is unaffected by this boundary.

Requirement: This workflow requires Fleet Essentials. If you use pool-managed persistent disks, configure `DCSIpHostnamePool.spec.pool[].persistentDisk` through YAML because the web UI does not expose that field.

If the new cluster will rely on pool-managed persistent disks, create or update the backing `DCSIpHostnamePool` with YAML and then use the web UI for the rest of the cluster workflow.

Creation Workflow

The cluster creation follows a 5-step wizard:

```
Step 1: Basic Info
  ↓
Step 2: Control Plane Node Pool
  ↓
Step 3: Worker Node Pools
  ↓
Step 4: Networking
  ↓
Step 5: Review
```

Navigation: Clusters → Clusters → Create Cluster → Select Huawei DCS

Step 1: Basic Info

Field	Type	Required	Description
Infrastructure Credential	dropdown	Yes	Select an existing Cloud Credential
Name	text	Yes	Unique cluster identifier (lowercase letters, numbers, hyphens)
Display Name	text	No	Custom description for easy identification
Distribution Version	readonly	-	ACP version (matches the `global` cluster)
Kubernetes Version	readonly	-	Determined by Distribution Version
Cluster API Address	text	Yes	Format: `https://<load-balancer-address>:6443`. For `Self-built VIP`, reserve the VIP and use the YAML path in this document.

Prerequisites Check:

Before creating a cluster, ensure:

- DCS VM Templates exist in the DCS platform, and the Alauda OS version matches the Kubernetes version
- The Kubernetes API endpoint is ready. For an external LoadBalancer, create it before cluster creation. For `Self-built VIP`, reserve the VIP and use the YAML path.

Version Constraint: Only the latest Kubernetes version supported by the platform can be created.

Step 2: Control Plane Node Pool

The control plane node pool is fixed at 3 replicas for high availability.

Field	Type	Required	Description
Machine Template	dropdown	Yes	Filter templates by Type: Control Plane and compatible Kubernetes version
Replicas	readonly	-	Fixed at 3
SSH Authorized Keys	text	No	Add multiple SSH public keys for node access

Validation: The associated IP Pool must have sufficient available IP addresses (≥ 3).

Step 3: Worker Node Pools

You can add multiple worker node pools. Each pool has the following configuration:

Field	Type	Required	Description
Pool Name	text	Yes	Unique identifier for this node pool
Machine Template	dropdown	Yes	Filter templates by Type: Worker Node and compatible Kubernetes version
Replicas	number	Yes	Default: 3
Max Surge	number	No	Default: 0, must be ≥ 0. Keep this value at `0` if the node pool will use pool-managed persistent disks
Max Unavailable	number	No	Default: 1, must be ≥ 0. When maxSurge = 0, maxUnavailable must be > 0 and ≤ Replicas
SSH Authorized Keys	text	No	Add multiple SSH public keys

Validation Rules:

- Pool names must be unique within the cluster
- IP Pool must have sufficient available IP addresses (≥ Replicas)
- maxSurge and maxUnavailable must satisfy the constraint: if maxSurge = 0, then maxUnavailable > 0
- If the cluster will rely on pool-managed persistent disks, keep `maxSurge = 0` so nodes are replaced one by one during future upgrades

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

Step 4: Networking

Field	Type	Required	Description
Pods CIDR	CIDR	Yes	Pod network address range
Services CIDR	CIDR	Yes	Service network address range
Join CIDR	CIDR	Yes	Kube-OVN join CIDR parameter

Validation: Pods CIDR and Services CIDR must not overlap.

Step 5: Review

Review all configuration settings before creating the cluster:

Basic Info:

- Name, Display Name, Infrastructure Credential
- Distribution Version, Kubernetes Version
- Cluster API Address

Control Plane Node Pool:

- Machine Template with VM Template Name, OS Version, Kubernetes Version
- CPU, Memory, Replicas, SSH Keys

Worker Node Pools (list view):

- Pool Name, Machine Template, Replicas
- Max Surge, Max Unavailable, SSH Keys

If the cluster will rely on pool-managed persistent disks, keep **Max Surge** set to `0` for worker node pools.

Networking:

- Pods CIDR, Services CIDR, Join CIDR

Click **Create** to start the cluster creation process.

Using YAML

Cluster Creation Workflow

When using YAML, you create Cluster API resources in the `global` cluster to provision infrastructure and bootstrap a functional Kubernetes cluster.

> **WARNING**
>
> **Important Namespace Requirement**
>
> To ensure proper integration as workload clusters, all resources must be deployed in the `cpaas-system` namespace. Deploying resources in other namespaces may result in integration issues.

> **WARNING**
>
> **Workload Cluster Naming**
>
> The workload `cluster-name` **must not** be `global`. That name is reserved for the `global` cluster, and reusing it causes the workload cluster's resources to collide with `global` cluster resources in `cpaas-system`. The `global-` prefix is reserved for resources owned by the

`global` cluster's DR workflow; see Common Prerequisites. Do not use `global-` for workload-cluster resources, because failover operations can select those resources as if they belonged to the `global` cluster.

As a convention, keep the CAPI `Cluster` and provider cluster resource (`DCSCluster`) named exactly `<cluster-name>`, and prefix non-root CAPI and provider resources (`KubeadmControlPlane`, `KubeadmConfigTemplate`, `MachineDeployment`, machine templates, IP/hostname pools, etc.) with `<cluster-name>-` — for example, the example manifests use `<cluster-name>-kcp`. This is a recommendation rather than a controller-enforced rule, but it prevents same-namespace collisions when multiple workload clusters live in `cpaas-system` and makes resource ownership obvious during operations.

Configuration Workflow

Follow these steps in order to provision a functional cluster (control plane **and** worker nodes):

1. Configure `KubeadmControlPlane` (control-plane spec and kubeadm bootstrap).
2. Configure `DCSCluster` (infrastructure binding and load balancer reference).
3. Create the `Cluster` resource (top-level CAPI object that links the two above).
4. Configure worker resources: `KubeadmConfigTemplate` (worker bootstrap), worker `DCSMachineTemplate`, worker `DCSIpHostnamePool`, and `MachineDeployment`. **The control plane alone is not a usable cluster.** See Deploy Worker Nodes for the four worker YAML resources.

Note: Infrastructure resources (Secret, control-plane `DCSIpHostnamePool`, control-plane `DCSMachineTemplate`) should be configured separately. See Infrastructure Resources for Huawei DCS for instructions.

If you need any disk to survive rolling replacement, declare it in the matching `DCSIpHostnamePool.spec.pool[].persistentDisk` entry. This includes the platform-required `/var/cpaas` disk. The provider creates new persistent volumes as independent persistent normal volumes. When the DCS environment requires an explicit thin-provisioning setting, set `persistentDisk[].isThin`; if you omit it, the provider does not send `isThin` and DCS uses the platform default.

If the cluster needs additional NICs, declare them in `DCSIpHostnamePool.spec.pool[].additionNic[]` before the corresponding Machines are created. The provider applies additional NICs only during new VM creation. Existing VMs do not receive hot-added NICs when the Pool is edited later.

Resolving Placeholder Values

The example manifests below use `<placeholder>` syntax for values that are environment-specific. Several of these have a canonical source of truth that you should query rather than hand-pick:

Placeholder	Source of truth
`<control-plane-kubernetes-version>` / `<worker-kubernetes-version>`	A `cpaas.io/dcs-vm-template` C `cpaas-system` namespace, one p version.
`<dns-image-tag>`	Same ConfigMap, `data.corednsT`
`<etcd-image-tag>`	Same ConfigMap, `data.etcdTag`
`<vm-template-name>` (in `DCSMachineTemplate.spec.template.spec.vmTemplateName`)	The ConfigMap's `cpaas.io/dcs-v` value (must match the VM template the DCS platform).
`<ssh-authorized-keys>`	Operator-supplied. Each entry is a format public key (`ssh-ed25519 A AAAA…`). The field is required by the must be non-empty. For test or PoC need interactive SSH, supply any s public key (you do not need to hold key); for production, use the operat
`<auth-secret-name>`	The Secret you authored in Infrastr Credentials.

Placeholder	Source of truth
`<cluster-name>`	Operator-chosen; must satisfy DNS be `global` (that name is reserved cluster). Reused across `Cluster` the `cluster.x-k8s.io/cluster-n` `Machine`. The `KubeadmControlP` prefixed form `<cluster-name>-kc` Cluster Naming for the full conventi
`<load-balancer-ip>`	Operator-supplied endpoint clients cluster API server. For `type: exte` external load balancer VIP (an IP a names are not supported for the clu single-control-plane clusters with no the IP of the sole control plane node plane layout).
`<self-built-vip>`	Operator-supplied IPv4 VIP for `Se` Reserve this address before cluster be assigned to a node primary IP o `DCSIpHostnamePool.spec.pool[]`
`<pods-cidr>` / `<services-cidr>` / `<kube-ovn-join-cidr>`	Operator-supplied. Must not overla network, the `global` cluster's CID CAPI cluster's CIDRs you intend to them empty falls back to the kube-c with the `global` cluster, which is for production: explicit CIDRs avoid multiple workload clusters live on th cluster.

Magic-Token Placeholders

A few values in the example manifests look like placeholders but are **actually literal tokens** substituted by the DCS Provider at machine-join time. Leave them exactly as written:

Literal token	Meaning	Substituted by
`PROVIDER_ID`	The per-Machine provider ID (e.g. `dcs://<dcsmachine-name>`).	DCS Provider — overwritten in the generated kubelet config before `kubeadm init` / `kubeadm join` runs.
`NODE_IP`	The node IP allocated from the `DCSIpHostnamePool` entry attached to this Machine.	DCS Provider — overwritten with the value of `DCSIpHostnamePool.spec.pool[].ip`.

Replacing or quoting these tokens manually breaks the join flow and produces nodes that never register with the control plane.

Network Planning and Load Balancer

Before creating control plane resources, plan the network architecture and choose the control plane endpoint mode.

Requirements:

- **Network segmentation**: Plan IP address ranges for control plane nodes
- **Additional NICs**: If nodes require storage, management, or isolated application networks, plan `DCSIpHostnamePool.spec.pool[].additionNic[]` values for each IP slot, including DVS and Port Group names
- **API endpoint mode**: Use an external LoadBalancer, or use `Self-built VIP` with ACP `v4.4+`
- **API server address**: Prepare a stable VIP or load balancer address for the Kubernetes API server
- **Connectivity**: Ensure network connectivity between all components

For `Self-built VIP`, validate that the target DCS port group allows VIP ownership, GARP or ARP updates, and VRRP traffic between the control plane VMs. The VIP must be in the same Layer 2 network as the NIC that will hold it. For an external LoadBalancer, validate the Layer 4 listener, TCP `6443` backends, HTTPS `/healthz` checks, backend ownership, and endpoint reachability in Plan the Control Plane Endpoint.

Configure KubeadmControlPlane

The KubeadmControlPlane resource defines the control plane configuration including Kubernetes version, node specifications, and bootstrap settings.

Kubernetes 1.35 kubelet settings

`imagePullCredentialsVerificationPolicy: NeverVerify` is required only starting with Kubernetes 1.35. Omit this parameter when creating a cluster with Kubernetes 1.34 or earlier. If `contentFrom.secret` supplies the patch, verify that the referenced Secret key contains this parameter for Kubernetes 1.35.

TIP

Full Configuration Reference

The example below truncates long configuration files for readability. For the complete configuration (including default audit policies, admission controls, and file contents), refer to the Complete KubeadmControlPlane Configuration in the Appendix.

`kubeadmcontrolplane.yaml`

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-kcp
  namespace: cpaas-system
  annotations:
    controlplane.cluster.x-k8s.io/skip-coredns: ""
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0 # Required when the cluster relies on pool-managed pers
istent disks
  kubeadmConfigSpec:
    users:
    - name: boot
      sshAuthorizedKeys:
      - "<ssh-authorized-keys>"
    format: ignition
    files:
    - path: /etc/kubernetes/admission/psa-config.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        # ... (Admission Configuration Content) ...
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.json
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "_comment": "... (Kubelet Configuration Content) ..."
        }
    # ... (other files) ...
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
```

```
          imageTag: <etcd-image-tag>
      # ... (apiServer, controllerManager, scheduler) ...
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-ip: NODE_IP
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: DCSMachineTemplate
      name: <cp-dcs-machine-template-name>
  replicas: 3
  version: <control-plane-kubernetes-version>
```

Parameter Descriptions:

Parameter	Type	Description	Re
`.spec.kubeadmConfigSpec`	object	kubeadm bootstrap provider startup parameters	Yes
`.spec.machineTemplate.infrastructureRef`	object	DCSMachineTemplate reference	Yes

Parameter	Type	Description	Re
`.spec.replicas`	int	Control plane replica count. Must satisfy `1 ≤ replicas ≤ IP Pool size`. Set to `1` for development / PoC single-control-plane deployments (see Single-control-plane layout); a cluster with a single control plane node cannot be upgraded. Production usually uses `3` for HA.	Yes
`.spec.version`	string	Kubernetes version (must match VM template — see Resolving Placeholder Values)	Yes

For component versions (e.g., `<dns-image-tag>`, `<etcd-image-tag>`), refer to OS Support Matrix.

Configure DCSCluster

DCSCluster is the infrastructure cluster declaration that references the load balancer and DCS platform credentials.

Use `type: external` when you manage the load balancer outside the cluster.

dcscluster.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSCluster
metadata:
  name: "<cluster-name>"
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    host: <load-balancer-ip>
    port: 6443
    type: external
  credentialSecretRef:
    name: <auth-secret-name>
  controlPlaneEndpoint:
    host: <load-balancer-ip>
    port: 6443
  # Optional. Enable only when the target DCS compute cluster has DRS ena
bled.
  controlPlaneHA:
    enabled: true
  networkType: kube-ovn
  site: <site>
```

With ACP `v4.4+`, use `type: internal` to enable `Self-built VIP`:

dcscluster-self-built-vip.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSCluster
metadata:
  name: "<cluster-name>"
  namespace: cpaas-system
spec:
  controlPlaneLoadBalancer:
    host: <self-built-vip>
    port: 6443
    type: internal
    vrid: 42
    interface: eth0
  credentialSecretRef:
    name: <auth-secret-name>
  controlPlaneEndpoint:
    host: <self-built-vip>
    port: 6443
  networkType: kube-ovn
  site: <site>
```

Parameter Descriptions:

Parameter	Type	Description
`.spec.controlPlaneLoadBalancer`	object	Control plane API server exposure method
`.spec.controlPlaneLoadBalancer.type`	string	`external` uses an operator managed load balancer. `internal` enables `Self-built VIP` and requires ACP `v4.4+`.
`.spec.controlPlaneLoadBalancer.host`	string	For `external`, the external load balancer VIP (an IP address). For `internal`, the reserved IPv4 VIP.
`.spec.controlPlaneLoadBalancer.port`	int	Kubernetes API server port. The valid range is `1-65535`.

Parameter	Type	Description
`.spec.controlPlaneLoadBalancer.vrid`	int	Keepalived VRID used by `Self-built VIP`. Set a value from `1` to `255` when `type` is `internal`. This field is ignored when `type` i `external`.
`.spec.controlPlaneLoadBalancer.interface`	string	Optional NIC name used by the bootstrap VIP and `alive` Leave it empty to detect the NIC by node IP. Set it explicitl for multi-NIC clusters when th VIP must be held by a specifi interface.
`.spec.credentialSecretRef.name`	string	DCS authentication Secret name. The Secret defines whether DCS Provider authenticates as an interface interconnection user (default) or a domain user — see Credential User Types.
`.spec.controlPlaneEndpoint`	object	Kubernetes API endpoint that CAPI and kubeadm use. Set i to the same host and port as `controlPlaneLoadBalancer`
`.spec.controlPlaneHA.enabled`	bool	Enables a provider-managed DCS DRS mutual-exclusion rule for control-plane VMs. Se it to `true` only when the target DCS compute cluster has DRS enabled and has enough hosts and capacity fo cross-host placement.
`.spec.networkType`	string	Currently only supports "kube ovn"

Parameter	Type	Description
`.spec.site`	string	DCS platform site ID

`controlPlaneHA` is optional. When enabled, the provider creates and maintains one DRS `ruleType=2` mutual-exclusion rule for the current control-plane VMs in this workload cluster. DCS performs the actual placement and runtime migration. The provider does not run DRS or apply DRS recommendations. If the rule is maintained but placement has not converged, wait for the DCS scheduling mechanism or trigger/apply DRS from the DCS platform side. For infrastructure prerequisites, see Cross-Host High Availability for Control Plane.

Self-built VIP behavior and limitations

`Self-built VIP` requires ACP `v4.4` or later. Plan this mode before cluster creation.

Recommendation: Use `type: internal` only for new highly available clusters with three control plane replicas. For existing clusters that use `type: external`, keep the external load balancer unless you have a tested migration plan. `controlPlaneHA.enabled` is an independent DRS placement feature and is not required for Self-built VIP.

The DCS provider handles `Self-built VIP` in two phases:

1. During bootstrap, the provider injects `dcs-bootstrap-vip.service` only on the first `kubeadm init` control plane node. This service adds `<self-built-vip>/32` before `kubeadm` starts.
2. After the control plane nodes register, the provider creates or updates the provider-managed `alive` `ModuleInfo`. The platform plugin flow renders the workload `AppRelease`, and `alive` installs the VIP runtime on the control plane nodes.

Important constraints:

- Both modes accept only an IP address: `type: external` takes the external load balancer VIP, `type: internal` the reserved IPv4 VIP. Domain names are not supported for the cluster endpoint in either mode.
- The VIP must be excluded from `DCSIpHostnamePool` primary IPs and additional NIC IPs.
- The target DCS port group must allow VIP ownership, GARP or ARP updates, and VRRP traffic.

- `interface` must be a Linux interface name of 15 characters or fewer. Use it when the VIP must be bound to a specific NIC, such as `eth0` or `eth1`.
- `type`, `host`, `port`, `vrid`, and `interface` are not intended to be changed after the cluster control plane is initialized.
- The DCS provider creates or patches the `alive` `ModuleInfo`; the platform plugin flow creates the workload `AppRelease` and runtime resources.
- The DCS provider does not inject or persist IPVS kernel modules or sysctl parameters. Use the VM template and plugin package delivered with the supported ACP release.

Verify Self-built VIP

If you set `DCSCluster.spec.controlPlaneLoadBalancer.type` to `internal`, verify the provider-managed VIP path after the cluster is created.

```
kubectl -n cpaas-system get dcscluster <cluster-name> \
  -o jsonpath='{.spec.controlPlaneLoadBalancer.type}{" "}{.spec.controlPl
aneEndpoint.host}{":"}{.spec.controlPlaneEndpoint.port}{"\n"}'
kubectl -n cpaas-system get cluster <cluster-name> \
  -o jsonpath='{.spec.controlPlaneEndpoint.host}{":"}{.spec.controlPlaneE
ndpoint.port}{"\n"}'
kubectl -n cpaas-system get moduleinfo \
  -l cpaas.io/cluster-name=<cluster-name>,cpaas.io/module-name=alive
kubectl --kubeconfig <workload-kubeconfig> -n cpaas-system get apprelease
alive
kubectl --kubeconfig <workload-kubeconfig> -n kube-system get pods -l app
=alive -o wide
kubectl --kubeconfig <vip-kubeconfig> get --raw=/version
```

Expected results are the configured VIP endpoint, a provider-created `alive` `ModuleInfo`, a synced and healthy `alive` `AppRelease`, Ready `alive` pods on every control plane node, and a successful `/version` request through the VIP.

Single-Control-Plane (No External LB) Layout

For development, PoC, or any deployment where the control plane has only **one** replica (`KubeadmControlPlane.spec.replicas: 1`), you do not have a real load balancer in front of the API server. Two fields nevertheless still require a value:

- `.spec.controlPlaneLoadBalancer.host` and `.spec.controlPlaneEndpoint.host` — set both to **the IP of the sole control plane node** (the same IP allocated to that node in the control-plane `DCSIpHostnamePool`).
- `.spec.controlPlaneLoadBalancer.type` — keep the schema-compatible value `external`; no load-balancing component is created in this layout.

Concretely:

```
spec:
  controlPlaneLoadBalancer:
    host: 10.226.82.150     # same IP as the control plane node from the
IP pool
    port: 6443
    type: external
  controlPlaneEndpoint:
    host: 10.226.82.150     # same as above
    port: 6443
```

This layout has no HA — losing the single control plane node makes the cluster API unreachable until the node is recovered. Do not use `Self-built VIP` for this layout. For production, use `replicas: 3` with an external LoadBalancer or `Self-built VIP`.

A cluster with fewer than three control plane replicas cannot be upgraded.

Configure Cluster

The Cluster resource declares the cluster and references the control plane and infrastructure resources.

`cluster.yaml`

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: DCSCluster
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>
  labels:
    cluster-type: DCS
    # Optional. Bind this cluster to a dedicated image registry instead of the
    # registry of the `global` cluster. Read only at cluster creation.
    # cpaas.io/registry-reference: <registry-credential-name>
  name: <cluster-name>
  namespace: cpaas-system
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - <pods-cidr>
    services:
      cidrBlocks:
      - <services-cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster-name>-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: DCSCluster
    name: <cluster-name>
```

Parameter Descriptions:

Parameter	Type	Description	Requir
`.spec.clusterNetwork.pods.cidrBlocks`	[]string	Pod CIDR. Optional in the CAPI schema but **recommended**	Recom

Parameter	Type	Description	Requir
		to set explicitly so multiple CAPI clusters can co-exist without overlap. If unset, kube-ovn falls back to a default that may conflict with another cluster on the same global.	
`.spec.clusterNetwork.services.cidrBlocks`	[]string	Service CIDR. Same recommendation as the pod CIDR — set explicitly to avoid collisions across clusters.	Recom
`.spec.controlPlaneRef`	object	Control plane reference	Yes
`.spec.infrastructureRef`	object	Infrastructure cluster reference	Yes

Cluster Annotations:

The example above shows three annotations, but a complete `Cluster` resource carries a few more. The table below lists the annotations the **operator authors** (some others are written by ACP controllers and you should not pre-set them):

Annotation	Required	Value source
`capi.cpaas.io/resource-group-version`	Yes	Literal `infrastructure.cluster.x-k8s.io/v1`

Annotation	Required	Value source
`capi.cpaas.io/resource-kind`	Yes	Literal `DCSCluster`
`capi.cpaas.io/kubernetes`	Yes	Same value as `KubeadmControlPlane.spec.ve` `MachineDeployment.spec.template.spec.ver`
`cpaas.io/kube-ovn-join-cidr`	Yes	Operator-chosen `/16` CIDR, must not overlap v service CIDRs or any other cluster's join CIDR.
`cpaas.io/kube-ovn-version`	Yes	The kube-ovn release the `global` cluster ships it from the `cpaas.io/kube-ovn-version` anno `global` CAPI `Cluster` (`kubectl get clust -n cpaas-system -o jsonpath='{.metadata.annotations.cpaas\.i version}'`). The same value lives on every hea workload cluster on the same `global` cluster, s also read it from any of those if they exist.
`cpaas.io/registry-address`	Yes	The image registry the `global` cluster uses (ty `<registry-host>:11443`). Read it from the `cpaas.io/registry-address` annotation on th CAPI `Cluster` (`kubectl get cluster globa system -o jsonpath='{.metadata.annotations.cpaas\.i address}'`).

Annotation	Required	Value source
`cpaas.io/nodes-mode`	Yes	Literal `self-managed` for clusters that DCS Pro provisions.
`cpaas.io/os-family`	KubeOS only	Literal `kubeos` when the cluster's DCS VM tem from a KubeOS image. Omit it for Alauda OS: DC handles an unset value, or any value other than `slemicro`.

ACP controllers may write additional read-only annotations (`cpaas.io/cpu-cores-number`, `cpaas.io/memories`, `cpaas.io/nodes-number`, and so on) after the cluster is up — these are computed and **must not** be pre-set in the YAML you apply.

Cluster Labels:

Label	Required	Value source	Purpose
`cluster-type`	Yes	Literal `DCS`.	Marks the cluster as provisioned by DCS Provider.
`cpaas.io/registry-reference`	No	Name of a RegistryCredential Secret in `cpaas-system`.	Binds this cluster to a dedicated image registry. Omit it to inherit the registry of the `global` cluster. Read only at cluster creation — see Choose the Image Registry for a Workload Cluster.

Deploy Worker Nodes

The control plane alone is not a usable cluster. Create the worker resources in the following order:

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

Step 1: Configure IP-Hostname Pool

The IP-Hostname Pool defines the network configuration for worker node virtual machines. You must plan and configure the IP addresses, hostnames, DNS servers, additional NICs, and other network parameters before deployment.

On Huawei DCS, the IP pool is also where you declare persistent disks that must survive VM replacement. Use `persistentDisk` for the platform-required `/var/cpaas` disk and for any other worker-node disk that must be preserved during delete-recreate operations.

When a node template declares a dedicated data disk at `/var/lib/containerd`, the DCS provider's bootstrap configuration stores pod logs under `/var/lib/containerd/pods` and links `/var/log/pods` to that path. Keep the disk mounted at `/var/lib/containerd` when replacing the node if you need the pod-log data to remain on the containerd data disk.

> **WARNING**
>
> **Pool Size Requirement** The pool must include at least as many entries as the number of worker nodes you plan to deploy. Insufficient entries will prevent node deployment.

Example:

Create a `DCSIpHostnamePool` named `<cluster-name>-worker-pool`:

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSIpHostnamePool
metadata:
  name: <cluster-name>-worker-pool
  namespace: cpaas-system
spec:
  pool:
  - ip: "<worker-ip-1>"
    mask: "<worker-cidr-prefix-length>"
    gateway: "<worker-gateway>"
    dns: "<worker-dns>"
    hostname: "<worker-hostname-1>"
    machineName: "<worker-machine-name-1>"
    additionNic:
    - ip: "<worker-additional-ip-1>"
      mask: "<worker-additional-cidr-prefix-length>"
      gateway: "<worker-additional-gateway>"
      dns: "<worker-additional-dns>"
      dvSwitchName: "<additional-dvs>"
      portGroupName: "<additional-port-group>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
  - ip: "<worker-ip-2>"
    mask: "<worker-cidr-prefix-length>"
    gateway: "<worker-gateway>"
    dns: "<worker-dns>"
    hostname: "<worker-hostname-2>"
    machineName: "<worker-machine-name-2>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
  - ip: "<worker-ip-3>"
    mask: "<worker-cidr-prefix-length>"
    gateway: "<worker-gateway>"
    dns: "<worker-dns>"
    hostname: "<worker-hostname-3>"
```

```
    machineName: "<worker-machine-name-3>"
    persistentDisk:
    - slot: 0
      quantityGB: 40
      datastoreClusterName: <datastore-cluster-name>
      path: /var/cpaas
      format: xfs
```

Key parameters:

Parameter	Type	Description	Require
`.spec.pool[].ip`	string	IP address for the worker virtual machine	Yes
`.spec.pool[].mask`	string	Subnet mask in CIDR prefix-length format without `/`, for example `24`	Yes
`.spec.pool[].gateway`	string	Gateway IP address	Yes
`.spec.pool[].dns`	string	DNS server IP addresses (semicolon-separated for multiple)	No
`.spec.pool[].machineName`	string	Virtual machine name in the DCS platform	No
`.spec.pool[].hostname`	string	Hostname for the virtual machine	No
`.spec.pool[].additionNic[]`	[]object	Additional NICs bound to this IP slot.	No

Parameter	Type	Description	Require
		Use this for storage, management, or isolated application networks.	
`.spec.pool[].additionNic[].ip`	string	Additional NIC IP address for the generated guest network configuration.	Recomm
`.spec.pool[].additionNic[].mask`	string	Additional NIC subnet mask in CIDR prefix-length format without `/`, for example `24`.	Recomm
`.spec.pool[].additionNic[].gateway`	string	Gateway for the additional NIC network. Confirm route behavior before setting this value.	No
`.spec.pool[].additionNic[].dns`	string	DNS server IPs for the additional NIC network, separated by semicolons when multiple values are required.	No
`.spec.pool[].additionNic[].dvSwitchName`	string	DCS distributed virtual switch	Yes*

Parameter	Type	Description	Require
		used to resolve the Port Group for the additional NIC.	
`.spec.pool[].additionNic[].portGroupName`	string	DCS Port Group used to attach the additional NIC.	Yes*
`.spec.pool[].persistentDisk[]`	[]object	Persistent disks bound to this IP slot. Use this for `/var/cpaas` and any disk that must survive node replacement.	No

*Fields marked `Yes*` are required for the provider to resolve the target network and attach the additional NIC. Fields marked `Recommended*` are needed for the generated static guest network configuration. The CRD does not enforce these fields.

Step 2: Configure Machine Template

The DCSMachineTemplate defines the specifications for worker node virtual machines, including VM templates, compute resources, storage configuration, and network settings.

The fields in this resource reference several Huawei DCS platform concepts (DCS VM Template, DCS VM Folder, DCS Datastore, and related items). For the definitions of those concepts and how they map to Cluster API resources, see Huawei DCS Concepts and Terminology.

WARNING

Required Disk Configurations The following disk mount points are mandatory. Do not remove them:

- System volume (`systemVolume: true`)
- `/var/lib/kubelet` - Kubelet data directory
- `/var/lib/containerd` - Container runtime data

Configure `/var/cpaas` in the IP pool as a persistent disk, not in `DCSMachineTemplate`.

You may add additional template disks, but these essential template disks must be preserved.

Example:

Create a `DCSMachineTemplate` named `<cluster-name>-worker-template` :

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: DCSMachineTemplate
metadata:
  name: <cluster-name>-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      vmTemplateName: <vm-template-name>
      location:
        type: folder
        name: <folder-name>
      vmConfig:
        dvSwitchName: <dv-switch-name> # Optional
        portGroupName: <port-group-name> # Optional
        dcsMachineCpuSpec:
          quantity: <worker-cpu>
        dcsMachineMemorySpec: # MB
          quantity: <worker-memory>
        dcsMachineDiskSpec: # GB
        - quantity: 0
          datastoreClusterName: <datastore-cluster-name>
          systemVolume: true
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/kubelet
          format: xfs
        - quantity: 100
          datastoreClusterName: <datastore-cluster-name>
          path: /var/lib/containerd
          format: xfs
      ipHostPoolRef:
        name: <cluster-name>-worker-pool
```

Key parameters:

Parameter	Type
`.spec.template.spec.vmTemplateName`	string

Parameter	Type
`.spec.template.spec.location`	object
`.spec.template.spec.location.type`	string
`.spec.template.spec.location.name`	string
`.spec.template.spec.vmConfig`	object
`.spec.template.spec.vmConfig.dvSwitchName`	string
`.spec.template.spec.vmConfig.portGroupName`	string

Parameter	Type
`.spec.template.spec.vmConfig.dcsMachineCpuSpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineMemorySpec.quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec`	[]object
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].quantity`	int
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].datastoreClusterName`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].systemVolume`	bool
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].path`	string
`.spec.template.spec.vmConfig.dcsMachineDiskSpec[].format`	string
`.spec.template.spec.ipHostPoolRef.name`	string

*Required when parent object is specified

Persistent Disks Managed by the IP Pool

Declare any upgrade-preserved disk in the matching `DCSIpHostnamePool.spec.pool[].persistentDisk` entry.

- Use this for `/var/cpaas`, which is required by the platform.

- Keep `DCSMachineTemplate` for the system disk and template-local disks that may be recreated with the VM.
- Choose a unique `slot` per IP entry. The controller uses `(ip, slot)` as the persistent-disk identity.
- On replacement nodes, the guest disk setup logic checks for an existing filesystem. If the disk is already formatted, it skips `mkfs` and mounts the disk directly.
- Persistent-disk workflows require one-by-one replacement, so keep `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0`.
- You can append new `persistentDisk` entries, but deleting existing entries is not supported. The controller attaches the newly added disk to the running VM on the DCS side, but it does not format or mount the disk inside the guest OS. Guest formatting and mounting take effect only after the VM is replaced and the replacement VM runs the generated disk setup during bootstrap.
- The controller creates new persistent volumes as independent persistent normal volumes. When it reuses an existing volume, it only requires the volume to be independent and persistent; it does not require a specific DCS volume type.
- Treat `format`, `options`, and `pciType` as immutable after creation.
- Use `isThin` only when the DCS environment requires an explicit thin-provisioning value for newly created persistent volumes. If omitted, the provider does not send `isThin` and DCS uses the platform default.
- Treat `quantityGB` and datastore changes as rollout-sensitive changes. The webhook performs best-effort validation against the DCS platform when it has enough cluster context. `isThin` is creation-only: changing it does not validate, reconcile, or convert existing volumes, and affects only persistent volumes created later.

To inspect the runtime state of persistent disks during node operations, check `status.persistentDiskStatus` on the pool:

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

Additional NICs Managed by the IP Pool

Declare worker additional NICs in `DCSIpHostnamePool.spec.pool[].additionNic[]`.

- The provider applies `additionNic[]` only when it creates a new VM.
- Existing worker VMs are not hot-updated when you edit the Pool.
- New worker Machines use the `additionNic[]` values from the Pool slot they claim.
- Each additional NIC should include both `dvSwitchName` and `portGroupName` so the provider can resolve the target DCS Port Group before VM clone.
- If you also use `persistentDisk[]`, keep `MachineDeployment.spec.strategy.rollingUpdate.maxSurge = 0` so fixed IP slots and preserved disks move one node at a time.
- If an additional NIC has a gateway, verify that the guest OS route table still prefers the intended primary network for the default route.

To inspect the runtime state of additional NICs, check `DCSMachine.status.additionalNic`:

```
kubectl -n cpaas-system get dcsmachine -l cluster.x-k8s.io/cluster-name=<cluster-name> \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.networkConfig.ip}{"\t"}{.status.additionalNic}{"\n"}{end}'
```

Step 3: Configure Bootstrap Template

The KubeadmConfigTemplate defines the bootstrap configuration for worker nodes, including user accounts, SSH keys, system files, and kubeadm join settings.

`imagePullCredentialsVerificationPolicy: NeverVerify` is required only starting with Kubernetes 1.35. Omit this parameter when creating workers with Kubernetes 1.34 or earlier. If `contentFrom.secret` supplies the patch, verify that the referenced `worker-kubelet-patch.json` contains this parameter for Kubernetes 1.35.

> **INFO**
>
> **Template Optimization** The template includes pre-optimized configurations for security and performance. Modify only the parameters that require customization for your environment.

Example:

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster-name>-worker-kct
  namespace: cpaas-system
spec:
  template:
    spec:
      format: ignition
      users:
      - name: boot
        sshAuthorizedKeys:
        - "<ssh-authorized-keys>"
      files:
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
        owner: "root:root"
        permissions: "0644"
        content: |
          {
            "apiVersion": "kubelet.config.k8s.io/v1beta1",
            "kind": "KubeletConfiguration",
            "imagePullCredentialsVerificationPolicy": "NeverVerify",
            "protectKernelDefaults": true,
            "staticPodPath": null,
            "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
            "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
            "streamingConnectionIdleTimeout": "5m",
            "clientCAFile": "/etc/kubernetes/pki/ca.crt"
          }
      preKubeadmCommands:
      - while ! ip route | grep -q "default via"; do sleep 1; done; echo
"NetworkManager started"
      - mkdir -p /run/cluster-api && restorecon -Rv /run/cluster-api
      - if [ -f /etc/disk-setup.sh ]; then bash /etc/disk-setup.sh; fi
      postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
      joinConfiguration:
        patches:
          directory: /etc/kubernetes/patches
        nodeRegistration:
          kubeletExtraArgs:
            node-ip: NODE_IP
```

```
            provider-id: PROVIDER_ID
            volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/v
olume/exec/"
            protect-kernel-defaults: "true"
```

`PROVIDER_ID` and `NODE_IP` are literal magic tokens that DCS Provider substitutes at machine-join time — keep them exactly as written. See Magic-Token Placeholders for the full list.

TIP

Alternative: reference a centrally managed Secret for the worker kubelet patch

The DCS Provider plugin ships a Secret named `dcs-kubernetes-<kubernetes-major-minor>-files` in the `cpaas-system` namespace (for example, `dcs-kubernetes-1.33-files` for Kubernetes 1.33). In addition to the control-plane files it serves to `KubeadmControlPlane` (see Complete KubeadmControlPlane Configuration), this Secret also contains a `worker-kubelet-patch.json` key suitable for the worker `KubeadmConfigTemplate`. When that Secret is present, you can replace the inline `files` entry above with a `contentFrom.secret` reference — this keeps the worker kubelet patch in sync with the installed plugin version and avoids manual updates on cluster upgrades.

```
files:
- contentFrom:
    secret:
      key: worker-kubelet-patch.json
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
  permissions: "0644"
```

The inline and Secret-referenced forms are functionally equivalent; the Secret form is the recommended path when the plugin Secret is available in the target cluster.

Step 4: Configure Machine Deployment

The MachineDeployment orchestrates the creation and management of worker nodes by referencing the previously configured DCSMachineTemplate and KubeadmConfigTemplate

resources. It manages the desired number of nodes and handles rolling updates.

Example:

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster-name>-md-0
  namespace: cpaas-system
spec:
  strategy:
    rollingUpdate:
      maxSurge: 0 # Required when this node pool relies on persistent dis
ks
      maxUnavailable: 1
    type: RollingUpdate
  clusterName: <cluster-name>
  replicas: 3
  selector:
    matchLabels: null
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: <cluster-name>
        pool.name: <cluster-name>-md-0
    spec:
      nodeDrainTimeout: 1m
      nodeDeletionTimeout: 5m
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster-name>-worker-kct
          namespace: cpaas-system
      clusterName: <cluster-name>
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: DCSMachineTemplate
        name: <cluster-name>-worker-template
        namespace: cpaas-system
      version: <worker-kubernetes-version>
```

Key parameters:

Parameter	Type	Description
`.spec.clusterName`	string	Target cluster name for node deployment
`.spec.replicas`	int	Number of worker nodes (must not exceed IP pool size)
`.spec.template.spec.bootstrap.configRef`	object	Reference to KubeadmConfigTemplate
`.spec.template.spec.infrastructureRef`	object	Reference to DCSMachineTemplate
`.spec.template.spec.version`	string	Kubernetes version (must match VM template)
`.spec.strategy.rollingUpdate.maxSurge`	int	Maximum nodes above desired during update. Keep this at `0` when the node pool relies on persistent disks
`.spec.strategy.rollingUpdate.maxUnavailable`	int	Maximum unavailable nodes during update. When persistent disks are used with `maxSurge = 0`, keep this value greater than `0` and within the replica count

Cluster Verification

After deploying all cluster resources, verify that the cluster has been created successfully and is operational.

Using the Console

1. Navigate to **Clusters** → **Clusters**
2. Locate your newly created cluster in the cluster list
3. Verify that the cluster status shows as **Running**
4. Check that all control plane and worker nodes are **Ready**

Using kubectl

Alternatively, verify the cluster using kubectl commands:

```
# Check cluster status
kubectl get cluster -n cpaas-system <cluster-name>

# Verify control plane
kubectl get kubeadmcontrolplane -n cpaas-system <cluster-name>-kcp

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

# Verify cluster deployment
kubectl get clustermodule <cluster-name> -o jsonpath='{.status.base.deployStatus}'
```

For clusters that use additional NICs, also verify that the provider recorded them on the matching `DCSMachine` objects:

```
kubectl -n cpaas-system get dcsmachine -l cluster.x-k8s.io/cluster-name=<cluster-name> \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.networkConfig.ip}{"\t"}{.status.additionalNic}{"\n"}{end}'
```

Verify Control Plane HA

If you enabled `DCSCluster.spec.controlPlaneHA.enabled`, inspect the `DCSCluster` condition first:

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{range .status.conditions[?(@.type=="ControlPlaneHARead
y")]}{.status}{" "}{.reason}{" "}{.message}{"\n"}{end}'
```

Interpret the condition as follows:

Status and reason	Meaning	Next step
`True` `ControlPlaneHAReady`	The provider-managed DRS rule exists and the provider has observed the current control-plane VMs on different DCS hosts.	No action is required.
`False` `ControlPlaneHAPending`	The provider is waiting for a recoverable state, such as enough VM URNs, enough members, or DCS-side placement.	Read the message. If it mentions placement, check DCS host capacity, DRS status, and whether runtime migration is blocked.
`False` `ControlPlaneHAFailed`	The provider could not query, validate, create, update, delete, or check the DRS rule.	Read the message, then check the DCS credential, DRS settings, rule-name conflicts, and controller logs if needed.

Inspect the observed rule and membership snapshot:

```
kubectl get dcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='rule={.status.controlPlaneHA.ruleName}{" index="}{.status.
controlPlaneHA.ruleIndex}{" cluster="}{.status.controlPlaneHA.clusterUrn}
{"\n"}{range .status.controlPlaneHA.members[*]}{.machineName}{" "}{.vmUr
n}{"\n"}{end}'
```

The `members[]` list contains the CAPI Machine name and DCS VM URN for each control-plane VM in the rule. The current DCS status snapshot does not include the DCS host name. If you need to confirm the exact host placement, query the VM details from the DCS platform or DCS API and compare the VM `hostName` or `hostUrn` values.

Expected Results

A successfully created cluster should show:

- Cluster status: **Running** or **Provisioned**
- All control plane machines: **Running**
- All worker nodes (if deployed): **Running**
- Kubernetes nodes: **Ready**
- Cluster Module Status: **Completed**
- For multi-NIC clusters, each created VM has the expected additional NICs in `DCSMachine.status.additionalNic`, and the guest OS shows the corresponding `ethN` interfaces.

Appendix

Complete KubeadmControlPlane Configuration

Below is the complete `KubeadmControlPlane` configuration, including all default audit policies, admission controls, and file contents.

The embedded kubelet patch includes `imagePullCredentialsVerificationPolicy: NeverVerify`, which is required only starting with Kubernetes 1.35. Remove this parameter for Kubernetes 1.34 or earlier.

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-kcp
  namespace: cpaas-system
  annotations:
    controlplane.cluster.x-k8s.io/skip-coredns: ""
    controlplane.cluster.x-k8s.io/skip-kube-proxy: ""
spec:
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0 # Required when the cluster relies on pool-managed pers
istent disks
  kubeadmConfigSpec:
    users:
    - name: boot
      sshAuthorizedKeys:
      - "<ssh-authorized-keys>"
    format: ignition
    files:
    - path: /etc/kubernetes/admission/psa-config.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: AdmissionConfiguration
        plugins:
        - name: PodSecurity
          configuration:
            apiVersion: pod-security.admission.config.k8s.io/v1
            kind: PodSecurityConfiguration
            defaults:
              enforce: "privileged"
              enforce-version: "latest"
              audit: "baseline"
              audit-version: "latest"
              warn: "baseline"
              warn-version: "latest"
            exemptions:
              usernames: []
              runtimeClasses: []
              namespaces:
```

```
            - kube-system
            - cpaas-system
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.json
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "protectKernelDefaults": true,
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
          "streamingConnectionIdleTimeout": "5m",
          "clientCAFile": "/etc/kubernetes/pki/ca.crt"
        }
    - path: /etc/kubernetes/audit/policy.yaml
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        omitStages:
        - "RequestReceived"
        rules:
        - level: None
          users:
          - system:kube-controller-manager
          - system:kube-scheduler
          - system:serviceaccount:kube-system:endpoint-controller
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
          - group: ""
            resources: ["endpoints"]
        - level: None
          nonResourceURLs:
          - /healthz*
          - /version
          - /swagger*
        - level: None
          resources:
          - group: ""
```

```
      resources: ["events"]
    - level: None
      resources:
      - group: "devops.alauda.io"
    - level: None
      verbs: ["get", "list", "watch"]
    - level: None
      resources:
      - group: "coordination.k8s.io"
        resources: ["leases"]
    - level: None
      resources:
      - group: "authorization.k8s.io"
        resources: ["subjectaccessreviews", "selfsubjectaccessreview
s"]
      - group: "authentication.k8s.io"
        resources: ["tokenreviews"]
    - level: None
      resources:
      - group: "app.alauda.io"
        resources: ["imagewhitelists"]
      - group: "k8s.io"
        resources: ["namespaceoverviews"]
    - level: Metadata
      resources:
      - group: ""
        resources: ["secrets", "configmaps"]
    - level: Metadata
      resources:
      - group: "operator.connectors.alauda.io"
        resources: ["installmanifests"]
      - group: "operators.katanomi.dev"
        resources: ["katanomis"]
    - level: RequestResponse
      resources:
      - group: ""
      - group: "aiops.alauda.io"
      - group: "apps"
      - group: "app.k8s.io"
      - group: "authentication.istio.io"
      - group: "auth.alauda.io"
      - group: "autoscaling"
      - group: "asm.alauda.io"
      - group: "clusterregistry.k8s.io"
```

```
          - group: "crd.alauda.io"
          - group: "infrastructure.alauda.io"
          - group: "monitoring.coreos.com"
          - group: "operators.coreos.com"
          - group: "networking.istio.io"
          - group: "extensions.istio.io"
          - group: "install.istio.io"
          - group: "security.istio.io"
          - group: "telemetry.istio.io"
          - group: "opentelemetry.io"
          - group: "networking.k8s.io"
          - group: "portal.alauda.io"
          - group: "rbac.authorization.k8s.io"
          - group: "storage.k8s.io"
          - group: "tke.cloud.tencent.com"
          - group: "devopsx.alauda.io"
          - group: "core.katanomi.dev"
          - group: "deliveries.katanomi.dev"
          - group: "integrations.katanomi.dev"
          - group: "artifacts.katanomi.dev"
          - group: "builds.katanomi.dev"
          - group: "versioning.katanomi.dev"
          - group: "sources.katanomi.dev"
          - group: "tekton.dev"
          - group: "operator.tekton.dev"
          - group: "eventing.knative.dev"
          - group: "flows.knative.dev"
          - group: "messaging.knative.dev"
          - group: "operator.knative.dev"
          - group: "sources.knative.dev"
          - group: "operator.devops.alauda.io"
          - group: "flagger.app"
          - group: "jaegertracing.io"
          - group: "velero.io"
            resources: ["deletebackuprequests"]
          - group: "connectors.alauda.io"
          - group: "operator.connectors.alauda.io"
            resources: ["connectorscores", "connectorsgits", "connectorso
cis"]
        - level: Metadata
    preKubeadmCommands:
    - while ! ip route | grep -q "default via"; do sleep 1; done; echo "N
etworkManager started"
    - mkdir -p /run/cluster-api && restorecon -Rv /run/cluster-api
```

```
    - if [ -f /etc/disk-setup.sh ]; then bash /etc/disk-setup.sh; fi
    postKubeadmCommands:
    - chmod 600 /var/lib/kubelet/config.yaml
    clusterConfiguration:
      imageRepository: cloud.alauda.io/alauda
      dns:
        imageTag: <dns-image-tag>
      etcd:
        local:
          imageTag: <etcd-image-tag>
      apiServer:
        extraArgs:
          audit-log-format: json
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "200"
          profiling: "false"
          audit-log-mode: batch
          audit-log-path: /etc/kubernetes/audit/audit.log
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          tls-cipher-suites: "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS
_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY130
5,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1
305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"
          encryption-provider-config: /etc/kubernetes/encryption-provide
r.conf
          admission-control-config-file: /etc/kubernetes/admission/psa-co
nfig.yaml
          tls-min-version: VersionTLS12
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
        extraVolumes:
        - name: vol-dir-0
          hostPath: /etc/kubernetes
          mountPath: /etc/kubernetes
          pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
          flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
      scheduler:
        extraArgs:
```

```
          bind-address: "::"
          tls-min-version: VersionTLS12
          profiling: "false"
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-ip: NODE_IP
          node-labels: "kube-ovn/role=master"
          provider-id: PROVIDER_ID
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
          protect-kernel-defaults: "true"
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: DCSMachineTemplate
      name: <cp-dcs-machine-template-name>
  replicas: 3
  version: <control-plane-kubernetes-version>
```

TIP

Alternative: reference a centrally managed Secret instead of inline content

The Alauda Container Platform Infrastructure Provider for Huawei DCS plugin ships a Secret named `dcs-kubernetes-<kubernetes-major-minor>-files` in the `cpaas-system` namespace (for example, `dcs-kubernetes-1.33-files` for Kubernetes 1.33). It contains the canonical

content of `psa-config.yaml`, `control-plane-kubelet-patch.json`, and `audit-policy.yaml`, and is updated together with each release.

When that Secret is present, you can replace the three inline `files` entries with `contentFrom.secret` references. Inline and Secret-referenced forms are functionally equivalent; using the Secret keeps file content aligned with the installed plugin version and avoids manual updates on cluster upgrades.

```
files:
- contentFrom:
    secret:
      key: psa-config.yaml
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/admission/psa-config.yaml
  permissions: "0644"
- contentFrom:
    secret:
      key: control-plane-kubelet-patch.json
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
  permissions: "0644"
- contentFrom:
    secret:
      key: audit-policy.yaml
      name: dcs-kubernetes-1.33-files
  owner: "root:root"
  path: /etc/kubernetes/audit/policy.yaml
  permissions: "0644"
```

`encryption-provider.conf` is not part of this Secret, and it is not part of `files` at all — the DCS provider delivers it. See How the Encryption Provider Configuration Is Delivered.

To check whether the Secret is present on the target cluster before deciding which form to use:

```
# Replace <kubernetes-major-minor> with the value matching this clu
ster
# (for example, 1.33 for Kubernetes v1.33.x).
K8S_MM=<kubernetes-major-minor>
kubectl -n cpaas-system get secret "dcs-kubernetes-${K8S_MM}-files"
>/dev/null 2>&1 \
  && echo "Secret present — contentFrom.secret form is supported" \
  || echo "Secret missing — use inline content form"
```

How the Encryption Provider Configuration Is Delivered

`/etc/kubernetes/encryption-provider.conf` is owned by the DCS provider, not by `KubeadmControlPlane`. When the provider builds a control-plane machine's Ignition config it writes the file itself, and it removes any entry for that path that arrives from `kubeadmConfigSpec.files`. An entry you add there has no effect — it is neither merged nor applied, so a key written into it is silently unused.

What the provider writes depends on the cluster:

Condition	File content
`DCSCluster.spec.encryptionProviderConfigRef` is set	The `encryption-provider.conf` key of the referenced Secret. Used for disaster recovery, where both sides must hold the same key. The Secret must exist in the cluster the provider runs in; see Optional Disaster Recovery Deployment
Not set, and the control plane is not yet initialized (the first control-plane machine)	A freshly generated random AES-CBC key
Not set, and the control plane is already running (every later control-plane machine)	The file read back from the running `kube-apiserver`, so all control-plane nodes share one key

Worker nodes never receive the file.

`clusterConfiguration.apiServer.extraVolumes` mounts `/etc/kubernetes` into the API server container, so `apiServer.extraArgs.encryption-provider-config: /etc/kubernetes/encryption-provider.conf` resolves on every control-plane node. Keep that argument — as the appendix has it — to run with etcd secret encryption-at-rest. Removing it turns encryption-at-rest off; the provider still writes the file, the API server just stops using it.

Next Steps

After creating a cluster:

- Manage Nodes
- Upgrade Cluster

Troubleshooting

If the cluster reaches `Provisioned` but never becomes Ready — for example, workload nodes stay `NotReady` because the CNI is not deployed — see Troubleshoot a Workload Cluster Stuck in Provisioned.

Troubleshoot a DCSMachine Stuck in Deleting

Use this guide when a `DCSMachine` on a Huawei DCS workload cluster stays in `Phase: Deleting` for longer than the expected ~60 seconds, blocking node turnover during scale-down, rolling upgrade, or full-cluster teardown.

TOC

- Scope
- Symptoms
- Reading the VMStopPending Condition
- Diagnostic Flows
  - VM Stuck in `stopping`
  - VM Stuck in Mid-Transition
  - Controller Cannot Reach DCS API
  - DCS Account Lockout
  - FC Site Policy "仅能删除已停止虚拟机" (errorCode `102212808`)
- Use `cpaas.io/retain-vm` as an Escape Hatch
- Verify Recovery
- See Also

Scope

This guide covers the DCS-side failure modes that prevent `cluster-api-provider-dcs` from completing the `reconcileDelete` flow on a node. The provider's normal delete sequence is:

```
running VM → safe StopVm → stopped → (detach persistent disks if any) → DeleteVm → VM gone in DCS
```

Each step depends on the DCS platform responding. When DCS is unresponsive or refuses a step, the controller surfaces the wait reason on `DCSMachine.Status.Conditions.VMStopPending`. Reading that condition first usually identifies the problem without needing controller log access.

This guide does **not** cover normal pre-deletion gating (CAPI `Machine` drain timeouts, finalizers held by other controllers) — those belong to upstream Cluster API troubleshooting.

Symptoms

Where to look	What you see
`Machine.status.phase` (on the `global` cluster)	`Deleting` for > 5 minutes
`DCSMachine.status.conditions[?(@.type=="VMStopPending")]`	`Status: False` with a `Reason` and `Message` describing the wait
DCS portal VM view	The VM is still listed (running, stopping, or some other non-stopped state)
Controller log on the `global` cluster	A `cluster-api-provider-dcs-manager` line such as `issued safe StopVm` or `DeleteVm rejected by FC site policy`

If `VMStopPending` is absent and `Machine.status.phase` is still `Deleting`, the issue is more likely upstream of `reconcileDelete` (e.g., CAPI drain, kubeadm pre-delete hooks).

Consult the CAPI troubleshooting docs ↗ for that case.

Reading the VMStopPending Condition

The `VMStopPending` condition is the primary signal for this guide.

```
kubectl -n <ns> describe dcsmachine <name>
```

Look for:

```
Conditions:
  Type:                VMStopPending
  Status:              False
  Severity:            Warning
  Reason:              VMMidTransition
  Message:             VM stuck in stopping; awaiting transition to stoppe
d
```

The `Reason` field is the diagnostic:

Reason	Severity	What it means	Next step
`WaitingForStop`	Info	Safe `StopVm` was issued; controller polls every 10s for `stopped`	Wait up to ~60s. If the VM stays in `stopping` past that, jump to VM Stuck in `stopping`.
`VMMidTransition`	Warning	VM is in a transient state other than `running` / `stopped` (e.g., `stopping`, `starting`, `migrating`, `paused`, `hibernated`)	Inspect the `Message` field for the specific status. See VM Stuck in Mid-Transition.
(condition absent)	—	VM is either already `stopped` or already	Check the DCS portal for VM existence and

Reason	Severity	What it means	Next step
		gone — controller is past the stop phase.	the `Machine` finalizers.

Diagnostic Flows

VM Stuck in `stopping`

The controller issued a graceful `StopVm` (calling DCS API `/action/stop?mode=safe`) and the VM acknowledged but never converged.

Common causes:

1. **VM `pvDriverStatus` is not `running`** — guest tools (vmtools / pvdriver) needed for graceful shutdown isn't responding. Verify on the DCS portal: open the VM detail page, check the `pvDriverStatus` field. If it shows anything other than `running`, the VM cannot accept a safe stop.
2. **VM template predates `4.2.1`** — older DCS VM templates lack the guest tools required for safe stop. Confirm the `vmTemplateName` in the DCSMachine matches a `4.2.1+` template (label `cpaas.io/dcs-vm-template` on the `cpaas-system/<release>-dcs-vm-template` ConfigMap, see Resolving Placeholder Values).
3. **Guest OS hung** — the VM kernel is stuck and is not processing the ACPI shutdown.

Resolution path:

- For (1) and (3): Manually issue a `force` stop from the DCS portal (VM detail → Operations → Stop → Force). Once the VM reaches `stopped`, the controller's next reconcile picks up the new state and continues with `DeleteVm`.
- For (2): The VM cannot be safely deleted by this provider on that template; back the workload off the node, manually power-off the VM, then use `cpaas.io/retain-vm` to skip controller-side delete.

VM Stuck in Mid-Transition

`Message` shows a status like `migrating`, `starting`, `paused`, or `hibernated`. The controller will not issue any operation on a VM in these states (DCS would reject) and is waiting indefinitely for the VM to converge to `running` or `stopped`.

Status in message	Likely cause	Resolution
`migrating`	Platform-initiated vMotion / live migration	Wait. Migration completes on its own; controller resumes when status reaches `running` or `stopped`.
`starting`	Platform initiated VM start after a power event	Wait until `running`, then deletion proceeds.
`paused` / `hibernated`	Platform-paused VM (does **not** self-recover)	Resume the VM manually from the DCS portal (Operations → Resume), or use `cpaas.io/retain-vm` to bypass and clean up the VM later.

Controller Cannot Reach DCS API

If the `VMStopPending` condition is **absent** but `Machine.status.phase` is still `Deleting` for an extended period, the controller may be unable to talk to DCS at all. Check the controller log:

```
kubectl -n cpaas-system logs deployment/cluster-api-provider-dcs-manager
--tail=50
```

Look for one of:

Log keyword	Cause
`connection refused` / `timeout`	DCS API endpoint unreachable from the `global` cluster
`errorCode: 10100116` "帐户锁定中"	DCS admin account locked. See DCS Account Lockout.

Log keyword	Cause
`401 Unauthorized`	Credential Secret is stale; rotate per Cloud Credentials.

DCS Account Lockout

`errorCode: 10100116` indicates the DCS portal account in the cluster's credential Secret is in a brute-force lockout window. The DCS portal lockout policy resets per failed login, so a controller stuck in a retry loop will **indefinitely extend** the lockout.

To break the cycle:

```
# 1. Scale the manager to 0 so it stops retrying
kubectl -n cpaas-system scale deployment cluster-api-provider-dcs-manager
--replicas=0

# 2. Wait 5 minutes for the DCS lockout window to expire naturally
sleep 300

# 3. Restore the manager
kubectl -n cpaas-system scale deployment cluster-api-provider-dcs-manager
--replicas=1

# 4. Verify the controller is happy
kubectl -n cpaas-system logs deployment/cluster-api-provider-dcs-manager
--tail=20
```

Avoid reusing the DCS portal `admin` account for the provider's Secret in production. Configure a dedicated DCS account (interconnect or domain user, see Credential User Types) with the lockout policy relaxed, so a transient controller failure cannot lock the platform out for everyone.

FC Site Policy "仅能删除已停止虚拟机" (errorCode `102212808`)

If the workload cluster runs on a DCS site whose underlying FusionCompute platform has the "delete-only-when-stopped" safety policy enabled, **DeleteVm will be rejected when the VM**

is still `running`. The provider's normal delete flow handles this automatically by issuing safe `StopVm` before `DeleteVm`, but in rare race conditions an external actor (a human operator on the DCS portal, or another automation) can restart the VM between the controller's stop check and the `DeleteVm` call. The controller surfaces the race as:

```
Controller log:
  INFO  DeleteVm rejected by FC site policy, requeuing
        vmUrn=urn:sites:<site>:vms:<vm-id>
        errorCode=102212808
```

This is **not** an error — the controller treats `errorCode 102212808` as recoverable and requeues. The next reconcile re-checks VM status; if running again, the controller re-issues `StopVm(safe)` and proceeds. Normal recovery time is ~30 seconds.

If you see this errorCode repeatedly (more than 3 cycles), an external actor is racing the controller. Suspend the external automation, or apply `cpaas.io/retain-vm` to fully take control of VM lifecycle outside the controller.

Use `cpaas.io/retain-vm` as an Escape Hatch

When the controller cannot finish `reconcileDelete` for any reason and you need to release the `Machine` finalizer so the workload cluster can proceed (scale-down, upgrade, removal), annotate either the `Machine` or the `DCSMachine`:

```
kubectl -n <ns> annotate machine <machine-name> cpaas.io/retain-vm=true
```

Effect:

- The controller's `reconcileDelete` recognizes the annotation, skips both `ensureVmStopped` and `DeleteVm`, and releases the `DCSMachine` finalizer.
- The `Machine` then transitions out of `Deleting` and Cluster API can complete the parent operation.

Side effect — important:

- **The VM is not deleted from DCS** — it remains on the DCS platform, still consuming compute and storage resources, **outside Cluster API's lifecycle management**.
- The IP / hostname slot in `DCSIpHostnamePool` is **also not released** — the pool entry will show as still in-use until you manually free it.

After applying `retain-vm`, plan to manually:

1. Stop and delete the VM on the DCS portal (or via DCS API).
2. Free the IP / hostname entry from the corresponding `DCSIpHostnamePool`.
3. (Optional) Clean up any persistent disks that were attached.

Verify Recovery

After applying any of the resolutions above, watch the `DCSMachine` until the condition clears:

```
kubectl -n <ns> describe dcsmachine <name>
# Look for: Conditions: VMStopPending  Status: True
#           (or the condition disappears as the resource is deleted)

kubectl -n <ns> get machines | grep <machine-name>
# Phase should transition: Deleting → (Machine gone within ~30s)
```

If the `Machine` is fully gone and the VM is no longer listed on the DCS portal, recovery is complete.

See Also

- Create a Cluster on Huawei DCS — full cluster manifest reference
- Cloud Credentials for Huawei DCS — Secret format and User Types
- Troubleshoot a Workload Cluster Stuck in Provisioned — symptom-adjacent guide for cluster bring-up issues
- Troubleshoot a Cluster Stuck in Deleting — top-level `Cluster` CR stuck in `Deleting` after every child machine is gone

Huawei Cloud Stack

Overview

Install the Provider

Infrastructu

Create a Cluster

Troubleshoot Huawei Cloud Stack Workload C

Huawei Cloud Stack Provider

Install the Provider

Install the Provider

When You Need This

Prerequisites

Steps

Next Steps

Infrastructure Resources

Infrastructure Resources

Overview

Before Writing YAML

Credential Secret Inputs

Compute Values

Version and Component Baseline

Network Inventory

Control Plane ELB Address Plan

Control Plane HA Placement Plan

Static IP Pool Plan

Value-to-YAML Mapping

Next Steps

Create a Cluster

Create a Cluster

Prerequisites

Using the Web UI

Using YAML

Cluster Verification

Managing Nodes

Upgrading Clusters

Troubleshooting

Troubleshoot Huawei Cloud Stack Workload Clusters

Troubleshoot Huawei Cloud Stack Workload Clusters

Pattern: kubeadm init Never Completes Because the Hostname Is Wrong

Related

The Infrastructure Provider for Huawei Cloud Stack creates and manages Alauda OS clusters on the Huawei Cloud Stack (HCS) platform. This provider integrates with HCS to provision and manage cloud resources for Kubernetes clusters using Cluster API.

TOC

- Install the Provider
- Infrastructure Resources
- Create a Cluster
- Troubleshoot Huawei Cloud Stack Workload Clusters
- Overview
- Key Features
- Cluster Management Capabilities
  - Web UI-Based Management (Fleet Essentials)
  - YAML-Based Management
- Supported Resources
- Documentation

Overview

Huawei Cloud Stack is a hybrid cloud solution that extends public cloud capabilities to private data centers. The HCS Provider leverages HCS APIs to manage infrastructure resources including virtual machines, networks, storage, and load balancers.

The HCS Provider supports pool-managed persistent disks. Declare disks that must survive node replacement in `HCSMachineConfigPool.spec.configs[].persistentDisks[]`, not in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.

Key Features

- **Kube-OVN Integration**: Native support for Kube-OVN container network interface
- **ELB Load Balancer**: Built-in elastic load balancer support for control plane high availability
- **Static IP Configuration**: Static IP address allocation for enterprise network environments
- **Machine Configuration Pools**: Pre-defined hostnames, static IP addresses, and persistent disk slots for predictable VM provisioning
- **Pool-Managed Persistent Disks**: EVS disks can be detached from an old ECS and attached to the replacement ECS during rolling upgrades
- **Control Plane Security Bootstrap**: Supports kube-apiserver audit, admission, encryption provider, and kubelet certificate configuration through kubeadm bootstrap data
- **Per-Node-Pool Availability Zone Selection**: Select one HCS availability zone for the control plane or each worker node pool; separate worker pools can target different zones. See Availability Zone Placement

Cluster Management Capabilities

The HCS Provider always supports **YAML-based** cluster management through Cluster API manifests. It also supports **web UI-based** management when **Fleet Essentials** `1.0.2` or later is installed.

Web UI-Based Management (Fleet Essentials)

When Fleet Essentials is installed, you can manage HCS clusters through the web UI:

- Create clusters with a 6-step guided wizard that selects assignable static IP addresses per subnet.
- View Control Plane and Worker Node Pools, add or delete Worker Node Pools, and manage reserved node IP addresses for an existing cluster.

YAML-Based Management

For automation and GitOps workflows, manage HCS clusters using Cluster API manifests. This workflow does not depend on Fleet Essentials. See Create a Cluster on Huawei Cloud Stack for YAML-based instructions.

Supported Resources

Resource	Description
`HCSCluster`	Represents HCS infrastructure including VPC, subnets, and load balancer configuration
`HCSMachine`	Represents a virtual machine instance in HCS
`HCSMachineTemplate`	Template for creating HCS machines with instance specifications
`HCSMachineConfigPool`	Pool of pre-defined hostnames, static IP addresses, and persistent disk slots

Documentation

- Installation Guide
- Create a Cluster
- Manage Nodes
- Upgrade Kubernetes
- Migrate Existing Clusters to Pool-Managed Persistent Disks
- Provider APIs

Install the Huawei Cloud Stack Provider

This document describes how to install the Infrastructure Provider for Huawei Cloud Stack and the Cluster Bootstrap Provider on the `global` cluster.

TOC

When You Need This
Prerequisites
Steps
Download the packages
Upload the packages
Install the cluster plugins
Verify the installation
Next Steps

When You Need This

Complete this page after the `global` cluster is installed and before you create the first workload cluster on Huawei Cloud Stack. It is required regardless of the IaaS platform that the `global` cluster itself runs on. In a disaster recovery (DR) pair, complete it on **both** `global` clusters, so that the standby can reconcile the same workload clusters after a failover. Upload the packages to each `global` cluster.

This is a separate task from the provider steps in Installing the global Cluster. Those steps upload provider packages to the temporary bootstrap cluster to create the `global` cluster itself.

Prerequisites

Before installing the provider, ensure you have:

- Access to the Alauda Container Platform `global` cluster
- Access to Customer Portal for downloading plugins

Steps

1 Download the packages

To create clusters on Huawei Cloud Stack, you need two cluster plugins. Download both from the Customer Portal. For ACP 4.5, they are listed under **Common ACP Extensions**; see Download.

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Huawei Cloud Stack**

2 Upload the packages

Upload both packages to the platform with `violet push`. For the command and its parameters, refer to Upload Packages.

3 Install the cluster plugins

Install both cluster plugins on the `global` cluster. For the procedure, refer to Cluster Plugin.

4 Verify the installation

After installing the provider, verify that it is correctly installed:

```
# Check if the HCS provider pods are running
kubectl get pods -n cpaas-system | grep hcs

# Verify HCS CRDs are registered
kubectl get crd | grep -E '^hcs.*\.infrastructure\.cluster\.x-k8s\.io
$'

# Expected output should include:
# hcsclusters.infrastructure.cluster.x-k8s.io
# hcsmachines.infrastructure.cluster.x-k8s.io
# hcsmachinetemplates.infrastructure.cluster.x-k8s.io
# hcsmachineconfigpools.infrastructure.cluster.x-k8s.io
```

Next Steps

After installing the provider, you can proceed to:

- Prepare Infrastructure Resources
- Create a Cluster

Infrastructure Resources for Huawei Cloud Stack

TOC

- Overview
- Before Writing YAML
- Credential Secret Inputs
- Compute Values
  - Availability Zone Placement
- Version and Component Baseline
- Network Inventory
  - Multi-NIC Constraints
- Control Plane ELB Address Plan
  - Operational constraints
- Control Plane HA Placement Plan
- Static IP Pool Plan
  - Pool-Managed Persistent Disks
- Value-to-YAML Mapping
- Next Steps

Overview

Before you write YAML for a Huawei Cloud Stack (HCS) cluster, prepare all required HCS inputs first. This page lists the values, sources, and constraints that must be ready before you fill any `Secret`, `HCSMachineConfigPool`, `HCSMachineTemplate`, `KubeadmControlPlane`, or `HCSCluster` manifest.

Use this page as the preparation checklist. After you complete it, continue with Create a Cluster on Huawei Cloud Stack and Manage Nodes on Huawei Cloud Stack for the manifest workflow.

> **INFO**
>
> **Namespace Requirement**
>
> All HCS infrastructure resources must be deployed in the `cpaas-system` namespace to ensure proper integration with the platform as workload clusters.

Before Writing YAML

Prepare the following inputs before you create or edit any cluster manifests:

Input	Used by	Source
Cluster name	`Cluster`, `KubeadmControlPlane`, `HCSCluster`, templates, pools	Your cluster naming plan
Kubernetes version and component baseline	`Cluster`, `KubeadmControlPlane`	Approved release baseline; OS Support Matrix only for the component versions it lists

Input	Used by	Source
`accessKey` and `secretKey`	HCS credential `Secret`	HCS **My Settings** > **Access Keys**
`projectID`	HCS credential `Secret`	HCS **My Settings** > **Resource Spaces**
`externalGlobalDomain`	HCS credential `Secret`	HCS platform access domain
`schema`	HCS credential `Secret`	HCS administrator
`region`	HCS credential `Secret`	HCS administrator
`imageName`	`HCSMachineTemplate`	HCS image inventory
`flavorName`	`HCSMachineTemplate`	HCS administrator
`availabilityZone`	`HCSMachineTemplate`	HCS administrator or the ACP web UI
Root, temporary data volume, and persistent disk layout	`HCSMachineTemplate`, `HCSMachineConfigPool`	Cluster storage plan
VPC name and security group name	`HCSCluster`	HCS network inventory

Input	Used by	Source
Cluster subnet inventory	`HCSCluster`, `HCSMachineConfigPool`, control plane ELB	HCS network inventory
Control plane and worker hostnames and static IPs	`HCSMachineConfigPool`	HCS subnet planning
`vipAddress` and `vipSubnetName`	`HCSCluster.spec.controlPlaneLoadBalancer`	HCS ELB address plan
`elbVirsubnetL4Ips` and `elbVirsubnetL7Ips`	`HCSCluster.spec.controlPlaneLoadBalancer`	HCS ELB address plan
`vipDomainName`	`HCSCluster.spec.controlPlaneLoadBalancer`	HCS Cloud DNS Private Zones
Control plane HA placement policy	`HCSCluster.spec.controlPlaneHA`	HCS scheduling policy plan
`controlPlaneEndpoint`	`Cluster.status` / derived cluster endpoint	Controller-managed

Credential Secret Inputs

Create the HCS credential `Secret` only after you collect all required values.

Secret key	Meaning	Where to get it
`accessKey`	HCS access key ID	HCS **My Settings** > **Access Keys**

Secret key	Meaning	Where to get it
`secretKey`	HCS secret access key	HCS **My Settings** > **Access Keys**
`projectID`	Resource Space ID	HCS **My Settings** > **Resource Spaces**
`externalGlobalDomain`	HCS platform access domain	HCS platform domain provided for API access
`region`	HCS region API value used by the provider	HCS administrator
`schema`	Optional API scheme for the HCS IAM endpoint. If omitted, the provider uses `https`	HCS administrator

Existing credential Secrets created without `schema` continue to work unchanged. Only set `schema` if your HCS IAM endpoint uses `http` instead of the default `https`.

Note: Tenant administrators cannot retrieve `region` from the HCS UI. Get the exact provider-recognized value from the HCS administrator before you encode the Secret.

Compute Values

Prepare the VM image, flavor, availability zone, temporary data volume layout, and persistent disk layout before you write the `HCSMachineTemplate` and `HCSMachineConfigPool`.

Input	Guidance
`imageName`	Use the validated HCS image name for the Alauda OS image you want to deploy
`flavorName`	Use the provider-recognized HCS API value matched against `Flavor.Name`. Do not use the tenant UI display name

Input	Guidance
`availabilityZone`	Use the provider-recognized HCS API value matched against `ZoneName`. Do not use the tenant UI display name
Root and temporary data volumes	Plan system and temporary data disks in advance. Control plane templates typically use temporary data volumes for `/var/lib/etcd`, `/var/lib/kubelet`, and `/var/lib/containerd`. Worker templates typically use temporary data volumes for `/var/lib/kubelet` and `/var/lib/containerd`.
Pool-managed persistent disks	Plan disks that must survive VM replacement in advance. Use this model for `/var/cpaas` and other node-local state that must be retained during rolling replacement.

Note: The ACP web UI loads flavors and availability zones through the selected HCS credential. When you write YAML, get the exact provider-recognized `Flavor.Name` and `ZoneName` values from the HCS administrator because the native HCS tenant console might not expose them.

Availability Zone Placement

An `HCSMachineTemplate` selects one availability zone for every machine created from that template. The ACP web UI creates one template for the control plane node pool and one template for each worker node pool. As a result:

- All control plane nodes use one availability zone.
- All replicas in one worker node pool use one availability zone.
- Different worker node pools can select different availability zones.
- A single node pool does not automatically spread replicas across multiple availability zones.

For predictable placement, select an availability zone explicitly. If `HCSMachineTemplate.spec.template.spec.availabilityZone` is omitted, the Provider queries HCS and selects the first availability zone that HCS reports as available. The ordering of that list is controlled by HCS.

When you use the ACP web UI, the Availability Zone dropdown is loaded through the selected HCS credential, uses the HCS API `ZoneName`, and disables zones that HCS reports as unavailable. When you write YAML, use the provider-recognized `ZoneName`, not a display name from the native HCS tenant console. The Provider rejects a name that is not returned by HCS. For a named zone that exists but is currently unavailable, ECS provisioning can still reach HCS and fail there, so verify that the zone is available before you apply the manifest.

WARNING

Pool-managed persistent disks and control plane high availability add placement constraints:

- An existing pool-managed persistent disk locks its replacement ECS instance to the disk's availability zone. If the template explicitly selects a different zone, the replacement fails with an availability zone conflict.
- Control Plane High Availability spreads control plane ECS instances across physical hosts by using an HCS server group. It does not spread the control plane across availability zones; all control plane nodes remain in the zone selected by the control plane `HCSMachineTemplate`.

Version and Component Baseline

Use the OS Support Matrix only for the component versions it explicitly lists, such as Kubernetes, coredns, etcd, and pause versions for supported Alauda OS images.

The OS Support Matrix is not a complete source for all HCS manifest values. Before writing YAML, also get the approved release baseline for values such as the container image repository, DNS image repository, Kube-OVN version, Kube-OVN join CIDR, Pod CIDR, and Service CIDR. If your release does not publish a complete baseline source yet, use values validated for your environment by the platform or release owner.

When you create clusters through the web UI instead of YAML, you do not enter these component versions by hand. The wizard reads them from the OS-release ConfigMap you register for the OS image. See OS Release Registration for the registration steps and field sources.

Network Inventory

Prepare the complete cluster network inventory before you write `HCSCluster` or `HCSMachineConfigPool` resources.

Your network plan must include:

- The target VPC name
- The target security group name
- Every subnet name the cluster will use
- The subnet ID and ELB-related subnet metadata for each subnet
- The CIDR of each subnet
- The gateway and DNS values of each subnet
- The planned free IP ranges for control plane nodes, worker nodes, the ELB VIP, and ELB L4/L7 virtual subnet IPs

If a single cluster uses multiple subnets, those subnets must belong to the same VPC and must allow cluster nodes to reach each other.

Important: `HCSCluster.spec.network.subnets` is the cluster subnet inventory. Every `subnetName` referenced by `HCSMachineConfigPool`, `vipSubnetName`, `elbVirsubnetL4Ips[].subnetName`, and `elbVirsubnetL7Ips[].subnetName` must already exist in that list.

For the initial cluster create flow, the controller can resolve existing subnets by name before the cluster becomes Ready. For an existing Ready cluster, do not append only a subnet name. Add the full subnet object, including at least `name`, `id`, and, when the subnet is used by the control plane ELB, `neutronSubnetId`. Keep `cidr`, `gatewayIp`, `primaryDNS`, and `secondaryDNS` in the inventory as well so the cluster subnet list remains complete.

Multi-NIC Constraints

Multiple NICs are declared in `HCSMachineConfigPool.spec.configs[].networks[]`. Each `networks[]` entry only selects a subnet and a static IP for one NIC.

The current provider does not let you declare:

- The role or purpose of each NIC, such as management, service, or storage traffic
- The default gateway for a specific NIC
- Static routes or route metrics
- Per-NIC DNS settings

Treat the current multi-NIC capability as NIC attachment plus subnet and static IP selection.

Control Plane ELB Address Plan

The HCS provider creates the control plane Elastic Load Balance (ELB) automatically. Plan the ELB inputs before you write `HCSCluster`.

This documentation uses the fixed-address ELB workflow. Prepare all ELB-related addresses before you write `HCSCluster`:

- `vipSubnetName`
- `vipAddress`
- `elbVirsubnetL4Ips` with exactly two L4 IPs
- `elbVirsubnetL7Ips` with exactly two L7 IPs

Leave `vipDomainName` unset: cluster access by domain is not supported (the API server certificate can only be issued by the cluster's own CA), and the provider only uses the field to backfill the derived `controlPlaneEndpoint`.

Operational constraints

- The provider creates the ELB and enables Hybrid Load Balancing so cluster nodes can reach the API server through the ELB address.
- Do not disable Hybrid Load Balancing on the HCS ELB after the cluster is created.
- Do not write `spec.controlPlaneEndpoint` in the create manifest. The controller fills that field after the ELB is ready.

Control Plane HA Placement Plan

For a multi-replica control plane, you can configure the HCS provider to create and maintain a provider-owned server group for control-plane ECS placement. Enable this through `HCSCluster.spec.controlPlaneHA` when you write the cluster manifest.

Plan the placement policy before you enable the feature:

Policy	Behavior	Operational impact
`anti-affinity`	Hard anti-affinity. HCS must place control-plane ECS instances on different hosts.	Use this when strict host separation is required. ECS creation can fail if HCS cannot find enough capacity that satisfies the rule.
`soft-anti-affinity`	Best-effort anti-affinity. HCS tries to place control-plane ECS instances on different hosts.	Use this when you prefer spread but do not want the rule to block ECS creation. Final placement may still share a host.

This feature is cluster-scoped and applies only to control-plane machines. It does not configure worker node pools. The provider creates the server group, attaches control-plane ECS creation requests to that group, records the server group in `HCSCluster.status.controlPlaneHA`, and reports progress through the `ControlPlaneHAReady` condition.

For an existing cluster, enabling `HCSCluster.spec.controlPlaneHA` creates and records the provider-owned server group, but it does not retroactively add already-running control-plane ECS instances to that group. Server group placement is applied when a control-plane ECS is created with `os:scheduler_hints.group`. To converge an existing cluster, enable `controlPlaneHA`, wait for `HCSCluster.status.controlPlaneHA.id`, then trigger a control-plane rolling replacement so replacement ECS instances are created with the server group. Until the current control-plane ECS members have been replaced and observed in the group, `ControlPlaneHAReady` can remain `False` with reason `ControlPlaneHAPending`.

For the manifest example and verification commands, see Configure HCSCluster and Verify Control Plane HA.

Static IP Pool Plan

This page focuses on the planned static IP workflow.

Prepare the following before you create `HCSMachineConfigPool` resources:

- Control plane hostnames and static IPs
- Worker hostnames and static IPs, if workers are created
- Enough entries to cover the initial replica count
- Any pool-managed persistent disks that must be bound to each fixed hostname

For static IP control planes with at least three replicas, the recommended upgrade path uses `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0`. This scale-down-then-scale-up approach usually does not require extra control plane IPs. If you plan a single-control-plane create-only topology (`replicas: 1`), do not copy that rollout setting into the create manifest. For pools that use persistent disks, keep `maxSurge: 0` because each disk is bound to one fixed hostname and slot.

Pool-Managed Persistent Disks

Declare HCS disks that must survive VM delete-recreate replacement in `HCSMachineConfigPool.spec.configs[].persistentDisks[]`.

Use this model for `/var/cpaas`, monitoring data, log data, or other node-local state that must be retained during rolling replacement. Keep `HCSMachineTemplate.spec.template.spec.dataVolumes[]` for temporary disks that may be recreated with each new ECS.

hcs-machine-config-pool-with-persistent-disk.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <cluster-name>-cp-pool
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: <cluster-name>
spec:
  configs:
    - hostname: <node-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <node-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
          mountOptions:
            - defaults
            - noatime
```

Persistent disk field descriptions:

Field	Description
`slot`	Disk position within one machine configuration. Slots must be unique and contiguous from `0` for the same hostname.
`size`	EVS disk size in GB. For newly created EVS data disks, use `10` to `32768` GB. Existing claimed disks must match their current size.
`type`	EVS disk type name that is available in the target availability zone.
`mountPath`	Mount path inside the guest OS. Use `/var/cpaas` for platform state that must survive VM replacement. Size this disk for platform monitoring, log, and plug-in data.
`format`	File system used when a new disk is initialized. If omitted, the provider uses `xfs`. Existing file systems are not reformatted during reattachment.

Field	Description
`mountOptions`	Mount options applied when the replacement VM mounts the disk. If omitted, the provider uses `defaults,noatime`.

Update rules:

- Do not declare the same `mountPath` in both `HCSMachineConfigPool.spec.configs[].persistentDisks[]` and `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.
- You can append new `persistentDisks[]` entries to an existing machine configuration. The provider creates or claims the EVS disk, but it does not hot-mount the disk into the running ECS. Trigger a rolling replacement with `maxSurge: 0` before you expect the new disk to be formatted and mounted inside the guest OS.
- Do not remove or change `slot`, `size`, `type`, `format`, or `mountPath` after the provider has accepted a persistent disk entry. These fields define the disk identity and claim contract.
- You can update `mountOptions`. The change takes effect when the node is replaced and the replacement VM renders its disk mounts.
- Keep `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge: 0` and `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0` for pools that use persistent disks.

Value-to-YAML Mapping

Use the following mapping after you complete the preparation checklist:

Prepared input	YAML fields
`accessKey`, `secretKey`, `projectID`, `externalGlobalDomain`, `region`, optional `schema`	`Secret.data.*`
`imageName`, `flavorName`, `availabilityZone`, root	`HCSMachineTemplate.spec.template.spec.*`

Prepared input	YAML fields
volume, and temporary data volume layout	
Control plane and worker hostnames, static IPs, and pool-managed persistent disks	`HCSMachineConfigPool.spec.configs[]`
VPC name, subnet inventory, security group name	`HCSCluster.spec.network.*`
`vipAddress`, `vipSubnetName`, `vipDomainName`, `elbVirsubnetL4Ips`, `elbVirsubnetL7Ips`	`HCSCluster.spec.controlPlaneLoadBalancer.*`
Control plane HA placement policy	`HCSCluster.spec.controlPlaneHA.*`
Kubernetes version and component baseline	`KubeadmControlPlane.spec.version`, `Cluster.spec.clusterNetwork.*`, cluster annotations, and related bootstrap settings

Next Steps

After you complete the preparation checklist:

- Follow Create a Cluster on Huawei Cloud Stack to write the control plane and infrastructure manifests
- Follow Manage Nodes on Huawei Cloud Stack to add or modify worker nodes

Create a Cluster on Huawei Cloud Stack

This document provides instructions for creating Kubernetes clusters on the Huawei Cloud Stack platform. YAML-based cluster creation is available through Cluster API manifests. If Fleet Essentials `1.0.2` or later is installed, you can also create clusters through a guided web UI.

INFO

The web UI provides a guided workflow with built-in validation, while YAML offers more automation flexibility.

TOC

Prerequisites

1. Required Plugin Installation
2. HCS Infrastructure Input Preparation
3. OS Release Registration
4. Workload Cluster Image Registry
5. Node Labels and Taints

Using the Web UI

Creation Workflow

Step 1: Basic Info

Step 2: Networking

Step 3: Infrastructure

Step 4: Control Plane Node Pool
Machine Config entry
Step 5: Worker Node Pools
Step 6: Review
Using YAML
Cluster Creation Overview
Control Plane Configuration
Configure HCS Authentication
Configure Machine Configuration Pool
Configure Machine Template
Configure KubeadmControlPlane
Configure HCSCluster
Configure Cluster
Deploy Worker Nodes
Step 1: Configure Machine Configuration Pool
Pool-Managed Persistent Disks for Workers
Step 2: Configure Machine Template
Step 3: Configure Bootstrap Template
Step 4: Configure Machine Deployment
Cluster Verification
Using kubectl
Verify Control Plane HA
Expected Results
Managing Nodes
Upgrading Clusters
Troubleshooting

Prerequisites

Before creating clusters, ensure all of the following prerequisites are met:

1. Required Plugin Installation

Before you create the first workload cluster on Huawei Cloud Stack, upload and install the following plugins on the Alauda Container Platform's `global` cluster. This applies regardless of the IaaS platform that the `global` cluster itself runs on:

- **Alauda Container Platform Cluster Bootstrap Provider**
- **Alauda Container Platform Infrastructure Provider for Huawei Cloud Stack**

For the procedure, see Install the Provider.

2. HCS Infrastructure Input Preparation

Prepare all HCS-specific inputs before writing any YAML in this document:

- HCS credential Secret values
- Provider-recognized compute values such as `imageName`, `flavorName`, and `availabilityZone`
- Cluster network inventory, including the subnets and free IP ranges used by the cluster
- Control plane ELB address planning, including `vipAddress`, `vipSubnetName`, and fixed L4 and L7 IPs
- Static IP pool planning for control plane and worker nodes

See Infrastructure Resources for Huawei Cloud Stack for the complete checklist, source information, and constraints.

3. OS Release Registration

This prerequisite applies to the **web UI** workflow only. YAML-based creation sets the OS image and component versions directly and does not need it.

The web UI does not ask you to type the OS image or its Kubernetes and component versions. It reads them from an **OS-release ConfigMap** that you register once per Alauda OS image. The wizard resolves the platform Distribution version from the `global` cluster, finds the registered OS image whose `supportedACPVersions` includes that Distribution, and derives the node image and control plane component versions from it.

If no registered OS image matches the current Distribution, the wizard blocks creation with an error such as:

> No OS registered for the current platform Distribution v4.3.2. Register an OS-release ConfigMap first.

Register the OS image on the `global` cluster before you open the wizard:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  # Any DNS-1123 name. The name is not a lookup key.
  # Do not copy the image version verbatim: it contains "_" (for example, x86_64), which is invalid in a ConfigMap name.
  name: alaudaos-43-m55-x86-64
  namespace: cpaas-system
  labels:
    cpaas.io/os-release: ""
data:
  osImageVersion: "alaudaos-43.m55.202606251459-0.x86_64"
  supportedACPVersions: "v4.3.2"
  osVariant: "m55"
  architecture: "x86_64"
  osBuild: "202606251459-0"
  kubernetesVersion: "v1.34.5-3"
  containerdVersion: "2.2.1-5"
  coreDNSImageTag: "1.14.2-v4.3.11"
  etcdImageTag: "v3.5.28-260625"
  pauseImageTag: "3.10"
```

Apply it with `kubectl apply -f os-release.yaml`.

Field sources:

Field	Value source
`cpaas.io/os-release` label	Fixed marker with an empty value. The wizard discovers the ConfigMap by this label.
`osImageVersion`	The exact name of the Alauda OS image uploaded to the HCS platform. The provider uses it to locate the ECS

Field	Value source
	image, and it becomes the node pool default image. For a standard release it matches the name the Alauda OS image published for that ACP release was uploaded under. See Downloading Alauda OS.
`supportedACPVersions`	The ACP Distribution versions this OS image serves, comma-separated (for example, `v4.3.2`, or `v4.3.0,v4.3.1`). The wizard matches the `global` cluster's Distribution against this list.
`architecture`	`x86_64` or `aarch64`. Must match the architecture of the node flavors you plan to use.
`osVariant`, `osBuild`	Descriptive fields shown for reference; not used for matching.
`kubernetesVersion`, `containerdVersion`, `coreDNSImageTag`, `etcdImageTag`, `pauseImageTag`	Component versions for this OS image, taken from the matching row of the OS Support Matrix. The wizard injects them into the new cluster's control plane, so they must exist in your air-gap registry.

INFO

Do not add a Kube-OVN version to this ConfigMap. The wizard reads the Kube-OVN version from the `global` cluster's Kube-OVN configuration, not from the OS image.

The `osImageVersion` must name an image that already exists on the HCS platform. To read the exact name used in an environment that already has clusters, list an existing machine template:

```
kubectl -n cpaas-system get hcsmachinetemplate \
  -o custom-columns=NAME:.metadata.name,IMAGE:.spec.template.spec.imageName
```

4. Workload Cluster Image Registry

Decide which registry this cluster pulls its platform components from — Kube-OVN and the platform modules that Alauda Container Platform installs on the cluster. This choice does not affect the images kubeadm manages, which come from `KubeadmControlPlane.spec.kubeadmConfigSpec.clusterConfiguration.imageRepository`.

Option	What you configure	When to use it
Registry of the `global` cluster	Nothing. The cluster inherits it, and the wizard fills in the address for you.	Workload nodes reach the registry the `global` cluster uses with acceptable throughput.
Dedicated registry	A RegistryCredential Secret in `cpaas-system`, referenced from the `Cluster` by the `cpaas.io/registry-reference` label.	Workload nodes sit in a different region or network segment and should pull from a registry close to them.

Either way, the registry must already hold Alauda Container Platform Core for the target version plus the packages for every platform plugin you intend to install on this cluster, and every workload cluster node must reach its address.

For the dedicated-registry procedure and its constraints, see Choose the Image Registry for a Workload Cluster.

WARNING

A dedicated registry is configured through YAML only. The web UI wizard always uses the registry of the `global` cluster: it reads that address from the `global` cluster and writes it into the new cluster for you, and it does not offer a registry credential to select. Use Using YAML when this cluster needs its own registry.

The binding is also read only when the cluster is first created. Changing or removing `cpaas.io/registry-reference` on an existing `Cluster` does not move that cluster to another registry.

5. Node Labels and Taints

To give the nodes of a pool their own role label and taint — for example to reserve dedicated pools for monitoring, logging, or ingress — declare them on the pool's resources as described in Infra Nodes. Plan them before you create the pools. This applies to every node pool, whether the cluster is created through the web UI or with YAML; the web UI node pool steps do not set labels or taints.

Using the Web UI

Requirement: This workflow requires Fleet Essentials `1.0.2` or later. Without it, create clusters with the YAML manifests described in Using YAML.

The web UI loads VPCs, subnets, security groups, flavors, and assignable IP addresses from the HCS platform using the Infrastructure Credential you select. Make sure the prerequisites above are met before you start, and that an HCS credential Secret exists in the `cpaas-system` namespace.

Creation Workflow

The cluster creation follows a 6-step wizard:

```
Step 1: Basic Info
  ↓
Step 2: Networking
  ↓
Step 3: Infrastructure
  ↓
Step 4: Control Plane Node Pool
  ↓
Step 5: Worker Node Pools
  ↓
Step 6: Review
```

Navigation: Clusters → Clusters → Create Cluster → Select Huawei Cloud Stack

> **INFO**
>
> **How IP selection works**

Every IP field in this wizard — the control plane ELB addresses in Step 3 and the node IP addresses in Steps 4 and 5 — is a picker, not a free-text box. Select the subnet first, and the picker lists only the assignable IP addresses of that subnet. HCS clusters use static IP addresses, so you must choose an address from the list; manual entry and DHCP are not supported. Changing the subnet clears the address you already picked. An IP address already chosen elsewhere in the same subnet is removed from the list, so the same address is never assigned twice. When a subnet has many free addresses, the list shows the first batch only; type to filter for a specific address.

Step 1: Basic Info

Field	Type	Required	Description
Infrastructure Credential	dropdown	Yes	Existing HCS credential Secret. Used to load VPCs, subnets, security groups, flavors, and assignable IP addresses. If the dropdown is empty, create a Secret in the `cpaas-system` namespace labeled `capi.cpaas.io/provider=cluster-api-provider-hcs`.
Name	text	Yes	Unique cluster identifier (lowercase letters, numbers, and hyphens), up to 63 characters. Must not be `global`.
Display Name	text	No	Custom description for easy identification, up to 256 characters.
Distribution Version	readonly	-	Platform distribution version, inherited from the `global` cluster.
Kubernetes Version	readonly	-	Determined by the Distribution Version.
OS Version	readonly	-	Alauda OS image version derived from the registered OS-release ConfigMap that matches the Distribution Version.

Step 2: Networking

Field	Type	Required	Description
Kube-OVN Version	readonly	-	Detected from the `global` cluster.
Pod CIDR	CIDR	Yes	Pod network address range.
Service CIDR	CIDR	Yes	Service network address range. Must not overlap the Pod CIDR.
Join CIDR	CIDR	Yes	Kube-OVN join network address range.

Step 3: Infrastructure

This step has two parts: the HCS infrastructure resources and the control plane load balancer.

Infrastructure resources:

Field	Type	Required	Description
VPC Name	dropdown	Yes	Existing VPC loaded from the selected credential.
Security Group Name	dropdown	Yes	Existing security group.
Subnets	dropdown list	Yes	One or more existing subnets. Every subnet referenced by the load balancer or by a node pool must be added here.

Control Plane Load Balancer: The HCS provider creates an Elastic Load Balance (ELB) for the Kubernetes API server and owns its VIP. Hybrid Load Balancing requires fixed L4 and L7 virtual subnet IPs.

Field	Type	Required	Description
VIP Subnet Name	dropdown	Yes	Subnet that holds the ELB VIP. Must be one of the subnets added above.
VIP Address	IP picker	Yes	Fixed VIP for the control plane ELB.
VIP Domain Name	text	No	Leave empty. Cluster access by domain is not supported — the field only backfills the derived cluster endpoint, and the API server certificate is issued by the cluster's own CA.
L4 Virtual Subnet Name	dropdown	Yes	Subnet for the L4 Hybrid Load Balancing IPs.
L4 Virtual Subnet IP 1 / IP 2	IP picker	Yes	Two fixed L4 IP addresses.
L7 Virtual Subnet Name	dropdown	Yes	Subnet for the L7 Hybrid Load Balancing IPs.
L7 Virtual Subnet IP 1 / IP 2	IP picker	Yes	Two fixed L7 IP addresses.

Step 4: Control Plane Node Pool

The control plane node pool is fixed at 3 replicas for high availability.

Field	Type	Required	Description
Control Plane Node Pool Name	text	Yes	Pool name, automatically prefixed with `<cluster-name>-`.
Flavor	dropdown	Yes	HCS flavor; the list shows vCPU and memory for each option.

Field	Type	Required	Description
Availability Zone	dropdown	No	Target availability zone for every control plane node. Zones reported as unavailable are disabled. See Availability Zone Placement.
Machine Configs	list	Yes	One entry per control plane node. At least 3 entries are required. See Machine Config entry below.
System Disk	number	Yes	Root disk size in GB; the type is fixed to SSD.
Data Volumes	list	No	Fixed data disks mounted on every node in the pool; only the size is adjustable. The etcd data volume is included for the control plane.
Replicas	readonly	-	Fixed at 3.
Control Plane High Availability	toggle	No	On by default. Spreads control plane nodes across different physical hosts within the selected availability zone by using a provider-managed HCS server group.
Placement Policy	radio	Conditional	Shown when High Availability is on. **Preferred (soft anti-affinity)** (default) spreads nodes on a best-effort basis. **Required (strict anti-affinity)** forces each node onto a different host and fails creation if there are not enough hosts.
SSH Authorized Keys	list	No	Public keys for node access.

Machine Config entry

Each entry in **Machine Configs** describes one node:

- **Hostname** (required): Lowercase letters, numbers, hyphens, and dots, up to 253 characters (for example, `master-1` or `master-1.example.org`). For how a dotted hostname affects the node, see Hostname behavior on the node.
- **Networks** (required): One row per NIC. Each row has an **IP Address** picker and a **Subnet Name**. Add rows for multi-NIC nodes.
- **Persistent Disks**: The platform-required `/var/cpaas` disk is pre-filled; its size is adjustable but the row cannot be removed. Add rows only for extra host-bound data that must survive node rebuild.

Step 5: Worker Node Pools

Click **Add Worker Node Pool** to open the pool dialog. You can add multiple pools.

Field	Type	Required	Description
Worker Node Pool Name	text	Yes	Pool name, automatically prefixed with `<cluster-name>-`. Must be unique within the cluster.
Flavor	dropdown	Yes	HCS flavor; the list shows vCPU and memory for each option.
Availability Zone	dropdown	No	Target availability zone for every node in this worker pool. Different worker pools can select different zones; zones reported as unavailable are disabled. See Availability Zone Placement.
Machine Configs	list	Yes	One entry per worker node. Uses the same Machine Config entry layout as the control plane.
System Disk	number	Yes	Root disk size in GB; the type is fixed to SSD.
Data Volumes	list	No	Fixed data disks mounted on every node in the pool.

Field	Type	Required	Description
Replicas	number	Yes	Number of nodes in the pool. Must not exceed the number of machine configs.
Rollout Strategy	number	No	**Max Surge** and **Max Unavailable**. When Max Surge is `0`, Max Unavailable must be ≥ 1 and ≤ Replicas.
SSH Authorized Keys	list	No	Public keys for node access.

Step 6: Review

Review the **Basic Info**, **Infrastructure**, **Control Plane Load Balancer**, **Control Plane Node Pool**, **Worker Node Pools**, and **Networking** sections. Click **View YAML** to inspect the exact resources that will be created. Click **Create** to start the cluster creation process.

Using YAML

Cluster Creation Overview

At a high level, you'll create the following Cluster API resources in the Alauda Container Platform's `global` cluster to provision infrastructure and bootstrap a functional Kubernetes cluster.

Before you write any YAML in this page, complete the preparation checklist in Infrastructure Resources for Huawei Cloud Stack. This checklist covers the values that the provider expects, where to get them, and which values must be planned before you fill the manifests.

WARNING

Important Namespace Requirement

To ensure proper integration with the Alauda Container Platform as workload clusters, all resources must be deployed in the `cpaas-system` namespace. Deploying resources in other namespaces may result in integration issues.

WARNING

Workload Cluster Naming

The workload `cluster-name` **must not** be `global`. That name is reserved for the `global` cluster, and reusing it causes the workload cluster's resources to collide with `global` cluster resources in `cpaas-system`. The `global-` prefix is reserved for resources owned by the `global` cluster's DR workflow; see Common Prerequisites. Do not use `global-` for workload-cluster resources, because failover operations can select those resources as if they belonged to the `global` cluster.

As a convention, keep the CAPI `Cluster` and provider cluster resource (`HCSCluster`) named exactly `<cluster-name>`, and prefix non-root CAPI and provider resources (`KubeadmControlPlane`, `KubeadmConfigTemplate`, `MachineDeployment`, machine templates, machine config pools, etc.) with `<cluster-name>-` — for example, the example manifests use `<cluster-name>-kcp`. This is a recommendation rather than a controller-enforced rule, but it prevents same-namespace collisions when multiple workload clusters live in `cpaas-system` and makes resource ownership obvious during operations.

The cluster creation process follows this order:

1. Configure HCS authentication (Secret)
2. Create machine configuration pool (HCSMachineConfigPool)
3. Configure machine template (HCSMachineTemplate)
4. Configure KubeadmControlPlane
5. Configure HCSCluster
6. Create the Cluster
7. Deploy worker nodes (`HCSMachineConfigPool`, `HCSMachineTemplate`, `KubeadmConfigTemplate`, and `MachineDeployment`). See Deploy Worker Nodes.

Control Plane Configuration

The control plane manages cluster state, scheduling, and the Kubernetes API. This section shows how to configure a highly available control plane.

> **WARNING**
>
> **Configuration Parameter Guidelines**
>
> When configuring resources, exercise caution with parameter modifications:
>
> - **Replace only values enclosed in** `<>` with your environment-specific values
> - **Preserve all other parameters** as they represent optimized or required configurations
> - Modifying non-placeholder parameters may result in cluster instability or integration issues

Configure HCS Authentication

HCS authentication information is stored in a Secret resource in `cpaas-system`.

Author the credential file with an editor rather than typing the values at a shell prompt. A shell records the whole command, secret key included, in its history; a file written by an editor leaves no such copy.

hcs-credential.env

```
accessKey=<access-key>
secretKey=<secret-key>
projectID=<project-id>
region=<region>
externalGlobalDomain=<domain>
schema=https
```

Restrict the file, create the Secret from it, then remove it:

```
chmod 600 hcs-credential.env

HCS_SECRET_NAME="<credential-secret-name>"

kubectl create secret generic "${HCS_SECRET_NAME}" \
  --namespace cpaas-system \
  --from-env-file=hcs-credential.env

rm -f hcs-credential.env
```

Write the values in plain text — `kubectl` base64-encodes them when it creates the Secret, so there is no separate encoding step. Each line is `key=value`. Do not quote the values: `kubectl` takes quote characters literally. A value may contain `=`; only the first `=` on a line separates the key from the value.

> **Do not create credential Secrets with kubectl apply**
>
> Client-side apply stores the manifest it applied — including the HCS secret key — in the `kubectl.kubernetes.io/last-applied-configuration` annotation on the Secret itself. `kubectl create` writes no such annotation. This applies whether the values were written as `stringData` or as base64 `data`; base64 is an encoding, not protection.

Key	Description
`accessKey`	HCS access key ID from **My Settings** > **Access Keys**
`secretKey`	HCS secret access key from **My Settings** > **Access Keys**
`projectID`	Resource Space ID from **My Settings** > **Resource Spaces**
`region`	HCS region API value used by the provider. Tenant administrators cannot retrieve this value from the HCS UI; get it from the HCS administrator
`externalGlobalDomain`	HCS platform access domain used by the provider
`schema`	Optional API scheme for the HCS IAM endpoint, usually `https`. Omit the line to use the default

Existing credential Secrets created without `schema` continue to work unchanged. Only set `schema` if your HCS IAM endpoint uses `http` instead of the default `https`.

You can reuse an existing HCS credential Secret. Its name does not need to match the cluster name, but `HCSCluster.spec.identityRef.name` must reference this Secret.

Configure Machine Configuration Pool

The `HCSMachineConfigPool` defines pre-configured hostnames, static IP addresses, and any pool-managed persistent disks for VMs.

> **WARNING**
>
> **Pool Size Requirement**
>
> The configuration pool must include at least as many entries as the number of control plane nodes you plan to deploy.

Use one subnet selector per `networks[]` entry. For new manifests, set either `subnetName` or `subnetId`, but not both. Existing manifests may keep the deprecated `subenetName` field; if you also add `subnetName` while updating that manifest, its value must exactly match `subenetName`. Do not supply conflicting values across `subenetName`, `subnetName`, and `subnetId`.

If you use `subnetName` in the machine configuration pool, include the same subnet name in `HCSCluster.spec.network.subnets`.

For the initial cluster create flow, listing an existing subnet by name is enough because the controller resolves subnet metadata before the cluster becomes Ready. If you later add another subnet to an existing Ready `HCSCluster`, do not append only `name`. Patch the parent `HCSCluster.spec.network.subnets` entry with the full subnet object so later machine or ELB operations can reuse the resolved subnet metadata.

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <cluster-name>-cp-pool
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: <cluster-name>
spec:
  configs:
    - hostname: master-1
      networks:
        - subnetName: <subnet-name>
          ipAddress: 192.168.1.11
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: master-2
      networks:
        - subnetName: <subnet-name>
          ipAddress: 192.168.1.12
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: master-3
      networks:
        - subnetName: <subnet-name>
          ipAddress: 192.168.1.13
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
```

Parameter	Type	Required	D
`.spec.configs[]`	array	Yes	N m c
`.spec.configs[].hostname`	string	Yes	V lo nu (th ar lo nu no ch D lir L de ar le ar su th S **b n** be re va
`.spec.configs[].networks[]`	array	Yes	N ne co V
`.spec.configs[].networks[].subnetName`	string	No*	R su

Parameter	Type	Required	D
			n
`.spec.configs[].networks[].subnetId`	string	No*	S fi `s` th a
`.spec.configs[].networks[].ipAddress`	string	Yes	S th
`.spec.configs[].persistentDisks[]`	array	No	E su d re
`.spec.configs[].persistentDisks[].slot`	int	Yes*	D m c m c fo h
`.spec.configs[].persistentDisks[].size`	int	Yes*	E F E `1` E di th
`.spec.configs[].persistentDisks[].type`	string	Yes*	E a ta z
`.spec.configs[].persistentDisks[].mountPath`	string	No	G U fo

Parameter	Type	Required	D
			th re
`.spec.configs[].persistentDisks[].format`	string	No	Fi or us
`.spec.configs[].persistentDisks[].mountOptions`	array	No	M or us `c`

*For new manifests, set either `subnetName` or `subnetId`. Existing manifests may continue to use `subenetName`, and may add `subnetName` only if both fields use the same value. Do not provide conflicting subnet selector values.

Persistent disk fields are required when `persistentDisks` is specified.

Use `persistentDisks[]` for node-local state that must survive VM replacement. Do not declare the same mount path in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.

Note: The CRD schema lists `subnetName`, `subenetName`, and `subnetId` as optional fields and does not express their allowed combinations. Follow the provider-level rules above when writing manifests.

Note: To attach multiple NICs to one node, add multiple `networks[]` entries. The provider only uses these entries to attach NICs and assign subnet selectors plus static IPs. It does not support declaring per-NIC roles, default gateways, static routes, or per-NIC DNS settings.

Hostname behavior on the node

The provider derives the node's POSIX hostname and FQDN from `hostname` as follows:

Pool `hostname` value	Node `hostname`	Node `hostname -f`	`/etc/hosts` entry written
`master-1` (no dot)	`master-1`	`master-1` (resolver returns the short name)	none added by the provider
`master-1.example.org` (dotted)	`master-1` (short label, dot and everything after it stripped)	`master-1.example.org`	`<node-ip> master-1.example.org master-1`

The dotted form is the path to set up applications that depend on FQDN resolution (`hostname -f` , certificates with SAN entries, log labels). The provider sets `prefer_fqdn_over_hostname: false` and enables cloud-init `manage_etc_hosts` only when a dotted hostname is supplied, so POSIX tools continue to see the short name and `hostname -f` returns the full FQDN.

Invalid hostnames (leading dot, trailing dot, all-dot strings, uppercase letters, or anything that violates the field constraint) are rejected before the VM boots. The error is set on the owning `Machine.status` and surfaced on the `Cluster.status.conditions` so it shows up under `kubectl describe cluster <name>` and `kubectl get machines -n cpaas-system -o wide` .

Configure Machine Template

The `HCSMachineTemplate` defines the VM specifications for control plane nodes.

WARNING

Storage Requirements

The following data disk mount points are recommended for control plane nodes:

- `/var/lib/etcd` - etcd data (10GB+)
- `/var/lib/kubelet` - kubelet data (100GB+)
- `/var/lib/containerd` - container runtime data (100GB+)

The `/var/cpaas` path stores platform state and logs. Declare it in `HCSMachineConfigPool.spec.configs[].persistentDisks[]` when it must survive VM replacement.

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineTemplate
metadata:
  name: <cluster-name>-control-plane
  namespace: cpaas-system
spec:
  template:
    spec:
      imageName: <vm-image-name>
      flavorName: <instance-flavor>
      availabilityZone: <availability-zone>
      rootVolume:
        type: SSD
        size: 100
      configPoolRef:
        name: <cluster-name>-cp-pool
      dataVolumes:
        - size: 10
          type: SSD
          mountPath: /var/lib/etcd
          format: xfs
        - size: 100
          type: SSD
          mountPath: /var/lib/kubelet
          format: xfs
        - size: 100
          type: SSD
          mountPath: /var/lib/containerd
          format: xfs
```

Parameter	Type	Required	Desc
`.spec.template.spec.imageName`	string	Yes	HCS the n OS ir the ta

Parameter	Type	Required	Desc
			was u place `<ala` in the The e with e releas hardc value
`.spec.template.spec.flavorName`	string	Yes	Provi HCS again Do nc UI dis
`.spec.template.spec.availabilityZone`	string	No	Provi HCS again Every from the s omitt selec as av See A Place
`.spec.template.spec.rootVolume.type`	string	Yes	Volur `SATA`
`.spec.template.spec.rootVolume.size`	int	Yes	Syste
`.spec.template.spec.configPoolRef.name`	string	Yes	Refer HCSI name
`.spec.template.spec.dataVolumes[]`	array	No	Data confi

Parameter	Type	Required	Desc
`.spec.template.spec.dataVolumes[].size`	int	Yes*	Data
`.spec.template.spec.dataVolumes[].type`	string	Yes*	Volur
`.spec.template.spec.dataVolumes[].mountPath`	string	Yes*	Mour
`.spec.template.spec.dataVolumes[].format`	string	Yes*	File s (`xfs`

*Required when dataVolumes is specified.

`dataVolumes[]` are recreated with the ECS. Do not use them for `/var/cpaas` or any other path that must survive rolling replacement. Put those paths in `HCSMachineConfigPool.spec.configs[].persistentDisks[]`.

Note: Do not set runtime identity fields such as `providerID` or `serverId` in `HCSMachineTemplate` manifests. The provider assigns these values when it creates HCS instances.

Note: The ACP web UI loads flavors and availability zones through the selected HCS credential. When you write YAML, get the exact provider-recognized `Flavor.Name` and `ZoneName` values from the HCS administrator because the native HCS tenant console might not expose them.

Configure KubeadmControlPlane

The `KubeadmControlPlane` defines the Kubernetes control plane configuration.

> **Kubernetes 1.35 kubelet settings**
>
> `imagePullCredentialsVerificationPolicy: NeverVerify` is required only starting with Kubernetes 1.35. Omit this parameter when creating a cluster with Kubernetes 1.34 or earlier.

```yaml
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster-name>-kcp
  namespace: cpaas-system
spec:
  replicas: 3
  version: <kubernetes-version>
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  kubeadmConfigSpec:
    files:
      - path: /etc/kubernetes/admission/psa-config.yaml
        owner: root:root
        permissions: "0644"
        content: |
          apiVersion: apiserver.config.k8s.io/v1
          kind: AdmissionConfiguration
          plugins:
          - name: PodSecurity
            configuration:
              apiVersion: pod-security.admission.config.k8s.io/v1
              kind: PodSecurityConfiguration
              defaults:
                enforce: "privileged"
                enforce-version: "latest"
                audit: "baseline"
                audit-version: "latest"
                warn: "baseline"
                warn-version: "latest"
              exemptions:
                usernames: []
                runtimeClasses: []
                namespaces:
                - kube-system
                - cpaas-system
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.jso
n
        owner: root:root
        permissions: "0644"
        content: |
```

```
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "imagePullCredentialsVerificationPolicy": "NeverVerify",
          "protectKernelDefaults": true,
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
          "streamingConnectionIdleTimeout": "5m",
          "clientCAFile": "/etc/kubernetes/pki/ca.crt"
        }
    - path: /etc/kubernetes/encryption-provider.conf
      owner: root:root
      permissions: "0600"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: EncryptionConfiguration
        resources:
        - resources:
          - secrets
          providers:
          - aescbc:
              keys:
              - name: key1
                secret: bootstrap-placeholder
    - path: /etc/kubernetes/audit/policy.yaml
      owner: root:root
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        rules:
        - level: Metadata
  clusterConfiguration:
    imageRepository: <image-repository>
    dns:
      imageTag: <dns-image-tag>
    etcd:
      local:
        imageTag: <etcd-image-tag>
    apiServer:
      extraArgs:
        audit-log-format: json
        audit-log-mode: batch
        audit-log-path: /etc/kubernetes/audit/audit.log
```

```
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          admission-control-config-file: /etc/kubernetes/admission/psa-co
nfig.yaml
          encryption-provider-config: /etc/kubernetes/encryption-provide
r.conf
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
          profiling: "false"
          tls-min-version: VersionTLS12
        extraVolumes:
          - name: vol-dir-0
            hostPath: /etc/kubernetes
            mountPath: /etc/kubernetes
            pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
          flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
    postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
    initConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          protect-kernel-defaults: "true"
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
    joinConfiguration:
      patches:
        directory: /etc/kubernetes/patches
      nodeRegistration:
        kubeletExtraArgs:
          node-labels: "kube-ovn/role=master"
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/vol
ume/exec/"
```

```
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: HCSMachineTemplate
      name: <cluster-name>-control-plane
```

The HCS controller also injects files while resolving cloud-init data. It writes `/etc/kubernetes/pki/kubelet.crt`, `/etc/kubernetes/pki/kubelet.key`, and `/etc/kubernetes/encryption-provider.conf` for control plane machines. For the first control plane machine, the controller generates the encryption provider configuration. After the control plane is initialized, it tries to reuse the existing `kube-apiserver` encryption provider configuration. If you include a bootstrap file at `/etc/kubernetes/encryption-provider.conf`, treat it as a placeholder because the controller-generated or synchronized file takes precedence.

Note: Configure `apiServer.extraArgs` and `apiServer.extraVolumes` together. If the volume is not mounted, `kube-apiserver` cannot read the files written under `/etc/kubernetes`.

Note: The `rolloutStrategy.rollingUpdate.maxSurge: 0` example above is for highly available static-IP control planes. Keep this setting for fixed-size control plane pools with at least three replicas so replacements happen in a scale-down-then-scale-up order. If you create a single-control-plane HCS cluster (`spec.replicas: 1`), do not copy the `rolloutStrategy` block into the create manifest. `KubeadmControlPlane` validation rejects that scale-in style rollout configuration for a single replica.

Note: HCS also supports creating a single-control-plane cluster by setting `spec.replicas: 1` and preparing one control plane config entry in the referenced `HCSMachineConfigPool`. Treat this as a creation-only topology, and leave the rollout strategy unset in that create manifest. A single-control-plane HCS cluster cannot be upgraded.

Use the OS Support Matrix only for the component versions it explicitly lists, such as coredns and etcd image tags for supported Alauda OS images. It is not a complete source for all HCS manifest values. Before you apply this YAML, also use the approved release baseline for values such as `imageRepository`, DNS image repository, Kube-OVN version, Kube-OVN join CIDR, Pod CIDR, and Service CIDR.

Configure HCSCluster

The `HCSCluster` resource defines the HCS infrastructure configuration.

The HCS provider creates an Elastic Load Balance (ELB) on the HCS platform for the Kubernetes API server. This ELB must keep Hybrid Load Balancing enabled so cluster nodes can also reach the API server through the ELB address.

For the documented HCS workflow, provide `vipAddress`, `elbVirsubnetL4Ips`, and `elbVirsubnetL7Ips`. Each `elbVirsubnetL4Ips[].ips` and `elbVirsubnetL7Ips[].ips` entry must contain two IPs.

Leave `vipDomainName` unset — cluster access by domain is not supported; see the field table below.

List every cluster subnet in `spec.network.subnets` before you reference it anywhere else. `vipSubnetName`, `elbVirsubnetL4Ips[].subnetName`, `elbVirsubnetL7Ips[].subnetName`, and the `subnetName` values used by `HCSMachineConfigPool` must all exist in `spec.network.subnets`.

For the initial cluster create flow, the controller can resolve existing subnet metadata from `name`. For an existing Ready cluster, append a full subnet object instead of only `name`. Include `id`, and include `neutronSubnetId` for any subnet that the control plane ELB will use. Keep `cidr`, `gatewayIp`, `primaryDNS`, and `secondaryDNS` in the subnet inventory as well.

Do not disable Hybrid Load Balancing on the provider-created ELB after the cluster is created. The cluster depends on that ELB mode so nodes can reach the API server through the ELB address.

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSCluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
spec:
  # Optional. Include only when you want provider-managed control-plane E
CS placement.
  controlPlaneHA:
    enabled: true
    policy: soft-anti-affinity
  controlPlaneLoadBalancer:
    vipAddress: <control-plane-vip-address>
    vipSubnetName: <vip-subnet-name>
    elbVirsubnetL4Ips:
      - subnetName: <subnet-name>
        ips:
          - <l4-ip-1>
          - <l4-ip-2>
    elbVirsubnetL7Ips:
      - subnetName: <subnet-name>
        ips:
          - <l7-ip-1>
          - <l7-ip-2>
  networkType: kube-ovn
  network:
    vpc:
      name: <vpc-name>
    subnets:
      - name: <subnet-name>
    securityGroup:
      name: <security-group-name>
  identityRef:
    name: <credential-secret-name>
```

Parameter	Type	Req
`.spec.networkType`	string	Yes
`.spec.network.vpc.name`	string	Yes

Parameter	Type	Req
`.spec.network.subnets[].name`	string	Yes
`.spec.network.subnets[].id`	string	Reco
`.spec.network.subnets[].neutronSubnetId`	string	Reco ELB
`.spec.network.subnets[].cidr`	string	Reco
`.spec.network.subnets[].gatewayIp`	string	Reco
`.spec.network.subnets[].primaryDNS`	string	Reco
`.spec.network.subnets[].secondaryDNS`	string	Reco
`.spec.network.securityGroup.name`	string	Yes
`.spec.identityRef.name`	string	Yes

Parameter	Type	Requ
`.spec.controlPlaneHA.enabled`	bool	Requ `con` is se
`.spec.controlPlaneHA.policy`	string	Requ `con` is se
`.spec.controlPlaneLoadBalancer`	object	Yes
`.spec.controlPlaneLoadBalancer.vipAddress`	string	Yes
`.spec.controlPlaneLoadBalancer.vipSubnetName`	string	Yes
`.spec.controlPlaneLoadBalancer.vipDomainName`	string	No
`.spec.controlPlaneLoadBalancer.elbVirsubnetL4Ips[]`	array	Yes
`.spec.controlPlaneLoadBalancer.elbVirsubnetL4Ips[].subnetName`	string	Yes

Parameter	Type	Req
`.spec.controlPlaneLoadBalancer.elbVirsubnetL4Ips[].ips[]`	string	Yes
`.spec.controlPlaneLoadBalancer.elbVirsubnetL7Ips[]`	array	Yes
`.spec.controlPlaneLoadBalancer.elbVirsubnetL7Ips[].subnetName`	string	Yes
`.spec.controlPlaneLoadBalancer.elbVirsubnetL7Ips[].ips[]`	string	Yes

Do not include `spec.controlPlaneEndpoint` in the create manifest. In the HCS create flow, the controller derives and populates this field from `spec.controlPlaneLoadBalancer` after the `HCSCluster` is created. Do not set `controlPlaneEndpoint` manually, and do not add an empty `controlPlaneEndpoint` object. If `controlPlaneEndpoint` is explicitly present in the manifest, it must include both `host` and `port`.

`controlPlaneHA` is optional. When you include it, both `enabled` and `policy` are required. Use `anti-affinity` for strict host separation. Use `soft-anti-affinity` when you prefer host spread but do not want ECS creation to fail only because HCS cannot satisfy the hard placement rule. For planning guidance, including the rolling replacement requirement when enabling the feature on an existing cluster, see Control Plane HA Placement Plan.

Configure Cluster

The `Cluster` resource in Cluster API declares the cluster and references the control plane and infrastructure resources.

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster-name>
  namespace: cpaas-system
  annotations:
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8s.io/v1beta1
    capi.cpaas.io/resource-kind: HCSCluster
    cpaas.io/kube-ovn-join-cidr: <kube-ovn-join-cidr>
    cpaas.io/kube-ovn-version: <kube-ovn-version>
  labels:
    cluster-type: HCS
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - <pods-cidr>
    services:
      cidrBlocks:
        - <services-cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster-name>-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: HCSCluster
    name: <cluster-name>
```

Annotation	Description
`cpaas.io/sentry-deploy-type`	Deployment type, set to `Baremetal`
`cpaas.io/alb-address-type`	ALB address type, set to `ClusterAddress`
`capi.cpaas.io/resource-group-version`	Resource group version
`capi.cpaas.io/resource-kind`	Resource type, set to `HCSCluster`

Annotation	Description
`cpaas.io/kube-ovn-join-cidr`	Kube-OVN join subnet CIDR
`cpaas.io/kube-ovn-version`	Kube-OVN version

Deploy Worker Nodes

The control plane alone is not a usable cluster. After the control plane is Ready, create the worker resources in the following order.

Before you prepare worker YAML, complete the HCS input checklist in Infrastructure Resources for Huawei Cloud Stack. In particular, list every worker subnet in `HCSCluster.spec.network.subnets`, allocate worker IPs from planned free IP ranges, and collect the provider-recognized `flavorName` and `availabilityZone` API values. If you add a new worker subnet to an existing Ready cluster, patch `HCSCluster.spec.network.subnets` with the full subnet object instead of adding only the subnet name.

Step 1: Configure Machine Configuration Pool

The `HCSMachineConfigPool` defines the network configuration and any pool-managed persistent disks for worker node VMs. You must plan and configure the IP addresses, hostnames, persistent disk slots, and other parameters before deployment.

> **WARNING**
>
> **Pool Size Requirement**
>
> The pool must include at least as many entries as the number of worker nodes you plan to deploy. Insufficient entries will prevent node deployment.

Use one subnet selector per `networks[]` entry. For new manifests, set either `subnetName` or `subnetId`, but not both. Existing manifests may keep the deprecated `subenetName` field; if you also add `subnetName` while updating that manifest, its value must exactly match `subenetName`. Do not supply conflicting values across `subenetName`, `subnetName`, and `subnetId`.

If you use `subnetName` for worker nodes, include the same subnet name in the parent `HCSCluster.spec.network.subnets` list before you create or scale the worker pool. For an existing Ready cluster, append the full subnet object, including the subnet ID, instead of adding only the subnet name.

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineConfigPool
metadata:
  name: <cluster-name>-worker-pool
  namespace: cpaas-system
  labels:
    cluster.x-k8s.io/cluster-name: <cluster-name>
spec:
  configs:
    - hostname: <worker-1-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <worker-1-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: <worker-2-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <worker-2-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
    - hostname: <worker-3-hostname>
      networks:
        - subnetName: <subnet-name>
          ipAddress: <worker-3-ip>
      persistentDisks:
        - slot: 0
          size: 100
          type: SSD
          mountPath: /var/cpaas
          format: xfs
```

Parameter	Type	Required	D
`.spec.configs[]`	array	Yes	N w co
`.spec.configs[].hostname`	string	Yes	V lo nu (th ar lo nu no ch
`.spec.configs[].networks[]`	array	Yes	N no co V
`.spec.configs[].networks[].subnetName`	string	No*	R su ne
`.spec.configs[].networks[].subnetId`	string	No*	S fie s th ar
`.spec.configs[].networks[].ipAddress`	string	Yes	S th
`.spec.configs[].persistentDisks[]`	array	No	E su de re
`.spec.configs[].persistentDisks[].slot`	int	Yes*	D m

Parameter	Type	Required	D
			c m c fo h
`.spec.configs[].persistentDisks[].size`	int	Yes*	E F E `1` E di th
`.spec.configs[].persistentDisks[].type`	string	Yes*	E av ta zo
`.spec.configs[].persistentDisks[].mountPath`	string	No	G U fo th re
`.spec.configs[].persistentDisks[].format`	string	No	Fi or us
`.spec.configs[].persistentDisks[].mountOptions`	array	No	M or us `d`

*For new manifests, set either `subnetName` or `subnetId`. Existing manifests may continue to use `subenetName`, and may add `subnetName` only if both fields use the same value. Do not provide conflicting subnet selector values.

Persistent disk fields are required when `persistentDisks` is specified.

Use `persistentDisks[]` for node-local state that must survive worker replacement. Do not declare the same mount path in `HCSMachineTemplate.spec.template.spec.dataVolumes[]`.

Note: The CRD schema lists `subnetName`, `subenetName`, and `subnetId` as optional fields and does not express their allowed combinations. Follow the provider-level rules above when writing manifests.

Note: `networks[]` can contain more than one entry when a worker node needs multiple NICs. The current provider only uses each entry to attach a NIC with a subnet selector and static IP. It does not support per-NIC role declarations, default gateway selection, static routes, route metrics, or per-NIC DNS settings.

Pool-Managed Persistent Disks for Workers

Declare worker-node disks that must survive replacement in the matching `HCSMachineConfigPool.spec.configs[].persistentDisks[]` entry. Use this model for `/var/cpaas` and for any other node-local state that must be retained during rolling replacement.

- Keep `HCSMachineTemplate.spec.template.spec.dataVolumes[]` for temporary disks that may be recreated with each ECS.
- Keep slots unique and contiguous from `0` for each hostname. The provider uses `(hostname, slot)` as the persistent-disk identity.
- Treat `slot`, `size`, `type`, `format`, and `mountPath` as immutable after the provider accepts the entry.
- You can update `mountOptions`. The change takes effect after the worker is replaced.
- You can append new `persistentDisks[]` entries. The provider creates or claims the EVS disk, but it does not hot-mount the disk into the running ECS. Trigger a rolling replacement with `MachineDeployment.spec.strategy.rollingUpdate.maxSurge: 0` before you expect the new disk to be formatted and mounted inside the guest OS.

To inspect persistent-disk runtime state during worker operations, check the pool status:

```
kubectl get hcsmachineconfigpool <cluster-name>-worker-pool -n cpaas-syst
em -o yaml
```

Step 2: Configure Machine Template

The `HCSMachineTemplate` defines the VM specifications for worker nodes.

Configure worker nodes with a system volume and temporary data volumes for paths that may be recreated with each ECS, such as `/var/lib/kubelet` and `/var/lib/containerd`. Put `/var/cpaas` in `HCSMachineConfigPool.spec.configs[].persistentDisks[]` when platform state must survive worker replacement.

Use the provider-recognized `flavorName` and `availabilityZone` API values when you prepare the worker template. These values are not the tenant UI display names.

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: HCSMachineTemplate
metadata:
  name: <cluster-name>-worker-template
  namespace: cpaas-system
spec:
  template:
    spec:
      imageName: <vm-image-name>
      flavorName: <instance-flavor>
      availabilityZone: <availability-zone>
      rootVolume:
        type: SSD
        size: 100
      configPoolRef:
        name: <cluster-name>-worker-pool
      dataVolumes:
        - size: 20
          type: SSD
          mountPath: /var/lib/kubelet
          format: xfs
        - size: 20
          type: SSD
          mountPath: /var/lib/containerd
          format: xfs
```

Parameter	Type	Required	Desc
`.spec.template.spec.imageName`	string	Yes	VM ir
`.spec.template.spec.flavorName`	string	Yes	Provi HCS again
`.spec.template.spec.availabilityZone`	string	No	Provi HCS again Every from the s omitt selec as av See A Place
`.spec.template.spec.rootVolume.type`	string	Yes	Volur
`.spec.template.spec.rootVolume.size`	int	Yes	Syste
`.spec.template.spec.configPoolRef.name`	string	Yes	Refer HCSI name
`.spec.template.spec.dataVolumes[]`	array	No	Data confi
`.spec.template.spec.dataVolumes[].size`	int	Yes*	Disk
`.spec.template.spec.dataVolumes[].type`	string	Yes*	Volur
`.spec.template.spec.dataVolumes[].mountPath`	string	Yes*	Mour
`.spec.template.spec.dataVolumes[].format`	string	Yes*	File s

*Required when dataVolumes is specified.

`dataVolumes[]` are recreated with the ECS. Do not use them for `/var/cpaas` or any other path that must survive rolling replacement.

Note: Do not set runtime identity fields such as `providerID` or `serverId` in `HCSMachineTemplate` manifests. The provider assigns these values when it creates HCS instances.

Note: The ACP web UI loads flavors and availability zones through the selected HCS credential. When you write YAML, get the exact provider-recognized `Flavor.Name` and `ZoneName` values from the HCS administrator because the native HCS tenant console might not expose them.

Step 3: Configure Bootstrap Template

The `KubeadmConfigTemplate` defines the bootstrap configuration for worker nodes.

`imagePullCredentialsVerificationPolicy: NeverVerify` is required only starting with Kubernetes 1.35. Omit this parameter when creating workers with Kubernetes 1.34 or earlier.

```yaml
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster-name>-worker-kct
  namespace: cpaas-system
spec:
  template:
    spec:
      files:
        - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.j
son
          owner: root:root
          permissions: "0644"
          content: |
            {
              "apiVersion": "kubelet.config.k8s.io/v1beta1",
              "kind": "KubeletConfiguration",
              "imagePullCredentialsVerificationPolicy": "NeverVerify",
              "protectKernelDefaults": true,
              "staticPodPath": null,
              "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
              "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key",
              "streamingConnectionIdleTimeout": "5m",
              "clientCAFile": "/etc/kubernetes/pki/ca.crt"
            }
      postKubeadmCommands:
        - chmod 600 /var/lib/kubelet/config.yaml
      joinConfiguration:
        patches:
          directory: /etc/kubernetes/patches
        nodeRegistration:
          kubeletExtraArgs:
            volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugins/v
olume/exec/"
```

The HCS controller injects `/etc/kubernetes/pki/kubelet.crt` and `/etc/kubernetes/pki/kubelet.key` while resolving worker cloud-init data. The kubelet patch above configures kubelet to use those controller-provided certificate files.

Step 4: Configure Machine Deployment

The `MachineDeployment` orchestrates the creation and management of worker nodes.

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster-name>-md-0
  namespace: cpaas-system
spec:
  clusterName: <cluster-name>
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
  template:
    spec:
      clusterName: <cluster-name>
      version: <kubernetes-version>
      nodeDrainTimeout: 1m
      nodeDeletionTimeout: 5m
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster-name>-worker-kct
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: HCSMachineTemplate
        name: <cluster-name>-worker-template
```

Parameter	Type	Required	Desc
`.spec.clusterName`	string	Yes	Targe
`.spec.replicas`	int	Yes	Numl
`.spec.template.spec.bootstrap.configRef`	object	Yes	Refe Kube

Parameter	Type	Required	Desc
`.spec.template.spec.infrastructureRef`	object	Yes	Refer HCSI
`.spec.template.spec.version`	string	Yes	Kube
`.spec.strategy.rollingUpdate.maxSurge`	int	No	Maxi desir
`.spec.strategy.rollingUpdate.maxUnavailable`	int	No	Maxi node

Cluster Verification

After deploying all cluster resources, verify that the cluster has been created successfully.

Using kubectl

```
# Check cluster status
kubectl get cluster -n cpaas-system <cluster-name>

# Verify control plane nodes
kubectl get kubeadmcontrolplane -n cpaas-system <cluster-name>-kcp

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

# Verify cluster deployment status
kubectl get clustermodule <cluster-name> -o jsonpath='{.status.base.deplo
yStatus}'
```

Verify Control Plane HA

If you enabled `HCSCluster.spec.controlPlaneHA`, inspect the `HCSCluster` condition first:

```
kubectl get hcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='{range .status.conditions[?(@.type=="ControlPlaneHARead
y")]}{.status}{" "}{.reason}{" "}{.message}{"\n"}{end}'
```

Interpret the condition as follows:

Status and reason	Meaning	Next step
`True` `ControlPlaneHAReady`	The provider-managed server group exists and the current control-plane ECS members are tracked.	No action is required.
`False` `ControlPlaneHAPending`	The provider is waiting for a recoverable state, such as ECS creation or server group membership convergence.	Read the message and wait for the referenced control-plane machine or ECS operation to finish.
`False` `ControlPlaneHAFailed`	The provider could not create, validate, update, or clean up the server group, or HCS rejected ECS scheduling for the selected policy.	Read the message, then check HCS capacity, the selected policy, and controller logs if needed.

Inspect the observed server group and membership snapshot:

```
kubectl get hcscluster <cluster-name> -n cpaas-system \
  -o jsonpath='serverGroup={.status.controlPlaneHA.name}{" id="}{.status.
controlPlaneHA.id}{" policy="}{.status.controlPlaneHA.policy}{"\n"}{range
.status.controlPlaneHA.members[*]}{.machineName}{" "}{.serverID}{"\n"}{en
d}'
```

For `anti-affinity`, HCS treats the server group policy as a hard scheduling constraint. If capacity is insufficient, ECS creation can fail and the condition message should be used as the first diagnostic signal. For `soft-anti-affinity`, HCS tries to spread members but may still place multiple ECS instances on the same host.

Expected Results

A successfully created cluster should show:

- Cluster status: **Running** or **Provisioned**
- All control plane machines: **Running**
- Kubernetes nodes: **Ready**
- Cluster Module Status: **Completed**

Managing Nodes

For day-2 node operations such as scaling, template updates, and web UI node pools, refer to Manage Nodes on Huawei Cloud Stack.

Upgrading Clusters

For instructions on upgrading cluster components, refer to Upgrade Kubernetes.

Troubleshooting

If the cluster reaches `Provisioned` but never becomes Ready — for example, workload nodes stay `NotReady` because the CNI is not deployed — start with the provider-agnostic Troubleshoot a Workload Cluster Stuck in Provisioned.

For HCS-specific failure patterns (for example, `kubeadm init` never completes because a dotted `HCSMachineConfigPool` hostname produced a POSIX hostname containing dots), see Troubleshoot Huawei Cloud Stack Workload Clusters.

Troubleshoot Huawei Cloud Stack Workload Clusters

This guide covers troubleshooting patterns that are specific to Huawei Cloud Stack (HCS) workload clusters managed through Cluster API.

If your cluster reaches `Cluster.status.phase = Provisioned` but the import flow does not complete and workload nodes stay `NotReady` because the CNI is missing, start with the provider-agnostic guide Troubleshoot a Workload Cluster Stuck in Provisioned. That guide covers the shared diagnostic flow (`global` cluster import controller, `sentry` ServiceAccount, `clusters.platform.tkestack.io` invariants) for every Alauda OS infrastructure provider.

If you have already completed the generic flow and the symptoms below also appear, see the HCS-specific pattern on this page.

TOC

Pattern: kubeadm init Never Completes Because the Hostname Is Wrong

Diagnose

Mitigate

Prevent

Related

Pattern: kubeadm init Never Completes Because the Hostname Is Wrong

Symptoms — all true at the same time:

- `Cluster.status.phase=Provisioned`, `HCSCluster.status.ready=true`, the ELB is up.
- `HCSMachine.status.instanceState=ACTIVE` with an `InternalIP` populated, but `Machine.status.nodeRef` never becomes set.
- `KubeadmControlPlane.status.initialized` stays empty and `status.readyReplicas=0` past the usual init window of about 5 to 10 minutes.
- The `cluster-api-provider-hcs` controller logs (in `cpaas-system`) repeatedly print `connect: connection refused` against the control plane ELB VIP on port 6443.
- The `HCSMachineConfigPool.spec.configs[].hostname` value chosen for the node contains a dot (FQDN style, for example `master-1.example.org`).

In `cluster-api-provider-hcs` releases earlier than v1.0.1, a dotted `HCSMachineConfigPool.spec.configs[].hostname` is rendered into cloud-init as the full FQDN string in the `hostname` field, with `prefer_fqdn_over_hostname: true`. The resulting node has a POSIX hostname that contains dots, which kubeadm init does not handle, so kube-apiserver never starts.

Diagnose

If you can reach the node (HCS console, jump host, or an in-cluster debug pod):

```
# On the affected node
hostname
hostname -f
cat /etc/hosts
sudo cat /var/lib/cloud/instance/user-data.txt | head -40
sudo journalctl -u cloud-init -b 0 | grep -E "hostname|fqdn|update_etc_ho
sts"
```

Indicators that you are hitting this pattern:

- `hostname` returns the full dotted string instead of just the short label.
- `hostname -f` returns `Name or service not known` or `Temporary failure in name resolution`.
- `/etc/hosts` does **not** contain a line of the form `<node-ip> <fqdn> <short>`.
- The cloud-init user-data on the node shows `prefer_fqdn_over_hostname: true` and does not set `manage_etc_hosts`.

Mitigate

Upgrade the `cluster-api-provider-hcs` plugin on the `global` cluster to the version delivered with this release, then trigger a rolling replacement of the affected control plane and worker machines so the new cloud-init runs on freshly booted VMs. The step-by-step procedure is documented in Configure FQDN Hostname on Existing HCS Clusters.

Manual edits to `/etc/hostname` and `/etc/hosts` on an existing Alauda OS node do not persist: cloud-init re-renders these files on every boot, including reboots triggered by `transactional-update`, OS patching, or `systemctl reboot`. Rolling replacement is the supported migration path.

Prevent

Creating the cluster with the `cluster-api-provider-hcs` plugin delivered with this release avoids the pattern entirely. New manifests for HCS should follow Hostname behavior on the node on the cluster create page when picking dotted versus short hostnames in `HCSMachineConfigPool`.

Related

- Troubleshoot a Workload Cluster Stuck in Provisioned — provider-agnostic import-controller diagnostic flow that applies before any HCS-specific pattern on this page.
- Configure FQDN Hostname on Existing HCS Clusters — rolling-replacement migration procedure for clusters created before `cluster-api-provider-hcs` v1.0.1.

- Create Cluster on Huawei Cloud Stack — manifest reference for new clusters, including Hostname behavior on the node.

VMware vSphere

Overview

Install the Provider

Infrastructu

Create a Cluster

VMware vSphere Provider

Install the Provider

Install the Provider

When You Need This

Prerequisites

Steps

Upgrading the Provider

Standby Management Clusters

Next Steps

Infrastructure Resources

Infrastructure Resources

Scenarios

Prerequisites

How to Use This Checklist

Parameter Sources and Safe Collection

Terminology

Management Cluster Prerequisites

vCenter and Template Prerequisites

Load Balancer Prerequisites
Cluster Baseline Parameters
Minimum Single-Datacenter Parameters
Optional Creation-Time Topology Parameters
Final Readiness Check
Next Steps

Create a Cluster

Create a Cluster

Provider Requirements
Scenarios
How to Use This Page
Prerequisites
Control Plane Endpoint Modes
Key Objects
Local File Layout
Steps
Verification
Troubleshooting
Creation-Time Topology Variants
Next Steps

The Infrastructure Provider for Virtualization Platform creates and manages Alauda OS clusters on the VMware vSphere platform. This provider is built on top of Cluster API Provider vSphere (CAPV) and integrates with vCenter to provision and manage virtual machines, networks, and storage for Kubernetes clusters.

TOC

- Install the Provider
- Infrastructure Resources
- Create a Cluster
- Overview
- Key Features
- Supported Resources
- Documentation

Overview

VMware vSphere is the industry-leading virtualization platform from VMware. The vSphere Provider connects directly to vCenter to manage VM lifecycle, and extends CAPV with declarative node-slot allocation so that hostnames, static IP addresses, and persistent disks can be pre-defined before a cluster is provisioned.

INFO

Endpoint modes and disk lifecycle

The provider supports two control plane endpoint modes: an external LoadBalancer, which is the default, and a provider-managed Self-built VIP (`VSphereCluster.spec.controlPlaneLoadBalancer.type: internal`, Alive with keepalived and IPVS). The Self-built VIP requires ACP `v4.4` or later and is a creation-time choice: it is available only to clusters created with provider `v1.0.17` or later, and a cluster that already exists —

including one whose provider was upgraded from an earlier release — keeps its external LoadBalancer and cannot adopt it. Machine-config-pool persistent disks follow the provider-managed `InUse → Released → reuse or reclaim` lifecycle; see VMware vSphere Infrastructure Preparation.

Key Features

- **vCenter Integration**: Direct vCenter connectivity to clone VM templates, manage VM lifecycle, and place workloads on target datacenters, compute clusters, and datastores
- **Two Control Plane Endpoint Modes**: an external LoadBalancer you own, or a provider-managed Self-built VIP on the control-plane nodes (`type: internal` , Alive with keepalived and IPVS, where the bootstrap VIP is handed over to Alive after `kubeadm init`), which requires ACP `v4.4` or later. The mode is immutable after cluster creation
- **Machine Configuration Pools**: `VSphereMachineConfigPool` defines per-node slots with fixed hostnames, static IPs, and per-slot data-disk layouts. Released slots can be reused by replacement Machines in the same existing pool, but slots and VMDKs are not shared across different pool objects
- **Static IP Allocation**: Static IP, gateway, and DNS configuration per node slot for enterprise network environments where DHCP is not available
- **Multi-NIC Support**: Primary and additional NICs per node, where the primary NIC drives the kubelet `node-ip` and additional NICs are attached in declared order for management, storage, or service networks
- **Multiple Datacenters and Failure Domains**: Distribute control plane and worker nodes across multiple vCenter datacenters or compute clusters through `VSphereFailureDomain` and `VSphereDeploymentZone` for high availability
- **Flexible Data Disks**: Declarative per-slot persistent and ephemeral data disks with optional format/mount, `wipeFilesystem` control on persistent disks, raw-device mode exposed at `/dev/disk/by-capv/<name>` , and delayed release so replacement Machines can reuse the slot and VMDKs before provider-managed reclaim
- **Immutable-Template Rolling Updates**: Control plane and worker updates use new `VSphereMachineTemplate` and `KubeadmConfigTemplate` resources, enabling reproducible replacement and same-minor template recovery

Supported Resources

The table below lists the primary user-facing resources used by the workflows in this documentation. Controller-reconciled resources (for example `VSphereVM`) and resources not exercised by the documented workflows (for example `VSphereClusterIdentity`, which is an alternative to the `Secret`-based `identityRef` used here) are covered in the Provider APIs reference for completeness.

Resource	Description
`VSphereCluster`	Represents vSphere infrastructure including vCenter endpoint, control plane endpoint, and failure-domain selector
`VSphereMachine`	Represents a virtual machine instance in vSphere
`VSphereMachineTemplate`	Template for creating vSphere machines with VM, CPU, memory, network, and disk specifications
`VSphereMachineConfigPool`	Pool of pre-defined node slots with hostnames, static IPs, and per-slot data-disk layouts
`VSphereFailureDomain`	Describes a datacenter, compute cluster, datastore, and network topology used as a failure domain
`VSphereDeploymentZone`	Binds a `VSphereFailureDomain` to a vCenter server and placement constraints such as resource pool

Documentation

- Installation Guide
- Infrastructure Preparation
- Create a Cluster
- Manage Nodes
- Upgrade Kubernetes
- Provider APIs

Install the VMware vSphere Provider

This document explains how to install the Infrastructure Provider for Virtualization Platform and the Cluster Bootstrap Provider, and how to verify that the required Cluster API components are available on the `global` cluster.

INFO

The provider package bundles the CAPV controller, the vSphere CPI image, the provider chart, and the admission webhooks that enforce the fixed-IP rollout rules. For those rules, see Provider Requirements.

TOC

- When You Need This
- Prerequisites
- Steps
  - Download the required packages
  - Upload the packages
  - Install the cluster plugins
  - Verify the installation
- Upgrading the Provider
- Standby Management Clusters
- Next Steps

When You Need This

Complete this page after the `global` cluster is installed and before you create the first workload cluster on VMware vSphere. It is required regardless of the IaaS platform that the `global` cluster itself runs on. In a disaster recovery (DR) pair, complete it on **both** `global` clusters, so that the standby can reconcile the same workload clusters after a failover. Upload the packages to each `global` cluster.

This is a separate task from the provider steps in Installing the global Cluster. Those steps upload provider packages to the temporary bootstrap cluster to create the `global` cluster itself.

Prerequisites

Before installing the provider, ensure the following conditions are met:

- You can access the Alauda Container Platform `global` cluster.
- You can access Customer Portal to download provider packages.
- The vCenter account that clusters will reference through `VSphereCluster.spec.identityRef` can list, create, and set VM custom attributes. Grant the **Global** → **Manage custom attributes** and **Global** → **Set custom attribute** privileges. The provider requires them whenever a `VSphereCluster` or its CAPI `Cluster` carries any `vsphere.cluster.x-k8s.io/*` annotation, and a missing privilege fails VM reconciliation rather than skipping the metadata write.
- To create or upgrade workload clusters, you work against the active management cluster, not a disaster-recovery standby. See Standby Management Clusters.

Steps

1 Download the required packages

To create clusters on VMware vSphere, you need two cluster plugins. Download both from the Customer Portal. For ACP 4.5, they are listed under **Common ACP Extensions**; see Download.

1. **Alauda Container Platform Cluster Bootstrap Provider**
2. **Alauda Container Platform Infrastructure Provider for Virtualization Platform**

2 Upload the packages

Upload both packages to the platform with `violet push`. For the command and its parameters, refer to Upload Packages.

3 Install the cluster plugins

Install both cluster plugins on the `global` cluster. For the procedure, refer to Cluster Plugin.

4 Verify the installation

Check the module state:

```
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-vsphere
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-kubeadm
```

Both plugins must be present and in the `Running` state.

Check the registered CRDs:

```
kubectl get crd | grep -E '^vsphere[a-z]+\.infrastructure\.cluster\.x-k8s\.io\b'
```

Expected output includes all nine CRDs:

1. `vphereclusters.infrastructure.cluster.x-k8s.io`
2. `vphereclusteridentities.infrastructure.cluster.x-k8s.io`

3. `vsphereclustertemplates.infrastructure.cluster.x-k8s.io`
4. `vspheredeploymentzones.infrastructure.cluster.x-k8s.io`
5. `vspherefailuredomains.infrastructure.cluster.x-k8s.io`
6. `vspheremachines.infrastructure.cluster.x-k8s.io`
7. `vspheremachineconfigpools.infrastructure.cluster.x-k8s.io`
8. `vspheremachinetemplates.infrastructure.cluster.x-k8s.io`
9. `vspherevms.infrastructure.cluster.x-k8s.io`

Check the registered admission webhooks. The `ValidatingWebhookConfiguration` object name varies by packaging, so match on the webhook names instead:

```
kubectl get validatingwebhookconfiguration \
  -o jsonpath='{range .items[*]}{range .webhooks[*]}{.name}{"\n"}{en
d}{end}' \
  | grep -E 'vsphere|capv'
```

Expected output includes:

- `validation.vsphere cluster.infrastructure.cluster.x-k8s.io`
- `validation.vspheremachineconfigpool.infrastructure.cluster.x-k8s.io`
- `validation.vspheremachinetemplate.infrastructure.cluster.x-k8s.io`
- `validation.kubeadmcontrolplane.capv.cluster.x-k8s.io`

The last one enforces the `KubeadmControlPlane` fixed-IP rollout rules described in Provider Requirements. The same rules for a `MachineDeployment` are still required by the slot model — a non-compliant rollout stalls for want of a free slot instead of being rejected up front. The `vspheremachineconfigpool` webhook also intercepts `DELETE`, which is what turns the deletion of a pool that still owns `InUse` slots into an immediate rejection rather than a resource held in `Terminating`.

Check the management-cluster permissions if your platform restricts ClusterRoles. Beyond the upstream set, the provider chart grants `capv-manager-role` the following:

- Read access to `clustermodules`, `moduleconfigs`, and `moduleplugins` in `cluster.alauda.io`.

> - Write access to `moduleinfoes` and read access to `moduleinfoes/status`.
> - Read access to `clusters` in `clusterregistry.k8s.io` and `platform.tkestack.io`.
> - `patch` and `update` on `controlplane.cluster.x-k8s.io/kubeadmcontrolplanes`.
> - Access to `vspheremachineconfigpools`.
>
> The `kubeadmcontrolplanes` patch permission is what lets the provider set the CoreDNS and kube-proxy skip annotations described in Provider-managed fields on your manifests.

Upgrading the Provider

The provider is an Aligned extension and is upgraded together with the Distribution Version upgrade of the `global` cluster. Publish the target-version provider package with `violet` as described in Pre-Upgrade Preparation; CVO then upgrades the installed cluster plugin as part of the cluster upgrade. See Upgrade the Distribution Version of the `global` Cluster.

The upgrade also updates the provider CRDs. Some fields move between releases, but **existing resources are migrated for you** — do not hand-edit `VSphereMachineConfigPool` or any other provider object to match a new schema, and do not delete and recreate one. On its first reconcile after the upgrade the controller brings each object up to the current layout:

- The control plane endpoint is not touched. An existing cluster leaves `spec.controlPlaneLoadBalancer` unset, which is what an external LoadBalancer looks like, and it stays that way: the API server address does not change and nothing is deployed into the workload cluster.
- Per-disk state that used to be recorded on `spec.configs[].persistentDisks[].volumePath` and `.diskUUID`, and per-slot reclaim state under `status.configStatuses[].reclaimStatus`, are folded into `status.persistentDiskStatuses[]`. A reclaim that was in flight keeps its phase and task reference and continues, and `reclaimStatus` is then cleared so status has a single source of truth.

- The deprecated `spec` fields are left in the object. Treat them as history and read `status.persistentDiskStatuses[]` instead.
- The migration never overwrites a value that is already set, so it is safe to repeat. The disk part is status-only bookkeeping and runs before vCenter is contacted, so it completes even while vCenter is unreachable.
- It covers the provider's own resources only. The rollout values required on the Cluster API objects — `maxSurge` and `replicas` — are never rewritten, so an existing `KubeadmControlPlane` or `MachineDeployment` that does not carry them still has to be corrected by hand. See Check the manifests against the admission rules.

Nothing in the workload cluster is touched, and no VM is replaced. Three consequences are worth knowing before you upgrade:

- A cluster created on an earlier provider keeps its external LoadBalancer and cannot adopt the Self-built VIP. The endpoint is a deploy-time choice: it cannot be adjusted while the cluster runs, and an upgrade is not an opportunity to change it — filling in `spec.controlPlaneLoadBalancer` on a running cluster is rejected. See Control Plane Endpoint Modes.
- A pool keeps the disk lists it was created with. `spec.configs[].ephemeralDisks[]` is new in provider `v1.0.17`, but the upgrade moves no existing disk into it: a cluster created on an earlier provider keeps `/var/cpaas`, `/var/lib/containerd`, and `/var/lib/etcd` as persistent disks. The persistent-plus-ephemeral split documented for new clusters is a creation-time layout. See Standard data disks.
- Guest-side changes delivered through `guestinfo` user data, such as the pod-log relocation, reach a node only when it is replaced. See Relocating the pod log directory.

Confirm the result with the checks in Verify the installation, then read one pool back:

```
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> \
  -o jsonpath='{range .status.persistentDiskStatuses[*]}{.hostname}{" "}
{.name}{" "}{.phase}{"\n"}{end}'
```

Each disk that was already provisioned before the upgrade appears with a phase, and `status.configStatuses[].reclaimStatus` is gone. A disk on a slot that no Machine has ever allocated has no VMDK yet, so it has no record; that is expected rather than an incomplete migration.

Standby Management Clusters

While the `etcd-sync` ConfigMap exists in the **`kube-public`** namespace, the provider treats the management cluster as a disaster-recovery standby:

- Objects owned by the reserved `global` cluster are still reconciled.
- Every other object is skipped and requeued every 30 seconds.
- An object whose cluster name cannot be determined is skipped as well.

Do not create or upgrade a workload cluster against a standby management cluster. The reconcilers no-op silently — no condition changes and no error appears on the resource, so the workflow looks stalled rather than blocked.

Check before you start:

```
kubectl -n kube-public get configmap etcd-sync -o name 2>/dev/null || ech
o "not standby"
```

A common mistake is to look in `kube-system`. The ConfigMap is in `kube-public`.

For the disaster-recovery workflow itself, see Global Cluster Disaster Recovery.

Next Steps

After the provider is installed, continue with:

- VMware vSphere Infrastructure Preparation
- Create a Cluster on VMware vSphere
- Upgrade Kubernetes on VMware vSphere
- VMware vSphere Provider

VMware vSphere Infrastructure Preparation

This document helps you prepare the infrastructure and collect the values required to create a VMware vSphere workload cluster. Complete this checklist before you apply the manifests in Create a Cluster on VMware vSphere.

TOC

- Scenarios
- Prerequisites
- How to Use This Checklist
- Parameter Sources and Safe Collection
  - Optional vCenter discovery with govc
- Terminology
  - machine config pool
  - Node slot
  - Slot network layout
  - `deviceName`
  - vCenter resource pool
  - Compute cluster
  - Datastore
  - VM template
  - Thumbprint
- Management Cluster Prerequisites
- vCenter and Template Prerequisites

vCenter connection information
VM template requirements
Load Balancer Prerequisites
Cluster Baseline Parameters
Registry tag validation
Validate the locally preloaded images
Minimum Single-Datacenter Parameters
Datacenter and resource placement
Primary NIC parameters
Control plane machine config pool
Worker machine config pool
Compute sizing
Disk sizing model
Standard data disks
Persistent disk fields
Ephemeral disk fields
Persistent disk lifecycle
Control plane nodes (4 disks per node)
Worker nodes (3 disks per node)
Size parameters
vSphere CPI parameters
Optional Creation-Time Topology Parameters
Multiple datacenters and multiple failure domains
Second NIC parameters
Final Readiness Check
Next Steps

Scenarios

Use this checklist in the following scenarios:

- You are preparing a new VMware vSphere cluster deployment.
- You want to validate external dependencies before you start the deployment.
- You plan to enable optional creation-time topology variants such as multiple datacenters, multiple NICs, or extra worker nodes.

Prerequisites

Before you begin, ensure the following conditions are met:

- You can access the `global` cluster with `kubectl`.
- The workload cluster objects must be stored in the `cpaas-system` namespace.
- You have access to the target vCenter inventory, networks, datastores, and templates.

How to Use This Checklist

Use this checklist in the following order:

1. Collect the deployment parameters listed in this document.
2. Replace every placeholder in the manifest templates with the actual values collected here.
3. Reuse the same value everywhere a placeholder appears in multiple manifests.
4. If an optional feature is not enabled, omit the corresponding YAML block exactly as described in Creation-Time Topology Variants.
5. If an optional field (such as `deviceName`) is not needed, remove the entire line from the YAML manifest.

Parameter Sources and Safe Collection

Separate parameter collection into the following three sources. A value that can be discovered from vCenter is not necessarily a product-supported version, and a product-derived value is not a substitute for a network or capacity decision.

Source	Examples	Collection rule
Automatically discoverable	Datacenters, compute clusters, resource pools, networks, datastores, VM templates, and the vCenter certificate thumbprint	Query vCenter with read-only tools, then have the vSphere administrator confirm the selected objects.
Product-derived	Kubernetes, Alauda OS image, Kube-OVN chart, CoreDNS, etcd, provider chart, and CPI versions	Select the exact set published for the target ACP release. Use the OS Support Matrix for component versions and the OS download page for the OS image, together with provider release notes and delivered package metadata; do not choose the newest Registry tag independently.
Operator-decided	Cluster and resource names, endpoint, CIDRs, DNS, node count, CPU, memory, disk capacity, and credentials	Record an explicit design decision and validation owner. These values cannot be inferred safely from inventory alone.

WARNING

The commands in this section are discovery and existence checks only. Do not generate a final manifest automatically from their output, and do not redirect vCenter passwords, Kubernetes Secret data, or other credentials into the checklist.

Optional vCenter discovery with govc

If `govc` is already authenticated through your protected local environment, use read-only inventory commands such as:

```
govc find / -type d
govc find / -type c
govc find / -type p
govc find / -type n
govc find / -type s
govc find / -type m
```

Interpret the results as candidates, not automatic selections:

- `d` : datacenters
- `c` : compute clusters
- `p` : resource pools
- `n` : networks or distributed port groups
- `s` : datastores
- `m` : VMs and templates; confirm the selected object is a template

Use the `openssl` command in Thumbprint to retrieve the certificate fingerprint without printing vCenter credentials.

Terminology

The following terms are used consistently throughout the VMware vSphere cluster-creation documents.

machine config pool

A machine config pool is the `VSphereMachineConfigPool` custom resource. It predefines node slots. Each slot can include:

- A node hostname
- A target datacenter
- Static IP configuration for each NIC
- Persistent disk definitions

WARNING

Each `VSphereMachineConfigPool` can only be referenced by a single `KubeadmControlPlane` or a single `MachineDeployment`. Do not share one `VSphereMachineConfigPool` across multiple control plane or worker groups. If a pool is already bound to another consumer, the `VSphereMachine` will report a `MachineConfigPoolReady=False` condition with reason `PoolBoundToOtherConsumer`.

Node slot

A node slot is an entry under `VSphereMachineConfigPool.spec.configs[]`. A single slot usually maps to one node, such as `cp-01` or `worker-01`. The slot `hostname` drives the Kubernetes node name, the kubelet serving certificate DNS SAN, and (combined with the resolved primary NIC addresses) the kubelet `node-ip`; it must be a valid DNS-1123 subdomain.

Slot network layout

Each slot declares its NIC layout under `network.primary` and `network.additional`:

- `network.primary` is required. Its `networkName` must be set and is used as the node-ip source for the kubelet.
- `network.additional` is an optional list of extra NICs merged after the primary NIC in the order listed.

The `dns` values in `VSphereMachineConfigPool.spec.configs[].network` describe the static network metadata for each slot. In affected VMware deployments, these values might not update the guest operating system's `/etc/resolv.conf` reliably. The deployment manifests therefore also write `/etc/resolv.conf` through both `KubeadmControlPlane.spec.kubeadmConfigSpec.files` and `KubeadmConfigTemplate.spec.template.spec.files`.

`deviceName`

`deviceName` is an optional field in the `VSphereMachineConfigPool` network configuration. It is used to control the NIC name seen inside the guest operating system, such as `eth0` or `eth1`.

Use the following distinctions when you fill the values:

- `networkName` is the vCenter network or port group name.
- `deviceName` is the NIC name inside the guest operating system.
- If `deviceName` is omitted, CAPV typically assigns names such as `eth0`, `eth1`, and `eth2` by NIC order.

vCenter resource pool

A vCenter resource pool is the native vCenter inventory object, for example:

```
/Datacenter1/host/cluster1/Resources
```

When the failure-domain creation variant is enabled, this path is used by `VSphereDeploymentZone.spec.placementConstraint.resourcePool`.

Compute cluster

The compute cluster is the target vCenter compute-cluster name. In these documents, it is primarily used when a `VSphereFailureDomain` is mapped to a specific deployment target.

Datastore

The datastore is the vSphere storage location that stores VM disks. Both system disks and data disks must be placed on concrete datastores.

VM template

The VM template is the source template used to create node virtual machines. When you enable multiple datacenters, the same template must already exist in every target datacenter and must be resolvable by the same template name.

Thumbprint

The thumbprint is the SHA-1 fingerprint of the vCenter server certificate. CAPV uses it to validate the target vCenter server.

Use the following command to retrieve it:

```
openssl s_client -connect <vsphere_server>:443 -servername <vsphere_serve
r> </dev/null 2>/dev/null | openssl x509 -noout -fingerprint -sha1
```

Management Cluster Prerequisites

Use the following table to record the required values and validation results.

Parameter	Placeholder	Required	Validation or Notes
`global` cluster kubeconfig path	-	Yes	`kubectl get ns` succeeds with this kubeconfig.
Workload object namespace	`<namespace>`	Yes	The namespace that stores workload cluster objects. Must be `cpaas-system`.
`cluster-api-provider-vsphere` is installed	-	Yes	`kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-vsphere` returns a result.
`cluster-api-provider-kubeadm` is installed	-	Yes	`kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-kubeadm` returns a result.
`ClusterResourceSet=true` is enabled	-	Yes	The `capi-controller-manager` arguments include `ClusterResourceSet=true`

vCenter and Template Prerequisites

vCenter connection information

Parameter	Placeholder	Required	Validation or Notes	Example
vCenter server	`<vsphere_server>`	Yes	Use the vCenter IP address or FQDN.	`vc.example.`
vCenter username	`<vsphere_username>`	Yes	Used by CAPV to authenticate to vCenter.	`svc-capv@example`
vCenter password	`<vsphere_password>`	Yes	Used by CAPV to authenticate to vCenter.	`******`
Thumbprint	`<thumbprint>`	Yes	Retrieve it with the `openssl` command shown earlier.	`AA:BB:CC:..`

Note: These documents assume the default vCenter HTTPS port `443`.

VM template requirements

Parameter	Placeholder	Required	Validation or Notes	Ex
VM template name	`<template_name>`	Yes	Used to clone control plane and worker nodes.	`a k8`

Parameter	Placeholder	Required	Validation or Notes	Ex
vCenter credential secret name	`<credentials_secret_name>`	Yes	Use a stable name such as `<cluster_name>-vsphere-credentials`.	`d` `cl` `vs` `cr`
Clone mode	`<clone_mode>`	Yes	Allowed values: `linkedClone`, `fullClone`. Default is `linkedClone`. `linkedClone` requires the VM template to have at least one snapshot; if none exists, CAPV falls back to `fullClone`. When `linkedClone` is used, `diskGiB` is ignored; the system disk stays at the template's size. Choose `fullClone` if you need `diskGiB` to take effect.	`l`
Power-off mode	`<power_off_mode>`	Yes	Allowed values: `hard`, `soft`, `trySoft`. Default is `hard`. `soft` and `trySoft` require VMware Tools / `open-vm-tools` in the template for	`t`

Parameter	Placeholder	Required	Validation or Notes	Ex
			graceful shutdown; `trySoft` falls back to `hard` after the guest-soft-power-off timeout.	
SSH public key	`<ssh_public_key>`	Yes	Injected into control plane and worker nodes.	`s` `AA`

The template should also meet the following requirements:

- It uses an operating system supported by your platform image policy.
- It includes `cloud-init`.
- It includes VMware Tools or `open-vm-tools`.
- It includes `containerd`.
- It includes the baseline components required by kubeadm bootstrap.
- It includes pre-exported container image tar files under `/root/images/`. These files are imported into containerd by `capv-load-local-images.sh` before kubeadm runs, so that node bootstrap does not depend on pulling images from a remote registry.
- The tar files include the kubeadm control-plane images supplied with the OS template, including etcd and kube-apiserver, under the exact image references generated from `clusterConfiguration.imageRepository`, the Kubernetes version, and the component image tags. Although etcd and kube-apiserver run as static Pods, they still require container images. A missing or mismatched local reference causes the runtime to try the configured Registry instead.
- The `/root/images/*.tar` files **must** include the sandbox (pause) image whose reference exactly matches the `sandbox_image` value (containerd v1) or `sandbox` value (containerd v2) configured in `/etc/containerd/config.toml`. For example, if containerd is configured with `sandbox_image = "registry.example.com/tkestack/pause:3.10"`, one of the tar files must contain that exact image reference. A mismatch causes containerd

to pull the sandbox image from the network, which defeats the purpose of local preloading and fails in air-gapped environments.

Load Balancer Prerequisites

This section covers the external LoadBalancer endpoint, which the baseline cluster uses. Provision it before cluster creation and follow Plan the Control Plane Endpoint. A cluster can instead be created with the provider-managed Self-built VIP, which needs no external LoadBalancer and requires ACP `v4.4` or later; see Control Plane Endpoint Modes.

Parameter	Placeholder	Required	Validation or Notes	Exampl
Control plane VIP	`<vip>`	Yes	The frontend address is allocated, stable, and always an IP address — the external load balancer VIP, or an IPv4 address for Self-built VIP. Domain names are not supported for the cluster endpoint.	`10.10.`
API server port	`<api_server_port>`	Yes	The default port is `6443`.	`6443`
Listener mode	-	Yes	Layer 4 TCP forwarding with TLS passthrough.	`TCP`
Backends	-	Yes	Every planned control-plane node IP is registered on backend port `6443`; worker IPs are excluded.	`3 cont` `plane I`
Health check	-	Yes	Prefer HTTPS `/healthz` with HTTP `200`. Record when the device can perform	`HTTPS` `/healt`

Parameter	Placeholder	Required	Validation or Notes	Exampl
			only a TCP-connect check.	
Endpoint connectivity	-	Yes	The `global` cluster and every control-plane node can reach `<vip>:6443`.	`Yes`
Backend maintenance ownership	-	Yes	Identify who updates backends after control-plane scale, replacement, or deletion.	`Platfo team`
Virtual router ID	`<vrid>`	When the cluster uses the Self-built VIP	Keepalived `virtual_router_id`, `1` – `255`, unique in the control-plane Layer-2 domain.	`51`
VIP interface	`<vip_interface>`	When the cluster uses the Self-built VIP	Host interface that carries the VIP, at most 15 characters. Leave empty to let the provider match the interface holding the node's primary IP; set it explicitly on multi-NIC nodes, because it cannot be changed later.	`ens192`

Cluster Baseline Parameters

Parameter	Placeholder	Required	Validation or Notes
Cluster name	`<cluster_name>`	Yes	Use the same value in

Parameter	Placeholder	Required	Validation or Notes
Kubernetes version	`<k8s_version>`	Yes	Use the version require release. The baseline r `imagePullCredentia` for Kubernetes 1.34 or 1.35 or later, add `imagePullCredentia` `NeverVerify` to the ku control-plane and work
Control plane replicas	`<cp_replicas>`	Yes	Must be `3` or more. T webhook rejects a pool `KubeadmControlPlane` and a `replicas: 1` c upgraded later. See Pr
Worker replicas	`<worker_replicas>`	Yes	The baseline topology
Pod CIDR	`<pod_cidr>`	Yes	Must not overlap with e
Service CIDR	`<service_cidr>`	Yes	Must not overlap with e
Image registry	`<image_registry>`	Yes	Registry address used `imageRepository`, w `/v2/`. The value becc `<image_registry>/t`
Cluster Registry address	`<registry_address>`	Yes	Permanent Registry ad workload nodes. Use for `cpaas.io/registr` the retired bootstrap Re append `/v2/`.
`kube-ovn` version	`<kube_ovn_version>`	Yes	Must match the platforr requirements.
`kube-ovn-join-cidr`	`<kube_ovn_join_cidr>`	Yes	Must not overlap with o

Parameter	Placeholder	Required	Validation or Notes
CoreDNS image tag	`<dns_image_tag>`	Yes	Use the tag published f release, copied verbati suffix: at ACP `v4.4` a under `acp/coredns`; under `tkestack/core` reads the suffix to selec tag without it resolves t and fails to pull on ACP `dns.imageRepository` managed fields on you
etcd image tag	`<etcd_image_tag>`	Yes	Use the tag approved f version. Take it from th Matrix row as the other
Encryption provider secret	`<encryption_provider_secret>`	Yes	Base64-encoded AES at rest. Generate with `/dev/urandom | base` example values.

Registry tag validation

First read the permanent Registry address from the `global` cluster, or confirm the equivalent address with the platform administrator:

```
kubectl -n cpaas-system get cluster global \
  -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}{"\n"}'
kubectl -n cpaas-system get secret public-registry-credential -o name
```

The Secret command verifies existence only and does not print credential data. Export the confirmed Cluster Registry address and the image Registry used by the CPI and kubeadm manifests. If both placeholders identify the same Registry, use the same value for both variables.

```
export CLUSTER_REGISTRY_ADDRESS="<registry_address>"
export IMAGE_REGISTRY_ADDRESS="<image_registry>"
# Repository names depend on the target version. The values below are the
# ACP v4.4 and later ones; for an earlier target, change them to:
#   Kube-OVN chart -> acp/chart-cpaas-kube-ovn
#   CoreDNS        -> tkestack/coredns
export KUBE_OVN_CHART_REPO="acp/chart-kube-ovn"
export COREDNS_REPO="acp/coredns"

curl -sk "https://${CLUSTER_REGISTRY_ADDRESS}/v2/${KUBE_OVN_CHART_REPO}/tags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/${COREDNS_REPO}/tags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/acp/cloud-provider-vsphere/tags/list"
```

A `NAME_UNKNOWN` response means the repository name does not match the target version; correct `KUBE_OVN_CHART_REPO` or `COREDNS_REPO` from the mapping above and retry, rather than reading it as a missing tag.

The authoritative chart-name boundary is in Upgrade Kube-OVN Before the Control Plane.

The `/v2/` segment belongs only to these Registry HTTP API URLs. Image references use `<registry>/<repository>:<tag>`, and the Cluster annotation uses `<host>:<port>`.

Confirm that the responses include the values you plan to use for:

- `<kube_ovn_version>`
- `<dns_image_tag>`
- `<cpi_image_tag>`

The Registry does not need to serve the etcd or kube-apiserver images when the VM template contains their exact image references and `capv-load-local-images.sh` imports them before kubeadm runs. Static-Pod deployment alone is not sufficient: verify the local images whenever the OS template, Kubernetes version, `imageRepository`, or component tags change.

Validate the locally preloaded images

Create and boot a disposable VM from the exact candidate template. Run the following commands inside that VM to exercise the same import path used before kubeadm and list the image references that are actually available to containerd:

```
sudo bash -euo pipefail <<'EOF'
shopt -s nullglob
image_files=(/root/images/*.tar)
if (( ${#image_files[@]} == 0 )); then
  echo "ERROR: no tar files found in /root/images" >&2
  exit 1
fi

systemctl restart containerd
for image_file in "${image_files[@]}"; do
  ctr -n k8s.io images import "${image_file}"
done
ctr -n k8s.io images list -q | sort -u
EOF

sudo grep -nE '^[[:space:]]*(sandbox_image|sandbox)[[:space:]]*=' \
  /etc/containerd/config.toml
```

Compare the first command's output with the complete references required by the selected Kubernetes version and manifest values. Confirm that every required kubeadm image is present, including the exact etcd and kube-apiserver references generated from `<image_registry>/tkestack`, `<k8s_version>`, and the component tags. Compare the second command's sandbox reference with the imported pause-image reference; they must be identical.

Perform this validation again whenever the template, Kubernetes version, `imageRepository`, etcd tag, or containerd sandbox setting changes. Discard the validation VM afterward; do not convert it back into the source template.

Set `cpaas.io/registry-address` in the workload `Cluster` annotation to `<registry_address>`.

Minimum Single-Datacenter Parameters

Datacenter and resource placement

Parameter	Placeholder	Required	Validation or Notes
Default datacenter	`<default_datacenter>`	Yes	Used by the baseline topology.
VM folder	`<vm_folder>`	Yes	Folder path for VMs created from `VSphereMachineTemplate`. Us `/<datacenter>/vm/<cluster_n` so operators can identify which c the VMs belong to in vCenter. Fo `global` DR deployments, add child folder under `<cluster_nar` to distinguish the primary and sta clusters.

Primary NIC parameters

Parameter	Placeholder	Required	Validation or Notes	Example
vCenter network name	`<nic1_network_name>`	Yes	The first network or port group name in vCenter.	`pg-busines`
Guest NIC name	`<nic1_device_name>`	No	Set it only when you need to force the guest NIC name. If not needed, remove the `deviceName` line from the YAML manifests.	`eth0`

Parameter	Placeholder	Required	Validation or Notes	Example
Gateway	`<nic1_gateway>`	Yes	Default gateway for the primary NIC.	`10.10.10.1`
Prefix length	`<nic1_prefix>`	Yes	Used with each node IP address.	`24`
DNS server 1	`<nic1_dns_1>`	Yes	DNS server for the primary NIC.	`10.10.0.10`

Control plane machine config pool

Parameter	Placeholder	Required	Validation or Notes	Example
Control plane pool name	`<cluster_name>-cp-pool`	Yes	Machine config pool name for control plane nodes. Keep the same existing pool reference during Machine replacement. A different pool object owns independent slots and VMDKs.	`demo-cluster-pool`
Control plane node 1 hostname	`<cp_node_name_1>`	Yes	Node name and kubelet serving cert SAN for the first control plane node.	`cp-01`

Parameter	Placeholder	Required	Validation or Notes	Example
			Must be a valid DNS-1123 subdomain.	
Control plane node 1 datacenter	`<master_01_datacenter>`	Yes	Usually the same as the default datacenter.	`dc-a`
Control plane node 1 IP address	`<master_01_nic1_ip>`	Yes	IPv4 address only, without the prefix length.	`10.10.1`
Control plane node 2 hostname	`<cp_node_name_2>`	Yes	Node name and kubelet serving cert SAN for the second control plane node. Must be a valid DNS-1123 subdomain.	`cp-02`
Control plane node 2 datacenter	`<master_02_datacenter>`	Yes	Usually the same as the default datacenter.	`dc-a`
Control plane node 2 IP address	`<master_02_nic1_ip>`	Yes	IPv4 address only, without the prefix length.	`10.10.1`
Control plane node 3 hostname	`<cp_node_name_3>`	Yes	Node name and kubelet serving cert SAN for the third control plane node.	`cp-03`

Parameter	Placeholder	Required	Validation or Notes	Example
			Must be a valid DNS-1123 subdomain.	
Control plane node 3 datacenter	`<master_03_datacenter>`	Yes	Usually the same as the default datacenter.	`dc-a`
Control plane node 3 IP address	`<master_03_nic1_ip>`	Yes	IPv4 address only, without the prefix length.	`10.10.1`

Worker machine config pool

Parameter	Placeholder	Required	Validation or Notes	Ex
Worker pool name	`<cluster_name>-worker-pool`	Yes	Machine config pool name for worker nodes. Keep the same existing pool reference during Machine replacement. A different pool object owns independent slots and VMDKs.	`d` `cl` `wo`
Worker node 1 hostname	`<worker_node_name_1>`	Yes	Node name and kubelet serving cert SAN for the first worker	`w`

Parameter	Placeholder	Required	Validation or Notes	Ex
			node. Must be a valid DNS-1123 subdomain.	
Worker node 1 datacenter	`<worker_01_datacenter>`	Yes	Usually the same as the default datacenter.	`d`
Worker node 1 IP address	`<worker_01_nic1_ip>`	Yes	IPv4 address only, without the prefix length.	`1`
Worker node 2 hostname	`<worker_node_name_2>`	No	Used when you scale out the worker pool.	`w`
Worker node 2 datacenter	`<worker_02_datacenter>`	No	Used when you scale out the worker pool.	`d`
Worker node 2 IP address	`<worker_02_nic1_ip>`	No	Used when you scale out the worker pool.	`1`
`releaseDelayHours`	`<release_delay_hours>`	Yes	Delay before the provider reclaims persistent VMDKs from an unused `Released` slot. The default is `24`. A replacement	`2`

Parameter	Placeholder	Required	Validation or Notes	Ex
			Machine that references the same existing pool can reuse a `Released` slot immediately; it does not wait for this delay to expire.	

Compute sizing

Parameter	Placeholder	Required	Validation or Notes	Exar
Control plane CPU	`<cp_num_cpus>`	Yes	CPU count per control plane node.	`4`
Control plane memory MiB	`<cp_memory_mib>`	Yes	Memory per control plane node.	`819`
Control plane system datastore	`<cp_system_datastore>`	Yes	Datastore for the control plane system disk.	`dat` `cp`
Control plane system disk size GiB	`<cp_system_disk_gib>`	Yes	Baseline example value is `300`. Ignored when `<clone_mode>` is `linkedClone`; the system disk then stays at	`300`

Parameter	Placeholder	Required	Validation or Notes	Exar
			the template's size.	
Worker CPU	`<worker_num_cpus>`	Yes	CPU count per worker node.	`2`
Worker memory MiB	`<worker_memory_mib>`	Yes	Memory per worker node.	`409`
Worker system datastore	`<worker_system_datastore>`	Yes	Datastore for the worker system disk.	`dat` `work`
Worker system disk size GiB	`<worker_system_disk_gib>`	Yes	Baseline example value is `300`. Ignored when `<clone_mode>` is `linkedClone`; the system disk then stays at the template's size.	`300`

Disk sizing model

`VSphereMachineTemplate.spec.template.spec.diskGiB` and the two disk lists under `VSphereMachineConfigPool.spec.configs[]` define different disks.

Field	Disk type	Meaning
`VSphereMachineTemplate.spec.template.spec.diskGiB`	System disk	Sets the V system di the VM is `fullClo` must be g equal to t

Field	Disk type	Meaning
		size in th template. sum of al
`VSphereMachineConfigPool.spec.configs[].persistentDisks[]`	Persistent data disks	Extra VM the node reattache replacem as `/var/` Separate system di
`VSphereMachineConfigPool.spec.configs[].ephemeralDisks[]`	Ephemeral data disks	Extra VM empty wit deleted w `/var/li` `/var/li` and `/va` Separate system di

Do not add the data disk sizes to `diskGiB`. For example, if the OS image has a 100 GiB system disk and the node also needs a 100 GiB persistent disk plus three ephemeral disks, use a system disk value that is at least 100 GiB and declare the data disks in `persistentDisks` and `ephemeralDisks`. Setting `diskGiB` to the combined size would create an oversized system disk in addition to the data disks.

Standard data disks

Each node role requires a set of dedicated data disks. The disks listed below are the minimum required disks. You can append additional disks to either list if your workload requires them. Plan the disks, their sizes and mount paths before the first Machine takes the slot: a slot that is `InUse` or `Released` rejects removing a persistent disk or changing its `sizeGiB` or `mountPath`.

The split follows the same model as the Huawei Cloud Stack and Huawei DCS providers: only platform state that must survive VM replacement (`/var/cpaas`) is a persistent disk, while the

rebuildable runtime directories (`/var/lib/kubelet`, `/var/lib/containerd`, `/var/lib/etcd`) are recreated with every VM.

> **WARNING**
>
> **The split applies to new clusters only**
>
> `ephemeralDisks[]` is new in provider `v1.0.17`, so this layout is a creation-time choice. A cluster created on an earlier provider declares `/var/cpaas`, `/var/lib/containerd`, and `/var/lib/etcd` as persistent disks, with `wipeFilesystem: true` on the etcd disk, and it keeps that layout. A provider upgrade migrates no disk. See Upgrading the Provider.

Persistent disk fields

Field	Required	Default	Description
`name`	Yes	—	Unique disk name within the slot.
`sizeGiB`	Yes	—	Disk size in GiB.
`mountPath`	No	(empty)	Mount path inside the guest OS. If empty, the disk is attached as a raw device with a symlink at `/dev/disk/by-capv/<name>` but is not formatted or mounted. This is useful when an external process manages the disk at runtime.
`mountOptions`	No	(empty)	Mount options applied when `mountPath` is set.
`fsFormat`	No	`ext4` (when `mountPath` is set)	Filesystem format. Ignored when `mountPath` is empty.
`wipeFilesystem`	No	`false`	When `true`, disk content is wiped on the first boot of a new VM. Reboots and manual service restarts are not affected. The

Field	Required	Default	Description
			standard disk set does not use it: the only persistent disk, `/var/cpaas`, is meant to keep its contents across replacement, and every disk that must start empty is declared in `ephemeralDisks[]`.
`unitNumber`	No	(controller-assigned)	SCSI unit that keeps guest device ordering stable across VM recreations. Between `0` and `15`, never `7` (reserved for the SCSI controller), and unique within the slot.
`datastore`	No	(pool default)	Override the datastore for this disk.
`storagePolicy`	No	(pool default)	Override the vSphere storage policy for this disk.

Ephemeral disk fields

`ephemeralDisks[]` mirrors the guest-facing fields of a persistent disk — `name`, `sizeGiB`, `mountPath`, `mountOptions`, `fsFormat`, `datastore`, and `storagePolicy` — and drops the persistence fields. There is no `wipeFilesystem`, because the backing VMDK is always newly created and formatted from scratch, and no `unitNumber`, because the controller assigns the SCSI unit at clone time and reports it in `status.ephemeralDiskStatuses[]`.

Disk names and mount paths share one namespace across both lists within a slot. For the field-by-field comparison and the guidance on which list to use, see Choose persistent or ephemeral.

Persistent disk lifecycle

Persistent disks are owned by a slot in one specific `VSphereMachineConfigPool`, not by the short-lived VM. A slot in another pool is a different allocation even when it declares the same hostname, IP address, or disk names. The expected replacement lifecycle is:

1. While a Machine uses the slot, `status.configStatuses[].state` is `InUse` and the VMDKs are attached to its VM.
2. When the Machine and VM are deleted, the slot becomes `Released`. The VMDKs remain available for replacement and are not permanent orphaned disks.
3. A replacement Machine that references the same existing pool can reuse the `Released` slot immediately. The provider selects a released slot before an unused slot and reattaches the declared persistent VMDKs to the replacement VM. `releaseDelayHours` is not a wait time for reuse.
4. If the slot remains unused until `releaseDelayHours` expires, the controller can make it generally available and reclaim persistent VMDKs only after it confirms they are no longer attached. Asynchronous progress appears in `status.persistentDiskStatuses[].phase`.

`wipeFilesystem` controls data handling after a persistent disk is attached to a new VM; it does not control slot selection or datastore reclamation. `false` preserves the existing filesystem contents. `true` deletes the mounted contents during the new VM's first boot; it does not make VMDK reclamation faster and must not be treated as a datastore-capacity setting.

Redeployment and teardown capacity

Creating a new `VSphereMachineConfigPool` with a different `metadata.name` creates a separate set of slots. Those slots do not reuse VMDKs owned by an earlier pool. Repeated teardown and redeployment with different pool names can therefore temporarily consume multiple complete sets of persistent disks until every earlier pool finishes provider-managed reclaim.

Use these rules when retrying or tearing down a deployment:

- For Machine replacement or rolling updates within the same cluster, keep each `VSphereMachineTemplate.spec.template.spec.machineConfigPoolRef.name` pointed at the same existing pool. Changing `KubeadmControlPlane.spec.machineNamingStrategy` does not transfer disks between pools and is not a datastore cleanup control.
- If a new deployment intentionally uses a new pool name, include the new pool's complete persistent-disk allocation in datastore capacity planning. Do not assume matching hostnames or disk names enable reuse across pools.

- Before starting another deployment against a capacity-constrained datastore, verify that earlier pool objects have completed deletion and that their VMDKs have been reclaimed. Pool deletion is asynchronous because the finalizer waits for Machines to be removed and for persistent-disk reclaim to finish.
- Do not remove the pool finalizer or manually delete backing VMDKs to accelerate teardown. If reclaim does not progress, collect the pool status, Events, provider-controller logs, and vCenter attachment state before treating the condition as a provider defect.
- Size the datastore for both lists. Ephemeral VMDKs occupy datastore space for as long as their VM exists, but they are deleted with it, so they never accumulate across pools and never hold a slot release. Only persistent VMDKs survive a teardown and wait for provider-managed reclaim.

Use the following fields when you investigate a released slot or a datastore capacity alert:

Status field	Meaning
`state`	Slot allocation state: `Available`, `InUse`, or `Released`.
`lastReleasedTime`	When the slot entered `Released`. Compare this with `releaseDelayHours`.
`reclaimStatus.*`	Deprecated. The provider folds it into `status.persistentDiskStatuses[]` on upgrade and then clears it.

Read per-disk state from `status.persistentDiskStatuses[]`. Its `phase` field carries `Creating`, `Attached`, `Available`, `Reclaiming`, `Reclaimed`, or `Error`, and each entry also reports `lastError`, `retryAfter`, `taskRef`, and the owning machine. The `status.configStatuses[].reclaimStatus` subtree and the `spec.configs[].persistentDisks[].volumePath` and `.diskUUID` fields are deprecated, frozen, and scheduled for removal. Existing pools are migrated onto `persistentDiskStatuses[]` by the provider itself; see Upgrading the Provider.

Inspect the pool and related Events without reading credentials:

```
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> -o yaml
kubectl -n <namespace> describe vspheremachineconfigpool <pool_name>
kubectl -n <namespace> get events \
  --field-selector involvedObject.kind=VSphereMachineConfigPool,involvedO
bject.name=<pool_name>
```

WARNING

Do not manually delete a VMDK while its slot is `Released`, or while its `persistentDiskStatuses[].phase` is `Reclaiming` or `Error`. A replacement Machine may still reuse it, or the provider may be retrying a verified reclaim operation. Confirm the pool state, retention window, attachment state, and controller status before performing any manual datastore cleanup.

Control plane nodes (4 disks per node)

Disk Name	Mount Path	Disk List	Purpose	Recor Minim
`var-cpaas`	`/var/cpaas`	`persistentDisks[]`	Stores logs, audit data, and other platform data	100 G
`var-lib-kubelet`	`/var/lib/kubelet`	`ephemeralDisks[]`	Stores kubelet working data, including pod volumes and plugin sockets	100 G
`var-lib-containerd`	`/var/lib/containerd`	`ephemeralDisks[]`	Stores containerd runtime data (images, layers, snapshots)	100 G

Disk Name	Mount Path	Disk List	Purpose	Recor Minim
`var-lib-etcd`	`/var/lib/etcd`	`ephemeralDisks[]`	Stores etcd data. The disk is created empty for every new VM, so a replacement control-plane node never carries stale member data into `kubeadm join`.	20 GiB

Worker nodes (3 disks per node)

Disk Name	Mount Path	Disk List	Purpose	Recor Minim
`var-cpaas`	`/var/cpaas`	`persistentDisks[]`	Stores logs, audit data, and other platform data	100 G
`var-lib-kubelet`	`/var/lib/kubelet`	`ephemeralDisks[]`	Stores kubelet working data, including pod volumes and plugin sockets	100 G
`var-lib-containerd`	`/var/lib/containerd`	`ephemeralDisks[]`	Stores containerd runtime data (images, layers, snapshots)	100 G

Size parameters

Parameter	Placeholder	Required	Vali Not
CP `/var/cpaas` size GiB	`<cp_var_cpaas_size_gib>`	Yes	Rec mini GiB.
CP `/var/lib/kubelet` size GiB	`<cp_var_lib_kubelet_size_gib>`	Yes	Rec mini GiB.
CP `/var/lib/containerd` size GiB	`<cp_var_lib_containerd_size_gib>`	Yes	Rec mini GiB.
CP `/var/lib/etcd` size GiB	`<cp_var_lib_etcd_size_gib>`	Yes	Rec mini GiB.
Worker `/var/cpaas` size GiB	`<worker_var_cpaas_size_gib>`	Yes	Rec mini GiB.
Worker `/var/lib/kubelet` size GiB	`<worker_var_lib_kubelet_size_gib>`	Yes	Rec mini GiB.
Worker `/var/lib/containerd` size GiB	`<worker_var_lib_containerd_size_gib>`	Yes	Rec mini GiB.

vSphere CPI parameters

Parameter	Placeholder	Required	Validation or Notes
CPI datacenter list	`<cpi_datacenters>`	Yes	Include every target datacenter when multiple datacenters are enabled.

Parameter	Placeholder	Required	Validation or Notes
CPI image tag	`<cpi_image_tag>`	Yes	Tag for the vSphere CPI component. The full reference is `<image_registry>/acp/cloud-provider-vsphere:<cpi_image_tag>`.

Optional Creation-Time Topology Parameters

Multiple datacenters and multiple failure domains

Parameter	Placeholder	Required	Validation or Notes
Datacenter 1 failure domain name	`<fd_name_1>`	No	Required only when you enable failure domains.
Datacenter 1 deployment zone name	`<dz_name_1>`	No	Required only when you enable failure domains.
Compute cluster name	`<compute_cluster_1>`	No	Required only when you enable failure domains.
Default datastore	`<default_datastore_1>`	No	Required only when you enable failure domains.
vCenter resource pool path	`<resource_pool_path_1>`	No	Required only when you enable failure domains. Must exist in vCenter inventory.
Datacenter 2 name	`<dc_name_2>`	No	Used only for multiple datacenters or failure domains.

Parameter	Placeholder	Required	Validation or Notes
Datacenter 2 failure domain name	`<fd_name_2>`	No	Used only for multiple datacenters or failure domains.
Datacenter 2 deployment zone name	`<dz_name_2>`	No	Used only for multiple datacenters or failure domains.
Datacenter 2 compute cluster	`<compute_cluster_2>`	No	Used only for multiple datacenters or failure domains.
Datacenter 2 default datastore	`<default_datastore_2>`	No	Used only for multiple datacenters or failure domains.
Datacenter 2 vCenter resource pool path	`<resource_pool_path_2>`	No	Used only for multiple datacenters or failure domains.
Worker deployment zone	`<worker_failure_domain>`	No	Use a `VSphereDeploymentZone` name, not a `VSphereFailureDomain` name.
Datacenter 3 name	`<dc_name_3>`	No	Used only for additional datacenter or failure-domain scenarios.
Datacenter 3 failure domain name	`<fd_name_3>`	No	Used only for additional datacenter or failure-domain scenarios.
Datacenter 3 deployment zone name	`<dz_name_3>`	No	Used only for additional datacenter or failure-domain scenarios.

Parameter	Placeholder	Required	Validation or Notes
Datacenter 3 compute cluster	`<compute_cluster_3>`	No	Used only for additional datacenter or failure-domain scenarios.
Datacenter 3 default datastore	`<default_datastore_3>`	No	Used only for additional datacenter or failure-domain scenarios.
Datacenter 3 vCenter resource pool path	`<resource_pool_path_3>`	No	Used only for additional datacenter or failure-domain scenarios.

Second NIC parameters

Parameter	Placeholder	Required	Validation or Notes	Example
Secondary network name	`<nic2_network_name>`	No	Used only when nodes require a second NIC.	`pg-management`
Secondary guest NIC name	`<nic2_device_name>`	No	Set it only when you need to force the guest NIC name.	`eth1`
Secondary gateway	`<nic2_gateway>`	No	Used only when nodes require a second NIC.	`10.20.10.1`
Secondary prefix length	`<nic2_prefix>`	No	Used with each secondary NIC IP address.	`24`

Parameter	Placeholder	Required	Validation or Notes	Example
Secondary DNS server 1	`<nic2_dns_1>`	No	Used only when nodes require a second NIC.	`10.20.0.10`
Control plane node 1 secondary IP	`<master_01_nic2_ip>`	No	Used only when nodes require a second NIC.	`10.20.10.1`
Control plane node 2 secondary IP	`<master_02_nic2_ip>`	No	Used only when nodes require a second NIC.	`10.20.10.1`
Control plane node 3 secondary IP	`<master_03_nic2_ip>`	No	Used only when nodes require a second NIC.	`10.20.10.1`
Worker node 1 secondary IP	`<worker_01_nic2_ip>`	No	Used only when nodes require a second NIC.	`10.20.10.2`
Worker node 2 secondary IP	`<worker_02_nic2_ip>`	No	Used when workers are expanded and require a second NIC.	`10.20.10.2`

Final Readiness Check

Before you start the deployment, confirm all of the following items:

1. The `global` cluster is reachable.

2. The two cluster plugins are installed: Alauda Container Platform Cluster Bootstrap Provider and Alauda Container Platform Infrastructure Provider for Virtualization Platform.
3. `ClusterResourceSet=true` is enabled.
4. The vCenter server, username, password, and thumbprint are collected.
5. The external LoadBalancer satisfies the Layer 4 listener, backend, health-check, ownership, and reachability contract. If the cluster uses the Self-built VIP instead, the Self-built VIP prerequisites are met and `<vrid>` and `<vip_interface>` are decided.
6. The Pod CIDR, Service CIDR, and `kube-ovn-join-cidr` do not overlap with existing networks.
7. The VM template is available in every required datacenter.
8. The required datastores and vCenter resource pool paths are confirmed.
9. The permanent platform Registry contains the required Kube-OVN, CoreDNS, and vSphere CPI tags.
10. The VM template contains the exact locally preloaded etcd and kube-apiserver image references required by the selected Kubernetes and component versions.
11. The `cpaas.io/registry-address` value is planned as `<registry_address>` without a scheme or `/v2/` path.
12. The control plane and worker bootstrap manifests both write `/etc/resolv.conf` through kubeadm `files`.
13. The machine config pool values for the minimum single-datacenter topology are complete.
14. The baseline system disk and data disk sizing is confirmed, each required disk is declared in the correct list (`/var/cpaas` persistent; `/var/lib/kubelet`, `/var/lib/containerd`, and `/var/lib/etcd` ephemeral), and data disk sizes are not added into `diskGiB`.
15. `releaseDelayHours`, same-pool persistent-disk reuse, new-pool capacity allocation, and provider-managed teardown reclaim are understood before datastore capacity is allocated.
16. Every required parameter has a real value.

Next Steps

After you complete this checklist, continue with Create a Cluster on VMware vSphere.

Create a Cluster on VMware vSphere

This document explains how to create a VMware vSphere workload cluster by applying Cluster API manifests to the ACP `global` management cluster. The procedure does not create the `global` cluster itself. It covers a minimum supported topology with one datacenter, one NIC per node, and static IP allocation through `VSphereMachineConfigPool`.

TOC

- Provider Requirements
- Scenarios
- How to Use This Page
- Prerequisites
- Control Plane Endpoint Modes
- Key Objects
  - `ClusterResourceSet`
  - vSphere CPI component
  - Provider-managed fields on your manifests
  - machine config pool
  - VM template requirements
- Local File Layout
- Steps
  - Validate the environment
  - Create the namespace and vCenter credential secret
  - Create the `Cluster` and `VSphereCluster` objects

Create the vSphere CPI delivery resources
Create the machine config pools
Apply the optional failure-domain objects
Create the control plane objects
Create the worker objects
Wait for the cluster to become ready
Verification
Troubleshooting
Admission rejections
Creation-Time Topology Variants
Use a Self-built VIP
Self-built VIP prerequisites
`controlPlaneLoadBalancer` fields
Enable the Self-built VIP in `10-cluster.yaml`
Verify the Self-built VIP
Add a second NIC
Multiple datacenters and failure domains
Add data disks
Choose persistent or ephemeral
Pin a SCSI unit number
Relocating the pod log directory
Verify creation-time variants
Next Steps

Provider Requirements

Read the installed provider version from the provider `AppRelease` on the management cluster; do not infer it from the ACP release. See Identify the Installed Provider Version.

The following rules apply when the `VSphereMachineTemplate` referenced by a controller sets `machineConfigPoolRef` — the fixed-IP topology this page teaches. The provider's

admission webhooks enforce the `KubeadmControlPlane` and pool rules on `CREATE` and `UPDATE`, and the last rule on `DELETE`.

- `KubeadmControlPlane.spec.rolloutStrategy.rollingUpdate.maxSurge` is the integer `0`, and `spec.replicas` is `3` or more. An unset `maxSurge` means the Cluster API default `1`, which is not accepted.
- `MachineDeployment.spec.strategy.rollingUpdate.maxSurge` is `0`. With the default `maxSurge` of `1`, a rollout needs an extra slot that a full pool does not have, and it stalls. `strategy.type: OnDelete` needs no surge setting.
- `VSphereMachineConfigPool.spec.configs` declares at least one slot, and every slot declares `network`. Hostnames and primary IP addresses are unique across every pool bound to the same `Cluster`, and one pool serves one controller.
- A slot that is already `InUse` or `Released` rejects some changes. See Allocated slot restrictions.
- A pool that still owns `InUse` slots cannot be deleted.

Confirm the webhooks are registered in your environment with the webhook check in Verify the installation. Where a webhook is not registered, the same values are still required: the fixed-IP slot model has no free slot for a surge machine, so a non-compliant rollout stalls instead of being rejected up front.

Scenarios

Use this document in the following scenarios:

- You want to create the first baseline VMware vSphere workload cluster in your environment.
- You use one datacenter and one NIC per node for the initial validation.
- You want to keep the first deployment simple before enabling advanced placement or networking features.

This document applies to the following deployment model:

- CAPV connects directly to vCenter.

- Control plane and worker nodes both use `VSphereMachineConfigPool` for static IP allocation and data disks.
- `ClusterResourceSet` delivers the vSphere CPI component automatically.
- The first validation uses one datacenter and one NIC per node.

This document does not apply to the following scenarios:

- A deployment that depends on vSphere Supervisor or `vm-operator`.
- A deployment that does not use `VSphereMachineConfigPool`.

This document is written for the current platform environment. The `kube-ovn` delivery path depends on platform controllers that consume annotations on the `Cluster` resource, so this workflow is not intended to be a generic standalone CAPV deployment guide outside the platform context.

How to Use This Page

1. Complete the infrastructure and parameter checklist.
2. Prepare the baseline manifest files in the procedure below.
3. Before applying them, add only the required creation-time topology variants.
4. Apply the complete manifest set and finish the verification checks.
5. After the cluster is running, use Manage Nodes on VMware vSphere for scale-out, machine template changes, and runtime topology changes.

The baseline is the validation reference. If the target cluster needs several optional topology features, introduce and validate one manifest change at a time before combining them.

Prerequisites

Before you begin, ensure the following conditions are met:

1. The Cluster Bootstrap Provider and the Infrastructure Provider for Virtualization Platform are uploaded and installed on the `global` cluster. This applies regardless of the IaaS platform that the `global` cluster itself runs on. See Install the Provider.

2. You completed VMware vSphere Infrastructure Preparation.
3. The `global` cluster can reach vCenter.
4. The target template, networks, datastores, and vCenter resource pool are available.
5. The control plane endpoint mode is selected. For an external LoadBalancer, it satisfies the endpoint contract. For a Self-built VIP, the Self-built VIP prerequisites are met. The mode cannot be changed once the cluster exists, so decide it now: see Control Plane Endpoint Modes.
6. All required static IP addresses are allocated, are not in use, and are not already declared by a slot in another `VSphereMachineConfigPool` bound to the same `Cluster`.
7. `ClusterResourceSet=true` is enabled.
8. The image registry for this cluster is decided. By default the cluster inherits the registry of the `global` cluster, which requires a valid `public-registry-credential` Secret when that registry is authenticated. To pull platform images from a dedicated registry instead, follow Choose the Image Registry for a Workload Cluster before you create the cluster; the binding cannot be changed afterwards.
9. The registry chosen above is reachable from workload nodes and contains the required Kube-OVN, CoreDNS, and vSphere CPI tags.
10. The platform can process the cluster annotations required to install the network plugin.
11. The vCenter account referenced by `VSphereCluster.spec.identityRef` satisfies the Custom Fields privilege requirement in Install the VMware vSphere Provider.
12. The `global` cluster is the active management cluster, not a disaster-recovery standby. See Standby Management Clusters.
13. Role labels and taints for any dedicated node pools are planned. They are declared on the pool's resources, as described in Infra Nodes.

Control Plane Endpoint Modes

Every cluster reaches its API server through a stable endpoint. Select the mode before you write `10-cluster.yaml`, because it cannot be corrected afterwards. The choice is recorded in `VSphereCluster.spec.controlPlaneLoadBalancer`, and leaving the field out is itself a choice: an absent field means an external LoadBalancer.

> **The Self-built VIP is a creation-time choice only**
>
> The Self-built VIP exists from provider `v1.0.17`, requires ACP `v4.4` or later, and is available only to clusters **created** with it. An existing cluster cannot adopt it, and neither can a cluster whose provider was upgraded to `v1.0.17` from an earlier release: the VIP is injected while the first control-plane node bootstraps, and Alive then takes it over, so there is no in-place migration path. Those clusters keep their external LoadBalancer, and moving one onto a Self-built VIP means creating a new cluster. See Upgrading the Provider.

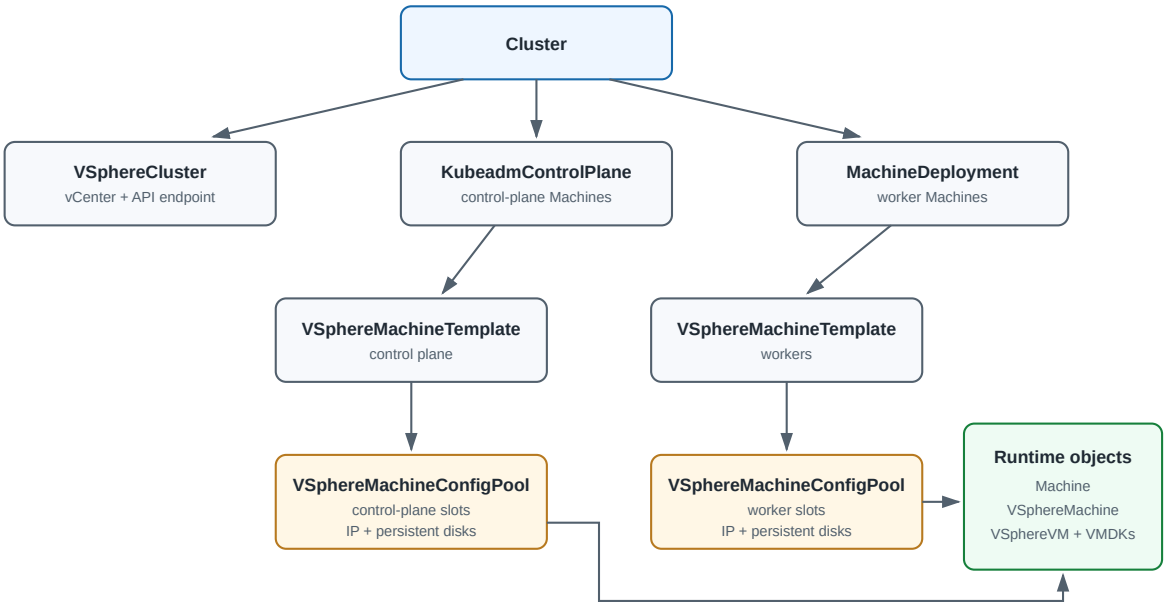
`spec.controlPlaneLoadBalancer`	**Who owns the endpoint**
Absent (the baseline on this page)	You do. Provision an external Layer 4 TCP LoadBalancer and write `spec.controlPlaneEndpoint` yourself.
`type: external`	You do. The provider checks the value against `spec.controlPlaneEndpoint` and deploys no VIP component.
`type: internal`	The provider does. It injects a bootstrap VIP on the first control-plane node, then installs Alive (keepalived and IPVS) into the workload cluster and holds the VIP on the control-plane nodes.

None of the three can be changed once the cluster exists. The field is write-once; see `controlPlaneLoadBalancer` fields.

Recommendation: for an external LoadBalancer, leave the field out rather than writing `type: external`. An absent field already means an external LoadBalancer, and writing it only to record that intent pins the endpoint to a frozen second copy of the same address.

For `type: internal`, the prerequisites, the field reference, the manifest block, and the verification commands are in Use a Self-built VIP.

Key Objects



The top-level `Cluster` references `VSphereCluster` and the control-plane or worker controllers. Those controllers reference `VSphereMachineTemplate` objects. Each template references one `VSphereMachineConfigPool`, whose slots provide hostnames, static network configuration, and persistent disks to the runtime `Machine`, `VSphereMachine`, and `VSphereVM` objects.

Stage	Input	Apply	Expected result	Diagnc
Endpoint and Registry	LoadBalancer, permanent Registry, version set	Validate external dependencies.	Endpoint and Registry are reachable from the required networks.	LoadBa Registr rules.
Cluster root	Network CIDRs, vCenter identity, endpoint	Apply `Cluster` and `VSphereCluster`.	Both resources exist and report progressing conditions.	`Clust` `VSphe`

Stage	Input	Apply	Expected result	Diagnc
CPI delivery	vCenter CPI config and credentials	Apply the `ClusterResourceSet` resources.	The workload cluster receives the vSphere CPI after its API is reachable.	`Clust` and CP
Node slots	Hostnames, static IPs, datastores, persistent disks	Apply both `VSphereMachineConfigPool` objects.	Each pool is Ready and exposes the expected slots.	Pool cc `statu`
Control plane	VM template, kubeadm config, Kubernetes versions	Apply `VSphereMachineTemplate` and `KubeadmControlPlane`.	Three control-plane Machines become Ready.	KCP, M `VSphe` `VSphe` health.
Workers	Worker template and bootstrap config	Apply `KubeadmConfigTemplate` and `MachineDeployment`.	Worker Machines and Nodes become Ready.	Machin VM sta

`ClusterResourceSet`

`ClusterResourceSet` is a Cluster API resource in the `global` cluster. After the workload API server becomes reachable, it applies the referenced `ConfigMap` and `Secret` resources to the workload cluster.

In this workflow, `ClusterResourceSet` is used to deliver the vSphere CPI resources automatically.

vSphere CPI component

The vSphere CPI component is delivered to the workload cluster through `ClusterResourceSet`. It connects workload nodes to the vSphere infrastructure so the cluster can report infrastructure identities and complete cloud-provider initialization.

Provider-managed fields on your manifests

Some fields on the objects you apply are owned by the provider after creation. Knowing which ones prevents a repository fight during upgrades.

- The provider patches your `KubeadmControlPlane` after you apply it. Expect the CoreDNS and kube-proxy skip annotations set to `"true"`, and `spec.kubeadmConfigSpec.clusterConfiguration.dns.imageRepository` set by the controller.
- **Set `dns.imageTag` yourself and leave `dns.imageRepository` unset.** Provide the CoreDNS tag published for the target ACP release: at ACP `v4.4` and later, the tag staged under `acp/coredns`; below `v4.4`, the one under `tkestack/coredns`. The provider reads the `-v<acp>` suffix in the tag to select that repository, so a tag without the suffix resolves to `tkestack/coredns` — on ACP `v4.4` or later that repository does not carry the tag, and the CoreDNS pods stay in `ImagePullBackOff`.
- The kube-proxy repository is derived from the Kubernetes version: `acp/k8s` at `v1.35` and later, `tkestack` below it. The Kube-OVN chart is `acp/chart-kube-ovn` at chart `v4.4` and later, `acp/chart-cpaas-kube-ovn` below it.
- The registry prefix for all three comes from the `Cluster` annotation `cpaas.io/registry-address`.
- The workload cluster's `kube-system/kube-proxy` DaemonSet and `kube-system/coredns` Deployment image repositories are reconciled as well, but **only after the control-plane rollout completes**. A lag between the `KubeadmControlPlane` reaching the target version and those workloads changing repository is expected.
- Image pull credentials for these managed components come from the workload cluster's `cpaas-system/sentry` ServiceAccount.

> **WARNING**
>
> Do not remove the CoreDNS and kube-proxy skip annotations, and do not re-apply a stored `KubeadmControlPlane` manifest that omits them. Returning CoreDNS or kube-proxy to

kubeadm's management while the provider also manages them produces a repository conflict that flips the images back and forth.

machine config pool

The machine config pool is the `VSphereMachineConfigPool` custom resource. In the baseline workflow:

- One machine config pool is used for control plane nodes.
- One machine config pool is used for worker nodes.

Each node slot includes the hostname, datacenter, static IP assignment, and optional data disk definitions.

For network configuration, distinguish the following fields:

- `networkName` is the vCenter network or port group name.
- `deviceName` is the NIC name inside the guest operating system.

If `deviceName` is set, CAPV writes that value into the generated guest-network metadata. If it is omitted, the current implementation typically uses NIC names such as `eth0`, `eth1`, and `eth2` by NIC order.

Also distinguish the following value formats:

- A node IP address is used together with a prefix length, for example `10.10.10.11/24`.
- The gateway field contains only the gateway IP address, for example `10.10.10.1`.

`spec.configs` must declare at least one slot, and every slot must declare `network` with a non-empty `network.primary.networkName`.

Each slot can declare two disk lists. `persistentDisks[]` survive VM replacement and are reattached to the machine that replaces the previous one; `ephemeralDisks[]` are deleted with the VM and recreated empty. The standard set declares `/var/cpaas` as a persistent disk and `/var/lib/kubelet`, `/var/lib/containerd`, and `/var/lib/etcd` as ephemeral disks. Disk names and mount paths share one namespace across both lists within a slot.

`ephemeralDisks[]` is new in provider `v1.0.17`, so this split is available only to a cluster created on `v1.0.17` or later. See Choose persistent or ephemeral.

One pool serves exactly one controller. When a machine allocates a slot, the provider records the owning `KubeadmControlPlane` or `MachineDeployment` in `status.consumerRef`. A second controller whose template references the same pool is not accepted. Use one pool for the control plane and one pool per `MachineDeployment`.

VM template requirements

The VM template used by this workflow should meet the following minimum requirements:

1. It uses the required operating system for the target platform environment.
2. It includes `cloud-init`.
3. It includes VMware Tools or `open-vm-tools`.
4. It includes `containerd`.
5. It includes the baseline components required by kubeadm bootstrap.
6. It includes pre-exported container image tar files under `/root/images/`. These files are imported into containerd by `capv-load-local-images.sh` before kubeadm runs, so that node bootstrap does not depend on pulling images from a remote registry.
7. The tar files include the kubeadm control-plane images supplied with the OS template, including etcd and kube-apiserver, under the exact image references generated from `clusterConfiguration.imageRepository`, the Kubernetes version, and the component image tags. Although etcd and kube-apiserver run as static Pods, they still require container images. A missing or mismatched local reference causes the runtime to try the configured Registry instead.
8. The `/root/images/*.tar` files **must** include the sandbox (pause) image whose reference exactly matches the `sandbox_image` value (containerd v1) or `sandbox` value (containerd v2) configured in `/etc/containerd/config.toml`. For example, if containerd is configured with `sandbox_image = "registry.example.com/tkestack/pause:3.10"`, one of the tar files must contain that exact image reference. A mismatch causes containerd to pull the sandbox image from the network, which defeats the purpose of local preloading and fails in air-gapped environments.

Static IP configuration, hostname injection, and other initialization settings depend on `cloud-init`. Node IP reporting depends on guest tools.

Local File Layout

> **Workload Cluster Naming**
>
> The workload `cluster_name` **must not** be `global`. That name is reserved for the `global` cluster, and reusing it causes the workload cluster's resources to collide with `global` cluster resources in `cpaas-system`. The `global-` prefix is reserved for resources owned by the `global` cluster's DR workflow; see Common Prerequisites. Do not use `global-` for workload-cluster resources, because failover operations can select those resources as if they belonged to the `global` cluster.
>
> As a convention, keep the CAPI `Cluster` and provider cluster resource (`VSphereCluster`) named exactly `<cluster_name>`, and prefix non-root CAPI and provider resources (`KubeadmControlPlane` , `KubeadmConfigTemplate` , `MachineDeployment` , `VSphereMachineTemplate` , `VSphereMachineConfigPool` , etc.) with `<cluster_name>-` — for example, the example manifests use `<cluster_name>-kcp` and `<cluster_name>-md-0`. This is a recommendation rather than a controller-enforced rule, but it prevents same-namespace collisions when multiple workload clusters live in `cpaas-system` and makes resource ownership obvious during operations.

Create a local working directory and store the manifests with the following layout:

```
capv-cluster/
├── 00-namespace.yaml
├── 10-cluster.yaml
├── 15-vsphere-cpi-clusterresourceset.yaml
├── 16-vspheremachineconfigpool-control-plane.yaml
├── 17-vspheremachineconfigpool-worker.yaml
├── 18-failure-domains.yaml  # only when failure domains are enabled
├── 20-control-plane.yaml
└── 30-workers-md-0.yaml
```

Use the following commands to create the directory:

```
mkdir -p ./capv-cluster
cd ./capv-cluster
```

Steps

1 Validate the environment

Run the following commands from the `global` cluster to verify the minimum prerequisites:

```
kubectl get ns
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-vspher
e
kubectl get minfo -l cpaas.io/module-name=cluster-api-provider-kubead
m
kubectl -n cpaas-system get deploy capi-controller-manager -o jsonpat
h='{.spec.template.spec.containers[0].args}'
kubectl -n cpaas-system get secret public-registry-credential -o name
kubectl -n cpaas-system get cluster global \
  -o jsonpath='{.metadata.annotations.cpaas\.io/registry-address}
{"\n"}'
kubectl -n kube-public get configmap etcd-sync -o name 2>/dev/null ||
echo "not standby"
```

Export the permanent platform Registry address returned by the `Cluster` annotation and the image Registry used by the CPI and kubeadm manifests. If both placeholders identify the same Registry, use the same value for both variables.

```
export CLUSTER_REGISTRY_ADDRESS="<registry_address>"
export IMAGE_REGISTRY_ADDRESS="<image_registry>"
# Repository names depend on the target version. The values below are
the
# ACP v4.4 and later ones; for an earlier target, change them to:
#   Kube-OVN chart -> acp/chart-cpaas-kube-ovn
#   CoreDNS        -> tkestack/coredns
export KUBE_OVN_CHART_REPO="acp/chart-kube-ovn"
export COREDNS_REPO="acp/coredns"

curl -sk "https://${CLUSTER_REGISTRY_ADDRESS}/v2/${KUBE_OVN_CHART_REP
O}/tags/list"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/${COREDNS_REPO}/tags/l
ist"
curl -sk "https://${IMAGE_REGISTRY_ADDRESS}/v2/acp/cloud-provider-vsp
here/tags/list"
```

A `NAME_UNKNOWN` response means the repository name does not match the target version; correct `KUBE_OVN_CHART_REPO` or `COREDNS_REPO` from the mapping above and retry, rather than reading it as a missing tag.

The repository boundaries above are the same ones the provider applies at runtime; see Provider-managed fields on your manifests for the selection rules, and Upgrade Kube-OVN Before the Control Plane for the authoritative chart-name table.

The `/v2/` segment is part of the Registry HTTP API URL only. In the workload `Cluster` manifest, set the Registry annotation to `<registry_address>` without a scheme or path.

Confirm the following results:

- The `global` cluster is reachable.
- Alauda Container Platform Cluster Bootstrap Provider and Alauda Container Platform Infrastructure Provider for Virtualization Platform are running.
- The controller arguments include `ClusterResourceSet=true`.
- The `public-registry-credential` Secret exists; its contents are not printed by this procedure.

- The `global` cluster is not a disaster-recovery standby. The `kube-public/etcd-sync` ConfigMap must not exist. See Standby Management Clusters.
- The permanent platform Registry contains the required Kube-OVN, CoreDNS, and vSphere CPI tags.
- The VM template passed the executable local-image validation, including exact etcd, kube-apiserver, and sandbox image references. Their static-Pod deployment does not remove the image requirement.

Before you continue, also verify the following items:

- The vCenter server address is reachable.
- The vCenter username and password are valid.
- The thumbprint is correct.
- The template name is correct.
- The template is resolvable in the target datacenter.
- If the VM is cloned as a `fullClone`, the template system disk is not larger than the `diskGiB` value used later in the manifests. If CAPV completes a `linkedClone`, the system disk size stays at the template's size and `diskGiB` is ignored.
- VMware Tools or `open-vm-tools` is installed in the template.
- The external LoadBalancer follows the Layer 4 listener, backend, health-check, reachability, and ownership requirements in Plan the Control Plane Endpoint.
- If you selected `type: internal`, the Self-built VIP prerequisites are satisfied instead: the VIP is unclaimed, VRRP and gratuitous ARP are not blocked by the dvPortGroup security policy or an NSX-T distributed firewall rule, the template can load `ip_vs`, and the `alive` plugin is deployable on the `global` cluster.

2 Create the namespace and vCenter credential secret

Create the namespace that stores the workload cluster objects.

This workflow stores workload cluster objects in the `cpaas-system` namespace. In the manifests and commands below, replace every `<namespace>` placeholder with `cpaas-system`.

00-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
```

Apply the namespace:

```
kubectl apply -f 00-namespace.yaml
```

Create the vCenter credential secret referenced by `VSphereCluster.spec.identityRef`.

Author the credential file with an editor rather than typing the values at a shell prompt. A shell records the whole command, password included, in its history; a file written by an editor leaves no such copy. Keep it outside `capv-cluster/` so it is never mistaken for a manifest.

vsphere-credentials.env

```
username=<vsphere_username>
password=<vsphere_password>
```

Each line is `key=value`. Do not quote the values — `kubectl` takes quote characters literally. A value may contain `=`; only the first `=` on a line separates the key from the value.

Restrict the file, create the Secret from it, then remove it:

```
chmod 600 ../vsphere-credentials.env

VSPHERE_SECRET_NAME="<credentials_secret_name>"
VSPHERE_NAMESPACE="<namespace>"

kubectl create secret generic "${VSPHERE_SECRET_NAME}" \
  --namespace "${VSPHERE_NAMESPACE}" \
  --from-env-file=../vsphere-credentials.env

rm -f ../vsphere-credentials.env
```

Do not create credential Secrets with kubectl apply

Client-side apply stores the manifest it applied — including the vCenter password — in the `kubectl.kubernetes.io/last-applied-configuration` annotation on the Secret itself. `kubectl create` writes no such annotation. This applies whether the values were written as `stringData` or as base64 `data`; base64 is an encoding, not protection. The same applies to the CPI credential later on this page.

3 Create the `Cluster` and `VSphereCluster` objects

Create the base cluster manifest with the workload cluster network settings, the control plane endpoint, and the vCenter connection settings. Set `cpaas.io/registry-address` to the permanent platform Registry in the form `<registry_address>` (`<host>:<port>` only).

The manifest below uses an external LoadBalancer, which is the baseline on this page. For a Self-built VIP, add the `spec.controlPlaneLoadBalancer` block from Enable the Self-built VIP in `10-cluster.yaml` to the `VSphereCluster` object.

`10-cluster.yaml`

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: <cluster_name>
  namespace: <namespace>
  labels:
    cluster.x-k8s.io/cluster-name: <cluster_name>
    cluster-type: VSphere
    addons.cluster.x-k8s.io/vsphere-cpi: "enabled"
    # Optional. Bind this cluster to a dedicated image registry inste
ad of the
    # registry of the `global` cluster. Read only at cluster creatio
n.
    # cpaas.io/registry-reference: <registry_credential_name>
  annotations:
    capi.cpaas.io/resource-group-version: infrastructure.cluster.x-k8
s.io/v1beta1
    capi.cpaas.io/resource-kind: VSphereCluster
    cpaas.io/sentry-deploy-type: Baremetal
    cpaas.io/alb-address-type: ClusterAddress
    cpaas.io/network-type: kube-ovn
    cpaas.io/kube-ovn-version: <kube_ovn_version>
    cpaas.io/kube-ovn-join-cidr: <kube_ovn_join_cidr>
    cpaas.io/registry-address: <registry_address>
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - <pod_cidr>
    services:
      cidrBlocks:
      - <service_cidr>
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: <cluster_name>-kcp
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: VSphereCluster
    name: <cluster_name>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereCluster
```

```
metadata:
  name: <cluster_name>
  namespace: <namespace>
spec:
  controlPlaneEndpoint:
    host: "<vip>"
    port: <api_server_port>
  identityRef:
    kind: Secret
    name: <credentials_secret_name>
  server: "<vsphere_server>"
  thumbprint: "<thumbprint>"
```

Apply the manifest:

```
kubectl apply -f 10-cluster.yaml
```

4 Create the vSphere CPI delivery resources

Create a `ClusterResourceSet` so the workload cluster receives the vSphere CPI configuration and manifests automatically after the workload API server becomes reachable.

INFO

In the baseline workflow `VSphereCluster.spec.failureDomainSelector` is intentionally not set and the CPI `vsphere.conf` does not include a `[Labels]` block. Both are required only after you enable failure domains; configure them together as described in Multiple datacenters and failure domains. Adding `[Labels]` to `vsphere.conf` without matching `VSphereFailureDomain` objects causes the CPI to look up zone and region tags that do not exist.

INFO

The vSphere CPI TLS bypass option is `insecure-flag`. Keep `insecure-flag = "1"` in the `[Global]` section of `<cluster_name>-vsphere-cpi-config` so the CPI can connect

when the vCenter certificate is self-signed or not trusted by the workload cluster nodes. The CPI applies the global value to vCenter entries that do not set their own `insecure-flag`.

WARNING

The CPI `ConfigMap`, `Secret`, and `ClusterResourceSet` resources **must** be created in the same namespace as the `Cluster` resource. In this guide that namespace is `cpaas-system`. A `ClusterResourceSet` can only match clusters within its own namespace; deploying it in a different namespace will silently prevent resource delivery.

INFO

The `kube-ovn` configuration in the `Cluster` annotations is consumed by platform controllers. This document does not install the network plugin directly.

TIP

This manifest is long and contains nested YAML inside `data` fields. Validate the manifest before applying: `kubectl apply --dry-run=client -f 15-vsphere-cpi-clusterresourceset.yaml`.

The manifest declares the two ConfigMaps and the `ClusterResourceSet`. It does not declare the CPI credential Secret that the `ClusterResourceSet` also references — that Secret carries the vCenter password and is created separately in the next step.

`15-vsphere-cpi-clusterresourceset.yaml`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <cluster_name>-vsphere-cpi-config
  namespace: <namespace>
data:
  data: |
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: cloud-config
      namespace: kube-system
    data:
      vsphere.conf: |
        [Global]
        secret-name = "vsphere-cloud-secret"
        secret-namespace = "kube-system"
        service-account = "cloud-controller-manager"
        port = "443"
        insecure-flag = "1"
        datacenters = "<cpi_datacenters>"

        [VirtualCenter "<vsphere_server>"]
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: <cluster_name>-vsphere-cpi-manifests
  namespace: <namespace>
data:
  data: |
    apiVersion: v1
    kind: ServiceAccount
    metadata:
      name: cloud-controller-manager
      namespace: kube-system
    automountServiceAccountToken: false
    ---
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: system:cloud-controller-manager
    rules:
```

```
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "patch", "update"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["*"]
  - apiGroups: [""]
    resources: ["nodes/status"]
    verbs: ["patch"]
  - apiGroups: [""]
    resources: ["services"]
    verbs: ["list", "patch", "update", "watch"]
  - apiGroups: [""]
    resources: ["services/status"]
    verbs: ["patch"]
  - apiGroups: [""]
    resources: ["serviceaccounts"]
    verbs: ["create", "get", "list", "watch", "update"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "update", "watch"]
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["create", "get", "list", "watch", "update"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["coordination.k8s.io"]
    resources: ["leases"]
    verbs: ["get", "list", "watch", "create", "update"]
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: servicecatalog.k8s.io:apiserver-authentication-reader
    namespace: kube-system
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: extension-apiserver-authentication-reader
  subjects:
  - apiGroup: ""
    kind: ServiceAccount
    name: cloud-controller-manager
```

```
    namespace: kube-system
- apiGroup: ""
    kind: User
    name: cloud-controller-manager
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
    name: system:cloud-controller-manager
roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: system:cloud-controller-manager
subjects:
- kind: ServiceAccount
    name: cloud-controller-manager
    namespace: kube-system
- kind: User
    name: cloud-controller-manager
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
    annotations:
      scheduler.alpha.kubernetes.io/critical-pod: ""
    labels:
      component: cloud-controller-manager
      tier: control-plane
      k8s-app: vsphere-cloud-controller-manager
    name: vsphere-cloud-controller-manager
    namespace: kube-system
spec:
    selector:
      matchLabels:
        k8s-app: vsphere-cloud-controller-manager
    updateStrategy:
      type: RollingUpdate
    template:
      metadata:
        labels:
          component: cloud-controller-manager
          k8s-app: vsphere-cloud-controller-manager
      spec:
        securityContext:
```

```
            runAsUser: 1001
          automountServiceAccountToken: true
          # Optional: required when the CPI image is stored in a priv
ate
          # registry that needs authentication. The platform automati
cally
          # syncs a dockerconfigjson secret named "global-registry-au
th"
          # into every namespace of the workload cluster when the
          # `global` cluster secret "public-registry-credential"
          # (data.content) is configured. If your environment does no
t
          # use a private registry, remove the imagePullSecrets bloc
k.
          imagePullSecrets:
          - name: global-registry-auth
          serviceAccountName: cloud-controller-manager
          hostNetwork: true
          tolerations:
          - operator: Exists
          - key: node.cloudprovider.kubernetes.io/uninitialized
            value: "true"
            effect: NoSchedule
          - key: node-role.kubernetes.io/master
            effect: NoSchedule
          - key: node.kubernetes.io/not-ready
            effect: NoSchedule
            operator: Exists
          containers:
          - name: vsphere-cloud-controller-manager
            image: <image_registry>/acp/cloud-provider-vsphere:<cpi_i
mage_tag>
            args:
            - --v=2
            - --cloud-provider=vsphere
            - --cloud-config=/etc/cloud/vsphere.conf
            volumeMounts:
            - mountPath: /etc/cloud
              name: vsphere-config-volume
              readOnly: true
            resources:
              requests:
                cpu: 200m
          volumes:
```

```
          - name: vsphere-config-volume
            configMap:
              name: cloud-config
    ---
    apiVersion: v1
    kind: Service
    metadata:
      labels:
        component: cloud-controller-manager
      name: vsphere-cloud-controller-manager
      namespace: kube-system
    spec:
      type: NodePort
      ports:
      - port: 43001
        protocol: TCP
        targetPort: 43001
      selector:
        component: cloud-controller-manager
---
apiVersion: addons.cluster.x-k8s.io/v1beta1
kind: ClusterResourceSet
metadata:
  name: <cluster_name>-vsphere-cpi
  namespace: <namespace>
spec:
  strategy: Reconcile
  clusterSelector:
    matchLabels:
      addons.cluster.x-k8s.io/vsphere-cpi: "enabled"
  resources:
  - name: <cluster_name>-vsphere-cpi-config
    kind: ConfigMap
  - name: <cluster_name>-vsphere-cpi-secret
    kind: Secret
  - name: <cluster_name>-vsphere-cpi-manifests
    kind: ConfigMap
```

Create the CPI credential Secret. It wraps the `vsphere-cloud-secret` that the vSphere CPI reads on the workload cluster, so its payload carries the vCenter password. Author the payload with an editor, for the reason given earlier on this page, and keep it outside `capv-cluster/` so it is not mistaken for a manifest:

vsphere-cpi-secret-payload.txt

```
apiVersion: v1
kind: Secret
metadata:
  name: vsphere-cloud-secret
  namespace: kube-system
type: Opaque
stringData:
  <vsphere_server>.username: <vsphere_username>
  <vsphere_server>.password: <vsphere_password>
```

Restrict the file, create the Secret from it, then remove it:

```
chmod 600 ../vsphere-cpi-secret-payload.txt

CLUSTER_NAME="<cluster_name>"

kubectl create secret generic "${CLUSTER_NAME}-vsphere-cpi-secret" \
  --namespace "${VSPHERE_NAMESPACE}" \
  --type=addons.cluster.x-k8s.io/resource-set \
  --from-file=data=../vsphere-cpi-secret-payload.txt

rm -f ../vsphere-cpi-secret-payload.txt
```

Use the same `<cluster_name>` as everywhere else on this page. `VSPHERE_NAMESPACE` was set when you created the vCenter credential; set it again if you are in a new shell. The Secret's only key must be `data`, which `--from-file=data=` produces.

Then apply the manifest:

```
kubectl apply -f 15-vsphere-cpi-clusterresourceset.yaml
```

5 Create the machine config pools

Create the control plane machine config pool.

INFO

Each node slot declares its NIC layout under `network.primary` (required) and `network.additional` (optional list). The primary NIC's `networkName` is required, and the provider derives the Kubernetes node name, the kubelet serving certificate DNS SAN, and the kubelet `node-ip` from `hostname` and the resolved primary NIC addresses. The `hostname` must be a valid DNS-1123 subdomain.

INFO

`deviceName` is optional. If you do not need to force the guest NIC name, remove the `deviceName` line from every node slot. The provider assigns NIC names such as `eth0`, `eth1` by NIC order.

WARNING

The `dns` entries in `VSphereMachineConfigPool` are kept in the static network configuration, but they might not update the guest operating system's `/etc/resolv.conf` reliably in affected VMware deployments. The control plane and worker bootstrap manifests below therefore also write `/etc/resolv.conf` explicitly through kubeadm `files`.

Slot validation rules

The provider validates the following:

- `spec.configs` declares at least one slot, and every slot declares `network` with a non-empty `network.primary.networkName`.
- Every disk declares a `sizeGiB` of `1` or more.
- A persistent disk's `unitNumber`, when set, is between `0` and `15` and is **never** `7`, which is reserved for the SCSI controller.
- Disk names and mount paths are unique across `persistentDisks` and `ephemeralDisks` within a slot.

- Hostnames and primary IP addresses are unique within the pool **and** across every other pool bound to the same `Cluster` in the namespace.

Allocated slot restrictions

Once a slot's `status.configStatuses[].state` is `InUse` or `Released`, you cannot remove the slot, change its `network.primary.ip` or `.ipv6`, remove a disk from `persistentDisks[]`, or change a persistent disk's `sizeGiB` or `mountPath`.

Do not add a disk to a slot that is `InUse` or `Released`: the provider cannot attach it to the running node and reports errors for that node, and an added persistent disk cannot be removed again until the node is replaced. Plan all disks of a slot before the first Machine takes it.

Plan hostnames and addresses before the first Machine allocates the slot. For scale-out and runtime changes, see Manage Nodes on VMware vSphere.

16-vspheremachineconfigpool-control-plane.yaml

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineConfigPool
metadata:
  name: <cluster_name>-cp-pool
  namespace: <namespace>
spec:
  clusterRef:
    apiVersion: cluster.x-k8s.io/v1beta1
    kind: Cluster
    name: <cluster_name>
  datacenter: "<default_datacenter>"
  releaseDelayHours: <release_delay_hours>
  configs:
  - hostname: "<cp_node_name_1>"
    datacenter: "<master_01_datacenter>"
    network:
      primary:
        networkName: "<nic1_network_name>"
        deviceName: "<nic1_device_name>"
        ip: "<master_01_nic1_ip>/<nic1_prefix>"
        gateway: "<nic1_gateway>"
        dns:
        - "<nic1_dns_1>"
    persistentDisks:
    - name: var-cpaas
      sizeGiB: <cp_var_cpaas_size_gib>
      mountPath: /var/cpaas
      fsFormat: ext4
    ephemeralDisks:
    - name: var-lib-kubelet
      sizeGiB: <cp_var_lib_kubelet_size_gib>
      mountPath: /var/lib/kubelet
      fsFormat: ext4
    - name: var-lib-containerd
      sizeGiB: <cp_var_lib_containerd_size_gib>
      mountPath: /var/lib/containerd
      fsFormat: ext4
    - name: var-lib-etcd
      sizeGiB: <cp_var_lib_etcd_size_gib>
      mountPath: /var/lib/etcd
      fsFormat: ext4
  - hostname: "<cp_node_name_2>"
    datacenter: "<master_02_datacenter>"
```

```
      network:
        primary:
          networkName: "<nic1_network_name>"
          deviceName: "<nic1_device_name>"
          ip: "<master_02_nic1_ip>/<nic1_prefix>"
          gateway: "<nic1_gateway>"
          dns:
          - "<nic1_dns_1>"
      persistentDisks:
      - name: var-cpaas
        sizeGiB: <cp_var_cpaas_size_gib>
        mountPath: /var/cpaas
        fsFormat: ext4
      ephemeralDisks:
      - name: var-lib-kubelet
        sizeGiB: <cp_var_lib_kubelet_size_gib>
        mountPath: /var/lib/kubelet
        fsFormat: ext4
      - name: var-lib-containerd
        sizeGiB: <cp_var_lib_containerd_size_gib>
        mountPath: /var/lib/containerd
        fsFormat: ext4
      - name: var-lib-etcd
        sizeGiB: <cp_var_lib_etcd_size_gib>
        mountPath: /var/lib/etcd
        fsFormat: ext4
    - hostname: "<cp_node_name_3>"
      datacenter: "<master_03_datacenter>"
      network:
        primary:
          networkName: "<nic1_network_name>"
          deviceName: "<nic1_device_name>"
          ip: "<master_03_nic1_ip>/<nic1_prefix>"
          gateway: "<nic1_gateway>"
          dns:
          - "<nic1_dns_1>"
      persistentDisks:
      - name: var-cpaas
        sizeGiB: <cp_var_cpaas_size_gib>
        mountPath: /var/cpaas
        fsFormat: ext4
      ephemeralDisks:
      - name: var-lib-kubelet
        sizeGiB: <cp_var_lib_kubelet_size_gib>
```

```
      mountPath: /var/lib/kubelet
      fsFormat: ext4
    - name: var-lib-containerd
      sizeGiB: <cp_var_lib_containerd_size_gib>
      mountPath: /var/lib/containerd
      fsFormat: ext4
    - name: var-lib-etcd
      sizeGiB: <cp_var_lib_etcd_size_gib>
      mountPath: /var/lib/etcd
      fsFormat: ext4
```

Create the worker machine config pool.

17-vspheremachineconfigpool-worker.yaml

```yaml
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineConfigPool
metadata:
  name: <cluster_name>-worker-pool
  namespace: <namespace>
spec:
  clusterRef:
    apiVersion: cluster.x-k8s.io/v1beta1
    kind: Cluster
    name: <cluster_name>
  datacenter: "<default_datacenter>"
  releaseDelayHours: <release_delay_hours>
  configs:
  - hostname: "<worker_node_name_1>"
    datacenter: "<worker_01_datacenter>"
    network:
      primary:
        networkName: "<nic1_network_name>"
        deviceName: "<nic1_device_name>"
        ip: "<worker_01_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
```

Apply both manifests:

```
kubectl apply -f 16-vspheremachineconfigpool-control-plane.yaml
kubectl apply -f 17-vspheremachineconfigpool-worker.yaml
```

Verify the pool binding and slot state before you create Machines:

```
kubectl -n <namespace> get vspheremachineconfigpool
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp
-pool \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status} ({.reas
on}){"\n"}{end}'
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp
-pool -o yaml
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-wo
rker-pool -o yaml
```

The list output carries `TOTAL`, `AVAILABLE`, and `ALLOCATED` columns, taken from `status.total`, `status.available`, and `status.allocated`. Before you create Machines, expect `ClusterRefReady=True`, `VCenterAvailable=True`, `MembersValid=True`, `MembersUnique=True`, and `Ready=True`.

`SlotAvailable` is deliberately not part of the `Ready` summary. A pool whose slots are all allocated still reports `Ready=True` with `SlotAvailable=False`, and that combination is normal during a `maxSurge: 0` rollout rather than a contradiction.

The pool is also the authority for persistent-disk replacement state. When a VM is replaced, its slot changes from `InUse` to `Released`; a replacement Machine that references the same existing pool can reuse that slot and its VMDKs immediately. `releaseDelayHours` controls reclaim when the slot remains unused; it is not a reuse delay.

Do not create a new pool name as a routine retry for Machine replacement. A different `VSphereMachineConfigPool.metadata.name` creates independent slots and VMDKs, so repeated teardown and redeployment can temporarily consume multiple complete disk sets. Before another deployment on a capacity-constrained datastore, wait for earlier pools and their VMDKs to finish provider-managed deletion. See Persistent disk lifecycle for the teardown checks. The live signal is `status.persistentDiskStatuses[].phase`, whose values are `Creating`,

`Attached`, `Available`, `Reclaiming`, `Reclaimed`, and `Error`. The `status.configStatuses[].reclaimStatus` subtree and the `spec.configs[].persistentDisks[].volumePath` and `.diskUUID` fields are deprecated, frozen, and scheduled for removal.

6 Apply the optional failure-domain objects

Skip this step for the baseline single-datacenter topology.

If failure domains are enabled, complete Multiple datacenters and failure domains, then apply the failure-domain objects before creating the control plane:

```
kubectl apply -f 18-failure-domains.yaml
kubectl get vspherefailuredomain,vspheredeploymentzone
```

Verify that every `VSphereFailureDomain` and `VSphereDeploymentZone` referenced by the cluster exists. Do not proceed until the deployment zones are available. Treat the following as one configuration set:

- The `VSphereFailureDomain` and `VSphereDeploymentZone` objects in `18-failure-domains.yaml`
- `VSphereCluster.spec.failureDomainSelector` in `10-cluster.yaml`
- The CPI `[Labels]` block in `15-vsphere-cpi-clusterresourceset.yaml`

Do not add `failureDomainSelector` or the CPI `[Labels]` block to the baseline manifests when failure domains are not enabled.

7 Create the control plane objects

Create the `VSphereMachineTemplate` and `KubeadmControlPlane` objects. Replace the placeholders in the following full template with the values collected in the checklist document.

> **Kubernetes 1.35 kubelet settings**
>
> The baseline control-plane and worker manifests omit `imagePullCredentialsVerificationPolicy` and are valid for Kubernetes 1.34 or earlier.

For Kubernetes 1.35 or later, add the following field to the `KubeletConfiguration` JSON in both `20-control-plane.yaml` and `30-workers-md-0.yaml` before applying either manifest:

```
"imagePullCredentialsVerificationPolicy": "NeverVerify",
```

`cloneMode` and `diskGiB` both remain present in the template because CAPV accepts both fields. In practice, `diskGiB` only affects the system disk when the actual clone operation is `fullClone`. If `cloneMode` is `linkedClone` and the template has a usable snapshot, CAPV completes a linked clone and the system disk size remains equal to the source template. If no usable snapshot exists, CAPV falls back to `fullClone`, and `diskGiB` applies again.

System disk and persistent disks are separate

`VSphereMachineTemplate.spec.template.spec.diskGiB` sets only the VM system disk size. It is not the total capacity of all disks on the node.

Data disks are declared separately under `VSphereMachineConfigPool.spec.configs[].persistentDisks[]` and `.ephemeralDisks[]`. Do not add the data disk sizes to `diskGiB`; otherwise the VM can receive a larger system disk plus the separate data disks, which doubles the intended capacity.

For `fullClone`, `diskGiB` must be greater than or equal to the system disk size in the OS image template. For `linkedClone`, the system disk remains at the template size and `diskGiB` is ignored.

`20-control-plane.yaml`

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineTemplate
metadata:
  name: <cluster_name>-control-plane
  namespace: <namespace>
spec:
  template:
    spec:
      server: "<vsphere_server>"
      template: "<template_name>"
      cloneMode: <clone_mode>
      folder: "<vm_folder>"
      datastore: "<cp_system_datastore>"
      diskGiB: <cp_system_disk_gib>
      memoryMiB: <cp_memory_mib>
      numCPUs: <cp_num_cpus>
      os: Linux
      powerOffMode: <power_off_mode>
      network:
        devices:
        - networkName: "<nic1_network_name>"
      machineConfigPoolRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: VSphereMachineConfigPool
        name: <cluster_name>-cp-pool
        namespace: <namespace>
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: <cluster_name>-kcp
  namespace: <namespace>
spec:
  rolloutStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
  version: "<k8s_version>"
  replicas: <cp_replicas>
  machineTemplate:
    nodeDrainTimeout: 1m
    nodeDeletionTimeout: 5m
    metadata:
```

```
      labels:
        node-role.kubernetes.io/control-plane: ""
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      kind: VSphereMachineTemplate
      name: <cluster_name>-control-plane
  kubeadmConfigSpec:
    users:
    - name: boot
      sudo: ALL=(ALL) NOPASSWD:ALL
      sshAuthorizedKeys:
      - "<ssh_public_key>"
    files:
    - path: /etc/resolv.conf
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        nameserver <nic1_dns_1>
    - path: /etc/kubernetes/admission/psa-config.yaml
      owner: "root:root"
      permissions: "0644"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: AdmissionConfiguration
        plugins:
        - name: PodSecurity
          configuration:
            apiVersion: pod-security.admission.config.k8s.io/v1
            kind: PodSecurityConfiguration
            defaults:
              enforce: "privileged"
              enforce-version: "latest"
              audit: "baseline"
              audit-version: "latest"
              warn: "baseline"
              warn-version: "latest"
            exemptions:
              usernames: []
              runtimeClasses: []
              namespaces:
              - kube-system
              - <namespace>
    - path: /etc/kubernetes/patches/kubeletconfiguration0+strategic.j
```

```
son
      owner: "root:root"
      permissions: "0644"
      content: |
        {
          "apiVersion": "kubelet.config.k8s.io/v1beta1",
          "kind": "KubeletConfiguration",
          "protectKernelDefaults": true,
          "streamingConnectionIdleTimeout": "5m",
          "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
          "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key"
        }
    # Generate the encryption key with: head -c 32 /dev/urandom | bas
e64
    - path: /etc/kubernetes/encryption-provider.conf
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        apiVersion: apiserver.config.k8s.io/v1
        kind: EncryptionConfiguration
        resources:
        - resources:
          - secrets
          providers:
          - aescbc:
              keys:
              - name: key1
                secret: <encryption_provider_secret>
    - path: /etc/kubernetes/audit/policy.yaml
      owner: "root:root"
      append: false
      permissions: "0644"
      content: |
        apiVersion: audit.k8s.io/v1
        kind: Policy
        omitStages:
        - "RequestReceived"
        rules:
        - level: None
          users:
          - system:kube-controller-manager
          - system:kube-scheduler
          - system:serviceaccount:kube-system:endpoint-controller
```

```
      verbs: ["get", "update"]
      namespaces: ["kube-system"]
      resources:
      - group: ""
        resources: ["endpoints"]
    - level: None
      nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
    - level: None
      resources:
      - group: ""
        resources: ["events"]
    - level: None
      resources:
      - group: "devops.alauda.io"
    - level: None
      verbs: ["get", "list", "watch"]
    - level: None
      resources:
      - group: "coordination.k8s.io"
        resources: ["leases"]
    - level: None
      resources:
      - group: "authorization.k8s.io"
        resources: ["subjectaccessreviews", "selfsubjectaccessrev
iews"]
      - group: "authentication.k8s.io"
        resources: ["tokenreviews"]
    - level: None
      resources:
      - group: "app.alauda.io"
        resources: ["imagewhitelists"]
      - group: "k8s.io"
        resources: ["namespaceoverviews"]
    - level: Metadata
      resources:
      - group: ""
        resources: ["secrets", "configmaps"]
    - level: Metadata
      resources:
      - group: "operator.connectors.alauda.io"
        resources: ["installmanifests"]
```

```
    - group: "operators.katanomi.dev"
      resources: ["katanomis"]
  - level: RequestResponse
    resources:
    - group: ""
    - group: "aiops.alauda.io"
    - group: "apps"
    - group: "app.k8s.io"
    - group: "authentication.istio.io"
    - group: "auth.alauda.io"
    - group: "autoscaling"
    - group: "asm.alauda.io"
    - group: "clusterregistry.k8s.io"
    - group: "crd.alauda.io"
    - group: "infrastructure.alauda.io"
    - group: "monitoring.coreos.com"
    - group: "operators.coreos.com"
    - group: "networking.istio.io"
    - group: "extensions.istio.io"
    - group: "install.istio.io"
    - group: "security.istio.io"
    - group: "telemetry.istio.io"
    - group: "opentelemetry.io"
    - group: "networking.k8s.io"
    - group: "portal.alauda.io"
    - group: "rbac.authorization.k8s.io"
    - group: "storage.k8s.io"
    - group: "tke.cloud.tencent.com"
    - group: "devopsx.alauda.io"
    - group: "core.katanomi.dev"
    - group: "deliveries.katanomi.dev"
    - group: "integrations.katanomi.dev"
    - group: "artifacts.katanomi.dev"
    - group: "builds.katanomi.dev"
    - group: "versioning.katanomi.dev"
    - group: "sources.katanomi.dev"
    - group: "tekton.dev"
    - group: "operator.tekton.dev"
    - group: "eventing.knative.dev"
    - group: "flows.knative.dev"
    - group: "messaging.knative.dev"
    - group: "operator.knative.dev"
    - group: "sources.knative.dev"
    - group: "operator.devops.alauda.io"
```

```
          - group: "flagger.app"
          - group: "jaegertracing.io"
          - group: "velero.io"
            resources: ["deletebackuprequests"]
          - group: "connectors.alauda.io"
          - group: "operator.connectors.alauda.io"
            resources: ["connectorscores", "connectorsgits", "connect
orsocis"]
        - level: Metadata
    - path: /usr/local/bin/capv-load-local-images.sh
      owner: "root:root"
      permissions: "0755"
      content: |
        #!/bin/bash
        set -euo pipefail
        until mountpoint -q /var/lib/containerd; do
          echo "waiting for /var/lib/containerd mount"
          sleep 1
        done
        systemctl restart containerd
        until systemctl is-active --quiet containerd; do
          echo "waiting for containerd"
          sleep 1
        done
        if [ ! -d "/root/images" ]; then
          echo "ERROR: /root/images directory not found" >&2
          exit 1
        fi
        image_count=0
        for image_file in /root/images/*.tar; do
          if [ -f "$image_file" ]; then
            echo "importing image: $image_file"
            ctr -n k8s.io images import "$image_file"
            image_count=$((image_count + 1))
          fi
        done
        if [ "$image_count" -eq 0 ]; then
          echo "ERROR: no tar files found in /root/images" >&2
          exit 1
        fi
        echo "imported $image_count images"
    preKubeadmCommands:
    - hostnamectl set-hostname "{{ ds.meta_data.hostname }}"
    - echo "::1         ipv6-localhost ipv6-loopback localhost6 local
```

```
      echo "::1         ipv6-localhost ipv6-loopback localhost6 local
host6.localdomain6" >/etc/hosts
    - echo "127.0.0.1   {{ ds.meta_data.hostname }} {{ local_hostname
}} localhost localhost.localdomain localhost4 localhost4.localdomain
4" >>/etc/hosts
    - while ! ip route | grep -q "default via"; do sleep 1; done; ech
o "NetworkManager started"
    - /usr/local/bin/capv-load-local-images.sh
    postKubeadmCommands:
    - chmod 600 /var/lib/kubelet/config.yaml
    clusterConfiguration:
      imageRepository: <image_registry>/tkestack
      dns:
        imageTag: <dns_image_tag>
      etcd:
        local:
          imageTag: <etcd_image_tag>
      apiServer:
        extraArgs:
          admission-control-config-file: /etc/kubernetes/admission/ps
a-config.yaml
          audit-log-format: json
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "200"
          audit-log-mode: batch
          audit-log-path: /etc/kubernetes/audit/audit.log
          audit-policy-file: /etc/kubernetes/audit/policy.yaml
          encryption-provider-config: /etc/kubernetes/encryption-prov
ider.conf
          kubelet-certificate-authority: /etc/kubernetes/pki/ca.crt
          profiling: "false"
          tls-cipher-suites: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_P
OLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHAC
HA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
          tls-min-version: VersionTLS12
        extraVolumes:
        - hostPath: /etc/kubernetes
          mountPath: /etc/kubernetes
          name: vol-dir-0
          pathType: Directory
      controllerManager:
        extraArgs:
          bind-address: "::"
```

```
          bind-address: "::"
          cloud-provider: external
          flex-volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-pl
ugins/volume/exec/"
          profiling: "false"
          tls-min-version: VersionTLS12
      scheduler:
        extraArgs:
          bind-address: "::"
          profiling: "false"
          tls-min-version: VersionTLS12
    initConfiguration:
      nodeRegistration:
        criSocket: /var/run/containerd/containerd.sock
        ignorePreflightErrors:
        - ImagePull
        kubeletExtraArgs:
          cloud-provider: external
          node-labels: kube-ovn/role=master
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
        name: '{{ local_hostname }}'
      patches:
        directory: /etc/kubernetes/patches
    joinConfiguration:
      nodeRegistration:
        criSocket: /var/run/containerd/containerd.sock
        ignorePreflightErrors:
        - ImagePull
        kubeletExtraArgs:
          cloud-provider: external
          node-labels: kube-ovn/role=master
          volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugin
s/volume/exec/"
        name: '{{ local_hostname }}'
      patches:
        directory: /etc/kubernetes/patches
```

`rolloutStrategy.rollingUpdate.maxSurge: 0` and a `replicas` value of `3` or more are not stylistic choices. A `KubeadmControlPlane` whose infrastructure template sets `machineConfigPoolRef` requires both. See Provider Requirements.

Apply the manifest:

```
kubectl apply -f 20-control-plane.yaml
```

8 Create the worker objects

Create the worker machine template, bootstrap template, and `MachineDeployment`.

The baseline worker kubelet patch omits the Kubernetes 1.35-only field. If the selected version is Kubernetes 1.35 or later, add `imagePullCredentialsVerificationPolicy: NeverVerify` here as well as in `20-control-plane.yaml`, as described in the control-plane step.

30-workers-md-0.yaml

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereMachineTemplate
metadata:
  name: <cluster_name>-worker
  namespace: <namespace>
spec:
  template:
    spec:
      server: "<vsphere_server>"
      template: "<template_name>"
      cloneMode: <clone_mode>
      folder: "<vm_folder>"
      datastore: "<worker_system_datastore>"
      diskGiB: <worker_system_disk_gib>
      memoryMiB: <worker_memory_mib>
      numCPUs: <worker_num_cpus>
      os: Linux
      powerOffMode: <power_off_mode>
      network:
        devices:
        - networkName: "<nic1_network_name>"
      machineConfigPoolRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: VSphereMachineConfigPool
        name: <cluster_name>-worker-pool
        namespace: <namespace>
---
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: KubeadmConfigTemplate
metadata:
  name: <cluster_name>-worker-bootstrap
  namespace: <namespace>
spec:
  template:
    spec:
      files:
      - path: /etc/resolv.conf
        owner: "root:root"
        append: false
        permissions: "0644"
        content: |
          nameserver <nic1_dns_1>
      - path: /etc/kubernetes/patches/kubeletconfiguration0+strategi
```

```
c.json
        owner: "root:root"
        permissions: "0644"
        content: |
          {
            "apiVersion": "kubelet.config.k8s.io/v1beta1",
            "kind": "KubeletConfiguration",
            "protectKernelDefaults": true,
            "staticPodPath": null,
            "streamingConnectionIdleTimeout": "5m",
            "tlsCertFile": "/etc/kubernetes/pki/kubelet.crt",
            "tlsPrivateKeyFile": "/etc/kubernetes/pki/kubelet.key"
          }
      - path: /usr/local/bin/capv-load-local-images.sh
        owner: "root:root"
        permissions: "0755"
        content: |
          #!/bin/bash
          set -euo pipefail
          until mountpoint -q /var/lib/containerd; do
            echo "waiting for /var/lib/containerd mount"
            sleep 1
          done
          systemctl restart containerd
          until systemctl is-active --quiet containerd; do
            echo "waiting for containerd"
            sleep 1
          done
          if [ ! -d "/root/images" ]; then
            echo "ERROR: /root/images directory not found" >&2
            exit 1
          fi
          image_count=0
          for image_file in /root/images/*.tar; do
            if [ -f "$image_file" ]; then
              echo "importing image: $image_file"
              ctr -n k8s.io images import "$image_file"
              image_count=$((image_count + 1))
            fi
          done
          if [ "$image_count" -eq 0 ]; then
            echo "ERROR: no tar files found in /root/images" >&2
            exit 1
          fi
```

```
          echo "imported $image_count images"
      joinConfiguration:
        nodeRegistration:
          criSocket: /var/run/containerd/containerd.sock
          ignorePreflightErrors:
          - ImagePull
          kubeletExtraArgs:
            cloud-provider: external
            volume-plugin-dir: "/opt/libexec/kubernetes/kubelet-plugi
ns/volume/exec/"
          name: '{{ local_hostname }}'
        patches:
          directory: /etc/kubernetes/patches
      preKubeadmCommands:
      - hostnamectl set-hostname "{{ ds.meta_data.hostname }}"
      - echo "::1         ipv6-localhost ipv6-loopback localhost6 loc
alhost6.localdomain6" >/etc/hosts
      - echo "127.0.0.1   {{ ds.meta_data.hostname }} {{ local_hostna
me }} localhost localhost.localdomain localhost4 localhost4.localdoma
in4" >>/etc/hosts
      - while ! ip route | grep -q "default via"; do sleep 1; done; e
cho "NetworkManager started"
      - /usr/local/bin/capv-load-local-images.sh
      postKubeadmCommands:
      - chmod 600 /var/lib/kubelet/config.yaml
      users:
      - name: boot
        sudo: ALL=(ALL) NOPASSWD:ALL
        sshAuthorizedKeys:
        - "<ssh_public_key>"
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster_name>-md-0
  namespace: <namespace>
spec:
  clusterName: <cluster_name>
  replicas: <worker_replicas>
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
```

```
  selector:
    matchLabels:
      nodepool: md-0
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: <cluster_name>
        nodepool: md-0
    spec:
      clusterName: <cluster_name>
      version: "<k8s_version>"
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: KubeadmConfigTemplate
          name: <cluster_name>-worker-bootstrap
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: VSphereMachineTemplate
        name: <cluster_name>-worker
```

`strategy.rollingUpdate.maxSurge: 0` is likewise required for a `MachineDeployment` whose infrastructure template sets `machineConfigPoolRef`. With the default `maxSurge` of `1`, a rollout needs an extra slot that a full pool does not have, and it stalls. The example uses `maxUnavailable: 1`.

Apply the manifest:

```
kubectl apply -f 30-workers-md-0.yaml
```

In the baseline workflow, note the following worker-specific rules:

- `failureDomain` is not set by default in the main worker manifest because the baseline workflow assumes a single datacenter. If you need a worker `MachineDeployment` to land in a specific `VSphereDeploymentZone`, add `failureDomain` as described in Multiple datacenters and failure domains.
- Some environments add extra runtime-image replacement commands or service-restart commands to `KubeadmConfigTemplate`. Those commands are intentionally not included in the baseline sample. Add them only when the platform requirements in your environment explicitly require them.

9 Wait for the cluster to become ready

After all manifests are applied, the cluster creation is asynchronous. Monitor the progress with:

```
kubectl -n <namespace> get cluster,kubeadmcontrolplane,machinedeployment,machine -w
```

Wait until `KubeadmControlPlane` reports the expected number of ready replicas and all `Machine` objects reach the `Running` phase before proceeding to verification.

Verification

Use the following commands to verify the cluster creation workflow.

1. Check the CPI delivery resources in the `global` cluster:

```
kubectl -n <namespace> get clusterresourceset
kubectl -n <namespace> get clusterresourcesetbinding
```

2. Export the workload kubeconfig:

```
kubectl -n <namespace> get secret <cluster_name>-kubeconfig -o jsonpath='{.data.value}' | base64 -d > /tmp/<cluster_name>.kubeconfig
```

3. Check whether the vSphere CPI daemonset is created in the workload cluster:

```
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n kube-system get daemonset
```

4. Check the `global` cluster objects:

```
kubectl -n <namespace> get cluster,vspherecluster,kubeadmcontrolplane,machinedeployment,machine,vspheremachine,vspherevm
```

5. Check the workload nodes:

```
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
```

6. Check the pool slot counters and the per-disk state of both lists:

```
kubectl -n <namespace> get vspheremachineconfigpool
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp-p
ool \
  -o jsonpath='{range .status.persistentDiskStatuses[*]}{.hostname}{"
"}{.name}{" "}{.phase}{" unit="}{.unitNumber}{"\n"}{end}'
kubectl -n <namespace> get vspheremachineconfigpool <cluster_name>-cp-p
ool \
  -o jsonpath='{range .status.ephemeralDiskStatuses[*]}{.hostname}{" "}
{.name}{" unit="}{.unitNumber}{"\n"}{end}'
```

Confirm the following results:

- `vsphere-cloud-controller-manager` appears in the workload cluster.
- Control plane and worker nodes are created.
- The nodes eventually become `Ready`.
- Each pool reports `Ready=True`, and its `ALLOCATED` count equals the number of running Machines drawn from that pool.
- Every persistent disk on an allocated slot reports `phase: Attached`.
- Every ephemeral disk on an allocated slot appears in `status.ephemeralDiskStatuses[]` with an assigned unit number. Ephemeral disks carry no phase; they are created with the VM.

Troubleshooting

Use the following commands first when the workflow fails:

```
kubectl -n <namespace> describe cluster <cluster_name>
kubectl -n <namespace> describe vsphere cluster <cluster_name>
kubectl -n <namespace> describe kubeadmcontrolplane <cluster_name>-kcp
kubectl -n <namespace> describe machinedeployment <cluster_name>-md-0
kubectl -n <namespace> get cluster,vsphere cluster,kubeadmcontrolplane,mac
hinedeployment,machine,vspheremachine,vspherevm
kubectl -n cpaas-system logs deploy/capi-controller-manager
```

Prioritize the following checks:

- If the CPI resources are not delivered, verify `ClusterResourceSet=true`, `ClusterResourceSet`, and `ClusterResourceSetBinding`.
- If `ClusterResourceSet` exists but no `ClusterResourceSetBinding` is created, check whether the controller has the required delete permission on the referenced `ConfigMap` and `Secret` resources.
- If the network plugin is not installed, verify that the required cluster annotations are present and that the platform controllers processed them.
- If the `cpaas.io/registry-address` annotation is missing or incorrect, verify that it is set to the permanent `<registry_address>` without a scheme or `/v2/`. Then verify the credential that backs it: `public-registry-credential` when the cluster inherits the registry of the `global` cluster, or the Secret named by `cpaas.io/registry-reference` when the cluster uses a dedicated registry.
- If a machine is stuck in `Provisioning`, check `VSphereMachine` conditions for `MachineConfigPoolReady` — it shows whether slot allocation failed due to pool binding or datacenter mismatch. Then read the pool's own conditions: `MembersValid`, `MembersUnique`, `PersistentDisksReady`, `ClusterRefReady`, and `VCenterAvailable`.
- If a machine never bootstraps, check the `BootstrapReady` condition on `VSphereVM` and `VSphereMachine`. The reason `BootstrapSecretGetFailed` means the bootstrap Secret is missing or unreadable; `BootstrapSecretContentInvalid` means it exists but cannot be used. Inspect the `KubeadmConfig` for that Machine and its bootstrap data Secret before looking at vCenter.
- If a node was powered off in vCenter and stays off, that is expected behavior. `VSphereVM` latches `InitialPowerOnCompleted=True` after its first successful power-on and never clears it; from then on a powered-off VM is treated as a deliberate out-of-band action and is

not powered back on. `PoweredOn=False` propagates into `VSphereVM.Ready` and then into `VSphereMachine.Ready`, so reporting not-ready is correct. Power the VM on in vCenter to recover; a rollout is not required. A replacement VM created by a rollout has an unset latch, so its first boot is unaffected.

- If `VSphereMachine` clears `Ready` without an obvious cause, note that `Ready` is cleared while the backing `VSphereVM`, its `providerID`, or its network information is unavailable, and that out-of-band power changes are now observed. Read the `VSphereVM` conditions before treating it as a controller fault.
- If nothing reconciles at all and no condition changes, check whether the management cluster is a disaster-recovery standby. See Standby Management Clusters.
- If a VM is waiting for IP allocation, verify VMware Tools, the static IP settings, and `VSphereVM.status.addresses`.
- If workload `Node` objects remain without `spec.providerID`, first verify the CPI delivery resources and then check for duplicate vCenter guest hostnames. When an old VM in the same datacenter still reports the same guest hostname as a new node, `cloud-provider-vsphere` can fall back to node-name lookup, cache the old VM, and reject the new node because the VM IP does not match the kubelet node IP. Check the leader `vsphere-cloud-controller-manager` logs, the node `SystemUUID`, the real VM UUID, and vCenter guest hostname/IP values. After you fix or remove the duplicate hostname or old VM conflict, restart the workload cluster's `vsphere-cloud-controller-manager` Pods to clear the bad in-memory cache:

  ```
  kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n kube-system dele
  te pod \
    -l k8s-app=vsphere-cloud-controller-manager
  ```

- If datastore space is exhausted, first list all current and terminating `VSphereMachineConfigPool` objects and compare their names with every `VSphereMachineTemplate.spec.template.spec.machineConfigPoolRef.name`. Different pool names own independent slots and VMDKs; repeated deployments can therefore accumulate multiple disk sets. Then inspect `status.configStatuses[]` for `state` and `lastReleasedTime`, and `status.persistentDiskStatuses[]` for `phase`, `lastError`, `retryAfter`, and `taskRef`, together with related Events and the vCenter attachment state. `insufficient disk available` confirms a datastore capacity failure

but does not by itself prove that reclaim failed. Do not manually delete a VMDK or remove a pool finalizer while reuse or provider-managed reclaim is pending.

- Do not delete a `phase: Reclaimed` entry from `status.persistentDiskStatuses[]`. It is a deliberate tombstone: the backing VMDK is already gone, and the record is what stops the controller from re-seeding the disk from the frozen `spec` `volumePath`. Deleting it produces a seed, delete, and re-seed loop that prevents the pool finalizer from ever clearing.
- If the template system disk size does not match the manifest values, first check the actual clone mode. When the VM was created as `linkedClone`, the system disk stays at the template's size and `diskGiB` is ignored. Only `fullClone` uses `diskGiB`, and in that case `diskGiB` must not be smaller than the template disk size.
- If the control plane endpoint does not come up, verify TCP `6443` passthrough, all control-plane backends, HTTPS `/healthz` results, DNS and certificate SANs, and reachability from the `global` cluster and control-plane nodes.
- If the TLS connection to vCenter fails, verify the thumbprint, the vCenter address, and whether proxy settings interfere with the connection.

When you review controller logs, use the following rules:

- `deploy/capi-controller-manager` runs in the `cpaas-system` namespace of the `global` cluster.
- Do not use the workload-cluster kubeconfig to inspect `capi-controller-manager` logs.
- If platform controllers process the cluster network annotations, also inspect the platform network-controller logs and the platform cluster-lifecycle-controller logs.

Admission rejections

An `apply` or `edit` can be rejected before anything is created. Match a fragment of the rejection message against the following table.

Rejection contains	Cause and fix
`maxSurge must be 0 when the infrastructure template references a machineConfigPoolRef`	Set `maxSurge` to the integer `0` on the `KubeadmControlPlane`. Leaving it unset means the Cluster API default `1`.
`must have at least 3 replicas`	Set `KubeadmControlPlane.spec.replicas` to `3` or more. A fixed-IP rollout uses `maxSurge: 0`, which Cluster API permits only for control planes with three or more replicas.
`unit number 7 is reserved for the SCSI controller`	Choose another `unitNumber` between `0` and `15`.
`also used by pool`	A hostname or primary IP collides with a slot in another pool bound to the same `Cluster` in this namespace.
`allocated slot`	The slot is `InUse` or `Released`. The message names what is frozen — the primary IP or IPv6, or a persistent disk's `sizeGiB` or `mountPath` — or reports that the slot, or one of its persistent disks, cannot be removed. See Allocated slot restrictions.
`cannot delete pool while`	The pool still owns `InUse` slots. Delete the Machines that hold them first.
`is already referenced by` or `is bound to`	Two controllers reference one pool. Give each `KubeadmControlPlane` and `MachineDeployment` its own pool.
`cannot change clusterRef while consumerRef is set`	The pool is already bound to a controller. Create a new pool instead of retargeting this one.
`VSphereMachineTemplate spec.template.spec field is immutable`	Create a new template under a new name instead of editing the existing one. See Roll out updated worker node configuration.
`is immutable once controlPlaneLoadBalancer is set`	`spec.controlPlaneLoadBalancer` is write-once. See `controlPlaneLoadBalancer` fields.

Rejection contains	Cause and fix
`cannot be set after the cluster is created`	The cluster was created without `spec.controlPlaneLoadBalancer`, which already means an external LoadBalancer. The endpoint is fixed at creation and cannot be filled in later.

Creation-Time Topology Variants

The baseline manifest intentionally starts with one datacenter and one NIC. Before you create the cluster, use the following variants when the initial topology requires a Self-built control plane VIP, additional NICs, multiple datacenters, failure domains, or extra data disks. Apply one variant at a time and validate the complete manifest set before combining them.

Use a Self-built VIP

Use this variant when the cluster is created with `VSphereCluster.spec.controlPlaneLoadBalancer.type: internal` instead of an external LoadBalancer. Decide the mode first in Control Plane Endpoint Modes; it cannot be changed after the cluster exists.

Self-built VIP prerequisites

Complete all of the following before you select `type: internal`.

- The platform is ACP `v4.4` or later. An earlier platform release does not support the Self-built VIP, whatever the provider version.
- The VIP is IPv4, is currently unclaimed, and lives in the same Layer-2 broadcast domain as the control-plane node IPs. IPv6 and dual-stack endpoints are not accepted.
- The VIP is not one of the slot addresses (`network.primary.ip` or `network.additional[].ip`) declared in this cluster's `VSphereMachineConfigPool` objects, and it falls outside every external IPAM pool range. The provider checks the slot addresses; it cannot read external IPAM pools and will not detect that conflict.
- The `vrid` is unique in that Layer-2 domain.

- The network path allows VRRP (IP protocol `112`, multicast `224.0.0.18`) and gratuitous ARP between the control-plane nodes. Confirm that the dvPortGroup security policy and any NSX-T distributed firewall rule do not block them.
- The VM template exposes the Linux IPVS subsystem (the `ip_vs` module family is loadable) and allows Alive to set `net.ipv4.conf.all.arp_accept=1` and `net.ipv4.vs.conntrack=1`.
- The `alive` cluster plugin is present and deployable on the `global` cluster.

The provider does not carry Alive itself. It creates a `ModuleInfo` on the `global` cluster, and the platform renders the workload `cpaas-system/alive` `AppRelease` from it. Confirm the plugin on the `global` cluster:

```
kubectl get clustermodule global -o jsonpath='{.spec.version}{"\n"}'
kubectl get moduleplugin alive -o jsonpath='{.status.targetClusterVersion
s}{"\n"}'
kubectl get moduleplugin alive -o jsonpath='{.status.latestVersion}
{"\n"}'
kubectl get moduleconfig alive-<alive_version> -o jsonpath='{.status.read
yForDeploy}{"\n"}'
```

The provider resolves `<alive_version>` in this order: the `--plugin-alive-version` override on the provider deployment when it is set, then the `ModulePlugin/alive` entry `status.targetClusterVersions[<ClusterModule spec.version>].version`, and only then `status.latestVersion`. Read the `targetClusterVersions` entry that matches the version printed by the first command first; `latestVersion` applies only when the map has no entry for it.

The workload cluster's own `ClusterModule` does not exist until the cluster is created, so at planning time read the `global` cluster's version as the planning value. It is the closest available proxy, not a guarantee: once the cluster exists, read its own version with `kubectl get clustermodule <cluster_name>` and repeat the `readyForDeploy` check against the `alive` version that this version resolves to. Do that before the control plane finishes bootstrapping, because `type: internal` cannot be changed afterwards. When you are creating the `global` cluster itself, confirm the component version and the presence of the `alive` plugin from the installation package instead.

The last command must print `true` for the version that resolution selects. If `ModulePlugin/alive` is absent, or the matching `ModuleConfig` is missing or reports `false`, the Self-built VIP cannot be installed in this environment. Create the cluster with an external LoadBalancer instead, and leave `spec.controlPlaneLoadBalancer` absent. Contact Alauda technical support for the plugin package that matches your platform release.

`controlPlaneLoadBalancer` fields

Parameter	Type	Description	R
`.spec.controlPlaneLoadBalancer.type`	string (`internal` / `external`)	`internal` installs Alive and manages the VIP. `external` skips Alive and requires a pre-provisioned Layer 4 TCP LoadBalancer. The values are lowercase; other providers use different spelling.	N
`.spec.controlPlaneLoadBalancer.host`	string	Control plane endpoint address. For `internal`, the Self-built VIP, which must be IPv4. For `external`, the load balancer frontend address.	Y
`.spec.controlPlaneLoadBalancer.port`	int (1–65535)	Control plane port. Typically `6443`.	Y
`.spec.controlPlaneLoadBalancer.vrid`	int	Keepalived `virtual_router_id`, `1` – `255`, unique in the control-plane Layer-2 domain.	V `i`
`.spec.controlPlaneLoadBalancer.interface`	string (max 15 characters)	Host interface that carries the VIP. Leave	N w

Parameter	Type	Description	R
		empty to let the provider match the interface holding the node's primary IP.	

> **controlPlaneLoadBalancer is write-once**
>
> The provider accepts `spec.controlPlaneLoadBalancer` only when the cluster is created. After that, `type`, `host`, `port`, `vrid`, and `interface` are all frozen, the field cannot be cleared, and a cluster created without it cannot have it added. Correcting a mistake requires rebuilding the cluster.
>
> Two consequences follow. On multi-NIC nodes, set `interface` explicitly at creation — a wrong auto-detected interface cannot be patched afterwards. And `host` and `port` must describe the same address as `spec.controlPlaneEndpoint`; a mismatch is not accepted.
>
> The `vrid` range is enforced by the provider's admission webhook rather than by the CRD schema. A client-side dry run contacts no server, so it catches neither this nor the schema rules; use `kubectl apply --dry-run=server` to see the rejection without creating the object.

Enable the Self-built VIP in `10-cluster.yaml`

For `type: internal`, add the following block to the `VSphereCluster` object in Create the `Cluster` and `VSphereCluster` objects. Everything else in the baseline manifest is unchanged.

```yaml
spec:
  controlPlaneLoadBalancer:
    type: internal
    host: "<vip>"           # must equal spec.controlPlaneEndpoint.host
    port: <api_server_port> # must equal spec.controlPlaneEndpoint.port
    vrid: <vrid>
    interface: "<vip_interface>" # set explicitly on multi-NIC nodes
```

The provider also reconciles the workload cluster's `kube-system/kube-proxy` configuration for you, setting `ipvs.strictARP: true` and adding `<vip>/32` to `ipvs.excludeCIDRs`,

then rolling the kube-proxy DaemonSet. Do not revert those values by hand; the provider reapplies them and rolls the DaemonSet again.

Verify the Self-built VIP

```
kubectl -n <namespace> get vsphere cluster <cluster_name> \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status} ({.reason})
{"\n"}{end}'
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n kube-system get po
ds -l app=alive -o wide
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig -n cpaas-system get a
pprelease alive
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get --raw='/version'
```

Expect `SelfBuiltLoadBalancerReady=True`, one Alive pod per control-plane node, and a successful API response through the VIP without disabling TLS verification.

The Alive pods run in `kube-system`, not in `cpaas-system`; `cpaas-system` holds the `alive` `AppRelease` the platform renders from the provider's `ModuleInfo`. `SelfBuiltLoadBalancerReady` covers the whole chain, the `AppRelease` and the pods, so check both namespaces when it is `False`.

> **INFO**
>
> `SelfBuiltLoadBalancerReady` reports the VIP chain only. It does not gate `VSphereCluster.status.ready`, so the infrastructure cluster can report ready while the VIP is still converging. The condition reasons are `SelfBuiltLoadBalancerReady`, `SelfBuiltLoadBalancerReconciling`, `SelfBuiltLoadBalancerNotReady`, and `InvalidSelfBuiltLoadBalancerConfiguration`.
>
> While the VIP is not ready, the provider also defers the workload-cluster CoreDNS and kube-proxy repository reconciliation described in Provider-managed fields on your manifests.

Add a second NIC

When nodes require an additional management, storage, or service network, extend the manifests in the following resources:

- `16-vspheremachineconfigpool-control-plane.yaml`
- `17-vspheremachineconfigpool-worker.yaml`
- `20-control-plane.yaml`
- `30-workers-md-0.yaml`
- `18-failure-domains.yaml` if failure domains are enabled

Each node slot declares its NIC layout under `network.primary` and `network.additional`. The primary NIC is used to derive the kubelet `node-ip` and remains the node's primary identity; additional NICs are merged after it in the order listed.

Add the second NIC to each control plane node slot in the machine config pools:

```
network:
  primary:
    networkName: "<nic1_network_name>"
    deviceName: "<nic1_device_name>"
    ip: "<master_01_nic1_ip>/<nic1_prefix>"
    gateway: "<nic1_gateway>"
    dns:
    - "<nic1_dns_1>"
  additional:
  - networkName: "<nic2_network_name>"
    deviceName: "<nic2_device_name>"
    ip: "<master_01_nic2_ip>/<nic2_prefix>"
    gateway: "<nic2_gateway>"
    dns:
    - "<nic2_dns_1>"
```

Apply the same pattern to the worker node slots:

```
network:
  primary:
    networkName: "<nic1_network_name>"
    deviceName: "<nic1_device_name>"
    ip: "<worker_01_nic1_ip>/<nic1_prefix>"
    gateway: "<nic1_gateway>"
    dns:
    - "<nic1_dns_1>"
  additional:
  - networkName: "<nic2_network_name>"
    deviceName: "<nic2_device_name>"
    ip: "<worker_01_nic2_ip>/<nic2_prefix>"
    gateway: "<nic2_gateway>"
    dns:
    - "<nic2_dns_1>"
```

Add the second NIC to the machine templates:

```
network:
  devices:
  - networkName: "<nic1_network_name>"
  - networkName: "<nic2_network_name>"
```

If the DNS server used by the node changes when you add the second NIC, update the `/etc/resolv.conf` file entries in both `20-control-plane.yaml` and `30-workers-md-0.yaml`. The `dns` values in the machine config pool network blocks do not replace the explicit `/etc/resolv.conf` bootstrap file entries.

If failure domains are enabled, update the network list in `VSphereFailureDomain.spec.topology.networks`:

```
topology:
  networks:
  - <nic1_network_name>
  - <nic2_network_name>
```

When you define the second NIC values, prepare the following placeholders in the infrastructure checklist and manifests:

- `<master_01_nic2_ip>`
- `<master_02_nic2_ip>`
- `<master_03_nic2_ip>`
- `<worker_01_nic2_ip>`
- `<worker_02_nic2_ip>` when you also expand the worker pool

For a running cluster, change all three network definitions together when you replace the machines. See Runtime Topology Changes.

Multiple datacenters and failure domains

Use multiple datacenters and failure domains when you need node placement across different vCenter datacenters or compute clusters.

The following principles apply:

- One cluster can define multiple `VSphereFailureDomain` objects.
- Each `VSphereDeploymentZone` references one `VSphereFailureDomain`.
- The control plane uses `VSphereCluster.spec.failureDomainSelector`.
- A worker `MachineDeployment` uses `spec.template.spec.failureDomain` when it must target a specific deployment zone.

Prepare the following placeholders for the first datacenter:

- `<compute_cluster_1>`
- `<default_datastore_1>`
- `<resource_pool_path_1>`
- `<fd_name_1>`
- `<dz_name_1>`

Prepare the following placeholders for the second datacenter:

- `<dc_name_2>`
- `<fd_name_2>`

- `<dz_name_2>`
- `<compute_cluster_2>`
- `<default_datastore_2>`
- `<resource_pool_path_2>`

If you add a third datacenter, continue with the same placeholder pattern:

- `<dc_name_3>`
- `<fd_name_3>`
- `<dz_name_3>`
- `<compute_cluster_3>`
- `<default_datastore_3>`
- `<resource_pool_path_3>`

Create the failure-domain objects in `18-failure-domains.yaml`. The first datacenter also needs a `VSphereFailureDomain` and `VSphereDeploymentZone` when failure domains are enabled:

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereFailureDomain
metadata:
  name: <fd_name_1>
spec:
  region:
    name: region-a
    type: Datacenter
    tagCategory: k8s-region
    autoConfigure: true
  zone:
    name: zone-1
    type: ComputeCluster
    tagCategory: k8s-zone
    autoConfigure: true
  topology:
    datacenter: <default_datacenter>
    computeCluster: <compute_cluster_1>
    datastore: <default_datastore_1>
    networks:
    - <nic1_network_name>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereDeploymentZone
metadata:
  name: <dz_name_1>
spec:
  server: <vsphere_server>
  failureDomain: <fd_name_1>
  controlPlane: true
  placementConstraint:
    resourcePool: <resource_pool_path_1>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereFailureDomain
metadata:
  name: <fd_name_2>
spec:
  region:
    name: region-a
    type: Datacenter
    tagCategory: k8s-region
    autoConfigure: true
```

```
  zone:
    name: zone-2
    type: ComputeCluster
    tagCategory: k8s-zone
    autoConfigure: true
  topology:
    datacenter: <dc_name_2>
    computeCluster: <compute_cluster_2>
    datastore: <default_datastore_2>
    networks:
    - <nic1_network_name>
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereDeploymentZone
metadata:
  name: <dz_name_2>
spec:
  server: <vsphere_server>
  failureDomain: <fd_name_2>
  controlPlane: true
  placementConstraint:
    resourcePool: <resource_pool_path_2>
```

Enable control plane selection across the available failure domains by adding `failureDomainSelector` to the `VSphereCluster` spec in `10-cluster.yaml`:

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: VSphereCluster
metadata:
  name: <cluster_name>
  namespace: <namespace>
spec:
  controlPlaneEndpoint:
    host: "<vip>"
    port: <api_server_port>
  identityRef:
    kind: Secret
    name: <credentials_secret_name>
  server: "<vsphere_server>"
  thumbprint: "<thumbprint>"
  failureDomainSelector: {}
```

An empty selector `{}` matches every `VSphereDeploymentZone` that has `controlPlane: true`. Use match labels to restrict the control plane to a subset of zones.

Also add the `[Labels]` block to the CPI `ConfigMap` so the vSphere CPI publishes the matching zone and region labels on workload nodes. The keys must match the `tagCategory` values used in `VSphereFailureDomain.spec.zone.tagCategory` and `VSphereFailureDomain.spec.region.tagCategory`. Update the `vsphere.conf` data in `15-vsphere-cpi-clusterresourceset.yaml`:

```
    vsphere.conf: |
      [Global]
      secret-name = "vsphere-cloud-secret"
      secret-namespace = "kube-system"
      service-account = "cloud-controller-manager"
      port = "443"
      insecure-flag = "1"
      datacenters = "<cpi_datacenters>"

      [Labels]
      zone = "k8s-zone"
      region = "k8s-region"

      [VirtualCenter "<vsphere_server>"]
```

`failureDomainSelector` and the CPI `[Labels]` block must be enabled together. Adding either one alone leaves the cluster in an inconsistent state: nodes get unresolved zone or region labels, or the control plane cannot select a deployment target.

Set a worker deployment zone when a worker `MachineDeployment` must be pinned to one deployment target. Add `failureDomain` to `spec.template.spec` in `30-workers-md-0.yaml`:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: <cluster_name>-md-0
  namespace: <namespace>
spec:
  clusterName: <cluster_name>
  replicas: <worker_replicas>
  template:
    spec:
      clusterName: <cluster_name>
      failureDomain: <worker_failure_domain>
      version: "<k8s_version>"
      # ... rest of spec
```

Use a `VSphereDeploymentZone` name for `<worker_failure_domain>`, not a `VSphereFailureDomain` name.

Before you enable multiple datacenters, confirm all of the following prerequisites:

1. The template is already synchronized to every target datacenter.
2. The network names are resolvable in every target datacenter.
3. The datastore names are resolvable in every target datacenter.
4. The vSphere CPI datacenter list covers every target datacenter.

Add data disks

The baseline deployment includes the following required data disks:

- **Control plane nodes** (4 disks per node): `var-cpaas` in `persistentDisks[]`, plus `var-lib-kubelet`, `var-lib-containerd`, and `var-lib-etcd` in `ephemeralDisks[]`. Do not remove any of these disks.
- **Worker nodes** (3 disks per node): `var-cpaas` in `persistentDisks[]`, plus `var-lib-kubelet` and `var-lib-containerd` in `ephemeralDisks[]`. Do not remove any of these disks.

`VSphereMachineTemplate.spec.template.spec.diskGiB` is the system disk size, not the total disk capacity of the VM. Keep the data disks under `VSphereMachineConfigPool.spec.configs[].persistentDisks[]` and `.ephemeralDisks[]`. Do not add the data disk sizes to `diskGiB` unless you intentionally want a larger system disk.

If a node needs additional data disks beyond the required set, append more entries to the matching list in the corresponding `VSphereMachineConfigPool` node slot: `persistentDisks` when the data must survive VM replacement, `ephemeralDisks` when it is rebuildable. The following optional fields are especially relevant here:

- **`mountPath`**: If set, the disk is formatted and mounted at the specified path. If omitted, the disk is attached as a raw device with a symlink at `/dev/disk/by-capv/<name>`, allowing an external process to manage it at runtime.
- **`wipeFilesystem`**: Persistent disks only. When `true`, disk content is wiped on the first boot of a new VM. Normal reboots and manual service restarts are not affected. Defaults to `false`.

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

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

The disk is accessible inside the guest OS at `/dev/disk/by-capv/app-data`. On rolling updates, the same VMDK is re-attached to the new VM and the symlink is recreated. The disk is never formatted or mounted automatically; the application is responsible for managing it at runtime.

Choose persistent or ephemeral

A slot declares two disk lists: `spec.configs[].persistentDisks[]` and `spec.configs[].ephemeralDisks[]`.

	`persistentDisks[]`	`ephemeralDisks[]`
Spec fields	`name`, `sizeGiB`, `datastore`, `storagePolicy`, `unitNumber`, `mountPath`, `mountOptions`, `fsFormat`, `wipeFilesystem`	`name`, `sizeGiB`, `datastore`, `storagePolicy`, `mountPath`, `mountOptions`, `fsFormat`
When the VM is deleted	Detached and kept, then reattached to the replacement VM at the same slot	Deleted with the VM
When the VM is recreated	Same VMDK and same content, unless `wipeFilesystem: true`	Always a new empty disk, formatted from scratch
Reclaim and slot-release gating	Participates. Blocks slot release and pool reclaim while still attached	Does not participate
Observed state	`status.persistentDiskStatuses[]`, with path, UUID, unit, phase, and owning machine	`status.ephemeralDiskStatuses[]`, with hostname, name, and unit only
SCSI unit	Pin it with `unitNumber`, or let the controller assign one and read it back from status	Controller-assigned only; there is no spec field
Guest addressing	`/dev/disk/by-capv/<name>`. The guest script locates the device by VMDK `diskUUID`, then by SCSI unit	`/dev/disk/by-capv/<name>`. The guest script locates the device by SCSI unit only

Use `ephemeralDisks` for rebuildable node-local data that should not sit on the system disk — the standard `/var/lib/kubelet`, `/var/lib/containerd`, and `/var/lib/etcd` disks, plus any scratch space. Use `persistentDisks` for anything whose loss is a data loss.

Keep the standard disks in their declared list

`/var/cpaas` must stay in `persistentDisks[]`: it holds platform state that has to survive VM replacement. `/var/lib/kubelet`, `/var/lib/containerd`, and `/var/lib/etcd` belong in `ephemeralDisks[]`; they are rebuilt on every new VM, and declaring them as persistent disks carries stale kubelet, runtime, and etcd member data onto the replacement node.

This split is a creation-time choice. `ephemeralDisks[]` is new in provider `v1.0.17`, so a cluster created on an earlier provider cannot use the standard ephemeral set at all: it declares `/var/cpaas`, `/var/lib/containerd`, and `/var/lib/etcd` as persistent disks, with `wipeFilesystem: true` on the etcd disk. That cluster keeps its layout: a provider upgrade migrates no disk. See Upgrading the Provider.

The following example adds one ephemeral scratch disk alongside the standard worker set:

```
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
- name: scratch
  sizeGiB: 50
  mountPath: /var/scratch
  fsFormat: ext4
```

Disk names and mount paths must be unique across both lists within one slot.

Pin a SCSI unit number

`unitNumber` on a persistent disk keeps guest device ordering stable across VM recreations. Valid values are `0` through `15`, excluding `7`, which is reserved for the SCSI controller, and each value must be unique within the slot.

If you omit `unitNumber`, the controller assigns one at clone time, records it in `status.persistentDiskStatuses[].unitNumber`, and reuses that value when the VM is recreated.

Ephemeral disks have no `unitNumber` field at all. The controller assigns their unit at every clone and reports it in `status.ephemeralDiskStatuses[].unitNumber`, so the standard `/var/lib/kubelet`, `/var/lib/containerd`, and `/var/lib/etcd` disks cannot be pinned.

Relocating the pod log directory

When any disk in a slot mounts at `/var/lib/containerd`, the guest disk-reconcile script points `/var/log/pods` at a directory on that disk, so pod logs land on the data disk instead of the root filesystem. In the standard layout that disk is ephemeral, so relocated pod logs are destroyed with the VM they belong to; ship anything you need to keep to the platform log store.

> **WARNING**
>
> This relocation is written into the guest through `guestinfo` user data **before first boot only**. Upgrading the provider does not retrofit it onto existing machines; the change is picked up when nodes are rolled and replaced. See OS Image Update Without a Kubernetes Version Change. Remediating a running node in place requires an out-of-band script and is not part of this workflow.

Verify creation-time variants

After you apply a topology variant, validate the cluster state with the following commands:

```
kubectl -n <namespace> get cluster,vsphirecluster,kubeadmcontrolplane,machinedeployment,machine,vspheremachine,vspherevm
kubectl --kubeconfig=/tmp/<cluster_name>.kubeconfig get nodes -o wide
kubectl -n <namespace> get vspheremachineconfigpool <pool_name> \
  -o jsonpath='{range .status.ephemeralDiskStatuses[*]}{.hostname}{" "}{.name}{" unit="}{.unitNumber}{"\n"}{end}'
```

Confirm the following results:

- The placement, NIC, or disk definitions are reflected in the target resources.
- New nodes reach the `Ready` state.
- Existing nodes remain healthy after the change.

- Every declared ephemeral disk has an assigned unit number and is addressable in the guest at `/dev/disk/by-capv/<name>`.

Next Steps

For worker scale-out and later template or runtime-topology changes, see Manage Nodes on VMware vSphere. Apply one change at a time and validate the result before you combine multiple changes in the same cluster.

Troubleshooting

Troubleshoot a Workload Clust Troubleshoot a Cluster Stuck in Deleting

Troubleshoot a Workload Cluster Stuck in Provisioned

Use this guide when a workload cluster on Alauda OS has reached `Cluster.status.phase = Provisioned` and the Kubernetes control plane reports ready, but the cluster never becomes fully usable.

The diagnostic flow assumes the workload cluster was created by applying provider Cluster API manifests to the `global` cluster.

TOC

- Scope
- Symptoms
- Invariants of a Healthy Imported Cluster
- Diagnostic Steps
  - Step 1 — Confirm the Cluster API surface is healthy
  - Step 2 — Check Machine and Node conditions
  - Step 3 — Inspect the infrastructure provider reconciler logs
  - Step 4 — Verify the global cluster has imported the workload cluster
  - Step 5 — Verify the ClusterCredential count invariant
- Pattern That Indicates a Failed Import
- What Does Not Resolve This Pattern
- Next Step
- See Also

Scope

The diagnostic flow in this guide targets the `global` **cluster's import controller**, which is shared across all infrastructure providers. The four invariants in Invariants of a Healthy Imported Cluster — `clusters.platform.tkestack.io` presence, the `capi.cpaas.io/imported` label, the `ClusterCredential` count, and the workload-cluster `sentry` ServiceAccount — apply to every provider.

The log strings in this guide use placeholders for reconciler object names because each infrastructure provider uses its own object types (for example, the provider's own `Cluster` and `Machine` infrastructure CRDs). When you read your provider's logs, substitute the placeholders for the real object names that appear in your provider's reconciler output. The diagnostic pattern itself does not depend on the provider — only the object names in the log strings do.

If your environment is Huawei Cloud Stack and the symptoms below also include a stalled `kubeadm init` on a node whose pool config carries a dotted hostname, see Troubleshoot HCS Workload Clusters for the provider-specific pattern after you finish the generic diagnostic flow on this page.

Symptoms

The cluster reaches a partial-success state and stays there. The following indicators appear together:

Where to look	What you see
`Cluster.status.phase` (on the `global` cluster)	`Provisioned`
`KubeadmControlPlane.status.ready`	`true`
`Machine.status.phase`	`Running`

Where to look	What you see
`Machine.status.conditions[?(@.type=="NodeHealthy")]`	`status: False`, `reason: NodeConditionsFailed`, `message: Node condition Ready is False`
`MachineDeployment.status.conditions[?(@.type=="Available")]`	`status: False`, `reason: NotAvailable`, `message: WorkersAvailable: 0 available replicas`
Inside the workload cluster: `kubectl get nodes`	One or more nodes report `STATUS=NotReady` because the CNI is not running

A typical secondary signature appears in the infrastructure provider reconciler logs on the `global` cluster:

```
ERROR  failed to reconcile <CNI-AppRelease-object>: ServiceAccount "sentr
y" not found
ERROR  failed to reconcile <Infra-Cluster-object>: ServiceAccount "sentr
y" not found
```

`<CNI-AppRelease-object>` and `<Infra-Cluster-object>` are placeholders for the actual object types your provider uses. For example, the CNI AppRelease object is typically named after the CNI being deployed, and the infrastructure cluster object name matches your provider's CRD (your provider's own `*Cluster` CRD).

The reconciler tries to apply the CNI through an `AppRelease` resource that is executed by the workload cluster's own `sentry` ServiceAccount. If that ServiceAccount is not present, the CNI is never deployed and the workload nodes stay `NotReady`.

Invariants of a Healthy Imported Cluster

Once a workload cluster reaches a healthy steady state, all of the following must hold. Verify each invariant from the `global` cluster context unless noted otherwise.

1. A `clusters.platform.tkestack.io/<cluster-name>` object exists.

```
kubectl get clusters.platform.tkestack.io <cluster-name>
```

2. The Cluster API `Cluster` resource carries the label `capi.cpaas.io/imported: ""`. The label value is the empty string `""` on a healthy import, so a `jsonpath` query alone cannot distinguish "label missing" from "label value is empty". Use `jq` to check label presence explicitly:

```
kubectl get cluster <cluster-name> -n <namespace> -o json | \
  jq -r 'if (.metadata.labels | has("capi.cpaas.io/imported"))
         then "present (value=\"\(.metadata.labels["capi.cpaas.io/impor
ted"])\")"
         else "missing"
         end'
```

   Expected output: `present (value="")`. `missing` means the import has not completed.

3. Exactly one `clustercredentials.platform.tkestack.io` object is labelled for the cluster.

```
kubectl get clustercredentials.platform.tkestack.io -o json | \
  jq -r --arg c <cluster-name> \
    '.items[] | select(.metadata.labels["cluster.x-k8s.io/cluster-nam
e"]==$c) | .metadata.name'
```

   Exactly one name should print.

4. Inside the workload cluster, ServiceAccount `sentry` exists in `cpaas-system`.

```
kubectl --kubeconfig <workload-kubeconfig> -n cpaas-system get servicea
ccount sentry
```

If any invariant fails, the workload cluster has not been imported by the `global` cluster's import controller, and the symptom pattern above is expected.

Diagnostic Steps

Step 1 — Confirm the Cluster API surface is healthy

```
kubectl get cluster <cluster-name> -n <namespace> -o wide
kubectl get kubeadmcontrolplane -n <namespace>
kubectl get machinedeployment -n <namespace>
```

A failure here is unrelated to the import pattern; investigate the infrastructure provider directly.

Step 2 — Check Machine and Node conditions

```
kubectl get machine -n <namespace> \
  -l cluster.x-k8s.io/cluster-name=<cluster-name> \
  -o jsonpath='{range .items[*]}{.metadata.name}{" phase="}{.status.phas
e}{" "}{range .status.conditions[*]}{.type}={.status} {end}{"\n"}{end}'
```

If `phase=Running` but the `NodeHealthy` condition is `False`, the underlying workload node is `NotReady`. Continue to Step 3.

Step 3 — Inspect the infrastructure provider reconciler logs

```
kubectl logs -n cpaas-system <infra-provider-manager-pod> --tail=200 | \
  grep -E 'ServiceAccount|sentry|<cluster-name>'
```

If you see repeated `ServiceAccount "sentry" not found` errors paired with a failure to reconcile the CNI AppRelease object, the workload cluster's CNI is not being deployed because the import flow has not completed. Continue to Step 4.

Step 4 — Verify the global cluster has imported the workload cluster

Run the invariant checks from the previous section. The combined diagnostic signature for a failed import is:

- `clusters.platform.tkestack.io/<cluster-name>` returns `NotFound`.
- `Cluster.metadata.labels` does not include `capi.cpaas.io/imported`.

Step 5 — Verify the ClusterCredential count invariant

```
kubectl get clustercredentials.platform.tkestack.io -o json | \
  jq -r --arg c <cluster-name> \
    '[.items[] | select(.metadata.labels["cluster.x-k8s.io/cluster-name"]
==$c)] | length'
```

A healthy cluster prints `1`. A value greater than `1` (for example, `8`) indicates that the import flow tried and failed repeatedly, leaving orphan credentials behind.

Pattern That Indicates a Failed Import

The diagnostics indicate a failed import when **all** of the following are true at the same time:

- The Cluster API surface is healthy (`Cluster.status.phase=Provisioned`, `KubeadmControlPlane.status.ready=true`).
- `Machine.condition.NodeHealthy=False` and at least one workload node is `NotReady`.
- Infrastructure provider logs repeatedly report `ServiceAccount "sentry" not found`.
- `clusters.platform.tkestack.io/<cluster-name>` does not exist.
- The Cluster resource does not carry the `capi.cpaas.io/imported` label.
- More than one `clustercredentials.platform.tkestack.io` object is labelled for the cluster.

When this pattern holds, the workload cluster's initial import on the `global` cluster failed and the import controller does not currently retry from scratch.

What Does Not Resolve This Pattern

WARNING

Restarting the cluster-transformer pod on the `global` cluster does **not** re-import a workload cluster whose initial import already failed in this pattern. After a manual pod restart no log entries reference the affected cluster and the missing `clusters.platform.tkestack.io` entry is not created.

Do not assume the controller will eventually recover on its own when more than one orphan `ClusterCredential` is present for the same cluster.

Next Step

Engage platform support with the following information:

- The cluster name and namespace.
- The output of the diagnostic commands in Steps 1 through 5.
- The names and creation timestamps of every `clustercredentials.platform.tkestack.io` object that shares the same `cluster.x-k8s.io/cluster-name` label as the affected cluster.

Recovery currently requires deleting and recreating the workload cluster after cleaning up the orphan `ClusterCredential` entries. Do not perform this on a production cluster without platform-support guidance: the recreate path triggers deletion of the underlying virtual machines on the IaaS platform, and provider-specific details (for example, DCS IP-pool and hostname planning, or HCS `HCSMachineConfigPool` re-allocation) must be reviewed before the recreate is applied.

See Also

- Troubleshoot a Cluster Stuck in Deleting — covers ACP-built clusters that stall in `Deleting` rather than failing to import.
- Troubleshoot a DCSMachine Stuck in Deleting — node-level deletion stalls; different scope.

Troubleshoot a Cluster Stuck in Deleting

TOC

Scope
Symptoms
Diagnosis
Why This Happens
Workaround
Why Not Just Wait?
Permanent Fix
See Also

Scope

This page covers the case where a workload cluster that was **built by ACP** is being deleted, every child resource (provider-specific `Machine` CRs and the IaaS VMs they represent) is already gone, but the top-level `Cluster` CR remains in `Phase: Deleting` indefinitely.

This page applies to any ACP-built workload cluster, regardless of the underlying IaaS:

- Huawei DCS (DCS Provider)
- Huawei Cloud Stack (HCS Provider)

- VMware vSphere (vSphere Provider)
- Bare Metal (Bare Metal Provider)
- Clusters created through the SSH UPI path managed by `cluster-manager`

Direct observation evidence comes from DCS; the other providers share the same `global`-cluster control-plane path.

Not in scope:

- A provider-specific `Machine` CR (such as `DCSMachine`) stuck in `Deleting` while the IaaS VM still exists — see Troubleshoot a DCSMachine Stuck in Deleting.
- A workload cluster that was **imported** (onboarded as a Third-party Cluster) and never reached the ready state — see Troubleshoot a Workload Cluster Stuck in Provisioned.

Symptoms

The Cluster CR shows the pattern below.

Where to look	What you see
`Cluster.status.phase` on the `global` cluster	`Deleting`, persisting far longer than expected — typically from one hour to many hours.
`Cluster.metadata.deletionTimestamp`	Set.
`Cluster.metadata.finalizers`	Still contains `capi.cpaas.io/imported`.
`Cluster.spec.infrastructureRef` and `Cluster.spec.controlPlaneRef`	Both are set (for example `infrastructureRef.kind: DCSCluster`, `controlPlaneRef.kind: KubeadmControlPlane`). This marks an ACP-built cluster — a Third-party Cluster has neither reference.

Where to look	What you see
Provider-specific `Machine` CRs in the same namespace (`DCSMachine` / `HCSMachine` / `VSphereMachine` / etc.), plus the matching infrastructure `Cluster` , `KubeadmControlPlane` , `MachineDeployment` , and `Machine` resources	None remain — cascade deletion has already completed.
The IaaS platform itself (DCS portal, vCenter, HCS portal, bare-metal inventory)	None of the cluster's VMs remain.
Provider controller logs (for example `cluster-api-provider-dcs-manager`)	Show successful machine deletion entries such as `removed DCSMachine finalizer` .

The `capi.cpaas.io/imported` finalizer (and a `capi.cpaas.io/alauda-cluster: imported` label, if present) does **not** by itself mean the cluster was imported — ACP-built clusters carry these too. Use the `infrastructureRef` / `controlPlaneRef` row above to tell the two apart.

Diagnosis

Run the two checks below on the `global` cluster to confirm the pattern.

Confirm the `capi.cpaas.io/imported` finalizer is still present:

```
kubectl get cluster <name> -n <ns> -o yaml | grep -A 2 finalizers
```

Confirm the cluster is ACP-built. An ACP-built cluster has both `spec.infrastructureRef` and `spec.controlPlaneRef` set; a Third-party Cluster (onboarded by import) has neither:

```
kubectl get cluster <name> -n <ns> \
  -o jsonpath='infrastructureRef={.spec.infrastructureRef.kind} controlPl
aneRef={.spec.controlPlaneRef.kind}{"\n"}'
```

If both values are non-empty (for example `infrastructureRef=DCSCluster controlPlaneRef=KubeadmControlPlane`), the cluster is ACP-built and the workaround on this page applies. If both are empty, the cluster was onboarded as a Third-party Cluster; this page does not apply and the workaround below must not be used.

Why This Happens

The `capi.cpaas.io/imported` finalizer on a `Cluster` CR is managed by a platform controller on the `global` cluster. For a cluster onboarded as a Third-party Cluster, that finalizer is released as part of the platform-side cleanup when the cluster is removed.

An ACP-built cluster carries the same finalizer. When an ACP-built cluster is deleted, its child resources and IaaS VMs are removed correctly, but the `capi.cpaas.io/imported` finalizer on the top-level `Cluster` CR is not released — so the CR stays in `Deleting`.

This is a platform-level behavior, not a fault in any IaaS provider, and it does not affect imported (Third-party) clusters. The exact handling is under investigation by the platform team.

Workaround

Before clearing the finalizer, verify that nothing depends on the `Cluster` CR's continued presence — every child resource and every IaaS VM must already be gone.

Step 1 — Confirm all provider-specific `Machine` CRs in the namespace are gone:

```
# Replace dcsmachines with the resource for your provider (hcsmachines, v
spheremachines, ...).
kubectl get dcsmachines,hcsmachines,vspheremachines -n <ns>

# Also confirm the higher-level CAPI resources are gone:
kubectl get machines,machinedeployments,kubeadmcontrolplanes,dcsclusters,
hcsclusters,vsphereclusters -n <ns>
```

Each command must return no resources for this cluster.

Step 2 — Confirm the cluster's VMs are gone on the IaaS platform. Use the provider-specific verification method:

- DCS: list VMs through the DCS portal or the DCS API and verify that none of the cluster's VMs remain. See Create a Cluster on Huawei DCS for the relevant cluster fields.
- HCS: list VMs in the HCS portal and verify that none remain.
- vSphere: list VMs in vCenter under the cluster's folder and verify that none remain.
- Bare-metal: confirm machine deprovisioning has completed in the bare-metal inventory.

Step 3 — Only after Step 1 and Step 2 both confirm zero remaining resources, clear the Cluster CR's finalizers:

```
kubectl patch cluster <name> -n <ns> --type=merge \
  -p '{"metadata":{"finalizers":null}}'
```

The Cluster CR disappears immediately. Because every child resource and every IaaS VM was already gone before this step, removing the top-level finalizer cannot leave behind an orphan — it only clears a marker that no longer has any reconciler willing to release it.

Why Not Just Wait?

The finalizer is not released on a timer, and for an ACP-built cluster it is not released by the normal platform cleanup path. Waiting does not change the state — observations of nine hours or more without progress have been recorded. Apply the workaround once Step 1 and Step 2 are confirmed.

Permanent Fix

The fix belongs in the `global`-cluster control plane: the `capi.cpaas.io/imported` finalizer must be released for an ACP-built cluster once its child resources are gone. This is tracked outside this documentation set; no release date is committed here. Until the fix lands, the workaround above is the supported recovery path.

See Also

- Troubleshoot a DCSMachine Stuck in Deleting — node-level deletion stalls; different scope.
- Troubleshoot a Workload Cluster Stuck in Provisioned — covers true Third-party Cluster bring-up issues; different scope.

Create a Cluster on Non-Alauda OS

TOC

- Prerequisites
  - Node Requirements
  - Load Balancing
  - Connecting `global` Cluster and Workload Cluster
  - Image Registry
  - Container Networking
- Creation Procedure
  - Basic Info
  - Container Network
  - Node Settings
  - Extended Parameters
- Post-Creation Steps
  - Viewing Creation Progress
  - Associating with Projects

Prerequisites

Node Requirements

1. If you downloaded a single-architecture installation package from Download Installation Package, ensure your node machines have the same architecture as the package. Otherwise, nodes won't start due to missing architecture-specific images.
2. Verify that your node operating system and kernel are supported. See Supported OS and Kernels for details.
3. Perform availability checks on node machines. For specific check items, see Node Preprocessing > Node Checks.
4. If node machine IPs cannot be directly accessed via SSH, provide a SOCKS5 proxy for the nodes. The `global` cluster will access nodes through this proxy service.

Load Balancing

For production environments, a load balancer is required for cluster control plane nodes to ensure high availability.

If a hardware LoadBalancer is available in your environment, it is **recommended** to use it. If not, you can enable `Self-built VIP`, which provides software load balancing using keepalived.

Recommendation: When using a hardware LoadBalancer, configuring the Cluster Endpoint with a domain name is recommended. If the hardware LoadBalancer VIP changes after cluster installation, you can update the DNS record instead of reconfiguring the cluster.

Note: `Self-built VIP` does not support domain name configuration.

The load balancer must correctly forward traffic to ports `6443` and `11780` on all control plane nodes in the cluster.

If your cluster has only one control plane node and you use that node's IP as the Cluster Endpoint parameter, the cluster cannot be scaled from a single node to a highly available multi-node setup later. Therefore, we recommend providing a load balancer even for single-node clusters.

When enabling `Self-built VIP`, you need to prepare:

1. An available VRID
2. A host network that supports the VRRP protocol

3. All control plane nodes and the VIP must be on the same subnet, and the VIP must be different from any node IP.

Connecting `global` Cluster and Workload Cluster

The platform requires mutual access between the `global` cluster and workload clusters. If they're not on the same network, you need to:

1. Provide `External Access` for the workload cluster to ensure the `global` cluster can access it. Network requirements must ensure `global` can access ports `6443` and `11780` on all control plane nodes.
2. Add an additional address to `global` that the workload cluster can access. When creating a workload cluster, add this address to the cluster's annotations with the key `cpaas.io/platform-url` and the value set to the public access address of `global`.

Image Registry

Cluster images support Platform Built-in, Private Repository, and Public Repository options. These choices configure the platform image source used by the workload cluster.

- **Platform Built-in**: Uses the platform image source configured for the `global` cluster. This source can be the Registry deployed with `global` or the external platform image registry selected during installation. If the source is the built-in Registry and the workload cluster cannot access `global`, see Add External Address for Built-in Registry.
- **Private Repository**: Uses a different customer-managed image registry for this workload cluster. Before selecting it, upload the complete payload required by the selected Kubernetes version and architecture, and meet the required capabilities and access conditions described in Prepare an External Platform Image Registry. For production use, also follow the production recommendations for HTTPS, DNS, availability, monitoring, backup, recovery, and retention.
- **Public Repository**: Uses the platform's public image registry. Before using, complete Updating Public Repository Credentials.

Container Networking

If you plan to use Kube-OVN's Underlay for your cluster, prepare the Kube-OVN Underlay physical network.

Creation Procedure

1. Enter the **Administrator** view, and click **Clusters/Clusters** in the left navigation bar.
2. Click **Create Cluster**.
3. Configure the following sections according to the instructions below: Basic Info, Container Network, Node Settings, and Extended Parameters.

Basic Info

Parameter	Description
Kubernetes Version	All optional versions are rigorously tested for stability and compatibility. **Recommendation:** Choose the latest version for optimal features and support.
Container Runtime	Containerd is provided as the default container runtime.
Cluster Network Protocol	Supports three modes: IPv4 single stack, IPv6 single stack, IPv4/IPv6 dual stack. **Note:** If you select dual stack mode, ensure all nodes have correctly configured IPv6 addresses; the network protocol cannot be changed after setting.

	`IP Address / Domain`: Enter the pre-prepared domain name or VIP if no domain name is available.
Cluster Endpoint	`Self-built VIP`: Disabled by default. Only enable if you haven't provided a LoadBalancer. When enabled, the installer will automatically deploy `keepalived` for software load balancing support.
	`External Access`: Enter the externally accessible address prepared for the cluster when it's not in the same network environment as the `global` cluster.

Container Network

Kube-OVN

Kube-OVN is a cloud native Kubernetes container network orchestration system developed by Alauda. It provides Kubernetes networking capabilities for cross-cloud network management, traditional network architecture integration, infrastructure interconnection, and edge cluster deployment scenarios.

Parameter	Description
Subnet	Also known as Cluster CIDR, represents the **default subnet** segment. After cluster creation, additional subnets can be added.
Transmit Mode	**Overlay**: A virtual network abstracted over the infrastructure that doesn't consume physical network resources. When creating an Overlay default subnet, all Overlay subnets in the cluster use the same cluster NIC and node NIC configuration.

	Underlay: This transmission method relies on physical network devices. It can directly allocate physical network addresses to Pods, ensuring better performance and connectivity with the physical network. Nodes in an Underlay subnet must have multiple NICs, and the NIC used for bridge networking must be exclusively used by Underlay and not carry other traffic like SSH. When creating an Underlay default subnet, the cluster NIC is actually a default NIC for bridge networking, and the node NIC is the node NIC configuration in the bridge network. • **Default Gateway**: The physical network gateway address, which is the gateway address for the Cluster CIDR segment (must be within the Cluster CIDR address range). • **VLAN ID**: Virtual LAN identifier (VLAN number), e.g., `0`. • **Reserved IPs**: Set reserved IPs that won't be automatically allocated, such as IPs in the subnet that are already used by other devices.
Service CIDR	IP address range used by Kubernetes Services of type ClusterIP. Cannot overlap with the default subnet range.
Join CIDR	In Overlay transmission mode, this is the IP address range used for communication between nodes and pods. Cannot overlap with the default subnet or Service CIDR.

Custom

If you need to install other network plugins, select **Custom** mode. You can manually install network plugins after the cluster is successfully created.

Parameter	Description

Cluster CIDR	IP address range used by pods created when the cluster starts.
Service CIDR	IP address range used by Kubernetes Services of type ClusterIP. Cannot overlap with the container subnet range.

Node Settings

Parameter	Description
Network Interface Card	The name of the host network interface device used by the cluster network plugin. **Note**: • When selecting Underlay transmission mode for the Kube-OVN default subnet, you must specify the network interface name, which will be the default NIC for bridge networking. - The platform's network interface traffic monitoring by default recognizes traffic on interfaces named like `eth.|en.|wl.|ww.`. If you use interfaces with different naming conventions, see Collect Network Data from Custom-Named Network Interfaces after cluster onboarding to modify the relevant resources and ensure the platform can properly monitor network interface traffic.
Node Name	You can choose to use either the node IP or hostname as the node name on the platform.

	Note: When choosing to use hostname as the node name, ensure that the hostnames of nodes added to the cluster are unique.
Nodes	**Add nodes** to the cluster, or **Recovery from draft** temporarily saved node information. See the detailed parameter descriptions for adding nodes below.
Monitoring Type	Supports **Prometheus** and **VictoriaMetrics**. When selecting **VictoriaMetrics** as the monitoring component, you must configure the **Deploy Type**: **- Deploy VictoriaMetrics**: Deploys all related components, including **VMStorage**, **VMAlert**, **VMAgent**, etc. **- Deploy VictoriaMetrics Agent**: Only deploys the log collection component, **VMAgent**. When using this deployment method, you need to associate with a VictoriaMetrics instance already deployed on another cluster in the platform to provide monitoring services for the cluster.
Monitoring Nodes	Select nodes for deploying cluster monitoring components. Supports selecting worker nodes and control plane nodes that allow application deployment. To avoid affecting cluster performance, it's recommended to prioritize worker nodes. After the cluster is successfully created, monitoring components with storage type **Local Volume** will be deployed on the selected nodes.

Node Addition Parameters

Parameter	Description

Type	**Control Plane Node**: Responsible for running components such as kube-apiserver, kube-scheduler, kube-controller-manager, etcd, container network, and some platform management components in the cluster. When **Application Deployable** is enabled, control plane nodes can also be used as worker nodes. **Worker Node**: Responsible for hosting business pods running on the cluster.
IPv4 Address	The IPv4 address of the node. For clusters created in internal network mode, enter the node's **private IP**.
IPv6 Address	Valid when the cluster has IPv4/IPv6 dual stack enabled. The IPv6 address of the node.
Application Deployable	Valid when **Node Type** is **Control Plane Node**. Whether to allow business applications to be deployed on this control plane node, scheduling business-related pods to this node.
Display Name	The display name of the node.
SSH Connection IP	The IP address that can connect to the node when accessing it via SSH service. If you can log in to the node using `ssh <username>@<node's IPv4 address>`, this parameter is not required; otherwise, enter the node's public IP or NAT external IP to ensure the `global` cluster and proxy can connect to the node via this IP.

Network Interface Card	Enter the name of the network interface used by the node. The priority of network interface configuration effectiveness is as follows (from left to right, in descending order): **Kube-OVN Underlay**: Node NIC > Cluster NIC **Kube-OVN Overlay**: Node NIC > Cluster NIC > NIC corresponding to the node's default route **Calico**: Cluster NIC > NIC corresponding to the node's default route **Flannel**: Cluster NIC > NIC corresponding to the node's default route
Associated Bridge Network	**Note**: When creating a cluster, bridge network configuration is not supported; this option is only available when **adding nodes** to a cluster that already has Underlay subnets created. Select an existing Add Bridge Network. If you don't want to use the bridge network's default NIC, you can configure the node NIC separately.
SSH Port	SSH service port number, e.g., `22`.
SSH Username	SSH username, needs to be a user with root privileges, e.g., `root`.

Proxy	Whether to access the node's SSH port through a proxy. When the `global` cluster cannot directly access the node to be added via SSH (e.g., the `global` cluster and workload cluster are not in the same subnet; the node IP is an internal IP that the `global` cluster cannot directly access), this switch needs to be turned on and proxy-related parameters configured. After configuring the proxy, node access and deployment can be achieved through the proxy. **Note**: Currently, only SOCKS5 proxy is supported. **Access URL**: Proxy server address, e.g., `192.168.1.1:1080`. **Username**: Username for accessing the proxy server. **Password**: Password for accessing the proxy server.
SSH Authentication	Authentication method and corresponding authentication information for logging into the added node. Options include: **Password**: Requires a username with root privileges and the corresponding **SSH password**. **Key**: Requires a **private key** with root privileges and the **private key password**.
Save Draft	Saves the currently configured data in the dialog as a draft and closes the **Add Node** dialog.

Without leaving the **Create Cluster** page, you can select **Restore from draft** to open the **Add Node** dialog and restore the configuration data saved as a draft.

Note: The data restored from the draft is the most recently saved draft data.

Extended Parameters

Note:

- Apart from required configurations, it's not recommended to set extended parameters, as incorrect settings may make the cluster unavailable and cannot be modified after cluster creation.
- If a entered **Key** duplicates a default parameter **Key**, it will override the default configuration.

Procedure

1. Click **Extended Parameters** to expand the extended parameter configuration area. You can optionally set the following extended parameters for the cluster:

Parameter	Description
Kubelet Parameters	`kubeletExtraArgs`, additional configuration parameters for Kubelet. **Note**: When the **Container Network**'s **Node IP Count** parameter is entered, a default **Kubelet Parameter** configuration with the key `max-pods` and a value of **Node IP Count** is automatically generated. This sets the maximum number of pods that can run on any node in the cluster. This configuration is not displayed in the interface.

	Adding a new `max-pods: maximum number of runnable pods` key-value pair in the **Kubelet Parameters** area will override the default value. Any positive integer is allowed, but it's recommended to use the default value (Node IP Count) or enter a value not exceeding `256`.
Controller Manager Parameters	`controllerManagerExtraArgs`, additional configuration parameters for the Controller Manager.
Scheduler Parameters	`schedulerExtraArgs`, additional configuration parameters for the Scheduler.
APIServer Parameters	`apiServerExtraArgs`, additional configuration parameters for the APIServer.
APIServer URL	`publicAlternativeNames`, APIServer access addresses issued in the certificate. Only IPs or domain names can be entered, with a maximum of 253 characters.
Cluster Annotations	Cluster annotation information, marking cluster characteristics in metadata in the form of key-value pairs for platform components or business components to obtain relevant information.

4. Click **Create**. You'll return to the cluster list page where the cluster will be in the **Creating** state.

Post-Creation Steps

Viewing Creation Progress

On the cluster list page, you can view the list of created clusters. For clusters in the **Creating** state, you can check the execution progress.

Procedure

1. Click the small icon **View Execution Progress** to the right of the cluster status.
2. In the execution progress dialog that appears, you can view the cluster's execution progress (status.conditions).
   Tip: When a certain type is in progress or in a failed state with a reason, hover your cursor over the corresponding reason (shown in blue text) to view detailed information about the reason (status.conditions.reason).

Associating with Projects

After the cluster is created, you can add it to projects in the project management view.

Managed Clusters

Managed Clusters

Managed Clusters

Import Third-Party Clusters

Import Third-Party Clusters

Import Standard Kubernetes Cl

Import Ope

Import GKE Cluster

Import Huawei Cloud CCE Cluster (Public Clo

Import an existing CCE cluster as a third-party cluster.

Import Azure AKS Cluster

Import Aliba

Import Tencent Cloud TKE Cluster

Register Cluster

Register Cluster

Public Cloud Cluster Initialization

Network Initialization

Public cloud cluster network initialization.

Storage Initialization

Public cloud cluster storage initialization.

How to

Network Configuration for Impc

Fetch import cluster informatio

Trust an ins

Collect Network Data from Cust

Collect network data from custom named r

Configure audit collection for imported standa

Enable Kubernetes audit logging on imported standard Kuberr collect audit data.

Third-Party Cluster Onboarding

Third-party clusters are existing Kubernetes environments whose Kubernetes distribution, node lifecycle, or provider infrastructure are managed outside Alauda. In the console, these environments appear under **Managed Clusters**.

ACP can provide centralized governance, resource visibility, Extension installation, application operations, and observability integration for third-party clusters when connectivity, credentials, component prerequisites, provider requirements, and Extension compatibility are satisfied. Onboarding does not make ACP the owner of the external cluster's Kubernetes lifecycle, node lifecycle, or provider infrastructure lifecycle.

TOC

Choose An Onboarding Method

Choose An Onboarding Method

Method	Connection model	Use when
Import cluster	The `global` cluster connects to the target cluster API server with supplied address, CA, and credentials.	The platform can reach the target API server, and the administrator can provide the required cluster information.

Method	Connection model	Use when
Register cluster	A reverse proxy service in the target cluster initiates registration and establishes a tunnel to the platform.	The target cluster should initiate the connection, or direct access from the `global` cluster to the target API server is restricted.

After onboarding, high-level day-2 operations are treated consistently in the cluster list. Provider-specific limitations can still apply to node operations, audit data, control-plane metrics, certificates, ingress, storage, and Extension compatibility.

For import procedures, see Import Third-Party Clusters. For reverse-connect onboarding, see Register Cluster.

Import Third-Party Clusters

Import Third-Party Clusters

Import Standard Kubernetes Cl

Import Ope

Import GKE Cluster

Import Huawei Cloud CCE Cluster (Public Clo

Import an existing CCE cluster as a third-party cluster.

Import Azure AKS Cluster

Import Aliba

Import Tencent Cloud TKE Cluster

Import Third-Party Clusters

Use import when the `global` cluster can reach the target cluster API server and the administrator can provide the required cluster address, CA, and credentials.

Target environment	Start with
Existing standard Kubernetes environment	Import Standard Kubernetes Cluster
External Kubernetes distribution	Import OpenShift Cluster
Public cloud Kubernetes service	Import Amazon EKS Cluster, Import GKE Cluster, Import Azure AKS Cluster, Import Alibaba Cloud ACK Cluster, Import Tencent Cloud TKE Cluster, or Import Huawei Cloud CCE Cluster

Provider-specific procedures include provider console or CLI steps where required. The provider name identifies the procedure path; it is not a separate ACP cluster model.

Import Standard Kubernetes Cluster

Use this procedure to import an existing standard Kubernetes cluster, such as a cluster deployed with **kubeadm**, as a third-party cluster.

TOC

Terminology
Prerequisites
Notes
Obtain Registry Address
Check if Extra Registry Config is Needed
Get Cluster Info
Integrate Cluster
Network Configuration
Post-import Configuration
FAQ
Why is the "Add Node" button disabled?
Which certificates are supported?
Which features are unsupported?
How to fix Containerd runtime causing distributed storage deployment failures?

Terminology

Term	Description
Provider-managed Kubernetes service	A Kubernetes service where the provider owns the control plane nodes and control plane components. Users usually cannot log in to or directly maintain the control plane nodes.
Self-managed Kubernetes cluster	A Kubernetes cluster where the cluster owner maintains the control plane nodes and cluster components.

Prerequisites

- Kubernetes and related components in the cluster must meet the version and parameter requirements.
- If the runtime is Containerd, update the Containerd configuration before integration to ensure distributed storage can be deployed successfully.

Notes

By default, the platform monitors NIC traffic matching `eth.*|en.*|wl.*|ww.*`. If your NIC uses a different naming convention, update the configuration after integration following Collect Network Data from Custom Named Network Cards.

Obtain Registry Address

- To use the registry deployed by the platform during **global cluster** installation, run the following on a global control plane node:

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.preferPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.platformURL}' | awk -F // '{print $NF}')
fi
echo "Registry address: $REGISTRY"
```

- To use an **external registry**, set **REGISTRY** manually:

```
REGISTRY=<external-registry-address>  # e.g., registry.example.cn:60080 or 192.168.134.43
echo "Registry address: $REGISTRY"
```

Check if Extra Registry Config is Needed

1. Run the following to check if the registry supports HTTPS with a trusted CA certificate:

```
REGISTRY=<registry-address-from-previous-step>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Pass: Registry uses a trusted CA certificate. No extra config needed.'
else
    echo 'Fail: Registry does not support HTTPS or uses an untrusted certificate. Follow "Trust Insecure Registry".'
fi
```

2. If check fails, see How to trust an insecure registry?

Get Cluster Info

See How to fetch cluster information?.

Integrate Cluster

1. In the left navigation, go to **Cluster Management** > **Clusters**.
2. Click **Import Cluster**.
3. Configure parameters as below:

Parameter	Description
Registry	Registry storing required platform component images. Options: **Platform Default** (configured during global setup), **Private Registry** (requires address, port, username, password), **Public Registry** (requires cloud credential update).
Cluster Info	Can be entered manually or parsed from a KubeConfig file. Required fields: **Cluster Address**, **CA Certificate** (Base64 decoded if entered manually), and **Authentication** (token or client certificate with cluster-admin rights).

4. Click **Check Connectivity**. The platform verifies network access and auto-detects cluster type.
5. If successful, click **Import** to complete.
   *Progress can be viewed via the **execution progress** dialog (status.conditions). Once integrated, the cluster appears as healthy in the list.*

Network Configuration

Ensure connectivity between the global cluster and the imported cluster.

Post-import Configuration

If you need the platform to collect audit data from an imported standard Kubernetes cluster, configure Kubernetes API server audit logging on the cluster after import. See How to configure audit collection for imported standard Kubernetes clusters?.

FAQ

Why is the "Add Node" button disabled?

Adding nodes through the platform UI is not supported for imported standard Kubernetes clusters. Add nodes directly in the target cluster or through the cluster provider.

Which certificates are supported?

1. **Kubernetes Certificates**: Only API Server certificates can be viewed; other certificates are unsupported and will not auto-rotate.
2. **Platform Component Certificates**: Viewable and auto-rotatable.

Which features are unsupported?

- **Provider-managed Kubernetes services**: Audit logs are not available.
- **Provider-managed Kubernetes services**: ETCD, Scheduler, and Controller Manager monitoring are not supported. Only API Server metrics are available.
- **All clusters**: Certificates other than API Server are not supported.

How to fix Containerd runtime causing distributed storage deployment failures?

When using Containerd, distributed storage deployment fails unless you adjust Containerd settings on **all nodes**:

1. Edit `/etc/systemd/system/containerd.service`, set `LimitNOFILE=1048576`.
2. Run `systemctl daemon-reload`.
3. Restart Containerd: `systemctl restart containerd`.
4. On control plane nodes, restart distributed storage pods:

```
kubectl delete pod --all -n rook-ceph
```

Import OpenShift Cluster

Use this provider-specific procedure to import an existing OpenShift cluster as a third-party cluster. The external distribution owner remains responsible for the Kubernetes distribution, node lifecycle, and provider infrastructure lifecycle unless a documented provider-specific capability provides that lifecycle.

TOC

- Prerequisites
- Obtain Registry Address
- Check if Extra Registry Config is Needed
  - Trust Insecure Registry
- Configure DNS for the Cluster
- Get Cluster Info
  - Method 1 (Recommended): Get the KubeConfig File
  - Method 2: Use Token, API Server Address, and CA Certificate
- Import Cluster
- Network Configuration
- Deploy Add-ons
- Update Audit Policy
- FAQ
  - Why is the "Add Node" button disabled?
  - Which certificates are supported?
  - Limitations for this provider-specific import path

Prerequisites

- The Kubernetes version and parameters of the cluster must meet the Standard Kubernetes Cluster Requirements.
- During integration, `kubectl` commands are required. Install the CLI tool on the bastion host that can access the cluster.
- To enable real-time monitoring of metrics such as nodes, workloads (Deployment, StatefulSet, DaemonSet), Pods, and containers, ensure **Prometheus** is already deployed in the target cluster.

Obtain Registry Address

- To use the registry deployed by the platform during **global cluster** installation, run the following command on a global control plane node:

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.pr
eferPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath
='{.data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath
='{.data.platformURL}' | awk -F // '{print $NF}')
fi
echo "Registry address is: $REGISTRY"
```

- To use an **external registry**, manually set the **REGISTRY** variable:

```
REGISTRY=<external-registry-address>  # e.g., registry.example.cn:60080
or 192.168.134.43
echo "Registry address is: $REGISTRY"
```

Check if Extra Registry Config is Needed

1. Run the following command to check if the registry supports HTTPS and uses a trusted CA certificate:

```
REGISTRY=<registry-address-from-previous-step>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Pass: Registry uses a trusted CA certificate. No extra config needed.'
else
    echo 'Fail: Registry does not support HTTPS or uses an untrusted certificate. Follow "Trust Insecure Registry".'
fi
```

2. If the check fails, follow the steps below.

Trust Insecure Registry

1. **Log in to all OCP cluster nodes.**
2. On each node, configure the registry settings:

```
sudo -i
sudo chattr -i /

sudo mkdir -p /etc/systemd/system/crio.service.d/
cat | sudo tee /etc/systemd/system/crio.service.d/99-registry-cpaas-system.conf << 'EOF'
[Service]
ExecStart=
ExecStart=/usr/bin/crio \
          --insecure-registry='<registry-address>' \  # e.g., registry.example.cn:60080 or 192.168.134.43
          $CRIO_CONFIG_OPTIONS \
          $CRIO_RUNTIME_OPTIONS \
          $CRIO_STORAGE_OPTIONS \
          $CRIO_NETWORK_OPTIONS \
          $CRIO_METRICS_OPTIONS
EOF
```

3. Restart `crio`:

```
sudo systemctl daemon-reload && sudo systemctl restart crio
```

Configure DNS for the Cluster

Modify the CoreDNS `ConfigMap` in the global cluster to configure DNS.

1. From the bastion host, get the OCP cluster base domain:

   ```
   oc get dns cluster -o jsonpath='{.spec.baseDomain}'
   ```

   Example output:

   ```
   ocp.example.com
   ```

2. Log in to the platform management console, switch to the **global** cluster, then go to **Cluster Management** > **Resource Management**.

3. Edit the `cpaas-coredns` `ConfigMap` in the `kube-system` namespace.
   Add a new block using the OCP base domain and DNS server address (from `/etc/resolv.conf` on a cluster node).
   Example:

   ```
   Corefile: |
   ocp.example.com:1053 {
       log
       forward . 192.168.31.220
   }
   .:1053 {
       log
       forward . 192.168.31.220
   }
   ```

Get Cluster Info

Choose one of the following:

Method 1 (Recommended): Get the KubeConfig File

1. On the bastion host, search for the `kubeconfig` file and verify it contains an admin context.
2. Copy the kubeconfig file from the bastion host to your local machine:

```
scp root@<bastion-ip>:</path/to/kubeconfig> <local-path>
```

Method 2: Use Token, API Server Address, and CA Certificate

See How to fetch cluster information?.

Import Cluster

1. In the left navigation, go to **Cluster Management** > **Clusters**.
2. Click **Import Cluster**.
3. Configure the parameters:

Parameter	Description
Registry	Registry storing platform component images.**Platform Default**: registry configured during global setup.**Private Registry**: requires registry address, port, username, and password.**Public Registry**: requires updating cloud credentials.
Cluster Info	Either upload the KubeConfig file or enter manually.**Cluster Address**: API Server address.**CA Certificate**: decoded Base64 CA certificate.**Authentication**: token or client certificate with cluster-admin permissions.

4. Click **Check Connectivity**.
5. If successful, click **Import**. Progress can be viewed in the execution log. Once imported, the cluster appears healthy in the list.

Network Configuration

Ensure network connectivity between the global cluster and the imported cluster. See Network Configuration for Imported Clusters.

Deploy Add-ons

After successful integration, go to **Marketplace** to deploy required add-ons such as monitoring, log collection, and log storage.

Before deploying log collection, ensure `/var/cpaas/` *has more than 50GB free space:*

```
df -h /var/cpaas
```

Update Audit Policy

You can modify the audit policy (`spec.audit.profile`) of the cluster:

- **Default**: logs metadata of read/write requests (OAuth access token creation logs the body).
- **WriteRequestBodies**: logs metadata for all requests and bodies of write requests.
- **AllRequestBodies**: logs metadata and bodies of all requests.

Sensitive resources (e.g., Secrets, Routes, OAuthClient) only log metadata.

Update with:

```
oc edit apiserver cluster
```

FAQ

Why is the "Add Node" button disabled?

Adding nodes through the platform UI is not supported. Use the distribution provider's method.

Which certificates are supported?

1. **Kubernetes Certificates**: Only API Server certificates are visible, no auto-rotation.
2. **Platform Component Certificates**: Visible and auto-rotated.

Limitations for this provider-specific import path

- Audit data collection.
- ETCD, Scheduler, Controller Manager monitoring (only API Server metrics available).
- Certificates other than API Server.

Import Amazon EKS Cluster

Use this provider-specific procedure to import an existing Amazon EKS cluster as a third-party cluster. The external provider remains responsible for provider infrastructure and the provider-managed Kubernetes and node lifecycle unless a documented provider-specific capability provides that lifecycle.

TOC

- Prerequisites
- Prepare the environment
- Get cluster information
  - Get the import token
- Import the cluster
- Network configuration
- Next steps
  - Initialize Ingress and storage
- FAQ
  - The Add Node button is disabled after import. How can I add nodes?
  - Which certificates are supported by certificate management for imported clusters?
  - Limitations for this provider-specific import path

Prerequisites

- The cluster's Kubernetes version and settings meet the requirements in Version compatibility for importing standard Kubernetes clusters.
- The image registry must support HTTPS and provide a valid TLS certificate issued by a public CA.

Prepare the environment

To comply with AWS EKS security practices, perform the following steps in AWS CloudShell.

1. Ensure network connectivity to the AWS Management Console.
2. Search for `cloudshell`, then open CloudShell ↗.
3. Verify that the selected region matches your target cluster's region; switch if needed.
4. After CloudShell is ready, clear the terminal and run:

```
# List clusters in the current region and verify your permissions
aws eks list-clusters

# <region-code> is the region of the cluster, e.g., us-west-1
# <my-cluster> is the cluster name from the previous output
aws eks update-kubeconfig --region <region-code> --name <my-cluster>

# The kubeconfig file is saved to "${HOME}/.kube/config"
# Save its content to a file, then upload it to the platform for parsin
g
cat "${HOME}/.kube/config"
```

5. The environment is now ready. For subsequent steps such as **Get cluster information** and **Import cluster**, run any commands against the target cluster from within CloudShell.

Get cluster information

Get the import token

KubeConfig from public-cloud clusters cannot be used directly for import.

To obtain the cluster import token, see How do I get cluster information?.

Import the cluster

1. In the left navigation, go to **Cluster Management** > **Clusters**.
2. Click **Import Cluster**.
3. Configure the parameters as follows.

Parameter	Description
Image registry	Registry that stores platform component images required by the cluster. **- Platform default**: the registry configured when the global cluster was deployed. **- Private registry**: a pre-provisioned registry hosting required images. Provide the **private registry address**, **port**, **username**, and **password**. **- Public registry**: a public internet registry. Before use, obtain credentials as described in Update public registry cloud credentials.
Cluster information	**Tip**: Upload the kubeconfig file and let the platform parse it automatically. **Cluster endpoint**: the external API server address exposed by the target cluster. **CA certificate**: the cluster's CA certificate. **Authentication**: use the **token** created in the previous step with **cluster administrator** privileges.

4. Click **Check connectivity** to verify network connectivity and automatically detect the cluster type. The detected type appears as a badge in the top-right of the form.
5. After the connectivity check passes, click **Import**, then confirm.
   Tips:
   - For clusters in the **Importing** state, click the details icon to view progress in the **Execution progress** dialog (`status.conditions`).
   - After a successful import, the cluster list shows key information. The cluster status is Normal and cluster operations are available.

Network configuration

Ensure the global cluster and the imported cluster have network connectivity. See Network Configuration for Imported Clusters.

Next steps

Initialize Ingress and storage

If you need Ingress and storage capabilities, see Initialize Ingress for AWS EKS and Initialize storage for AWS EKS.

FAQ

The Add Node button is disabled after import. How can I add nodes?

Adding nodes from the platform UI is not supported. Add nodes through your cluster provider.

Which certificates are supported by certificate management for imported clusters?

1. **Kubernetes certificates**: You can view the API server certificate only. Other Kubernetes certificates are not visible and are not auto-rotated.
2. **Platform component certificates**: Visible in the platform and support automatic rotation.

Limitations for this provider-specific import path

- Audit data is not available.
- ETCD, Scheduler, and Controller Manager metrics are not supported; a subset of API server charts is available.

- Certificate details other than the Kubernetes API server certificate are not available.

Import GKE Cluster

Use this provider-specific procedure to import an existing Google GKE cluster as a third-party cluster. The external provider remains responsible for provider infrastructure and the provider-managed Kubernetes and node lifecycle unless a documented provider-specific capability provides that lifecycle.

TOC

- Prerequisites
- Preparing the Operating Environment
- Obtaining Cluster Information
  - Obtaining the API Server Address and CA Certificate of the Target Cluster
  - Obtaining the Target Cluster Token
- Importing the Cluster
- Network Configuration
- Post-Import Operations
  - Ingress and Storage Initialization
- Frequently Asked Questions
  - How to add nodes when the "Add Node" button is grayed out after importing the cluster?
  - What certificates are supported by the certificate management functionality for imported clusters?

Prerequisites

- The Kubernetes version and components on the cluster meet the version requirements for importing public cloud clusters.
- Ensure the cluster type is a standard cluster and the account has permissions to maintain the control plane. Autopilot clusters are not currently supported.
- The image repository must support HTTPS access and provide a valid TLS certificate authenticated by a public certification authority.

Preparing the Operating Environment

To comply with GKE security standards, the following steps must be performed using Cloud Shell.

1. Ensure network connectivity with Google.
2. Access the **Clusters** page↗ in the Kubernetes Engine feature; find the cluster to be imported, click on cluster details, and select the **Connect** button.
3. In the popup dialog, copy the command for configuring kubectl command-line access permissions and click the **Run in Cloud Shell** button.
4. Wait for Cloud Shell to be ready, clear the command line, paste the content copied in the previous step, and execute it.
5. The environment is now ready. All subsequent commands executed in the importing cluster environment for steps such as **Obtaining Cluster Information** and **Importing Cluster** should be executed in Cloud Shell.

Obtaining Cluster Information

Obtaining the API Server Address and CA Certificate of the Target Cluster

1. Access the **Clusters** page↗ in the Kubernetes Engine feature and click to enter the details page of the target cluster.

2. The API Server address can be found in the **External endpoints** section.
3. To obtain the CA certificate, use one of the following methods in Cloud Shell:

   Method A: Get the CA certificate from your kubeconfig:

   ```
   gcloud container clusters get-credentials <cluster-name> --zone <zone>
   kubectl config view --raw -o jsonpath='{.clusters[0].cluster.certificat
   e-authority-data}' | base64 -d
   ```

   Method B: Get the CA certificate directly from the cluster:

   ```
   gcloud container clusters describe <cluster-name> --zone <zone> --forma
   t='get(masterAuth.clusterCaCertificate)' | base64 -d
   ```

   Note: The certificate must be Base64-decoded before pasting into the import form.

Obtaining the Target Cluster Token

The KubeConfig file of public cloud clusters cannot be directly used for importing clusters.

See How to obtain cluster information? to obtain the target cluster token.

Importing the Cluster

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click **Manage Cluster** > **Import Cluster**.
3. Configure the relevant parameters according to the following instructions.

Parameter	Description
Image Repository	Repository for storing platform component images required by the cluster. **- Platform Default**: Image repository configured during global deployment. **- Private Repository**: Pre-built repository storing platform required components. Enter **Private Image Repository Address**, **Port**, **Username**, and **Password** for accessing the image repository. **- Public Repository**: Use public image repository services on the internet. Before use, update

Parameter	Description
	public repository cloud credentials to obtain repository authentication permissions.
Cluster Information	**Cluster Information**: Includes the target cluster token and the API Server address and CA certificate of the target cluster. **Cluster Address**: The access address where the target cluster exposes the API Server for platform access to the cluster's API Server. **CA Certificate**: CA certificate of the target cluster. **Note**: When manually inputting, you need to enter the Base64 decoded certificate. **Authentication Method**: Authentication method for the target cluster, requires using the **token** (Token) with **cluster management permissions** created in the previous step for authentication.

4. Click **Check Connectivity** to verify network connectivity with the target cluster and automatically identify the cluster type, which will be displayed as a badge in the top-right corner of the form.
5. After connectivity check passes, click **Import** and confirm.

> **TIP**
>
> - Click the **Details** icon on the right side of clusters in **Importing** status to view the cluster execution progress (status.conditions) in the popup **Execution Progress** dialog.
> - After successful cluster import, you can view key cluster information in the cluster list, the cluster status shows as normal, and you can perform cluster-related operations.

Network Configuration

Ensure network connectivity between the global cluster and the imported cluster. See Network Configuration for Imported Clusters.

Post-Import Operations

Ingress and Storage Initialization

After importing the cluster, if you need to use Ingress and storage-related features, see Google GKE Ingress Controller Configuration and Google GKE Storage Configuration.

Frequently Asked Questions

How to add nodes when the "Add Node" button is grayed out after importing the cluster?

Adding nodes through the platform interface is not supported. Contact the cluster provider to add nodes.

What certificates are supported by the certificate management functionality for imported clusters?

1. **Kubernetes Certificates**: All imported clusters only support viewing APIServer certificate information in the platform certificate management interface. Other Kubernetes certificates cannot be viewed and automatic rotation is not supported.
2. **Platform Component Certificates**: All imported clusters can view platform component certificate information in the platform certificate management interface and support automatic rotation.

Import Huawei Cloud CCE Cluster (Public Cloud)

Use this provider-specific procedure to import an existing Huawei Cloud CCE cluster as a third-party cluster. The external provider remains responsible for provider infrastructure and the provider-managed Kubernetes and node lifecycle unless a documented provider-specific capability provides that lifecycle.

TOC

- Prerequisites
- Obtain Image Registry Address
- Determine if Image Registry Requires Additional Configuration
- Obtain Cluster Information
  - Obtain Import Cluster Token
- Import Cluster
- Network Configuration
- Follow-up Operations
  - Ingress (Inbound Rules) and Storage Initialization
- FAQ
  - After importing the cluster, the add node button is grayed out. How to add nodes?
  - What certificates does the certificate management feature support for imported clusters?
  - Limitations for this provider-specific import path

Prerequisites

- The Kubernetes version and parameters on the cluster meet the Standard Kubernetes Cluster Component Version and Parameter Requirements.
- Ensure the cluster type is Huawei Cloud CCE cluster and the account has permissions to maintain the control plane. Turbo clusters are not currently supported.
- Huawei Cloud CCE clusters do not have the ability to access external network resources by default after creation. Before importing the cluster, ensure that the cluster to be imported can access the platform access address.

Obtain Image Registry Address

- To use the **platform-deployed** image registry from the global cluster deployment, execute the following command on the **control plane node of the global cluster** to obtain the address:

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.pr
eferPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath
='{.data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath
='{.data.platformURL}' | awk -F \// '{print $NF}')
fi
echo "Image registry address is: $REGISTRY"
```

- To use an **external image registry**, manually set the **REGISTRY** variable.

```
REGISTRY=<external image registry address>  # Valid examples: registry.
example.cn:60080 or 192.168.134.43
echo "Image registry address is: $REGISTRY"
```

Determine if Image Registry Requires Additional Configuration

1. Execute the following command to determine whether the specified image registry supports HTTPS access and uses certificates issued by trusted CA authorities:

```
REGISTRY=<image registry address obtained from the "Obtain Image Regist
ry Address" section>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTR
Y}/v2/"; then
    echo 'Test passed: The image registry uses certificates issued by t
rusted CA authorities. You do not need to execute the content in the "T
rust Insecure Image Registry" section.'
else
    echo 'Test failed: The image registry does not support HTTPS or the
certificate is not trusted. See the "Trust Insecure Image Registry" sec
tion for configuration.'
fi
```

2. If the test fails, see How to trust an insecure image registry?.

Obtain Cluster Information

1. Ensure network connectivity with the Huawei Cloud console.
2. Access the Cluster Management page ↗ of the `Cloud Container Engine CCE` feature; find the cluster to be imported and click the cluster name to enter the details page.
3. As shown in the figure below, follow the navigation to find the download KubeConfig file button: `Cluster Information - Connection Information - kubectl - Configuration`, and download the KubeConfig file.

Obtain Import Cluster Token

The KubeConfig file of public cloud clusters cannot be directly used for cluster import.

See How to obtain cluster information? to obtain the import cluster token.

Import Cluster

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click **Import Cluster**.
3. Configure the `Image Registry` related parameters according to the following instructions.

Parameter	Description
Image Registry	Repository for storing platform component images required by the cluster. **- Platform Default**: Image registry configured during global cluster deployment. **- Private Registry**: Pre-built registry storing platform required components. Enter the **private image registry address**, **port**, **username**, and **password** for accessing the image registry. **- Public Registry**: Use image registry services located on the public network. Before use, update public image registry cloud credentials to obtain registry authentication permissions.
Cluster Information	**Tip**: Upload the KubeConfig file for automatic parsing and filling by the platform. **Cluster Address**: The access address of the API Server exposed by the imported cluster, used for the platform to access the API Server of the imported cluster. **CA Certificate**: The CA certificate of the imported cluster. **Authentication Method**: The authentication method of the imported cluster, which requires using a **token** with **cluster management permissions** created in the previous step for authentication.

4. Click the `Parse KubeConfig File` button and submit the KubeConfig file downloaded in the previous step. The platform will automatically parse and fill in the `Cluster`

`Information` related parameters.

5. Click **Check Connectivity** to check network connectivity with the imported cluster and automatically identify the type of the imported cluster. The cluster type will be displayed as a badge in the upper right corner of the form.
6. After connectivity check passes, click **Import** and confirm.

   Tips:

   - Click the  icon on the right side of a cluster in **Importing** status to view the cluster's execution progress (status.conditions) in the popup **Execution Progress** dialog.
   - After successful cluster import, you can view the cluster's key information in the cluster list. The cluster status displays as normal and you can perform cluster-related operations.

Network Configuration

To ensure network connectivity between the global cluster and the imported cluster, see Imported Cluster Network Configuration.

Follow-up Operations

Ingress (Inbound Rules) and Storage Initialization

After importing the cluster, if you need to use Ingress (inbound rules) and storage-related features, see Huawei Cloud CCE Cluster Ingress Initialization Configuration and Huawei Cloud CCE Cluster Storage Initialization Configuration.

FAQ

After importing the cluster, the add node button is grayed out. How to add nodes?

Adding nodes through the platform interface is not supported. Contact the cluster provider to add nodes.

What certificates does the certificate management feature support for imported clusters?

1. **Kubernetes Certificates**: All imported clusters only support viewing APIServer certificate information in the platform certificate management interface. Viewing other Kubernetes certificates and automatic rotation are not supported.
2. **Platform Component Certificates**: All imported clusters can view platform component certificate information in the platform certificate management interface and support automatic rotation.

Limitations for this provider-specific import path

- Audit data retrieval is not supported.
- ETCD, Scheduler, and Controller Manager related monitoring information are not supported. APIServer partial monitoring charts are supported.
- Cluster certificate related information other than Kubernetes APIServer certificates cannot be retrieved.

Import Azure AKS Cluster

Use this provider-specific procedure to import an existing Azure AKS cluster as a third-party cluster. The external provider remains responsible for provider infrastructure and the provider-managed Kubernetes and node lifecycle unless a documented provider-specific capability provides that lifecycle.

TOC

- Prerequisites
- Prepare the Operating Environment
- Obtain Cluster Information
  - Obtain Import Token
- Import Cluster
- Network Configuration
- Post-Import Operations
  - Ingress (Inbound Rules) and Storage Initialization
- Frequently Asked Questions
  - How to configure AKS node external IP security group rules
  - How to access AKS node
  - Azure ALB using internal load balancer
  - Azure ALB using external load balancer
  - The add node button is grayed out after importing the cluster. How to add nodes?
  - What certificates are supported by the certificate management feature for imported clusters?
  - Limitations for this provider-specific import path

Prerequisites

- The Kubernetes version and parameters on the cluster must meet the Standard Kubernetes Cluster Component Version and Parameter Requirements.

  TIP

  - If AKS nodes cannot access the global cluster, see How to configure AKS node external IP security group rules.

- The image registry must support HTTPS access and provide a valid TLS certificate authenticated by a public certification authority.

Prepare the Operating Environment

To comply with Azure AKS security standards, the following steps must be performed using Cloud Shell.

1. Ensure network connectivity with Azure Console.
2. Open the Kubernetes Services page ↗, locate the cluster you want to import, and click to enter the cluster overview page.
3. Click the `Connect` button, which will open a floating window titled `Connect to <import cluster name>`. Follow the instructions to open Cloud Shell and configure the operating environment.

Obtain Cluster Information

Obtain Import Token

The KubeConfig file of public cloud clusters cannot be directly used for cluster import.

See How to obtain cluster information? to obtain the import cluster token.

Import Cluster

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click **Import Cluster**.
3. Configure the relevant parameters according to the following instructions.

Parameter	Description
Image Registry	The registry that stores platform component images required by the cluster. **- Platform Default**: The image registry configured when deploying the global cluster. **- Private Registry**: A pre-built registry that stores platform-required component images. Enter the **Private Image Registry Address**, **Port**, **Username**, and **Password** for accessing the image registry. **- Public Registry**: Use a public image registry service on the internet. Before use, update public image registry cloud credentials to obtain registry authentication permissions.
Cluster Information	**Tip**: Upload a KubeConfig file, and the platform will automatically parse and fill in the information. **Cluster Address**: The access address of the API Server exposed by the import cluster, used by the platform to access the import cluster's API Server. **CA Certificate**: The CA certificate of the import cluster. **Authentication Method**: The authentication method of the import cluster, which requires using a **Token** with **cluster management permissions** created in the previous step for authentication.

4. Click **Check Connectivity** to verify network connectivity with the import cluster and automatically identify the import cluster type. The cluster type will be displayed as a badge in the upper right corner of the form.
5. After connectivity check passes, click **Import** and confirm.

> **TIP**

- Click the **Details** icon on the right side of a cluster in **Importing** status to view the cluster's execution progress (status.conditions) in the popup **Execution Progress** dialog.
- After the cluster is successfully imported, you can view the cluster's key information in the cluster list. The cluster status will show as normal, and you can perform cluster-related operations.

Network Configuration

Ensure the global cluster and the imported cluster have network connectivity. See Network Configuration for Imported Clusters.

Post-Import Operations

Ingress (Inbound Rules) and Storage Initialization

After importing the cluster, if you need to use Ingress (inbound rules) and storage-related features, see Azure AKS Cluster Ingress Initialization Configuration and Azure AKS Cluster Storage Initialization Configuration.

Frequently Asked Questions

How to configure AKS node external IP security group rules

Nodes only have internal IPs by default. The external IP is configured on a frontend load balancer (LB), which is used for outbound traffic by default. This LB is controlled by the AKS principal. Direct manual modification of this configuration may cause issues. You can allow traffic through **Kubernetes** > **Properties** > **Infrastructure Resource Group** > **Network Security Group** > **Add Outbound/Inbound All Rules**.

How to access AKS node

To view logs of system components such as Kubelet, CNI, and kernel, you need to SSH into the node first. It is recommended to use the `kubectl-node-shell` plugin instead of assigning public IP addresses to each node.

Option 1: Using kubectl node-shell

See the kubectl-node-shell project ↗.

Option 2: Using debug

See the provider guidance for connecting to AKS nodes ↗.

> **NOTE**
>
> This example requires kubectl version 1.25 or later, which includes the GA `kubectl debug` command.

```
kubectl debug node/aks-newadd-41368356-vmss000002 -it --image=mcr.microsoft.com/dotnet/runtime-deps:6.0
chroot /host
```

Azure ALB using internal load balancer

For internal load balancer behavior, see the provider guidance ↗.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: internal-app
  namespace: cpaas-system
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
  type: LoadBalancer
  ports:
  - name: http-port
    port: 80
    protocol: TCP
  - name: https-port
    port: 443
    protocol: TCP
  selector:
    service.cpaas.io/name: deployment-aks-alb
    service_name: alb2-aks-alb
```

Azure ALB using external load balancer

Deploy a highly available ALB with the access address configured as the external LB.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: azure-alb
  namespace: cpaas-system
spec:
  type: LoadBalancer
  ports:
  - name: http-port
    port: 80
    protocol: TCP
  - name: https-port
    port: 443
    protocol: TCP
  - name: prom-port
    port: 11780
    protocol: TCP
  - name: prom2-port
    port: 11781
    protocol: TCP
  - name: prom3-port
    port: 15012
    protocol: TCP
  selector:
    service_name: alb2-cpaas-system
```

If it has been deployed in advance, you can use the following command to modify it.

```
kubectl edit helmrequest -n cpaas-system uat-cluster-aks-alb
```

The add node button is grayed out after importing the cluster. How to add nodes?

Adding nodes through the platform interface is not supported. Contact the cluster provider to add nodes.

What certificates are supported by the certificate management feature for imported clusters?

1. **Kubernetes Certificates**: All imported clusters only support viewing APIServer certificate information in the platform certificate management interface. Other Kubernetes certificates cannot be viewed and automatic rotation is not supported.
2. **Platform Component Certificates**: All imported clusters can view platform component certificate information in the platform certificate management interface and support automatic rotation.

Limitations for this provider-specific import path

- Audit data retrieval is not supported.
- ETCD, Scheduler, and Controller Manager related monitoring information is not supported. APIServer partial monitoring charts are supported.
- Cluster certificate-related information other than Kubernetes APIServer certificates cannot be retrieved.

Import Alibaba Cloud ACK Cluster

Use this provider-specific procedure to import existing Alibaba Cloud ACK managed clusters or dedicated clusters as third-party clusters. The external provider remains responsible for provider infrastructure and the provider-managed Kubernetes and node lifecycle unless a documented provider-specific capability provides that lifecycle.

TIP

For provider cluster type details, see the provider guide ↗.

TOC

- Prerequisites
- Get Image Registry Address
- Determine if Image Registry Requires Additional Configuration
- Get KubeConfig
- Import Cluster
- Network Configuration
- FAQ
  - How to handle port conflicts between Alibaba Cloud monitoring and platform monitoring components?
  - How to use public network access for Alibaba Cloud clusters?
  - After importing a cluster, the add node button is grayed out. How to add nodes?
  - Which certificates are supported by the certificate management function for imported clusters?
  - Limitations for this provider-specific import path

Prerequisites

- The Kubernetes version and parameters on the cluster meet the component version and parameter requirements for importing standard Kubernetes clusters.

Get Image Registry Address

- To use the **platform-deployed** image registry from the global cluster deployment, execute the following command on the **control plane node of the global cluster** to get the address:

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.preferPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.data.platformURL}' | awk -F \// '{print $NF}')
fi
echo "Image registry address is: $REGISTRY"
```

- To use an **external image registry**, manually set the **REGISTRY** variable.

```
REGISTRY=<external image registry address>  # Valid examples: registry.example.cn:60080 or 192.168.134.43
echo "Image registry address is: $REGISTRY"
```

Determine if Image Registry Requires Additional Configuration

1. Execute the following command to determine if the specified image registry supports HTTPS access and uses certificates issued by trusted CA authorities:

```
REGISTRY=<image registry address obtained from the "Get Image Registry Address" section>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTRY}/v2/"; then
    echo 'Test passed: The image registry uses certificates issued by trusted CA authorities. You do not need to execute the content in the "Trust Insecure Image Registry" section.'
else
    echo 'Test failed: The image registry does not support HTTPS or the certificate is not trusted. See the "Trust Insecure Image Registry" section for configuration.'
fi
```

2. If the test fails, see How to trust insecure image registries?.

Get KubeConfig

1. Log in to the Alibaba Cloud Container Service management platform.
2. In the left navigation bar of the console, click **Clusters**.
3. On the **Cluster List** page, click the target cluster name or **Details** under the **Actions** column on the right side of the target cluster.
4. On the **Cluster Information** page, click the **Connection Information** tab, then click **Generate Temporary KubeConfig**.
5. In the **Temporary KubeConfig** dialog, set the validity period of the temporary credentials and the method to access the cluster (including public network access and internal network access).
6. Click **Generate Temporary KubeConfig**, then click **Copy** to copy the content and save it to the **KubeConfig** file on your local computer.
7. After the cluster is successfully imported, you can revoke the temporary credentials.

Import Cluster

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

2. Click **Import Cluster**.
3. Configure the relevant parameters according to the following instructions.

Parameter	Description
Image Registry	Repository for storing platform component images required by the cluster. **- Platform Default**: Image registry configured during global cluster deployment. **- Private Registry**: Pre-built registry that stores platform-required component images. You need to enter the **private image registry address**, **port**, **username**, and **password** for accessing the image registry. **- Public Registry**: Use public image registry services on the internet. Before use, update public repository cloud credentials to obtain repository authentication permissions.
Cluster Information	**Tip**: Can be filled manually or uploaded via KubeConfig file for automatic parsing and filling by the platform. **Parse KubeConfig File**: After uploading the obtained KubeConfig file, the platform will automatically parse and fill the **Cluster Information**. You can modify the automatically filled information. **Cluster Address**: The access address of the cluster's externally exposed API Server, used by the platform to access the cluster's API Server. **CA Certificate**: The cluster's CA certificate. **Note**: When entering manually, you need to enter the Base64-decoded certificate. **Authentication Method**: Authentication method for accessing the cluster. You need to use a **token** or **certificate authentication (client certificate and key)** with **cluster management permissions** for authentication.

4. Click **Check Connectivity** to check network connectivity with the cluster to be imported and automatically identify the type of cluster to be imported. The cluster type will be displayed as a badge in the upper right corner of the form.
5. After connectivity check passes, click **Import** and confirm.

TIP

- Click the **Details** icon on the right side of a cluster in **Importing** status to view the cluster's execution progress (status.conditions) in the popup **Execution Progress** dialog.

- After the cluster is successfully imported, you can view the cluster's key information in the cluster list. The cluster status shows as normal and you can perform cluster-related operations.

Network Configuration

Ensure network connectivity between the global cluster and the cluster to be imported. See Network Configuration for Imported Clusters.

FAQ

How to handle port conflicts between Alibaba Cloud monitoring and platform monitoring components?

When Alibaba Cloud's built-in monitoring and platform monitoring components coexist, port conflicts will occur. It is recommended to uninstall Alibaba Cloud monitoring and keep only platform monitoring.

How to use public network access for Alibaba Cloud clusters?

If using public network access for Alibaba Cloud clusters, you can bind a public IP on Alibaba Cloud.

After importing a cluster, the add node button is grayed out. How to add nodes?

Both **Alibaba Cloud ACK managed clusters** and **ACK dedicated clusters** do not support adding nodes through the platform interface. Add nodes in the provider backend or contact the cluster provider.

Which certificates are supported by the certificate management function for imported clusters?

1. **Kubernetes Certificates**: All imported clusters only support viewing APIServer certificate information in the platform certificate management interface. They do not support viewing other Kubernetes certificates and do not support automatic rotation.
2. **Platform Component Certificates**: All imported clusters can view platform component certificate information in the platform certificate management interface and support automatic rotation.

Limitations for this provider-specific import path

- **Alibaba Cloud ACK managed clusters** do not support obtaining audit data.
- **Alibaba Cloud ACK managed clusters** do not support ETCD, Scheduler, Controller Manager related monitoring information, but support some APIServer monitoring charts.
- Both **Alibaba Cloud ACK managed clusters** and **ACK dedicated clusters** do not support obtaining cluster certificate-related information except for Kubernetes APIServer certificates.

Import Tencent Cloud TKE Cluster

Use this provider-specific procedure to import existing Tencent Cloud TKE dedicated clusters or managed clusters as third-party clusters. The external provider remains responsible for provider infrastructure and the provider-managed Kubernetes and node lifecycle unless a documented provider-specific capability provides that lifecycle.

TIP

For provider cluster type details, see the provider guide ↗.

TOC

Prerequisites

Obtain Image Registry Address

Determine if Image Registry Requires Additional Configuration

Obtain KubeConfig

Import Cluster

Network Configuration

FAQ

After importing the cluster, the "Add Node" button is grayed out. How to add nodes?

What certificates does the certificate management function for imported clusters support?

Limitations for this provider-specific import path

Prerequisites

- The Kubernetes version and parameters on the cluster meet the component version and parameter requirements for importing standard Kubernetes clusters.
- The image registry must support HTTPS access and provide a valid TLS certificate issued by a public certificate authority.

Obtain Image Registry Address

- To use the **platform-deployed** image registry configured during global cluster deployment, execute the following command on the **control plane node of the global cluster** to obtain the address:

```
if [ "$(kubectl get productbase -o jsonpath='{.items[].spec.registry.prefe
rPlatformURL}')" = 'false' ]; then
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.registryAddress}')
else
    REGISTRY=$(kubectl get cm -n kube-public global-info -o jsonpath='{.
data.platformURL}' | awk -F \// '{print $NF}')
fi
echo "Image registry address is: $REGISTRY"
```

- To use an **external image registry**, manually set the **REGISTRY** variable.

```
REGISTRY=<external image registry address>  # Valid examples: registry.e
xample.cn:60080 or 192.168.134.43
echo "Image registry address is: $REGISTRY"
```

Determine if Image Registry Requires Additional Configuration

1. Execute the following command to determine whether the specified image registry supports HTTPS access and uses a certificate issued by a trusted CA:

```
REGISTRY=<image registry address obtained from the "Obtain Image Regist
ry Address" section>

if curl -s -o /dev/null --retry 3 --retry-delay 5 -- "https://${REGISTR
Y}/v2/"; then
    echo 'Verification passed: The image registry uses a certificate is
sued by a trusted CA. It is not necessary to execute the content in the
"Trust Unsafe Image Registry" section.'
else
    echo 'Verification failed: The image registry does not support HTTP
S or the certificate is not trusted. See the "Trust Unsafe Image Regist
ry" section for configuration.'
fi
```

2. If verification fails, see How to trust an unsafe image registry?.

Obtain KubeConfig

1. Log in to the Tencent Cloud Container Service management platform.
2. In **Cluster Details** > **Basic Information**, view the **Cluster APIServer** information.
3. Select **Internet Access** or **Intranet Access** based on the actual customer network, then download **Kubeconfig** and save it to your local computer.

Import Cluster

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click **Import Cluster**.
3. Configure the relevant parameters according to the following instructions.

Parameter	Description
Image Registry	Registry for storing platform component images required by the cluster. - **Platform Default**: Image registry configured during global deployment. - **Private Registry**: Pre-built registry that stores platform-required component images. Enter the **private image registry address**, **port**, **username**, and

Parameter	Description
	password for accessing the image registry. **- Public Registry**: Use image registry services located on the public network. Before use, update public registry cloud credentials to obtain registry authentication permissions.
Cluster Information	**Tip**: Can be filled manually or uploaded via KubeConfig file for automatic parsing and filling by the platform. **Parse KubeConfig File**: After uploading the obtained KubeConfig file, the platform will automatically parse and fill in the **Cluster Information**, and you can modify the automatically filled information. **Cluster Address**: The access address of the cluster's externally exposed API Server, used by the platform to access the cluster's API Server. **CA Certificate**: The cluster's CA certificate. **Note**: When manually inputting, you need to input the Base64-decoded certificate. **Authentication Method**: Authentication method for accessing the cluster, requires using a **token** (Token) or **certificate authentication (client certificate and key)** with **cluster management permissions**.

4. Click **Check Connectivity** to verify network connectivity with the cluster to be imported and automatically identify the type of cluster to be imported. The cluster type will be displayed as a badge in the upper right corner of the form.
5. After connectivity check passes, click **Import** and confirm.
   Tip:
   - Click the **Details** icon on the right side of a cluster in **Importing** status to view the cluster's execution progress (status.conditions) in the popup **Execution Progress** dialog.
   - After successful cluster import, you can view the cluster's key information in the cluster list. The cluster status displays as normal, and cluster-related operations can be performed.

Network Configuration

Ensure network connectivity between the global cluster and the cluster to be imported. See Network Configuration for Importing Clusters.

FAQ

After importing the cluster, the "Add Node" button is grayed out. How to add nodes?

Both **TKE Dedicated clusters** and **TKE Managed clusters** do not support adding nodes through the platform interface. Add nodes in the provider backend or contact the cluster provider.

What certificates does the certificate management function for imported clusters support?

1. **Kubernetes Certificates**: All imported clusters only support viewing APIServer certificate information in the platform certificate management interface. They do not support viewing other Kubernetes certificates and do not support automatic rotation.
2. **Platform Component Certificates**: All imported clusters can view platform component certificate information in the platform certificate management interface and support automatic rotation.

Limitations for this provider-specific import path

- **TKE Managed clusters** do not support obtaining audit data.
- **TKE Managed clusters** do not support ETCD, Scheduler, Controller Manager related monitoring information, but support partial APIServer monitoring charts.
- Both **TKE Managed clusters** and **TKE Dedicated clusters** do not support obtaining cluster certificate-related information except for Kubernetes APIServer certificates.

Register Cluster

Use register when the target third-party cluster should initiate the connection to the platform through a reverse proxy tunnel.

TOC

Prerequisites

Important Notes

Register Cluster

View Registration Command

FAQ

How to Resolve Distributed Storage Deployment Failure When the Runtime Component of the Connected Cluster is Containerd?

Prerequisites

- Depending on the target environment, the Kubernetes version and component parameters in the target cluster must meet the version and parameter requirements.
- The image registry must support HTTPS access and provide a valid TLS certificate authenticated by a public certification authority. If this requirement cannot be met, see How to Trust Insecure Image Registries?

Note: The **Public Registry** provided by the platform on the public network already meets HTTPS access requirements. You only need to verify whether the **Platform Default** and **Private Registry** support HTTPS access.

- If the runtime component of the cluster to be connected is Containerd, you need to modify the Containerd configuration before connecting the cluster to ensure successful deployment of distributed storage.

Important Notes

Network interface traffic monitoring recognizes network interfaces with names matching `eth\.\|en\.\|wl\.*\|ww\.*` by default. If the target cluster uses other naming conventions, see Collecting Network Data from Custom Named Network Cards and update the corresponding resources after registration.

Register Cluster

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click **Managed Clusters** > **Register Cluster**.
3. Configure the registry parameters for storing platform component images required by the registered cluster according to the following instructions.

Parameter	Description
Platform Default	Image registry used when deploying global components.
Private Registry	External image registry that you have set up in advance. You need to enter the **Private Image Registry Address**, **Port**, **Username**, and **Password** for accessing the image registry.

Parameter	Description
Public Registry	Pull required images through the public image registry provided by the platform. Ensure that the target cluster can access the public network. Before use, see Update Public Network Image Registry Cloud Credentials to obtain registry authentication permissions.

4. Click **Create**, obtain the registration command on the **Registration Command** page and run the command in the cluster to be registered.
   Note: The registration command is valid for 24 hours. Obtain a new command after it expires.

View Registration Command

Find the cluster waiting for registration in the cluster list and click **View Registration Command**. Run the registration command before it expires.

FAQ

How to Resolve Distributed Storage Deployment Failure When the Runtime Component of the Connected Cluster is Containerd?

When the runtime component of the connected cluster is Containerd, distributed storage deployment will fail. To resolve this issue, you need to manually modify the Containerd configuration information on **all nodes** of the cluster and restart Containerd.

Note: If you modify the Containerd configuration by following the steps below before deploying distributed storage, you do not need to execute step four.

1. Log in to the cluster node and edit the `/etc/systemd/system/containerd.service` file, changing the `LimitNOFILE` parameter value to `1048576`.
2. Execute the command `systemctl daemon-reload` to reload the configuration.

3. Execute the command `systemctl restart containerd` to restart Containerd.
4. Execute the command `kubectl delete pod --all -n rook-ceph` on the cluster control plane node to restart all Pods in the rook-ceph namespace to make the configuration effective.

Public Cloud Cluster Initialization

Network Initialization

AWS EKS Cluster Network Initialization Configuration

TOC

Support Overview
Prerequisites
Configuration Steps
Deploy AWS Load Balancer Controller
Create Ingress and LoadBalancer Services
Related Operations
Test AWS CLI and eksctl Installation
Get ACCOUNT_ID
Kubeconfig Configuration File
Add Tags to Subnets
Create Certificate

Support Overview

Feature	Support Status	Requirements
LoadBalancer Service	Supported	Optionally deploy AWS Load Balancer Controller. Without this controller, LoadBalancer capabilities are limited.
Ingress	Supported	Optionally deploy AWS Load Balancer Controller. Optionally enable **Ingress Class** functionality (once enabled, you can manually select ingress classes when creating ingress through the form interface).

Prerequisites

- Prepare two subnets with the **kubernetes.io/role/elb** tag. For shared subnets, add the **kubernetes.io/cluster/<cluster-name>: shared** tag. See Adding Tags to Subnets.
- If you have created an EKS cluster, import the Amazon EKS cluster.
- Ensure kubectl, Helm, AWS CLI, and eksctl tools are available before deploying AWS Load Balancer Controller.
  Note: After installing the tools, configure login information using the user who created the cluster via AWS CLI, and test if AWS CLI and eksctl tools are correctly installed.
- Obtain ACCOUNT_ID, **REGION**, and **CLUSTER_NAME** in advance, and replace `<ACCOUNT_ID>`, `<REGION>`, and `<CLUSTER_NAME>` with the actual values.
  Note: **ACCOUNT_ID** is the Account ID of the user who created the cluster, **REGION** is the cluster region, and **CLUSTER_NAME** is the cluster name.
- Update and verify the Kubeconfig configuration file.

Configuration Steps

Deploy AWS Load Balancer Controller

Note: For detailed information on deploying AWS Load Balancer Controller, see the provider guide ↗.

Configure OIDC Provider

Kubernetes clusters use OpenID Connect (OIDC) for identity management and are associated with an OIDC issuer URL. To enable AWS Identity in the cluster and allow IAM roles for Service Accounts, create an IAM OIDC Provider associated with the cluster's OIDC issuer URL.

Execute the following command in eksctl to configure the OIDC Provider:

```
eksctl utils associate-iam-oidc-provider --region=<REGION> --cluster=<CLU
STER_NAME> --approve
```

Configure Service Account

Execute the following commands to create an IAM policy and create a Service Account named `aws-load-balancer-controller`, associating it with an IAM role:

```
curl -o aws-load-balancer-controller-iam-policy.json https://raw.githubus
ercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.7/docs/in
stall/iam_policy.json
aws iam create-policy \
    --policy-name <CLUSTER_NAME>-AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://aws-load-balancer-controller-iam-policy.json

eksctl create iamserviceaccount \
  --cluster=<CLUSTER_NAME> \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::<ACCOUNT_ID>:policy/<CLUSTER_NAME>-AWS
LoadBalancerControllerIAMPolicy \
  --approve
```

Deploy AWS Load Balancer Controller to Cluster

Execute the following commands in eksctl to deploy AWS Load Balancer Controller:

1. Add the eks-charts repository:

```
helm repo add eks https://aws.github.io/eks-charts
```

2. Update the local repository:

```
helm repo update eks
```

3. Deploy the AWS Load Balancer Controller Helm Chart to the cluster:
   Note: `aws-load-balancer-controller` is the **Service Account** created in Configure Service Account.

```
helm install aws-load-balancer-controller eks/aws-load-balancer-control
ler \
  -n kube-system \
  --version=v2.4.7 \
  --set ingressClassConfig.default=true \
  --set clusterName=<CLUSTER_NAME> \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

Create Ingress and LoadBalancer Services

You can create ingress and LoadBalancer services simultaneously or choose one based on your needs.

Create Ingress

1. In **Container Platform**, click **Network** > **Ingress** in the left navigation.
2. Click **Create Ingress** and select **EKS Ingress Class** for **Ingress Class**.
3. Select **Protocol**. Default is **HTTP**. For **HTTPS**, first create a certificate and select it.
4. Switch to **YAML** and add the following annotations. For details, see the annotation reference ↗:

```
alb.ingress.kubernetes.io/scheme: internet-facing ## Specify public acc
ess
alb.ingress.kubernetes.io/target-type: ip ## Route traffic directly to
pods
```

5. Click **Create**.

Create LoadBalancer Service

1. In **Container Platform**, click **Network** > **Services** in the left navigation.
2. Click **Create Service** and select **LoadBalancer** for **External Access**.
3. Expand **annotations** and fill in LoadBalancer service annotations as needed.
4. Click **Create**.

Related Operations

Test AWS CLI and eksctl Installation

- Execute the following command. If it returns a cluster list, AWS CLI is correctly installed:

```
aws eks list-clusters
```

- Execute the following command. If it returns a cluster list, eksctl is correctly installed:

```
eksctl get clusters
```

Get ACCOUNT_ID

Execute `aws sts get-caller-identity` to get **ACCOUNT_ID**. The `651168850570` in the response is the **ACCOUNT_ID**:

```
{
  "ARN": "arn:aws:iam::651168850570:user/jwshi"
}
```

Kubeconfig Configuration File

1. Execute the following command to update the Kubeconfig file for the specified region:

```
aws eks --region <REGION>  update-kubeconfig --name <CLUSTER_NAME>
```

2. Execute the following command to verify the Kubeconfig file. If it returns information normally, the configuration is correct:

```
kubectl get svc -n cpaas-system
```

Add Tags to Subnets

1. Execute the following command to get cluster subnets:

```
eksctl get cluster --name <CLUSTER_NAME>
```

2. Execute the following command to get subnet details:

```
aws ec2 describe-subnets
```

3. Execute the following commands to add tags to subnets. Replace `<subnet-id>` with actual values. See Subnet auto-discovery ↗:

   - Add the `kubernetes.io/role/elb` tag to subnets:

   ```
   aws ec2 create-tags --resources <subnet-id> --tags Key=kubernetes.io/
   role/elb,Value="1"
   ```

   - Add the `kubernetes.io/cluster/<CLUSTER_NAME>: shared` tag to shared subnets:

   ```
   aws ec2 create-tags --resources <subnet-id> --tags Key=kubernetes.io/
   cluster/<CLUSTER_NAME>,Value="shared"
   ```

Create Certificate

When using HTTPS protocol, save HTTPS certificate credentials as a Secret (TLS type) in advance.

1. In **Container Platform**, click **Configuration** > **Secrets** in the left navigation.
2. Click **Create Secret**.
3. Select **TLS** type and import or fill in **Certificate** and **Private Key** as needed.
4. Click **Create**.

AWS EKS Supplementary Information

TOC

Terminology

Important Notes

EKS Using aws-lb to Provide External Access for Container Network Load Balancers

Service Annotation Configuration Instructions

Access Address Acquisition Method

Terminology

Abbreviation	Full Name	Description
eks-clb	Classic Load Balancer	AWS default load balancer. Has issues in certain situations and is not recommended.
eks-nlb	Network Load Balancer	AWS Layer 4 load balancer that performs load balancing at TCP/UDP level, suitable for scenarios requiring higher-level network control.
eks-alb	Application Load Balancer	AWS Layer 7 load balancer. Compared to eks-nlb, eks-alb can parse HTTP/HTTPS protocols and distribute requests more intelligently, suitable for web applications.

Abbreviation	Full Name	Description
aws-lb	AWS Load Balancer	Load balancer installed on Kubernetes that can automatically create eks-nlb and eks-alb based on LoadBalancer Services and Ingress in Kubernetes to meet application load balancing needs.
Platform Load Balancer	-	Platform's proprietary Layer 7 load balancer.
Service Annotations	-	Metadata attached to objects in key-value pairs. This additional information can be recognized and utilized to enhance and simplify management of various aspects of Kubernetes resources. Annotations can be explanatory text without specific functionality, specify cloud provider configurations or behaviors, or specify configuration parameters and tools. Very powerful functionality.

Important Notes

When creating load balancers, it's recommended to manually configure service annotations to ensure the platform load balancer correctly uses aws-lb. If the appropriate service annotations are not configured correctly, the platform will default to using eks-clb, which has UDP-related issues that may cause unexpected situations.

EKS Using aws-lb to Provide External Access for Container Network Load Balancers

Service Annotation Configuration Instructions

1. In the corresponding cluster, execute the following command using kubectl to find all Pods in the kube-system namespace with names containing "aws-load":

```
kubectl get pod -n kube-system |grep aws-load
```

2. Create a load balancer; for detailed creation steps and parameters, see the Load Balancer creation section in AWS EKS Service Annotation Instructions.

   - If the above command returns no related Pods, it means the cluster does not have AWS Load Balancer Controller installed. No service annotations are needed; create the load balancer directly.
   - If the above command returns related Pods, it means the cluster has AWS Load Balancer Controller installed. When creating a load balancer in the corresponding cluster, add the following service annotations. For annotation details, see AWS EKS Service Annotation Instructions:
     - `service.beta.kubernetes.io/aws-load-balancer-type: external //Required`
     - `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip //Required`
     - `service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing // Optional. Add this annotation if public network support is needed.`

Access Address Acquisition Method

- When creating container network type load balancers, the filled service annotations will be set on the LoadBalancer Service corresponding to the platform load balancer.
- In public clouds, LoadBalancer Services with appropriate service annotations will be recognized by the public cloud and assigned addresses. The platform load balancer will read this address and set it as its own access address.

Huawei Cloud CCE Cluster Network Initialization Configuration

TOC

Support Overview
Prerequisites
Configuration Steps
Create Ingress
Create LoadBalancer Service
Related Operations
Create Certificate

Support Overview

Feature	Support Status	Requirements
LoadBalancer Service	Default Support	No additional deployment required.
Ingress	Default Support	Optionally enable **Ingress Class** functionality (once enabled, you can manually select ingress classes when

Feature	Support Status	Requirements
		creating ingress through the form interface). No additional deployment required.

Prerequisites

If you have created a CCE cluster, import the CCE cluster (Public Cloud).

Configuration Steps

You can create ingress and LoadBalancer services simultaneously or choose one based on your needs.

Create Ingress

There are two methods to create ingress. **Method 1: Manual Ingress Class Selection** is recommended.

Note: Avoid creating two ingress resources with the same **path**.

(Recommended) Method 1: Manual Ingress Class Selection

1. In **Container Platform**, click **Network** > **Ingress** in the left navigation.
2. Click **Create Ingress** and select **CCE Ingress Class** for **Ingress Class**.
3. Select **Protocol**. Default is **HTTP**. For **HTTPS**, first create a certificate and select it.
4. Switch to **YAML** and add the following annotations based on your default Ingress Controller type. For annotation details, see Using Annotations to Configure Load Balancers ↗:
   Note: Replace the **values** in the annotations below with actual environment values.

Default Ingress Controller Type	Annotations
Shared (Auto-create)	`kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"{random}","bandwidth_chargemode":"traffic","bandwidth_size":5,"bandwi` `kubernetes.io/elb.class: union`

Shared (Reuse)	`kubernetes.io/elb.class: union`
	`kubernetes.io/elb.id: <Load Balancer Instance ID>`
	`kubernetes.io/elb.port: '80'`

Dedicated (Auto-create)	`kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"<E`
	`Name>","bandwidth_chargemode":"traffic","bandwidth_size":5,"bandwidth`
	`["<AZ A>","<AZ B>","<AZ C>"],"elb_virsubnet_ids":["<ELB Virtual Subne`
	`ID>"],"l7_flavor_name":"L7_flavor.elb.s1.small","l4_flavor_name":"L4_`
	`kubernetes.io/elb.class: performance`
	`kubernetes.io/elb.port: "80"`

Dedicated (Reuse)	`kubernetes.io/elb.class: performance`
	`kubernetes.io/elb.id: <Load Balancer Instance ID>`
	`kubernetes.io/elb.port: "80"`

5. Click **Create**. Once created, you can access cluster services through ELB.

Method 2: Use Default Ingress Class

1. Create an IngressClass YAML file with the following content. For details, see Default Ingress Class ↗:

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
  name: cce
spec:
  controller: alauda/cce
```

2. Save the file and apply it to the imported cluster. Replace `<filename.yaml>` with your actual YAML filename:

```
kubectl apply -f <filename.yaml>
```

3. In **Container Platform**, click **Network** > **Ingress** in the left navigation.
4. Select **Protocol**. Default is **HTTP**. For **HTTPS**, first create a certificate and select it.
5. Click **Create**. Once created, you can access cluster services through ELB.

Create LoadBalancer Service

1. In **Container Platform**, click **Network** > **Services** in the left navigation.
2. Click **Create Service** and select **LoadBalancer** for **External Access**.
3. Expand **annotations** and fill in LoadBalancer service annotations as needed.
4. Click **Create**.

Related Operations

Create Certificate

When using HTTPS protocol, save HTTPS certificate credentials as a Secret (TLS type) in advance.

1. In **Container Platform**, click **Configuration** > **Secrets** in the left navigation.

2. Click **Create Secret**.

3. Select **TLS** type and import or fill in **Certificate** and **Private Key** as needed.

4. Click **Create**.

Azure AKS Cluster Network Initialization Configuration

TOC

Support Overview
Prerequisites
Configuration Steps
Deploy Ingress Controller
Create Ingress and LoadBalancer Services
Related Operations
Create Certificate

Support Overview

Feature	Support Status	Requirements
LoadBalancer Service	Default Support	No additional deployment required.
Ingress	Supported	Optionally deploy Ingress Controller. Optionally enable **Ingress Class** functionality (once enabled, you can

Feature	Support Status	Requirements
		manually select ingress classes when creating ingress through the form interface).

Prerequisites

If you have created an AKS cluster, import the Azure AKS cluster.

Configuration Steps

Deploy Ingress Controller

AKS uses **container network mode** and leverages **Nginx Ingress Controller** to manage load balancers, while providing external access addresses for virtual IP addresses (VIPs) in the container internal network through **LoadBalancer** type **Services**.

1. Log in to Microsoft Azure and access your created AKS cluster.
2. In the left navigation, click **Kubernetes Resources** > **Services and Ingresses**.
3. Click **Create**, select **Ingress (Preview)** from the dropdown, and it will prompt and automatically create an Ingress Controller.
4. Click **Enable** and wait for completion.

Create Ingress and LoadBalancer Services

You can create ingress and LoadBalancer services simultaneously or choose one based on your needs.

Create Ingress

1. In **Container Platform**, click **Network** > **Ingress** in the left navigation.
2. Click **Create Ingress** and select **webapprouting.kubernetes.azure.com** for **Ingress Class**.

3. Select **Protocol**. Default is **HTTP**. For **HTTPS**, first create a certificate and select it.
4. Click **Create**.

Create LoadBalancer Service

1. In **Container Platform**, click **Network** > **Services** in the left navigation.
2. Click **Create Service** and select **LoadBalancer** for **External Access**.
3. Expand **annotations** and fill in LoadBalancer service annotations as needed.
4. Click **Create**.

Related Operations

Create Certificate

When using HTTPS protocol, save HTTPS certificate credentials as a Secret (TLS type) in advance.

1. In **Container Platform**, click **Configuration** > **Secrets** in the left navigation.
2. Click **Create Secret**.
3. Select **TLS** type and import or fill in **Certificate** and **Private Key** as needed.
4. Click **Create**.

Google GKE Cluster Network Initialization Configuration

TOC

Support Overview
Prerequisites
Configuration Steps
Deploy Ingress Controller
Create Ingress and LoadBalancer Services
Related Operations
View Ingress Resources in Google Cloud
Create Certificate

Support Overview

Feature	Support Status	Requirements
LoadBalancer Service	Default Support	No additional deployment required.

Feature	Support Status	Requirements
Ingress	Default Support	Optionally enable **Ingress Class** functionality (once enabled, you can manually select ingress classes when creating ingress through the form interface). No additional deployment required.

Prerequisites

If you have created a GKE cluster, import the GKE cluster.

Configuration Steps

Deploy Ingress Controller

No manual deployment is required. GKE provides a managed built-in Ingress controller called GKE Ingress. This controller maps Ingress resources to Google Cloud Load Balancers to handle HTTP(S) workloads in GKE, making configuration simpler and more automated.

Create Ingress and LoadBalancer Services

You can create ingress and LoadBalancer services simultaneously or choose one based on your needs.

Create Ingress

1. In **Container Platform**, click **Network** > **Ingress** in the left navigation.
2. Click **Create Ingress** and select **GKE Ingress Class** for **Ingress Class**.
3. Select **Protocol**. Default is **HTTP**. For **HTTPS**, first create a certificate and select it.
4. Click **Create**. Wait approximately 5 minutes for GKE platform to automatically assign a public IP address to the ingress.
   Note: Different ingress resources will be assigned different public IP addresses.

Create LoadBalancer Service

1. In **Container Platform**, click **Network** > **Services** in the left navigation.
2. Click **Create Service** and select **LoadBalancer** for **External Access**.
3. Expand **annotations** and fill in LoadBalancer service annotations as needed.
4. Click **Create**.

Related Operations

View Ingress Resources in Google Cloud

1. Go to **Google Cloud > Kubernetes Engine** and click **Services and Ingress** in the left navigation.
2. Click **INGRESS**.
3. View information about corresponding Ingress resources in the list.

Create Certificate

When using HTTPS protocol, save HTTPS certificate credentials as a Secret (TLS type) in advance.

1. In **Container Platform**, click **Configuration** > **Secrets** in the left navigation.
2. Click **Create Secret**.
3. Select **TLS** type and import or fill in **Certificate** and **Private Key** as needed.
4. Click **Create**.

Storage Initialization

Public Cloud Storage Initialization Overview

Use the storage initialization procedures after importing a supported public cloud Kubernetes service and before relying on persistent storage features in workloads. The external provider remains responsible for provider infrastructure and provider-managed storage backends; ACP configures and uses the storage classes exposed to the imported cluster.

The following tables summarize the default storage classes and operations documented for each provider-specific initialization path.

TOC

Storage Class Support

AWS EKS Clusters

Huawei Cloud CCE Clusters

Azure AKS Clusters

Google GKE Clusters

Storage Class Support

AWS EKS Clusters

Storage Type	Default Storage Class	Create PVC with RWO Access Mode	Create PVC with RWX Access Mode	PVC Expansion	P S
File Storage	efs-sc	Supported	Supported	Not Supported	N
Block Storage	ebs-sc	Supported	Not Supported	Supported	N

Huawei Cloud CCE Clusters

Storage Type	Default Storage Class	Create PVC with RWO Access Mode	Create PVC with RWX Access Mode	PVC Expansion	P S
File Storage	csi-nas	Not Supported	Supported	Supported	N
Block Storage	csi-disk	Supported	Not Supported	Supported	N

Azure AKS Clusters

Storage Type	Default Storage Class	Create PVC with RWO Access Mode	Create PVC with RWX Access Mode	PVC Expansion	P S
File Storage	azurefile	Supported	Supported	Supported	N
Block Storage	default	Supported	Not Supported	Supported	N

Google GKE Clusters

Storage Type	Default Storage Class	Create PVC with RWO Access Mode	Create PVC with RWX Access Mode	PVC Expansion	P S
File Storage	standard-rwx	Supported	Supported	Supported	N

Storage Type	Default Storage Class	Create PVC with RWO Access Mode	Create PVC with RWX Access Mode	PVC Expansion	P S
Block Storage	standard-rwo	Supported	Not Supported	Supported	N

AWS EKS Cluster Storage Initialization Configuration

Platform integration with AWS EKS and storage initialization configuration.

TOC

Constraints and Limitations
Prerequisites
Configuration Steps
Create Storage Classes
Modify Storage Class Project Assignment
Related Operations
Configure Available Storage Class Parameters

Constraints and Limitations

- The default efs-sc file storage class may not support permission modifications after mounting, which may cause some applications like PostgreSQL and Jenkins to fail to run properly.
- A1 series instances are not supported by AL2023 AMIs, which prevents the EBS block storage plugin (Amazon EBS CSI Driver) from deploying properly. The EBS CSI driver has

GA multi-architecture/ARM support, so the limitation is with AMI/instance support rather than the driver itself. If you need to use EBS block storage classes, avoid using the following instance types and consider Graviton2/3 alternatives instead:

- a1.medium
- a1.large
- a1.xlarge
- a1.2xlarge
- a1.4xlarge

Recommended alternatives: Use Graviton2/3 instance families such as m6g, c6g, r6g, t4g, etc., which provide better performance and full EBS CSI driver support.

Prerequisites

- Ensure kubectl ↗ and AWS CLI tools are available.
- If you have created an EKS cluster, import the Amazon EKS cluster; if not, create an AWS EKS cluster.
- Deploy the EFS file storage plugin **Amazon EFS CSI Driver** and EBS block storage plugin **Amazon EBS CSI Driver** in the EKS cluster.
  Note: If using EFS file storage, create file storage in the EKS region and record the **File System ID** from the **File System**.

Configuration Steps

Create Storage Classes

1. Go to **Platform Management** and click **Storage Management** > **Storage Classes** in the left navigation.
2. Click the dropdown next to **Create Storage Class** > **Create from YAML**.
3. Add the following content to the YAML file to create default storage classes as needed. The default storage class name for file storage ↗ is **efs-sc**, and for block storage ↗ is **ebs-sc**.

- EFS File Storage

  Note: Replace `<File System ID>` with the actual **File System ID**, e.g., `fileSystemId: fs-05aef9e1edd309f2b`.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: <File System ID>
  directoryPerms: "755"
```

- EBS Block Storage

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

4. Click **Create**.

   Note: If the default storage classes don't meet requirements, create new storage classes following the above steps and modify parameters as needed. See Available Storage Class Parameters.

Modify Storage Class Project Assignment

1. In the left navigation, click **Storage Management** > **Storage Classes**.
2. Click the three dots next to the storage class named **efs-sc** or **ebs-sc** > **Update Project**.
3. Select the **Project Assignment** method as needed and click **Update** to assign the storage class to projects.

Related Operations

Configure Available Storage Class Parameters

- EFS File Storage Available Parameters

Parameter	Optional Values	Default Value	Optional	Description
az		""	true	Used for cr account mounting. I specified, u the mount t associated az for cross account mounting; i specified, randomly s a mount tar cross-accou mounting.
basePath			true	Path for cre dynamically provisioned access poir not specifie access poir created unc file system directory.
directoryPerms			false	Directory permissions

Parameter	Optional Values	Default Value	Optional	Descriptio
				creating Ac Point root directory ↗.
uid			true	POSIX use creating Ac Point root directory ↗.
gid			true	POSIX grou for creating Access Poi directory ↗.
gidRangeStart		50000	true	Starting ran POSIX grou to apply wh creating ac point root directory ↗. needed if ui are set.
gidRangeEnd		7000000	true	Ending rang POSIX grou Not needed uid/gid are s
subPathPattern			true	Template fo constructing subpaths w each acces created unc dynamic

Parameter	Optional Values	Default Value	Optional	Description
				provisioning located. Ca consist of fi strings and variables, s to the "subPathPa variable in r subdir-exte provisioner Optional parameters .PVC.name .PVC.name and .PV.nar
ensureUniqueDirectory		true	true	Used when dynamic provisioning enabled. W set to true, appends UI the pattern specified in subPathPat ensure acc points don't accidentally to the same directory.**No** Only set to you're certa is the desire behavior.

Parameter	Optional Values	Default Value	Optional	Description
provisioningMode	efs-ap		false	EFS volume currently su access poir
fileSystemId			false	File system the created access poir

- EBS Block Storage Available Parameters

  Note: For performance parameters of different volume types, see Amazon EBS Volume Types ↗.

Parameter	Optional Values	Default Value	Description
"allowAutoIOPSPerGBIncrease"	true, false	false	When set to "true", the CSI driver increases volume IOPS when iopsPerGB * <volume size> is too low to meet AWS supported IOPS range. This ensures dynamic provisioning always succeeds even when user-specified PVC capacity or iopsPerGB values are too small, but may incur additional costs

Parameter	Optional Values	Default Value	Description
			as such volumes have higher IOPS than required by iopsPerGB.
"blockExpress"	true, false	false	Creates io2 Block Express volumes by raising IOPS limits for io2 volumes to 256000, but volumes created with IOPS exceeding 64000 cannot be mounted on instances that don't support io2 Block Express.
"blockSize"			Block size used when formatting the underlying filesystem. Only applies to Linux nodes with ext2, ext3, ext4, or xfs filesystem types.
"bytesPerINode"			Bytes per inode used when formatting the underlying filesystem. Only

Parameter	Optional Values	Default Value	Description
			applies to Linux nodes with ext2, ext3, or ext4 filesystem types.
"csi.storage.k8s.io/fstype"	xfs, ext2, ext3, ext4	ext4	Filesystem type to format when creating volumes. Case-sensitive.
"encrypted"	true, false	false	Whether the volume needs encryption.
"inodeSize"			Inode size used when formatting the underlying filesystem. Only applies to Linux nodes with ext2, ext3, ext4, or xfs filesystem types. Inodes are data structures in filesystems that store file and directory metadata.
"iops"			I/O operations per second, applicable to IO1, IO2, and GP3 volumes.

Parameter	Optional Values	Default Value	Description
"iopsPerGB"			I/O operations per GiB per second, applicable to IO1, IO2, and GP3 volumes.
"kmsKeyId"			Full ARN of the key to use for encrypting volumes. If not specified, AWS uses the default KMS key for the volume's region and automatically generates a key named /aws/ebs.
"numberOfINodes"			Number of inodes specified when formatting the underlying filesystem. Only applies to Linux nodes with ext2, ext3, or ext4 filesystem types.
"throughput"		125	Throughput in MiB/s. Only valid when specifying gp3 volume type. If empty, defaults to 125 MiB/s. See

Parameter	Optional Values	Default Value	Description
			Amazon EBS Volume Types↗.
"type"	io1, io2, gp2, gp3, sc1, st1, standard, sbp1, sbg1	gp3	EBS volume type.

Huawei Cloud CCE Cluster Storage Initialization Configuration

Platform integration with Huawei Cloud CCE and storage initialization configuration.

TOC

- Constraints and Limitations
- Prerequisites
- Configuration Steps
- Default Storage Class Description
- Common Issues
  - PVC Creation Failure
  - Account Overdue

Constraints and Limitations

Cluster PVC quantity has limits, and account storage capacity has quotas. You can request increases through support tickets.

Prerequisites

- If you have created a CCE cluster, import the CCE cluster (Public Cloud).

Configuration Steps

1. Go to **Platform Management** and click **Storage Management** > **Storage Classes** in the left navigation.
2. Click the three dots next to the storage class named **csi-nas** or **csi-disk** > **Update Project**. **Note**: After importing a CCE cluster, default storage classes are generated. **csi-disk** is recommended for block storage, and **csi-nas** for file storage. See Default Storage Class Description.
3. Select the **Project Assignment** method as needed and click **Update** to assign **csi-nas** or **csi-disk** storage classes to projects.

Default Storage Class Description

Storage Class Name	Storage Class Type	Description
(Recommended) csi-disk	Block Storage	Recommended for use.
(Recommended) csi-nas	File Storage	Recommended for use. Only available in regions that support csi-nas service.
csi-disk-topology	Block Storage	Automatic cloud disk topology when nodes span AZs.
csi-local	Local Storage	
csi-local-topology	Local Storage	
csi-obs	Object Storage	
csi-sfsturbo	Ultra-fast File Storage	Ultra-fast file storage cannot dynamically create persistent volumes.

Common Issues

PVC Creation Failure

The following error occurs because the PVC quantity limit has been reached. You can request an increase through support tickets:

```
message: "ShareLimitExceeded: Maximum number of shares allowed (10) excee
ded."
```

Account Overdue

PVC creation fails with the following error due to overdue payments. Resolve the payment issue before retrying:

```
message: "Your account is suspended and resources cannot be used"
```

Azure AKS Cluster Storage Initialization Configuration

Platform integration with Azure AKS and storage initialization configuration.

TOC

Constraints and Limitations
Prerequisites
Configuration Steps
Related Information
Default Storage Class Description
Available Storage Class Parameters

Constraints and Limitations

The default azurefile file storage class may not support permission modifications after mounting, which may cause some applications like PostgreSQL and Jenkins to fail to run properly.

Prerequisites

- If you have created an AKS cluster, import the Azure AKS cluster.

Configuration Steps

1. Go to **Platform Management** and click **Storage Management** > **Storage Classes** in the left navigation.
2. Click the three dots next to the storage class named **default** or **azurefile** > **Update Project**.
   Note: After importing an AKS cluster, the following default storage classes are generated. **default** is recommended for block storage, and **azurefile** for file storage. See Default Storage Class Description.
3. Select the **Project Assignment** method as needed and click **Update** to assign **default** or **azurefile** storage classes to projects.
   Note: If the default storage classes don't meet requirements, create new storage classes following the above steps and modify parameters as needed. See Available Storage Class Parameters.

Related Information

Default Storage Class Description

Storage Class Name	Storage Class Type	Description
(Recommended) azurefile	File Storage	Creates Azure file shares using Azure standard storage.
(Recommended) default	Block Storage	Creates managed disks using Azure StandardSSD storage.
azurefile-csi	File Storage	Creates Azure file shares using Azure standard storage.

Storage Class Name	Storage Class Type	Description
azurefile-csi-nfs	File Storage	Creates Azure file shares using Azure standard storage, NFS protocol.
azurefile-csi-premium	File Storage	Creates Azure file shares using Azure premium storage.
azurefile-premium	File Storage	Creates Azure file shares using Azure premium storage.
managed	Block Storage	Creates managed disks using Azure StandardSSD storage.
managed-csi	Block Storage	Creates managed disks using Azure StandardSSD locally redundant storage (LRS).
managed-csi-premium	Block Storage	Creates managed disks using Azure premium locally redundant storage (LRS).
managed-premium	Block Storage	Creates managed disks using Azure premium storage.

Available Storage Class Parameters

- For block storage optional parameters and meanings, see Create and use volumes with Azure disks in Azure Kubernetes Service (AKS)↗.
- For file storage optional parameters and meanings, see Create and use volumes with Azure Files in Azure Kubernetes Service (AKS)↗.

Google GKE Cluster Storage Initialization Configuration

Platform integration with Google GKE and storage initialization configuration.

TOC

- Constraints and Limitations
- Prerequisites
- Configuration Steps
- Related Information
  - Default Storage Class Description
  - Available Storage Class Parameters
- Common Issues
  - File Storage Type Storage Class PVC Creation Failure
  - Block Storage Type Storage Class PVC Cannot Bind Properly

Constraints and Limitations

- The default file storage type PVC has a minimum capacity of 1T. If the capacity is set to less than 1T during creation, it will automatically expand to 1T.
- Default file storage has capacity limits. You can request expansion through support tickets.

- Default file storage creation and deletion operations take considerable time. If the resource remains in `Creating` status for a long time, continue monitoring the operation.

Prerequisites

- When creating clusters, on the Google Cloud Platform **Cluster > Features** page under the **Other** section, check **Enable Compute Engine Persistent Disk CSI Driver** and **Enable Filestore CSI Driver** options.
- Enable **Cloud Filestore API** and **Google Kubernetes Engine API** on Google Cloud Platform. See Access Filestore instances using the Filestore CSI driver ↗.
- Adjust regional file storage quotas on Google Cloud Platform. See Resource quotas and limits ↗.
- If you have created a GKE cluster, import the GKE cluster.

Configuration Steps

1. Go to **Platform Management** and click **Storage Management** > **Storage Classes** in the left navigation.
2. Click the three dots next to the storage class named **standard-rwx** or **standard-rwo** > **Update Project**.
   Note: After importing a GKE cluster, default storage classes are generated. **standard-rwx** is recommended for file storage, and **standard-rwo** for block storage. See Default Storage Class Description.
3. Select the **Project Assignment** method as needed and click **Update** to assign **standard-rwx** or **standard-rwo** storage classes to projects.
   Note: If the default storage classes don't meet requirements, create new storage classes following the above steps and modify parameters as needed. See Available Storage Class Parameters.

Related Information

Default Storage Class Description

Storage Class Name	Storage Class Type	Description
(Recommended) standard-rwx	File Storage	Uses Basic HDD Filestore service tier↗.
(Recommended) standard-rwo	Block Storage	Uses balanced persistent disks.
premium-rwx	File Storage	Uses Basic SSD Filestore service tier↗.
premium-rwo	Block Storage	SSD persistent disks.
enterprise-rwx	File Storage	Uses Enterprise Filestore tier↗.
enterprise-multishare-rwx	File Storage	Uses Enterprise Filestore tier↗. See Filestore multishares for Google Kubernetes Engine↗.

Available Storage Class Parameters

- For block storage optional parameters and meanings, see Storage options↗.
- For file storage optional parameters and meanings, see Service tiers↗.

Common Issues

File Storage Type Storage Class PVC Creation Failure

- The following error occurs because Cloud Filestore API is not enabled in the project or lacks appropriate permissions. See Prerequisites to resolve:

```
failed to provision volume with StorageClass "standard-rwx": rpc error:
code = PermissionDenied desc = googlespi: Error 403: Cloud Filestore AP
I has not been used in project alauda-proj-1234 before or it is disable
d.
...
resion: SERVICE_DISABLED
```

- The following error occurs due to exceeding storage quotas. See Prerequisites to resolve:

```
failed to provision volume with StorageClass "standard-rwx": rpc error:
code = ResourceExhausted desc = googlespi: Error 429: Quora limit 'Stan
dardStorageGbPerRegion' has been exceeded. Limit 2048 in region asia-ea
st1.
 rateLimitExceeded
```

Block Storage Type Storage Class PVC Cannot Bind Properly

The following error occurs because the node's CSINode lacks configuration for the pd.csi.storage.gke.io driver. You can resolve this by restarting the **Compute Engine Persistent Disk CSI Driver**.

Note: Updating this plugin will make the cluster unavailable. The update process takes approximately 5-10 minutes.

```
Warning ProvisioningFailed 18m (x14 over 39m) pd.csi.storage.gke.io_gke-5
cb9bddae4d1430eb8ad-01f4-2084-vm_4b4e70bd-e2db-4779-9102-fee83a657ced fai
led to provision volume with StorageClass "standard": error generating ac
cessibility requirements: no available topology found
Normal ExternalProvisioning 4m35s (x143 over 39m) persistentvolume-contro
ller waiting for a volume to be created, either by external provisioner
"pd.csi.storage.gke.io" or manually created by system administrator
Normal Provisioning 3m19s (x18 over 39m) pd.csi.storage.gke.io_gke-5cb9bd
dae4d1430eb8ad-01f4-2084-vm_4b4e70bd-e2db-4779-9102-fee83a657ced External
provisioner is provisioning volume for claim "acp-gke-test/standard"
```

How to

Network Configuration for Impc **Fetch import cluster informatio** **Trust an ins**

Collect Network Data from Cust

Collect network data from custom named r

Configure audit collection for imported standa

Enable Kubernetes audit logging on imported standard Kuberr collect audit data.

Network Configuration for Imported Clusters

Before import, the required connectivity path lets the `global` cluster access the target cluster. After import, platform components in the imported cluster may also need to access the `global` cluster. Add the platform URL annotation when bidirectional connectivity is required for components such as alerting and log collection.

TOC

Prerequisites

Add The Platform URL Annotation

Prerequisites

Prepare the **domain name**, the **IP address** that the domain name points to, and the corresponding **valid certificate** that the imported cluster can access.

Note:

- This domain name must not be the same as the **platform access address**.
- Ensure that the domain's port (HTTPS port, which is the same port as the platform access address) can forward traffic to all control plane nodes of the global cluster.

Add The Platform URL Annotation

Add the annotation to the imported cluster resource.

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click **global**.
3. Click **Operations** > **CLI Tools**, and replace the relevant parameters using the following command:

```
kubectl annotate --overwrite -n cpaas-system clusters.cluster.x-k8s.io
<imported cluster name> \
        cpaas.io/platform-url=<prepared domain address, e.g.: https://w
ww.domain.cn>
```

Code example:

```
kubectl annotate --overwrite -n cpaas-system clusters.cluster.x-k8s.io
cluster-imported \
        cpaas.io/platform-url=https://www.domain.cn
```

How to fetch import cluster information?

TOC

Problem description
Prerequisites
Get cluster information
Get cluster token
Get the import cluster API server address and CA certificate

Problem description

Obtain the configuration required to connect to the import cluster so the platform can be authorized to access and manage it.

Prerequisites

- A working `kubectl` environment. **For public cloud clusters, it is strongly recommended to use the provider's CloudShell.** If CloudShell is not available, Linux or macOS is recommended.
- You have obtained the import cluster's KubeConfig file and copied it to the environment where `kubectl` is installed. **To avoid operating on the wrong environment, it is**

strongly recommended to use one of the following non-destructive approaches:

- **Backup approach**: Copy your existing kubeconfig to a safe location first: `cp "${HOME}/.kube/config" "${HOME}/.kube/config.backup"`
- **Isolated approach**: Set the `KUBECONFIG` environment variable to point to the imported kubeconfig: `export KUBECONFIG="/path/to/imported/kubeconfig"`
- **Merge approach**: Use kubectl's merge/flatten without losing existing contexts:
  1. `export KUBECONFIG="/path/to/imported/kubeconfig:${HOME}/.kube/config"`
  2. `kubectl config view --flatten > "${HOME}/.kube/merged.kubeconfig"`
  3. `export KUBECONFIG="${HOME}/.kube/merged.kubeconfig"`

Get cluster information

Get cluster token

1. Run the following commands:

```
# [Important] The following operations support bash only

# Manually create a secret, bind a service account, and generate a non-expiring token
kubectl get ns cpaas-system > /dev/null 2>&1 || kubectl create namespace cpaas-system
kubectl create serviceaccount k8sadmin -n cpaas-system
kubectl create clusterrolebinding k8sadmin --clusterrole=cluster-admin --serviceaccount=cpaas-system:k8sadmin

cat | kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: k8sadmin
  namespace: cpaas-system
  annotations:
    kubernetes.io/service-account.name: "k8sadmin"
type: kubernetes.io/service-account-token
EOF

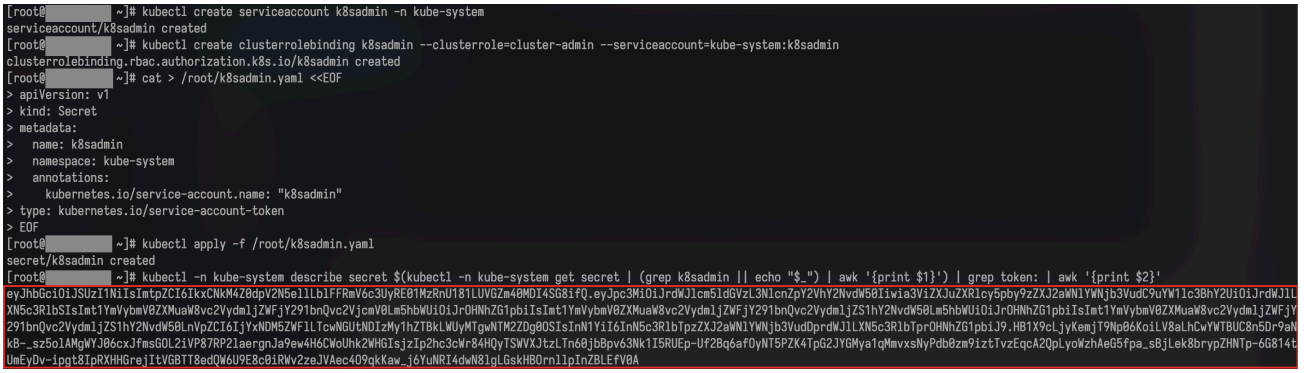
kubectl -n cpaas-system describe secret \
    $(kubectl -n cpaas-system get secret | (grep k8sadmin || echo "$_") | awk '{print $1}') \
    | grep -F 'token:' | awk '{print $2}'
```

WARNING

This procedure creates a cluster-admin credential intended to be non-expiring.

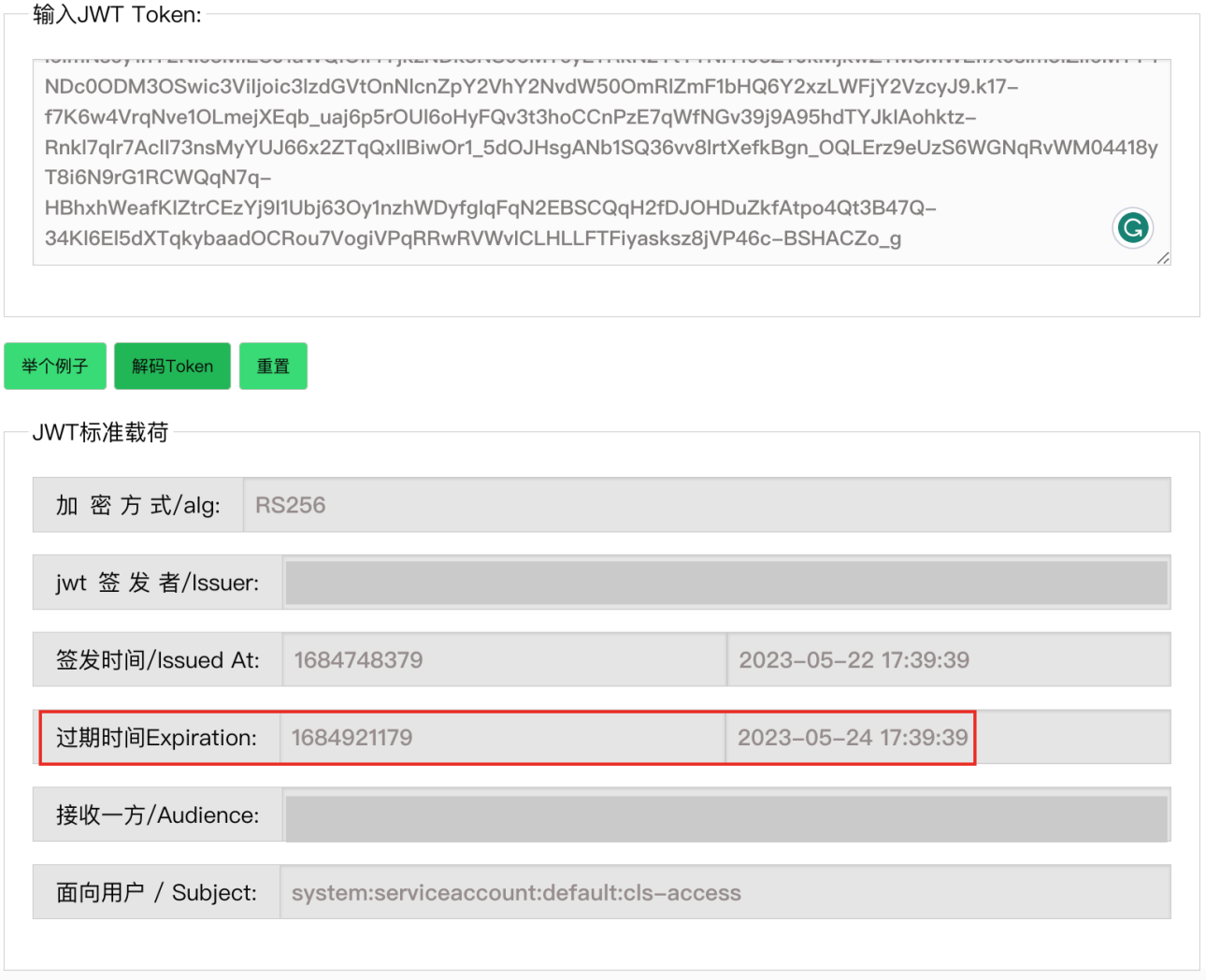
- Prefer least-privilege RBAC scoped to required resources if possible.
- Store the token securely; rotate/revoke when no longer needed.
- Limit who can read `Secret` objects in `cpaas-system`.

2. An example of the token obtained in the previous step is shown below.



3. Validate the token expiration.

   Use any tool that supports parsing JWT tokens to analyze the token and confirm its expiration time. If you can find an expiration field in the parsed result (a key containing "exp", as shown below), the platform will be unable to manage the import cluster after that time. In this case, stop and contact technical support.



TIP

The expiration is recorded as `"exp": 1684486916,` in the original JWT payload. The value is a UNIX timestamp and can be converted to UTC time.

Cleanup (revoke access) when done:

```
kubectl delete clusterrolebinding k8sadmin
kubectl -n cpaas-system delete secret k8sadmin
kubectl -n cpaas-system delete serviceaccount k8sadmin
```

Get the import cluster API server address and CA certificate

> **TIP**
>
> If you have already obtained the API server address and CA certificate using the platform's `Parse KubeConfig File` feature on the import cluster page, skip this step.

1. Run the following commands:

```
# View the import cluster API server addresses. There may be multiple a
ddresses; choose the one that fits your environment.
kubectl --kubeconfig "${KUBECONFIG:-${HOME}/.kube/config}" config view
--show-managed-fields=false --flatten --raw -ojsonpath='{$.clusters..cl
uster.server}'
addr_apiserver='<Selected API server address>'

# Get the CA certificate for the API server specified above
kubectl --kubeconfig "${KUBECONFIG:-${HOME}/.kube/config}" config view
--show-managed-fields=false --flatten --raw \
    -ojsonpath="{$.clusters[?(@.cluster.server == '${addr_apiserve
r}')].cluster.certificate-authority-data}" \
    | { base64 -d 2>/dev/null || base64 -D; }
```

How to trust an insecure image registry?

TOC

Problem description

Configure trust for an insecure image registry

Problem description

The image registry hosting platform component images may not provide HTTPS service or may not have a valid TLS certificate issued by a public certificate authority. If you trust this registry, configure your container runtime by following the steps below.

Configure trust for an insecure image registry

Notes:

- All nodes that need to use images, including newly added nodes, must be configured and have Containerd restarted.
- The configuration differs slightly between Containerd v1.4/v1.5 and v1.6. Follow the appropriate steps for your version.

1. Run the following on every node in the import cluster:

- Back up the configuration file

```
mkdir -p '/var/backup-containerd-confs/'
if ! [ -f /etc/containerd/config.toml ]; then
    echo 'Containerd config not found. Please check if containerd is
correctly installed. If you still cannot resolve the issue, contact t
echnical support.'
    exit 1
else
    cp /etc/containerd/config.toml /var/backup-containerd-confs/confi
g.toml_$(date +%F_%T)
fi
```

- Get the Containerd runtime version

```
# Get the containerd version
# Compare this version to v1.6. Choose steps accordingly
ctr --version | grep -Eo 'v[0-9]+\.[0-9]+\.[0-9]+'
```

Containerd v1.4 v1.5 configuration for insecure registries

2. Run the following on every node in the import cluster:

- Edit `/etc/containerd/config.toml`

```
# Example content to add to the config file
# Lines in brackets are sections. If the file already has sections wi
th the same name, merge their contents.
[plugins."io.containerd.grpc.v1.cri".registry]
    [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<regis
try-address>"]
            endpoint = ["https://<registry-address>", "http://<regist
ry-address>"]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."192.16
8.134.43"]
            endpoint = ["https://192.168.134.43", "http://192.168.13
4.43"]
    [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."<regis
try-address>".tls]
            insecure_skip_verify = true
        [plugins."io.containerd.grpc.v1.cri".registry.configs."192.16
8.134.43".tls]
            insecure_skip_verify = true
```

- Restart Containerd.

```
systemctl daemon-reload && systemctl restart containerd
```

Containerd v1.6 configuration for insecure registries

2. Run the following on every node in the import cluster:

- Check whether `config_path` exists in the config.

```
if ! grep -qF 'config_path' /etc/containerd/config.toml; then
    if grep -qE '\[plugins."io.containerd.grpc.v1.cri".registry.(mirr
ors|configs)(\.|\])' /etc/containerd/config.toml; then
        echo 'Follow the steps in "Containerd v1.4 v1.5 configuration
for insecure registries".'
    else
        cat >> /etc/containerd/config.toml << 'EOF'
[plugins."io.containerd.grpc.v1.cri".registry]
    config_path = "/etc/containerd/certs.d/"
EOF
    fi
fi

config_path_var=$(grep -F '/etc/containerd/certs.d' /etc/containerd/c
onfig.toml)
if [ -z "$config_path_var" ]; then
   echo 'The value of config_path in the file is unexpected. Please c
heck!'
   exit 1
fi
```

- Create the `hosts.toml` file.

  If the previous command printed `Follow the steps in "Containerd v1.4 v1.5 configuration for insecure registries".`, see *Containerd v1.4 v1.5 configuration for insecure registries*.

```
REGISTRY='<registry address obtained in the "Get the registry addres
s" section>'

mkdir -p "/etc/containerd/certs.d/$REGISTRY/"
cat > "/etc/containerd/certs.d/$REGISTRY/hosts.toml" << EOF
server = "$REGISTRY"
[host."http://$REGISTRY"]
  capabilities = ["pull", "resolve", "push"]
  skip_verify = true
[host."https://$REGISTRY"]
  capabilities = ["pull", "resolve", "push"]
  skip_verify = true
EOF
```

- Restart Containerd.

```
systemctl daemon-reload && systemctl restart containerd
```

Collect Network Data from Custom Named Network Cards

TOC

Scenario Description

Procedure

Scenario Description

After creating a workload cluster, the platform monitoring can only recognize network card names matching patterns like `eth.*|en.*|wl.*|ww.*` by default. For user-defined network card names, network traffic data cannot be viewed on the monitoring page. To address this, the platform supports modifying relevant resource parameters to manually capture network card traffic data.

Procedure

1. Log in to the control plane node of the global cluster and execute the following commands using kubectl.
2. First, find the moduleinfo resource name corresponding to the workload cluster in the global cluster:

```
kubectl get moduleinfo | grep -E 'prometheus|victoriametrics'
```

Example output:

```
global-6448ef7f7e5e3924c1629fad826372e7     global     prometheus
prometheus                          Running   v3.15.0-zz231204040711-9d
1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz2312040407
11-9d1fc12474c2
ovn-0954f21f0359720e8c115804376b3e7e        ovn        prometheus
prometheus                          Running   v3.15.0-zz231204040711-9d
1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz2312040407
11-9d1fc12474c2
```

3. Edit the moduleinfo resource of the workload cluster, replacing `ovn-0954f21f0359720e8c115804376b3e7e` with the workload cluster moduleinfo resource name from the previous step:

```
kubectl edit moduleinfo ovn-0954f21f0359720e8c115804376b3e7e
```

4. Add the valuesOverride field and modify the field and regular expression according to the comment information:

```
spec:
  valuesOverride: # If this field does not exist, you need to add the v
aluesOverride field and the following parameters under spec
    ait/chart-cpaas-monitor:
      ovn: # Replace with the workload cluster name
        indicator:
          networkDevice: eth.*|em.*|en.*|wl.*|ww.*|[A-Z].*i|custom_inte
rface # Replace custom_interface with a custom regular expression to en
sure correct network card name matching
```

5. Wait 10 minutes, then check the network-related charts on the node monitoring page to ensure the changes take effect.

How to configure audit collection for imported standard Kubernetes clusters?

TOC

Scenario Description

Prerequisites

Procedure

Scenario Description

After a standard Kubernetes cluster is imported into the platform, you must enable Kubernetes API server audit logging on the cluster before the platform can collect audit data from that cluster.

Use this procedure for standard Kubernetes clusters whose control plane nodes are managed by you, such as kubeadm-based clusters. It does not apply to provider-managed cloud Kubernetes services where you cannot log in to or modify control plane nodes.

Prerequisites

- The standard Kubernetes cluster has already been imported into the platform.

- You can log in to every control plane node in the cluster.
- The cluster uses the standard kubeadm-style API server static Pod manifest path: `/etc/kubernetes/manifests/kube-apiserver.yaml`.

Procedure

1. Create a local `policy.yaml` file for the audit policy.
   Set `apiVersion` according to the Kubernetes version:

   - Kubernetes earlier than 1.24: `audit.k8s.io/v1beta1`
   - Kubernetes 1.24 and later: `audit.k8s.io/v1`

   Use the following content:

```
apiVersion: audit.k8s.io/v1
kind: Policy
omitStages:
  - "RequestReceived"
rules:
  - level: None
    users:
      - system:kube-controller-manager
      - system:kube-scheduler
      - system:serviceaccount:kube-system:endpoint-controller
    verbs: ["get", "update"]
    namespaces: ["kube-system"]
    resources:
      - group: ""
        resources: ["endpoints"]
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
  - level: None
    resources:
      - group: ""
        resources: ["events"]
  - level: None
    resources:
      - group: "devops.alauda.io"
  - level: None
    verbs: ["get", "list", "watch"]
  - level: None
    namespaces:
      - kube-system
      - cpaas-system
      - alauda-system
      - istio-system
      - kube-node-lease
    resources:
      - group: "coordination.k8s.io"
        resources: ["leases"]
  - level: None
    resources:
      - group: "authorization.k8s.io"
        resources: ["subjectaccessreviews", "selfsubjectaccessreviews"]
```

```
    - group: "authentication.k8s.io"
      resources: ["tokenreviews"]
  - level: Metadata
    resources:
    - group: ""
      resources: ["secrets", "configmaps"]
  - level: RequestResponse
    resources:
    - group: ""
    - group: "aiops.alauda.io"
    - group: "apps"
    - group: "app.k8s.io"
    - group: "authentication.istio.io"
    - group: "auth.alauda.io"
    - group: "autoscaling"
    - group: "asm.alauda.io"
    - group: "clusterregistry.k8s.io"
    - group: "crd.alauda.io"
    - group: "infrastructure.alauda.io"
    - group: "monitoring.coreos.com"
    - group: "networking.istio.io"
    - group: "networking.k8s.io"
    - group: "portal.alauda.io"
    - group: "rbac.authorization.k8s.io"
    - group: "storage.k8s.io"
    - group: "tke.cloud.tencent.com"
    - group: "devopsx.alauda.io"
    - group: "core.katanomi.dev"
    - group: "deliveries.katanomi.dev"
    - group: "integrations.katanomi.dev"
    - group: "builds.katanomi.dev"
    - group: "operators.katanomi.dev"
    - group: "tekton.dev"
    - group: "operator.tekton.dev"
    - group: "eventing.knative.dev"
    - group: "flows.knative.dev"
    - group: "messaging.knative.dev"
    - group: "operator.knative.dev"
    - group: "sources.knative.dev"
    - group: "operator.devops.alauda.io"
  - level: Metadata
```

TIP

If the cluster version is earlier than 1.24, change only the `apiVersion` field to `audit.k8s.io/v1beta1`. The rest of the policy content stays the same.

2. Upload `policy.yaml` to `/etc/kubernetes/audit/` on every control plane node.

> **WARNING**
>
> - If the cluster has multiple control plane nodes, upload the file to every node.
> - Create the directory manually if it does not exist: `/etc/kubernetes/audit/`

3. Update `/etc/kubernetes/manifests/kube-apiserver.yaml` on every control plane node.
   Add or update the following audit-related flags in `spec.containers[].command`:

Flag	Required	Description
`--audit-policy-file`	Yes	Must be set to `/etc/kubernetes/audit/policy.yaml`.
`--audit-log-format`	Yes	Must be set to `json`.
`--audit-log-path`	Yes	Must be set to `/etc/kubernetes/audit/audit.log`.
`--audit-log-mode`	No	Recommended value: `batch`.
`--audit-log-maxsize`	No	Maximum audit log file size in MiB. Recommended value: `200`.
`--audit-log-maxbackup`	No	Number of retained audit log files. Recommended value: `2`.

Example:

```
- --audit-log-format=json
- --audit-log-maxbackup=2
- --audit-log-maxsize=200
- --audit-log-mode=batch
- --audit-log-path=/etc/kubernetes/audit/audit.log
- --audit-policy-file=/etc/kubernetes/audit/policy.yaml
```

4. Add the audit directory mount configuration to the same `kube-apiserver.yaml` file.
   Add the following item under `spec.containers[].volumeMounts`:

```
- mountPath: /etc/kubernetes/audit
  name: k8s-audit
```

   Add the following item under `spec.volumes`:

```
- hostPath:
    path: /etc/kubernetes/audit
    type: DirectoryOrCreate
  name: k8s-audit
```

   WARNING

   - Update the manifest on every control plane node when the cluster has multiple control plane nodes.
   - The `volumeMounts[].name` value must match the corresponding `volumes[].name` value.
   - Do not change the mount path `/etc/kubernetes/audit`.

5. Save the file and verify that the configuration takes effect.
   Check whether `/etc/kubernetes/audit/audit.log` is generated on each control plane node. If the file exists and contains audit records, the configuration is effective.

```
ls -l /etc/kubernetes/audit/audit.log
tail -n 20 /etc/kubernetes/audit/audit.log
```

How to

Access a Cluster with KubeCor Add External Address for Built- Updating Pu

Access a Cluster with KubeConfig

TOC

- Introduction
- Scenarios
- Choose the Access Path
- Prerequisites
- Procedure
  - Download and Use the Platform-Provided KubeConfig
  - Download the KubeConfig file
  - Select the KubeConfig file
  - Validate cluster access
- Build a Token-Based KubeConfig with Your Own ACP API Token
  - Create a ACP API token
  - Prepare a KubeConfig template
  - Set the ACP API token in the KubeConfig file
  - Validate the effective permissions
- Use a Dedicated Account for Pipeline or Connector Access
- Lifecycle and Revocation
- Verification
- Next Steps

Introduction

Use KubeConfig when you need local cluster access from ACP CLI (`ac`), `kubectl`, `helm`, `client-go`, or similar Kubernetes clients. The procedures on this page apply to the `global` cluster as well as to workload clusters.

A KubeConfig file is a YAML client configuration file. It is not a Kubernetes resource. A KubeConfig file typically defines the target cluster endpoint, the client credential, and the context that combines them.

ACP CLI (`ac`) uses standard kubeconfig and provides a kubectl-like experience. It is compatible with `kubectl` workflows and also adds platform-specific context, cluster, and session operations. If you already use `ac login`, ACP CLI can configure kubeconfig and create contexts automatically. The access paths below include downloaded KubeConfig files and manually assembled token-based KubeConfig files.

Scenarios

Use these access paths for the following scenarios:

- A cluster administrator needs a ready-to-use KubeConfig for cluster administration, troubleshooting, or inspection.
- A developer or project user needs a KubeConfig that matches their own ACP identity and RBAC permissions.
- A pipeline or connector needs an independent credential that is separate from a personal day-to-day account.

Choose the Access Path

Use the following table to choose the appropriate access path:

Access path	Credential source	Effective permissions	Lifetime control	Best fit
Platform-downloaded KubeConfig	Credential embedded in the KubeConfig file downloaded from the platform	Determined by the credential embedded in the downloaded file	Controlled by the current platform behavior for the downloaded file	Cluster administrators and template preparation
Token-based KubeConfig	ACP API token created for a user account	Limited to the identity and RBAC scope of the token owner	Controlled by the token expiration or revocation state	Developers, project users, and automation accounts

> **Current behavior**
>
> Currently, the KubeConfig file downloaded from ACP defaults to a 10-year validity period. ACP does not currently provide CA replacement for this path and does not support user-defined cluster CA for this path.

If you need stricter identity isolation or shorter lifecycle control, use a token-based KubeConfig instead of long-term reuse of the downloaded file.

Prerequisites

Before you begin, ensure the following conditions are met:

- You can sign in to ACP and access the target cluster.
- You have ACP CLI (`ac`) or `kubectl` installed on the local machine where you will use the KubeConfig file.
- If you want to use a token-based KubeConfig, you can create a ACP API token from **Profile** > **API Tokens**. For details, see Introduction.
- If you want to use KubeConfig for a pipeline or connector, prepare a dedicated account and grant only the required permissions before creating its token.

Procedure

Download and Use the Platform-Provided KubeConfig

Use this path when you need direct cluster access from the platform-provided KubeConfig file.

1 Download the KubeConfig file

In the **Administrator** view, go to **Clusters** > **Clusters**, select the target cluster, open the cluster details page, click **Actions**, and select **Download Kubeconfig**.

Save the downloaded file to a local path such as `~/.kube/<cluster-name>.yaml`.

2 Select the KubeConfig file

Point ACP CLI (`ac`) or `kubectl` to the downloaded file:

```
export KUBECONFIG="$HOME/.kube/<cluster-name>.yaml"
kubectl config get-contexts
kubectl config current-context
ac config current-context
```

If you manage multiple clusters, use `kubectl config use-context <context-name>` or `ac config use-context <context-name>` to switch to the correct context before running commands. If you use ACP CLI sessions, you can also use `ac config use-cluster <cluster-name>` to switch clusters within the current platform session.

3 Validate cluster access

Run read-only commands to confirm that the KubeConfig file works as expected:

```
kubectl get nodes
kubectl get pods -A
ac get nodes
ac get pods -A
```

If the commands return the expected cluster resources, the KubeConfig file is valid for direct use.

Use this downloaded file directly when you need cluster administration. You can also use it as a template for a token-based KubeConfig that works with both `ac` and `kubectl`.

Build a Token-Based KubeConfig with Your Own ACP API Token

Use this path when you want the KubeConfig permissions to match a specific ACP user or automation account.

This path does not elevate permissions. The KubeConfig file only carries the connection and authentication settings. The effective access boundary still depends on the token owner and that identity's RBAC assignments.

1 Create a ACP API token

Open **Profile** > **API Tokens**, create a new token, and set an expiration time that matches your use case.

Store the token securely. If you plan to use it for automation, avoid using a personal daily account.

2 Prepare a KubeConfig template

Use one of the following methods:

- Download the platform-provided KubeConfig file and save a separate copy for token-based access.
- Create a standard KubeConfig file manually.

If you create the file manually, use the target cluster endpoint in the following format:

```
https://<platform-access-address>/kubernetes/<cluster-name>
```

If you do not want to build the cluster section from scratch, download the platform-provided KubeConfig first and reuse its cluster and CA fields as the template.

3 Set the ACP API token in the KubeConfig file

Replace the `users[].user.token` value with the ACP API token that you created for the target user or automation account.

The following example shows a token-based KubeConfig:

KubeConfig.yaml

```yaml
apiVersion: v1
kind: Config
clusters:
  - name: <cluster-name>
    cluster:
      server: https://<platform-access-address>/kubernetes/<cluster-name>
      certificate-authority-data: <base64-encoded-ca-data>
users:
  - name: <user-name>
    user:
      token: <platform-api-token>
contexts:
  - name: <context-name>
    context:
      cluster: <cluster-name>
      user: <user-name>
      namespace: <namespace>
current-context: <context-name>
```

`namespace` is optional. It sets the default namespace for commands, but it does not increase the permission scope of the token.

4 Validate the effective permissions

Use the new KubeConfig file and verify that the returned permissions match the intended role:

```
export KUBECONFIG="$HOME/.kube/<user-or-automation>.yaml"
kubectl auth can-i get pods -n <namespace>
kubectl get pods -n <namespace>
ac config current-context
ac get pods -n <namespace>
```

If the command returns `no` for `kubectl auth can-i`, grant the required role or use a different account. Do not treat KubeConfig changes as a substitute for RBAC changes.

Use a Dedicated Account for Pipeline or Connector Access

For pipeline or connector access, use a dedicated account instead of a personal user account:

1. Create a dedicated account for the pipeline or connector.
2. Grant only the required project-level or tenant-level permissions to that account.
3. Create a ACP API token for that account.
4. Build a token-based KubeConfig by using that token.
5. Store the token or KubeConfig file in the secret management system of the automation platform.
6. Rotate the credential by creating a new token, updating the secret, and revoking the old token.

This pattern provides an independent credential for automation, but it is not a native project-level KubeConfig object in ACP.

Lifecycle and Revocation

The lifetime behavior depends on which access path you use:

Access path	What controls validity	Current behavior
Platform-downloaded KubeConfig	The current platform behavior for the downloaded file	The downloaded KubeConfig currently defaults to a 10-year validity period. ACP does not currently provide CA replacement or user-defined cluster CA for this path.
Token-based KubeConfig	The ACP API token in `users[].user.token`	The KubeConfig remains usable only while the token is valid. When the token expires or is revoked, the KubeConfig also stops working.

If you need a shorter validity period or easier revocation, prefer the token-based path.

Verification

After you prepare the KubeConfig file, perform the following checks:

1. Run `kubectl config get-contexts` and confirm that the expected context exists.
2. Run a read-only command such as `kubectl get pods -A` or `kubectl get pods -n <namespace>` and confirm that the response matches the intended access scope.
3. For a token-based KubeConfig, run `kubectl auth can-i <verb> <resource> -n <namespace>` and confirm that the result matches the token owner's role.
4. If you use ACP CLI, run `ac config current-context` or `ac namespace` and confirm that the active context matches the intended cluster and namespace.
5. If you use ACP CLI, run `ac get pods -n <namespace>` or `ac get nodes` and confirm that the response matches the intended access scope.

Next Steps

- For ACP API token management, see Introduction.
- For ACP CLI basics, see Getting Started with CLI.
- For ACP CLI context and session management, see Managing CLI Profiles.

- For ACP CLI and kubectl compatibility, see Usage of ac and kubectl Commands.

Add External Address for Built-in Registry

TOC

Overview

Prerequisites

Procedure

Configure Certificate and Routing Rules for the Platform Registry

Overview

When the `global` cluster uses the `Platform Built-in` registry, workload clusters typically also use this registry to pull images. The registry not only serves components within the `global` cluster but must also be accessible to workload cluster nodes.

In certain scenarios, workload cluster nodes cannot directly access the `global` cluster's registry address - for example, when the `global` cluster is in a private data center while workload clusters are in public clouds or edge environments.

Configure an externally accessible address for the platform's default registry so workload clusters can pull images.

Prerequisites

Before you begin, prepare the following:

- A domain name accessible by workload cluster nodes
- The IP address that the domain name points to
- A valid SSL certificate for the domain name

> **WARNING**
>
> - The domain name must be different from the platform access address
> - Ensure the domain's IP address can forward traffic to all control plane nodes of the `global` cluster

Procedure

Configure Certificate and Routing Rules for the Platform Registry

1. Copy the domain's valid certificate to any control plane node of the `global` cluster
2. Create a TLS secret containing the domain certificate:

   ```
   kubectl create secret tls registry-address.tls --cert=<certificate-file
   name> --key=<key-filename> -n kube-system
   ```

   Example:

   ```
   kubectl create secret tls registry-address.tls --cert=custom.crt --key=
   custom.key -n kube-system
   ```

   Note: After creating the certificate, monitor the expiration date of the **registry-address.tls** secret in the **kube-system** namespace of the `global` cluster. Replace the certificate before it expires.
3. Create ingress rules on any control plane node of the `global` cluster:

```
REGISTRY_DOMAIN_NAME=<www.registry.com> # Replace with your accessible
domain name
cat << EOF | kubectl create -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
  name: registry-address
  namespace: kube-system
  labels:
    service_name: registry
spec:
  rules:
    - host: $REGISTRY_DOMAIN_NAME
      http:
        paths:
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/_catalog
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/tags/list
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
```

```
            path: /v2/.+/manifests/[A-Za-z0-9_+.-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobs/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobs/uploads/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /auth/token
            pathType: ImplementationSpecific
  tls:
    - secretName: registry-address.tls
      hosts:
        - $REGISTRY_DOMAIN_NAME
EOF
```

A response similar to `... created` indicates successful ingress creation.

4. Check if a Registry Service resource exists:

```
kubectl -n kube-system get svc | grep registry
```

If the Service doesn't exist, create it with:

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Service
metadata:
  labels:
    name: registry
    service_name: registry
  name: registry
  namespace: kube-system
spec:
  ports:
    - protocol: TCP
      port: 443
      targetPort: 60080
  selector:
    component: registry
  type: ClusterIP
EOF
```

5. Test the configuration by pulling an image from the registry using the domain name:

```
crictl pull <registry-domain-name>/automation/qaimages:helloworld
```

Or

```
podman pull <registry-domain-name>/automation/qaimages:helloworld
```

Updating Public Repository Credentials

TOC

Overview

Procedure

Overview

The `Public Repository` is a platform-provided image registry service available on the public internet. When you want your clusters to use the `Public Repository` as their image registry, you need to update the built-in `public-registry-credential` Cloud Credentials. This ensures your platform has permission to pull images from the public registry.

`public-registry-credential` is the platform's built-in registry credential, and its effect is not limited to clusters configured to use the `Public Repository`. When the platform registry requires authentication, an Alauda OS cluster provisioned through an infrastructure provider resolves its platform image source from this credential unless it references one of its own. Updating this credential therefore affects those clusters as well. (When the platform registry is anonymous, this credential is not populated and those clusters resolve their registry from the address recorded at installation time, so an update here does not reach them.)

List the clusters that would be affected before you change the contents — they are the ones with no registry reference of their own:

```
kubectl get cluster.platform.tkestack.io \
  -o custom-columns='NAME:.metadata.name,REGISTRY_REFERENCE:.metadata.labels.cpaas\.io/registry-reference'
```

Any cluster showing `<none>` in the second column inherits this credential.

> **Giving a cluster its own registry credential**
>
> A cluster can be bound to a registry credential of its own instead of inheriting this one, so that it pulls platform images from a registry closer to it. The binding is chosen when the cluster is created and cannot be changed afterwards. See Choose the Image Registry for a Workload Cluster.

Procedure

1. Log in to the **Alauda Customer Portal** and download your organization's authentication file from the **Enterprise Management** section located in the **User Information** dropdown in the upper right corner.
2. Navigate to **Clusters** > **Cloud Credential** in the left navigation bar of the **Administrator** console.
3. Locate the cloud credential named `public-registry-credential` and click **Update** from the dropdown menu on the right.
4. In the **Upload Public Repository Address** section, upload the authentication file you downloaded from the **Alauda Customer Portal**.
5. Click **Update** to apply the changes.