Menu

Overview

Architecture

Release Notes

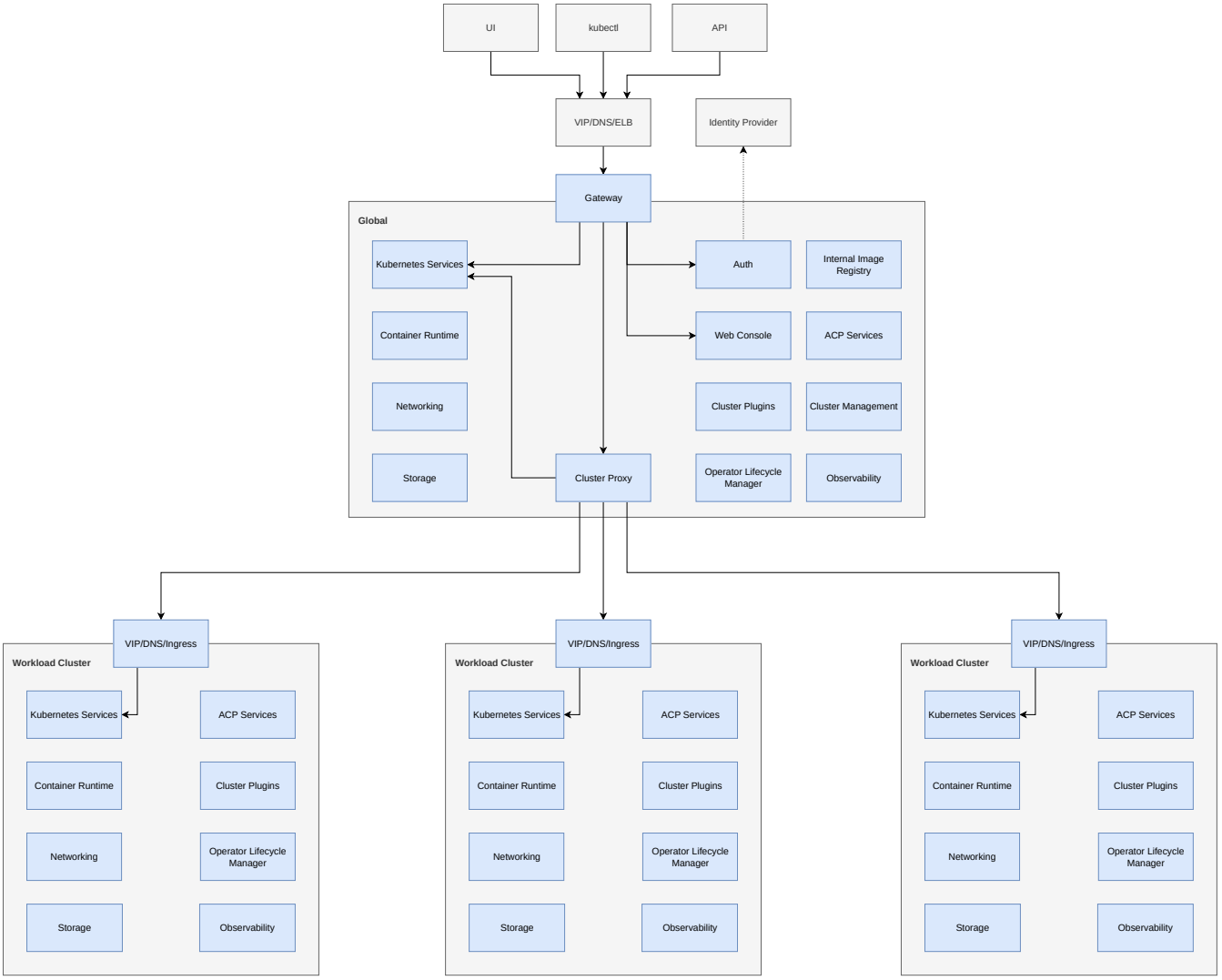
Architecture

TOC

Introduction to Alauda Container Platform

The Alauda Container Platform (ACP) provides an enterprise-grade Kubernetes-based platform that enables organizations to build, deploy, and manage applications consistently across hybrid and multi-cloud environments. ACP integrates core Kubernetes capabilities with enhanced management, observability, and security services, offering a unified control plane and flexible workload clusters.

The architecture follows a **hub-and-spoke** model, consisting of a `global` cluster and multiple workload clusters. This design provides centralized governance while allowing independent workload execution and scalability.



Core Architectural Components

Global Cluster

The `global` cluster serves as the centralized management and control hub of ACP. It provides platform-wide services such as authentication, policy management, cluster lifecycle operations, and observability. It's also a central hub for multi-cluster management and provides cross-cluster functionality.

Key components include:

- **Gateway** Acts as the main entry point to the platform. It manages API requests from the UI, CLI (kubectl), and automation tools, routing them to appropriate backend services.
- **Authentication and Authorization (Auth)** Integrates with external Identity Providers (**IdPs**) to provide Single Sign-On (SSO) and RBAC-based access control.

- **Web Console** Provides a web-based interface for ACP. It interfaces with platform APIs through the gateway.
- **Cluster Management** Handles the registration, provisioning, and lifecycle management of workload clusters.
- **ACP Services**
- **Operator Lifecycle Manager (OLM) and Cluster Plugins** Manages the installation, updates, and lifecycle of operators and cluster extensions.
- **Internal Image Registry** Offers an out-of-box integrated container image repository with role-based access.
- **Observability** Provides centralized logging, metrics, and tracing for both the `global` and workload clusters.
- **Cluster Proxy** Enables secure communication between the `global` cluster and workload clusters.

Workload Cluster

Workload clusters are Kubernetes-based environments managed by the `global` cluster. Each workload cluster runs isolated application workloads and inherits governance and configuration from the central control plane.

External Integrations

- **Identity Provider (IdP)** Supports federated authentication via standard protocols (OIDC, SAML) for unified user management.
- **API and CLI Access** Users can interact with ACP through RESTful APIs, the web console, or command-line tools like `kubectl` and `ac`.
- **Load Balancer (VIP/DNS/SLB)** Provides high availability and traffic distribution to the Gateway and ingress endpoints of the `global` and workload Clusters.

Scalability and High Availability

ACP is designed for horizontal scalability and high availability:

- Each component can be deployed redundantly to eliminate single points of failure.
- The `global` cluster supports managing dozens to hundreds of workload clusters.
- Workload clusters can scale independently according to workload demand.
- The use of VIP/DNS/Ingress ensures seamless routing and failover.

Functional Perspective

Alauda Container Platform (ACP)'s complete functionality consists of **ACP Core** and extensions based on two technical stacks: **Operator** and **Cluster Plugin**.

- **ACP Core**

  The minimal deliverable unit of ACP, providing core capabilities such as cluster management, container orchestration, projects, and user administration.

  - Meets the highest security standards
  - Delivers maximum stability
  - Offers the longest support lifecycle

- **Extensions**

  Extensions in both the Operator and Cluster Plugin stacks can be classified into:

  - **Aligned** – Life cycle strategy consisting of multiple maintenance streams, with alignment to ACP.
  - **Agnostic** – Life cycle strategy consisting of multiple maintenance streams, released independently from ACP.

  For more details about extensions, see Extend.

Technical Perspective

Platform Component Runtime All platform components run as containers within a Kubernetes management cluster (the `global` cluster).

High Availability Architecture

- The `global` cluster typically consists of at least three control plane nodes and multiple worker nodes
- High availability of etcd is central to cluster HA; see *Key Component High Availability Mechanisms* for details
- Load balancing can be provided by an external load balancer or a self-built VIP inside the cluster

Request Routing

- Client requests first pass through the load balancer or self-built VIP
- Requests are forwarded to **ALB** (the platform's default Kubernetes Ingress Gateway) running on designated ingress nodes (or control-plane nodes if configured)
- ALB routes traffic to the target component pods according to configured rules

Replica Strategy

- Core components run with at least two replicas
- Key components (such as registry, MinIO, ALB) run with three replicas

Fault Tolerance & Self-healing

- Achieved through cooperation between kubelet, kube-controller-manager, kube-scheduler, kube-proxy, ALB, and other components
- Includes health checks, failover, and traffic redirection

Data Storage & Recovery

- Control-plane configuration and platform state are stored in etcd as Kubernetes resources
- In catastrophic failures, recovery can be performed from etcd snapshots

Primary / Standby Disaster Recovery

- Two separate `global` clusters: **Primary Cluster** and **Standby Cluster**
- The disaster recovery mechanism is based on real-time synchronization of etcd data from the Primary Cluster to the Standby Cluster.

- If the Primary Cluster becomes unavailable due to a failure, services can quickly switch to the Standby Cluster.

Key Component High Availability Mechanisms

etcd

- Deployed on three (or five) control plane nodes
- Uses the RAFT protocol for leader election and data replication
- Three-node deployments tolerate up to one node failure; five-node deployments tolerate up to two
- Supports local and remote S3 snapshot backups

Monitoring Components

- **Prometheus**: Multiple instances, deduplication with Thanos Query, and cross-region redundancy
- **VictoriaMetrics**: Cluster mode with distributed VMStorage, VMInsert, and VMSelect components

Logging Components

- **Nevermore** collects logs and audit data
- **Kafka / Elasticsearch / Razor / Lanaya** are deployed in distributed and multi-replica modes

Networking Components (CNI)

- **Kube-OVN / Calico / Flannel**: Achieve HA via stateless DaemonSets or triple-replica control plane components

ALB

- Operator deployed with three replicas, leader election enabled
- Instance-level health checks and load balancing

Self-built VIP

- High-availability virtual IP based on Keepalived
- Supports heartbeat detection and active-standby failover

Harbor

- ALB-based load balancing
- PostgreSQL with Patroni HA
- Redis Sentinel mode
- Stateless services deployed in multiple replicas

Registry and MinIO

- Registry deployed with three replicas
- MinIO in distributed mode with erasure coding, data redundancy, and automatic recovery

Release Notes

TOC

Known Issues

4.2.0

Features and Enhancements

Support for Kubernetes 1.33

ACP now supports **Kubernetes 1.33**, delivering the latest upstream features, performance improvements, and security enhancements from the Kubernetes community.

ACP CLI (ac)

The new **ACP CLI (ac)** enables you to develop, build, deploy, and run applications on ACP with a seamless command-line experience.

Key capabilities include:

- **kubectl-compatible commands**
- **Integrated authentication** with ACP platform environments
- **Unified session management** across multiple environments
- **ACP-specific extensions** for platform access and cross-environment workflows

For full feature details, see: ACP CLI (ac)

Hosted Control Plane (HCP)

Released:

- **Alauda Container Platform Kubeadm Provider**
- **Alauda Container Platform Hosted Control Plane**
- **Alauda Container Platform SSH Infrastructure Provider**

Lifecycle: *Agnostic* (released asynchronously with ACP)

Hosted Control Plane decouples the control plane from worker nodes by hosting each cluster's control plane as containerized components within a management cluster. This architecture reduces resource usage, speeds up cluster creation and upgrades, and provides improved scalability for large multi-cluster environments.

For more information, see: About Hosted Control Plane

Enhanced User Permission Management

We've optimized RBAC management with the following enhancements to improve usability and maintainability:

- **Platform Role Management:**
  - **UI-based permission customization deprecated:** Platform roles no longer support custom permission configuration through the web console. All role permissions must be configured via YAML files.
  - **Backward compatibility maintained:** Existing platform preset roles and roles defined in previous versions remain fully functional. Users can continue to assign these roles to users and grant permissions as before.
- **Kubernetes Role Management:**
  - **Native Kubernetes role management:** A dedicated management interface for Kubernetes Role and ClusterRole resources is now available in the platform console, enabling direct association of Kubernetes roles with users and permission assignment.
  - **Modular permission definitions:** Platform plugin resource permissions will be progressively migrated to independent Role and ClusterRole resources, providing better isolation and easier management.

Enhanced Pod Security Policies with Kyverno

We've strengthened workload security capabilities through the Kyverno policy engine:

- Ready-to-Use Security Templates: 8 validated security policy templates built into the console, covering Pod Security Standards levels including Privileged, Baseline, Nonroot, and Restricted

- One-Click Configuration: Quickly create policies from templates in the business view without manual YAML writing, effective immediately in specified namespaces

Next-Generation Gateway API powered by Envoy Gateway

This release introduces a new Gateway API implementation based on Envoy Gateway. It provides a unified L7 traffic entry, stays aligned with the community Gateway API specification, and lays the foundation for richer traffic policies and ecosystem integrations.

Domain-Based Rules for Egress Firewall

Egress Firewall now supports allow/deny rules based on domain names instead of only IP addresses. This enables fine-grained outbound access control for public SaaS services and external resources whose IP addresses change frequently.

New Endpoint Health Checker for Faster Failover

A new Endpoint Health Checker is introduced to detect failures such as node crashes and network partitions more quickly and to remove unhealthy backends in time. This significantly shortens traffic failover duration and reduces the risk of service interruption.

New Local Storage Operator for Easier Ceph/TopoLVM Management

The newly introduced **Local Storage** (Alauda Build of Local Storage) Operator greatly simplifies deployment and disk management for Ceph and TopoLVM. During deployment, you can list all available disks across the cluster, including model, capacity, and other key attributes, and select which disks to bring under management. For disk binding, Ceph and TopoLVM now prefer using device IDs rather than mount paths, preventing storage issues caused by device name changes after node reboot or device re-detection.

Other Key Changes

Lifecycle Change for Logging Plugins

The lifecycle status for logging-related plugins has been changed from *Aligned* to *Agnostic* (released asynchronously with ACP).

Affected plugins:

- **Alauda Container Platform Log Essentials** (new in this release)
- **Alauda Container Platform Log Storage for ClickHouse**
- **Alauda Container Platform Log Storage for Elasticsearch**
- **Alauda Container Platform Log Collector**

For more information, see: About Logging Service

Enhanced Default Security Level for Namespaces

Starting from v4.2.0, the default PSA policy for newly created namespaces (via web console or CLI) is changed from Baseline to Restricted.

- Baseline: Prohibits known privilege escalations, provides moderate security
- Restricted: Follows Pod security best practices with strictest requirements

> **WARNING**
>
> The Restricted policy enforces very strict configuration requirements for Pods. If your business requires capabilities such as privileged mode, running as the root user, mounting host paths, or using the host network, these workloads will fail to run in namespaces that default to the Restricted policy.

Impact Analysis:

- This change only affects newly created namespaces
- Workloads requiring privileged capabilities (e.g., root user, hostPath mounts) will not run directly

Recommended Solutions:

- Modify your application configuration to meet the security requirements of the Restricted policy
- Manually set the namespace policy back to Baseline if necessary

MinIO in Maintenance Mode

The **MinIO** (Alauda Build of MinIO) has entered **maintenance mode**. Only security fixes will be provided in the future, and no new features are planned. Existing MinIO clusters can continue to run, while new object storage requirements should prefer Ceph Object as the recommended solution.

Calico in Maintenance Mode

The **Calico** (Alauda Container Platform Networking for Calico) CNI plugin has entered **maintenance mode**. We will only address security-related issues, and it is no longer the default recommended network option. Existing Calico clusters remain supported, while new clusters should use kube-ovn as the standard CNI.

Ingress Nginx Switched to Operator

The **Ingress Nginx** (Alauda Build of Ingress Nginx) has been migrated from a cluster plug-in to an Operator-based deployment and management model. Existing Ingress resources will continue to work after the upgrade, and subsequent operations are expected to be carried out through the Operator. Although the upstream community version of Ingress Nginx is no longer updated, we will continue to provide bug fixes and security patches for this distribution.

Deprecated and Removed Features

Kubernetes Version Upgrade Policy Update

Starting from **ACP 4.2**, upgrading the Kubernetes version is **no longer optional**. When performing a cluster upgrade, the Kubernetes version must be upgraded together with other platform components. This change ensures version consistency across the cluster and reduces future maintenance windows.

ALB Deprecated Starting from v4.2.0

The **ALB** (Alauda Container Platform Ingress Gateway) is marked as deprecated as of v4.2.0. New clusters and new users should adopt the Envoy Gateway–based Gateway API as the primary option. Existing clusters using ALB will keep working after the upgrade, but we strongly recommend planning and executing a migration to the Gateway API for long-term support and feature evolution.

Flannel Fully Removed

The **Flannel** (Alauda Container Platform Networking for Flannel) CNI plugin has been completely removed from the platform. Clusters still using Flannel must migrate to kube-ovn before upgrading to this release or any later version. Please plan and complete the migration in advance to avoid service disruption caused by switching the CNI.

Fixed Issues

- Previously, the status field of an upmachinepool resource stored the associated machine resources without any ordering. This caused the resource to be updated on every reconcile loop, resulting in excessively large audit logs. This issue has now been fixed.
- When the platform has a large number of clusters, the project quota of a single cluster cannot be updated after the quota is set for the project using the batch set project quota function. This issue has been fixed.
- When updating the password of the LDAP bind account, submitting the configuration returns a network validation error, causing the change to fail. This issue has been fixed.
- Previously, when creating a cluster-level Instance in OperatorHub, the web console automatically injected a metadata.namespace field, which caused a 404 error. This issue has now been fixed.
- When a user is automatically disabled by the system due to long-term lack of login, it will be automatically disabled again after being manually activated by the administrator. This issue has been fixed.
- Previously, after uninstalling an Operator, the Operator status was incorrectly displayed as Absent, even though the Operator was actually Ready. Users had to manually re-upload the Operator using violet upload. This issue has now been resolved, and the Operator correctly appears as Ready after uninstallation.
- In some cases, installing a new Operator version after uploading it via violet upload would fail unexpectedly. This intermittent issue has been fixed.
- When an Operator or Cluster Plugin included multiple frontend extensions, the left-side navigation of these extensions could become unresponsive. The temporary workaround required users to add the annotation cpaas.io/auto-sync: "false" to the extension’s ConfigMap. This behavior has now been permanently fixed in the code.
- Previously, if a cluster contained nodes with an empty Display Name, users were unable to filter nodes by typing in the node selector dropdown on the node details page. This issue has been resolved.

- The temporary files were not deleted after log archiving, preventing disk space from being reclaimed. This issue has been fixed.
- Uploading multiple packages from a folder using violet upload previously failed when disk space became insufficient. Violet now proactively cleans up uploaded packages in time, preventing these errors.
- Fixed an issue where modifying the Pod Security Policy when importing a namespace into a project did not take effect.
- Fixed an issue where the monitoring dashboards for workloads (e.g., Applications, Deployments) in Workload clusters failed to display when the global cluster was upgraded while the Workload clusters remained un-upgraded.
- Fixed an issue causing the KubeVirt Operator deployment to fail when upgrading on Kubernetes versions prior to 1.30.
- Fixed an inconsistency where Secrets created through the web console only stored the Username and Password, and lacked the complete authentication field (auth) compared to those created via kubectl create. This issue previously caused authentication failures for build tools (e.g., buildah) that rely on the complete auth data.

Known Issues

- When using violet push to upload a chart package, the push operation may complete successfully, but the package does not appear in the public-charts repository.
  Workaround: Push the chart package again.
- Although the installation page provides form fields for configuring labels and annotations for the global cluster, these configurations are not applied in practice.
- If a Custom Application includes an alert resource whose metrics expression uses customized metrics, deploying the application to a namespace whose name differs from the original—whether after exporting it as a chart or application YAML, importing the chart into the platform, or creating the application directly from the YAML — will cause the deployment to fail.

  This issue can be resolved by manually updating the metrics expression in the alert resource within the chart or YAML file, changing the namespace tag value to match the target deployment namespace.

- The default pool .mgr created by ceph-mgr uses the default Crush Rule, which may fail to properly select OSDs in a stretched cluster. To resolve this, the .mgr pool must be created using CephBlockPool. However, due to timing uncertainties, ceph-mgr might attempt to create the .mgr pool before the Rook Operator completes its setup, leading to conflicts.
  If encountering this issue, restart the rook-ceph-mgr Pod to trigger reinitialization.
  If unresolved, manually clean up the conflicting .mgr pool and redeploy the cluster to ensure proper creation order.
- Application creation failure triggered by the defaultMode field in YAML.
  Affected Path: Alauda Container Platform → Application Management → Application List → Create from YAML. Submitting YAML containing the defaultMode field (typically used for ConfigMap/Secret volume mount permissions) triggers validation errors and causes deployment failure.
  Workaround: Manually remove all defaultMode declarations before application creation.
- When pre-delete post-delete hook is set in helm chart.
  When the delete template application is executed and the chart is uninstalled, the hook execution fails for some reasons, thus the application cannot be deleted. It is necessary to investigate the cause and give priority to solving the problem of hook execution failure.