Menu

集群

概述

概述

不可变基础设施

不可变基础设施

集群节点规划

集群节点规划

创建本地集群

创建本地集群

etcd 加密

etcd 加密

实用指南

为内置注册表添加外部地址

选择容器运行时

更新公共仓库凭证

Overview

集群是运行容器化应用的基础资源集合，涵盖节点、负载均衡器、存储及其他关键组件。它是平台上成功运行容器化应用的前提条件。在平台初始安装时，会创建一个标准的 Kubernetes 集群，称为 `global` 集群。随后，可以将多个集群接入 `global` 集群，实现统一管理。

目录

Cluster Type

自建集群

自建集群是平台直接创建的 Kubernetes 集群。用户提供虚拟机或物理机，平台在这些机器上安装和配置 Kubernetes 集群。此方式适用于拥有现有硬件资源的企业，能够充分利用基础设施。

托管集群

托管集群是由云服务提供商提供的 Kubernetes 集群，集成到平台中进行统一管理。支持的集成方式包括：

方式	描述	使用场景	主要特点
导入	集成已有的 Kubernetes 集群	已有集群且具备直接网络访问能力	• 集群信息提交至 `global` 集群 • `global` 集群必须能访问该集群的网络
注册	集成具有严格安全要求的集群	安全约束较高的集群	• 目标集群安装特定插件 • 反向代理建立安全隧道 • 在保证集群安全的同时实现管理
代理创建	通过云服务提供商创建集群	利用公有云 Kubernetes 服务	• 需要云服务提供商的凭据 • 平台使用凭据创建 Kubernetes 集群

多云与混合云支持

这些集群管理方式满足企业在多云和混合云场景下的需求，支持不同阶段的容器化转型：

- 现有硬件：创建平台提供的集群
- 现有集群：导入或注册到平台
- 弹性需求：快速创建公有云集群

实施注意事项与限制

版本兼容性

- 支持的 Kubernetes 版本：1.28、1.29、1.30、1.31
- 自建集群和托管集群均需确保版本兼容
- 版本不匹配可能导致功能受限或兼容性问题

网络与安全要求

- 确保 `global` 集群与目标集群之间的网络连通性
- 实施适当的防火墙和网络安全策略
- 安全管理访问凭据和认证机制

集群管理最佳实践

1. 实施前评估

- 进行全面的基础设施和工作负载分析
- 明确每个集群的具体需求
- 制定完整的迁移与集成策略

2. 安全与合规

- 实施基于角色的访问控制（RBAC）
- 使用网络策略限制集群间通信

- 定期审计并更新安全配置
- 确保符合行业标准和法规要求

3. 监控与可观测性

- 搭建集中式日志和监控系统
- 实施主动告警机制
- 使用平台提供的可观测性工具
- 跟踪集群性能、资源利用率和健康状况

4. 备份与灾难恢复

- 建立定期备份流程
- 制定并测试灾难恢复方案
- 实施多集群备份策略
- 确保最小的停机时间和数据丢失

5. 持续优化

- 定期审查集群配置
- 优化资源分配
- 升级至最新支持的 Kubernetes 版本
- 利用平台功能实现自动更新和弹性伸缩

Menu

关于不可变基础设施

不可变基础设施使用不可变操作系统来配置 Kubernetes 集群。与传统的基于操作系统的集群不同，所有节点配置都预先集成在镜像中，部署后保持不变。集群升级和配置更改通过替换带有新镜像的节点来应用，确保整个集群生命周期中的一致性、可靠性和简化的运维。

Note

因为 Immutable Infrastructure 的发版周期与灵雀云容器平台不同，所以 Immutable Infrastructure 的文档现在作为独立的文档站点托管在 Immutable Infrastructure ↗。

集群节点规划

集群利用 Kubernetes 节点角色标签 `node-role.kubernetes.io/<role>` 来为节点分配不同的角色。为了描述方便，我们将此类标签称为角色标签。

默认情况下，集群包含两种类型的节点：控制平面节点和工作节点，分别用于承载控制平面工作负载和应用工作负载。

在集群中：

- 控制平面节点带有角色标签 `node-role.kubernetes.io/control-plane`。

  > 注意：
  > 在 Kubernetes v1.24 之前，社区也使用标签 `node-role.kubernetes.io/master` 来标记控制平面节点。为兼容历史版本，两个标签均被视为识别控制平面节点的有效标签。

- 工作节点默认没有角色标签，但你可以根据需要显式为工作节点分配角色标签 `node-role.kubernetes.io/worker`。

除了这些默认的角色标签外，你还可以在工作节点上定义自定义角色标签，以进一步将其划分为不同的功能类型。例如：

- 你可以添加角色标签 `node-role.kubernetes.io/infra`，将节点指定为 infra 节点，用于承载基础设施组件。

- 你可以添加角色标签 `node-role.kubernetes.io/log`，将节点指定为 log 节点，专门用于承载日志组件。

本文档将指导你如何创建 infra 节点和自定义角色节点，并将工作负载迁移到这些节点上。

目录

在非不可变集群上创建 **Infra** 节点

默认情况下，集群仅包含控制平面节点和工作节点。如果你想将某些工作节点指定为专门承载基础设施组件的 infra 节点，需要手动为这些节点添加相应的角色标签和污点。

> 注意：
> 本节操作仅适用于非不可变集群。即以下操作不支持在云集群（如通过 `Alauda Container Platform EKS Provider` 集群插件部署的 EKS 托管集群）、第三方集群或节点使用不可变操作系统的集群上执行。

添加 **Infra** 节点

步骤 **1**：为节点资源添加 **Infra** 角色标签

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

该命令为节点 192.168.143.133 添加 infra 角色标签：`node-role.kubernetes.io/infra: ""`，表示该节点为 infra 节点。

步骤 2：为节点资源添加污点

为防止其他工作负载调度到 infra 节点，添加污点。

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

该命令为节点 192.168.143.133 添加污点 `node-role.kubernetes.io/infra=reserved:NoSchedule`，表示只有容忍该污点的应用才能调度到此节点。

步骤 3：验证标签和污点

检查节点是否已被赋予 infra 角色标签和污点：

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

输出表明节点 192.168.143.133 已被配置为 infra 节点，并带有污点 `node-role.kubernetes.io/infra=reserved:NoSchedule`。

将 Pod 迁移到 Infra 节点

如果你想将特定 Pod 调度到 infra 节点，需要进行如下配置：

- 使用 nodeSelector 指定 infra 角色标签。
- 配置对应的容忍污点。

下面是一个配置为运行在 infra 节点上的 Deployment 示例清单。

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

nodeSelector 确保 Pod 仅调度到带有标签 `node-role.kubernetes.io/infra: ""` 的节点， 容忍配置允许 Pod 容忍污点 `node-role.kubernetes.io/infra=reserved:NoSchedule`。

通过这些配置，Pod 将被调度到 infra 节点。

> 注意：
> 通过 OLM Operator 或集群插件安装的 Pod 不一定能迁移到 infra 节点，是否支持迁移取决于各 Operator 或集群插件的配置。

自定义节点规划

除了 infra 节点，你可能还希望将工作节点指定为其他专用用途——例如承载日志组件、存储服务或监控代理。

你可以通过为工作节点分配更多自定义角色标签及相应污点，将其转变为自定义角色节点。

定义自定义角色节点的一般步骤

流程与创建 infra 节点类似。

步骤 1：添加自定义角色标签

```
kubectl label nodes <node> node-role.kubernetes.io/<role>="" --overwrite
```

将 <role> 替换为你期望的角色名称，如 monitoring、storage 或 log。

步骤 2：添加对应污点

```
kubectl taint nodes <node> node-role.kubernetes.io/<role>=<value>:NoSchedule
```

将 <role> 替换为自定义角色名称，<value> 替换为有意义的描述符，如 reserved 或 dedicated。该值为可选，但有助于文档说明和清晰度。

步骤 3：验证配置

```
kubectl describe node <node>
```

确保 Labels 和 Taints 字段反映了你的自定义角色配置。

示例：创建专门用于日志组件的节点

如果你想创建专门安装日志组件的节点，可以添加 log 角色。具体操作如下。

步骤 1：添加 **Log** 角色标签

```
kubectl label nodes 192.168.143.133 node-role.kubernetes.io/log="" --overwrite
```

该标签表示该节点被指定用于日志相关工作负载。

步骤 2：为节点添加污点

```
kubectl taint nodes 192.168.143.133 node-role.kubernetes.io/log=reserved:NoSchedule
```

该污点防止未容忍该污点的工作负载调度到该节点。

步骤 3：验证标签和污点

```
Name:               192.168.143.133
Roles:              log
Labels:             node-role.kubernetes.io/log=reserved
                    ...
Taints:             node-role.kubernetes.io/log=reserved:NoSchedule
```

这表明节点已成功配置为 log 节点，并带有相应的标签和污点。

通过以上实践，你可以根据节点的预期用途有效地划分 Kubernetes 节点，提升工作负载隔离性，确保特定组件部署到配置合适的节点上。

创建本地集群

目录

前提条件

节点要求

1. 如果您从下载安装包下载的是单架构安装包，请确保节点机器的架构与安装包一致，否则节点因缺少对应架构镜像无法启动。
2. 请确认节点操作系统和内核版本受支持，详情见支持的操作系统和内核。
3. 对节点机器进行可用性检查，具体检查项请参考节点预处理 > 节点检查。
4. 如果节点机器 IP 无法通过 SSH 直接访问，请为节点提供 SOCKS5 代理，`global` 集群将通过该代理服务访问节点。

负载均衡

生产环境中，集群控制平面节点需要负载均衡器以保证高可用性。 您可以提供自有硬件负载均衡器，或启用“自建 VIP”，该方式通过 haproxy + keepalived 实现软件负载均衡。 推荐使用硬件负载均衡器，原因如下：

- 性能更优：硬件负载均衡性能优于软件负载均衡。
- 复杂度更低：如果不熟悉 keepalived，配置错误可能导致集群不可用，排查耗时且严重影响集群可靠性。

使用自有硬件负载均衡器时，可将负载均衡器的 VIP 作为“IP 地址 / 域名”参数；如果有解析到负载均衡器 VIP 的域名，也可使用该域名作为“IP 地址 / 域名”参数。 注意：

- 负载均衡器必须正确转发流量到集群所有控制平面节点的端口 `6443`、`11780` 和 `11781`。
- 如果集群仅有一个控制平面节点，且使用该节点 IP 作为“IP 地址 / 域名”参数，后续无法将单节点集群扩展为高可用多节点集群。因此，即使是单节点集群，也建议提供负载均衡器。

启用“自建 VIP”时，需准备：

1. 可用的 VRID
2. 支持 VRRP 协议的主机网络
3. 所有控制平面节点与 VIP 必须在同一子网，且 VIP 不能与任何节点 IP 重复。

连接 `global` 集群与业务集群

平台要求 `global` 集群与业务集群之间互通访问。若不在同一网络，需要：

1. 为业务集群提供“外部访问”，确保 `global` 集群可访问业务集群。网络要求确保 `global` 可访问所有控制平面节点的端口 `6443`、`11780` 和 `11781`。

2. 为 `global` 集群添加业务集群可访问的额外地址。创建业务集群时，将该地址以键 `cpaas.io/platform-url`、值为 `global` 集群公网访问地址的形式添加到集群注解中。

镜像仓库

集群镜像支持平台内置、私有仓库和公共仓库三种选项。

- 平台内置：使用 `global` 集群提供的镜像仓库。若集群无法访问 `global`，请参见为内置仓库添加外部地址。
- 私有仓库：使用您自有的镜像仓库。推送所需镜像至私有仓库的具体操作，请联系技术支持。
- 公共仓库：使用平台公共镜像仓库。使用前请完成更新公共仓库凭据。

容器网络

若计划使用 Kube-OVN 的 Underlay 模式，请参考准备 Kube-OVN Underlay 物理网络。

创建操作步骤

1. 进入 管理员 视图，点击左侧导航栏的 集群/集群。
2. 点击 创建集群。
3. 按照以下说明配置基本信息、容器网络、节点设置和扩展参数。

基本信息

参数	说明

Kubernetes 版本	所有可选版本均经过严格测试，保证稳定性和兼容性。 推荐：选择最新版本以获得最佳功能和支持。
容器运行时	默认提供 containerd 作为容器运行时。 若偏好使用 Docker 作为容器运行时，请参考选择容器运行时。
集群网络协议	支持三种模式：IPv4 单栈、IPv6 单栈、IPv4/IPv6 双栈。 注意：选择双栈模式时，确保所有节点均正确配置 IPv6 地址；设置后网络协议不可更改。
集群访问端点	`IP 地址 / 域名`：填写预先准备的域名，若无域名则填写 VIP。 `自建 VIP`：默认关闭。仅当未提供 LoadBalancer 时启用，启用后安装程序会自动部署 `keepalived` 以支持软件负载均衡。 `外部访问`：当集群与 `global` 集群不在同一网络环境时，填写为集群准备的外部访问地址。

容器网络

Kube-OVN

Alauda 开发的企业级云原生 Kubernetes 容器网络编排系统。它将 OpenStack 领域成熟的网络能力引入 Kubernetes，支持跨云网络管理、传统网络架构与基础设施互联、边缘集群部署场景，同时大幅提升 Kubernetes 容器网络的安全性、管理效率和性能。

参数	说明
子网	又称 Cluster CIDR，表示默认子网段。集群创建后可添加额外子网。
传输模式	**Overlay**：基于基础设施抽象的虚拟网络，不占用物理网络资源。创建 Overlay 默认子网时，集群内所有 Overlay 子网使用相同的集群 NIC 和节点 NIC 配置。 **Underlay**：依赖物理网络设备的传输方式，可直接为 Pod 分配物理网络地址，保证更好性能和与物理网络的连通性。Underlay 子网的节点必须具备多块网卡，且用于桥接网络的网卡必须专用于 Underlay，不承载其他流量如 SSH。创建 Underlay 默认子网时，集群 NIC 实际为桥接网络默认网卡，节点 NIC 为桥接网络中的节点网卡配置。 • 默认网关：物理网络网关地址，即 Cluster CIDR 段的网关地址（必须在 Cluster CIDR 地址范围内）。 • **VLAN ID**：虚拟局域网标识符（VLAN 号），例如 `0`。 • 保留 **IP**：设置不会被自动分配的保留 IP，如子网内已被其他设备占用的 IP。
服务 **CIDR**	Kubernetes 类型为 ClusterIP 的服务使用的 IP 地址范围。不能与默认子网范围重叠。
Join CIDR	Overlay 传输模式下，节点与 Pod 之间通信使用的 IP 地址范围。不能与默认子网或服务 CIDR 重叠。

Calico

Calico 是一种三层网络方案，为容器提供安全的网络连接。

参数	说明
默认子网	又称 Cluster CIDR，表示默认子网段。集群创建后可添加额外子网。
服务 **CIDR**	Kubernetes 类型为 ClusterIP 的服务使用的 IP 地址范围。不能与默认子网范围重叠。

Flannel

Flannel 为集群内所有容器提供扁平网络环境，使不同节点主机上创建的容器拥有全集群唯一的虚拟 IP 地址。Pod 子网根据掩码均匀划分给集群节点，每个节点上的 Pod 从分配给该节点的段中获取 IP 地址。提升容器间通信效率，无需考虑 IP 转换问题。

参数	说明
集群 **CIDR**	集群启动时创建 Pod 使用的 IP 地址范围。支持设置当前容器网络下每个节点可分配给 Pod 的最大 IP 数量。 注意：平台会根据上述配置自动计算集群可容纳的最大节点数，并在输入框下方提示显示。 重要：集群创建后，集群网络不可更改，请谨慎规划网络。

服务 **CIDR**	Kubernetes 类型为 ClusterIP 的服务使用的 IP 地址范围。不能与容器子网范围重叠。

自定义

如果需要安装其他网络插件，选择自定义模式。集群创建成功后可手动安装网络插件。

参数	说明
集群 **CIDR**	集群启动时创建 Pod 使用的 IP 地址范围。
服务 **CIDR**	Kubernetes 类型为 ClusterIP 的服务使用的 IP 地址范围。不能与容器子网范围重叠。

节点设置

参数	说明
网卡名称	集群网络插件使用的主机网络接口设备名称。 注意： • 选择 Kube-OVN 默认子网的 Underlay 传输模式时，必须指定网卡名称，该网卡将作为桥接网络的默认网卡。 - 平台默认识别名称格式为 `eth.|en.|wl./ww.` 的网卡流量进行监控。如使用其他命名规则的网卡，请参考从自定义命名网卡采集网络数据修改相关资源，确保平台能正确监控网卡流量。

节点名称	可选择使用节点 IP 或主机名作为平台上的节点名称。 注意：选择主机名作为节点名称时，确保集群中所有节点主机名唯一。
节点列表	添加节点到集群，或从草稿中恢复暂存的节点信息。具体添加节点参数说明见下文。
监控类型	支持 **Prometheus** 和 **VictoriaMetrics**。 选择 **VictoriaMetrics** 监控组件时，需配置 部署类型： **-** 部署 **VictoriaMetrics**：部署所有相关组件，包括 **VMStorage**、**VMAlert**、**VMAgent** 等。 **-** 部署 **VictoriaMetrics Agent**：仅部署日志采集组件 **VMAgent**。此方式需关联平台上已部署的其他集群的 VictoriaMetrics 实例，为集群提供监控服务。
监控节点	选择部署集群监控组件的节点。支持选择允许部署应用的计算节点和控制平面节点。 为避免影响集群性能，建议优先选择计算节点。集群创建成功后，存储类型为本地卷 的监控组件将在所选节点部署。

添加节点参数

参数	说明
类型	控制平面节点：负责运行 kube-apiserver、kube-scheduler、kube-controller-manager、etcd、容器网络及部分平台管理组件。启用 允许部署应用 后，控制

	平面节点也可作为计算节点使用。 工作节点：负责承载集群中运行的业务 Pod。
IPv4 地址	节点的 IPv4 地址。内网模式创建的集群填写节点的 私有 **IP**。
IPv6 地址	集群启用 IPv4/IPv6 双栈时有效，节点的 IPv6 地址。
允许部署应用	集群节点类型为 控制平面节点 时有效。是否允许在该控制平面节点部署业务应用，将业务相关 Pod 调度到该节点。
显示名称	节点的显示名称。
SSH 连接 **IP**	访问节点 SSH 服务时可连接的 IP 地址。 若可通过 `ssh <用户名>@<节点 IPv4 地址>` 登录节点，则此参数可不填；否则需填写节点的公网 IP 或 NAT 外网 IP，确保 `global` 集群及代理可通过该 IP 连接节点。
网卡名称	输入节点使用的网卡名称。网卡配置生效优先级如下（从左到右，依次降低）： **Kube-OVN Underlay**：节点网卡 > 集群网卡 **Kube-OVN Overlay**：节点网卡 > 集群网卡 > 节点默认路由对应网卡

	Calico：集群网卡 > 节点默认路由对应网卡 **Flannel**：集群网卡 > 节点默认路由对应网卡
关联桥接网络	注意：创建集群时不支持桥接网络配置，仅在为已有 Underlay 子网的集群添加节点时可用。 选择已有的添加桥接网络。若不想使用桥接网络默认网卡，可单独配置节点网卡。
SSH 端口	SSH 服务端口号，例如 `22`。
SSH 用户名	SSH 用户名，需为具备 root 权限的用户，例如 `root`。
代理	是否通过代理访问节点 SSH 端口。当 `global` 集群无法直接通过 SSH 访问待添加节点（如 `global` 集群与业务集群不在同一子网，节点 IP 为 `global` 集群无法直接访问的内网 IP）时，需开启此开关并配置代理相关参数。配置代理后，可通过代理访问和部署节点。 注意：当前仅支持 SOCKS5 代理。 访问地址：代理服务器地址，例如 `192.168.1.1:1080`。 用户名：访问代理服务器的用户名。

	密码：访问代理服务器的密码。
SSH 认证	登录待添加节点的认证方式及对应认证信息。选项包括： 密码：需提供具备 root 权限的用户名及对应 **SSH** 密码。 密钥：需提供具备 root 权限的 私钥 及 私钥密码。
保存草稿	将当前对话框配置的数据保存为草稿并关闭 添加节点 对话框。 在不离开 创建集群 页面情况下，可选择 从草稿恢复 打开 添加节点 对话框，恢复保存的草稿配置数据。 注意：恢复的是最近一次保存的草稿数据。

扩展参数

注意：

- 除必填配置外，不建议设置扩展参数，错误配置可能导致集群不可用，且集群创建后无法修改。
- 若输入的 **Key** 与默认参数 **Key** 重复，则覆盖默认配置。

操作步骤

1. 点击 扩展参数 展开扩展参数配置区域。可选设置以下集群扩展参数：

参数	说明

Docker 参数	`dockerExtraArgs`，Docker 的额外配置参数，将写入 `/etc/sysconfig/docker`。不建议修改。若通过 `daemon.json` 配置 Docker，需以键值对形式配置。
Kubelet 参数	`kubeletExtraArgs`，Kubelet 的额外配置参数。 注意：当输入 容器网络 的 节点 **IP** 数量 参数时，系统会自动生成默认的 **Kubelet** 参数，键为 `max-pods`，值为 节点 **IP** 数量，用于设置集群中任一节点可运行的最大 Pod 数。该配置不在界面显示。 在 **Kubelet** 参数 区域新增 `max-pods: 最大可运行 Pod 数` 键值对将覆盖默认值。允许任意正整数，但建议使用默认值（节点 IP 数量）或不超过 `256`。
Controller Manager 参数	`controllerManagerExtraArgs`，Controller Manager 的额外配置参数。
Scheduler 参数	`schedulerExtraArgs`，Scheduler 的额外配置参数。
APIServer 参数	`apiServerExtraArgs`，APIServer 的额外配置参数。
APIServer URL	`publicAlternativeNames`，证书中颁发的 APIServer 访问地址。仅可填写 IP 或域名，最长 253 字符。
集群注解	集群注解信息，以键值对形式标记集群特征，供平台组件或业务组件获取相关信息。

4. 点击 创建，返回集群列表页面，集群状态为 创建中。

创建后操作

查看创建进度

在集群列表页面，可查看已创建集群列表。对于状态为 创建中 的集群，可查看执行进度。

操作步骤

1. 点击集群状态右侧的小图标 查看执行进度。

2. 弹出的执行进度对话框中，可查看集群执行进度（status.conditions）。

   提示：当某类型处于进行中或失败状态且带有原因时，将鼠标悬停在对应原因（蓝色文字）上，可查看原因详细信息（status.conditions.reason）。

关联项目

集群创建完成后，可在项目管理视图中将其添加至项目。

etcd 加密

本指南帮助您安装、理解并操作 ACP 中的 etcd Encryption Manager，以实现集群内 etcd 数据加密密钥的自动轮换。

它确保存储在 etcd 中的敏感数据（如 secrets 和 configmaps）使用安全算法加密，从而提升集群的安全性。

目录

安装

请参阅 Cluster Plugin 获取安装说明。

> 注意：
>
> - 当前支持：
>
> - On-Premises 集群

> - DCS 集群
> - 不支持：
>   - `global cluster`

工作原理

安装后，会在 `kube-system` 命名空间部署一个 `etcd-encryption-manager` 控制器，该控制器：

- 定期轮换 etcd 数据加密密钥。
- 保留最近 8 个密钥以支持回滚兼容。
- 更新所有控制节点上的加密配置。
- 触发 `kube-apiserver` 热加载新密钥。
- 自动迁移资源，使用新密钥重新加密数据。

在整个操作过程中，集群稳定性得到保障。

默认配置

参数	值
加密资源	secrets, configmaps
加密算法	256-bit AES-GCM
轮换间隔	168 小时（7 天）

操作指南

配置文件

路径	内容
`/etc/kubernetes/encryption-provider.conf`	当前加密配置
`/etc/kubernetes/encryption-provider-history.bak`	历史密钥记录（用于恢复）
`/etc/kubernetes/encryption-provider-bak/`	过期加密配置版本

状态检查

运行以下命令检查当前轮换状态：

```
kubectl get EtcdEncryptionConfig default -o yaml
```

示例输出：

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: EtcdEncryptionConfig
metadata:
  name: default
spec:
  resources:
    - secrets
    - configmaps
  rotationInterval: 168h0m0s
  type: aesgcm
status:
  deployStatus:
    192.168.100.1:
      revision: 3
      state: Success
    192.168.100.2:
      revision: 3
      state: Success
    192.168.100.3:
      revision: 3
      state: Success
  migration:
    completeTimestamp: "2025-05-27T05:47:01Z"
    resources:
      - secrets
      - configmaps
    revision: 3
    state: Success
  revision: 3
```

Menu

实用指南

为内置注册表添加外部地址

选择容器运行时

更新公共仓库凭证

为内置注册表添加外部地址

目录

概述

当 `global` 集群使用 `Platform Built-in` 注册表时，工作负载集群通常也使用此注册表来拉取镜像。该注册表不仅服务于 `global` 集群中的组件，还必须对工作负载集群的节点可访问。

在某些情况下，工作负载集群节点无法直接访问 `global` 集群的注册表地址——例如，当 `global` 集群位于私有数据中心，而工作负载集群位于公有云或边缘环境中时。

本指南说明如何为平台的默认注册表配置一个外部可访问的地址，以便工作负载集群能够拉取镜像。

先决条件

在开始之前，请准备以下内容：

- 一个工作负载集群节点可以访问的域名

- 域名指向的IP地址
- 域名的有效SSL证书

> 警告
>
> - 域名必须与平台访问地址不同
> - 确保域名的IP地址能够将流量转发到 `global` 集群的所有控制平面节点

步骤

配置平台注册表的证书和路由规则

1. 将域名的有效证书复制到 `global` 集群的任意控制平面节点

2. 创建一个包含域名证书的TLS密钥：

   ```
   kubectl create secret tls registry-address.tls --cert=<certificate-filename> --key=
   <key-filename> -n kube-system
   ```

   示例：

   ```
   kubectl create secret tls registry-address.tls --cert=custom.crt --key=custom.key -n
   kube-system
   ```

   注意：创建证书后，监控 **global** 集群 **kube-system** 命名空间中 **registry-address.tls** 密钥的到期日期。在证书到期之前，请替换证书。

3. 在 `global` 集群的任意控制平面节点上创建入口规则：

```
REGISTRY_DOMAIN_NAME=<www.registry.com> # 替换为您可访问的域名
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/.+/blobls/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobls/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
```

返回的响应类似于 `... created` 表示入口创建成功。

4. 检查是否存在注册表服务资源：

```
kubectl -n kube-system get svc | grep registry
```

如果该服务不存在，请使用以下命令创建它：

```
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. 使用域名从注册表拉取镜像以测试配置：

```
crictl pull <registry-domain-name>/automation/qaimages:helloworld
```

或者

```
docker pull <registry-domain-name>/automation/qaimages:helloworld
```

选择容器运行时

目录

概述

容器运行时是 Kubernetes 的核心组件，负责管理镜像和容器的生命周期。

在通过平台创建集群时，您可以选择 Containerd 或 Docker 作为您的运行时组件。

注意：Kubernetes 版本 1.24 及以上不再官方支持 Docker 运行时。官方推荐的运行时是 Containerd。如果您仍需使用 Docker 运行时，则在创建集群时必须首先在功能开关中启用 `cri-docker`，才能选择 Docker 作为运行时组件。有关使用功能开关的详细信息，请参见 功能开关配置。

快速选择指南

选择 **Containerd**	选择 **Docker**
• 更短的调用链 • 更少的组件 • 更加稳定 • 消耗较少的节点资源	• 支持 docker-in-docker • 允许在节点上使用 `docker build/push/save/load` 命令 • 可以调用 Docker API • 支持 Docker Compose 或 Docker Swarm

Docker 和 Containerd 的区别

常用命令

Containerd	Docker	描述
crictl ps	docker ps	查看正在运行的容器
crictl inspect	docker inspect	查看容器详细信息
crictl logs	docker logs	查看容器日志
crictl exec	docker exec	在容器中执行命令
crictl attach	docker attach	附加到容器
crictl stats	docker stats	显示容器资源使用情况
crictl create	docker create	创建容器
crictl start	docker start	启动容器
crictl stop	docker stop	停止容器
crictl rm	docker rm	移除容器
crictl images	docker images	查看镜像列表
crictl pull	docker pull	拉取镜像

Containerd	Docker	描述
None	docker push	推送镜像
crictl rmi	docker rmi	删除镜像
crictl pods	None	查看 Pod 列表
crictl inspectp	None	查看 Pod 详细信息
crictl runp	None	启动 Pod
crictl stopp	docker images	查看镜像
ctr images ls	None	停止 Pod
crictl stopp	docker load/save	导入/导出镜像
ctr images import/export	None	停止 Pod
ctr images pull/push	docker pull/push	拉取/推送镜像
ctr images tag	docker tag	标记镜像

调用链差异

- Docker 作为 Kubernetes 容器运行时有以下调用关系：

  kubelet > cri-dockerd > dockerd > containerd > runC

- Containerd 作为 Kubernetes 容器运行时有以下调用关系：

  kubelet > cri 插件（在 containerd 进程中）> containerd > runC

总结：虽然 dockerd 添加了像集群、docker build 和 Docker API 等功能，但可能会引入 bug，并增加调用链中的一个额外层。Containerd 拥有更短的调用链，更少的组件，更大的稳定性，并消耗较少的节点资源。

日志和参数比较

比较	Docker	Containerd
存储路径	当 Docker 作为 Kubernetes 容器运行时，容器日志由 Docker 存储在 `/var/lib/docker/containers/$CONTAINERID` 等目录中。Kubelet 在 `/var/log/pods` 和 `/var/log/containers` 中创建指向该目录中容器日志文件的符号链接。	当 Containerd 作为 Kubernetes 容器运行时，容器日志由 Kubelet 存储在 `/var/log/pods/$CONTAINER_NAME` 目录中，并在 `/var/log/containers` 目录中创建指向日志文件的符号链接。
配置参数	在 Docker 配置文件中指定：`"log-driver": "json-file",` `"log-opts": {"max-size": "100m","max-file": "5"}`	*方法 1*：在 kubelet 参数中指定：`--container-log-max-files=5` `--container-log-max-size="100Mi"` *方法 2*：在 KubeletConfiguration 中指定：`"containerLogMaxSize": "100Mi",` `"containerLogMaxFiles": 5,`
将容器日志保存到数据磁盘	将数据磁盘挂载到 "data-root"（默认是 `/var/lib/docker`）。	创建符号链接 `/var/log/pods` 指向数据磁盘挂载点下的目录。

CNI 网络比较

比较	**Docker**	**Containerd**
谁调用 **CNI**	cri-dockerd	内置在 Containerd 中的 cri 插件（在 containerd 1.1 之后）

如何配置 **CNI**	cri-dockerd 参数 `--cni-conf-dir` `--cni-bin-dir` `--cni-cache-dir`	Containerd 配置文件（toml）：
		`[plugins.cri.cni]`
		`bin_dir = "/opt/cni/bin"`
		`conf_dir = "/etc/cni/net.d"`

Updating Public Repository Credentials

目录

Overview

`Public Repository` 是一个平台提供的可在公共互联网访问的镜像仓库服务。当您希望集群使用 `Public Repository` 作为镜像仓库时，需要更新内置的 `public-registry-credential` Cloud Credentials。这样可以确保平台有权限从公共仓库拉取镜像。

Procedure

1. 登录 **Customer Portal**，并从右上角 **User Information** 下拉菜单中的 **Enterprise Management** 部分下载您组织的认证文件。
2. 在 **Administrator** 控制台左侧导航栏中，进入 **Clusters** > **Cloud Credential**。
3. 找到名为 `public-registry-credential` 的云凭证，点击右侧下拉菜单中的 **Update**。
4. 在 **Upload Public Repository Address** 部分，上传您从 **Customer Portal** 下载的认证文件。
5. 点击 **Update** 以应用更改。