

# 可观测性

## 概览

### 介绍

产品介绍

产品优势

产品应用场景

### 功能

监控

告警通知

分布式追踪

日志

事件

巡检

## 监控

### 介绍

模块概述

模块优势

应用场景

使用限制

### 安装

概述

安装准备

安装 ACP 监控与 Prometheus 插件

安装 ACP 监控与 VictoriaMetrics 插件

### 架构

### 核心概念

Monitoring

Alarms

Notifications

### 操作指南

### 实用指南

Das

权限说明

调用链

介绍

- 优势
- 应用场景
- 使用限制

安装

- 安装 Jaeger Operator
- 部署 Jaeger 实例
- 安装 OpenTelemetry Operator
- 部署 OpenTelemetry 实例

架构

- 核心组件
- 数据流程

操作指南

实用指南

问题处理

快速入门

OpenTelemetry

OpenTelemetry  
Jaeger

日志

介绍

- 模块介绍
- 模块优势
- 模块应用场景
- 模块使用限制

安装

- 安装 ACP Log Storage with ElasticSearch
- 安装 ACP Log Storage with Clickhouse
- 安装 ACP Log Collector 插件

架构

概念

- 开源组件
- 核心功能概念

操作指南

语

实用指南

权限说明

## 事件

介绍

Events

权限说明

Module Overview

操作流程

Functionality Overview

事件概览

Use Cases

Usage Limitations

## 巡检

介绍

架构

操作指南

模块介绍

巡检

模块优势

组件健康状态

模块应用场景

权限说明

使用限制

# 概览

## 介绍

产品介绍

产品优势

产品应用场景

## 功能

监控

告警通知

分布式追踪

日志

事件

巡检



# 介绍

## 目录

产品介绍

产品优势

产品应用场景

## 产品介绍

Observability 模块是一款以应用为中心的开箱即用的云原生观测平台，提供面向企业的全方位监控解决方案。它能够实时监控应用及其资源，收集各种指标、日志和事件数据，帮助分析应用的健康状态。该模块不仅具备强大的告警能力，还提供全面且清晰的多维度数据可视化，兼容主流开源组件，支持快速故障定位和一键监控诊断。

## 产品优势

Observability 模块的核心优势如下：

- 统一数据采集

实现指标、日志和追踪的统一采集，提供系统的全景视图，简化管理流程。

- 多维度告警机制

支持指标和日志的多维度告警设置，确保用户能够及时获知潜在问题。

- 直观的可视化界面

提供简洁明了的可视化管理界面，帮助用户快速获取关键信息，提高决策效率。

- 强大的兼容性

完美兼容主流开源组件，方便用户轻松集成，提升系统灵活性。

- 开箱即用

提供预配置模板和最佳实践，帮助用户快速上手，无需复杂配置。

## 产品应用场景

Observability 模块的主要应用场景包括：

- 微服务架构监控

在云原生微服务架构中，应用由多个独立服务组成。Observability 模块能够实时监控每个微服务的性能和健康状态，帮助开发团队快速识别和解决服务间依赖问题，保障整体系统稳定性。

- 容器及 Kubernetes 监控

针对部署在 Kubernetes 环境中的容器应用，Observability 模块可监控容器的资源使用、状态和日志，提供容器生命周期管理和故障排查，确保容器化应用的高可用性。

- 动态资源管理

在云环境中，资源使用可能随业务需求变化而波动。Observability 模块能够实时监控资源使用情况，支持动态调整资源分配，优化成本与性能。

- 多云环境监控

对于采用多云策略的企业，Observability 模块能够统一管理和监控跨不同云平台的应用及资源，确保云环境的可视性和一致性。

# 功能

## 目录

### 监控

告警通知

分布式追踪

日志

事件

巡检

## 监控

- 探针

平台基于 Blackbox Exporter 提供 Probe 功能（黑盒监控），支持通过 ICMP、TCP、HTTP 等协议进行网络服务检测。与依赖内部系统指标的白盒监控不同，Probe 从用户视角对服务进行外部评估，能够快速发现影响用户体验的故障。

例如，当业务接口响应异常（如 HTTP 5xx 错误）或关键服务不可用时，Probe 能够立即检测到异常，触发告警，帮助运维团队高效排查问题。

- 监控面板

平台具备现代化的监控面板管理功能，相较传统 Grafana 提供更友好的可视化配置体验。通过统一的监控视图，聚合展示各类监控指标数据，帮助用户快速构建所需的监控面板。

# 告警通知

- 告警策略

平台提供完善的告警能力，支持基于指标、日志和事件配置告警规则。借助丰富的内置监控指标和告警模板，用户能够快速配置符合业务需求的告警策略，实现及时发现和处理问题。

- 告警模板

告警模板对告警规则和通知策略进行标准化封装，支持在多个监控对象间快速复用。基于模板的配置显著降低了告警策略的管理成本，提高了运维效率。

- 告警历史

系统完整记录告警的生命周期，包括触发时间、恢复时间、告警状态、告警级别和告警内容。用户可通过告警历史追溯和分析问题，持续优化告警配置。

- 通知

平台支持多种告警通知渠道，包括邮件、钉钉、企业微信、飞书和 Webhook，确保告警信息及时送达相关人员。用户可根据实际需求灵活配置通知策略。

# 分布式追踪

分布式追踪为微服务架构提供全链路追踪能力。通过采集服务间调用的元数据，帮助用户快速定位跨服务调用中的问题。

# 日志

平台自动采集并集中管理集群、节点和容器的标准输出及文件日志，提供强大的日志存储、检索和分析能力，支持多维度日志查询和可视化展示，帮助用户快速定位问题。

# 事件



平台实时采集 Kubernetes 集群中的关键事件信息，记录资源状态变化的完整过程。当集群、节点、Pod 等出现异常时，可通过事件追踪定位根因，显著提升问题解决效率。

## 巡检

- 巡检

借鉴丰富的企业级运维经验，平台提供自动化巡检能力。通过多维度健康检查，帮助用户实时监控资源运行状态，提前发现潜在风险，降低人工巡检成本。

- 平台健康状态

提供平台功能健康状态的直观概览，支持查看部署情况及组件运行状态。具备平台管理权限的用户可深入查看详细的健康检查数据，快速定位并解决平台级问题。

# 监控

---

## 介绍

### 介绍

模块概述

模块优势

应用场景

使用限制

---

## 安装

### 安装

概述

安装准备

安装 ACP 监控与 Prometheus 插件

安装 ACP 监控与 VictoriaMetrics 插件

---

## 架构

监控模块架构

- 整体架构说明
- 监控系统
- 告警系统
- 通知系统

监控组件选型指南

- 重要说明
- 组件清单
- 架构对比
- 功能对比
- 安装方案建议

核心概念

核心概念

- Monitoring
- Alarms
- Notifications
- Monitoring Dashboard

操作指南

指标管理

- 查看平台组件暴露的指标
- 查看 Prometheus / VictoriaMetrics 存储的指标
- 查看平台内置定义的所有指标
- 集成外部指标

告警管理

- 功能概述
- 主要功能
- 功能优势
- 通过 UI 创建告警策略
- 通过 CLI 创建资源告警
- 通过 CLI 创建事件告警

通知管理

- 功能概述
- 主要功能
- 通知服务器
- 通知联系人组
- 通知模板
- 通知规则

管理监控面板

通知

- 功能概述
- 管理监控面板
- 管理 panels
- 通过 CLI 创建监控面板
- 常用函数和变量

探针管理

- 功能概述
- 黑盒监控
- 黑盒告警
- 自定义 Blackbox
- 通过 CLI 创建黑
- 参考信息

实用指南

- Prometheus 监控数据的备份与恢复

  - 功能概述
  - 使用场景
  - 前提条件
  - 操作流程
  - 操作结果
  - 了解更多
  - 后续操作
- VictoriaMetrics 监控数据备份与恢复

  - 功能简介
  - 使用场景
  - 前置条件
  - 操作步骤
  - 操作结果
  - 了解更多
  - 后续操作
- 从自定义命名空间导出数据

  - 功能概述
  - 适用场景
  - 前提条件
  - 操作步骤
  - 操作结果
  - 了解更多
  - 后续操作

权限说明

- 权限说明



# 介绍

## 目录

### 模块概述

### 模块优势

### 应用场景

### 使用限制

## 模块概述

监控模块为平台管理员和运维人员提供了指标、仪表板、告警和通知等运维能力。

该平台结合了开源组件如 Prometheus / VictoriaMetrics 和监控仪表板，使管理的集群、节点、组件、自定义应用程序、Pods、容器等实现实时监控。

支持快速设置集群、节点和计算组件层面的监控指标告警、日志告警（仅适用于计算组件）和事件告警。此外，允许根据实际需求自定义监控指标算法，以增加必要的告警指标和规则。通知策略可以配置，以便及时将告警信息发送给运维人员，帮助避免系统故障或迅速解决问题，降低系统运维成本并确保系统稳定性。

## 模块优势

监控模块具有以下核心优势：

- 全面监控覆盖

支持对多个层面的广泛监控，如集群、节点、组件和容器，实现从基础设施到应用的端到端监控链路。

- 灵活的告警配置

提供丰富的预设告警规则，同时支持自定义告警规则和算法，以满足不同的监控场景。

- 多样化的可视化展示

集成专业的监控仪表盘，支持多种数据可视化方法，直观呈现系统运行状态。

- 高效的告警通知

支持多渠道告警通知，包括电子邮件、短信、Webhook 等，确保及时传达告警信息。

- 可扩展的监控架构

基于行业领先的 Prometheus / VictoriaMetrics 技术栈，具备卓越的可扩展性和兼容性。

## 应用场景

监控模块适用于以下场景：

- 集群健康监控

实时监控集群中的资源使用情况、节点状态和组件运行状况，以便及时发现潜在问题。

- 应用性能分析

监控应用运行指标和容器资源使用情况，以优化应用性能。

- 故障预警与诊断

通过设置合理的告警规则，提前检测系统异常，快速识别和解决问题。

- 容量规划

基于历史监控数据进行趋势分析，为资源扩展和优化提供依据。

# 使用限制

使用监控模块时，请注意以下限制：

- 监控数据的存储时间取决于存储容量配置，默认保留期限为 7 天。
- Prometheus 和 VictoriaMetrics 不能同时安装在同一集群中，请做好选择规划并选择其一进行安装。
- 自定义监控指标的采集间隔最低支持 60 秒。
- 告警通知渠道需预先配置相应服务（例如电子邮件服务器、短信网关、企业微信/钉钉机器人等）。



# 安装

## 目录

### 概述

安装准备

安装 ACP 监控与 Prometheus 插件

安装步骤

访问方式

安装 ACP 监控与 VictoriaMetrics 插件

前提条件

安装步骤

## 概述

监控组件是可观测模块中监控、告警、巡检和健康检查功能的基础设施。本文描述了如何在集群中安装 ACP 监控与 Prometheus 插件或 ACP 监控与 VictoriaMetrics 插件。

## 安装准备

在安装监控组件之前，请确保满足以下条件：

- 已参考 [监控组件选型指南](#) 选择合适的监控组件。
- 在工作负载集群中安装时，确保 `global` 集群可以访问工作负载集群的 11780 端口。

- 如果需要使用存储类或持久卷存储监控数据，请提前在 存储 部分创建相应资源。

## 安装 ACP 监控与 Prometheus 插件

### 安装步骤

1. 导航至 应用商店管理 > 集群插件 并选择目标集群。
2. 找到 **ACP 监控与 Prometheus** 插件并点击 安装。
3. 配置以下参数：

参数	说明
规模配置	支持三种配置：小规模、中规模和大规模： <ul style="list-style-type: none"><li>- 默认值基于平台的推荐负载测试值设置</li><li>- 可根据实际集群规模选择或自定义配额</li><li>- 默认值会随着平台版本更新；对于固定配置，建议使用自定义设置</li></ul>
存储类型	<ul style="list-style-type: none"><li>- <b>LocalVolume</b>：本地存储，数据存储在指定节点</li><li>- <b>StorageClass</b>：使用存储类自动生成持久卷</li><li>- <b>PV</b>：利用已有的持久卷</li></ul> 注意：安装后存储配置无法修改
副本数	设置监控组件容器组的数量 注意：Prometheus 仅支持单节点安装
参数配置	可根据需要调整监控组件的数据参数

4. 点击 安装 完成安装。

### 访问方式

安装完成后，可通过以下地址访问各组件（需将 `<>` 替换为实际值）：

组件	访问地址
Thanos	<平台访问地址>/clusters/<cluster>/prometheus
Prometheus	<平台访问地址>/clusters/<cluster>/prometheus-0
Alertmanager	<平台访问地址>/clusters/<cluster>/alertmanager

## 安装 ACP 监控与 VictoriaMetrics 插件

### 前提条件

- 如果仅安装 VictoriaMetrics 代理，请确保已在其他集群中安装 VictoriaMetrics Center。

### 安装步骤

1. 导航至 应用商店管理 > 集群插件 并选择目标集群。
2. 找到 **ACP 监控与 VictoriaMetrics** 插件并点击 安装。
3. 配置以下参数：

参数	说明
规模配置	支持三种配置：小规模、中规模和大规模： - 默认值基于平台的推荐负载测试值设置 - 可根据实际集群规模选择或自定义配额 - 默认值会随着平台版本更新；对于固定配置，建议使用自定义设置
仅安装代理	- 关闭：安装完整的 VictoriaMetrics 组件套件 - 开启：仅安装 VMAgent 采集组件，该组件依赖于 VictoriaMetrics Center

参数	说明
<b>VictoriaMetrics Center</b>	选择已安装完整 VictoriaMetrics 组件的集群
存储类型	<ul style="list-style-type: none"><li>- <b>LocalVolume</b>：本地存储，数据存储在指定节点</li><li>- <b>StorageClass</b>：使用存储类自动生成持久卷</li><li>- <b>PV</b>：利用已有的持久卷</li></ul>
副本数	<p>设置监控组件容器组的数量：</p> <ul style="list-style-type: none"><li>- LocalVolume 存储类型不支持多副本</li><li>- 其他存储类型请参考界面提示进行配置</li></ul>
参数配置	<p>可调整监控组件的数据参数</p> <p>注意：数据可能会在被删除前暂时超过保留期限</p>

4. 点击 安装 完成安装。

# 架构

## 监控模块架构

整体架构说明

监控系统

告警系统

通知系统

## 监控组件选型指南

重要说明

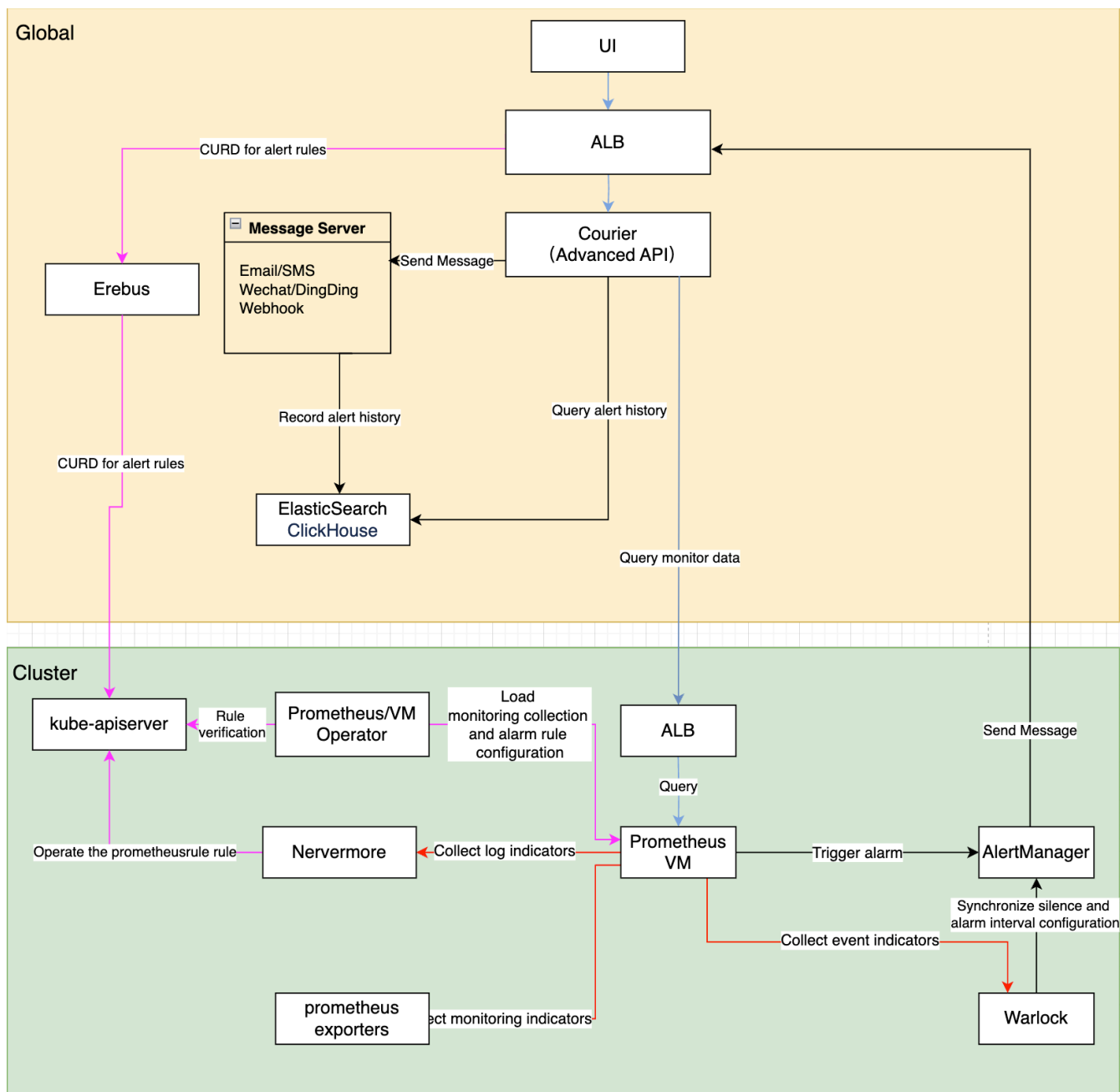
组件清单

架构对比

功能对比

安装方案建议

# 监控模块架构



## 目录

整体架构说明

## 监控系统

数据采集与存储

数据查询与可视化

## 告警系统

告警规则管理

告警处理流程

实时告警状态

## 通知系统

通知配置管理

通知服务器管理

---

# 整体架构说明

监控系统由以下核心功能模块组成：

### 1. 监控系统

- 数据采集与存储：从多个来源收集和持久化监控指标
- 数据查询与可视化：提供灵活的监控数据查询和展示能力

### 2. 告警系统

- 告警规则管理：配置和管理告警策略
- 告警触发与通知：评估告警规则并发送通知
- 实时告警状态：提供系统当前告警状态的实时视图

### 3. 通知系统

- 通知配置：管理通知模板、联系人组及通知策略
- 通知服务器：管理各种通知渠道的配置

# 监控系统

---

# 数据采集与存储

## 1. Prometheus/VictoriaMetrics Operator 的职责：

- 加载并验证监控采集配置
- 加载并验证告警规则配置
- 将配置同步到 Prometheus/VictoriaMetrics 实例

## 2. 监控数据来源：

- Nevermore：生成日志相关的指标
- Warlock：生成事件相关的指标
- Prometheus/VictoriaMetrics：通过 ServiceMonitor 发现并采集多种 exporters 的指标

# 数据查询与可视化

## 1. 监控数据查询流程：

- 浏览器发起查询请求（路径：`/platform/monitoring.alauda.io/v1beta1`）
- ALB 将请求转发至 Courier 组件
- Courier API 处理查询：
  - 内置指标：通过 indicators 接口获取 PromQL 并查询
  - 自定义指标：直接转发 PromQL 到监控组件
- 监控仪表板获取数据并展示

## 2. 监控仪表板管理流程：

- 用户访问 `global` 集群的 ALB（路径：`/kubernetes/集群名/apis/ait.alauda.io/v1alpha2/MonitorDashboard`）
- ALB 转发请求至 Erebus 组件
- Erebus 路由请求到目标监控集群
- Warlock 组件负责：
  - 验证监控仪表板配置的合法性



- 管理 MonitorDashboard CR 资源

# 告警系统

## 告警规则管理

告警规则配置流程：

1. 用户访问 `global` 集群的 ALB (路径：`/kubernetes/集群名/apis/monitoring.coreos.com/v1/prometheusrules`)
2. 请求经过 ALB -> Erebus -> 目标集群 kube-apiserver
3. 各组件的职责：
  - Prometheus/VictoriaMetrics Operator：
    - 验证告警规则的合法性
    - 管理 PrometheusRule CR
  - Nevermore：监听并处理日志告警指标
  - Warlock：监听并处理事件告警指标

## 告警处理流程

1. 告警评估：
  - PrometheusRule/VMRule 定义告警规则
  - Prometheus/VictoriaMetrics 定期评估规则
2. 告警通知：
  - 一旦触发，告警会发送至 Alertmanager
  - Alertmanager -> ALB -> Courier API
  - Courier API 负责通知的分发
3. 告警存储：

- 告警历史存储在 Elasticsearch/ClickHouse 中

## 实时告警状态

### 1. 状态收集：

- `global` 集群的 Courier 生成指标：
  - `cpaas_active_alerts`：当前活动告警
  - `cpaas_active_silences`：当前静默配置
- Global Prometheus 每 15 秒收集一次数据

### 2. 状态展示：

- 前端通过 Courier API 查询并展示实时状态

## 通知系统

### 通知配置管理

通知模板、通知联系人组和通知策略的管理流程如下：

#### 1. 用户通过浏览器访问 `global` 集群的标准 API

- 访问路径：`/apis/ait.alauda.io/v1beta1/namespaces/cpaas-system`

#### 2. 管理相关资源：

- 通知模板：`apiVersion: "ait.alauda.io/v1beta1", kind: "NotificationTemplate"`
- 通知联系人组：`apiVersion: "ait.alauda.io/v1beta1", kind: "NotificationGroup"`
- 通知策略：`apiVersion: "ait.alauda.io/v1beta1", kind: "Notification"`

#### 3. Courier 负责：

- 验证通知模板的合法性
- 验证通知联系人组的合法性
- 验证通知策略的合法性

# 通知服务器管理

## 1. 用户通过浏览器访问 `global` 集群的 ALB

- 访问路径：`/kubernetes/global/api/v1/namespaces/cpaas-system/secrets`

## 2. 管理并提交通知服务器配置

- 资源名称：`platform-email-server`

## 3. Courier 负责：

- 验证通知服务器配置的合法性

# 监控组件选型指南

在安装集群监控时，平台提供了 VictoriaMetrics 和 Prometheus 两种监控组件供您选择。本文将详细介绍这两种组件的特点和适用场景，帮助您做出最适合的选择。

## 目录

### 重要说明

#### 组件清单

- Prometheus 相关组件

- VictoriaMetrics 相关组件

#### 架构对比

- Prometheus 架构

- VictoriaMetrics 架构

#### 功能对比

#### 安装方案建议

- 监控安装架构总览

- Prometheus 安装方式

- VictoriaMetrics 安装方式

#### 选择建议

- 适合使用 VictoriaMetrics 的场景

- 适合使用 Prometheus 的场景

## 重要说明

- 集群监控组件安装时只能选择 VictoriaMetrics 或 Prometheus 其中之一。
- 从 3.18 版本开始，VictoriaMetrics 已升级至 Beta 版本，具备生产环境使用条件。
- VictoriaMetrics 适合有高可用需求和多集群监控场景。
- Prometheus 适合单集群且规模较小的监控场景。

## 组件清单

### Prometheus 相关组件

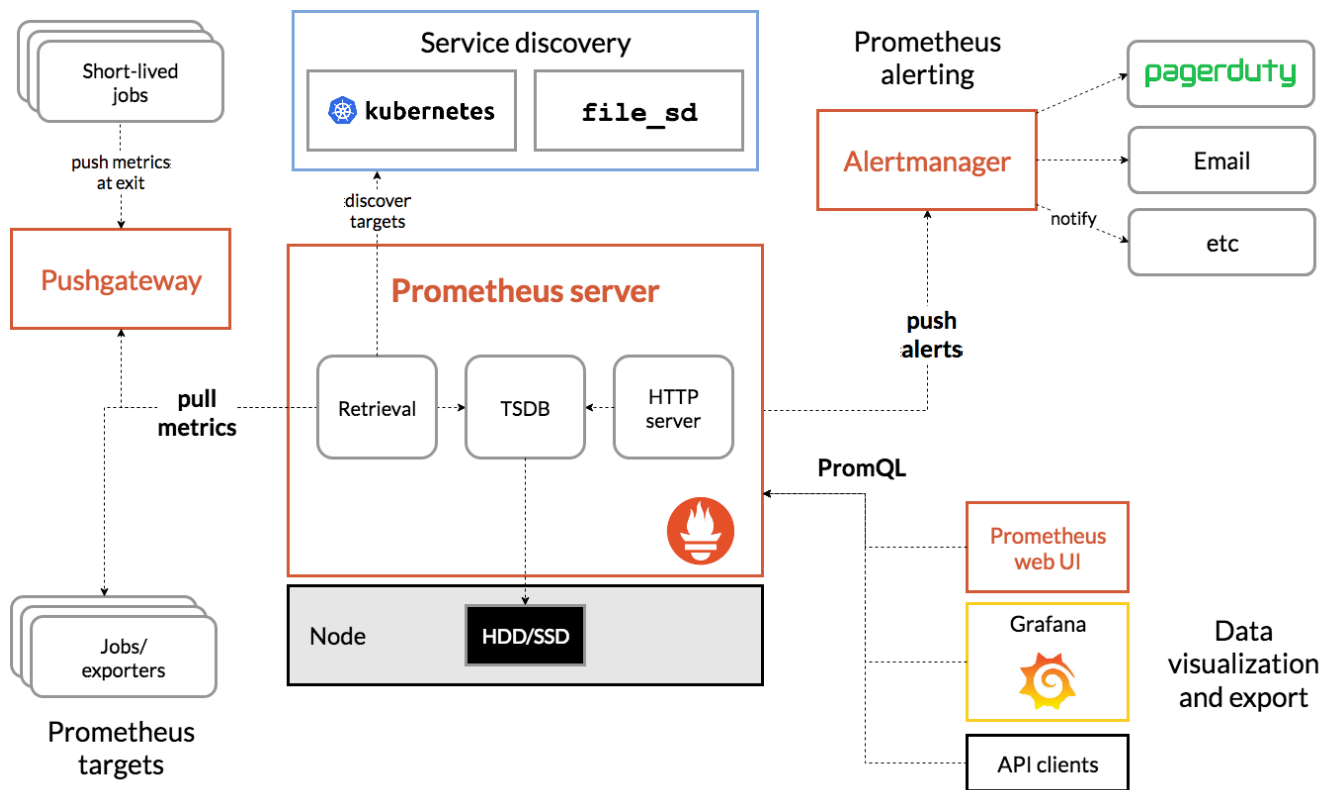
组件名称	功能描述
Prometheus Server	核心服务器，负责监控数据的采集、存储和查询
Exporters	监控数据采集组件，通过 HTTP 接口暴露监控指标
AlertManager	告警管理中心，处理告警规则和通知
PushGateway	支持监控数据的推送模式，用于特殊网络环境下的数据中转

### VictoriaMetrics 相关组件

组件名称	功能描述
VMStorage	监控数据存储引擎
VMInsert	数据写入组件，负责数据分发和存储
VMSelect	查询服务组件，提供数据查询能力
VMAlert	告警规则评估和处理组件
VMAgent	监控指标采集组件

# 架构对比

## Prometheus 架构



Prometheus 是一个成熟的开源监控系统，是 CNCF 继 Kubernetes 之后的第二个毕业项目。它具有以下特点：

- 强大的数据采集能力。
- 灵活的查询语言 PromQL。
- 完善的生态系统。
- 支持数千节点规模的集群监控。

## VictoriaMetrics 架构

## Clients

*vmselect* fully supports PromQL and can be used as Prometheus datasource in Grafana

## Stateless

*vmselect* fetches and merges data from *vmstorage* during queries

## Statefull

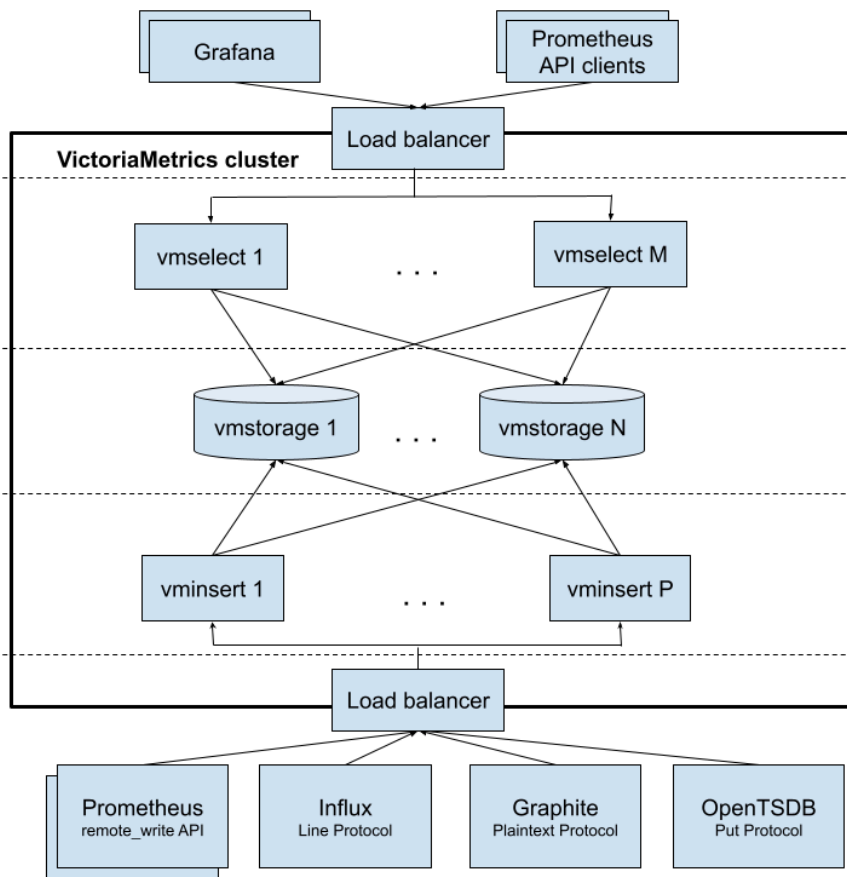
*vmstorage* stores time series data

## Stateless

*vminsert* spreads time series across available *vmstorage* nodes

## Writers

Multiple Prometheus instances may write data to VictoriaMetrics cluster  
There is support for other ingestion protocols



VictoriaMetrics 是新一代高性能时序数据库和监控解决方案，具有以下优势：

- 更高的数据压缩比。
- 更低的资源消耗。
- 原生支持集群高可用。
- 更简单的运维管理。

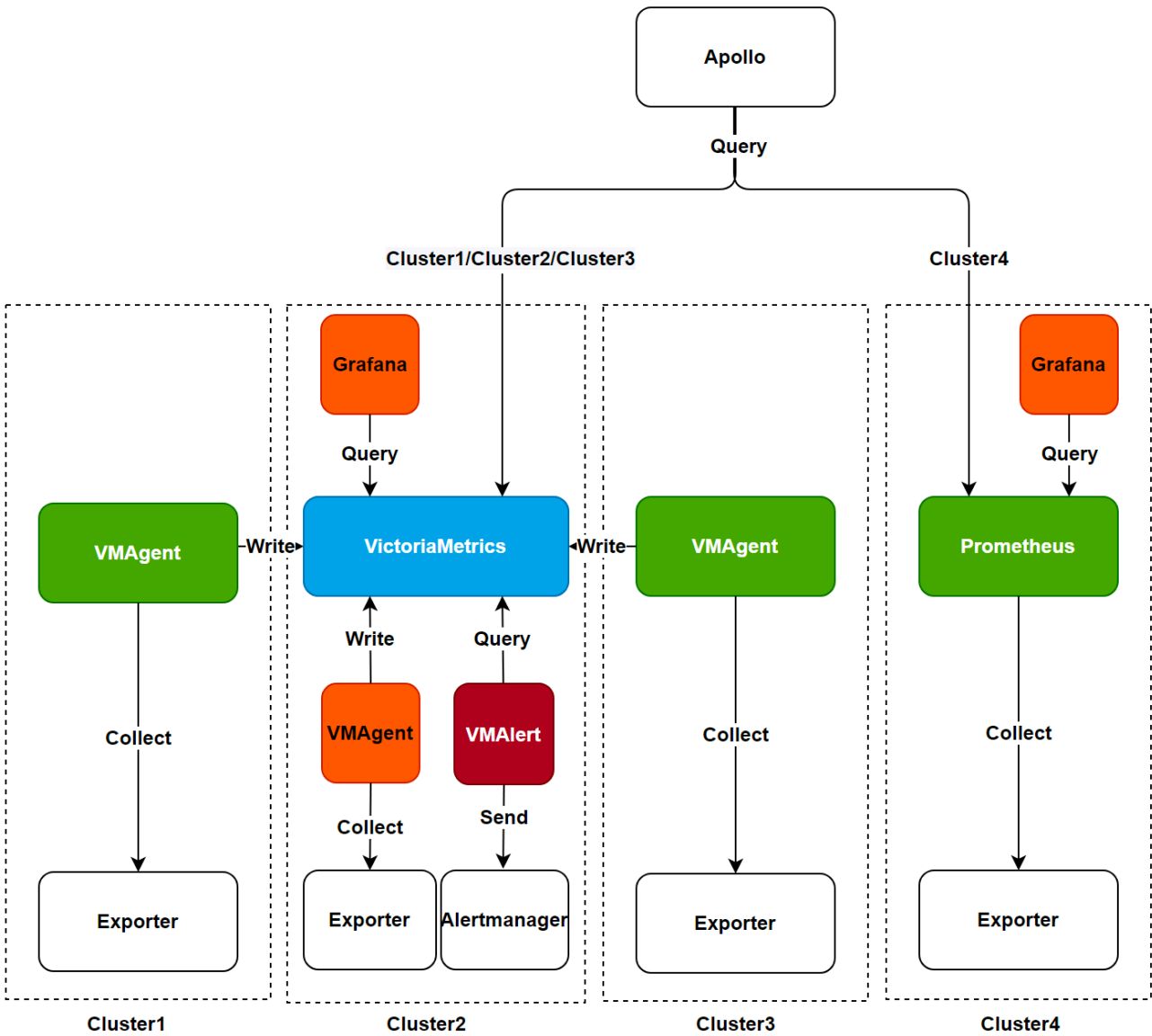
## 功能对比

特性	Prometheus	VictoriaMetrics	说明
高可用安装	✗	✓	VictoriaMetrics 支持真正的集群高可用，数据一致性更好
单节点安装	✓	✓	两者都支持单节点安装模式
长期数据存储	需要远程存储	原生支持	VictoriaMetrics 更适合长期数据存储

特性	Prometheus	VictoriaMetrics	说明
资源效率	较高	更优	VictoriaMetrics 具有更好的资源利用率
社区支持	非常成熟	快速发展	Prometheus 具有更大的社区生态

## 安装方案建议

### 监控安装架构总览



上图展示了平台支持的监控组件安装架构和数据流向。平台提供以下两种安装方式供选择：

注意：替换监控组件时，请确保已完全卸载现有组件，且监控数据不支持跨组件迁移。



## Prometheus 安装方式

该方式对应上图中 **cluster4** 的架构：

- 使用 Prometheus 组件采集和处理监控数据。
- 通过监控面板查询并展示数据。
- 适合单集群场景。

## VictoriaMetrics 安装方式

VictoriaMetrics 支持以下两种安装模式：

### 1. 单集群安装模式

- 对应上图中 **cluster2** 的架构。
- 所有 VictoriaMetrics 组件安装在同一集群。
- 使用 VMAgent 采集数据并写入 VictoriaMetrics。
- VMAAlert 负责告警规则评估。
- 通过监控面板查询展示数据。

提示：建议数据规模在每秒 100W 以下时使用此模式。

### 2. 多集群安装模式

- 对应上图中 **cluster1/cluster2/cluster3** 的架构。
- 在业务集群安装 VMAgent 作为数据采集代理。
- VMAgent 将数据写入中心监控集群的 VictoriaMetrics。
- 支持多集群统一监控管理。

提示：安装 VMAgent 前需确保已在监控集群中安装 VictoriaMetrics 服务。

## 选择建议

## 适合使用 **VictoriaMetrics** 的场景

- 高性能和可扩展性需求：适合处理高吞吐量数据和长期存储的监控场景。
- 成本效益考虑：需要优化存储和计算资源成本。
- 高可用需求：需要监控组件高可用保障。
- 多集群管理：需要统一管理多个集群的监控数据。

## 适合使用 **Prometheus** 的场景

- 单集群小规模：监控规模较小，无高可用需求。
- 现有 **Prometheus** 用户：已有完整的 Prometheus 监控体系。
- 简单稳定需求：追求简单可靠的监控方案。
- 深度生态集成：与 Prometheus 生态紧密集成且迁移成本高。

# 核心概念

---

## 目录

### Monitoring

Metrics

PromQL

Built-in Indicators

Exporter

ServiceMonitor

Alarms

Alarm Rules

Alarm Policies

Notifications

Notification Policies

Notification Templates

Monitoring Dashboard

Dashboard

Panels

Data Sources

Variables

---

## Monitoring

## Metrics

指标用于定量描述系统的运行状态，每个指标由四个基本要素组成：

- Metric Name：用于标识被监控对象，如 `cpu_usage`
- Metric Value：具体的测量数值，如 `85.5`
- Timestamp：记录测量时间
- Labels：用于多维度数据分类，如 `{pod="nginx-1", namespace="default"}`

## PromQL

PromQL 是 Prometheus 的查询语言，用于查询和聚合监控系统中的指标数据。

## Built-in Indicators

平台基于长期的运营经验预置了一系列常用的监控指标，配置告警规则或创建监控面板时可以直接使用，无需额外配置。

## Exporter

Exporter 是用于采集监控数据的组件，主要职责包括：

- 从目标系统采集原始监控数据
- 将数据转换为标准的时序指标格式
- 通过 HTTP 接口提供指标数据供查询

## ServiceMonitor

ServiceMonitor 用于声明式管理监控配置，主要定义：

- 监控目标的选择条件
- 指标采集接口的配置
- 采集任务的执行参数（间隔、超时等）

# Alarms

## Alarm Rules

告警规则定义触发告警的具体条件：

- Alarm Expression：使用 PromQL 语句描述触发告警的条件
- Alarm Threshold：触发的明确边界值
- Duration：条件需持续满足的时间
- Alarm Level：区分告警的严重程度（如 P0/P1/P2）

## Alarm Policies

告警策略将多个告警规则组织在一起进行统一配置：

- Alarm Targets：规则的目标范围
- Notification Method：告警发送的渠道
- Sending Interval：重复告警通知的时间间隔

# Notifications

## Notification Policies

通知策略管理告警消息的发送规则：

- Recipients：告警通知的目标用户
- Notification Channels：支持的消息发送方式
- Notification Templates：消息内容格式的定义

## Notification Templates

通知模板自定义告警消息的展示格式：

- Title Template : 告警消息标题的格式
- Content Template : 告警详情的组织方式
- Variable Replacement : 支持动态数据填充

# Monitoring Dashboard

## Dashboard

监控面板是多个相关面板的集合，提供系统状态的整体视图。支持灵活的布局排列，可按行或列组织面板。

## Panels

面板是监控数据的可视化表现，支持多种展示类型。

## Data Sources

监控数据源的配置。目前仅支持当前集群的监控组件作为数据源，暂不支持自定义数据源。

## Variables

变量作为值的占位符，可用于指标查询。通过监控面板顶部的变量选择器，可以动态调整查询条件，实现图表内容的实时更新。

# 操作指南

## 指标管理

查看平台组件暴露的指标

查看 Prometheus / VictoriaMetrics 存储的指标

查看平台内置定义的所有指标

集成外部指标

## 管理监控面板

功能概述

管理监控面板

管理 panels

通过 CLI 创建监控面板

常用函数和变量

## 告警管理

功能概述

主要功能

功能优势

通过 UI 创建告警策略

通过 CLI 创建资源告警

通过 CLI 创建事件告警

## 通知管理

功能概述

主要功能

通知服务器

通知联系人组

通知模板

通知规则

通知

## 探针管理

功能概述

黑盒监控

黑盒告警

自定义 Blackbox

通过 CLI 创建黑盒

参考信息

# 指标管理

平台的监控系统基于 Prometheus / VictoriaMetrics 收集的指标。本文档将指导您如何管理这些指标。

## 目录

### 查看平台组件暴露的指标

查看 Prometheus / VictoriaMetrics 存储的所有指标

前提条件

操作步骤

查看平台内置定义的所有指标

前提条件

操作步骤

集成外部指标

前提条件

操作步骤

## 查看平台组件暴露的指标

平台内集群组件的监控方式是通过 `ServiceMonitor` 抽取暴露的指标。平台中的指标均通过 `/metrics` 端点公开。您可以使用以下示例命令查看平台中某个组件暴露的指标：



```
curl -s http://<Component IP>:<Component metrics port>/metrics | grep 'TYPE\\|HELP'
```

示例输出：

```
# HELP controller_runtime_active_workers Number of currently used workers per controller
# TYPE controller_runtime_active_workers gauge
# HELP controller_runtime_max_concurrent_reconciles Maximum number of concurrent reconciles per controller
# TYPE controller_runtime_max_concurrent_reconciles gauge
# HELP controller_runtime_reconcile_errors_total Total number of reconciliation errors per controller
# TYPE controller_runtime_reconcile_errors_total counter
# HELP controller_runtime_reconcile_time_seconds Length of time per reconciliation per controller
```

## 查看 Prometheus / VictoriaMetrics 存储的所有指标

您可以查看集群中可用的指标列表，帮助您基于这些指标编写所需的 PromQL。

### 前提条件

1. 您已获取用户 Token
2. 您已获取平台地址

### 操作步骤

运行以下命令，通过 `curl` 获取指标列表：

```
curl -k -X 'GET' -H 'Authorization: Bearer <Your token>' 'https://<Your platform access address>/v2/metrics/<Your cluster name>/prometheus/label/___name___/values'
```

示例输出：

```
{
  "status": "success",
  "data": [
    "ALERTS",
    "ALERTS_FOR_STATE",
    "advanced_search_cached_resources_count",
    "alb_error",
    "alertmanager_alerts",
    "alertmanager_alerts_invalid_total",
    "alertmanager_alerts_received_total",
    "alertmanager_cluster_enabled"]
}
```

## 查看平台内置定义的所有指标

为了简化用户使用，平台内置了大量常用指标。您在配置告警或监控面板时可以直接使用这些指标，无需自行定义。以下介绍如何查看这些指标。

### 前提条件

1. 您已获取用户 Token
2. 您已获取平台地址

### 操作步骤

运行以下命令，通过 `curl` 获取指标列表：

```
curl -k -X 'GET' -H 'Authorization: Bearer <Your token>' 'https://<Your platform access address>/v2/metrics/<Your cluster name>/indicators'
```

示例输出：

```
[
  {
    "alertEnabled": true, ❶
    "annotations": {
      "cn": "计算组件中容器的 CPU 利用率",
      "descriptionEN": "Cpu utilization for pods in workload",
      "descriptionZH": "计算组件中容器的 CPU 利用率",
      "displayNameEN": "CPU utilization of the pods",
      "displayNameZH": "计算组件中容器的 CPU 利用率",
      "en": "Cpu utilization for pods in workload",
      "features": "SupportDashboard", ❷
      "summaryEN": "CPU usage rate {{.externalLabels.comparison}}{{.externalLabels.threshold}} of Pod ({{.labels.pod}})",
      "summaryZH": "Pod ({{.labels.pod}}) 的 CPU 使用率 {{.externalLabels.comparison}}{{.externalLabels.threshold}}"
    },
    "displayName": "计算组件中容器的 CPU 利用率",
    "kind": "workload",
    "multipleEnabled": true, ❸
    "name": "workload.pod.cpu.utilization",
    "query": "avg by (kind,name,namespace,pod) (avg by (kind,name,namespace,pod,container)(cpaas_advanced_container_cpu_usage_seconds_totalirate5m{kind=~\"{{.kind}}\",name=~\"{{.name}}\",namespace=~\"{{.namespace}}\",container!=\"\",container!=\"POD\"})) / avg by (kind,name,namespace,pod,container)(cpaas_advanced_kube_pod_container_resource_limits{kind=~\"{{.kind}}\",name=~\"{{.name}}\",namespace=~\"{{.namespace}}\",resource=\"cpu\"}))", ❹
    "summary": "Pod ({{.labels.pod}}) 的 CPU 使用率 {{.externalLabels.comparison}}{{.externalLabels.threshold}}",
    "type": "metric",
    "unit": "%",
    "legend": "{{.namespace}}/{{.pod}}",
    "variables": [ ❺
      "namespace",
      "name",
      "kind"
    ]
  }
]
```

## 1. 该指标是否支持用于配置告警

2. 该指标是否支持用于监控面板
3. 该指标是否支持用于配置多资源告警
4. 该指标定义的 PromQL 语句
5. 该指标 PromQL 语句中可用的变量

## 集成外部指标

除了平台内置指标外，您还可以通过 `ServiceMonitor` 或 `PodMonitor` 集成您的应用或第三方应用暴露的指标。本节以以 Pod 形式安装在同一集群中的 Elasticsearch Exporter 为例进行说明。

### 前提条件

您已安装应用并通过指定接口暴露指标。本文档假设您的应用安装在 `cpaas-system` 命名空间，并暴露了 `http://<elasticsearch-exporter-ip>:9200/_prometheus/metrics` 端点。

### 操作步骤

1. 创建 Service/Endpoint 以供 Exporter 暴露指标

```
apiVersion: v1
kind: Service
metadata:
  labels:
    chart: elasticsearch
    service_name: cpaas-elasticsearch
  name: cpaas-elasticsearch
  namespace: cpaas-system
spec:
  clusterIP: 10.105.125.99
  ports:
    - name: cpaas-elasticsearch
      port: 9200
      protocol: TCP
      targetPort: 9200
  selector:
    service_name: cpaas-elasticsearch
  sessionAffinity: None
  type: ClusterIP
```

2. 创建 `ServiceMonitor` 对象描述您的应用暴露的指标：

```

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: cpaas-monitor
    chart: cpaas-monitor
    heritage: Helm
    prometheus: kube-prometheus ❶
    release: cpaas-monitor
  name: cpaas-elasticsearch-Exporter
  namespace: cpaas-system ❷
spec:
  jobLabel: service_name ❸
  namespaceSelector: ❹
    any: true
  selector: ❺
    matchExpressions:
      - key: service_name
        operator: Exists
  endpoints:
    - port: cpaas-elasticsearch ❻
      path: /_prometheus/metrics ❼
      interval: 60s ❽
      honorLabels: true
      basicAuth: ❾
        password:
          key: ES_PASSWORD
          name: acp-config-secret
        username:
          key: ES_USER
          name: acp-config-secret

```

1. ServiceMonitor 应同步到哪个 Prometheus ; operator 会根据 Prometheus CR 的 serviceMonitorSelector 配置监听对应的 ServiceMonitor 资源。如果 ServiceMonitor 的标签不匹配 Prometheus CR 的 serviceMonitorSelector 配置，则该 ServiceMonitor 不会被 operator 监控。
2. operator 会根据 Prometheus CR 的 serviceMonitorNamespaceSelector 配置监听哪些命名空间的 ServiceMonitor ; 如果 ServiceMonitor 不在 Prometheus CR 的 serviceMonitorNamespaceSelector 中，则该 ServiceMonitor 不会被 operator 监控。

3. Prometheus 收集的指标会添加一个 job 标签，值为对应 jobLabel 的 service 标签值。
4. ServiceMonitor 根据 namespaceSelector 配置匹配对应的 Service。
5. ServiceMonitor 根据 selector 配置匹配 Service。
6. ServiceMonitor 根据 port 配置匹配 Service 的端口。
7. 访问 Exporter 的路径，默认为 /metrics。
8. Prometheus 抓取 Exporter 指标的间隔。
9. 如果访问 Exporter 路径需要认证，则需添加认证信息；也支持 bearer token、tls 认证等方式。

### 3. 检查 ServiceMonitor 是否被 Prometheus 监控

访问监控组件的 UI，查看是否存在 job `cpaas-elasticsearch-exporter`。

- Prometheus UI 地址：`https://<Your platform access address>/clusters/<Cluster name>/prometheus-0/targets`
- VictoriaMetrics UI 地址：`https://<Your platform access address>/clusters/<Cluster name>/vmselect/vmui/?#/metrics`

# 告警管理

---

## 目录

### 功能概述

主要功能

功能优势

通过 UI 创建告警策略

前提条件

操作步骤

选择告警类型

配置告警规则

其他配置

其他说明

通过 CLI 创建资源告警

前提条件

操作步骤

通过 CLI 创建事件告警

前提条件

操作步骤

通过告警模板创建告警策略

前提条件

操作步骤

创建告警模板

使用告警模板创建告警策略

设置告警静默



通过 UI 设置

通过 CLI 设置

配置告警规则的建议

---

## 功能概述

平台的告警管理功能旨在帮助用户全面监控并及时发现系统异常。通过利用预装的系统告警和灵活的自定义告警能力，结合标准化的告警模板和分级管理机制，为运维人员提供完整的告警解决方案。

无论是平台管理员还是业务人员，都可以在各自权限范围内便捷地配置和管理告警策略，实现对平台资源的有效监控。

## 主要功能

- 内置系统告警策略：基于常见故障诊断思路，预设丰富的告警规则，适用于 `global` 集群和工作负载集群。
- 自定义告警规则：支持基于多种数据源创建告警规则，包括预设监控指标、自定义监控指标、黑盒监控项、平台日志数据和平台事件数据。
- 告警模板管理：支持创建和管理标准化告警模板，便于快速应用于类似资源。
- 告警通知集成：支持通过多种渠道将告警信息推送给运维人员。
- 告警视图隔离：区分平台管理告警和业务告警，确保不同角色人员关注各自的告警信息。
- 实时告警查看：提供实时告警，集中展示当前处于告警状态的资源数量及详细告警信息。
- 告警历史查看：支持查看一段时间内的历史告警记录，方便运维人员和管理员分析近期监控告警情况。

## 功能优势

- 监控覆盖全面：支持对集群、节点、计算组件等多种资源类型的监控，内置丰富的系统告警策略，无需额外配置即可使用。
-

- 告警管理高效：通过告警模板实现标准化配置，提高运维效率；告警视图分离，便于不同角色人员快速定位相关告警。
- 问题发现及时：告警通知自动触发，确保及时发现问题，支持多渠道告警推送，实现主动规避问题。
- 权限管理严格：告警策略访问控制严格，确保告警信息安全可控。

## 通过 UI 创建告警策略

### 前提条件

- 已配置通知策略（若需配置自动告警通知）。
- 目标集群已安装监控组件（创建基于监控指标的告警策略时必需）。
- 目标集群已安装日志存储组件和日志采集组件（创建基于日志和事件的告警策略时必需）。

### 操作步骤

1. 进入 运维中心 > 告警 > 告警策略。
2. 点击 创建告警策略。
3. 配置基础信息。

### 选择告警类型

#### 资源告警

- 按资源类型分类的告警类型（如命名空间下的 deployment 状态）。
- 资源选择说明：
  - 未选择参数时默认为“任意”，支持自动关联新添加的资源。
  - 选择“全选”时，仅对当前资源生效。
  - 多个命名空间时，资源名称支持正则表达式（如 `cert.*`）。

#### 事件告警

- 按具体事件分类的告警类型（如 Pod 状态异常）。

- 默认选择指定资源下的所有资源，支持自动关联新添加的资源。

## 配置告警规则

点击 添加告警规则，根据告警类型配置以下参数：

### 资源告警参数

参数	说明
Expression	Prometheus 格式的监控指标算法表达式，如 <code>rate(node_network_receive_bytes{instance="\$server",device!~"lo"}[5m])</code>
Metric Unit	自定义监控指标单位，可手动输入或从平台预设单位中选择
Legend Parameter	控制图表中曲线对应的名称，格式为 <code>{{.LabelName}}</code> ，例如 <code>{{.hostname}}</code>
Time Range	日志/事件查询的时间窗口
Log Content	日志内容查询字段（如 Error），多个查询字段之间用 OR 连接
Event Reason	事件原因查询字段（Reason，如 BackOff、Pulling、Failed 等），多个查询字段之间用 OR 连接
Trigger Condition	由比较运算符、告警阈值和持续时间组成的条件（可选）。根据实时值/日志数/事件数与告警阈值的比较及实时值在阈值范围内的持续时间判断是否触发告警。
alert Level	分为 Critical、Serious、Warning 和 Info 四个等级。可根据告警规则对业务的影响合理设置对应资源的告警等级。

### 事件告警参数

参数	说明
Time Range	事件查询的时间窗口

参数	说明
Event Monitoring Item	支持监控事件级别或事件原因，多个字段之间用 OR 连接
Trigger Condition	基于事件数量进行比较判断
alert Level	与资源告警等级定义相同

## 其他配置

1. 选择一个或多个已创建的通知策略。
2. 配置告警发送间隔。
  - 全局：使用平台默认配置。
  - 自定义：可根据告警等级设置不同的发送间隔。
  - 选择“不重复”时，仅在告警触发和恢复时发送通知。

## 其他说明

1. 在告警规则的“更多”选项中，可以设置标签和注解。
2. 标签和注解的配置请参考 [Prometheus Alerting Rules Documentation](#)。
3. 注意：标签中不要使用 `$value` 变量，可能导致告警异常。

# 通过 CLI 创建资源告警

## 前提条件

- 已配置通知策略（若需配置自动告警通知）。
- 目标集群已安装监控组件（创建基于监控指标的告警策略时必需）。
- 目标集群已安装日志存储组件和日志采集组件（创建基于日志和事件的告警策略时必需）。

## 操作步骤

1. 新建 YAML 配置文件，命名为 `example-alerting-rule.yaml`。
2. 在 YAML 文件中添加 PrometheusRule 资源并提交。以下示例创建了名为 policy 的新告警策略：

告警名称	告警内容	告警时间	告警级别	告警状态	告警来源	告警处理人	告警处理时间	告警处理结果
容器CPU使用率过高	容器CPU使用率超过90%	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器内存使用率过高	容器内存使用率超过90%	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器网络I/O过高	容器网络I/O超过100MB/s	2023-10-27 10:10:10	中	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器磁盘I/O过高	容器磁盘I/O超过100MB/s	2023-10-27 10:10:10	中	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器重启次数过多	容器重启次数超过5次	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器退出码非0	容器退出码非0	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器健康检查失败	容器健康检查失败	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器日志大小超过限制	容器日志大小超过1GB	2023-10-27 10:10:10	中	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器资源配额超限	容器资源配额超限	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理
容器安全扫描失败	容器安全扫描失败	2023-10-27 10:10:10	高	未处理	容器引擎	张三	2023-10-27 10:15:10	已处理

```

apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global # 告警所在集群名称
    alert.cpaas.io/kind: Cluster # 资源类型
    alert.cpaas.io/name: global # 资源对象, 支持单个、多选（用 | 分隔）或任意
    (.*)
    alert.cpaas.io/namespace: cpaas-system # 告警对象所在命名空间, 支持单
    个、多选（用 | 分隔）或任意 (.*)
    alert.cpaas.io/notifications: '["test"]'
    alert.cpaas.io/repeat-config: '{"Critical":"never","High":"5m","Med
    ium":"5m","Low":"5m"}'
    alert.cpaas.io/rules.description: '{} '
    alert.cpaas.io/rules.disabled: '[]'
    alert.cpaas.io/subkind: ''
    cpaas.io/description: ''
    cpaas.io/display-name: policy # 告警策略显示名称
  labels:
    alert.cpaas.io/owner: System
    alert.cpaas.io/project: cpaas-system
    cpaas.io/source: Platform
    prometheus: kube-prometheus
    rule.cpaas.io/cluster: global
    rule.cpaas.io/name: policy
    rule.cpaas.io/namespace: cpaas-system
  name: policy
  namespace: cpaas-system
spec:
  groups:
    - name: general # 告警规则组名称
      rules:
        - alert: cluster.pod.status.phase.not.running-tx1ob-e998f0b9485
          4ee1eade5ae79279e005a
            annotations:
              alert_current_value: '{{ $value }}' # 当前值通知, 保持默认
              expr: (count(min by(pod)(kube_pod_container_status_ready{})) !
              =1) or on() vector(0))>2
              for: 30s # 持续时间
            labels:
              alert_cluster: global # 告警所在集群名称
              alert_for: 30s # 持续时间
              alert_indicator: cluster.pod.status.phase.not.running # 告警

```

规则指标名称（自定义告警指标名称为 custom）

```

alert_indicator_aggregate_range: '30' # 告警规则聚合时间，单位
秒

alert_indicator_blackbox_name: '' # 黑盒监控项名称
alert_indicator_comparison: '>' # 告警规则比较方式
alert_indicator_query: '' # 告警规则日志查询（仅日志告警）
alert_indicator_threshold: '2' # 告警规则阈值
alert_indicator_unit: '' # 告警规则指标单位
alert_involved_object_kind: Cluster # 告警规则所属对象类型：Cluster|Node|Deployment|Daemonset|Statefulset|Middleware|Microservice|Storage|VirtualMachine

alert_involved_object_name: global # 告警规则所属对象名称
alert_involved_object_namespace: '' # 告警规则所属对象命名空间
alert_name: cluster.pod.status.phase.not.running-tx1ob # 告警规则名称

alert_namespace: cpaas-system # 告警规则所在命名空间
alert_project: cpaas-system # 告警规则所属对象项目名称
alert_resource: policy # 告警规则所在告警策略名称
alert_source: Platform # 告警规则所在告警策略数据类型：Platform-平台数据 Business-业务数据

severity: High # 告警规则严重级别：Critical-严重、High-严重、Medium-警告、Low-信息

```

## 通过 CLI 创建事件告警

### 前提条件

- 已配置通知策略（若需配置自动告警通知）。
- 目标集群已安装监控组件（创建基于监控指标的告警策略时必需）。
- 目标集群已安装日志存储组件和日志采集组件（创建基于日志和事件的告警策略时必需）。

### 操作步骤

1. 新建 YAML 配置文件，命名为 `example-alerting-rule.yaml`。
2. 在 YAML 文件中添加 PrometheusRule 资源并提交。以下示例创建了名为 policy2 的新告警策略：





```

apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global
    alert.cpaas.io/events.scope:
      '[{"names":["argocd-gitops-redis-ha-haproxy"],"kind":"Deployment", "operator":"=", "namespaces":["*"]}]'
      # names: 事件告警的资源名称；若名称为空, operator 无效。
      # kind: 触发事件告警的资源类型。
      # namespace: 触发事件告警的资源所属命名空间。空数组表示非命名空间资源；当 ns 为 ['*'] 表示所有命名空间。
      # operator: 选择器 =, !=, =~, !~
    alert.cpaas.io/kind: Event # 告警类型, Event (事件告警)
    alert.cpaas.io/name: '' # 资源告警使用, 事件告警为空
    alert.cpaas.io/namespace: cpaas-system
    alert.cpaas.io/notifications: '["acp-qwtest"]'
    alert.cpaas.io/repeat-config: '{"Critical":"never","High":"5m","Medium":"5m","Low":"5m"}'
    alert.cpaas.io/rules.description: '{}'
    alert.cpaas.io/rules.disabled: '[]'
    cpaas.io/description: ''
    cpaas.io/display-name: policy2
  labels:
    alert.cpaas.io/owner: System
    alert.cpaas.io/project: cpaas-system
    cpaas.io/source: Platform
    prometheus: kube-prometheus
    rule.cpaas.io/cluster: global
    rule.cpaas.io/name: policy2
    rule.cpaas.io/namespace: cpaas-system
  name: policy2
  namespace: cpaas-system
spec:
  groups:
    - name: general
      rules:
        - alert: cluster.event.count-6sial-34c9a378e3b6dda8401c2d728994ce2f
          # 6sial-34c9a378e3b6dda8401c2d728994ce2f 可自定义以保证唯一性
          annotations:
            alert_current_value: '{{ $value }}' # 当前值通知, 保持默认
          expr: round(((avg

```

```

        by(kind,namespace,name,reason)(increase(cpaas_event_count{namespace=~".*",id="policy2-cluster.event.count-6sial"}[300s])))
        + (avg
        by(kind,namespace,name,reason)(abs(increase(cpaas_event_count{namespace=~".*",id="policy2-cluster.event.count-6sial"}[300s]))))
        / 2)>2
# policy2 中的 id 需为告警策略名称;6sial 必须与前置告警规则名称匹配
for: 15s # 持续时间
labels:
  alert_cluster: global # 告警所在集群名称
  alert_for: 15s # 持续时间
  alert_indicator: cluster.event.count # 告警规则指标名称（自定义告警指标名称为 custom）
  alert_indicator_aggregate_range: '300' # 告警规则聚合时间，单位秒
  alert_indicator_blackbox_name: ''
  alert_indicator_comparison: '>' # 告警规则比较方式
  alert_indicator_event_reason: ScalingReplicaSet # 事件原因
  alert_indicator_threshold: '2' # 告警规则阈值
  alert_indicator_unit: pieces # 告警规则指标单位，事件告警保持不变
  alert_involved_object_kind: Event
  alert_involved_object_options: Single
  alert_name: cluster.event.count-6sial # 告警规则名称
  alert_namespace: cpaas-system # 告警规则所在命名空间
  alert_project: cpaas-system # 告警规则所属对象项目名称
  alert_repeat_interval: 5m
  alert_resource: policy2 # 告警规则所在告警策略名称
  alert_source: Platform # 告警规则所在告警策略数据类型：Platform-平台数据 Business-业务数据
  severity: High # 告警规则严重级别：Critical-严重、High-严重、Medium-警告、Low-信息

```

## 通过告警模板创建告警策略

告警模板是针对类似资源的告警规则和通知策略的组合。通过告警模板，可以轻松快速地为平台上的集群、节点或计算组件创建告警策略。

### 前提条件

- 已配置通知策略（若需配置自动告警通知）。
- 目标集群已安装监控组件（创建基于监控指标的告警策略时必需）。

# 操作步骤

## 创建告警模板

1. 在左侧导航栏点击 运维中心 > 告警 > 告警模板。
2. 点击 创建告警模板。
3. 配置告警模板的基础信息。
4. 在 告警规则 区域，点击 添加告警规则，根据以下参数说明添加告警规则：

参数	说明
Expression	Prometheus 格式的监控指标算法表达式，如 <code>rate(node_network_receive_bytes{instance="\$server",device!~"lo"}[5m])</code>
Metric Unit	自定义监控指标单位，可手动输入或从平台预设单位中选择
Legend Parameter	控制图表中曲线对应的名称，格式为 <code>{{.LabelName}}</code> ，例如 <code>{{.hostname}}</code>
Time Range	日志/事件查询的时间窗口
Log Content	日志内容查询字段（如 Error），多个查询字段之间用 OR 连接
Event Reason	事件原因查询字段（Reason，如 BackOff、Pulling、Failed 等），多个查询字段之间用 OR 连接
Trigger Condition	由比较运算符、告警阈值和持续时间组成的条件（可选）。
alert Level	分为 Critical、Serious、Warning 和 Info 四个等级。可根据告警规则对业务的影响合理设置对应资源的告警等级。

5. 点击 创建。

## 使用告警模板创建告警策略

1. 在左侧导航栏点击 运维中心 > 告警 > 告警策略。提示：可通过顶部导航栏切换目标集群。

2. 点击 创建告警策略 按钮旁的展开按钮 > 模板创建告警策略。
3. 配置部分参数，参考以下说明：

参数	说明
模板名称	选择要使用的告警模板名称。模板按集群、节点和计算组件分类。选择模板后，可查看告警模板中设置的告警规则、通知策略等信息。
资源类型	选择模板是针对 集群、节点 还是 计算组件 的告警策略模板；对应的资源名称将显示。

4. 点击 创建。

## 设置告警静默

支持对集群、节点和计算组件的告警进行静默设置。通过对特定告警策略设置静默，可以控制该告警策略下所有规则在静默期间触发时不发送通知消息。支持永久静默和自定义时间静默。

例如：平台升级或维护时，许多资源可能出现异常状态，导致大量告警触发，运维人员在升级或维护完成前频繁收到告警通知。设置告警策略静默可以避免此类情况。

注意：静默状态持续至静默结束时间后，静默设置将自动清除。

## 通过 UI 设置

1. 在左侧导航栏点击 运维中心 > 告警 > 告警策略。
2. 点击要静默的告警策略右侧操作按钮 > 设置静默。
3. 切换 告警静默 开关至开启状态。

提示：该开关控制静默设置是否生效。取消静默只需关闭开关。

4. 根据以下说明配置相关参数：

提示：若未选择静默范围或资源名称，默认为 任意，表示后续的 删除/添加 资源操作将对应删除静默/添加静默 告警策略；选择“全选”时，仅对当前选中资源范围生效，后续的 删除/添加 资源操作不做处理。

参数	说明
静默范围	静默设置生效的资源范围。
资源名称	静默设置针对的资源对象名称。
静默时间	告警静默的时间范围。告警将在静默时间开始时进入静默状态，若静默结束后告警策略仍处于告警状态或触发告警，则恢复告警通知。永久：静默设置持续至告警策略被删除。自定义：自定义静默开始和结束时间，时间间隔不少于 5 分钟。

## 5. 点击 设置。

提示：从设置静默到静默开始这段时间内，告警策略的静默状态为 静默等待，此期间策略内规则触发告警时正常发送通知；静默开始至结束期间，告警策略状态为 静默中，规则触发告警时不发送通知。

## 通过 CLI 设置

### 1. 指定要设置静默的告警策略资源名称，执行以下命令：

```
kubectl edit PrometheusRule <TheNameOfTheAlertPolicyYouWantToSet>
```

### 2. 按示例修改资源，添加静默注解并提交。

```

---
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global
    alert.cpaas.io/kind: Node
    alert.cpaas.io/name: 0.0.0.0
    alert.cpaas.io/namespace: cpaas-system
    alert.cpaas.io/notifications: '[]'
    alert.cpaas.io/rules.description: '{} '
    alert.cpaas.io/rules.disabled: '[]'
    alert.cpaas.io/rules.version: '23'
    alert.cpaas.io/silence.config:
      '{"startsAt":"2025-02-08T08:01:37Z","endsAt":"2025-02-22T08:01:37Z","creator":"leizhu@alauda.io","resources":{"nodes":[{"name":"192.168.36.11","ip":"192.168.36.11"}, {"name":"192.168.36.12","ip":"192.168.36.12"}, {"name":"192.168.36.13","ip":"192.168.36.13"}]}}'
      # 节点级告警策略的静默配置，包括开始时间、结束时间、创建者等；若静默范围包含特定节点，请按示例追加 resources.node 信息。若需对所有资源静默，无需 resources 字段。
      # alert.cpaas.io/silence.config: '{"startsAt":"2025-02-08T08:04:50Z","endsAt":"2199-12-31T00:00:00Z","creator":"leizhu@alauda.io","name":["alb-operator-ctl","apollo"],"namespace":["cpaas-system"]}'
      # 工作负载级告警策略的静默配置，包括开始时间、结束时间、创建者等；若静默范围包含特定工作负载，请按示例追加 name 和 namespace 信息。若需对所有资源静默，无需 name 和 namespace 字段。
      # 设置 endsAt 字段为 2199-12-31T00:00:00Z 表示永久静默。
    alert.cpaas.io/subkind: ''
    cpaas.io/creator: leizhu@alauda.io
    cpaas.io/description: ''
    cpaas.io/display-name: policy3
    cpaas.io/updated-at: 2025-02-08T08:01:42Z
  labels:
    ## 排除无关信息

```

## 配置告警规则的建议

更多的告警规则并不总是更好。冗余或复杂的告警规则可能导致告警风暴，增加维护负担。建议在配置告警规则前阅读以下指导，确保自定义规则既能达到预期目的，又保持高效。

- 尽量使用最少的新规则：仅创建满足特定需求的规则。通过最少数量的规则，可以构建更易管理和集中化的监控告警体系。
- 关注症状而非原因：创建通知用户症状的规则，而非症状的根本原因。这样，当相关症状出现时，用户能收到告警，并可进一步调查触发告警的根因。此策略能显著减少所需创建的规则总数。
- 变更前规划和评估需求：首先明确哪些症状重要，以及希望用户在症状出现时采取何种行动。然后评估现有规则，判断是否可通过修改实现目标，避免为每个症状创建新规则。通过修改现有规则和谨慎创建新规则，有助于简化告警体系。
- 提供清晰的告警信息：创建告警信息时，应包含症状描述、可能原因和建议操作。信息应清晰简洁，提供排查步骤或相关信息链接，帮助用户快速评估情况并做出响应。
- 合理设置严重级别：为规则分配严重级别，指示用户在症状触发告警时应如何响应。例如，将严重级别设为 Critical，表示相关人员需立即采取行动。通过设定严重级别，帮助用户判断告警响应优先级，确保紧急问题得到及时处理。





# 通知管理

## 目录

### 功能概述

主要功能

通知服务器

企业通信工具服务器

邮件服务器

Webhook 类型服务器

通知联系人组

通知模板

创建通知模板

参考变量

邮件中的特殊格式标记语言

通知规则

前提条件

操作流程

为项目设置通知规则

前提条件

操作流程

## 功能概述

通过通知功能，您可以集成平台的监控和告警功能，及时向通知接收人发送预警信息，提醒相关人员采取必要措施解决问题或避免故障。

## 主要功能

- **通知服务器**：通知服务器为平台上的通知联系人组提供发送通知消息的服务，例如邮件服务器。
- **通知联系人组**：通知联系人组是一组具有相似逻辑特征的通知接收人，通过对接收通知消息的实体进行分类，可以减少您的维护负担。
- **通知模板**：通知模板是由自定义内容、内容变量和内容格式参数组成的标准化结构，用于规范通知策略的告警通知消息的内容和格式。例如，自定义邮件通知的主题和内容。
- **通知规则**：通知规则是一组定义如何向特定联系人发送通知消息的规则。对于需要通知外部服务的场景，如告警、巡检和登录认证，必须使用通知规则。

## 通知服务器

通知服务器为平台上的接收人提供发送通知消息的服务。平台目前支持以下通知服务器：

- **企业通信工具服务器**：支持集成微信企业号、钉钉和飞书内置应用，向个人发送通知。
- **邮件服务器**：通过邮件服务器发送邮件通知。
- **Webhook 类型服务器**：支持集成企业微信群机器人、钉钉群机器人、飞书群机器人，或向您指定的服务器发送 WebHook。

### WARNING

仅能添加一个企业通信工具服务器。

## 企业通信工具服务器

### 微信企业号

1. 按照以下示例配置通知服务器参数。填写参数后，切换至 **集群管理 > 资源管理** 中的 **global** 集群，创建资源对象。

```
# 微信企业号 corpId、corpSecret、agentId 获取方式参考官方文档：https://developer.work.weixin.qq.com/document/path/90665
apiVersion: v1
kind: Secret
type: NotificationServer
metadata:
  labels:
    cpaas.io/notification.server.type: CorpWeChat
    cpaas.io/notification.server.category: Corp
  name: platform-corp-wechat-server
  namespace: cpaas-system
data:
  displayNameZh: 企业微信           # 服务器中文显示名称，默认 base64 编码
  displayNameEn: WeChat            # 服务器英文显示名称，默认 base64 编码
  corpId:                          # 企业 ID，默认 base64 编码
  corpSecret:                      # 应用密钥，默认 base64 编码
  agentId:                         # 企业应用 ID，默认 base64 编码
```

2. 创建完成后，需在平台的 用户角色管理 > 用户管理 或用户的 个人信息 中更新用户的 微信企业号 ID，确保用户能正常接收消息。

## 钉钉

1. 按照以下示例配置通知服务器参数。填写参数后，切换至 集群管理 > 资源管理 中的

`global` 集群，创建资源对象。

```
# 钉钉 appKey、appSecret、agentId 获取方式: https://open-dev.dingtalk.com/fe/app#/corp/app
apiVersion: v1
kind: Secret
type: NotificationServer
metadata:
  labels:
    cpaas.io/notification.server.type: CorpDingTalk
    cpaas.io/notification.server.category: Corp
  name: platform-corp-dingtalk-server
  namespace: cpaas-system
data:
  displayNameZh: 钉钉 # 服务器中文显示名称, 默认 base64 编码
  displayNameEn: DingTalk # 服务器英文显示名称, 默认 base64 编码
  appKey: # 应用 key, 默认 base64 编码
  appSecret: # 应用密钥, 默认 base64 编码
  agentId: # 应用 agent_id, 默认 base64 编码
```

2. 创建完成后，需在平台的 用户角色管理 > 用户管理 或用户的 个人信息 中更新用户的 钉钉 ID，确保用户能正常接收消息。

## 飞书

1. 按照以下示例配置通知服务器参数。填写参数后，切换至 集群管理 > 资源管理 中的 `global` 集群，创建资源对象。

```
# 飞书 appId、appSecret 获取方式：https://open.feishu.cn/app/
apiVersion: v1
kind: Secret
type: NotificationServer
metadata:
  labels:
    cpaas.io/notification.server.type: CorpFeishu
    cpaas.io/notification.server.category: Corp
  name: platform-corp-feishu-server
  namespace: cpaas-system
data:
  displayNameZh: 飞书 # 服务器中文显示名称，默认 base64 编码
  displayNameEn: Feishu # 服务器英文显示名称，默认 base64 编码
  appId: # 应用 ID，默认 base64 编码
  appSecret: # 应用密钥，默认 base64 编码
```

2. 创建完成后，需在平台的 用户角色管理 > 用户管理 或用户的 个人信息 中更新用户的 飞书 ID，确保用户能正常接收消息。

## 邮件服务器

1. 在左侧导航栏点击 平台设置 > 通知服务器。
2. 点击 立即配置。
3. 参考以下说明配置相关参数。

参数	说明
服务地址	支持 SMTP 协议的通知服务器地址，例如 <code>smtp.yeah.net</code> 。
端口	通知服务器端口号。勾选 使用 <b>SSL</b> 时，需填写 SSL 端口号。
服务器配置	<p>使用 <b>SSL</b>：安全套接字层（SSL）是一种标准的安全技术。SSL 开关用于控制是否在服务器和客户端之间建立加密连接。</p> <p>跳过不安全验证：insecureSkipVerify 开关用于控制是否验证客户端证书和服务器主机名。启用后，将不验证证书及证书中主机名与服务器主机名的一致性。</p>

参数	说明
发件邮箱	通知服务器中的发件邮箱账号，用于发送通知邮件。
启用认证	如果需要认证，请配置邮件服务器的用户名和授权码。

4. 点击 确定。

## Webhook 类型服务器

支持集成企业微信群机器人、钉钉群机器人、飞书群机器人，或向您指定的 Webhook 服务器发送 HTTP 请求。

### 企业微信群机器人

1. 在左侧导航栏点击 集群管理 > 集群。
2. 点击 `global` 集群旁的操作按钮 > **CLI 工具**。
3. 在 `global` 集群的主节点执行以下命令：

```
kubectl patch secret -n cpaas-system platform-wechat-server -p '{"data":{"enable":"dHJ1ZQo="}}'
```

提示：`dHJ1ZQo=` 是 true 的 base64 编码；若要禁用，将 `dHJ1ZQo=` 替换为 `ZmFsc2UK`，即 false 的 base64 编码。

### 钉钉群机器人

1. 在左侧导航栏点击 集群管理 > 集群。
2. 点击 `global` 集群旁的操作按钮 > **CLI 工具**。
3. 在 `global` 集群的主节点执行以下命令：

```
kubectl patch secret -n cpaas-system platform-dingtalk-server -p '{"data":{"enable":"dHJ1ZQo="}}'
```

提示：`dHJ1ZQo=` 是 true 的 base64 编码；若要禁用，将 `dHJ1ZQo=` 替换为 `ZmFsc2UK`，即 false 的 base64 编码。

## 飞书群机器人

1. 在左侧导航栏点击 集群管理 > 集群。
2. 点击 `global` 集群旁的操作按钮 > **CLI** 工具。
3. 在 `global` 集群的主节点执行以下命令：

```
kubectl patch secret -n cpaas-system platform-feishu-server -p '{"data":{"enable":"dHJ1ZQo="}}'
```

提示：`dHJ1ZQo=` 是 true 的 base64 编码；若要禁用，将 `dHJ1ZQo=` 替换为 `ZmFsc2UK`，即 false 的 base64 编码。

## Webhook 服务器

1. 在左侧导航栏点击 集群管理 > 集群。
2. 点击 `global` 集群旁的操作按钮 > **CLI** 工具。
3. 在 `global` 集群的主节点执行以下命令：

```
kubectl patch secret -n cpaas-system platform-webhook-server -p '{"data":{"enable":"dHJ1ZQo="}}'
```

提示：`dHJ1ZQo=` 是 true 的 base64 编码；若要禁用，将 `dHJ1ZQo=` 替换为 `ZmFsc2UK`，即 false 的 base64 编码。

## 通知联系人组

通知联系人组是一组具有相似逻辑特征的通知接收人。例如，您可以将运维团队设置为通知联系人组，便于在配置通知策略时选择和管理。

#### INFO

1. 平台支持多种通知服务器，通知类型对应的配置选项会根据通知服务器配置进行展示。
2. 如果需要使用 Webhook 类型服务器作为通知接收人，必须在通知联系人组中配置相关 URL。

1. 在左侧导航栏点击 运维中心 > 通知。
2. 切换到 通知联系人组 标签页。
3. 点击 创建通知联系人组，并根据以下说明配置相关参数。

参数	说明
邮箱	为整个通知联系人组添加一个邮箱，平台会向该邮箱及组内所有联系人的邮箱发送通知。
<b>Webhook URL</b> /微信企业号群机器人/钉钉群机器人/飞书群机器人	请根据已配置的通知服务器填写对应的通知方式 URL，配置后该组内联系人将通过此方式接收通知。
联系人配置	点击 添加联系人，将已有平台用户添加至联系人组。请确保所选联系人的联系方式（电话、邮箱、接口回调）准确无误，避免消息通知遗漏。

4. 点击 添加。

## 通知模板

通知模板是由自定义内容、内容变量和内容格式参数组成的标准化结构，用于规范通知策略的告警通知消息的内容和格式。



平台管理员或运维人员可以设置通知模板，根据不同的告警通知方式自定义通知消息的内容和格式，帮助用户快速获取关键告警信息，提高运维效率。

### INFO

平台支持多种通知服务器，通知类型对应的通知模板会根据通知服务器配置进行展示。若未配置通知服务器，默认不显示对应的通知模板。

## 创建通知模板

1. 在左侧导航栏点击 运维中心 > 通知。
2. 切换到 通知模板 标签页。
3. 点击 创建通知模板。
4. 在 基本信息 部分，配置以下参数。

参数	说明
消息类型	根据通知目的选择消息类型。 告警消息：发送由告警规则触发的告警消息，配合平台告警功能使用； 组件异常消息：发送由某些组件异常触发的通知信息。

5. 在 模板配置 部分，参考不同模板类型配置变量和内容格式参数。

### INFO

1. 模板内容只能由变量、变量显示名以及平台支持的特殊格式标记语言组成。变量和其他元素可自由组合，只要符合语法规则。
2. 模板中只能使用平台支持的变量。您可以修改变量的显示名和内容格式，但不能修改变量本身。  
参考 [参考变量](#) 和 [邮件中的特殊格式标记语言](#)。
3. 平台基于实际运维场景提供了多种通知类型的默认通知模板内容，满足大多数通知消息设置需求。如无特殊需求，可直接使用默认模板内容。

## 6. 点击 创建。

## 参考变量

变量是通知消息（NotificationMessage）中标签或注解的键，格式为 `{{.labelKey}}`。为方便用户快速获取关键信息，可为变量指定自定义显示名；例如：`告警级别：{{.externalLabels.severity}}`。

当通知规则基于通知模板向用户发送通知消息时，模板中的变量会引用通知消息中对应标签的值（实际监控数据），最终以标准化内容格式发送监控数据给用户。

平台默认提供以下基础变量：

显示名	变量	说明
告警状态	<code>{{.externalLabels.status}}</code>	例如：告警中。
告警级别	<code>{{.externalLabels.severity}}</code>	例如：严重。
告警集群	<code>{{.labels.alert_cluster}}</code>	例如：发生告警的集群 1。
告警对象	<code>{{.externalLabels.object}}</code>	告警发生的资源类型及名称，例如节点 192.168.16.53。

显示名	变量	说明
规则名称	<code>{{ .labels.alert_resource }}</code>	告警规则名称，例如 cpaas-node-rules。
告警描述	<code>{{ .externalLabels.summary }}</code>	告警规则描述。
触发值	<code>{{ .externalLabels.currentValue }}</code>	触发告警的监控值。
告警时间	<code>{{ dateFormatWithZone .startsAt "2006-01-02 15:04:05" "Asia/Chongqing" }}</code>	告警开始时间。
恢复时间	<code>{{ dateFormatWithZone .endsAt "2006-01-02 15:04:05" "Asia/Chongqing" }}</code>	告警结束时间。
指标名称	<code>{{ .labels.alert_indicator }}</code>	监控指标名称。

## 邮件中的特殊格式标记语言

邮件通知中常用的 HTML 格式标签及说明如下表：

内容元素	标签	说明
文本	-	支持输入中英文文本内容。

内容元素	标签	说明
字体	<code>&lt;font color="#FF0000"&gt;设置字体颜色&lt;/font&gt;</code> <code>&lt;b&gt;加粗字体&lt;/b&gt;</code>	设置字体格式。
标题	<code>&lt;h1&gt;一级标题&lt;/h1&gt;</code> ，支持至 h6（标题 6）。	设置标题级别。
段落	<code>&lt;p&gt;段落&lt;/p&gt;</code>	插入普通段落文本。
引用	<code>&lt;q&gt;引用&lt;/q&gt;</code>	插入简短引用内容。
超链接	<code>&lt;a href="//www.example.com"&gt;超链接&lt;/a&gt;</code>	插入超链接。

## 通知规则

通知规则是一组定义如何向特定联系人发送通知消息的规则。对于需要通知外部服务的场景，如告警、巡检和登录认证，必须使用通知策略。

### INFO

平台支持多种通知服务器，通知类型对应的通知模式会根据通知服务器配置进行展示。若未配置通知服务器，默认不显示对应的通知模式。

## 前提条件

使用 企业通信工具服务器 通知联系人前，用户需先在 个人信息 中修改联系方式，填写 微信企业号 ID。

## 操作流程

- 在左侧导航栏点击 运维中心 > 通知。
- 点击 创建通知规则，并根据以下说明配置相关参数。

参数	说明
通知联系人组	通知联系人组是一组逻辑上的通知接收人，平台将使用指定的通知方式通知该组。
通知接收人	选择添加一个或多个通知接收人，平台将根据接收人的 个人信息 中的联系方式发送通知。
通知方式	支持多种方式，包括 微信企业号、钉钉、飞书、企业微信群机器人、钉钉群机器人、飞书群机器人、 <b>WebHook URL</b> ，支持多选。 注意：部分参数需配置通知服务器后才会显示。
通知模板	选择通知模板以展示通知信息。

3. 点击 创建。

## 为项目设置通知规则

平台的通知策略、通知模板和通知联系人组均为租户隔离。作为项目管理员，您无法查看或使用其他项目或平台管理员配置的通知策略、通知模板或通知联系人组。因此，您需要参考本文档为您的项目配置合适的通知策略。

## 前提条件

1. 您已联系平台管理员完成通知服务器的搭建。
2. 若需通过企业通信工具通知，还需确保被通知联系人已在 个人信息 中正确配置通信工具 ID。

## 操作流程

1. 在 项目管理 视图中，点击 项目名称。
2. 在左侧导航栏点击 通知。

3. 切换到 通知联系人组 标签页，参考 [通知联系人组](#) 创建通知联系人组。

**TIP**

如果不需要通过通知联系人组管理通知联系人，或不需要通知 webhook 类型通知服务器，可跳过此步骤。

4. 切换到 通知模板 标签页，参考 [通知模板](#) 创建通知模板。

5. 切换到 通知规则 标签页，参考 [通知规则](#) 创建通知规则。

# 监控面板管理

## 目录

### 功能概述

主要功能

优势

使用场景

前置条件

监控面板与监控组件的关系

### 管理监控面板

创建监控面板

导入面板

添加变量

添加 panels

添加分组

切换面板

其他操作

### 管理 panels

panel 说明

panel 配置说明

通用参数

panel 特殊参数

### 通过 CLI 创建监控面板

### 常用函数和变量

常用函数

常用变量

变量使用示例一

变量使用示例二

使用内置指标注意事项

---

## 功能概述

平台提供强大的面板管理功能，旨在替代传统的 Grafana 工具，为用户带来更全面灵活的监控体验。该功能汇聚平台内的各类监控数据，呈现统一的监控视图，显著提升您的配置效率。

## 主要功能

- 支持为业务视图和平台视图配置自定义监控面板。
- 支持在业务视图中查看平台视图中公开共享的面板，数据基于业务所属命名空间进行隔离。
- 支持管理面板内的 panels，允许用户添加、删除、修改 panels，支持 panels 放大/缩小及拖拽移动。
- 允许在面板中设置自定义变量，用于过滤查询数据。
- 支持在面板中配置分组，用于管理 panels。分组可基于自定义变量重复展示。
- 支持的 panel 类型包括：趋势图、阶梯折线图、柱状图、水平柱状图、柱状仪表图、仪表图、表格、统计图、XY 图、饼图、文本。
- 支持一键导入 Grafana 面板。

## 优势

- 支持用户自定义监控场景，不受预设模板限制，真正实现个性化监控体验。
  - 提供丰富的可视化选项，包括折线图、柱状图、饼图，以及灵活的布局和样式配置。
  - 与平台角色权限无缝集成，业务视图可定义自身监控面板，同时确保数据隔离。
  - 深度集成容器平台各项功能，支持即时访问容器、网络、存储等监控数据，为用户提供全面的性能观察和故障诊断。
  - 完全兼容 Grafana 面板 JSON，方便从 Grafana 迁移并继续使用。
-



## 使用场景

- **IT 运维管理**：作为 IT 运维团队成员，您可以使用监控面板统一展示和管理容器平台的各类性能指标，如 CPU、内存、网络流量等。通过自定义监控报表和告警规则，及时发现并定位系统问题，提高运维效率。
- **应用性能分析**：对于应用开发和测试人员，监控面板提供多样化的可视化选项，直观展示应用运行状态和资源消耗。您可以定制针对不同应用场景的专用监控视图，深入分析应用性能瓶颈，提供优化依据。
- **多集群管理**：对于管理多个容器集群的用户，监控面板可汇聚不同集群的监控数据，帮助您一目了然掌握系统整体运行状况。
- **故障诊断**：当系统出现问题时，监控面板为您提供全面的性能数据和分析工具，快速定位问题根因。您可以根据告警信息迅速查看相关监控指标的波动，进行深入故障分析。

## 前置条件

目前监控面板仅支持查看平台内监控组件采集的监控数据。因此，在配置监控面板前，请做好以下准备：

- 确保您要配置监控面板的集群已安装监控组件，具体为 **ACP Monitor with Prometheus** 或 **ACP Monitor with VictoriaMetrics** 插件。
- 确保您希望在面板中展示的数据已被监控组件采集。

## 监控面板与监控组件的关系

- 监控面板资源存储于 Kubernetes 集群中，您可通过顶部的 **Cluster** 标签页切换不同集群视图。
- 监控面板依赖集群内的监控组件查询数据源，因此使用前请确保当前集群已成功安装监控组件且运行正常。
- 监控面板默认请求对应集群的监控数据。如果您在集群中以代理模式安装了 **VictoriaMetrics** 插件，平台会为您请求存储集群查询该集群对应的数据，无需额外配置。

## 管理监控面板

面板是由一个或多个 panel 组成的集合，按一行或多行组织排列，提供清晰的相关信息视图。这些 panel 可从数据源查询原始数据，并转换为平台支持的一系列可视化效果。

## 创建监控面板

1. 点击 创建面板，参考以下说明配置相关参数。

参数	说明
文件夹	面板所在的文件夹，您可以输入或选择已有文件夹。
标签	监控面板的标签，切换时可通过顶部标签快速筛选已有面板。
设为主面板	启用后，创建成功时将当前面板设为主面板；再次进入监控面板功能时，默认展示主面板数据。
变量	创建面板时添加变量，供新增的 panels 作为指标参数引用，也可作为面板首页的过滤器使用。

2. 添加完成后，点击 创建 完成面板创建。接下来，您需要 添加变量、添加 **panels** 和 添加分组，完成整体布局设计。

## 导入面板

平台支持直接导入 Grafana JSON，将其转换为监控面板进行展示。

- 目前仅支持 Grafana JSON V8+ 版本，低版本将禁止导入。
- 导入的面板中若存在平台不支持的 panel 类型，可能显示为 不支持的 **panel** 类型，但您可修改面板设置实现正常展示。
- 导入后，您可像使用平台创建的 panels 一样进行管理操作，无差异。

## 添加变量

1. 在变量表单区域，点击 添加。

参数	说明
类型	目前仅支持 Query 类型变量，允许基于时序特征维度过滤数据。可指定查询表达式动态计算生成查询结果。
显示过滤器	面板首页下拉过滤器的默认显示值，支持显示名称和值、仅值或不显示（隐藏过滤框）。
查询设置	使用 Query 类型变量允许基于时序特征维度过滤数据。定义查询设置时，除使用 PromQL 查询时序外，平台还提供部分常用变量和函数。参考 <a href="#">常用函数和变量</a> 。
正则表达式	通过正则表达式过滤变量查询返回内容中的期望值，使变量的每个选项名称更符合预期。可在 <a href="#">变量值预览</a> 中预览过滤结果是否符合预期。
选择设置	<ul style="list-style-type: none"> <li>- 多选：面板首页顶部过滤器选择时，允许同时选择多个选项。您需在 panels 的查询表达式中引用该变量，查看对应变量值的数据。</li> <li>- 全部：勾选后，过滤选项中将启用包含 全部 的选项，可选择所有变量数据。</li> </ul>

2. 点击 **确定** 添加一个或多个变量。

## 添加 panels

提示：您可点击 panel 右下角自定义大小；点击 panel 任意位置可调整 panels 顺序。

1. 点击 **添加 panel**，参考以下说明配置相关参数。

- **panel 预览**：区域动态展示所添加指标对应的数据情况。
- **添加指标**：配置 panel 标题和监控指标。
- **添加方式**：支持使用内置指标或使用原生自定义指标，两者取并集同时生效。
  - **内置指标**：选择平台内置的常用指标和图例参数，展示当前 panel 下的数据情况。
  - **注意**：添加到 panel 的所有指标必须单位统一，不能在一个 panel 中添加多单位指标。

- 原生：自定义指标单位、指标表达式和图例参数。指标表达式遵循 PromQL 语法，详情请参考 [PromQL 官方文档](#)。
  - 图例参数：控制 panel 中曲线对应的名称，可使用文本或模板：
    - 规则：输入值必须为 `{{.xxxx}}` 格式，如 `{{.hostname}}` 将替换为表达式返回的 hostname 标签对应值。
    - 提示：若输入格式错误，panel 中曲线对应名称将按原格式显示。
  - 即时开关：开启时通过 Prometheus Query 接口查询即时值并排序，适用于统计图和仪表盘；关闭时使用 `query_range` 方式计算，查询特定时间段内的一系列数据。
  - **panel 设置**：支持选择不同 panel 类型展示指标数据。详情请参考 [管理 panels](#)。
2. 点击 保存 完成添加 panels。
  3. 您可在面板内添加一个或多个 panels。
  4. 添加完成后，您可通过以下操作确保 panels 显示和大小符合预期：
    - 点击 panel 右下角自定义大小。
    - 点击 panel 任意位置调整 panels 顺序。
  5. 调整完成后，点击面板页面的 保存 按钮保存修改。

## 添加分组

分组是面板内的逻辑划分，可将 panels 归类管理。

1. 点击 添加 **panel** 下拉菜单 > 添加分组，参考以下说明配置相关参数。
  - 分组：分组名称。
  - 重复：支持禁用重复或选择当前 panels 的变量。
    - 禁用重复：不选择变量，使用默认创建的分组。
    - 参数变量：选择当前 panels 创建的变量，监控面板将为变量的每个对应值生成一行相同的子分组。子分组不支持修改、删除或移动 panels。
2. 添加分组后，您可对分组执行以下操作以管理面板显示：

- 分组可折叠或展开，隐藏面板部分内容。折叠的分组内 panels 不会发送查询。
- 将 panel 移入分组，使该 panel 由该分组管理。分组管理其与下一个分组之间的所有 panels。
- 分组折叠时，也可整体移动该分组管理的所有 panels。
- 分组折叠与展开也属于面板调整，若希望下次打开面板时保持该状态，请点击 保存。

## 切换面板

将已创建的自定义监控面板设为主面板，重新进入监控面板功能时默认展示主面板数据。

1. 在左侧导航栏点击 运营中心 > 监控 > 监控面板。
2. 默认进入主监控面板，点击 切换面板。
3. 可通过标签筛选或名称搜索查找面板，通过 主面板 开关切换主面板。

## 其他操作

您可点击面板页面右侧操作按钮，根据需要对面板执行操作。

操作	说明
YAML	打开存储于 Kubernetes 集群中的面板实际 CR 资源代码，您可通过编辑 YAML 中参数修改面板所有内容。
导出表达式	可导出当前面板使用的指标及对应查询表达式，格式为 CSV。
复制	复制当前面板，您可根据需要编辑 panels 并保存为新面板。
设置	修改当前面板的基本信息，如更改标签和添加更多变量。
删除	删除当前监控面板。

## 管理 panels

平台提供多种可视化方式，支持不同使用场景。本章主要介绍这些 panel 类型、配置选项及使用方法。

## panel 说明

序号	panel 名称	说明	建议使用场景
1	趋势图	通过一条或多条折线展示数据随时间的变化趋势。	展示随时间变化的趋势，如 CPU 利用率、内存使用量等变化。
2	阶梯折线图	基于折线图，通过水平和垂直线段连接数据点，形成阶梯状结构。	适合展示离散事件的时间戳，如告警数量。
3	柱状图	使用垂直矩形柱表示数据大小，柱高代表数值。	柱状图直观对比数值差异，有助发现规律和异常，适合关注数值变化的场景，如 Pod 数量、节点数量等。
4	水平柱状图	类似柱状图，但使用水平矩形柱表示数据。	当数据维度较多时，水平柱状图能更好利用空间布局，提高可读性。
5	仪表图	使用半圆或环形表示指标当前值及其占比。	直观反映关键监控指标当前状态，如系统 CPU 利用率、内存使用率。建议结合告警阈值颜色变化指示异常。
6	柱状仪表图	使用垂直矩形柱展示指标当前值及其占比。	直观反映关键指标当前状态，如目标完成进度、系统负载。存在多类别同一指标时，柱状仪表图更推荐，如可用磁盘空间或利用率。
7	饼图	使用扇形展示部分与整体的比例关系。	适合展示整体数据在不同维度的组成，如一段时间内 4XX、3XX、2XX 响应码比例。
8	表格	以行列形式组织数据，便于查看和对比具体数值。	适合展示结构化多维数据，如节点详细信息、Pod 详细信息等。
9	统计图	展示单个关键指标的当前值，通常需要文字说明。	适合展示重要监控指标的实时数值，如 Pod 数量、节点数量、当前告警数等。

序号	panel 名称	说明	建议使用场景
10	散点图	使用笛卡尔坐标系绘制一系列数据点，反映两个变量间的相关性。	适合分析两个指标间关系，通过数据点分布发现线性相关、聚类等规律，帮助挖掘指标间关联。
11	文本卡片	以卡片形式展示关键信息文本，通常包含标题和简要说明。	适合展示文本信息，如 panel 描述、故障排查说明等。

## panel 配置说明

### 通用参数

参数	说明
基本信息	根据所选指标数据选择合适的 panel 类型，添加标题和描述；可添加一个或多个链接，点击标题旁对应链接名称可快速访问。
标准设置	原生指标数据使用的单位。此外，仪表图和柱状仪表图还支持配置 总值 字段，图表中将显示为 当前值/总值 的百分比。
提示信息	鼠标悬停 panel 时实时数据的显示开关，支持选择排序。
阈值参数	配置 panel 的阈值开关，启用后面板中以选定颜色显示阈值，支持阈值大小调整。
数值	设置数值的计算方式，如最新值或最小值。该配置仅适用于统计图和仪表图。
数值映射	重新定义指定数值或数值区间，如定义 100 为满载。该配置仅适用于统计图、表格和仪表图。

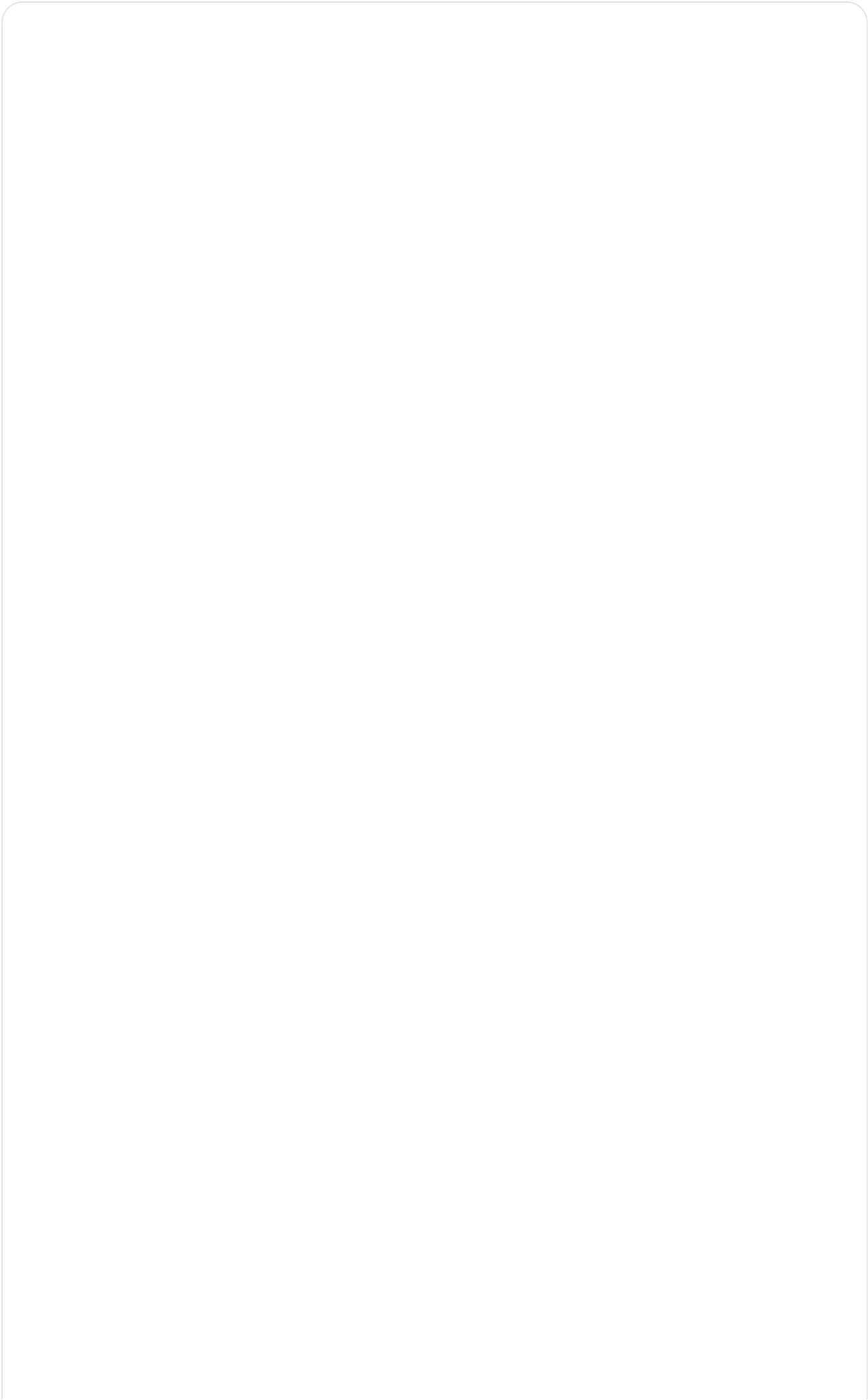
### panel 特殊参数

panel 类型	参数	说明
趋势图	图形样式	可选择折线图或面积图作为展示样式；折线图更侧重反映指标趋势变化，面积图更关注总量和部分占比变化。根据实际需求选择。
仪表图	仪表图设置	显示方向：需要在单图中查看多个指标时，可设置指标是横向还是纵向排列。 单位重定义：可为每个指标设置独立单位，未设置时平台显示 标准设置中的单位。
饼图	饼图设置	最大切片数：可设置减少饼图切片数，降低低比例但数量多类别的干扰。超出部分合并显示为 其他。 标签显示字段：可设置饼图标签中显示的字段。
饼图	图形样式	可选择饼图或甜甜圈图作为展示样式。
表格	表格设置	隐藏列：可减少表格列数，聚焦部分主要列信息。 列对齐：可修改列内数据对齐方式。 显示名称和单位：可通过该参数修改列名和单位。
文本卡片	图形样式	样式：可选择在富文本编辑框或 HTML 中编辑文本卡片内容。

## 通过 CLI 创建监控面板

- 新建名为 `example-dashboard.yaml` 的 YAML 配置文件。
- 在 YAML 文件中添加 MonitorDashboard 资源并提交。以下示例创建名为 demo-v2-dashboard1 的监控面板：





```

kind: MonitorDashboard
apiVersion: ait.alauda.io/v1alpha2
metadata:
  annotations:
    cpaas.io/dashboard.version: '3'
    cpaas.io/description: '{"zh":"描述信息","en":""}' # 描述字段
    cpaas.io/operator: admin
  labels:
    cpaas.io/dashboard.folder: demo-v2-folder1 # 文件夹
    cpaas.io/dashboard.is.home.dashboard: 'False' # 是否为主面板？
name: demo-v2-dashboard1 # 名称
namespace: cpaas-system # 命名空间（所有管理视图创建均在此 ns）
spec:
  body: # 所有信息字段
    titleZh: 更新显示名称 # 中文显示名称内置字段（该字段在中文语言下创建）
    title: english_display_name # 英文显示名称内置字段（该字段在英文语言下创建）
    templating: # 自定义变量
      list:
        - hide: 0 # 0 表示不隐藏；1 表示仅隐藏标签；2 表示标签和值均隐藏
          label: 集群 # 内置变量显示名称（标签根据语言设置为对应名称，如英文为 cluster）
          name: cluster # 内置变量名称（唯一）
          options: # 定义下拉选项；若查询获取数据则使用请求数据，否则使用 options。可设置默认值（一般仅用于设置默认值）
            - selected: false # 是否默认选中
              text: global
              value: global
              type: custom # 自定义变量类型；目前仅支持内置（custom）和查询（query）（导入 Grafana 会支持常量自定义区间（导入后会变为自定义变量，不支持自动））
            - allValue: '' # 选择全部，传递格式为 xxx|xxx|xxx 的 options；可设置 allValue 进行转换（Grafana 获取当前变量所有数据为 xxx|xxx|xxx，调整后保持一致）
              current: null # 变量当前值；未设置时默认列表第一个
              definition: query_result(kube_namespace_labels) # 查询表达式
              hide: 0 # 0 表示不隐藏；1 表示仅隐藏标签；2 表示标签和值均隐藏
              includeAll: true # 是否选择全部
              label: ns # 内置变量显示名称
              multi: true # 是否允许多选
              name: ns # 变量名称（唯一）
              options: []
              query: ''
              regex: /.namespace="(.*?)"/ # 变量值提取正则表达式

```

```

    sort: 2 # 排序: 1 - 升序字母; 2 - 降序字母 (暂仅支持这两种); 3 - 升
序数字; 4 - 降序数字
    type: query # 自定义变量类型
time: # 面板时间
    from: now-30m # 开始时间
    to: now # 结束时间
repeat: '' # 行重复配置; 选择自定义变量
collapsed: 'false' # 行折叠或展开配置
description: '123' # 描述 (标题后悬浮提示)
targets: # 数据源
  - indicator: cluster.node.ready # 指标
    expr: sum (cpaas_pod_number{cluster=\"\"}>0) # PromQL 表达式
    instant: false # 查询模式 true 表示查询某一时刻数据
    legendFormat: '' # 图例
    range: true # 查询数据时默认查询区间
    refId: 指标1 # 数据源显示名称唯一标识
gridPos: # 面板位置信息布局
  h: 8 # 高度
  w: 12 # 宽度 (宽度对应 24 格)
  x: 0 # 横向位置
  y: 0 # 纵向位置
panels: # panel 数据
  title: 图表标题tab # panel 名称
  type: table # panel 类型; 目前支持 timeseries、barchart、stat、gaug
e、table、bargauge、row、text、pie (阶梯图、散点图、柱状图通过 drawStyle 属性
配置)
  id: a2239830-492f-4d27-98f3-cb7ecb77c56f # 唯一标识
  links: # 链接
    - targetBlank: true # 新标签页打开
      title: '1' # 名称
      url: '1' # URL 地址
  transformations: # 数据转换
    - id: 'organize' # 类型 organize; 用于排序、调整顺序、显示字段、是否显
示
      options:
        excludeByName: # 隐藏字段
          cluster_cpu_utilization: true
        indexByName: # 排序
          cluster_cpu_utilization: 0,
          Time: 1
        renameByName: # 重命名
          Time: ''
          cluster_cpu_utilization: '222'
    - id: 'merge' # 合并数据

```

```

options:
fieldConfig: # 定义 panel 属性和外观
defaults: # 默认配置
custom: # 自定义图形属性
  align: 'left' # 表格对齐方式: left、center、right
  cellOptions: # 表格阈值配置
    type: color-text # 仅支持文本阈值颜色设置
  spanNulls: false # true 连接 null 值; false 不连接; 数字 == 0
按 0 连接 null 值
  drawStyle: line # panel 类型: line, bars 为柱状图, points 为点
图
  fillOpacity: 20 # drawStyle 为 area 时存在 (目前不支持配置, are
a 默认为 20)
  thresholdsStyle: # 配置阈值显示方式 (目前仅支持线)
    mode: line # 阈值显示格式 (area 目前不支持)
    lineInterpolation: 'stepBefore' # 阶梯图配置; 默认仅支持 stepBe
fore (后续支持 stepAfter)
  decimals: 3 # 小数点位数
  min: 0 # 最小值 (目前不支持页面配置, 仅支持已适配导入)
  max: 1 # 最大值 (页面配置仅 stat gauge barGauge pie 适用)
  unit: '%' # 单位
  mappings: # 数值映射配置 (目前仅支持 value 和 range 类型; 数据上支持
特殊类型)
    - options: # 数值映射规则
      '1': # 对应数值
        index: 0
        text: 'Running' # 数值为 1 时显示 Running
      type: value # 数值映射类型
    - options: # 区间映射规则
      from: 2 # 区间起始值
      to: 3 # 区间结束值
      result: # 映射结果
        index: 1
        text: 'Error' # 2~3 区间显示 Error
      type: range # 区间映射类型
    - type: special # 特殊场景映射类型
      options:
        match: null # nan null null+nan empty true false
        result:
          text: xxx
          index: 2
  thresholds: # 阈值配置
    mode: absolute # 阈值配置模式, 绝对值模式 (目前仅支持绝对值和百分比
模式; 百分比模式暂不支持)

```

```

    steps: # 阈值步骤
      - color: '#a7772f' # 阈值颜色
        value: '2' # 阈值数值
      - color: '#007AF5' # 默认值无值即为 Base
  overrides: # 覆盖配置
    - matcher:
        id: byName # 按字段名匹配
        options: node # 对应名称
      properties: # 覆盖配置; id 目前仅支持 displayName unit
        - id: displayName # 显示名称覆盖
          value: '1' # 覆盖显示名称
        - id: unit # 单位覆盖
          value: GB/s # 单位值
        - id: noValue # 无值显示
          value: 无值显示
  options:
    orientation: horizontal # 控制 panels 布局方向; 适用于 gauge 和 bar
Gauge (stat 后续支持)
  legend: # 图例配置
    calcs: # 计算方式 (仅图例位置为右侧时显示)
      - latest # 目前仅支持最新值
    placement: right # 图例位置 (右侧或底部, 默认底部)
    placementRightTop: '' # 右上角配置
    showLegend: true # 是否显示图例
  tooltip: # 提示信息
    mode: multi # 模式双选 (仅支持多模式) 鼠标悬停时显示所有数据
    sort: asc # 排序: asc 或 desc
  reduceOptions: # 数值计算方式 (用于聚合数据)
    calcs: # 计算方式 (最新、最小、最大、平均、求和)
      - latest
    limit: 3 # 饼图限制切片数
  textMode: 'value' # 统计图配置; 定义指标值显示样式; 选项有 auto、value、value_and_name、name、none (页面配置暂不支持, 导入支持)
  colorMode: 'value' # 统计图配置; 定义指标值显示颜色模式; 选项有 none、value、background (默认 value; 配置不支持, 导入适配)
  displayLabels: ['name', 'value', 'percent'] # 饼图标签显示字段
  pieType: 'pie' # 饼图类型; 选项 pie 和 donut
  mode: 'html' # 文本图类型模式; 选项 html 和 richText
  content: '<div>xxx</div>' # 文本图类型内容
  footer:
    enablePagination: true # 表格启用分页

```

## 常用函数和变量

## 常用函数

定义查询设置时，除使用 PromQL 设置查询外，平台还提供以下常用函数供您自定义查询设置参考。

函数名	作用
<b>label_names()</b>	返回 Prometheus 中所有标签，如 label_names()。
<b>label_values(label)</b>	返回 Prometheus 中所有监控指标中该标签名可选的所有值，如 label_values(job)。
<b>label_values(metric, label)</b>	返回 Prometheus 中指定指标中该标签名可选的所有值，如 label_values(up, job)。
<b>metrics(metric)</b>	返回满足指标字段定义的正则表达式的所有指标名，如 metrics(cpaas_active)。
<b>query_result(query)</b>	返回指定 Prometheus 查询的查询结果，如 query_result(up)。

## 常用变量

定义查询设置时，您可将常用函数组合成变量，快速定义自定义变量。以下为部分常用变量定义，供您参考：

变量名	查询函数
<b>cluster</b>	<code>label_values(cpaas_cluster_info, cluster)</code>
<b>node</b>	<code>label_values(node_load1, instance)</code>
<b>namespace</b>	<code>query_result(kube_namespace_labels)</code>
<b>deployment</b>	<code>label_values(kube_deployment_spec_replicas{namespace="\$namespace"}, deployment)</code>
<b>daemonset</b>	<code>label_values(kube_daemonset_status_number_ready{namespace="\$namespace"}, daemonset)</code>

变量名	查询函数
<b>statefulset</b>	<code>label_values(kube_statefulset_replicas{namespace="\$namespace"}, statefulset)</code>
<b>pod</b>	<code>label_values(kube_pod_info{namespace=~"\$namespace"}, pod)</code>
<b>vmcluster</b>	<code>label_values(up, vmcluster)</code>
<b>daemonset</b>	<code>label_values(kube_daemonset_status_number_ready{namespace="\$namespace"} daemonset)</code>

## 变量使用示例一

使用 **query\_result(query)** 函数查询值：`node_load5`，并提取 IP。

1. 在 查询设置 中填写 `query_result(node_load5)`。
2. 在 变量值预览 区域，预览示例为 `node_load5{container="node-exporter",endpoint="metrics",host_ip="192.168.178.182",instance="192.168.178.182:9100"}`。
3. 在 正则表达式 中填写 `/. *instance="(.*?) :. */` 过滤值。
4. 在 变量值预览 区域，预览示例为 `192.168.176.163`。

## 变量使用示例二

1. 添加第一个变量：namespace，使用 **query\_result(query)** 函数查询值：

`kube_namespace_labels`，并提取 namespace。

- 查询设置：`query_result(kube_namespace_labels)`。
- 变量值预览：`kube_namespace_labels{container="exporter-kube-state", endpoint="kube-state-metrics", instance="12.3.188.121:8080", job="kube-state", label_cpaas_io_project="cpaas-system", namespace="cert-manager", pod="kube-prometheus-exporter-kube-state-55bb6bc67f-lpgtx", project="cpaas-system", service="kube-prometheus-exporter-kube-state"}`。

- 正则表达式：`/.+namespace=\"(.*)\".*/`。
- 在 变量值预览 区域，预览示例包含多个 namespace，如 `argocd`、`cpaas-system` 等。

2. 添加第二个变量：`deployment`，引用之前创建的变量：

- 查询设置：`kube_deployment_spec_replicas{namespace=~"$namespace"}`。
- 正则表达式：`/.+deployment=\"(.*)\".*/`。

3. 在当前面板添加 panel，引用之前添加的变量，例如：

- 指标名称：计算组件下 **Pod** 内存使用。
- 键值对：`kind`：`Deployment`，`name`：`$deployment`，`namespace`：`$namespace`。

4. 添加 panels 并保存后，您可在面板首页查看对应 panel 信息。

## 使用内置指标注意事项

### WARNING

以下指标使用自定义变量 `namespace`、`name` 和 `kind`，不支持 多选 和选择 全部。

- `namespace` 仅支持选择具体命名空间；
- `name` 仅支持三种计算组件类型：`deployment`、`daemonset`、`statefulset`；
- `kind` 仅支持指定以下类型之一：`Deployment`、`DaemonSet`、`StatefulSet`。

- `workload.cpu.utilization`
- `workload.memory.utilization`
- `workload.network.receive.bytes.rate`
- `workload.network.transmit.bytes.rate`
- `workload.gpu.utilization`



- `workload.gpu.memory.utilization`
- `workload.vgpu.utilization`
- `workload.vgpu.memory.utilization`

# 探针管理

---

## 目录

### 功能概述

#### 黑盒监控

##### 前提条件

##### 操作流程

#### 黑盒告警

##### 前提条件

##### 操作流程

#### 自定义 BlackboxExporter 监控模块

##### 操作流程

#### 通过 CLI 创建黑盒监控项和告警

##### 前提条件

##### 操作流程

#### 参考信息

---

## 功能概述

平台的探针功能基于 Blackbox Exporter 实现，允许用户通过 ICMP、TCP 或 HTTP 方式对网络进行探测，快速定位平台上发生的故障。

与依赖平台已有各种监控指标的白盒监控系统不同，黑盒监控关注的是结果。当白盒监控无法覆盖影响服务可用性的所有因素时，黑盒监控能够快速发现故障并基于故障发出告警。例如，

当某个 API 接口异常时，黑盒监控能够及时将此类问题暴露给用户。

### WARNING

探针功能不支持在内核版本 3.10 及以下的节点上使用 ICMP 探测 IPv6 地址。如需使用此场景，请将节点内核版本升级至 3.11 及以上。

## 黑盒监控

创建黑盒监控项时，可选择 ICMP、TCP 或 HTTP 探测方式，周期性地探测指定的目标地址。

### 前提条件

监控组件需已安装在集群中，且监控组件运行正常。

### 操作流程

1. 在左侧导航栏点击 运维中心 > 监控 > 黑盒监控。

提示：黑盒监控为集群级功能，点击顶部导航栏可切换集群。

2. 点击 创建黑盒监控项。

3. 按照以下说明配置相关参数。

参数	说明
探测方式	<p><b>ICMP</b>：通过 ping 输入的目标地址中的域名或 IP 地址，探测服务器是否可用。</p> <p><b>TCP</b>：通过监听目标地址中指定的 <code>&lt;domain:port&gt;</code> 或 <code>&lt;IP:port&gt;</code>，探测主机的业务端口。</p> <p><b>HTTP</b>：探测输入的目标地址中的 URL，检查网站连通性。</p> <p>提示：HTTP 探测方式默认仅支持 GET 请求，若需 POST 请求，请参考 <a href="#">自定义 BlackboxExporter 监控模块</a>。</p>

参数	说明
探测间隔	探测的时间间隔。
目标地址	<p>探测的目标地址，最长支持 128 个字符。</p> <p>目标地址的输入格式根据探测方式不同而不同：</p> <p><b>ICMP</b>：域名或 IP 地址，例如 <code>10.165.94.31</code>。</p> <p><b>TCP</b>：<code>&lt;domain:port&gt;</code> 或 <code>&lt;IP:port&gt;</code>，例如 <code>172.19.155.133:8765</code>。</p> <p><b>HTTP</b>：以 http 或 https 开头的 URL，例如 <code>http://alauda.cn/</code>。</p>

#### 4. 点击 创建。

创建成功后，可在列表页实时查看最新探测结果，并基于黑盒监控项[创建告警策略](#)。当检测到故障时，系统会自动触发告警，通知相关人员进行处理。

#### WARNING

黑盒监控项创建成功后，系统需要约 5 分钟时间同步配置。同步期间不会进行探测，且无法查看探测结果。

## 黑盒告警

### 前提条件

- 监控组件需已安装在集群中，且监控组件运行正常。
- 黑盒监控项已成功创建，且系统已完成配置同步，黑盒监控页面可见探测结果。

### 操作流程

1. 在左侧导航栏点击 运维中心 > 告警 > 告警策略。

提示：告警策略为集群级功能，点击顶部导航栏可切换集群。请确保切换至刚配置黑盒监控项的集群。

2. 点击 创建告警策略。

3. 按照以下说明配置相关参数，更多参数信息请参考 [创建告警策略](#)。

- 告警类型：请选择 资源告警。
- 资源类型：请选择 集群。
- 点击 添加告警规则。
  - 告警类型：请选择 黑盒告警。
  - 黑盒监控项：请选择目标黑盒监控项。
  - 指标名称：请选择希望监控并告警的指标。平台当前支持的指标为 **Connectivity** 和 **HTTP Status Code**。
    - **Connectivity**：该指标可用于所有黑盒监控项，触发条件为 “**!= 1**” 表示黑盒监控项的目标地址不可达。
    - **HTTP Status Code**：该指标仅在所选黑盒监控项的探测方式为 **HTTP** 时可选。触发条件可输入三位正整数，例如条件设置为 “**> 299**”，表示响应码为 3XX、4XX 或 5XX 时触发告警。
  - 通知策略：请选择预先配置的通知策略。
  - 点击 添加。

4. 点击 创建。提交告警策略后，可在告警策略列表中查看该策略。

## 自定义 BlackboxExporter 监控模块

您还可以通过向 BlackboxExporter 配置文件中添加自定义监控模块，增强黑盒监控的功能。例如，向配置文件添加 **http\_post\_2xx** 模块后，当黑盒监控的探测方式设置为 **HTTP** 时，即可探测 POST 请求方法的状态。

黑盒监控的配置文件位于集群中 Prometheus 组件安装的命名空间内，默认名称为 `cpaas-monitor-prometheus-blackbox-exporter`，可根据实际名称进行修改。

### TIP

该配置文件为与命名空间相关的 ConfigMap 资源，可通过平台管理功能 [集群管理 > 资源管理](#) 快速查看和更新。

## 操作流程

1. 通过向配置文件的 **modules** 键添加自定义监控模块，更新黑盒监控配置文件。

以添加 **http\_post\_2xx** 模块为例：

```
blackbox.yaml: |
  modules:
    http_post_2xx:                # HTTP POST 探测模块
      prober: http
      timeout: 5s
      http:
        method: POST              # 探测请求方法
        headers:
          Content-Type: application/json
        body: '{}                 # 探测时携带的请求体内容
```

黑盒监控配置文件的完整 YAML 示例，请参考 [参考信息](#)。

2. 通过以下任一方式使配置生效。

- 删除 Blackbox Exporter 组件 **cpaas-monitor-prometheus-blackbox-exporter** 的 Pod，重启组件。
- 执行以下命令调用 reload API，刷新配置文件：

```
curl -X POST -v <Pod IP>:9115/-/reload
```

# 通过 CLI 创建黑盒监控项和告警

## 前提条件

- 已配置通知策略（若需告警自动通知）。
- 目标集群已安装监控组件。

## 操作流程

1. 新建 YAML 配置文件，命名为 `example-probe.yaml`。
2. 在 YAML 文件中添加 PrometheusRule 资源并提交。以下示例创建名为 `prometheus-liveness` 的新告警策略：

```

apiVersion: monitoring.coreos.com/v1
kind: Probe
metadata:
  annotations:
    cpaas.io/creator: jhshi@alauda.io # 探针项创建者
    cpaas.io/updated-at: '2021-05-25T08:08:45Z' # 探针项最后更新时间
    cpaas.io/display-name: 'Prometheus prober' # 探针项描述
  creationTimestamp: '2021-05-10T02:04:33Z' # 探针项创建时间
  labels:
    prometheus: kube-prometheus # 用于 prometheus 名称的标签值
  name: prometheus-liveness # 探针项名称
  namespace: cpaas-system # prometheus 命名空间
spec:
  jobName: prometheus-liveness # 探针项名称
  prober:
    url: cpaas-monitor-prometheus-blackbox-exporter:9115 # Blackbox 指标
    URL, 从特性中获取
  module: http_2xx # 探针项模块名称
  targets:
    staticConfig:
      static:
        - http://www.prometheus.io # 探针项目标地址
      labels:
        module: http_2xx # 探针项模块名称
        prober: http # 探测方式
  interval: 30s # 探针项探测间隔
  scrapeTimeout: 10s

```

3. 新建 YAML 配置文件，命名为 `example-alerting-rule.yaml`。

4. 在 YAML 文件中添加 PrometheusRule 资源并提交。以下示例创建名为 `policy` 的新告警策略：





```

apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global # 告警所在集群名称
    alert.cpaas.io/kind: Cluster # 资源类型
    alert.cpaas.io/name: global # 黑盒监控项所在集群名称
    alert.cpaas.io/namespace: cpaas-system # prometheus 命名空间, 保持默认
    alert.cpaas.io/notifications: '["test"]'
    alert.cpaas.io/repeat-config: '{"Critical":"never","High":"5m","Medium":"5m","Low":"5m"}'
    alert.cpaas.io/rules.description: '{} '
    alert.cpaas.io/rules.disabled: '[]'
    alert.cpaas.io/subkind: ''
    cpaas.io/description: ''
    cpaas.io/display-name: policy # 告警策略显示名称
  labels:
    alert.cpaas.io/owner: System
    alert.cpaas.io/project: cpaas-system
    cpaas.io/source: Platform
    prometheus: kube-prometheus
    rule.cpaas.io/cluster: global
    rule.cpaas.io/name: policy
    rule.cpaas.io/namespace: cpaas-system
  name: policy
  namespace: cpaas-system
spec:
  groups:
    - name: general # 告警规则名称
      rules:
        - alert: cluster.blackbox.probe.success-y97ah-9833444d918cab96c43e9ab6efc172cf
          annotations:
            alert_current_value: '{{ $value }}' # 通知时的当前值, 保持默认
          expr:
            max by (job, instance) (probe_success{job=~"test",
            instance=~"https://demo.at-servicecenter.com/"}!=1
            # 连通性告警场景, 请务必修改黑盒监控项名称和目标地址
          for: 30s # 持续时间
          labels:
            alert_cluster: global # 告警所在集群名称
            alert_for: 30s # 持续时间
            alert_indicator: cluster.blackbox.probe.success # 保持不变

```

```

alert_indicator_aggregate_range: '0' # 保持不变
alert_indicator_blackbox_instance: https://demo.at-servicecenter.com/ # 黑盒监控目标地址
alert_indicator_blackbox_name: test # 黑盒监控项名称
alert_indicator_comparison: '!=' # 连通性告警保持配置不变
alert_indicator_query: '' # 日志告警使用, 无需配置此参数
alert_indicator_threshold: '1' # 告警规则阈值, 1 表示连通性, 保持不变

alert_indicator_unit: '' # 告警规则指标单位
alert_involved_object_kind: cluster # 黑盒告警保持不变
alert_involved_object_name: global # 黑盒监控项所在集群
alert_involved_object_namespace: '' # 告警规则所属对象命名空间
alert_name: cluster.blackbox.probe.success-y97ah # 告警规则名称

alert_namespace: cpaas-system # 告警规则所在命名空间
alert_project: cpaas-system # 告警规则所属对象项目名称
alert_resource: policy # 告警规则所在告警策略名称
alert_source: Platform # 告警规则数据类型: Platform-平台数据, Business-业务数据

severity: High # 告警规则级别: Critical-灾难, High-严重, Medium-警告, Low-提示

- alert: cluster.blackbox.http.status.code-235el-99b0095b6b6669415043e14ae84f43bc
  annotations:
    alert_current_value: '{{ $value }}'
    alert_notifications: '["message"]'
  expr:
    max by(job, instance) (probe_http_status_code{job=~"test", instance=~"https://demo.at-servicecenter.com/"}>200
    # HTTP 状态码告警场景, 请务必修改黑盒监控项名称和目标地址
  for: 30s
  labels:
    alert_cluster: global
    alert_for: 30s
    alert_indicator: cluster.blackbox.http.status.code
    alert_indicator_aggregate_range: '0'
    alert_indicator_blackbox_instance: https://demo.at-servicecenter.com/
    alert_indicator_blackbox_name: test
    alert_indicator_comparison: '>'
    alert_indicator_query: ''
    alert_indicator_threshold: '299' # 告警规则阈值, HTTP 状态码告警场景应为三位数, 例如大于 299 (3XX、4XX、5XX) 表示错误
    alert_indicator_unit: ''

```

```
alert_involved_object_kind: Cluster
alert_involved_object_name: global
alert_involved_object_namespace: ''
alert_involved_object_options: Single
alert_name: cluster.blackbox.http.status.code-235el
alert_namespace: cpaas-system
alert_project: cpaas-system
alert_resource: policy33
alert_source: Platform
severity: High
```

## 参考信息

黑盒监控配置文件的完整 YAML 示例如下：



```
apiVersion: v1
data:
  blackbox.yaml: |
    modules:
      http_2xx_example:           # HTTP 探测示例
        prober: http
        timeout: 5s              # 探测超时时间
        http:
          valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
# 返回信息中的版本, 通常默认
          valid_status_codes: [] # 默认为 2xx          # 有效
响应码范围, 返回码在此范围内视为探测成功
```



# 实用指南

## Prometheus 监控数据的备份与恢复

功能概述

使用场景

前提条件

操作流程

操作结果

了解更多

后续操作

## VictoriaMetrics 监控数据备份与恢复

功能简介

使用场景

前置条件

操作步骤

操作结果

了解更多

后续操作

## 从自定义命名空间

功能概述

适用场景

前提条件

操作步骤

操作结果

了解更多

后续操作

# Prometheus 监控数据的备份与恢复

## 目录

### 功能概述

使用场景

前提条件

操作流程

备份数据

方法一：备份存储目录（推荐）

方法二：快照备份

恢复数据

操作结果

了解更多

TSDB 数据格式说明

数据备份注意事项

后续操作

## 功能概述

Prometheus 监控数据以 TSDB（时间序列数据库）格式存储，支持备份与恢复功能。监控数据存储在 Prometheus 容器内的指定路径（由配置项 `storage.tsdb.path` 指定，默认值为 `/prometheus`）。



```
template:
  spec:
    containers:
      - args:
        - '--storage.tsdb.path=/prometheus' # Prometheus 容器中存储监控数
据的目录
```

## 使用场景

- 系统迁移时保留历史监控数据
- 防止因意外事件导致的数据丢失
- 将监控数据迁移至新的 Prometheus 实例

## 前提条件

- 已安装 ACP Monitoring 中的 Prometheus 插件（计算组件名称为 `prometheus-kube-prometheus-0`，类型为 `StatefulSet`）
- 具备集群管理员权限
- 确保目标存储位置有足够的存储空间

## 操作流程

### 备份数据

开始备份前请注意：Prometheus 存储监控数据时，先将采集的数据放入缓存，随后定期写入存储目录。以下备份方式均以存储目录作为数据源，因此备份时不包含缓存中的数据。

#### 方法一：备份存储目录（推荐）

1. 使用 `kubectl cp` 命令备份：

```
kubectl cp -n cpaas-system prometheus-kube-prometheus-0-0:/prometheus -
c prometheus <目标存储路径>
```

2. 从存储后端备份（根据安装时选择的存储类型）：

- **LocalVolume**：从 `/cpaas/monitoring/prometheus` 目录复制
- **PV**：从 PV 挂载目录复制（建议将 PV 的 **persistentVolumeReclaimPolicy** 设置为 `Retain`）
- **StorageClass**：从 PV 挂载目录复制

## 方法二：快照备份

1. 启用 Admin API：

```
kubectl edit -n cpaas-system prometheus kube-prometheus-0
```

添加配置：

```
spec:
  enableAdminAPI: true
```

注意：更新保存配置后，Prometheus Pod（Pod 名称：prometheus-kube-prometheus-0-0）将重启。请等待所有 Pod 状态变为 Running 后再进行后续操作。

2. 创建快照：

```
curl -XPOST <Prometheus Pod IP>:9090/api/v1/admin/tsdb/snapshot
```

## 恢复数据

1. 将备份数据复制到 Prometheus 容器：

```
kubectl cp ./prometheus-backup cpaas-system/prometheus-kube-prometheus-
0-0:/prometheus/
```

## 2. 将数据移动到指定目录：

```
kubectl exec -it -n cpaas-system prometheus-kube-prometheus-0-0 -c prometheus sh
mv /prometheus/prometheus-backup/* /prometheus/
```

快捷方式：插件安装时若存储类型为 **LocalVolume**，可直接将备份数据复制到插件所在节点的 `/cpaas/monitoring/prometheus/prometheus-db/` 目录。

## 操作结果

- 备份完成后，可在目标存储路径看到完整的 TSDB 格式监控数据
- 恢复完成后，Prometheus 会自动加载历史监控数据

## 了解更多

### TSDB 数据格式说明

TSDB 格式数据结构示例：

```
├─ 01FXP317QBANGAX1XQAXCJK9DB
|   ├─ chunks
|   |   └─ 000001
|   ├─ index
|   ├─ meta.json
|   └─ tombstones
├─ chunks_head
|   ├─ 000022
|   └─ 000023
├─ queries.active
└─ wal
    ├─ 00000020
    ├─ 00000021
    ├─ 00000022
    ├─ 00000023
    └─ checkpoint.00000019
        └─ 00000000
```

## 数据备份注意事项

- 备份数据不包含备份时缓存中的数据
- 建议定期执行数据备份
- 使用 PV 存储时，建议将 **persistentVolumeReclaimPolicy** 设置为 **Retain**

## 后续操作

- 验证监控数据是否已正确恢复
- 定期制定数据备份计划
- 若使用快照备份方式，可选择关闭 Admin API

# VictoriaMetrics 监控数据备份与恢复

## 目录

### 功能简介

使用场景

前置条件

操作步骤

1. 确认存储路径

2. 执行数据备份

3. 执行数据恢复

操作结果

了解更多

后续操作

## 功能简介

VictoriaMetrics 监控数据备份与恢复功能允许您对监控数据进行定期备份和必要时的数据恢复，以确保监控数据的安全性和可用性。

## 使用场景

- 定期备份监控数据以防数据丢失

- 系统迁移时的数据迁移
- 灾难恢复
- 测试环境数据重建

## 前置条件

- 集群已安装 ACP Monitoring with VictoriaMetrics 插件
- 确保有足够的存储空间用于备份
- 具备对 VictoriaMetrics 存储路径的访问权限

## 操作步骤

### 1. 确认存储路径

VictoriaMetrics 的监控数据存储于容器的指定路径下，由 `-storageDataPath` 参数指定，默认为 `/vm-data`。

配置示例：

```
spec:
  template:
    spec:
      containers:
        - args:
          - '-storageDataPath=/vm-data'
```

说明：ACP Monitoring with VictoriaMetrics 插件的计算组件名称为 `vmstorage-cluster`，类型为 `StatefulSet`。

### 2. 执行数据备份

使用 `vmbackup` 工具进行数据备份，详细操作请参考 [vmbackup 官方文档](#)。

## 3. 执行数据恢复

使用 `vmrestore` 工具恢复备份数据，详细操作请参考 [vmrestore 官方文档](#)。

## 操作结果

完成备份后，您将获得一份完整的监控数据备份文件。执行恢复操作后，您的监控数据将恢复到备份时的状态。

## 了解更多

- [VictoriaMetrics 官方文档](#)
- [数据备份最佳实践](#)
- [数据恢复故障排查](#)

## 后续操作

- 验证备份数据的完整性
- 设置定期备份计划
- 定期测试恢复流程
- 监控备份任务的执行状态

# 从自定义命名的网络接口采集网络数据

## 目录

### 功能概述

适用场景

前提条件

操作步骤

操作结果

了解更多

后续操作

## 功能概述

平台支持通过修改监控组件的配置，从自定义命名的网络接口采集网络数据，使您能够在监控页面查看这些接口的网络流量信息。

## 适用场景

当您的节点使用自定义命名的网络接口（不符合 `eth.|en.|wl.*|ww.*` 命名规范）且需要在平台上监控这些接口的网络流量数据时适用。

## 前提条件



- 已创建业务集群
- 您拥有平台管理员权限
- 已知自定义网络接口的命名规范

## 操作步骤

1. 点击平台顶部导航栏中的 CLI 工具图标。
2. 点击 **global**。
3. 在 **global** 集群中，查找对应您业务集群的 moduleinfo 资源名称：

```
kubectl get moduleinfo | grep -E 'prometheus|victoriametrics'
```

示例输出：

```
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
```

4. 编辑业务集群的 moduleinfo 资源：

```
kubectl edit moduleinfo <业务集群的 moduleinfo 资源名称>
```

5. 添加或修改 valuesOverride 字段：

```
spec:
  valuesOverride: # 如果该字段不存在，则需要在 spec 下新增 valuesOverride 字段
                  及以下参数
    ait/chart-cpaas-monitor:
      global:
        indicator:
          networkDevice: eth.*|em.*|en.*|wl.*|ww.*|[A-Z].*i|custom_inte
rface # 将 custom_interface 替换为自定义的正则表达式，确保正确匹配网络接口名称
```

6. 等待 10 分钟后，检查节点监控页面上的网络相关图表，确认配置已生效。

## 操作结果

配置生效后，您可以在平台监控页面查看自定义命名网络接口的以下数据：

- 网络流量数据
- 网络吞吐量
- 网络包统计

## 了解更多

- 有关网络监控的更多信息，请参阅[监控架构文档](#)

## 后续操作

- 监控自定义网络接口的性能指标
- 根据监控数据设置告警规则

# 权限说明

监控模块可用的权限点以及平台内置角色的权限如下：

功能	操作	平台 管理员	平台 审计 人员	项目 管理 员	命名空 间管理 员	开发 人员
告警 aiops-alerts	查看	✓	✓	✓	✓	✓
	创建	✓	×	✓	✓	✓
	更新	✓	×	✓	✓	✓
	删除	✓	×	✓	✓	✓
告警模板 aiops-alerttemplate	查看	✓	✓	✓	✓	✓
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×
告警历史 aiops-alerthistories	查看	✓	✓	✓	✓	✓
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×
监控指标 aiops-monitoring- metrics	查看	✓	✓	✓	✓	✓
	创建	✓	×	×	×	×

功能	操作	平台 管理 员	平台 审计 人员	项目 管理 员	命名空 间管理 员	开发 人员
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×
监控面板 aiops-monitoring- dashboard	查看	✓	✓	✓	✓	✓
	创建	✓	×	✓	✓	✓
	更新	✓	×	✓	✓	✓
	删除	✓	×	✓	✓	✓
通知 aiops-notifications	查看	✓	✓	×	×	×
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×
通知管理 aiops- notificationsmanage	查看	✓	✓	✓	✓	✓
	创建	✓	×	✓	×	×
	更新	✓	×	✓	×	×
	删除	✓	×	✓	×	×

# 调用链

---

## 介绍

### 介绍

优势

应用场景

使用限制

## 安装

### 安装

安装 Jaeger Operator

部署 Jaeger 实例

安装 OpenTelemetry Operator

部署 OpenTelemetry 实例

启用功能开关

卸载追踪系统

## 架构

---

## 架构

核心组件

数据流程

# 核心概念

## 核心概念

Telemetry

OpenTelemetry

Span

Trace

Instrumentation

OpenTelemetry Collector

Jaeger

# 操作指南

## 查询追踪

功能概述

主要特性

功能优势

追踪查询

查询结果分析

## 查询追踪日志

特性概述

核心功能

前提条件

日志查询操作

## 实用指南

### Java 应用无侵入方式接入调用链

功能简介

使用场景

前置条件

操作步骤

操作结果

### 与 TraceID 相关的业务日志

背景

将 TraceID 添加到 Java 应用日志

将 TraceID 添加到 Python 应用日志

验证方法

## 问题处理

### 查询不到所需的调用链

问题描述

根因分析

根因 1 的解决方案

根因 2 的解决方案

### 调用链数据不完整

问题描述

根因分析

根因 1 的解决方案

根因 2 的解决方案

# 介绍

分布式追踪是容器平台可观测性体系中的关键模块，用于实现分布式系统的端到端追踪与分析。该模块基于 OpenTelemetry (OTel) 标准构建，提供从数据采集、存储到可视化分析的完整解决方案，帮助开发与运维人员快速定位服务调用异常、分析性能瓶颈，并追踪请求的整个生命周期行为。

通过整合开源技术栈和自研组件，该模块支持端到端的追踪能力：应用通过 **OTel 自动注入** 或 **SDK** 集成方式生成追踪数据，这些数据经过统一采集后存储于 Elasticsearch，最终通过定制化的 UI 实现多维度可视化分析。用户可以利用 **TraceID**、服务名称、标签等丰富条件进行精准检索。

## 目录

### 优势

应用场景

使用限制

## 优势

追踪的核心优势如下：

- 端到端追踪能力  
支持跨服务、跨进程、跨容器边界的完整追踪还原，精准展示微服务架构中的复杂调用关系。



- 灵活的数据采集方式  
提供自动注入（无代码改造）与 SDK 集成的双重模式，兼容 Java/Python/Go 等主流编程语言的应用。
- 高性能存储方案  
采用 Elasticsearch 作为存储后端，支持海量 span 数据的写入与快速检索。
- 灵活的查询和分析能力  
自研的 UI 通过集成 `jaeger-query` API，支持基于 TraceID、服务归属、标签、span 类型等多维条件的灵活查询，方便用户快速定位问题根源。
- 标准化协议支持  
基于 OpenTelemetry 标准构建，能够集成其他 OTel 云原生组件生成的追踪数据。

## 应用场景

追踪的主要应用场景如下：

- 分布式系统故障诊断  
在微服务架构中，完整的追踪能力可以快速定位服务故障和异常调用，减少故障诊断的时间。
- 性能瓶颈分析  
通过查看各服务之间的调用延迟，可以识别性能瓶颈，从而指导系统优化和资源调整。
- 服务依赖关系分析  
借助时序瀑布图，清晰展示服务间的调用路径和依赖关系，辅助架构师进行系统设计和改进。

## 使用限制

在使用追踪时，应注意以下限制：

- 采样策略与性能的平衡
  - 在高负载场景中，追踪数据的采集可能对 Elasticsearch 的性能和存储造成一定压力，因此建议根据业务情况合理配置采样率。



# 安装

## WARNING

本部署文档仅适用于容器平台与追踪系统集成的场景。

**Tracing** 组件与 **Service Mesh** 组件互斥。如果您已部署了 Service Mesh 组件，请先卸载它。

本指南为集群管理员提供在 Alauda Container Platform 集群上安装追踪系统的流程。

前提条件：

- 您拥有具备 `platform-admin-system` 权限的账号，可以访问 Alauda Container Platform 集群。
- 您已安装 `kubectl` CLI。
- 已搭建好用于存储追踪数据的 `Elasticsearch` 组件，包括访问 URL 和 `Basic Auth` 信息。

## 目录

### 安装 Jaeger Operator

通过 Web 控制台安装 Jaeger Operator

部署 Jaeger 实例

安装 OpenTelemetry Operator

通过 Web 控制台安装 OpenTelemetry Operator

部署 OpenTelemetry 实例

启用功能开关

卸载追踪系统

删除 OpenTelemetry 实例

卸载 OpenTelemetry Operator

删除 Jaeger 实例

卸载 Jaeger Operator

---

## 安装 Jaeger Operator

### 通过 Web 控制台安装 Jaeger Operator

您可以在 Alauda Container Platform 的 **Marketplace** → **OperatorHub** 中找到并安装 Jaeger Operator。

#### 步骤

- 在 Web 控制台的 平台管理 视图中，选择要部署 Jaeger Operator 的 集群，然后进入 **Marketplace** → **OperatorHub**。
- 在搜索框中搜索 `Alauda build of Jaeger`，点击 **Alauda build of Jaeger** 标题。
- 阅读 **Alauda build of Jaeger** 页面上的 Operator 介绍信息，点击 安装。
- 在 安装 页面：
  - 选择 手动 作为 升级策略。对于 `Manual` 审批策略，OLM 会创建更新请求。作为集群管理员，您需要手动批准 OLM 的更新请求以升级 Operator 到新版本。
  - 选择 **stable** (默认) 频道。
  - 选择 推荐 作为 安装位置。将 Operator 安装在推荐的 `jaeger-operator` 命名空间中，使 Operator 能够监控并在集群内所有命名空间中可用。
- 点击 安装。
- 确认 状态 显示为 **Succeeded**，以确认 Jaeger Operator 安装成功。
- 检查 Jaeger Operator 的所有组件是否成功安装。通过终端登录集群，执行以下命令：

```
kubectl -n jaeger-operator get csv
```

示例输出

NAME	DISPLAY	VERSION	REPLACES	PHASE
jaeger-operator.vx.x.0	Jaeger Operator	x.x.0		Succeeded

如果 **PHASE** 字段显示为 **Succeeded**，表示 Operator 及其组件已成功安装。

## 部署 Jaeger 实例

可以使用 `install-jaeger.sh` 脚本安装 Jaeger 实例及其相关资源，该脚本接受三个参数：

- `--es-url`：Elasticsearch 的访问 URL。
- `--es-user-base64`：Elasticsearch 的 **Basic Auth** 用户名，base64 编码。
- `--es-pass-base64`：Elasticsearch 的 **Basic Auth** 密码，base64 编码。

从 **DETAILS** 复制安装脚本，登录到目标集群，保存为 `install-jaeger.sh`，并赋予执行权限后运行：

### ► DETAILS

脚本执行示例：

```
./install-jaeger.sh --es-url='https://xxx' --es-user-base64='xxx' --es-pass-base64='xxx'
```

脚本输出示例：

```
ES_URL: https://xxx
ES_USER_BASE64: xxx
ES_PASS_BASE64: xxx
CLUSTER_NAME: cluster-xxx
PLATFORM_URL: https://xxx
INSTALLED_CSV: jaeger-operator.vx.x.x
OAUTH2_PROXY_IMAGE: build-harbor.alauda.cn/3rdparty/oauth2-proxy/oauth2-proxy:vx.x.x
configmap/jaeger-oauth2-proxy created
secret/jaeger-oauth2-proxy created
secret/jaeger-elasticsearch-basic-auth created
serviceaccount/jaeger-prod-acp created
role.rbac.authorization.k8s.io/jaeger-prod-acp created
rolebinding.rbac.authorization.k8s.io/jaeger-prod-acp created
jaeger.jaegertracing.io/jaeger-prod created
podmonitor.monitoring.coreos.com/jaeger-monitor created
ingress.networking.k8s.io/jaeger-query created
Jaeger installation completed
```

## 安装 OpenTelemetry Operator

### 通过 Web 控制台安装 OpenTelemetry Operator

您可以在 Alauda Container Platform 的 **Marketplace** → **OperatorHub** 中找到并安装 OpenTelemetry Operator。

#### 步骤

- 在 Web 控制台的 平台管理 视图中，选择要部署 OpenTelemetry Operator 的 集群，然后进入 **Marketplace** → **OperatorHub**。
- 在搜索框中搜索 `Alauda build of OpenTelemetry`，点击 **Alauda build of OpenTelemetry** 标题。
- 阅读 **Alauda build of OpenTelemetry** 页面上的 Operator 介绍信息，点击 安装。
- 在 安装 页面：

- 选择 手动 作为 升级策略。对于 `Manual` 审批策略，OLM 会创建更新请求。作为集群管理员，您需要手动批准 OLM 的更新请求以升级 Operator 到新版本。
- 选择 **alpha** (默认) 频道。
- 选择 推荐 作为 安装位置。将 Operator 安装在推荐的 `opentelemetry-operator` 命名空间中，使 Operator 能够监控并在集群内所有命名空间中可用。
- 点击 安装。
- 确认 状态 显示为 **Succeeded**，以确认 OpenTelemetry Operator 安装成功。
- 检查 OpenTelemetry Operator 的所有组件是否成功安装。通过终端登录集群，执行以下命令：

```
kubectl -n opentelemetry-operator get csv
```

示例输出

NAME	DISPLAY	VERSION	REPL
ACES	PHASE		
openTelemetry-operator.vx.x.0	OpenTelemetry Operator	x.x.0	
Succeeded			

如果 `PHASE` 字段显示为 `Succeeded`，表示 Operator 及其组件已成功安装。

## 部署 OpenTelemetry 实例

可以使用 `install-otel.sh` 脚本安装 OpenTelemetry 实例及其相关资源。

从 **DETAILS** 复制安装脚本，登录到目标集群，保存为 `install-otel.sh`，并赋予执行权限后运行：

► DETAILS

脚本执行示例：

```
./install-otel.sh
```

脚本输出示例：

```
CLUSTER_NAME: cluster-xxx
serviceaccount/otel-collector created
clusterrolebinding.rbac.authorization.k8s.io/otel-collector:cpaas-system:
cluster-admin created
opentelemetrycollector.opentelemetry.io/otel created
instrumentation.opentelemetry.io/acp-common-java created
servicemonitor.monitoring.coreos.com/otel-collector-monitoring created
servicemonitor.monitoring.coreos.com/otel-collector created
OpenTelemetry installation completed
```

## 启用功能开关

追踪系统目前处于 **Alpha** 阶段，您需要在 功能开关 视图中手动启用 `acp-tracing-ui` 功能开关。

然后，进入 容器平台 视图，导航至 可观测性 → **Tracing**，即可查看追踪功能。

## 卸载追踪系统

### 删除 OpenTelemetry 实例

登录已安装的集群，执行以下命令删除 OpenTelemetry 实例及其相关资源。

```
kubectl -n cpaas-system delete servicemonitor otel-collector-monitoring
kubectl -n cpaas-system delete servicemonitor otel-collector
kubectl -n cpaas-system delete instrumentation acp-common-java
kubectl -n cpaas-system delete opentelemetrycollector otel
kubectl delete clusterrolebinding otel-collector:cpaas-system:cluster-admin
kubectl -n cpaas-system delete serviceaccount otel-collector
```



## 卸载 OpenTelemetry Operator

您可以通过 Web 控制台的 平台管理 视图卸载 OpenTelemetry Operator。

### 步骤

- 进入 **Marketplace** → **OperatorHub**，使用 搜索框 搜索 `Alauda build of OpenTelemetry`。
- 点击 **Alauda build of OpenTelemetry** 标题进入详情页。
- 在详情页右上角点击 卸载 按钮。
- 在弹出的 卸载 "opentelemetry-operator"? 窗口中点击 卸载。

## 删除 Jaeger 实例

登录已安装的集群，执行以下命令删除 Jaeger 实例及其相关资源。

```
kubectl -n cpaas-system delete ingress jaeger-query
kubectl -n cpaas-system delete podmonitor jaeger-monitor
kubectl -n cpaas-system delete jaeger jaeger-prod
kubectl -n cpaas-system delete rolebinding jaeger-prod-acp
kubectl -n cpaas-system delete role jaeger-prod-acp
kubectl -n cpaas-system delete serviceaccount jaeger-prod-acp
kubectl -n cpaas-system delete secret jaeger-oauth2-proxy
kubectl -n cpaas-system delete secret jaeger-elasticsearch-basic-auth
kubectl -n cpaas-system delete configmap jaeger-oauth2-proxy
```

## 卸载 Jaeger Operator

您可以通过 Web 控制台的 平台管理 视图卸载 Jaeger Operator。

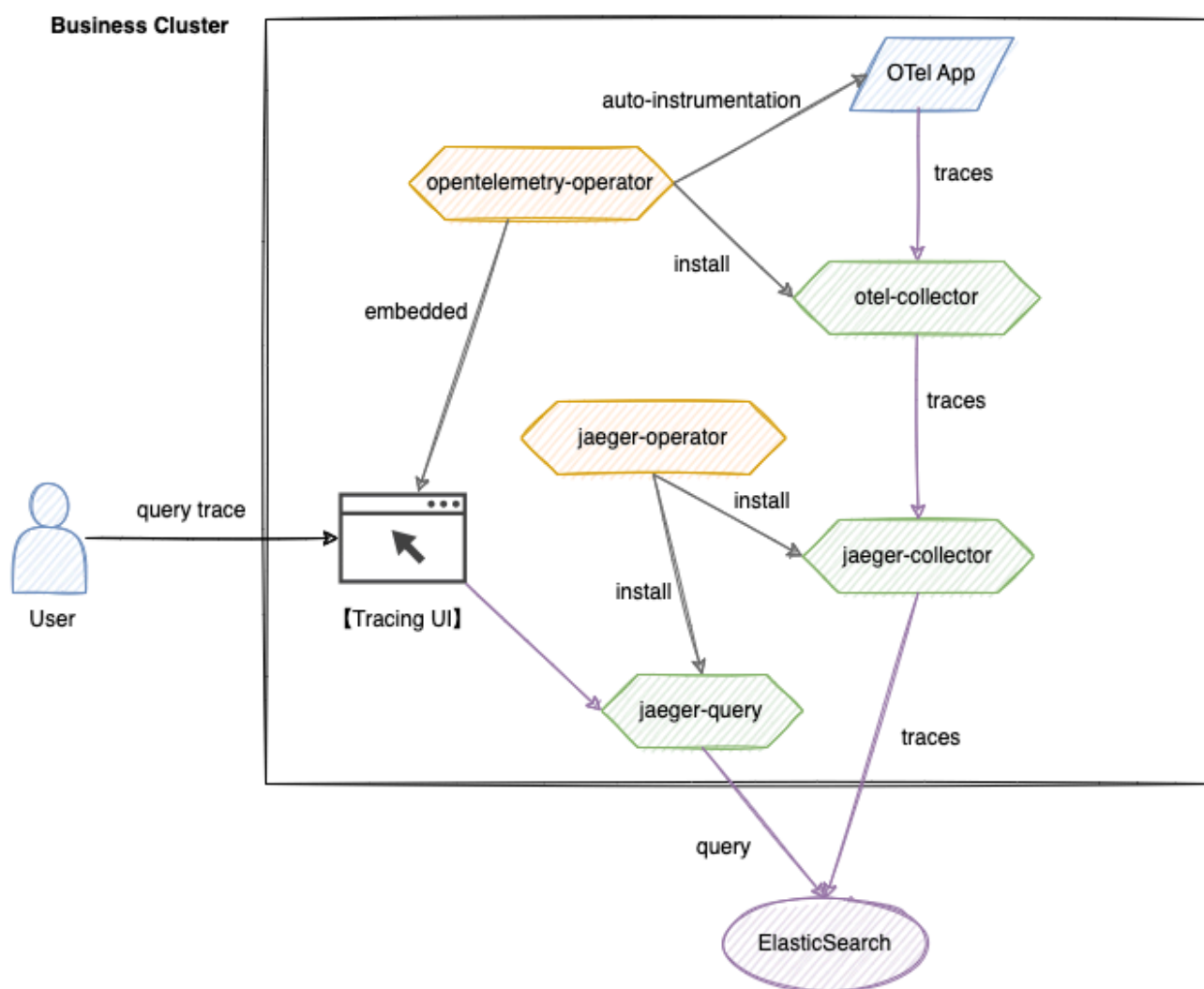
### 步骤

- 进入 **Marketplace** → **OperatorHub**，使用 搜索框 搜索 `Alauda build of Jaeger`。
- 点击 **Alauda build of Jaeger** 标题进入详情页。
- 在详情页右上角点击 卸载 按钮。
- 在弹出的 卸载 "jaeger-operator"? 窗口中点击 卸载。



# 架构

本架构基于 OpenTelemetry 和 Jaeger 技术栈构建，实现分布式调用链的全生命周期管理。系统包含数据采集、传输、存储、查询和可视化五大核心模块。



## 目录

核心组件

数据流程

# 核心组件

## 1. OpenTelemetry 体系

- **opentelemetry-operator**

集群级 Operator，负责部署与管理 otel-collector 组件，并提供 OTel 自动注入能力。

- **otel-collector**

接收来自应用程序的追踪数据，进行过滤、批处理后转发至 jaeger-collector。

- **调用链 UI**

集成 jaeger-query API 的自研可视化界面，支持多维查询条件。

## 2. Jaeger 体系

- **jaeger-operator**

部署与管理 jaeger-collector 和 jaeger-query 组件。

- **jaeger-collector**

接收 otel-collector 转发处理后的调用链数据，进行格式转换后写入 Elasticsearch。

- **jaeger-query**

提供调用链查询 API，支持 TraceID、标签等多条件检索。

## 3. 存储层

- **Elasticsearch**

分布式存储引擎，支持海量 Span 数据的高效写入与检索。

# 数据流程

- **写入流程**

```
应用程序 -> otel-collector -> jaeger-collector -> Elasticsearch
```

应用通过 SDK 或自动注入生成 Span 数据，经 otel-collector 标准化处理后，由 jaeger-collector 持久化到 Elasticsearch。

- 查询流程

用户 -> 调用链 UI -> jaeger-query -> Elasticsearch

用户通过 UI 提交查询条件，jaeger-query 从 Elasticsearch 检索数据，UI 根据返回结果进行可视化展示。

# 核心概念

---

## 目录

### Telemetry

OpenTelemetry

Span

Trace

Instrumentation

OpenTelemetry Collector

Jaeger

---

## Telemetry

Telemetry 指的是系统及其行为发出的数据，包括链路、指标和日志。

## OpenTelemetry

OpenTelemetry 是一个[可观测性](#) 框架和工具包，旨在创建和管理遥测数据，如[链路](#)、[指标](#) 和 [日志](#)。重要的是，OpenTelemetry 是供应商无关的，这意味着它可以与各种可观测性后端一起使用，包括 [Jaeger](#) 和 [Prometheus](#) 这类开源工具以及商业产品。

## Span

---

**Span**（跨度）是分布式追踪的基本构建块，代表一个具体的操作或工作单元。每个 span 记录了请求中的特定动作，帮助我们理解操作执行过程中发生的详细情况。

一个 span 包含名称、时间相关的数据、结构化的日志消息及其他元数据（属性），这些信息共同描绘了操作的完整图景。

## Trace

**Trace**（追踪）记录了请求（无论来自应用程序还是终端用户）在多服务架构（如微服务和无服务器应用）中的传播路径。

一个追踪由一个或多个 spans 组成。第一个 span 被称为根 span，它代表了请求从开始到结束的整个生命周期。根 span 下的子 span 提供更详细的上下文信息，关于请求处理过程中的各个步骤。

没有追踪的话，在分布式系统中识别性能问题的根本原因将非常具有挑战性。追踪通过分解请求在系统中的流动过程，简化了分布式系统的调试和理解。

## Instrumentation

为了实现可观测性，系统需要进行插桩（**Instrumentation**）：也就是系统的组件代码必须发出[链路](#)、[指标](#)和[日志](#)。

通过 OpenTelemetry，您可以通过两种主要方式对代码进行插桩：

1. [基于代码的解决方案](#)：使用官方提供的[适用于大多数语言的 API 和 SDK](#)
2. [零侵入解决方案](#)

基于代码的解决方案能让您从应用内部获得更深入的洞察和丰富的遥测数据。您可以通过 OpenTelemetry API 在应用中生成遥测数据，这是对零侵入解决方案生成的遥测数据的重要补充。

零侵入解决方案适合快速入门，或在您无法修改需获取遥测数据的应用程序时使用。它们可以通过您所使用的库或运行环境提供丰富的遥测数据。另一种理解是，它们提供关于应用程序边界（Edges）发生的事件信息。

这两种解决方案可以同时使用。

## OpenTelemetry Collector

OpenTelemetry Collector 是一个供应商无关的代理，能够接收、处理和导出遥测数据。它支持以多种格式接收遥测数据（如 OTLP、Jaeger、Prometheus 以及许多商业/专有工具），并将数据发送到一个或多个后端。它还支持在导出之前处理和过滤遥测数据。

更多信息请参见 [Collector](#) ↗。

## Jaeger

**Jaeger** 是一款开源的 分布式追踪系统。它旨在监控和诊断基于微服务架构的复杂分布式系统，帮助开发者可视化请求追踪、分析性能瓶颈及排查异常。Jaeger 兼容 **OpenTracing** 标准（现为 OpenTelemetry 的一部分），支持多种编程语言和存储后端，是云原生领域的关键可观测性工具。



# 操作指南

## 查询追踪

功能概述

主要特性

功能优势

追踪查询

查询结果分析

## 查询追踪日志

特性概述

核心功能

前提条件

日志查询操作



# 查询追踪

## 目录

### 功能概述

主要特性

功能优势

追踪查询

步骤 1：组合查询条件

步骤 2：执行查询

查询结果分析

Span 列表

时序瀑布图

Span 详情

## 功能概述

分布式追踪查询功能为微服务架构提供完整的链路追踪能力，通过收集服务间调用的元数据，帮助用户快速定位跨服务调用问题。此功能主要解决以下问题：

- 请求链路追踪：在复杂分布式系统中还原完整的请求路径。
- 性能瓶颈分析：识别链路中时间消耗异常的调用节点。
- 故障根因定位：通过错误标记快速定位问题发生的点。

适用场景包括：

- 在生产环境故障排查时快速定位异常服务。
- 在性能调优时识别高延迟调用链。
- 在新版本发布后验证服务间的调用关系。

核心价值：

- 提升分布式系统的可观测性。
- 缩短平均恢复时间（MTTR）。
- 优化服务间的调用性能。

## 主要特性

- 多维度查询：支持 TraceID、服务名称、标签等六种查询条件组合。
- 可视化分析：通过时序瀑布图直观地展示调用层级及时间分布。
- 精准定位：支持错误 Span 过滤和标签二次检索。

## 功能优势

- 快速识别问题：通过多维度查询条件缩小排查范围，加快问题定位速度。
- 可视化呈现：使用时序瀑布图直观展现调用关系，降低复杂性并提升故障分析的效率。
- 灵活多样：同时支持简单查询和复杂组合查询，适应各种运维和开发场景。

## 追踪查询

### 1 步骤 1：组合查询条件

提示：查询条件可组合使用，您可以通过添加多个查询条件来精细化查询。

查询条件	说明
TraceID	完整链路的唯一标识，可用于查询指定的追踪。

查询条件	说明
服务	发起/接收调用请求的服务（需要选择或输入）。
标签	您可以通过输入标签（Tag）过滤查询结果，支持的标签包括 Span 详情中的标签。
Span 耗时 大于	耗时大于或等于 输入值（毫秒）的 Span。
仅搜索错误 Spans	错误 Span 是指 Tag 值 <b>error</b> 为 <code>true</code> 的 Spans。
Span 类型	根 Span：搜索由已配置的服务发起的根 Span。当配置的服务是整个调用请求的发起者时使用此搜索模式。 服务入口 Span：搜索配置的服务被调用时生成的第一个 Span。
最大查询条 数	可查询的最大 Span 数量，默认为 200。 提示：出于性能考虑，平台一次最多展示 1000 个 Span。如果符合查询条件的 Span 数量超过 最大查询条数，您可以细化查询条件或缩小时间范围进行阶段性查询。

## 2 步骤 2：执行查询

- 选择查询条件并输入相应值后，单击 添加到查询条件 按钮，当前条件将显示在 查询条件 结果区域，并触发查询。
- 您还可以展开 常用查询条件，快速添加近期使用过的搜索条件。

## 查询结果分析

输入查询条件并搜索后，页面将生成查询结果区域。

### Span 列表

查询结果区域左侧显示符合条件的 Span 列表及其基本信息，包括：服务名称、调用的接口或处理请求的方法、耗时及开始时间。

## 时序瀑布图

查询结果区域右侧的时序瀑布图清晰展示了一次追踪中的 Span 之间的调用关系。在追踪分析中使用时序瀑布图的主要特点如下：

1. 自上而下的展开：时序瀑布图中的各个调用事件（Spans）通常自图表上方向下展开，每个水平条形代表一个服务调用或处理过程，位置反映了调用的逻辑顺序。
2. 时间轴对齐：时序瀑布图的横轴代表时间。每个条形的长度表示该调用的持续时间，允许直观比较不同调用间的时间关系。
3. 缩进描述：缩进表示调用的层级关系，缩进越深表示在该链路中调用的深度越大。
4. 交互性和详细数据展示：点击时序瀑布图中的条形可以显示该调用的更多详细信息。

## Span 详情

通过单击时序瀑布图中 Span 所在行，可以展开并查看 Span 的详细信息，其中包含：

- 服务：Span 中的服务。
- Span 耗时（毫秒）：Span 持续时间。
- URL：服务访问的 URL，对应于 Span 标签中的 **http.url**。
- 标签：Span 的标签信息，由键值对组成，可用于高级搜索的标签查询条件。通过点击标签旁的按钮，可以将当前标签条件加入查询条件，以进一步精确查询结果。
- JSON：Span 的原始 JSON 结构，允许进一步检查其内部信息。

# 查询追踪日志

## 目录

### 特性概述

核心功能

前提条件

日志查询操作

访问追踪日志

过滤日志

按 Pod 名称

按时间范围

按查询条件

包含 Trace ID

高级操作

导出日志

自定义显示字段

查看日志上下文

## 特性概述

追踪日志使用户能够使用唯一的 TraceID 查询和分析与特定分布式追踪相关的日志。此功能帮助开发人员和运维人员快速定位问题，理解请求流程，并将业务日志与追踪上下文关联起来。

主要优势：

- 根本原因分析：识别分布式系统中微服务的错误和延迟问题。
- 上下文关联：将业务日志与追踪跨度链接，以实现端到端可见性。
- 高效过滤：按 Pods 或 TraceID 过滤日志，以关注相关数据。

适用场景：

- 调试跨服务事务失败。
- 分析复杂工作流中的性能瓶颈。
- 审计请求处理流程以确保合规性。

## 核心功能

- 基于 **TraceID** 的查询：使用特定的 TraceID 检索与之关联的所有日志。
- 以 **Pod** 为中心的过滤：查看参与追踪的特定 Pods 的日志。
- 日志导出：以可定制格式导出过滤后的日志数据。
- 上下文日志查看：检查目标条目前后日志记录以进行更深入的分析。

## 前提条件

### TIP

在通过 TraceID 查询追踪日志之前，您必须先对服务进行监控，以便在业务日志中包含 TraceID。请遵循 [Business Log Correlation with TraceID Guide](#) 以获取配置详情。

默认行为：

- 显示整个追踪持续时间的日志。
- 对于少于 1 分钟的追踪，在追踪开始时间之后查询 1 分钟内的日志。

## 日志查询操作

## 1 访问追踪日志

1. 在查询追踪后，点击特定追踪以查看其详细信息。
2. 在追踪可视化面板中点击 [查看日志](#) 标签。

## 2 过滤日志

### 按 **Pod** 名称

在 **Pod** 名称 选择器中，从参与服务列表中选择目标 Pod。

### 按时间范围

在 时间范围 选择器中，选择目标时间范围。

### 按查询条件

在 查询条件 文本框中输入关键字，以根据特定内容过滤日志。

### 包含 **Trace ID**

选择 包含 **Trace ID** 复选框。

## 3 高级操作

### 导出日志

1. 点击 导出。
2. 使用列复选框选择要包含的字段。
3. 选择导出格式（JSON/CSV）。

### 自定义显示字段

点击 设置。 切换显示面板中日志字段的可见性。

### 查看日志上下文



1. 点击任何日志条目旁边的 洞察。
2. 探索目标时间戳前后的 5 条先前和后续日志。
3. 使用鼠标上下滚动以加载更多日志。

# 实用指南

## Java 应用无侵入方式接入调用链

功能简介

使用场景

前置条件

操作步骤

操作结果

## 与 TraceID 相关的业务日志

背景

将 TraceID 添加到 Java 应用日志

将 TraceID 添加到 Python 应用日志

验证方法

# Java 应用无侵入方式接入调用链

## INFO

自动注入的 [OpenTelemetry Java Agent](#) 支持 **Java 8+** 版本。

## 目录

### 功能简介

使用场景

前置条件

操作步骤

操作结果

## 功能简介

调用链追踪是分布式系统可观测性的核心能力，能够完整记录请求在系统内的调用路径与性能数据。本文介绍如何通过自动注入 OpenTelemetry Java Agent 的方式，实现 Java 应用无侵入接入调用链追踪体系。

## 使用场景

适用于以下场景的 Java 应用接入：

- 需要快速为 Java 应用添加调用链追踪能力
- 需要避免修改应用程序源代码
- 使用 Kubernetes 进行服务部署
- 需要可视化服务间调用关系和性能瓶颈分析

## 前置条件

使用本功能前，需确保：

- 目标服务部署在 Alauda 容器平台
- 服务使用 Java 8 或更高 JDK 版本
- 具有目标命名空间的 Deployment 编辑权限
- 平台已完成[调用链部署](#)

## 操作步骤

对要接入 Alauda Container Platform 调用链的 Java 应用，需要进行以下适配：

- 为 Java Deployment 配置自动注入注解。
- 设置 `SERVICE_NAME` 环境变量。
- 设置 `SERVICE_NAMESPACE` 环境变量。

Deployment 适配示例：

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-java-deploy
spec:
  template:
    metadata:
      annotations:
        instrumentation.opentelemetry.io/inject-java: cpaas-system/acp-co
mmon-java ①
      labels:
        app.kubernetes.io/name: my-java-app
    spec:
      containers:
        - env:
          - name: SERVICE_NAME ②
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.labels['app.kubernetes.io/name']
          - name: SERVICE_NAMESPACE ③
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: metadata.namespace

```

1. 选择 `cpaas-system/acp-common-java` Instrumentation 作为注入 Java Agent 的配置。
2. 配置 `SERVICE_NAME` 环境变量，可通过 labels 关联或固定值的方式。
3. 配置 `SERVICE_NAMESPACE` 环境变量，其值为 `metadata.namespace`。

## 操作结果

Java 应用适配后：

- 新启动的 Java 应用 pod 中若存在 `opentelemetry-auto-instrumentation-java` 初始化容器，则表示注入成功。
- 向 Java 应用发送测试请求。
- 在 **Container Platform** 视图中，选择 Java 应用所在的项目、集群和命名空间。

- 导航到 可观测性 -> 调用链 页面，查看 Java 应用的调用链数据和时序瀑布图。

**TIP**

本文将指导开发人员如何在应用代码中集成方法以 获取 **TraceID** 和 将 **TraceID** 添加到应用日志，适合具有一定开发经验的后端开发人员。

# 与 TraceID 相关的业务日志

## 目录

### 背景

将 TraceID 添加到 Java 应用日志

将 TraceID 添加到 Python 应用日志

验证方法

## 背景

- 为了正确地将多个自动发送的跨度（在单个请求中调用的不同模块/节点/服务）关联到一个跟踪中，服务的 HTTP 请求头中将包括 TraceID 和其他用于关联跟踪的信息。
- 一个跟踪表示单个请求的调用过程，TraceID 是标识该请求的唯一 ID。在日志中包含 TraceID 后，跟踪信息可以与应用日志进行关联。

基于上述背景，本文将解释如何从 HTTP 请求头中获取 TraceID 并将其添加到应用日志中，使您能够使用 TraceID 准确查询平台上的日志数据。

# 将 TracelD 添加到 Java 应用日志

## TIP

- 以下示例基于 **Spring Boot** 框架，并使用 **Log4j** 和 **Logback** 进行说明。
- 您的应用程序必须满足以下先决条件：
  - 日志库的类型和版本必须满足以下要求：

日志库	版本要求
<b>Log4j 1</b>	1.2+
<b>Log4j 2</b>	2.7+
<b>Logback</b>	1.0+

- 应用程序已注入 Java Agent。

## 方法1：配置 `logging.pattern.level`

修改您的应用程序配置中的 `logging.pattern.level` 参数，如下所示：

```
logging.pattern.level = trace_id=%mdc{trace_id}
```

## 方法2：配置 `CONSOLE_LOG_PATTERN`

1. 修改 logback 配置文件，如下所示：

## TIP

此处以控制台输出为例，其中 `%X{trace_id}` 表示从 MDC 中检索到的键 `trace_id` 的值。



```
<property name="CONSOLE_LOG_PATTERN"
    value="\${CONSOLE_LOG_PATTERN:-%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} [trace_id=%X{trace_id}] %clr(${LOG_LEVEL_PATTERN:-%5p}) %clr(${PID:- }){magenta} %clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}}"/>
```

2. 在需要输出日志的类中，添加 `@Slf4j` 注解并使用日志对象输出日志，如下所示：

```
@RestController
@Slf4j
public class ProviderController {

    @GetMapping("/hello")
    public String hello(HttpServletRequest request) {
        log.info("request /hello");
        return "hello world";
    }
}
```

## 将 TraceID 添加到 Python 应用日志

1. 在应用代码中，添加以下代码以从请求头中检索 TraceID。示例代码如下，可以根据需要进行调整：

### TIP

`getForwardHeaders` 函数从请求头中检索跟踪信息，其中 `x-b3-traceid` 的值即为 TraceID。

```
def getForwardHeaders(request):
    headers = {}
    incoming_headers = [
        'x-request-id', # 所有应用程序应传递 x-request-id 以用于访问日志
和一致的跟踪/日志采样决策
        'x-b3-traceid', # B3 跟踪头, 与 Zipkin、OpenCensusAgent 和 St
ackdriver 配置兼容
        'x-b3-spanid',
        'x-b3-parentspanid',
        'x-b3-sampled',
        'x-b3-flags',
    ]
    for ihdr in incoming_headers:
        val = request.headers.get(ihdr)
        if val is not None:
            headers[ihdr] = val

    return headers
```

2. 在应用代码中，添加以下代码以将检索到的 TraceID 包含在日志中。示例代码如下，可以根据需要进行调整：

```
headers = getForwardHeaders(request)
tracing_section = ' [%s,%s] ' % headers
logging.info(tracing_section + "Oops, unexpected error happens.")
```

## 验证方法

1. 点击左侧导航栏中的 **Tracing**。
2. 在查询条件中选择 TraceID，输入 TraceID 进行查询，然后点击 **Add to query**。
3. 在显示的跟踪数据中，点击 TraceID 旁边的 **View Log**。
4. 在 **Log Query** 页面上，勾选 **Contain Trace ID**；系统将仅显示包含 TraceID 的日志数据。

# 问题处理

## 查询不到所需的调用链

问题描述

根因分析

根因 1 的解决方案

根因 2 的解决方案

## 调用链数据不完整

问题描述

根因分析

根因 1 的解决方案

根因 2 的解决方案

# 查询不到所需的调用链

## 目录

### 问题描述

#### 根因分析

1. 调用链采样率配置过低
2. Elasticsearch 实时性限制

#### 根因 1 的解决方案

#### 根因 2 的解决方案

## 问题描述

在服务网格中查询调用链时，可能会遇到无法检索到目标调用链的情况。

## 根因分析

### 1. 调用链采样率配置过低

当调用链的采样率参数设置过低时，系统仅会按比例采集调用链数据。在请求量不足或低峰时段，这可能会导致采样数据量低于可见阈值。

### 2. Elasticsearch 实时性限制

Elasticsearch 索引的默认配置为 `"refresh_interval": "10s"`，这导致数据从内存缓冲区刷新到可搜索状态的延迟为10秒。在查询最近生成的调用链时，可能由于数据尚未持久化，结果可能会缺失。

这种索引配置可以有效减少 Elasticsearch 的数据合并压力，提升索引速度和首次查询速度，但也在一定程度上降低了数据的实时性。

## 根因1的解决方案

- 根据需求适当提高采样率。
- 使用更丰富的采样方式，如尾部采样。

## 根因2的解决方案

通过 `jaeger-collector` 的 `--es.asm.index-refresh-interval` 启动参数来调整刷新间隔，默认值为 `10s`。

如果该参数的值为 `"null"`，则不会对索引的 `refresh_interval` 进行配置。

注：配置为 `"null"` 时，会影响 Elasticsearch 的性能和查询速度。

# 调用链数据不完整

## 目录

### 问题描述

#### 根因分析

1. 数据持久化延迟
2. 时间范围限制

#### 根因 1 的解决方案

#### 根因 2 的解决方案

## 问题描述

调用链查询结果出现以下数据不完整现象：

- 近期（30 分钟内）查询结果缺失部分跨度。
- 超过 1 小时的调用链出现断链现象。

## 根因分析

### 1. 数据持久化延迟

Elasticsearch 的写入过程需要经过内存缓冲区（buffer）→ 操作日志（translog）→ 段文件（segment）的处理流程，最新写入的数据可能存在可见性延迟。

## 2. 时间范围限制

默认情况下，`jaeger-query` 查询跨度对应的调用链时，时间范围会在跨度的起始时间前后各延伸一个小时。

例如，某个跨度的起始时间为 `08:12:30`，结束时间为 `08:12:32`，则查询该调用链的时间范围为 `07:12:30` 到 `09:12:32`。

因此，若调用链跨度超过 1 小时，通过此跨度进行查询时，可能无法获得完整的调用链。

### 根因1的解决方案

稍作等待并刷新页面重新尝试查询。

### 根因2的解决方案

如果您环境中的调用链跨度较长，可以通过 `jaeger-query` 的 `--es.asm.span-trace-query-time-adjustment-hours` 启动参数来调整单个调用链的查询时间范围。

该参数默认值为 `1` 小时，您可以根据需求适当增大该值。

# 日志

---

## 介绍

### 介绍

模块介绍

模块优势

模块应用场景

模块使用限制

---

## 安装

### 安装

安装 ACP Log Storage with ElasticSearch

安装 ACP Log Storage with Clickhouse

安装 ACP Log Collector 插件

---

## 架构

---



## 日志模块架构

整体架构说明

日志采集

日志消费及存储

日志可视化

## 日志组件选型指南

架构对比

功能比对

选择建议

## 日志组件容量

ElasticSearch

Clickhouse

# 概念

## 概念

开源组件

核心功能概念

关键技术术语

数据流模型

# 操作指南

## 日志

日志查询分析

管理应用日志保留时间

配置部分应用日志停止采集

# 实用指南

如何归档日志到第三方存储

转存到外部 NFS

转存到外部 S3 存储

如何对接外部 ES 存储集群

资源准备

操作步骤

权限说明

权限说明



# 介绍

## 目录

### 模块介绍

模块优势

模块应用场景

模块使用限制

## 模块介绍

日志模块是 ACP 提供的一套高效、可靠的日志管理解决方案，旨在为用户提供全面的日志采集、存储、查询和分析功能。基于强大的开源组件，系统采用 Filebeat 进行日志采集，ElasticSearch 和 Clickhouse 作为日志存储后端，确保用户能够轻松处理大量日志数据，并实时获取关键业务洞察。

## 模块优势

- 高性能：凭借 ElasticSearch 和 Clickhouse 的强大性能，系统能够处理海量数据，支持快速查询和分析。
- 灵活性：支持多种日志源的采集，能够满足不同业务场景的需求。
- 实时性：提供实时日志处理能力，使用户能够迅速识别和应对系统故障或安全事件。
- 可扩展性：系统架构设计支持横向扩展，能够根据业务增长轻松扩展资源。
- 用户友好：提供可视化的界面和简单的查询语言，让用户能够轻松上手。

## 模块应用场景

- 系统监控：实时监控应用程序和服务器的运行状态，及时发现和处理异常情况。
- 安全审计：收集和分析安全日志，帮助企业识别潜在的安全威胁和违规行为。
- 故障排查：通过日志分析迅速定位故障根源，提高系统恢复效率。
- 业务分析：通过对用户行为和系统性能的日志分析，辅助决策，优化业务流程。

## 模块使用限制

- 容量规划：大规模日志数据存储需要较多的资源规划，请您提前评估您的日志规模，根据 [容量规划](#) 文档提前做好规划。
- 端口开通：如在业务集群安装日志存储组件，需确保 global 集群可访问业务集群的 11780 端口号。
- 组件选型：平台提供了 ElasticSearch 和 Clickhouse 两种不同的日志存储组件，请您根据您的需要，提前做好组件选型。
- 安装规划：平台支持在任意集群安装日志存储组件，任意集群的日志可以提前采集到指定集群的日志存储组件中，请您根据您的机房规划提前做好日志相关组件的安装规划，以避免跨区域流量导致的大量带宽成本。

# 安装

平台的日志系统由 ACP Log Collector 和 ACP Log Storage 两个插件构成，本章节将向您介绍如何安装这两种插件。

## WARNING

1. ACP Log Storage 插件安装成功后，安装组件的集群可被用作平台中各集群的存储集群（安装 ACP Log Collector 插件时可选），为平台中各集群提供日志存储服务。
2. `global` 集群能够查询存储在平台中任一工作负载集群上的日志数据。需确保 `global` 集群可访问工作负载集群的 11780 端口。
3. ACP Log Storage with Clickhouse 插件与 ACP Log Storage with ElasticSearch 插件无法安装在同一集群，请阅读 [选型建议](#) 并选择安装其中一种日志存储插件。

## 目录

[安装 ACP Log Storage with ElasticSearch](#)

[安装 ACP Log Storage with Clickhouse](#)

[安装 ACP Log Collector 插件](#)

## 安装 ACP Log Storage with ElasticSearch

1. 导航至 应用商店管理 > 集群插件，选择目标集群。

- 在 插件 页签下，单击 **ACP Log Storage with ElasticSearch** 右侧的操作按钮 > 安装。
- 参考以下说明，配置相关参数。

参数	说明
对接外部 <b>Elasticsearch</b>	保持关闭，在平台中安装日志存储插件。
组件安装设置	<p><b>LocalVolume</b>：本地存储，日志数据将存放于所选节点的本地存储路径中。使用该方式的优势在于日志组件和本地存储直接绑定，无需通过网络访问存储，可以提供更好的存储性能。</p> <p><b>StorageClass</b>：使用存储类动态创建存储资源以存储日志数据。使用该方式的优势在于具有更高的灵活性；当整个集群定义多个存储类时，管理员可以为日志组件根据使用场景选择对应的存储类，减少主机异常状态对存储的影响。但 StorageClass 的性能可能会受到网络带宽和延迟等因素的影响，并且需要依赖存储后端提供的冗余机制来实现存储的高可用性。</p>
保留时间	<p>各类日志、事件、审计数据可在集群上保留的最长时间，超出保留时间的数据会被自动清理。</p> <p>提示：您可自行备份需长期保留的数据，如需帮助，请联系技术支持人员。</p>

- 单击 安装。

## 安装 ACP Log Storage with Clickhouse

- 导航至 应用商店管理 > 集群插件，选择目标集群。
- 在 插件 页签下，单击 **ACP Log Storage with Clickhouse** 右侧的操作按钮 > 安装。
- 参考以下说明，配置相关参数。

参数	说明
组件安装设置	<p><b>LocalVolume</b>：本地存储，日志数据将存放于所选节点的本地存储路径中。使用该方式的优势在于日志组件和本地存储直接绑定，无需通过网络访问存储，可以提供更好的存储性能。</p> <p><b>StorageClass</b>：使用存储类动态创建存储资源以存储日志数据。使用该方式的优势在于具有更高的灵活性；当整个集群定义多个存储类时，管理员可以为日志组件根据使用场景选择对应的存储类，减少主机异常状态对存储的影响。但 StorageClass 的性能可能会受到网络带宽和延迟等因素的影响，并且需要依赖存储后端提供的冗余机制来实现存储的高可用性。</p>
保留时间	<p>各类日志、事件、审计数据可在集群上保留的最长时间，超出保留时间的数据会被自动清理。</p> <p>提示：您可自行备份需长期保留的数据，如需帮助，请联系技术支持人员。</p>

4. 单击 安装。

## 安装 ACP Log Collector 插件

1. 导航至 应用商店管理 > 集群插件，选择目标集群。
2. 在 插件 页签下，单击 **ACP Log Collector** 右侧的操作按钮 > 安装。
3. 选择 存储集群（已安装了 ACP Log Storage），并单击 *选中/取消选中* 日志类型，设置集群中日志采集的范围。
4. 单击 安装。

# 架构

## 日志模块架构

- 整体架构说明
- 日志采集
- 日志消费及存储
- 日志可视化

## 日志组件选型指南

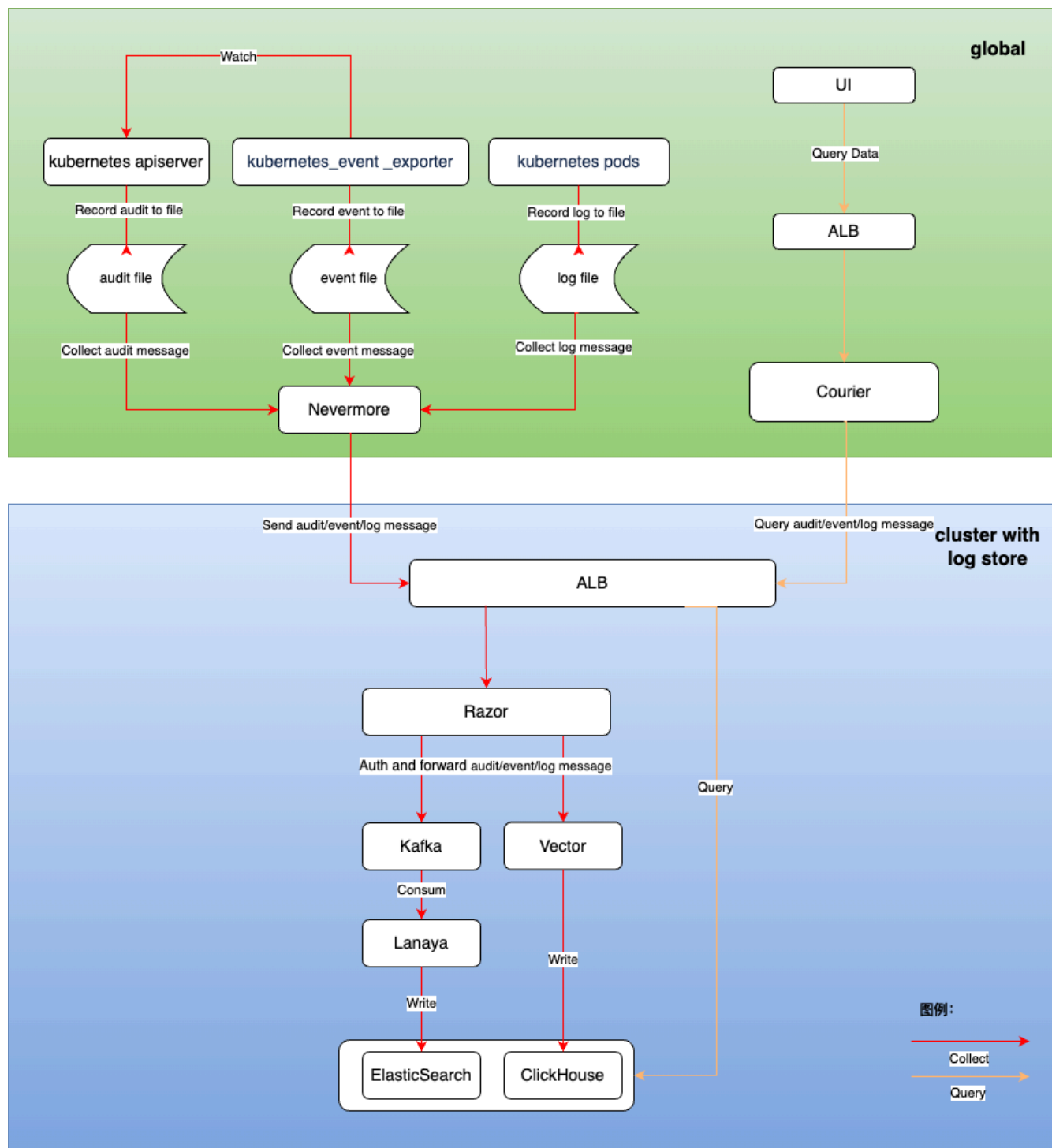
- 架构对比
- 功能比对
- 选择建议

## 日志组件容量

- ElasticSearch
- Clickhouse



# 日志模块架构



# 目录

整体架构说明

日志采集

组件安装方式

数据采集流程

日志消费及存储

Razor

Lanaya

Vector

日志可视化

---

## 整体架构说明

日志系统由以下核心功能模块组成：

### 1. 日志采集

- 基于开源组件 filebeat 提供
- 日志采集：支持采集标准输出日志、文件日志、Kubernetes 事件和审计

### 2. 日志存储

- 基于开源组件 Clickhouse 和 ElasticSearch 提供了两种不同的日志存储解决方案。
- 日志存储：支持长期存储日志文件。
- 日志存储时间管理：支持在项目级别管理日志存储时长。

### 3. 日志可视化

- 提供便捷可靠的日志查询、日志导出和日志分析能力。

## 日志采集

---

## 组件安装方式

nevermore 以 daemonset 形式安装在各个集群的 cpaas-system 命名空间下，该组件由 4 个容器组成：

名称	功能
audit	采集审计数据
event	采集事件数据
log	采集日志数据（包括标准输出和文件日志）
node-problem-detector	采集节点上的异常信息

## 数据采集流程

nevermore 采集审计、事件和日志信息后，会将数据发送到日志存储集群，经过 Razor 鉴权后，最终存放到 Elasticsearch 或 ClickHouse 中。

## 日志消费及存储

### Razor

Razor 负责鉴权及接收和转发日志消息。

- 在 Razor 接收到来自各个工作负载集群的 nevermore 发送的请求后，首先使用请求中的 Token 进行认证。如果认证失败，则拒绝请求。
- 如果安装的日志存储组件是 Elasticsearch，它会将相应的日志写入 Kafka 集群。
- 如果安装的日志存储组件是 Clickhouse，它会将相应的日志传递给 Vector，最终写入 Clickhouse。

### Lanaya

Lanaya 负责在 Elasticsearch 日志存储链路中消费和转发日志数据。

- Lanaya 订阅 Kafka 中的主题，在收到订阅消息后，会先对消息进行解压缩。
- 解压缩后，会对消息进行预处理，添加必要字段、转换字段及拆分数据。
- 最终，它会根据消息的时间和类型将消息存储到 ElasticSearch 相应的索引中。

## Vector

Vector 负责在 Clickhouse 日志存储链路中处理和转发日志数据，最终将日志存储到 Clickhouse 对应的表中。

## 日志可视化

1. 用户可以从产品 UI 界面查询审计、事件和日志的查询 URL 进行展示：
  - 日志查询 /platform/logging.alauda.io/v1
  - 事件查询 /platform/events.alauda.io/v1
  - 审计查询 /platform/audits.alauda.io/v1
2. 请求会由高级 API 组件 Courier 处理，Courier 会从日志存储集群 ElasticSearch 或 Clickhouse 查询日志数据并返回到页面。

# 日志组件选型指南

在安装集群监控时，平台提供了 Elasticsearch 和 Clickhouse 两种日志存储组件供您选择。本文将详细介绍这两种组件的特点和适用场景，帮助您做出最适合的选择。

## WARNING

- 集群日志存储组件安装时只能选择 Elasticsearch 或 Clickhouse 其中之一。
- 任意集群的日志存储组件均可选择用于日志采集，以对接存储数据。
- 当前版本 DevOps 产品不支持使用 Clickhouse 归档 Jenkins 流水线执行记录。如需使用 Jenkins 流水线功能，请谨慎选择包含 Clickhouse 插件的 ACP 日志存储。
- 当前版本的 ServiceMesh 服务网格不支持与 Clickhouse 集成。如需使用服务网格功能，请谨慎选择包含 Clickhouse 插件的 ACP 日志存储。
- 当前版本的 ACP 日志存储包含 Clickhouse 插件不支持 IPv6 单栈或 IPv6 双栈工作负载集群。

## 目录

### 架构对比

ElasticSearch 架构

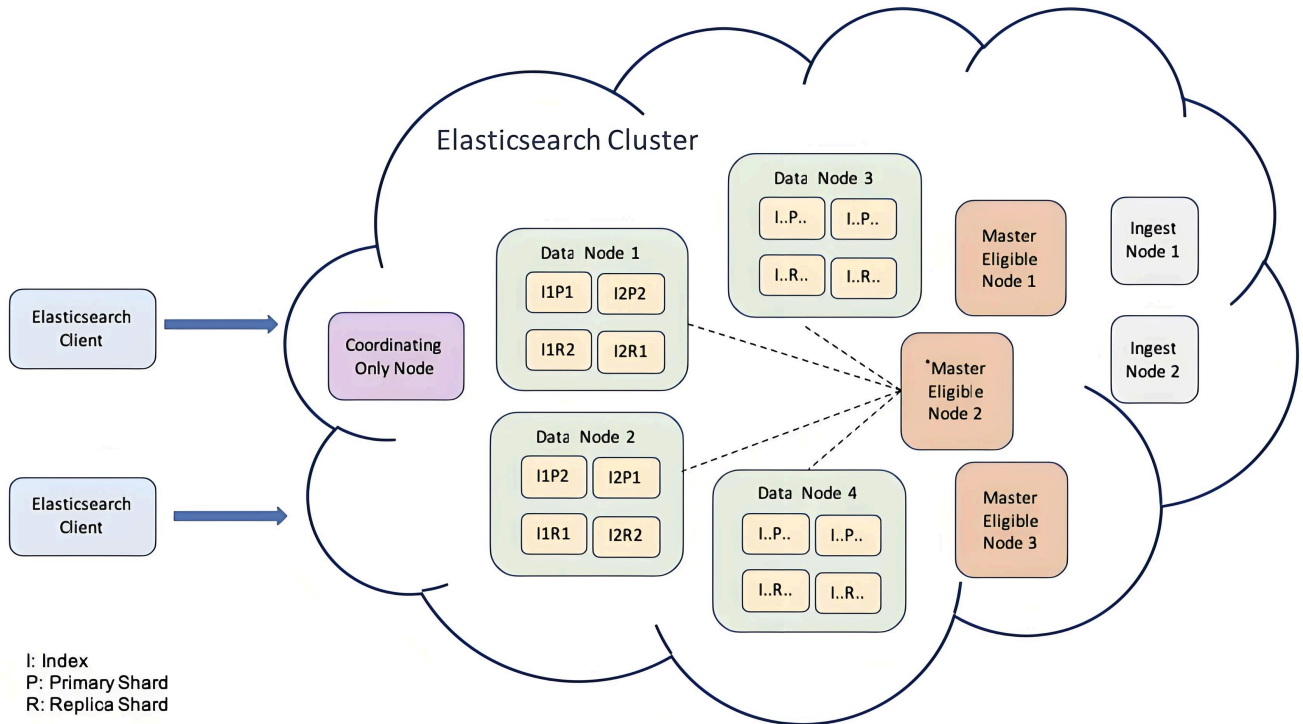
Clickhouse 架构

功能比对

选择建议

# 架构对比

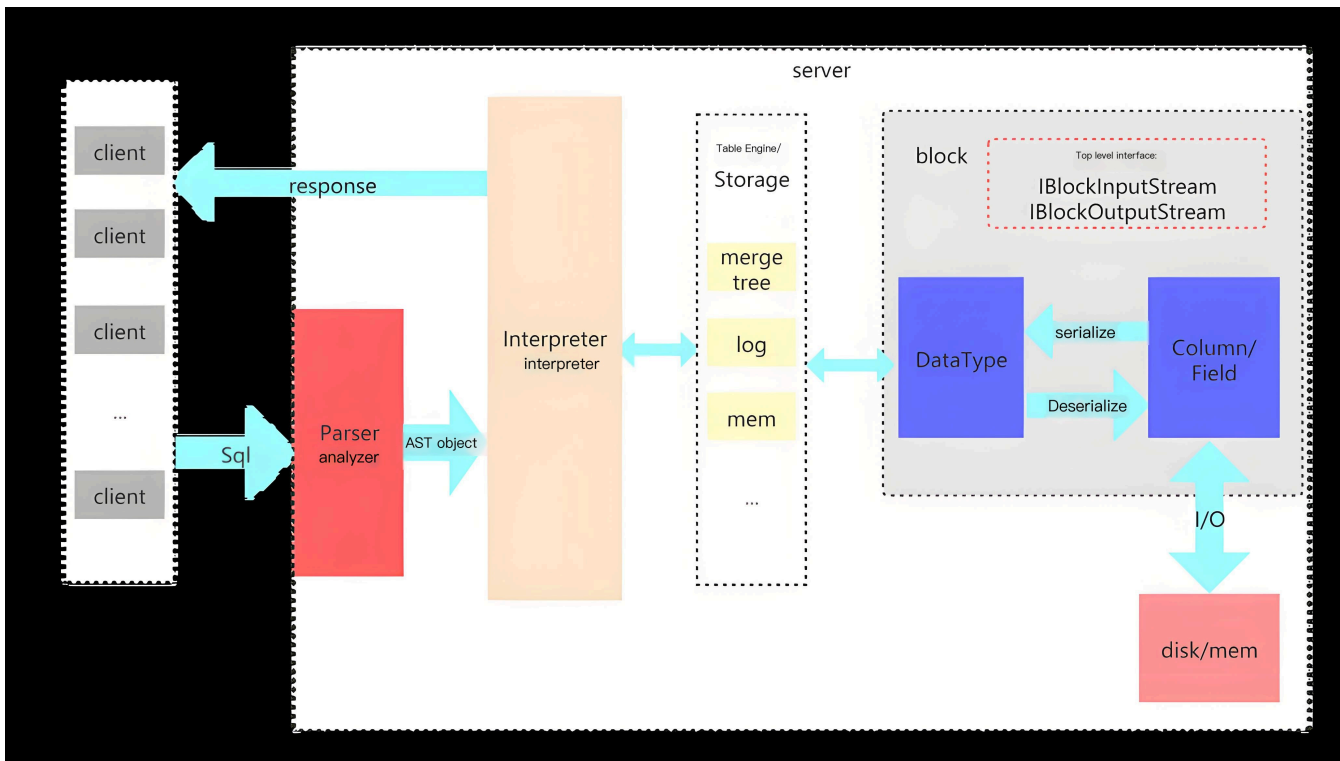
## ElasticSearch 架构



ElasticSearch 是一款开源的分布式搜索引擎，基于 Lucene 构建，专为快速全文搜索和分析而设计。其优势包括：

- 高性能搜索：支持实时搜索，能够快速处理海量数据。
- 灵活的查询能力：提供强大的查询 DSL，支持复杂的查询需求。
- 可扩展性：可根据需要轻松水平扩展，适合各类规模的应用。
- 多样的数据支持：能够处理结构化和非结构化数据，适用范围广泛。

## Clickhouse 架构



Clickhouse 是一款高性能的列式数据库，专为在线分析处理（OLAP）而设计。其优势包括：

- 快速数据处理：通过列式存储和数据压缩，支持快速查询和分析。
- 实时分析：能够处理实时数据流，适合实时数据分析场景。
- 高吞吐量：优化了大规模数据写入和查询的性能，非常适合大数据场景。
- 灵活的SQL支持：兼容标准SQL，易于上手，降低使用门槛。

## 功能比对

	Clickhouse	Elasticsearch	说明
高可用	支持	支持	
可扩展性	支持	支持	
查询	弱	强	ElasticSearch 基于 Lucene 语言提供了更强大的搜索能力，Clickhouse 仅支持 SQL 查询，搜索能

	Clickhouse	Elasticsearch	说明
体验			力有限。
资源占用	低	高	在相同的性能要求下，Clickhouse 对资源的需求低于 Elasticsearch。例如，支持 20,000 日志每秒的情况下，Elasticsearch 需要 3 个 es-master 和 7 个 es-node（2c4g+8c16g），而 Clickhouse 仅需要 3 个 2c4g 的副本。
性能	高	低	在相同的资源条件下，Clickhouse 支持的日志量远高于 Elasticsearch。
社区活跃度	中	高	Elasticsearch 社区活跃，文档丰富，而 Clickhouse 正在持续增长并完善的社区。

## 选择建议

- 若您比较习惯于 Elasticsearch 的使用方式，对 Lucene 语言有较高的依赖，建议继续选择使用包含 ElasticSearch 插件的 ACP 日志存储。
- 若您对平台的 Jenkins 流水线或服务网格功能存有依赖，建议继续选择使用包含 ElasticSearch 插件的 ACP 日志存储。
- 若您对日志组件的性能和资源消耗有较高的要求，但对于日志查询仅需基本功能，建议选择使用包含 Clickhouse 插件的 ACP 日志存储。



# 日志组件容量规划

日志存储组件负责存储由日志采集组件从平台中的一个或多个集群上采集的日志、事件和审计数据。因此，您需要提前评估日志规模，并根据本文档中的指导规划日志存储组件所需的资源。

## WARNING

- 以下数据为在实验室条件下测试得到的标准数据，仅供您在规划资源时参考。您必须确保您实际规划的资源超过以下所描述的测试资源，并且日志规模不超过相应的日志规模。
- 以下数据的磁盘配置为：`6000 iops`、`250MB/s 读写速率`、`SSD 独立挂载`。若您的实际存储资源低于测试资源，请参考更大规模的配置信息，并根据需要提供更多的CPU和内存资源。

## 目录

### ElasticSearch

小规模 3 节点 - 日志总量：6300/s

小规模 5 节点 - 日志总量：9900/s

大规模 3+5 节点 - 日志总量：25000/s

大规模 3+7 节点 - 日志总量：30000/s

### Clickhouse

单节点 - 日志总量：18000/s

三节点 - 日志总量：20000/s

六节点 - 日志总量：40000/s

九节点 - 日志总量：69000/s

## ElasticSearch

### 小规模 3 节点 - 日志总量：6300/s

组件	副本数	CPU 限制	内存限制
ElasticSearch	3	2C	4G
Kafka	3	2C	4G
Zookeeper	3	2C	4G
Lanaya	2	2C	4G
Razor	2	1C	2G

### 小规模 5 节点 - 日志总量：9900/s

组件	副本数	CPU 限制	内存限制
ElasticSearch	5	2C	4G
Kafka	3	2C	4G
Zookeeper	3	2C	4G
Lanaya	2	2C	4G
Razor	2	1C	2G

### 大规模 3+5 节点 - 日志总量：25000/s

组件	副本数	CPU限制	内存限制
ElasticSearch - Master	3	2C	4G
ElasticSearch - Data	5	8C	16G
Kafka	3	2C	4G
Zookeeper	3	2C	4G
Lanaya	2	2C	4G
Razor	2	1C	2G

## 大规模 3+7 节点 - 日志总量：30000/s

组件	副本数	CPU限制	内存限制
ElasticSearch - Master	3	2C	4G
ElasticSearch - Data	7	8C	16G
Kafka	3	2C	4G
Zookeeper	3	2C	4G
Lanaya	2	2C	4G
Razor	2	1C	2G

## Clickhouse

### 单节点 - 日志总量：18000/s

组件	副本数	CPU限制	内存限制	备注
Clickhouse	1	2C	4G	1副本1分片
Razor	1	1C	1G	-
Vector	1	2C	4G	-

### 三节点 - 日志总量：20000/s

组件	副本数	CPU限制	内存限制	备注
Clickhouse	3	2C	4G	3副本1分片
Razor	2	1C	1G	-
Vector	2	2C	4G	-

### 六节点 - 日志总量：40000/s

组件	副本数	CPU限制	内存限制	备注
Clickhouse	3	4C	8G	3副本2分片
Razor	2	1C	1G	-
Vector	2	4C	8G	-

### 九节点 - 日志总量：69000/s

组件	副本数	CPU限制	内存限制	备注
Clickhouse	9	4C	8G	3副本3分片
Razor	2	1C	1G	-

组件	副本数	CPU限制	内存限制	备注
Vector	2	4C	8G	-



# 概念

---

## 目录

### 开源组件

Filebeat

Elasticsearch

ClickHouse

Kafka

### 核心功能概念

日志采集管道

索引

分片与副本

列式存储

### 关键技术术语

Ingest Pipeline

消费者组

TTL (Time To Live)

副本因子

### 数据流模型

---

## 开源组件

## Filebeat

定位：轻量级日志采集器

说明：安装在容器节点上的开源日志采集组件，负责实时监控指定路径的日志文件。通过输入模块收集日志数据，经过处理后，通过输出模块将日志转发至 Kafka 或直接投递到存储组件。支持多行日志合并和字段过滤等预处理能力。

## Elasticsearch

定位：分布式搜索和分析引擎

说明：基于 Lucene 的全文检索引擎，以 JSON 文档格式存储日志数据，并提供近实时的搜索能力。支持动态映射以自动识别字段类型，并通过倒排索引实现快速关键词检索，适用于日志搜索和监报告警。

## ClickHouse

定位：列式分析型数据库

说明：高性能的列式存储数据库，专为 OLAP 场景设计，采用 MergeTree 引擎实现 PB 级日志数据存储。支持高速聚合查询、时间分区和数据 TTL 策略，适用于日志分析和统计报告等批量计算场景。

## Kafka

定位：分布式消息队列

说明：作为日志管道系统的消息中间件，提供高吞吐量的日志缓冲能力。当 Elasticsearch 集群出现处理瓶颈时，通过 Topic 接收 Filebeat 发送的日志数据，实现流量削峰和异步消费，确保日志采集端的稳定性。

## 核心功能概念

### 日志采集管道

说明：日志数据从产生到存储的完整链路，包含四个阶段：`采集 -> 传输 -> 缓存 -> 存储`。支持两种管道模式：

- 直写模式：Filebeat → Elasticsearch/ClickHouse
- 缓冲模式：Filebeat → Kafka → Elasticsearch

## 索引

说明：Elasticsearch中的逻辑数据分区单位，类似于数据库中的表结构。支持按时间滚动创建索引（例如：logstash-2023.10.01），并通过索引生命周期管理（ILM）实现自动化的热-温-冷分层存储。

## 分片与副本

说明：

- 分片：Elasticsearch将索引进行水平拆分的物理存储单元，支持分布式扩展。
- 副本：每个分片的复制品，提供数据的高可用性及查询负载均衡。

## 列式存储

说明：ClickHouse的核心存储机制，数据按列进行压缩存储，显著减少I/O消耗。支持以下特性：

- 向量化查询执行引擎
- 数据分区和分片
- 物化视图用于预聚合

## 关键技术术语

### Ingest Pipeline

说明：Elasticsearch中的数据预处理管道，能够在数据写入之前执行字段重命名、Grok解析和条件逻辑等ETL操作。

## 消费者组



说明：Kafka的并行消费机制，同一消费者组中的多个实例可以并行消费来自不同分区的消息，确保消息的顺序处理。

## TTL（Time To Live）

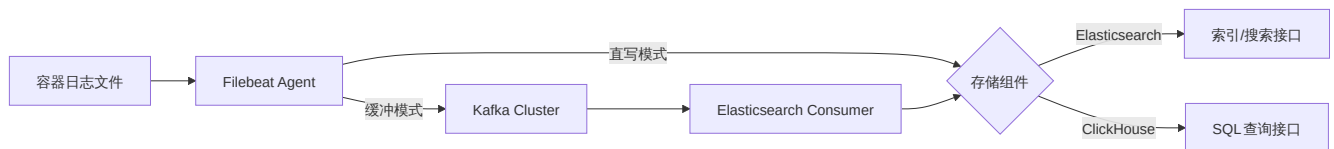
说明：数据存活时间策略，支持两种实现方式：

- Elasticsearch：通过 ILM 策略自动删除过期的索引。
- ClickHouse：通过 TTL 表达式自动删除表的分区。

## 副本因子

说明：Kafka Topic级别的数据冗余配置，定义每条消息在不同 Broker 上存在的副本数量，以增强数据可靠性。

## 数据流模型





# 操作指南

## 日志

日志查询分析

管理应用日志保留时间

配置部分应用日志停止采集

# 日志

---

## 目录

### 日志查询分析

搜索日志

导出日志数据

查看日志上下文

管理应用日志保留时间

平台管理员设置保留策略

项目管理员设置保留策略

通过 CLI 设置保留策略

配置部分应用日志停止采集

停止采集集群内所有应用日志

停止采集指定命名空间的应用日志

停止采集 Pod 日志

---

## 日志查询分析

在运维中心的日志查询分析面板中，您可以查看登录账号权限范围内的标准输出（stdout）日志，包括系统日志、产品日志、Kubernetes 日志和应用日志。通过这些日志，您可以洞察资源的运行情况。

- 系统日志：来自宿主节点的日志，如：dmesg、syslog/messages、secure 等。

- 产品日志：平台自身组件及集成到平台的第三方组件日志，如：Container-Platform、Platform-Center、DevOps、Service-Mesh 等。
- **Kubernetes** 日志：Kubernetes 容器编排相关组件的日志，以及 kubelet、kubeproxy、docker 产生的日志，如：docker、kube-apiserver、kube-controller-manager、etcd 等。
- 应用日志：业务应用产生的日志，包括文件日志和标准输出日志。

日志查询条件支持在指定时间范围内（可选时间或自定义时间）筛选日志，并通过柱状图和标准输出展示查询结果。

### WARNING

出于性能考虑，平台单次最多展示 10,000 条日志。如果平台上的日志量在某段时间内过大，请缩小查询时间范围，分阶段查询日志。

## 搜索日志

1. 在左侧导航栏点击 运维中心 > 日志 > 日志查询分析。
2. 选择指定的日志类型、查询条件，输入要检索的日志内容关键词，然后点击 搜索。

### TIP

- 不同的 日志类型 支持不同的可选查询条件。
- 可以选择或输入多个查询条件标签；不同资源类型的查询条件之间是 AND 关系。部分查询条件标签支持多选，选择后请务必按 **Enter** 键提交选项。
- 查询条件支持模糊搜索，例如查询条件 `pod = nginx` 可检索到 `nginx-1`、`nginx-2` 的日志。
- 日志内容搜索条件仅用于检索日志关键词，支持使用 **AND** 和 **OR** 参数进行关联查询。但请注意，单次查询中不要同时使用 **AND** 和 **OR** 参数。
- 柱状图展示当前查询时间范围内日志总数及不同时间点的日志数量。点击图表中的某个柱状条，可查看该柱状条与下一个柱状条时间段内的日志。

## 导出日志数据

页面最多展示 10,000 条日志。当检索日志数量过多时，可以使用日志导出功能，查看最多 100 万条日志。

1. 点击柱状图右上角的 **导出** 按钮，在弹出的导出日志对话框中配置以下参数。

- **范围**：日志导出范围，可选择 **当前页** 或 **全部结果**。
  - **当前页**：仅导出当前页查询结果，最多 1,000 条。
  - **全部结果**：导出符合当前查询条件的所有日志数据，最多 100 万条。
- **字段**：日志显示字段。可通过勾选字段名称旁的复选框选择导出日志文件中显示的字段信息。

注意：不同日志类型可选显示字段不同，请根据实际需求选择。

- **格式**：日志文件导出格式，支持 **txt** 或 **csv**。平台将以 **gzip** 压缩格式导出。

2. 点击 **导出**，浏览器将直接下载压缩文件到本地。

## 查看日志上下文

1. 双击日志内容区域，当前对话框将显示当前日志打印时间前后各 5 条日志，帮助运维人员更好地理解资源产生当前日志的原因。
2. 您可以设置日志上下文的显示字段或导出日志上下文。导出日志上下文时，无需选择 **范围**，点击 **导出** 按钮后，浏览器会直接下载日志上下文文件到本地。

## 管理应用日志保留时间





当未设置项目策略时，平台上应用日志的保留时间由应用所在集群安装的 **ACP Log Collector** 选择的 **存储集群** 上安装的 **日志存储插件** 的 **Application Log Retention Time** 决定。

您可以通过添加和管理项目日志策略，区分平台上 **应用日志** 的保留时间。





**TIP**

项目策略仅作用于特定项目下的应用日志。设置项目策略后，该项目下所有应用日志的保留时间均遵循项目策略。


## 平台管理员设置保留策略

1. 在左侧导航栏点击 运维中心 > 日志 > 策略管理。
2. 点击 添加项目策略。
3. 点击 项目 下拉框，选择一个项目。
4. 设置 日志保留时间。
  - 使用计数器两侧的  /  按钮减少/增加保留天数，或直接在计数器中输入数值。平台允许设置的保留时间范围为 1 至 30 天。
  - 若输入值为小数，将向上取整为整数；若输入值小于 1，则向上取整为 1，且  按钮不可点击；若输入值超过 30，则向下取整为 30，且  按钮不可点击。
5. 点击 添加。

## 项目管理员设置保留策略

1. 进入当前项目的项目详情页。
2. 点击日志策略字段旁的编辑按钮，在弹窗中启用日志策略。
3. 设置 日志保留时间。
  - 使用计数器两侧的  /  按钮减少/增加保留天数，或直接在计数器中输入数值。平台允许设置的保留时间范围为 1 至 30 天。
  - 若输入值为小数，将向上取整为整数；若输入值小于 1，则向上取整为 1，且  按钮不可点击；若输入值超过 30，则向下取整为 30，且  按钮不可点击。

## 通过 CLI 设置保留策略

1. 登录  集群，执行以下命令：

```
kubectl edit project <Project Name>
```

2. 按照以下示例修改 yaml，保存并提交。

```
apiVersion: auth.alauda.io/v1
kind: Project
metadata:
  annotations:
    cpaas.io/creator: mschen1@alauda.io
    cpaas.io/description: ''
    cpaas.io/display-name: ''
    cpaas.io/operator: leizhuc
    cpaas.io/project.esPolicyLastEnabledTimestamp: '2025-02-18T09:53:54Z'
    cpaas.io/updated-at: '2025-02-18T09:53:54Z'
  creationTimestamp: '2025-02-13T08:19:11Z'
  finalizers:
    - namespace
  generation: 1
  labels:
    cpaas.io/project: bookinfo
    cpaas.io/project.esIndicesKeepDays: '7' # 项目下应用日志的保留时长
    cpaas.io/project.esPolicyEnabled: 'true' # 启用项目策略
    cpaas.io/project.id: '95447321'
    cpaas.io/project.level: '1'
    cpaas.io/project.parent: ''
  name: bookinfo
### 省略未涉及修改的更多 yaml 信息。
```

## 配置部分应用日志停止采集

如果您只需查看集群内特定应用的 实时日志，但不希望存储这些日志（采集器会丢弃对应日志），可参考本节设置停止采集范围（集群、命名空间、Pod），实现对应用日志采集的精细化控制。

## 停止采集集群内所有应用日志

您可以更新集群 **ACP Log Collector** 的配置参数，关闭 应用日志 采集开关，从而统一更新该集群的日志采集范围。关闭某类日志采集开关后，将停止采集当前集群内该类日志的所有日志。

## 停止采集指定命名空间的应用日志

您可以通过给指定命名空间添加标签 `cpaas.io/log.mute=true`，关闭该命名空间的日志采集开关，从而停止采集该命名空间内所有 Pod 的标准输出日志和文件日志。

可选配置方式如下：

- 命令行方式：登录集群任一控制节点，执行以下命令更新命名空间标签。

```
kubectl label namespace <Namespace Name> cpaas.io/log.mute=true
```

- 界面操作方式：在 项目管理 视图中更新命名空间标签。
  1. 在 项目管理 视图的项目列表中，点击命名空间所在的 *项目名称*。
  2. 在左侧导航栏点击 命名空间。
  3. 点击要更新标签的 *命名空间名称*。
  4. 在 详情 标签页，点击 标签 右侧的操作按钮。
  5. 添加标签（键：`cpaas.io/log.mute`，值：`true`）或修改已有标签的值，然后点击 更新。

## 停止采集 Pod 日志

您可以通过给指定 Pod 添加标签 `cpaas.io/log.mute=true`，关闭该 Pod 的日志采集开关，从而停止采集该 Pod 的标准输出日志和文件日志。

登录集群任一控制节点，执行以下命令更新 Pod 标签。

```
kubectl label pod <Pod Name> -n <Namespace Name> cpaas.io/log.mute=true
```



注意：如果 Pod 属于计算组件（Workload），您可以更新计算组件（Deployment、StatefulSet、DaemonSet、Job、CronJob）的标签，统一更新该计算组件下所有 Pod 的标签，且 Pod 重建后标签不会丢失。

您可以通过以下方式更新计算组件标签。

1. 在 **Container Platform** 产品视图中，点击顶部导航切换到 Pod 所在的命名空间。
2. 在左侧导航栏点击 计算组件 > **Pod** 所属计算组件类型。
3. 点击要更新的计算组件右侧的操作按钮 > 更新。
4. 点击右上角的 **YAML**，切换到 YAML 编辑视图。
5. 在 **spec.template.labels** 字段下，添加 `cpaas.io/log.mute: 'true'` 标签。

示例如下：

```
spec:
  template:
    metadata:
      namespace: tuhao-test
      creationTimestamp: null
      labels:
        app: spilo
        cpaas.io/log.mute: 'true'
        cluster-name: acid-minimal-cluster
        role: exporter
        middleware.instance/name: acid-minimal-cluster
        middleware.instance/type: PostgreSQL
```

6. 点击 更新。

# 实用指南

## 如何归档日志到第三方存储

转存到外部 NFS

转存到外部 S3 存储

## 如何对接外部 **ES** 存储集群

资源准备

操作步骤

# 如何归档日志到第三方存储

目前，平台产生的日志会存储在日志存储组件中，但这些日志的保留周期较短。对于合规要求较高的企业，日志通常需要更长的保留时间以满足审计需求。此外，存储的经济性也是企业关注的重点之一。

基于以上场景，平台提供了日志归档方案，允许用户将日志转存到外部的 NFS 或对象存储。

## 目录

### 转存到外部 NFS

前提条件

创建日志同步资源

转存到外部 S3 存储

前提条件

创建日志同步资源

## 转存到外部 NFS

### 前提条件

资源	说明
NFS	需提前搭建好 NFS 服务，并确定要挂载的 NFS 路径。

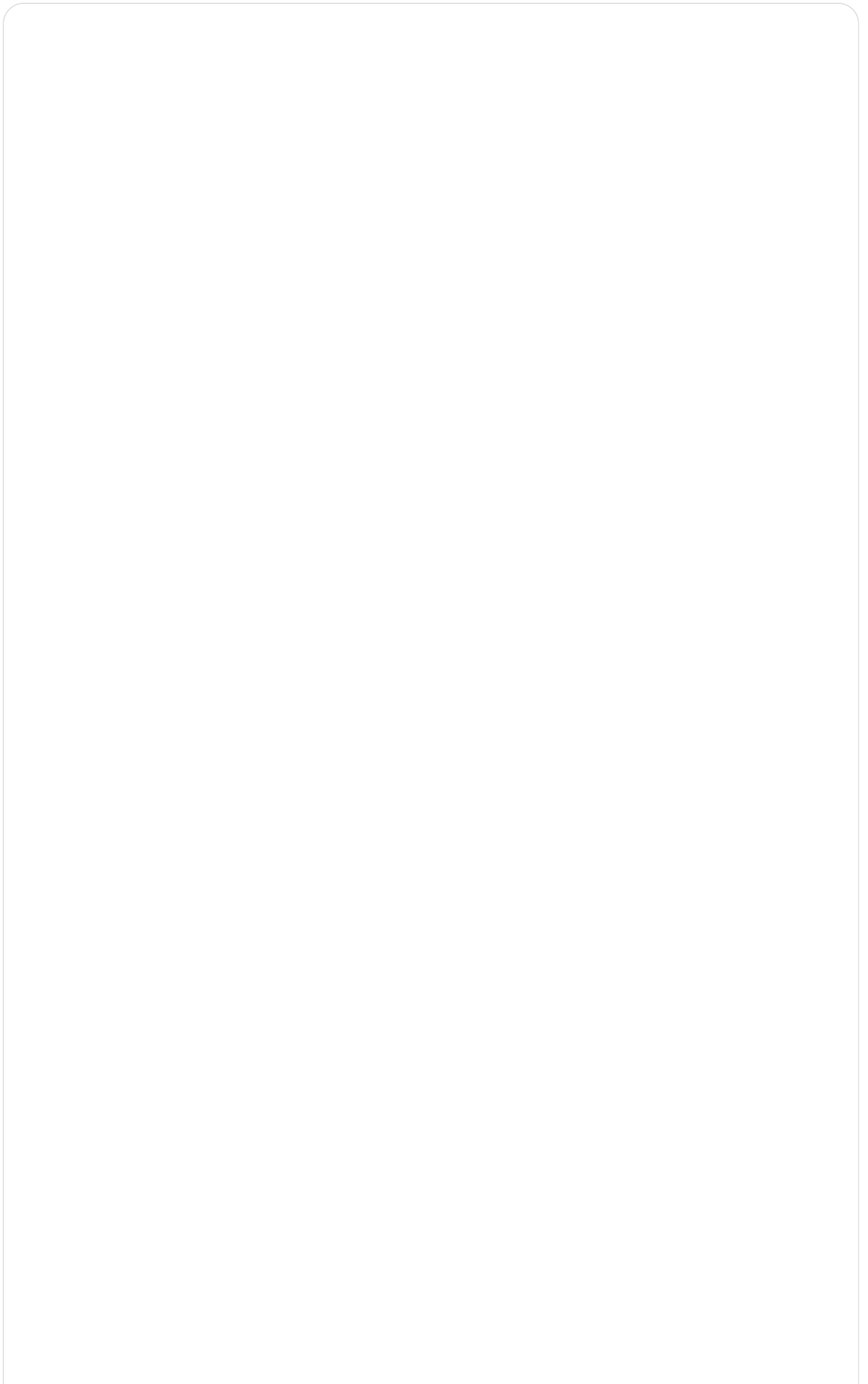
资源	说明
Kafka	需提前获取 Kafka 服务地址。
镜像地址	<div>需在 <code>global</code> 集群中使用 CLI 工具执行以下命令获取镜像地址：</div> <div>- 获取 alpine 镜像地址：<code>kubect<span>l</span> get daemonset nevermore -n cpaas-system -o jsonpath='{.spec.template.spec.initContainers[0].image}'</code></div> <div>- 获取 razor 镜像地址：<code>kubect<span>l</span> get deployment razor -n cpaas-system -o jsonpath='{.spec.template.spec.containers[0].image}'</code></div>

## 创建日志同步资源

1. 在左侧导航栏点击 集群管理 > 集群。
2. 点击要转存日志的集群右侧操作按钮 > CLI 工具。
3. 根据以下参数说明修改 YAML，修改完成后将代码粘贴到打开的 CLI 工具 命令行中，回车执行。

资源类型	字段路径	说明
ConfigMap	<code>data.export.yml.output.compression</code>	压缩日志文本；支持选项为 <b>none</b> （不压缩）、 <b>zlib</b> 、 <b>gzip</b> 。
ConfigMap	<code>data.export.yml.output.file_type</code>	导出日志文件类型；支持 txt、csv、json。
ConfigMap	<code>data.export.yml.output.max_size</code>	单个归档文件大小，单位为 MB。超过该值时，日志会根据 <code>compression</code> 字段配置自动压缩归档。

资源类型	字段路径	说明
ConfigMap	<code>data.export.yml.scopes</code>	日志转存范围； 当前支持的日志 包括：系统日 志、应用日志、 Kubernetes 日 志、产品日志。
Deployment	<code>spec.template.spec.containers[0].command[7]</code>	Kafka 服务地址。
Deployment	<code>spec.template.spec.volumes[3].hostPath.path</code>	要挂载的 NFS 路 径。
Deployment	<code>spec.template.spec.initContainers[0].image</code>	Alpine 镜像地 址。
Deployment	<code>spec.template.spec.containers[0].image</code>	Razor 镜像地 址。



```

cat << "EOF" |kubectl apply -f -
apiVersion: v1
data:
  export.yml: |
    scopes: # 日志转存范围, 默认只采集应用日志
      system: false # 系统日志
      workload: true # 应用日志
      kubernetes: false # Kubernetes 日志
      platform: false # 产品日志
    output:
      type: local
      path: /cpaas/data/logarchive
      layout: TimePrefixed
      # 单个归档文件大小, 单位 MB。超过该值时, 日志会根据 compression 字段配置自
      动压缩归档。
      max_size: 200
      compression: zlib # 可选: none (不压缩) / zlib / gzip
      file_type: txt # 可选: txt csv json
kind: ConfigMap
metadata:
  name: log-exporter-config
  namespace: cpaas-system

---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    service_name: log-exporter
  name: log-exporter
  namespace: cpaas-system
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 5
  selector:
    matchLabels:
      service_name: log-exporter
  strategy:
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
    type: RollingUpdate
-

```

```
template:
  metadata:
    creationTimestamp: null
    labels:
      app: lanaya
      cpaas.io/product: Platform-Center
      service_name: log-exporter
      version: v1
    namespace: cpaas-system
  spec:
    automountServiceAccountToken: true
    affinity:
      podAffinity: {}
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                  - key: service_name
                    operator: In
                    values:
                      - log-exporter
              topologyKey: kubernetes.io/hostname
              weight: 50
    initContainers:
      - args:
          - -ecx
          - |
            chown -R 697:697 /cpaas/data/logarchive
        command:
          - /bin/sh
        image: registry.example.cn:60080/ops/alpine:3.16 # Alpine 镜像
        imagePullPolicy: IfNotPresent
        name: chown
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 10m
            memory: 50Mi
        securityContext:
          runAsUser: 0
```

地址



```
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
      - mountPath: /cpaas/data/logarchive
        name: data
  containers:
    - command:
      - /razor
      - consumer
      - --v=1
      - --kafka-group-log=log-nfs
      - --kafka-auth-enabled=true
      - --kafka-tls-enabled=true
      - --kafka-endpoint=192.168.143.120:9092 # 根据实际环境填写
      - --database-type=file
      - --export-config=/etc/log-export/export.yml
    image: registry.example.cn:60080/ait/razor:v3.16.0-beta.3.g3d
f8e987 # Razor 镜像
    imagePullPolicy: Always
    livenessProbe:
      failureThreshold: 5
      httpGet:
        path: /metrics
        port: 8080
        scheme: HTTP
      initialDelaySeconds: 20
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 3
    name: log-export
    ports:
      - containerPort: 80
        protocol: TCP
    readinessProbe:
      failureThreshold: 5
      httpGet:
        path: /metrics
        port: 8080
        scheme: HTTP
      initialDelaySeconds: 20
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 3
    resources:
```

```
limits:
  cpu: "2"
  memory: 4Gi
requests:
  cpu: 440m
  memory: 1280Mi
securityContext:
  runAsGroup: 697
  runAsUser: 697
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
volumeMounts:
- mountPath: /etc/secrets/kafka
  name: kafka-basic-auth
  readOnly: true
- mountPath: /etc/log-export
  name: config
  readOnly: true
- mountPath: /cpaas/data/logarchive
  name: data
dnsPolicy: ClusterFirst
nodeSelector:
  kubernetes.io/os: linux
restartPolicy: Always
schedulerName: default-scheduler
securityContext:
  fsGroup: 697
serviceAccount: lanaya
serviceAccountName: lanaya
terminationGracePeriodSeconds: 10
tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
- effect: NoSchedule
  key: node-role.kubernetes.io/control-plane
  operator: Exists
- effect: NoSchedule
  key: node-role.kubernetes.io/cpaas-system
  operator: Exists
volumes:
- name: kafka-basic-auth
  secret:
    defaultMode: 420
```

```

        secretName: kafka-basic-auth
-   name: elasticsearch-basic-auth
    secret:
        defaultMode: 420
        secretName: elasticsearch-basic-auth
-   configMap:
        defaultMode: 420
        name: log-exporter-config
    name: config
-   hostPath:
        path: /cpaas/data/logarchive      # 要挂载的 NFS 路径
        type: DirectoryOrCreate
    name: data

```

EOF

4. 容器状态变为 Running 后，即可在 NFS 路径查看持续归档的日志，日志文件目录结构如下：

```
/cpaas/data/logarchive/$date/$project/$namespace-$cluster/logfile
```

## 转存到外部 S3 存储

### 前提条件

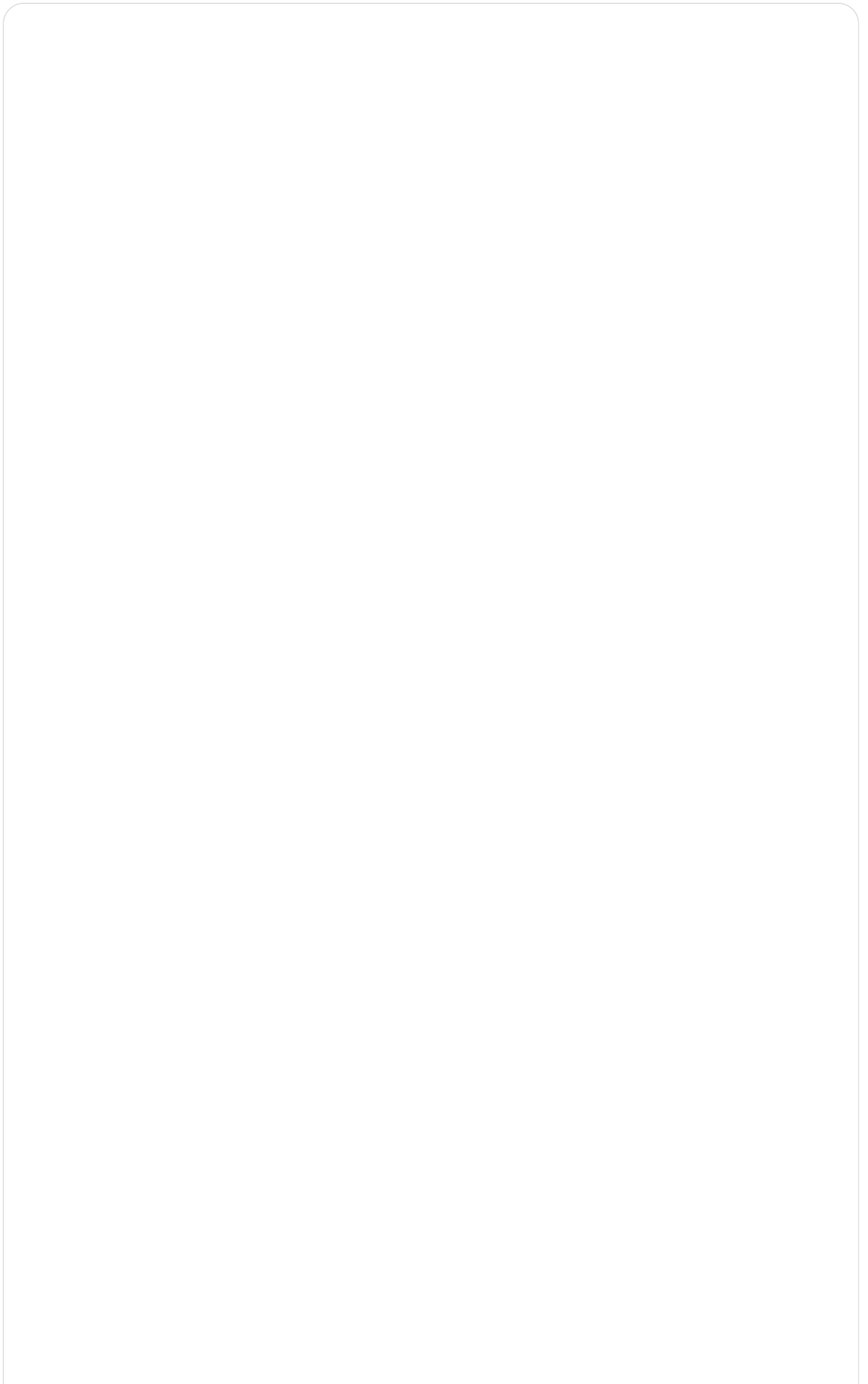
资源	说明
<b>S3 存储</b>	需提前准备好 S3 存储服务地址，并获取 <code>access_key_id</code> 和 <code>secret_access_key</code> 的值；创建存放日志的 bucket。
<b>Kafka</b>	需提前获取 Kafka 服务地址。
<b>镜像地址</b>	<p>需在 <code>global</code> 集群中使用 CLI 工具执行以下命令获取镜像地址：</p> <ul style="list-style-type: none"> <li>- 获取 alpine 镜像地址：<code>kubectl get daemonset nevermore -n cpaas-system -o jsonpath='{.spec.template.spec.initContainers[0].image}'</code></li> <li>- 获取 razor 镜像地址：<code>kubectl get deployment razor -n cpaas-system -o jsonpath='{.spec.template.spec.containers[0].image}'</code></li> </ul>

## 创建日志同步资源

1. 在左侧导航栏点击 集群管理 > 集群。
2. 点击要转存日志的集群右侧操作按钮 > **CLI 工具**。
3. 根据以下参数说明修改 YAML，修改完成后将代码粘贴到打开的 **CLI 工具** 命令行中，回车执行。

资源类型	字段路径	说明
Secret	<code>data.access_key_id</code>	对获取的 access_key_id 进行 Base64 编码。
Secret	<code>data.secret_access_key</code>	对获取的 secret_access_key 进行 Base64 编码。
ConfigMap	<code>data.export.yml.output.compression</code>	压缩日志文本；支持选项为 <b>none</b> （不压缩）、 <b>zlib</b> 、 <b>gzip</b> 。
ConfigMap	<code>data.export.yml.output.file_type</code>	导出日志文件类型；支持 txt、csv、json。
ConfigMap	<code>data.export.yml.output.max_size</code>	单个归档文件大小，单位为 MB。超过该值时，日志会根据 compression 字段配置自动压缩归档。

资源类型	字段路径	说明
ConfigMap	<code>data.export.yml.scopes</code>	日志转存范围；当前支持的日志包括：系统日志、应用日志、Kubernetes 日志、产品日志。
ConfigMap	<code>data.export.yml.output.s3.bucket_name</code>	Bucket 名称。
ConfigMap	<code>data.export.yml.output.s3.endpoint</code>	S3 存储服务地址。
ConfigMap	<code>data.export.yml.output.s3.region</code>	S3 存储服务的地域信息。
Deployment	<code>spec.template.spec.containers[0].command[7]</code>	Kafka 服务地址。
Deployment	<code>spec.template.spec.volumes[3].hostPath.path</code>	本地挂载路径，用于临时存储日志信息。日志文件同步到 S3 存储后会自动删除。
Deployment	<code>spec.template.spec.initContainers[0].image</code>	Alpine 镜像地址。
Deployment	<code>spec.template.spec.containers[0].image</code>	Razor 镜像地址。



```

cat << "EOF" |kubectl apply -f -
apiVersion: v1
type: Opaque
data:
  # 必须包含以下两个键
  access_key_id: bWluaW9hZG1pbG== # 对获取的 access_key_id 进行 Base64
编码
  secret_access_key: bWluaW9hZG1pbG== # 对获取的 secret_access_key 进行
Base64 编码
kind: Secret
metadata:
  name: log-export-s3-secret
  namespace: cpaas-system

---
apiVersion: v1
data:
  export.yml: |
    scopes: # 日志转存范围，默认只采集应用日志
      system: false # 系统日志
      workload: true # 应用日志
      kubernetes: false # Kubernetes 日志
      platform: false # 产品日志
    output:
      type: s3
      path: /cpaas/data/logarchive

      s3:
        s3forcepathstyle: true
        bucket_name: baucket_name_s3 # 填写准备好的 bucket 名称
        endpoint: http://192.168.179.86:9000 # 填写准备好的 S3 存储服务地
址
        region: "dummy" # 地域信息
        access_secret: log-export-s3-secret
        insecure: true

        layout: TimePrefixed
        # 单个归档文件大小，单位 MB。超过该值时，日志会根据 compression 字段配置自
动压缩归档。
        max_size: 200
        compression: zlib # 可选：none（不压缩）/ zli
b / gzip
        file_type: txt # 可选：txt、csv、json
  . . .

```

```
kind: ConfigMap
metadata:
  name: log-exporter-config
  namespace: cpaas-system

---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    service_name: log-exporter
  name: log-exporter
  namespace: cpaas-system
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 5
  selector:
    matchLabels:
      service_name: log-exporter
  strategy:
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: lanaya
        cpaas.io/product: Platform-Center
        service_name: log-exporter
        version: v1
      namespace: cpaas-system
    spec:
      affinity:
        podAffinity: {}
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - podAffinityTerm:
                labelSelector:
                  matchExpressions:
                    - key: service_name
                      operator: In
```



地址

ka 服务地址

f8e987 # Razor 镜像

```

      values:
        - log-exporter
      topologyKey: kubernetes.io/hostname
      weight: 50
    initContainers:
      - args:
          - -ecx
          - |
            chown -R 697:697 /cpaas/data/logarchive
        command:
          - /bin/sh
        image: registry.example.cn:60080/ops/alpine:3.16 # Alpine 镜像
        imagePullPolicy: IfNotPresent
        name: chown
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 10m
            memory: 50Mi
        securityContext:
          runAsUser: 0
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
          - mountPath: /cpaas/data/logarchive
            name: data
    containers:
      - command:
          - /razor
          - consumer
          - --v=1
          - --kafka-group-log=log-s3
          - --kafka-auth-enabled=true
          - --kafka-tls-enabled=true
          - --kafka-endpoint=192.168.179.86:9092 # 根据实际环境填写 Kaf
            ka 服务地址
          - --database-type=file
          - --export-config=/etc/log-export/export.yml
        image: registry.example.cn:60080/ait/razor:v3.16.0-beta.3.g3d
        f8e987 # Razor 镜像
        imagePullPolicy: Always

```

```
livenessProbe:
  failureThreshold: 5
  httpGet:
    path: /metrics
    port: 8080
    scheme: HTTP
  initialDelaySeconds: 20
  periodSeconds: 10
  successThreshold: 1
  timeoutSeconds: 3
name: log-export
ports:
  - containerPort: 80
    protocol: TCP
readinessProbe:
  failureThreshold: 5
  httpGet:
    path: /metrics
    port: 8080
    scheme: HTTP
  initialDelaySeconds: 20
  periodSeconds: 10
  successThreshold: 1
  timeoutSeconds: 3
resources:
  limits:
    cpu: "2"
    memory: 4Gi
  requests:
    cpu: 440m
    memory: 1280Mi
securityContext:
  runAsGroup: 697
  runAsUser: 697
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
volumeMounts:
  - mountPath: /etc/secrets/kafka
    name: kafka-basic-auth
    readOnly: true
  - mountPath: /etc/log-export
    name: config
    readOnly: true
  - mountPath: /cpaas/data/logarchive
```

```

      name: data
dnsPolicy: ClusterFirst
nodeSelector:
  kubernetes.io/os: linux
restartPolicy: Always
schedulerName: default-scheduler
securityContext:
  fsGroup: 697
serviceAccount: lanaya
serviceAccountName: lanaya
terminationGracePeriodSeconds: 10
tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
- effect: NoSchedule
  key: node-role.kubernetes.io/control-plane
  operator: Exists
- effect: NoSchedule
  key: node-role.kubernetes.io/cpaas-system
  operator: Exists
volumes:
- name: kafka-basic-auth
  secret:
    defaultMode: 420
    secretName: kafka-basic-auth
- name: elasticsearch-basic-auth
  secret:
    defaultMode: 420
    secretName: elasticsearch-basic-auth
- configMap:
    defaultMode: 420
    name: log-exporter-config
  name: config
- hostPath:
    path: /cpaas/data/logarchive      # 日志本地临时存储地址
    type: DirectoryOrCreate
  name: data

```

EOF

# 如何对接外部 ES 存储集群

您可以通过编写 YAML 配置文件的方式，对接外部的 Elasticsearch 或 Kafka 集群。根据您的业务需求，您可以选择仅对接外部的 Elasticsearch 集群（同时在当前集群安装 Kafka），或同时对接外部的 Elasticsearch 和 Kafka 集群。

## TIP

对接外部 Elasticsearch 支持的版本如下：

- Elasticsearch 6.x 支持版本 6.6 - 6.8 ；
- Elasticsearch 7.x 支持版本 7.0 - 7.10.2，推荐使用 7.10.2。

## 目录

资源准备

操作步骤

## 资源准备

在进行对接之前，您需要准备必需的凭证信息。

1. 在左侧导航栏中，单击 集群管理 > 资源管理，然后切换到需要安装插件的集群。
2. 单击 创建资源对象，根据代码注释修改参数后填写至代码框中。

- 对接外部 Elasticsearch 所需的凭据：

```

apiVersion: v1
type: Opaque
data:
  password: dEdWQVduSX5kUW1mc21acg== # 必须经过 base64 编码。参考命令：echo -n <password_value>| base64
  username: YWRtaW4= # 必须经过 base64 编码。参考命令：echo -n <username_value>| base64
kind: Secret
metadata:
  name: elasticsearch-basic-auth # 凭据名称，确保在日志存储 YAML 中 elasticsearch.basicAuthSecretName 的值与该参数一致。
  namespace: cpaas-system # Elasticsearch 组件所在的命名空间，一般为 cpaas-system。

```

- 如果您需要使用外部 Kafka 集群，则还需要创建用于对接外部 Kafka 集群的凭据：

```

apiVersion: v1
type: Opaque
data:
  password: dEdWQVduSX5kUW1mc21acg== # 必须经过 base64 编码。参考命令：echo -n <password_value>| base64
  username: YWRtaW4= # 必须经过 base64 编码。参考命令：echo -n <username_value>| base64
kind: Secret
metadata:
  name: kafka-basic-auth # 凭据名称，确保在日志存储 YAML 中 kafka.basicAuthSecretName 的值与该参数一致。
  namespace: cpaas-system # Kafka 组件所在的命名空间，一般为 cpaas-system。

```

3. 单击 创建。

## 操作步骤

1. 在左侧导航栏中，单击 应用商店 > 插件管理。
2. 在顶部导航中，选择待安装的 ACP Log Storage with Elasticsearch 插件的 集群名称。

3. 单击 **ACP Log Storage with Elasticsearch** 右侧的操作按钮 > 安装。
  4. 开启 对接外部 **Elasticsearch** 开关，配置 YAML 文件，配置示例和参数说明如下：
- 对接外部 Elasticsearch 集群，同时在当前集群安装 Kafka：

```

elasticsearch:
  install: false
  address: http://fake:9200 # 外部 ES 访问地址，例如：http://192.168.143.252:11780/es_proxy
  basicAuthSecretName: elasticsearch-basic-auth # 对接外部 Elasticsearch 所需的凭据
storageClassConfig:
  type: "LocalVolume" # 默认值为 LocalVolume，选项为 "LocalVolume" 或 "StorageClass"。
kafka:
  auth: true # 是否启用认证。
  k8sNodes:
    - log1 # 节点名称，从 kubectl get nodes 获取。
    - log2
    - log3
  storageSize: 10 # 存储大小，单位 Gi，默认值为 10 Gi。

```

- 对接外部 Elasticsearch 集群和外部 Kafka 集群：

```

elasticsearch:
  install: false
  address: http://fake:9200 # 外部 ES 访问地址，例如：http://192.168.143.252:11780/es_proxy
  basicAuthSecretName: elasticsearch-basic-auth # 对接外部 Elasticsearch 所需的凭据
kafka:
  auth: true # 是否启用认证。
  install: false
  basicAuthSecretName: kafka-basic-auth # 对接外部 Kafka 集群所需的凭据
  address: 192.168.130.169:9092,192.168.130.187:9092,192.168.130.193:9092 # Kafka 访问地址，用英文逗号分隔。

```



# 权限说明

日志模块中可用的权限点以及平台内置角色所具备的权限如下：

功能	操作	平台管理员	平台审计人员	项目管理员	命名空间管理员	开发人员
日志 aiops-logs	查看	✓	✓	✓	✓	✓
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×
日志导出 aiops-archivelogs	查看	✓	✓	×	×	×
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×



# 事件

## 介绍

Module Overview

Functionality Overview

Use Cases

Usage Limitations

## Events

操作流程

事件概览

## 权限说明

# 介绍

---

## 目录

### Module Overview

Functionality Overview

Use Cases

Usage Limitations

---

## Module Overview

该平台集成了 Kubernetes 事件，记录 Kubernetes 资源的重要状态变化及各种运行状态变更。同时提供存储、查询和可视化能力。当集群、节点或 Pod 等资源出现异常时，用户可以通过分析事件来确定具体原因。

基于事件中识别的根因，用户可以为工作负载[创建告警策略](#)。当关键事件数量达到告警阈值时，可自动触发告警，通知相关人员及时干预，从而降低平台的运维风险。

## Functionality Overview

事件模块主要提供以下功能：

### 事件采集与持久化

- 自动采集：模块会自动采集 Kubernetes 集群中发生的所有事件，包括 Pod 创建、删除、调度失败等。
-

- 持久化存储：采集到的事件将被持久化存储，确保用户可以根据需要回溯历史事件。

### 事件查询

- 灵活查询：用户可以根据多种条件（如事件类型、命名空间、资源名称等）查询事件，快速定位问题。
- 时间范围过滤：支持按时间范围查询事件，方便用户查看特定时间段内的集群活动。

### 事件汇总与展示

- 事件汇总：模块会对事件进行汇总并生成统计信息，帮助用户了解集群的整体状态。

## Use Cases

事件模块适用于以下场景：

- 集群监控：通过实时监控 Kubernetes 事件，用户可以及时发现集群异常。
- 故障排查：当集群出现问题时，用户可以通过查询事件日志快速定位根因。
- 性能优化：通过分析事件数据，用户可以了解集群中的资源使用情况，优化资源分配。

## Usage Limitations

该功能依赖于日志系统。请确保平台内已安装 ACP Log Collector 和 ACP Log Storage 插件。

# Events

## 目录

[操作流程](#)[事件概览](#)

## 操作流程

1. 点击左侧导航栏中的 **Operations Center > Events**。

提示：通过顶部导航栏的下拉选择框切换集群以查看对应的事件。

## 事件概览

事件页面默认展示最近 30 分钟内发生的重要事件概览（您也可以选择或自定义时间范围），以及资源事件记录。

- 重要事件概览：该卡片展示重要事件的原因及在所选时间范围内发生该事件的资源数量。
  - 注意：当同一资源多次发生该类型事件时，资源数量不会累加。
  - 例如：如果节点重启事件的资源数量为 20，表示在所选时间范围内，有 20 个资源发生了该事件，同一资源可能多次发生。

- 资源事件记录：在重要事件概览区域下方，展示所选时间范围内符合查询条件的所有事件记录。您可以点击重要事件卡片筛选对应类型的事件，也可以展开视图并输入查询条件进行搜索。查询条件如下：
  - 资源类型：发生事件的 Kubernetes 资源类型，例如 Pod。
  - **Namespace**：发生事件的 Kubernetes 资源所在的命名空间。
  - 事件原因：事件发生的原因。
  - 事件级别：事件的重要程度，如 Warning。
  - 资源名称：发生事件的 Kubernetes 资源名称，可选择或输入多个名称。

#### TIP

- 点击事件记录中资源名称旁的视图图标，可在弹出的 **Event Details** 对话框中查看事件详细信息。
- 事件原因左侧图标的颜色表示事件级别。绿色图标表示该事件级别为 **Normal**，此事件可忽略；橙色图标表示该事件级别为 **Warning**，说明资源存在异常，应关注该事件以防止事故发生。

# 权限说明

事件模块可用的权限点及平台内置角色的权限如下：

功能	操作	平台管理员	平台审计人员	项目管理员	命名空间管理员	开发人员
事件 aiops- events	查看	✓	✓	✓	✓	✓
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×

# 巡检

## 介绍

### 介绍

模块介绍

模块优势

模块应用场景

使用限制

## 架构

### 架构

巡检

组件健康状态

## 操作指南

### 巡检

执行巡检

### Component Health Status

巡检配置

Procedures to Operate

巡检报告说明

# 权限说明

权限说明



# 介绍

---

## 目录

### 模块介绍

模块优势

模块应用场景

使用限制

---

## 模块介绍

为企业客户降低人工巡检的成本，平台的基础巡检功能基于为企业客户执行人工巡检的经验设计。能够帮助企业客户实时了解平台上的所有业务资源的运行情况，及时感知异常，降低业务风险。

- 支持在线执行巡检任务，包括平台上所有集群、节点、容器组、证书资源的资源风险巡检以及常规资源的用量巡检，实时获取巡检进度；
- 巡检结束后，可视化展示巡检结果，包括资源风险和用量信息；
- 支持下载 PDF 或 Excel 格式的巡检报告；
- 为保障客户数据安全，仅允许具有相关访问权限的用户使用巡检功能。

## 模块优势

- 全面覆盖：支持对平台上所有关键资源的巡检，确保无遗漏。
-

- 实时监控：用户可实时查看巡检进度，及时了解资源状态。
- 可视化展示：通过直观的可视化界面展示巡检结果，便于快速识别问题。
- 灵活报告：支持下载 PDF 或 Excel 格式的巡检报告，满足不同用户的需求。

## 模块应用场景

- 日常运维：定期执行巡检任务，确保平台资源的正常运行和安全性。
- 故障排查：在出现问题时，通过巡检功能快速定位资源风险或用量异常。
- 合规审计：下载巡检报告，用于合规审计和内部审查，确保平台符合相关标准和规范。
- 资源优化：通过分析资源用量信息，识别资源浪费或不足，优化资源配置。

## 使用限制

- 平台部分巡检项依赖集群安装了监控组件，请提前确保各集群已安装了 ACP Monitoring with Prometheus 插件或 ACP Monitoring with VictoriaMetrics 插件。
- 平台巡检支持通过邮件发送巡检结果，配置前请提前确保平台已经完成了邮件通知服务器配置。

通过容器平台的巡检功能，用户可以更加高效地管理和维护容器环境，提升系统的稳定性和安全性。



# 架构

---

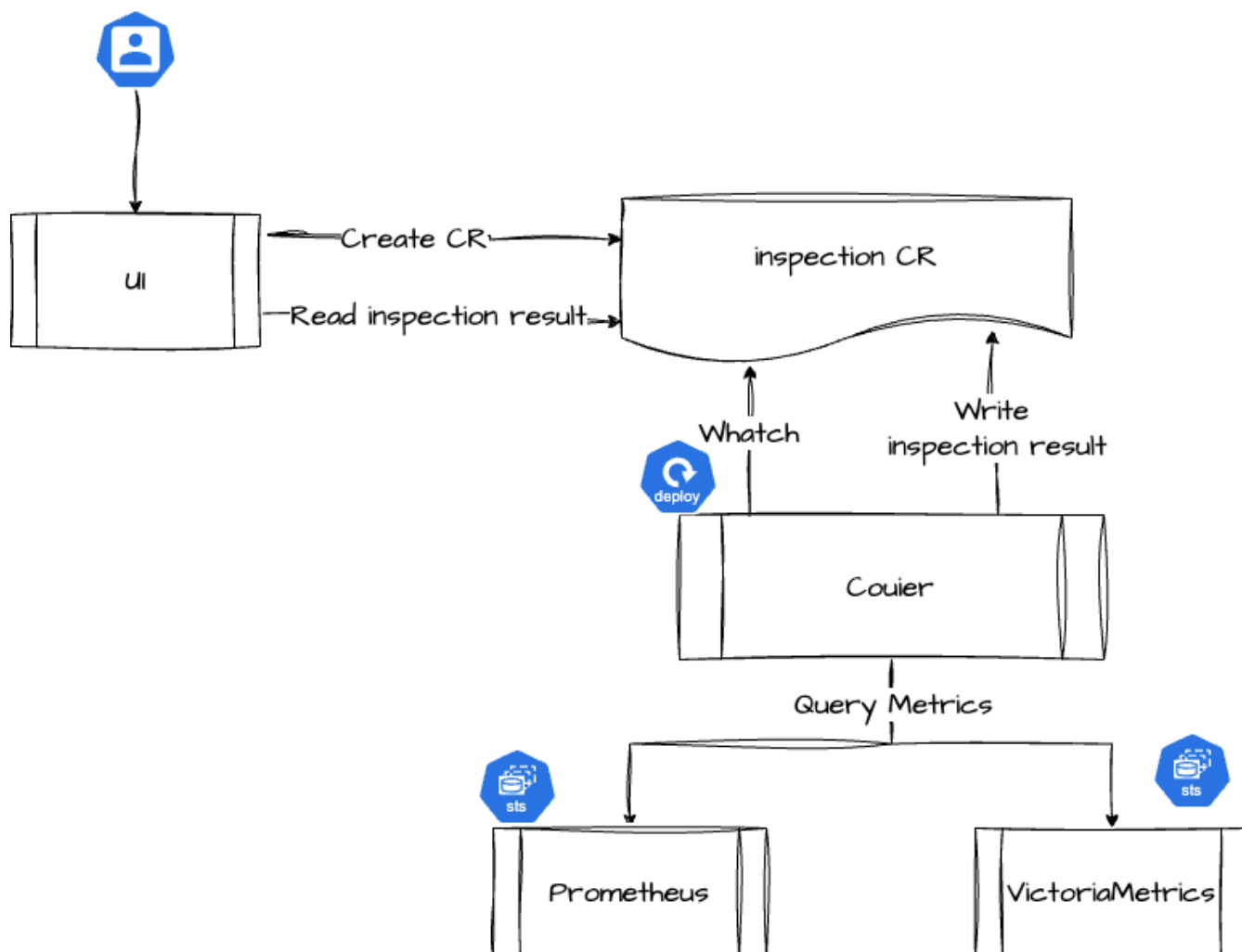
## 目录

| [巡检](#)

组件健康状态

---

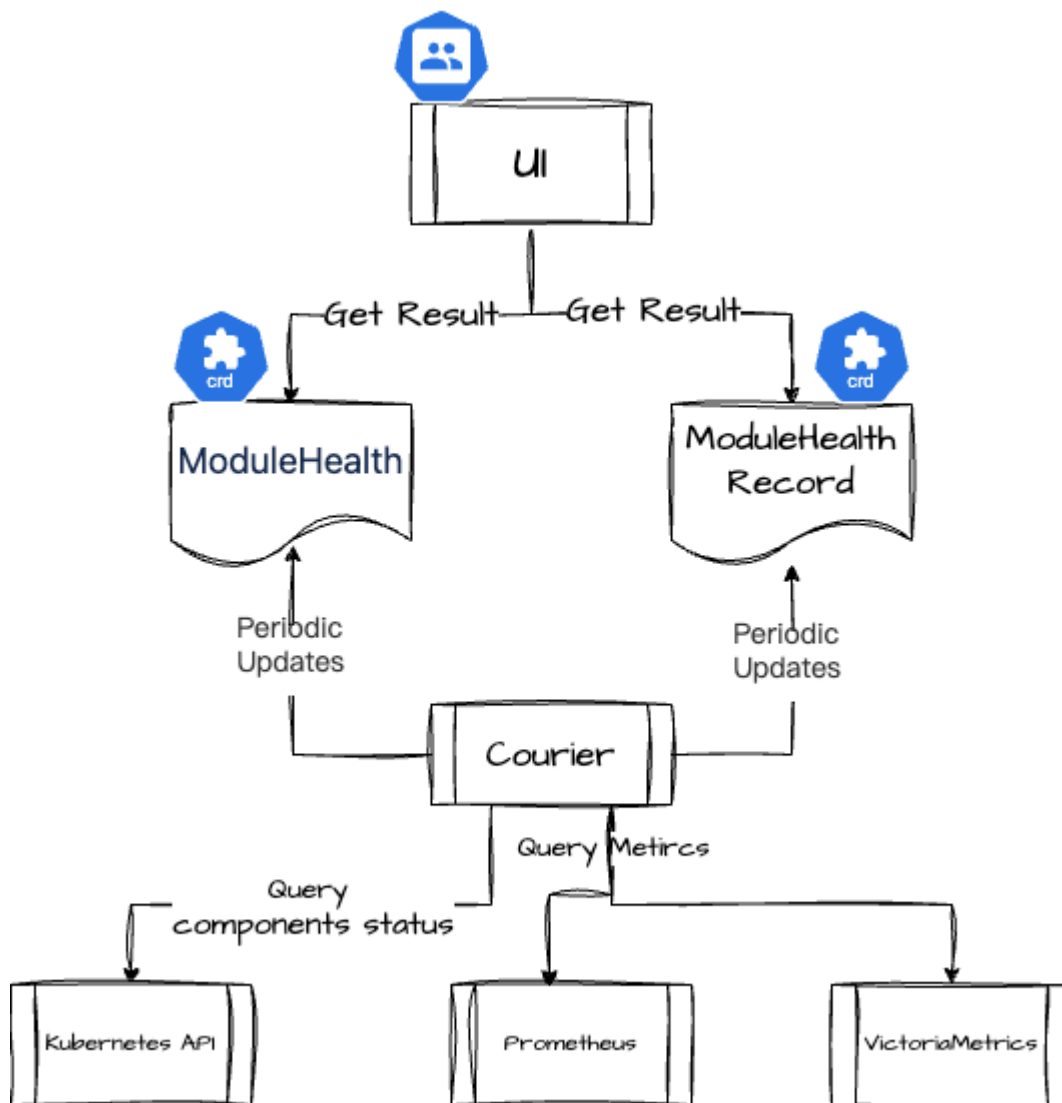
## 巡检



巡检模块由平台组件 Courier 和监控组件共同提供，涉及的业务流程如下：

- 创建巡检任务：平台向 `global` 集群提交一个巡检类型的 CR。
- 执行巡检任务：Courier 组件监测巡检类型 CR 的生成，并向各集群的监控组件查询与巡检相关的各种指标数据。
- 写入巡检结果：在完成对各巡检项的评估后，Courier 组件会将巡检结果写回到对应的巡检 CR 中。
- 查看巡检结果：用户可以通过平台查看巡检任务的状态和结果，数据将从对应的巡检 CR 中获取。

## 组件健康状态



组件健康状态由平台组件 Courier 和监控组件共同提供，涉及的业务流程如下：

- 预定义组件监测列表：平台在 `global` 集群中预定义了两种 CRD 用于定义需要监测的组件清单和监测方式：
  - ModuleHealth：定义需要监测的组件及监测方式。
  - ModuleHealthRecord：定义各集群中对应组件的监测结果。
- 定期监测组件状态：Courier 会监视 ModuleHealth，检查指定功能，然后将检查结果写入 ModuleHealth 和 ModuleHealthRecord 的 CR 资源中。
- 组件状态判断：Courier 会请求 Kubernetes 和监控组件的数据，以确定组件的实际状态及存在的问题。
  - Kubernetes：检查组件是否已安装，以及组件副本数是否正常。
  - Prometheus / VictoriaMetrics：根据各组件提供的指标，查询并判断组件是否能正常提供服务。

- 查看组件健康状态：用户可以通过平台查看各组件的健康状态，数据将从对应的 ModuleHealth 和 ModuleHealthRecord 的 CR 资源中获取。



# 操作指南

## 巡检

执行巡检

巡检配置

巡检报告说明

## Component Health Status

Procedures to Operate

# 巡检

## 目录

### 执行巡检

#### 巡检配置

#### 巡检报告说明

#### 最近一次巡检

#### 资源风险巡检

#### 资源用量巡检

## 执行巡检

1. 在左侧导航栏中，单击 运维中心 > 巡检 > 基础巡检。

提示：巡检页面展示的巡检数据信息为最近一次巡检的结果。巡检过程中，可实时查看完成巡检的资源数据。

2. 在基础巡检页面，支持以下操作：

- 执行巡检：单击页面右上角的 巡检 按钮，即可对平台进行巡检。
- 下载巡检报告：单击页面右上角的 下载报告 按钮，在弹出的对话框中选择报告格式（PDF 和 Excel）后单击下载，即可将相应格式的报告下载至本地。
  - PDF 格式巡检报告内容不包含资源风险详情页面数据；



- Excel 格式巡检报告内容为巡检的全部数据；
- 支持同时下载两种格式报告。

## 巡检配置

巡检配置	描述
定时巡检	自动触发任务执行的定时规则，支持输入 Crontab 表达式。 提示：单击输入框，可展开平台预设的 触发规则模板， 选择适合的模板并简单修改后即可快速设置触发规则。
巡检记录保留	保留巡检记录的条数。
邮件通知	选择邮件通知联系人。 注意：通知联系人需配置邮箱。
巡检报告名称	平台内置的巡检通知模板将使用该名称通知联系人。
巡检配置项	在平台默认的证书、集群主机和容器组巡检项中，根据需求修改预警阈值或关闭巡检项。

## 巡检报告说明

### 最近一次巡检

在 最近一次巡检 信息区域，可查看最近一次巡检的相关信息：

- 巡检时间：最近一次巡检的开始时间和结束时间。
- 巡检资源总数：最近一次巡检总共巡检的资源（集群、节点、容器组、证书）总数。
- 风险：存在风险的资源个数。包括发生 故障 和 预警 的资源个数。

## 资源风险巡检

在 资源风险巡检 页面，可查看平台上 global 集群、自建集群、接入集群以及所有集群下节点、容器组、证书的风险信息总览。

单击对应类型资源（集群、节点、容器组、证书）卡片上的 风险详情 按钮，即可进入对应类型资源的风险详情页面。在详情页面，可查看资源的最近一次巡检信息，以及存在故障和预警的资源列表。

- 单击资源名称，可跳转资源详情页面。
- 单击列表 名称 字段右侧的展开按钮可展开故障、预警的判断条件和原因。

资源的风险状态（故障、预警）判断条件说明参见下表。

说明：用于判断每类资源故障、预警的条件包含多条，当资源的巡检数据匹配到判断条件中任一条时，即作为一条风险数据。

资源类型	巡检范围	故障判断条件	预警判断条件
集群	<ul style="list-style-type: none"> <li>- global 集群</li> <li>- 自建集群</li> <li>- 接入集群</li> </ul>	<ul style="list-style-type: none"> <li>- 集群状态为 异常；</li> <li>- apiserver 连接异常</li> </ul>	<ul style="list-style-type: none"> <li>- 集群规模（节点/容器组/mrtrics 数量）增大后，监控组件资源配置未更新。</li> <li>- 日志数据量、日志采集频率增大后，日志组件资源配置未更新。</li> <li>- 集群的 CPU 使用率大于 60%；</li> <li>- 集群的内存使用率大于 60%；</li> <li>- 集群的 ETCD 组件的任一容器组处于非 Running 状态；</li> <li>- 集群中任一主机处于非 Ready 状态；</li> <li>- 集群内任意 2 个节点的系统时间差超过 40S；</li> <li>- 集群的 CPU 请求率（实际请求值 / 总额）大于 60%；</li> <li>- 集群的内存请求率（实际请求值 / 总额）大于 80%；</li> <li>- 集群未安装监控组件；</li> <li>- 集群的监控组件异常；</li> <li>- 集群中的 kube-controller-manager 组</li> </ul>

资源类型	巡检范围	故障判断条件	预警判断条件
			件的任一容器组处于非 Running 状态； - 集群中的 <b>kube-scheduler</b> 组件的任一容器组处于非 Running 状态； - 集群中的 <b>kube-apiserver</b> 组件的任一容器组处于非 Running 状态。
节点	- 所有控制节点 - 所有计算节点	- 节点状态为 异常； - 节点上的 <b>node-exporter</b> 组件的容器组处于非 Running 状态； - 节点上的 <b>kubelet</b> 组件的容器组处于非 Running 状态。	- 节点内 inode free 小于 1000 - 节点的 CPU 使用率大于 60%； - 节点的内存使用率大于 60%； - 节点目录的磁盘空间使用率大于 60%； - 节点的系统负载大于 200% 且运行时间大于 15 分钟； - 过去 1 天内，至少发生过一次 NodeDeadlock（节点死锁）事件； - 过去 1 天内，至少发生过一次 NodeOOM（节点上内存溢出）事件； - 过去 1 天内，至少发生过一次 NodeTaskHung（节点上任务被挂起）事件； - 过去 1 天内，至少发生过一次 NodeCorruptDockerImage（节点上有损坏的 Docker 镜像）事件。
容器组	所有容器组	- 容器组状态为 错误； - 容器组处于启动状态的时长超过 5 分钟。	- Pod 的 CPU 使用率大于 80%； - Pod 的内存使用率大于 80%； - Pod 在过去 5 分钟内的重启次数大于等于 1 次。
证书	- Certmanager 证书 - Kubernetes 证书	证书状态为 过期。	证书的有效期小于 29 天。

资源用量巡检

单击 [资源用量巡检](#) 页签，进入 [资源用量巡检](#) 页面。

在 [资源用量巡检](#) 页面，可查看平台上 global 集群、接入集群、自建集群的 CPU、内存、磁盘总量、用量、使用率，以及平台上集群、节点、容器组、项目等资源的个数。

- 资源使用量统计：可查看 global 集群、接入集群和自建集群的 CPU、内存、磁盘总量和总使用率。
- 平台资源数量：可查看平台上正在运行的资源的个数。

# Component Health Status

平台健康状态页面展示了已安装在平台上的功能的健康状态统计数据。当您的账号拥有与平台相关的管理或审计权限时，还可以查看特定功能的详细健康数据，包括：未安装该功能的集群列表、已安装该功能的集群健康状态，以及与该功能相关联的集群内组件的检测数据。这有助于您快速定位问题，提高平台的运维效率。

## 目录

[Procedures to Operate](#)

## Procedures to Operate

1. 进入已安装产品的视图页面或平台中心（平台管理、项目管理、运维中心）。
2. 点击导航栏右上角的问号按钮 > **Platform Health Status**。
3. 查看功能卡片；功能卡片展示该功能的健康状态信息。如果功能组件存在异常，会在卡片上以 `fault` 形式体现。
4. 点击功能卡片上的健康/故障值，展开页面右侧的详细健康状态页面，可查看故障组件的详细信息。

# 权限说明

巡检模块中的可用权限和平台内置角色的权限如下：

功能	操作	平台管理员	平台审计人员	项目管理员	命名空间管理员	开发人员
巡检 aiops-inspections	查看	✓	✓	×	×	×
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×