Menu

可观测性

概览

概览

监控

介绍

安装

Overview

安装准备

通过控制台安装 ACP Monitoring with Prometheus 插件

通过 YAML 安装 ACP Monitoring with Prometheus 插件

通过控制台安装 ACP Monitoring with VictoriaMetrics 插件

通过 YAML 安装 ACP Monitoring with VictoriaMetrics 插件

架构

核心概念

Monitoring

Alarms

Notifications

Monitoring Dashboard

操作指南

实用指南

分布式追踪

介绍

使用限制

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

核心组件

数据流

核心概念

Telemetry

OpenTelemetry

Span

Trace

Instrumentation

OpenTelemetry Collector

Jaeger

操作指南

实用指南

故障排除

日志

关于 **Logging Service**

事件

介绍

使用限制

Events

操作流程

事件概览

检查

介绍

使用限制

架构

Inspection

Component Health Status

操作指南

概览

Observability 模块是 ACP 平台的核心功能，提供面向云原生应用的全面监控和可观测性能力。

该模块集成了四个关键的可观测性支柱：

- **Synthetic monitoring (probe)** 用于主动的端点测试
- **Centralized logging** 实现统一的日志管理和分析
- **Real-time monitoring** 用于指标收集和告警
- **Distributed tracing** 实现跨微服务的端到端请求追踪

通过将这些能力整合到单一平台中，帮助组织实现对应用性能的全面可视化，快速诊断问题，确保系统可靠性，并优化整个技术栈中的用户体验。

Menu

监控

介绍

安装

架构

Monitoring Module Architecture

Overall Architecture Explanation

Monitoring System

Alerting System

Notification System

Monitoring Component Selection Guide

重要说明

组件列表

架构对比

功能对比

安装方案建议

监控组件容量规划

假设与方法论

Prometheus

VictoriaMetrics

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 存储的所有指标

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

通过告警模板创建告警策略

设置告警静默

配置告警规则建议

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

为项目设置通知规则

监控面板管理

功能概述

管理面板

管理图表

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 BlackboxExporter 监控模块

通过 CLI 创建黑盒监控项和告警

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

前置条件

操作流程

操作结果

了解更多

后续操作

VictoriaMetrics 监控数据的备份与恢复

功能概述

使用场景

前提条件

操作步骤

操作结果

了解更多

后续操作

从自定义命名的网络接口采集网络数据

功能概述

使用场景

前提条件

操作步骤

操作结果

了解更多

后续操作

介绍

Monitoring 模块是 ACP 平台可观测性套件的核心组件，为平台管理员和运维团队提供全面的监控和告警能力。

该模块提供四项关键的监控功能：

- 指标收集，用于实时采集集群、节点、应用和容器的性能数据
- 仪表盘，用于直观展示和分析系统健康状况及性能趋势
- 告警，通过可定制的规则和阈值实现问题的主动检测
- 通知，及时将告警信息传递给运维人员

通过与 Prometheus 和 VictoriaMetrics 等开源组件的集成，Monitoring 模块帮助组织维护系统可靠性，防止停机，降低运维成本，并确保整个基础设施的最佳性能。

安装

目录

Overview

监控组件作为可观测性模块中监控、告警、巡检和健康检查功能的基础设施。本文档介绍如何在集群内安装 ACP Monitoring with Prometheus 插件或 ACP Monitoring with VictoriaMetrics 插件。

安装准备

> **INFO**
>
> 监控的部分组件资源消耗较大，建议在 infra 节点上运行，并设置 nodeSelector 和 tolerations，确保只在这些节点上运行。如果您正在评估产品且尚未配置 infra 节点，可以去除这些设置，使组件在所有节点上运行。
>
> 关于 infra 节点规划的指导，请参见 Cluster Node Planning。

安装监控组件前，请确保满足以下条件：

- 已根据监控组件选择指南选择合适的监控组件。
- 在业务集群中安装时，确保 `global` 集群可以访问业务集群的 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 **Storage** 部分创建相应资源。

通过控制台安装 **ACP Monitoring with Prometheus** 插件

安装流程

1. 进入 **App Store Management** > **Cluster Plugins**，选择目标集群。
2. 找到 **ACP Monitoring with Prometheus** 插件，点击 **Install**。
3. 配置以下参数：

参数	说明
规模配置	支持三种配置：**Small Scale**、**Medium Scale** 和 **Large Scale**： - 默认值基于平台推荐的负载测试值 - 可根据实际集群规模选择或自定义配额 - 默认值会随平台版本更新，固定配置建议自定义设置
存储类型	- **LocalVolume**：本地存储，数据存放在指定节点 - **StorageClass**：通过存储类自动生成持久卷 - **PV**：使用已有持久卷 注意：安装后不可修改存储配置
Replica 数量	设置监控组件 Pod 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 **Install** 完成安装。

访问方式

安装完成后，可通过以下地址访问组件（将 `<>` 替换为实际值）：

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

通过 YAML 安装 ACP Monitoring with Prometheus 插件

1. 检查可用版本

在 `global` 集群中检查 ModulePlugin 和 ModuleConfig 资源，确保插件已发布：

```
# kubectl get moduleplugin | grep prometheus
prometheus                        30h
# kubectl get moduleconfig | grep prometheus
prometheus-v4.1.0                 30h
```

表示集群中存在 ModulePlugin `prometheus` ，且版本 `v4.1.0` 已发布。

2. 创建 ModuleInfo

创建 ModuleInfo 资源安装插件，示例不带配置参数：

```
kind: ModuleInfo
apiVersion: cluster.alauda.io/v1alpha1
metadata:
  name: global-prometheus
  labels:
    cpaas.io/cluster-name: global
    cpaas.io/module-name: prometheus
    cpaas.io/module-type: plugin
spec:
  version: v4.1.0
  config:
    storage:
      type: LocalVolume
      capacity: 40
      nodes:
        - xxx.xxx.xxx.xx
      path: /cpaas/monitoring
      storageClass: ""
      pvSelectorK: ""
      pvSelectorV: ""
    replicas: 1
    components:
      prometheus:
        retention: 7
        scrapeInterval: 60
        scrapeTimeout: 45
        resources: null
      nodeExporter:
        port: 9100
        resources: null
      alertmanager:
        resources: null
      kubeStateExporter:
        resources: null
      prometheusAdapter:
        resources: null
      thanosQuery:
        resources: null
    size: Small
  valuesOverride:
    ait/chart-kube-prometheus:
      global:
        nodeSelector:
```

```
        node-role.kubernetes.io/infra: "true"
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
        operator: Equal
```

资源设置参考，示例 prometheus：

```
spec:
  config:
    components:
      prometheus:
        resources:
          limits:
            cpu: 2000m
            memory: 2000Mi
          requests:
            cpu: 1000m
            memory: 1000Mi
```

更多详情请参见监控组件容量规划

YAML 字段说明（Prometheus）：

字段路径	说明
`metadata.labels.cpaas.io/cluster-name`	插件安装的目标集群名称。
`metadata.labels.cpaas.io/module-name`	必须为 `prometheus`。
`metadata.labels.cpaas.io/module-type`	必须为 `plugin`。
`metadata.name`	ModuleInfo 名称（如 `<cluster>-prometheus`）。
`spec.version`	要安装的插件版本。
`spec.config.storage.type`	存储类型：`LocalVolume`、`StorageClass` 或 `PV`。

字段路径	说明
`spec.config.storage.capacity`	Prometheus 存储大小（Gi），建议最小 30 Gi。
`spec.config.storage.nodes`	当 `storage.type=LocalVolume` 时的节点列表，支持最多 1 个节点。
`spec.config.storage.path`	当 `storage.type=LocalVolume` 时的本地路径。
`spec.config.storage.storageClass`	当 `storage.type=StorageClass` 时的存储类名称。
`spec.config.storage.pvSelectorK`	当 `storage.type=PV` 时的 PV 选择器键。
`spec.config.storage.pvSelectorV`	当 `storage.type=PV` 时的 PV 选择器值。
`spec.replicas`	Replica 数量，仅适用于 `StorageClass` / `PV` 类型。
`spec.config.components.prometheus.retention`	数据保留天数。
`spec.config.components.prometheus.scrapeInterval`	抓取间隔秒数，适用于无 `interval` 的 ServiceMonitors。
`spec.config.components.prometheus.scrapeTimeout`	抓取超时秒数，必须小于 `scrapeInterval`。
`spec.config.components.prometheus.resources`	Prometheus 的资源配置。
`spec.config.components.nodeExporter.port`	Node Exporter 端口（默认 9100）。
`spec.config.components.nodeExporter.resources`	Node Exporter 的资源配置。
`spec.config.components.alertmanager.resources`	Alertmanager 的资源配置。

字段路径	说明
`spec.config.components.kubeStateExporter.resources`	Kube State Exporter 的资源配置。
`spec.config.components.prometheusAdapter.resources`	Prometheus Adapter 的资源配置。
`spec.config.components.thanosQuery.resources`	Thanos Query 的资源配置。
`spec.config.size`	监控规模：`Small`、`Medium` 或 `Large`。
`spec.valuesOverride.ait/chart-kube-prometheus.global.nodeSelector`	可选。监控组件的 nodeSelector，用于选择监控组件运行节点，主要用于 infra 节点。
`spec.valuesOverride.ait/chart-kube-prometheus.global.tolerations`	可选。监控组件的 tolerations，若 nodeSelector 选择了带污点节点，使用此参数指定污点容忍的 key、value 和 effect，主要用于 infra 节点。

3. 验证安装

由于创建后 ModuleInfo 名称会变化，可通过标签定位资源，检查插件状态和版本：

```
kubectl get moduleinfo -l cpaas.io/module-name=prometheus
NAME                                             CLUSTER         MODULE
DISPLAY_NAME   STATUS    TARGET_VERSION   CURRENT_VERSION   NEW_VERSION
global-e671599464a5b1717732c5ba36079795          global          prometheus    prometheus
Running   v4.1.0           v4.1.0           v4.1.0
```

字段说明：

- `NAME`：ModuleInfo 资源名称
- `CLUSTER`：插件安装的集群

- `MODULE`：插件名称
- `DISPLAY_NAME`：插件显示名称
- `STATUS`：安装状态，`Running` 表示安装成功且正在运行
- `TARGET_VERSION`：目标安装版本
- `CURRENT_VERSION`：安装前版本
- `NEW_VERSION`：可安装的最新版本

通过控制台安装 **ACP Monitoring with VictoriaMetrics** 插件

前提条件

- 若仅安装 VictoriaMetrics agent，需确保 VictoriaMetrics Center 已在其他集群安装。

安装流程

1. 进入 **App Store Management** > **Cluster Plugins**，选择目标集群。
2. 找到 **ACP Monitoring with VictoriaMetrics** 插件，点击 **Install**。
3. 配置以下参数：

参数	说明
规模配置	支持三种配置：**Small Scale**、**Medium Scale** 和 **Large Scale**： - 默认值基于平台推荐的负载测试值 - 可根据实际集群规模选择或自定义配额 - 默认值会随平台版本更新，固定配置建议自定义设置
仅安装 **Agent**	- 关闭：安装完整 VictoriaMetrics 组件套件 - 开启：仅安装 VMAgent 采集组件，依赖 VictoriaMetrics Center

参数	说明
VictoriaMetrics Center	选择已安装完整 VictoriaMetrics 组件的集群
存储类型	- **LocalVolume**：本地存储，数据存放在指定节点 - **StorageClass**：通过存储类自动生成持久卷 - **PV**：使用已有持久卷
Replica 数量	设置监控组件 Pod 数量： - LocalVolume 存储类型不支持多副本 - 其他存储类型请根据页面提示配置
参数配置	可调整监控组件的数据参数 注意：数据可能会暂时超过保留期限后再删除

4. 点击 **Install** 完成安装。

通过 YAML 安装 ACP Monitoring with VictoriaMetrics 插件

1. 检查可用版本

在 `global` 集群中检查 ModulePlugin 和 ModuleConfig 资源，确保插件已发布：

```
# kubectl get moduleplugin | grep victoriametrics
victoriametrics                        30h
# kubectl get moduleconfig | grep victoriametrics
victoriametrics-v4.1.0                 30h
```

表示集群中存在 ModulePlugin `victoriametrics`，且版本 `v4.1.0` 已发布。

2. 创建 ModuleInfo

创建 ModuleInfo 资源安装插件，示例不带配置参数：

```
kind: ModuleInfo
apiVersion: cluster.alauda.io/v1alpha1
metadata:
  name: business-1-victoriametrics
  labels:
    cpaas.io/cluster-name: business-1
    cpaas.io/module-name: victoriametrics
    cpaas.io/module-type: plugin
spec:
  version: v4.1.0
  config:
    storage:
      type: LocalVolume
      capacity: 40
      nodes:
        - xxx.xxx.xxx.xx
      path: /cpaas/monitoring
      storageClass: ""
      pvSelectorK: ""
      pvSelectorV: ""
    replicas: 1
    agentOnly: false
    agentReplicas: 1
    crossClusterDependency:
      victoriametrics: ""
    components:
      nodeExporter:
        port: 9100
        resources: null
      vmstorage:
        retention: 7
        resources: null
      kubeStateExporter:
        resources: null
      vmalert:
        resources: null
      prometheusAdapter:
        resources: null
      vmagent:
        scrapeInterval: 60
        scrapeTimeout: 45
        resources: null
      vminsert:
```

```
        resources: null
      alertmanager:
        resources: null
      vmselect:
        resources: null
    size: Small
  valuesOverride:
    ait/chart-victoriametrics:
      global:
        nodeSelector:
          node-role.kubernetes.io/infra: "true"
        tolerations:
        - effect: NoSchedule
          key: node-role.kubernetes.io/infra
          value: reserved
          operator: Equal
```

资源设置参考，示例 prometheus：

```
spec:
  config:
    components:
      vmagent:
        resources:
          limits:
            cpu: 2000m
            memory: 2000Mi
          requests:
            cpu: 1000m
            memory: 1000Mi
```

更多详情请参见监控组件容量规划

YAML 字段说明（VictoriaMetrics）：

字段路径	说明
`metadata.labels.cpaas.io/cluster-name`	插件安装的目标集群名称。
`metadata.labels.cpaas.io/module-name`	必须为 `victoriametrics`。
`metadata.labels.cpaas.io/module-type`	必须为 `plugin`。

字段路径	说明
`metadata.name`	ModuleInfo 名称（如 `<cluster>-victoriametrics`）。
`spec.version`	要安装的插件版本。
`spec.config.storage.type`	存储类型：`LocalVolume`、`StorageClass` 或 `PV`。
`spec.config.storage.capacity`	VictoriaMetrics 存储大小（Gi），建议最小 30 Gi。
`spec.config.storage.nodes`	当 `storage.type=LocalVolume` 时的节点列表，支持最多 1 个节点。
`spec.config.storage.path`	当 `storage.type=LocalVolume` 时的本地路径。
`spec.config.storage.storageClass`	当 `storage.type=StorageClass` 时的存储类名称。
`spec.config.storage.pvSelectorK`	当 `storage.type=PV` 时的 PV 选择器键。
`spec.config.storage.pvSelectorV`	当 `storage.type=PV` 时的 PV 选择器值。
`spec.replicas`	Replica 数量，LocalVolume 不支持多副本。
`spec.config.components.vmstorage.retention`	vmstorage 的数据保留天数。
`spec.config.components.vmagent.scrapeInterval`	抓取间隔秒数，适用于无 `interval` 的 ServiceMonitors。
`spec.config.components.vmagent.scrapeTimeout`	抓取超时秒数，必须小于 `scrapeInterval`。
`spec.config.components.vmstorage.resources`	vmstorage 的资源配置。

字段路径	说明
`spec.config.components.nodeExporter.port`	Node Exporter 端口（默认 9100）。
`spec.config.components.nodeExporter.resources`	Node Exporter 的资源配置。
`spec.config.components.alertmanager.resources`	Alertmanager 的资源配置。
`spec.config.components.kubeStateExporter.resources`	Kube State Exporter 的资源配置。
`spec.config.components.prometheusAdapter.resources`	Prometheus Adapter 的资源配置（用于 HPA/自定义指标）。
`spec.config.components.vmagent.resources`	vmagent 的资源配置。
`spec.config.size`	监控规模：`Small`、`Medium` 或 `Large`。
`spec.valuesOverride.ait/chart-victoriametrics.global.nodeSelector`	可选。监控组件的 nodeSelector，用于选择监控组件运行节点，主要用于 infra 节点。
`spec.valuesOverride.ait/chart-victoriametrics.global.tolerations`	可选。监控组件的 tolerations，若 nodeSelector 选择了带污点节点，使用此参数指定污点容忍的 key、value 和 effect，主要用于 infra 节点。

3. 验证安装

由于创建后 ModuleInfo 名称会变化，可通过标签定位资源，检查插件状态和版本：

```
kubectl get moduleinfo -l cpaas.io/module-name=victoriametrics
NAME                                          CLUSTER         MODULE
DISPLAY_NAME     STATUS    TARGET_VERSION   CURRENT_VERSION   NEW_VERSION
global-e671599464a5b1717732c5ba36079795          global          victoriametrics
victoriametrics  Running   v4.1.0           v4.1.0            v4.1.0
```

字段说明：

- `NAME`：ModuleInfo 资源名称
- `CLUSTER`：插件安装的集群
- `MODULE`：插件名称
- `DISPLAY_NAME`：插件显示名称
- `STATUS`：安装状态，`Running` 表示安装成功且正在运行
- `TARGET_VERSION`：目标安装版本
- `CURRENT_VERSION`：安装前版本
- `NEW_VERSION`：可安装的最新版本

Menu

架构

Monitoring Module Architecture

Overall Architecture Explanation

Monitoring System

Alerting System

Notification System

Monitoring Component Selection Guide

重要说明

组件列表

架构对比

功能对比

安装方案建议

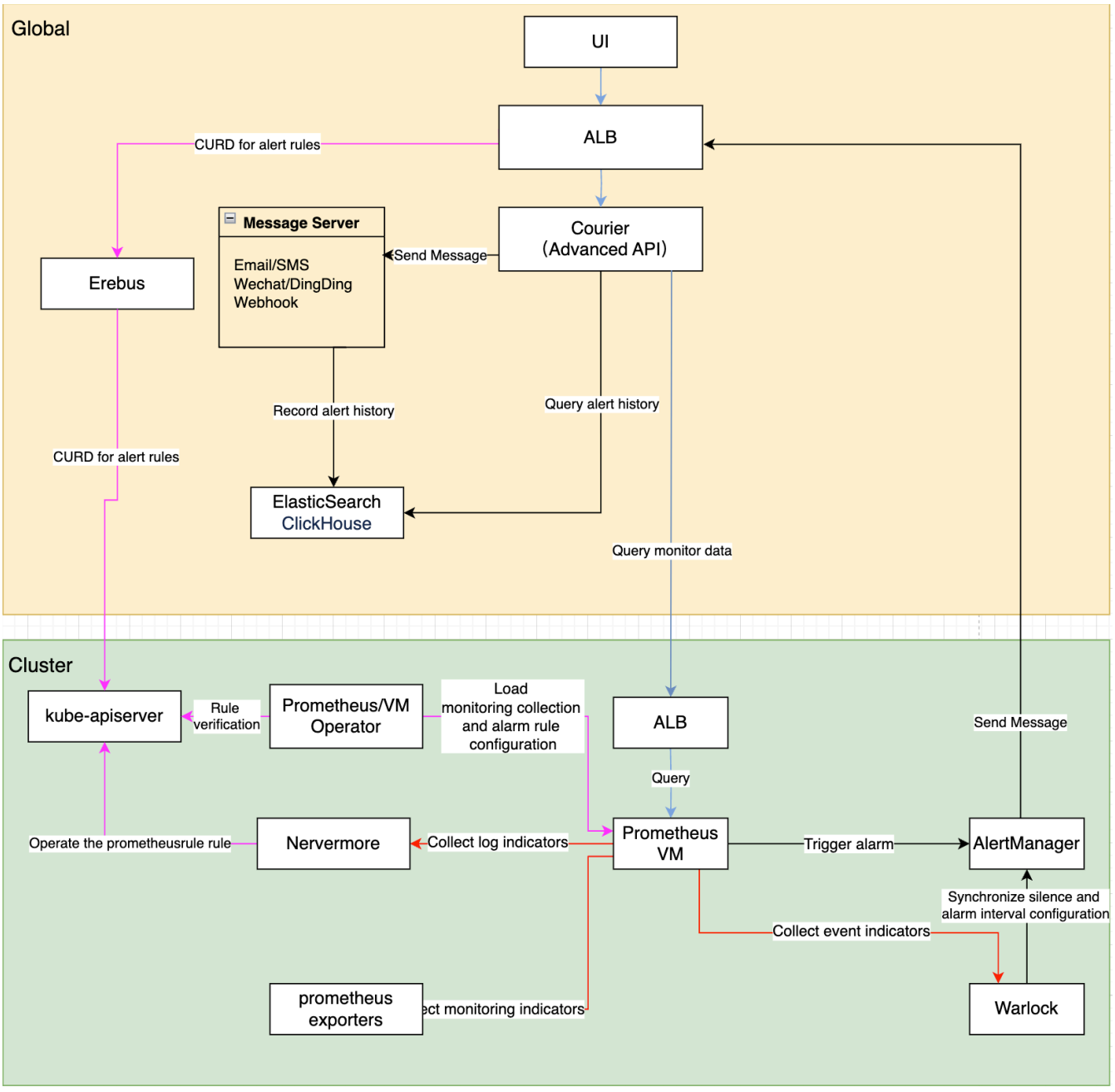
监控组件容量规划

假设与方法论

Prometheus

VictoriaMetrics

Monitoring Module Architecture



目录

Overall Architecture Explanation

监控系统由以下核心功能模块组成：

1. Monitoring System

- 数据采集与存储：采集并持久化来自多个来源的监控指标
- 数据查询与可视化：提供灵活的监控数据查询和可视化能力

2. Alerting System

- 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. Notification System

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

Monitoring System

Data Collection and Storage

1. Prometheus/VictoriaMetrics Operator 职责：
   - 加载并校验监控采集配置
   - 加载并校验告警规则配置
   - 同步配置到 Prometheus/VictoriaMetrics 实例
2. 监控数据来源：
   - Nevermore：生成日志相关指标
   - Warlock：生成事件相关指标
   - Prometheus/VictoriaMetrics：通过 ServiceMonitor 发现并采集各类 exporter 指标

Data Query and Visualization

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 转发请求至 Courier 组件
   - Courier API 处理查询：
     - 内置指标：通过指标接口获取 PromQL 并查询
     - 自定义指标：直接转发 PromQL 到监控组件
   - 监控面板获取数据并展示
2. 监控面板管理流程：
   - 用户访问 `global` 集群 ALB（路径：`/kubernetes/cluster_name/apis/ait.alauda.io/v1alpha2/MonitorDashboard`）
   - ALB 转发请求至 Erebus 组件
   - Erebus 路由请求至目标监控集群
   - Warlock 组件负责：
     - 校验监控面板配置的合法性

- 管理 MonitorDashboard CR 资源

Alerting System

Alert Rule Management

告警规则配置流程：

1. 用户访问 `global` 集群 ALB（路径：`/kubernetes/cluster_name/apis/monitoring.coreos.com/v1/prometheusrules`）
2. 请求经过 ALB -> Erebus -> 目标集群 kube-apiserver
3. 各组件职责：
   - Prometheus/VictoriaMetrics Operator：
     - 校验告警规则合法性
     - 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

Alert Processing Workflow

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：
   - 告警触发后发送至 Alertmanager
   - Alertmanager -> ALB -> Courier API
   - Courier API 负责派发通知
3. 告警存储：

- 告警历史存储于 ElasticSearch/ClickHouse

Real-time Alert Status

1. 状态采集：
   - `global` 集群 Courier 生成指标：
     - cpaas_active_alerts：当前活跃告警
     - cpaas_active_silences：当前静默配置
   - 全局 Prometheus 每 15 秒采集一次
2. 状态展示：
   - 前端通过 Courier API 查询并展示实时状态

Notification System

Notification Configuration Management

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：
   - Notification Template：apiVersion: "ait.alauda.io/v1beta1", kind: "NotificationTemplate"
   - Notification Contact Group：apiVersion: "ait.alauda.io/v1beta1", kind: "NotificationGroup"
   - Notification Policy：apiVersion: "ait.alauda.io/v1beta1", kind: "Notification"
3. Courier 负责：
   - 校验通知模板合法性
   - 校验通知联系人组合法性

- 校验通知策略合法性

Notification Server Management

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：
   - 校验通知服务器配置合法性

Monitoring Component Selection Guide

在安装集群监控时，平台提供了两种监控组件供您选择：VictoriaMetrics 和 Prometheus。本文将详细介绍这两种组件的特点及适用场景，帮助您做出最合适的选择。

目录

重要说明

- 安装集群监控组件时，只能选择 VictoriaMetrics 或 Prometheus 其中之一。
- 从 3.18 版本开始，VictoriaMetrics 已升级为 Beta 状态，满足生产环境使用条件。
- VictoriaMetrics 适用于高可用需求及多集群监控场景。
- Prometheus 适用于单集群监控场景，尤其是规模较小的情况。

组件列表

Prometheus 相关组件

组件名称	功能描述
Prometheus Server	负责采集、存储和查询监控数据的核心服务器
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，负责告警规则和通知处理
PushGateway	支持监控数据的推送模式，适用于特殊网络环境下的数据传输

VictoriaMetrics 相关组件

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

架构对比

Prometheus 架构

Prometheus 是成熟的开源监控系统，是 CNCF 继 Kubernetes 之后的第二个毕业项目，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。
- 支持千节点规模的集群监控。

VictoriaMetrics 架构

VictoriaMetrics 是新一代高性能时序数据库及监控解决方案，具备以下优势：

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

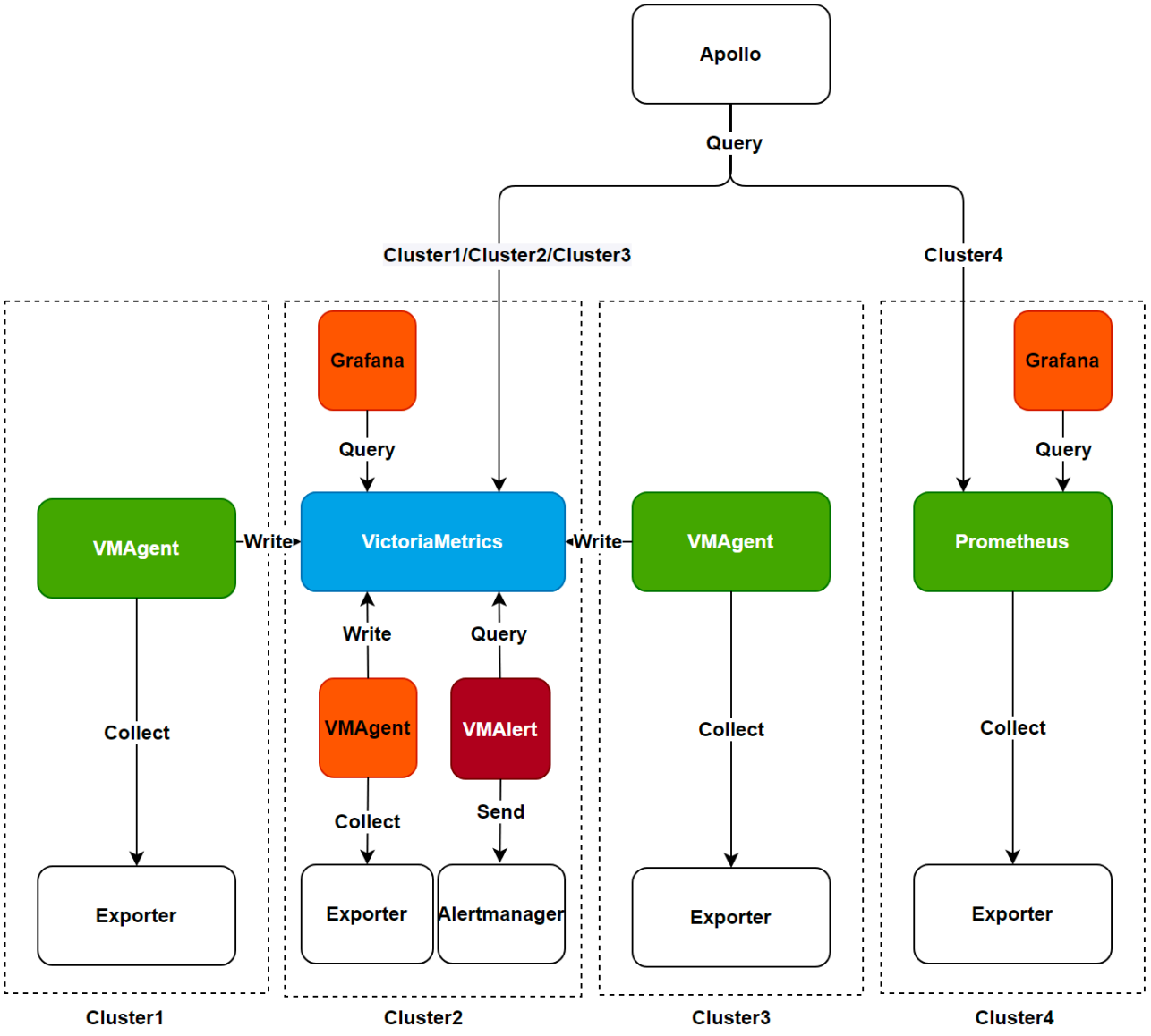
功能对比

功能	**Prometheus**	**VictoriaMetrics**	说明
高可用安装	❌	✅	VictoriaMetrics 支持真正的集群高可用，且数据一致性更好
单节点安装	✅	✅	两者均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
资源效率	较高	更优	VictoriaMetrics 资源利用率更高

功能	**Prometheus**	**VictoriaMetrics**	说明
社区支持	非常成熟	快速发展	Prometheus 拥有更大的社区生态

安装方案建议

监控安装架构概览



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

注意：更换监控组件时，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

该方式对应上图中的 **cluster4** 架构：

- 使用 Prometheus 组件采集和处理监控数据。
- 通过监控面板查询和展示数据。
- 适用于单集群场景。

VictoriaMetrics 安装方式

VictoriaMetrics 支持以下两种安装模式：

1. 单集群安装模式

   - 对应上图中的 **cluster2** 架构。
   - 所有 VictoriaMetrics 组件安装在同一集群内。
   - 使用 VMAgent 采集数据并写入 VictoriaMetrics。
   - VMAlert 负责告警规则评估。
   - 通过监控面板查询和展示数据。 提示：建议数据规模低于每秒 100 万时使用此模式。

2. 多集群安装模式

   - 对应上图中的 **cluster1/cluster2/cluster3** 架构。
   - 在业务集群中安装 VMAgent 作为数据采集智能体。
   - VMAgent 将数据写入中央监控集群中的 VictoriaMetrics。
   - 支持多集群统一监控管理。 提示：请确保在安装 VMAgent 前，VictoriaMetrics 服务已部署在监控集群中。

选择建议

适合使用 **VictoriaMetrics** 的场景

- 高性能及可扩展性需求：适用于处理高吞吐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：对监控组件的高可用性有保障要求。

- 多集群管理：需要跨多个集群统一管理监控数据。

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。
- 已有 **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，迁移成本较高。

监控组件容量规划

监控组件负责存储从平台中一个或多个集群收集的指标数据。因此，您需要提前评估监控规模，并根据本文档中的指南规划监控组件所需的资源。

目录

假设与方法论

- 本文档中的数据来自受控实验室性能报告，旨在作为生产规划的容量基线。
- 磁盘示例的保留时间为7天；其他保留目标请按比例调整。
- 存储基线符合上述警告（SSD，约6000 IOPS，约250MB/s 读写，独立挂载）。
- 测试工作负载涵盖了典型的监控页面，如“acp ns overview page”和“platform region detail page”。

Prometheus

以下是按规模划分的 Prometheus 及相关组件（Thanos Query、Thanos Sidecar 等）的容量建议。

小规模 — 10 个 worker 节点，500 个双容器 Pod

- 指标摄取速率：约2800 样本/秒

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Not
courier-api	courier	2	2C	4Gi	-	-
kube-prometheus-thanos-query	thanos-query	1	1C	1Gi	-	-
prometheus-kube-prometheus-0	prometheus	1	2C	8Gi	20G	7 天内约写入 10G

中等规模 — 50 个 worker 节点，2000 个双容器 Pod

- 指标摄取速率：约7294 样本/秒

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Not
courier-api	courier	2	4C	4Gi	-	-
kube-prometheus-thanos-query	thanos-query	1	2.5C	8Gi	-	-

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Not
prometheus-kube-prometheus-0	prometheus	1	4C	8Gi	40G	7 天内约写入 30G

大规模 — 500 个 worker 节点，10000 个双容器 Pod

- 指标摄取速率：约41575 样本/秒

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Not
courier-api	courier	2	6C	4Gi	-	-
kube-prometheus-thanos-query	thanos-query	1	2C	6Gi	-	现场署可使用个副
prometheus-kube-prometheus-0	prometheus	1	8C	20Gi	100G	峰值存约 15G 7 天约写 69G

VictoriaMetrics

以下是按规模划分的 VictoriaMetrics 组件容量建议。

小规模 — 10 个 worker 节点，500 个双容器 Pod

- 指标摄取速率：约3274 样本/秒

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
courier-api	courier	1	2C	4Gi	-	-
vmselect-cluster	proxy	1	1C	200Mi	-	-
vmselect	vmselect	1	500m	1Gi	-	-
vmstorage-cluster	vmstorage	1	500m	2Gi	3G	7 天内约写入 1.5G

中等规模 — 50 个 worker 节点，2000 个双容器 Pod

- 指标摄取速率：约6940 样本/秒

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
courier-api	courier	2	4C	4Gi	-	-
vmselect-cluster	proxy	1	1C	200Mi	-	-
vmselect	vmselect	1	2C	2Gi	-	-
vmstorage-cluster	vmstorage	1	2C	2Gi	10G	7 天内约写入 2.6G

大规模 — 500 个 worker 节点，10000 个双容器 Pod

- 指标摄取速率：约34300 样本/秒

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
courier-api	courier	2	6C	4Gi	-	-
vmselect-cluster	proxy	1	2C	200Mi	-	-
vmselect	vmselect	1	5C	3Gi	-	-
vmstorage-cluster	vmstorage	1	2C	6Gi	30G	7 天内约写入 16.8G

核心概念

目录

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：发送告警的渠道
- 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。

Menu

操作指南

指标管理

查看平台组件暴露的指标

查看 Prometheus / VictoriaMetrics 存储的所有指标

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

通过告警模板创建告警策略

设置告警静默

配置告警规则建议

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

为项目设置通知规则

监控面板管理

功能概述

管理面板

管理图表

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 BlackboxExporter 监控模块

通过 CLI 创建黑盒监控项和告警

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

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，使用 `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。以下将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，使用 `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, (1)
  "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", (2)
   "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,  (3)
  "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_total_irate5m{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\"}))", (4)
  "summary": "Pod ({{.labels.pod}}) 的 CPU 使用率 {{.externalLabels.comparison}}
{{.externalLabels.threshold}}",
  "type": "metric",
  "unit": "%",
  "legend": "{{.namespace}}/{{.pod}}",
  "variables": [ (5)
   "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

```yaml
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` 对象描述应用暴露的指标：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: cpaas-monitor
    chart: cpaas-monitor
    heritage: Helm
    prometheus: kube-prometheus (1)
    release: cpaas-monitor
  name: cpaas-elasticsearch-Exporter
  namespace: cpaas-system (2)
spec:
  jobLabel: service_name (3)
  namespaceSelector: (4)
    any: true
  selector: (5)
    matchExpressions:
      - key: service_name
        operator: Exists
  endpoints:
    - port: cpaas-elasticsearch (6)
      path: /_prometheus/metrics (7)
      interval: 60s (8)
      honorLabels: true
      basicAuth: (9)
        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，默认是 /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，检查是否存在 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

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。通过利用预置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层管理机制，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，针对 `global` 集群和工作负载集群预设丰富的告警规则。
- 自定义告警规则：支持基于多种数据源创建告警规则，包括预置监控指标、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，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，便于运维人员和管理员分析近期监控告警情况。

功能优势

- 全面的监控覆盖：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。
- 高效的告警管理：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。
- 及时的问题发现：告警通知自动触发，确保问题及时发现，支持多渠道告警推送，实现主动预防问题。
- 完善的权限管理：严格的告警策略访问控制，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（如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）。

操作流程

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（如某命名空间下的 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，仅作用于当前资源。
  - 选择多个命名空间时，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
表达式	Prometheus 格式的监控指标算法，如 `rate(node_network_receive_bytes{instance="$server",device!~"lo"}[5m])`
指标单位	自定义监控指标单位，可手动输入或从平台预置单位中选择
图例参数	控制图表中曲线对应的名称，格式为 `{{.LabelName}}`，如 `{{.hostname}}`
时间范围	日志/事件查询的时间窗口
日志内容	日志内容查询字段（如 Error），多个查询字段之间用 OR 连接

参数	说明
事件原因	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
触发条件	由比较运算符、告警阈值和持续时间（可选）组成。根据实时值/日志条数/事件条数与告警阈值的比较，以及实时值在告警阈值范围内的持续时间，判断是否触发告警。
告警级别	分为 Critical、Serious、Warning 和 Info 四个级别。可根据告警规则对业务的影响，为对应资源设置合理的告警级别。

事件告警参数

参数	说明
时间范围	事件查询的时间窗口
事件监控项	支持监控事件级别或事件原因，多个字段之间用 OR 连接
触发条件	基于事件数量进行比较判断
告警级别	与资源告警级别定义相同

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：可根据告警级别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，可设置 labels 和 annotations。
2. 具体配置请参考 Prometheus Alerting Rules Documentation↗。
3. 注意：labels 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（如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）。

操作流程

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：

```
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、多个（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、多个（用 | 分
隔）或任意（.*）
    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 # 告警策略展示名称
  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-e998f0b94854ee1eade5ae79279e00
          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
告警指标名称即为 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|VirtualMachi
            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-
息
```

通过 CLI 创建事件告警

前提条件

- 已配置通知策略（如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）。

操作流程

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
['*'] 时表示所有命名空间。
      # 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
称即为 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（如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）。

操作流程

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
表达式	Prometheus 格式的监控指标算法，如 `rate(node_network_receive_bytes{instance="$server",device!~"lo"}[5m])`
指标单位	自定义监控指标单位，可手动输入或从平台预置单位中选择
图例参数	控制图表中曲线对应的名称，格式为 `{{.LabelName}}` ，如 `{{.hostname}}`
时间范围	日志/事件查询的时间窗口
日志内容	日志内容查询字段（如 Error），多个查询字段之间用 OR 连接
事件	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

参数	说明
原因	
触发条件	由比较运算符、告警阈值和持续时间（可选）组成。
告警级别	分为 Critical、Serious、Warning 和 Info 四个级别。可根据告警规则对业务的影响，为对应资源设置合理的告警级别。

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，可控制该告警策略下所有规则在静默时间内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避免此类情况发生。

注意：静默状态持续到静默结束时间后，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. 点击需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；若选择“全选”，则仅作用于当前选定的资源范围，后续的 删除/添加 资源操作不再处理。

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	告警静默的时间范围。静默时间开始时，告警进入静默状态；若静默结束时间后告警策略仍处于告警状态或再次触发告警，告警通知将恢复。永久：静默设

参数	说明
时间	置持续到告警策略被删除。自定义：自定义静默开始和结束时间，时间间隔不得少于 5 分钟。

5. 点击 设置。

   提示：从设置静默到静默开始前，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。若需静默所有资源，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。若需静默所有资源，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。

- 尽量少创建新规则：仅创建满足具体需求的规则。通过尽量少的规则数量，可以构建更易管理和集中的监控告警体系。
- 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本原因。此策略可显著减少需创建的规则总数。
- 变更前规划和评估需求：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，而无需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，包含症状描述、可能原因和建议操作。信息应清晰简洁，提供排查流程或相关信息链接，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，将严重级别设置为 Critical，表示需相关人员立即处理。通过设定严重级别，帮助用户判断告警响应优先级，确保及时处理紧急问题。

通知管理

目录

功能概述

通过通知功能，您可以将平台的监控和告警功能集成起来，及时向通知接收者发送预警信息，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者，通过对接收通知消息的实体进行分类，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，用于规范通知策略中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者提供发送通知消息的服务。平台当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。

```yaml
# 微信企业号 corpId、corpSecret、agentId 获取方法参考官方文档：
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，需要在平台的 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。

```yaml
# 钉钉 appKey、appSecret、agentId 获取方法：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，需要在平台的 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。

```yaml
# 飞书 appId、appSecret 获取方法：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，需要在平台的 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 `smtp.yeah.net`。
端口	通知服务器端口号。勾选 使用 **SSL** 时，需填写 SSL 端口号。
服务器配置	使用 **SSL**：安全套接字层（SSL）是一种标准安全技术。SSL 开关用于控制是否建立服务器与客户端之间的加密连接。 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。

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者。例如，您可以将运维团队设置为通知联系人组，方便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者，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邮箱，平台将向该邮箱及组内所有联系人的邮箱发送通知。
Webhook URL/微信企业群机器人**/**钉钉群机器人**/**飞书群机器人	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使用该方式接收通知。
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，用于规范通知策略中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，则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，配合平台的告警功能使用； 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。您可以修改变量显示名和内容格式，但不能修改变量本身。参考 Reference Variables 和 Special Formatting Markup Language in Emails。
3. 平台根据实际运维场景提供了多种通知类型的默认通知模板内容，能满足大多数通知消息设置需求。如无特殊要求，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，可为变量自定义显示名；例如：`告警等级: {{ .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	`{{ .externalLabels.status }}`	例如：告警中。
告警等级	`{{ .externalLabels.severity }}`	例如：严重。
告警集群	`{{ .labels.alert_cluster }}`	例如：发生告警的集群 1。
告警对象	`{{ .externalLabels.object }}`	告警发生的资源类型和名称，例如节点 192.168.16.53。
规则名称	`{{ .labels.alert_resource }}`	告警规则名称，例如 cpaas-node-rules。
告警描述	`{{ .externalLabels.summary }}`	告警规则描述。
触发值	`{{ .externalLabels.currentValue }}`	触发告警的监控值。
告警时间	`{{ dateFormatWithZone .startsAt "2006-01-02 15:04:05" "Asia/Chongqing" }}`	告警开始时间。
恢复时间	`{{ dateFormatWithZone .endsAt "2006-01-02 15:04:05" "Asia/Chongqing" }}`	告警结束时间。
指标名称	`{{ .labels.alert_indicator }}`	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。
字体	`<font color="#FF0000">设置字体颜色</font>` `<b>加粗字体</b>`	设置字体格式。
标题	`<h1>一级标题</h1>`，支持至 h6（标题 6）。	设置标题级别。
段落	`<p>段落</p>`	插入普通段落文本。
引用	`<q>引用</q>`	插入简短引用内容。
超链接	`<a href="//www.example.com">超链接</a>`	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，则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 通知联系人时，用户需先在 个人信息 中修改联系方式，填写其 `微信企业号 ID`。

操作流程

1. 在左侧导航栏点击 运维中心 > 通知。

2. 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者，平台将使用指定的通知方式通知该组。
通知接收者	选择添加一个或多个通知接收者，平台将根据接收者的 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、**WebHook URL**，支持多选。 注意：部分参数会在配置通知服务器后显示。
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 中正确配置其通信工具 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

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、更灵活的监控体验。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配置的公共共享面板，数据根据业务所属的命名空间进行隔离。
- 支持管理面板内的图表，允许用户添加、删除、修改面板，支持面板的放大缩小及拖拽移动。
- 支持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，用于管理面板。分组可基于自定义变量重复展示。
- 支持的面板类型包括：趋势图、阶梯折线图、柱状图、水平柱状图、柱状仪表盘、仪表盘、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，布局和样式灵活多样。
- 与平台角色权限无缝集成，业务视图可定义自己的监控面板，同时保证数据隔离。

- 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，监控面板提供丰富的可视化选项，直观展示应用运行状态和资源消耗。您可以针对不同应用场景定制专属监控视图，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中安装的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** 标签切换不同集群视图。
- 监控面板依赖集群中的监控组件查询数据源，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，平台会自动请求存储集群查询该集群对应的数据，无需额外配置。

管理面板

面板是由一个或多个图表组成的集合，按一行或多行组织排列，提供清晰的相关信息视图。这些图表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，可输入或选择已有文件夹。
标签	监控面板的标签，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，供新增图表时引用作为指标参数，也可作为面板首页的过滤条件。

2. 添加完成后，点击 创建 完成面板创建。接下来需要 添加变量、添加图表 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，低版本禁止导入。
- 导入面板中若存在平台不支持的图表类型，可能显示为 不支持的面板类型，但可通过修改图表设置实现正常展示。
- 导入后可像普通面板一样进行管理操作，与平台创建的面板无异。

添加变量

1. 在变量表单区域，点击 添加。

查询型变量

查询型变量允许基于时间序列的特征维度过滤数据。可指定查询表达式动态计算生成查询结果。

参数	说明
查询设置	定义查询设置时，除使用 PromQL 查询时间序列外，平台还提供一些常用变量和函数。参考 常用函数和变量。
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 变量值预览 中查看过滤结果是否符合预期。
选择设置	- 多选：在面板首页顶部过滤器中选择时，允许同时选择多个选项。需在图表查询表达式中引用该变量，才能查看对应变量值的数据。 - 全部：勾选后，过滤选项中会启用包含 全部 的选项，用于选择所有变量数据。

常量变量

常量变量是静态变量，值固定不变，常用于存储环境标识、固定阈值或需跨多个图表引用但不作为过滤选项展示的配置参数。

参数	说明
常量值	常量变量的具体值。

自定义变量

自定义变量允许用户定义预设的静态选项列表，作为面板上的下拉过滤器，常用于手动选择特定服务、团队或类别，无需动态数据查询。

参数	说明
自定义设置	输入以逗号分隔的选项值，格式为 display_name : value（例如 Production : prod, Staging : stage, Development : dev），若显示名与值相同，可直接列出值。

文本框变量

文本框变量允许用户直接输入文本，常用于指定不需动态查询的特定值或参数。

参数	说明
文本框值	文本框变量的默认值。

2. 点击 确定 添加一个或多个变量。

添加图表

向当前创建的监控面板添加多个图表，用于展示不同资源的数据。

提示：可点击图表右下角自定义图表大小；点击图表任意位置可调整图表顺序。

1. 点击 添加图表，参考以下说明配置相关参数。

- 图表预览：动态展示所添加指标对应的数据。
- 添加指标：配置图表标题和监控指标。
- 添加方式：支持使用内置指标或原生自定义指标，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，展示当前图表下的数据。
    - 注意：添加到图表的所有指标必须单位统一，不能在同一图表添加多单位指标。
  - 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：控制图表中曲线对应的名称，可使用文本或模板：
  - 规则：输入值必须为 `{{.xxxx}}` 格式，例如 `{{.hostname}}` 会替换为表达式返回的 hostname 标签对应的值。
  - 提示：若输入格式错误，图表中曲线名称将按原格式显示。
- 即时切换：开启 即时 开关时，通过 Prometheus 的 Query 接口查询即时值并排序，适用于统计图和仪表盘图表。关闭时使用 `query_range` 方法计算，查询指定时间段内的一系列数据。
- 图表设置：支持选择不同图表类型可视化指标数据。详见 管理图表。

2. 点击 保存 完成图表添加。

3. 可在面板内添加一个或多个图表。

4. 添加图表后，可通过以下操作确保图表显示和大小符合预期：

- 点击图表右下角自定义大小。
- 点击图表任意位置调整图表顺序。
- 点击 编辑 按钮修改图表设置。
- 点击 删除 按钮删除图表。
- 点击 复制 按钮复制图表。

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，可将图表归类管理。

1. 点击 添加图表 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图表的变量。
  - 禁用重复：不选择变量，使用默认创建的分组。
  - 参数变量：选择当前图表中创建的变量，监控面板会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、删除或移动图表。

2. 添加分组后，可对分组执行以下操作管理面板显示：

- 分组可折叠或展开，隐藏面板部分内容。折叠分组内的图表不会发送查询。
- 将图表移动到分组内，使该图表由该分组管理。分组管理其与下一个分组之间的所有图表。
- 分组折叠时，也可整体移动该分组管理的所有图表。
- 分组的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 按钮。

切换面板

将创建的自定义监控面板设置为主面板。再次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，并通过 主面板 开关切换主面板。

其他操作

您可以点击面板页面右侧的操作按钮，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，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，可根据需要编辑图表并保存为新面板。
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。

管理图表

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

图表说明

序号	图表名称	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。
2	阶梯折线图	在折线图基础上，使用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，如告警次数。
3	柱状图	使用垂直矩形柱表示数据大小，柱高代表数值。	柱状图直观比较数值差异，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	仪表盘	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 利用率和内存使用率。建议配合告警阈值颜色变化，指示异常状态。
6	仪表条	使用垂直矩形条显示指标当前值及其占比。	直观反映关键指标当前状态，如目标完成进度和系统负载。存在多类别同一指标时，更推荐使用仪表条，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 响应码的比例。
8	表格	以行列形式组织数据，便于查看和比较具体数值。	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，通常需要文本说明。	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	图表名称	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、聚类等模式，帮助挖掘指标间关联。
11	文本卡	以卡片形式展示关键信息文本，通常包含标题和简要描述。	适合展示文本信息，如图表说明和故障排查说明。

图表配置说明

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图表类型，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可快速访问。
标准设置	原生指标数据使用的单位。此外，仪表盘和仪表条支持配置 总值 字段，图表中将显示为 当前值/总值 的百分比。
提示信息	鼠标悬停图表时实时数据的显示开关，支持选择排序。
阈值参数	配置图表阈值开关，启用后在图表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盘。
数值映射	重新定义指定值、范围、正则或特殊值，如将 100 定义为满载。该配置仅适用于统计图、表格和仪表盘。

图表特殊参数

图表类型	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，面积图更关注总量及部分比例变化，根据实际需求选择。
仪表盘	仪表盘设置	显示方向：当需要在单个图表中查看多个指标时，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。
统计图	统计图设置	显示方向：当需要在单个图表中查看多个指标时，可设置指标是横向还是纵向排列。 图形模式：可为统计图添加图形，展示指标随时间的趋势。
饼图	饼图设置	最大切片数：可设置减少饼图切片数量，降低比例较低但数量较多类别的干扰，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	可选择饼图或环形图作为展示样式。
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	图形样式	样式：可选择以富文本编辑框或 HTML 方式编辑文本卡中要展示的内容。

通过 **CLI** 创建监控面板

1. 新建 YAML 配置文件，命名为 `example-dashboard.yaml`。

2. 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：

```yaml
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 # 内置中文显示名字段（该字段在中文语言下创建）
    title: english_display_name # 内置英文显示名字段（该字段在英文语言下创建）内置面板可设置双语翻译。
    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（标签根据语言设置对应名称，如英文为 cluster）
          name: cluster # 内置变量名（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 的选项；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 # 当前变量值；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 # 是否包含全部
          label: ns # 内置变量显示名
          multi: true # 是否支持多选
          name: ns # 变量名（唯一）
          options: []
          query: ''
          regex: /.*namespace=\"(.*?)\".*/ # 提取变量值的正则表达式
          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 # 起始时间
      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（标题后的提示）
    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 个网格单位）
      x: 0 # 横向位置
      y: 0 # 纵向位置
    panels: # 图表数据
      title: 图表标题tab # 图表名称
      type: table # 图表类型；目前支持 timeseries、barchart、stat、gauge、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: # 定义图表属性和外观
        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 表示连接 null 值；false 不连接；数字 == 0 表示根据 0 连接 null 值
            drawStyle: line # 图表类型：line，柱状图为 bars，点图为 points
            fillOpacity: 20 # 当 drawStyle 为 area 时存在（当前不支持配置，area 默认为 20）
            thresholdsStyle: # 配置阈值显示方式（当前仅支持线条）
              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 stepBefore（后续支持 stepAfter）
          decimals: 3 # 小数点位数
          min: 0 # 最小值（当前页面配置不支持，仅支持已适配导入）
          max: 1 # 最大值（页面配置仅 stat gauge barGauge pie 支持）
          unit: '%' # 单位
          mappings: # 数值映射配置（当前仅支持值和范围类型；特殊类型支持数据）
            - options: # 值映射规则
                '1': # 对应值
                  index: 0
                  text: 'Running' # 值为 1 时显示 Running
              type: value # 值映射类型
            - options: # 范围映射规则
                from: 2 # 范围起始值
                to: 3 # 范围结束值
                result: # 映射结果
                  index: 1
                  text: 'Error' # 2 到 3 之间的值显示 Error
              type: range # 范围映射类型
            - type: special # 特殊映射类型
              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（当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' # 阈值值
              - color: '#007AF5' # 默认值无值为基准
        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 # 显示名覆盖
                value: '1' # 覆盖显示名
              - 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 # 控制图表布局方向；适用于仪表盘和仪表条（stat 后续支持）
        legend: # 图例配置
          calcs: # 计算方法（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（right 或 bottom，默认 bottom）
          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: # 计算方法（latest、minimum、maximum、average、sum）
            - latest
          limit: 3 # 饼图限制切片数量
        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（当前页面配置不支持，导入支持）
        colorMode: 'value' # 统计图配置；定义指标值颜色模式；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自定义查询设置参考。

函数名	作用
label_names()	返回 Prometheus 中所有标签，如 label_names()。
label_values(label)	返回 Prometheus 中所有监控指标中标签名对应的所有可选值，如 label_values(job)。
label_values(metric, label)	返回 Prometheus 中指定指标中标签名对应的所有可选值，如 label_values(up, job)。
metrics(metric)	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)。
query_result(query)	返回指定 Prometheus 查询的结果，如 query_result(up)。

常用变量

定义查询设置时，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，供参考：

变量名	查询函数
cluster	`label_values(cpaas_cluster_info,cluster)`
node	`label_values(node_load1, instance)`
namespace	`query_result(kube_namespace_labels)`
deployment	`label_values(kube_deployment_spec_replicas{namespace="$namespace"}, deployment)`
daemonset	`label_values(kube_daemonset_status_number_ready{namespace="$namespace"}, daemonset)`
statefulset	`label_values(kube_statefulset_replicas{namespace="$namespace"}, statefulset)`

变量名	查询函数
pod	`label_values(kube_pod_info{namespace=~"$namespace"}, pod)`
vmcluster	`label_values(up, vmcluster)`
daemonset	`label_values(kube_daemonset_status_number_ready{namespace="$namespace"}, daemonset)`

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` ，并提取命名空间。
   - 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，预览示例包含多个命名空间，如 `argocd` 、 `cpaas-system` 等。
2. 添加第二个变量：deployment，引用前面创建的变量：

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

3. 向当前面板添加图表，引用前面添加的变量，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. 添加图表并保存后，可在面板首页查看对应图表信息。

使用内置指标时注意事项

> **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` 仅支持三类计算组件：`deployment`、`daemonset`、`statefulset`；
> - `kind` 仅支持指定类型之一：`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

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

与依赖平台已有的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触发告警。例

如，当某个 API 接口异常时，黑盒监控可以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。若需使用此场景，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，可以选择 ICMP、TCP 或 HTTP 探测方式，定期探测指定的目标地址。

前提条件

监控组件必须已安装在集群中，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	**ICMP**：通过 ping 输入的 目标地址（域名或 IP）探测服务器可用性。 **TCP**：通过监听 目标地址 中指定的 `<域名:端口>` 或 `<IP:端口>` 探测主机业务端口。 **HTTP**：探测输入的 目标地址 URL，检查网站连通性。 提示：HTTP 探测方式默认仅支持 GET 请求，若需 POST 请求，请参考 自定义 BlackboxExporter 监控模块。
探测间隔	探测的时间间隔。

参数	说明
目标地址	探测目标地址，最长支持 128 字符。 不同探测方式的输入格式如下： **ICMP**：域名或 IP 地址，例如 `10.165.94.31`。 **TCP**：`<域名:端口>` 或 `<IP:端口>`，例如 `172.19.155.133:8765`。 **HTTP**：以 http 或 https 开头的 URL，例如 `http://alauda.cn/`。

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

- 监控组件必须已安装在集群中，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：请选择所需的黑盒监控项。
  - 指标名称：请选择希望监控并触发告警的指标。平台当前支持的指标为 **Connectivity** 和 **HTTP Status Code**。
    - **Connectivity**：所有黑盒监控项均可选择该指标，触发条件为 **“!= 1”** 表示黑盒监控项的目标地址不可达。
    - **HTTP Status Code**：仅当所选黑盒监控项的探测方式为 **HTTP** 时可选。可输入三位正整数作为触发条件值，例如设置为 **“> 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，增强黑盒监控功能。例如，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，可通过平台的管理功能 集群管理 > 资源管理 快速查看和更新。

操作流程

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

   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. 通过以下任一方式激活配置。

   - 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：

```yaml
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 # 探针目标地址
      labels:
        module: http_2xx # 探针项模块名称
        prober: http # 探测方式
  interval: 30s # 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：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global # 告警所属集群名称
    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，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 # 告警所属集群名称
            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
            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，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 YAML 配置文件的完整示例如下：

```
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
          method: GET                 # 请求方法
          headers:                    # 请求头
            Host: vhost.example.com
            Accept-Language: en-US
            Origin: example.com
          no_follow_redirects: false  # 是否允许重定向
          fail_if_ssl: false
          fail_if_not_ssl: false
          fail_if_body_matches_regexp:
            - "Could not connect to database"
          fail_if_body_not_matches_regexp:
            - "Download the latest version here"
          fail_if_header_matches: # 验证未设置 Cookie
            - header: Set-Cookie
              allow_missing: true
              regexp: '.*'
          fail_if_header_not_matches:
            - header: Access-Control-Allow-Origin
              regexp: '(\*|example\.com)'
          tls_config:                  # https 请求的 TLS 配置
            insecure_skip_verify: false
          preferred_ip_protocol: "ip4" # 默认为 "ip6"                 # 优先使用的 IP 协
议版本
          ip_protocol_fallback: false  # 不回退到 "ip6"
      http_post_2xx:                  # 带请求体的 HTTP 探测示例
        prober: http
        timeout: 5s
        http:
          method: POST                # 探测请求方法
          headers:
            Content-Type: application/json
```

```
      body: '{"username":"admin","password":"123456"}'                    # 探测时携带
的请求体
    http_basic_auth_example:          # 带用户名密码的探测示例
      prober: http
      timeout: 5s
      http:
        method: POST
        headers:
          Host: "login.example.com"
        basic_auth:                   # 探测时添加的用户名和密码
          username: "username"
          password: "mysecret"
    http_custom_ca_example:
      prober: http
      http:
        method: GET
        tls_config:                   # 探测时指定使用的根证书
          ca_file: "/certs/my_cert.crt"
    http_gzip:
      prober: http
      http:
        method: GET
        compression: gzip             # 探测时使用的压缩方式
    http_gzip_with_accept_encoding:
      prober: http
      http:
        method: GET
        compression: gzip
        headers:
          Accept-Encoding: gzip
    tls_connect:                      # TCP 探测示例
      prober: tcp
      timeout: 5s
      tcp:
        tls: true                     # 是否使用 TLS
    tcp_connect_example:
      prober: tcp
      timeout: 5s
    imap_starttls:                    # 配置 IMAP 邮件服务器探测示例
      prober: tcp
      timeout: 5s
      tcp:
        query_response:
          - expect: "OK.*STARTTLS"
```

```
      - send: ". STARTTLS"
      - expect: "OK"
      - starttls: true
      - send: ". capability"
      - expect: "CAPABILITY IMAP4rev1"
  smtp_starttls:                    # 配置 SMTP 邮件服务器探测示例
    prober: tcp
    timeout: 5s
    tcp:
      query_response:
      - expect: "^220 ([^ ]+) ESMTP (.+)$"
      - send: "EHLO prober\r"
      - expect: "^250-STARTTLS"
      - send: "STARTTLS\r"
      - expect: "^220"
      - starttls: true
      - send: "EHLO prober\r"
      - expect: "^250-AUTH"
      - send: "QUIT\r"
  irc_banner_example:
    prober: tcp
    timeout: 5s
    tcp:
      query_response:
      - send: "NICK prober"
      - send: "USER prober prober prober :prober"
      - expect: "PING :([^ ]+)"
        send: "PONG ${1}"
      - expect: "^:[^ ]+ 001"
  icmp_example:                     # ICMP 探测示例配置
    prober: icmp
    timeout: 5s
    icmp:
      preferred_ip_protocol: "ip4"
      source_ip_address: "127.0.0.1"
  dns_udp_example:                  # 使用 UDP 的 DNS 查询示例
    prober: dns
    timeout: 5s
    dns:
      query_name: "www.prometheus.io"                 # 解析的域名
      query_type: "A"               # 域名对应的类型
      valid_rcodes:
      - NOERROR
      validate_answer_rrs:
```

```
            fail_if_matches_regexp:
            - ".*127.0.0.1"
            fail_if_all_match_regexp:
            - ".*127.0.0.1"
            fail_if_not_matches_regexp:
            - "www.prometheus.io.\t300\tIN\tA\t127.0.0.1"
            fail_if_none_matches_regexp:
            - "127.0.0.1"
          validate_authority_rrs:
            fail_if_matches_regexp:
            - ".*127.0.0.1"
          validate_additional_rrs:
            fail_if_matches_regexp:
            - ".*127.0.0.1"
      dns_soa:
        prober: dns
        dns:
          query_name: "prometheus.io"
          query_type: "SOA"
      dns_tcp_example:               # 使用 TCP 的 DNS 查询示例
        prober: dns
        dns:
          transport_protocol: "tcp" # 默认为 "udp"
          preferred_ip_protocol: "ip4" # 默认为 "ip6"
          query_name: "www.prometheus.io"
kind: ConfigMap
metadata:
  annotations:
    skip-sync: 'true'
  labels:
    app.kubernetes.io/instance: cpaas-monitor
    app.kubernetes.io/managed-by: Tiller
    app.kubernetes.io/name: prometheus-blackbox-exporter
    helm.sh/chart: prometheus-blackbox-exporter-1.6.0
  name: cpaas-monitor-prometheus-blackbox-exporter
  namespace: cpaas-system
```

Menu

实用指南

Prometheus 监控数据的备份与恢复

功能概述

使用场景

前置条件

操作流程

操作结果

了解更多

后续操作

VictoriaMetrics 监控数据的备份与恢复

功能概述

使用场景

前提条件

操作步骤

操作结果

了解更多

后续操作

从自定义命名的网络接口采集网络数据

Prometheus 监控数据的备份与恢复

目录

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

前置条件

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

操作流程

备份数据

备份开始前请注意：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。

方式一：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

方式二：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 处于 Running 状态后再进行后续操作。

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
- 建议定期进行数据备份
- 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 监控数据的备份与恢复

目录

功能概述

VictoriaMetrics 监控数据的备份与恢复功能允许您定期备份监控数据，并在必要时恢复数据，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止数据丢失

- 系统迁移时的数据迁移
- 灾难恢复
- 重建测试环境数据

前提条件

- 集群中已安装 ACP Monitoring with VictoriaMetrics 插件
- 确保有足够的存储空间用于备份
- 拥有访问 VictoriaMetrics 存储路径的权限

操作步骤

1. 确认存储路径

VictoriaMetrics 的监控数据存储在容器指定路径中，该路径由 `-storageDataPath` 参数指定，默认值为 `/vm-data`。

配置示例：

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

注意：ACP Monitoring with VictoriaMetrics 插件中计算组件的名称为 `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

平台支持通过修改监控组件配置，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。

使用场景

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-zz231204040711-9d1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2
   v3.15.0-zz231204040711-9d1fc12474c2
   ovn-0954f21f0359720e8c115804376b3e7e        ovn        prometheus          prometheus
   Running   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2
   v3.15.0-zz231204040711-9d1fc12474c2
   ```

4. 编辑业务集群对应的 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，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_interface # 将
custom_interface 替换为自定义的正则表达式，以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

- 关于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

Menu

分布式追踪

介绍

介绍

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

核心概念

核心概念

Telemetry

OpenTelemetry

Span

Trace

Instrumentation

OpenTelemetry Collector

Jaeger

操作指南

Query Tracing

Feature Overview

Main Features

Feature Advantages

Tracing Query

Query Result Analysis

查询 Trace 日志

功能概述

核心功能

前提条件

日志查询操作

实用指南

Java 应用中追踪的非侵入式集成

功能概述

使用场景

前置条件

操作步骤

操作结果

业务日志与 TraceID 关联

背景

在 Java 应用日志中添加 TraceID

在 Python 应用日志中添加 TraceID

验证方法

故障排除

无法查询到所需的Tracing

问题描述

根因分析

根因1的解决方案

根因2的解决方案

不完整的追踪数据

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

Distributed Tracing 模块是 ACP 平台可观测性套件的核心组件，提供针对分布式微服务架构的端到端请求跟踪与分析能力。

该模块提供四项关键的追踪功能：

- **Trace collection**，通过 OpenTelemetry 自动注入或 SDK 集成，实现分布式请求数据的自动采集
- **Trace storage**，使用 Elasticsearch 作为后端存储，实现追踪数据的可扩展持久化
- **Trace visualization**，通过定制化 UI 仪表盘和服务依赖关系映射，实现多维度分析
- **Trace querying**，支持使用 TraceID、服务名称、标签及其他丰富搜索条件进行精准搜索和过滤

通过与 OpenTelemetry 标准及开源组件的集成，帮助组织快速定位服务异常，分析性能瓶颈，追踪完整请求生命周期，并优化微服务架构下的分布式系统性能。

目录

使用限制

使用追踪功能时，应注意以下限制：

- 采样策略与性能的平衡

- 在高负载场景下，追踪数据的采集可能会对 Elasticsearch 的性能和存储产生一定压力；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 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：

- 您拥有具有 `platform-admin-system` 权限的账户，可以访问 Alauda Container Platform 集群。
- 您已安装 `kubectl` CLI。
- 已部署 `Elasticsearch` 组件用于存储追踪数据，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**

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 管理员 视图中，选择要部署 Jaeger Operator 的 集群，然后进入 **Marketplace** → **OperatorHub**。
- 使用搜索框搜索目录中的 `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 **Install**。
- 在 **Install** 页面：
  - 选择 **Manual** 作为 **Upgrade Strategy**。对于 `Manual` 审批策略，OLM 会创建更新请求。作为集群管理员，您必须手动批准 OLM 更新请求以升级 Operator 到新版本。
  - 选择 **stable (Default)** 频道。
  - 选择 **Recommended** 作为 **Installation Location**。将 Operator 安装在推荐的 `jaeger-operator` 命名空间中，这样 Operator 可以监控并在集群内所有命名空间中可用。
- 点击 **Install**。
- 确认 **Status** 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 及其组件安装成功。

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，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`，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：

```
CLUSTER_NAME: <cluster>
ES_URL: https://xxx
ES_USER_BASE64: xxx
ES_PASS_BASE64: xxx
TARGET_NAMESPACE: cpaas-system
JAEGER_INSTANCE_NAME: jaeger-prod
JAEGER_BASEPATH_SUFFIX: /acp/jaeger
JAEGER_ES_INDEX_PREFIX: acp-tracing-<cluster>
PLATFORM_URL: https://xxx
configmap/jaeger-prod-oauth2-proxy created
secret/jaeger-prod-oauth2-proxy created
secret/jaeger-prod-es-basic-auth created
serviceaccount/jaeger-prod-sa created
role.rbac.authorization.k8s.io/jaeger-prod-role created
rolebinding.rbac.authorization.k8s.io/jaeger-prod-rb created
jaeger.jaegertracing.io/jaeger-prod created
podmonitor.monitoring.coreos.com/jaeger-prod-monitor created
ingress.networking.k8s.io/jaeger-prod-query created
Jaeger UI access address: <platform-url>/clusters/<cluster>/acp/jaeger
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 管理员 视图中，选择要部署 OpenTelemetry Operator 的 集群，然后进入 **Marketplace** → **OperatorHub**。
- 使用搜索框搜索目录中的 `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 **Install**。

- 在 **Install** 页面：
  - 选择 **Manual** 作为 **Upgrade Strategy**。对于 `Manual` 审批策略，OLM 会创建更新请求。作为集群管理员，您必须手动批准 OLM 更新请求以升级 Operator 到新版本。
  - 选择 **alpha (Default)** 频道。
  - 选择 **Recommended** 作为 **Installation Location**。将 Operator 安装在推荐的 `opentelemetry-operator` 命名空间中，这样 Operator 可以监控并在集群内所有命名空间中可用。
- 点击 **Install**。
- 确认 **Status** 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   REPLACES   PHASE
  openTelemetry-operator.vx.x.0   OpenTelemetry Operator   x.x.0
  Succeeded
  ```

  如果 `PHASE` 字段显示为 `Succeeded`，表示 Operator 及其组件安装成功。

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`，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，需要您在 **Feature Switch** 视图中手动启用 `acp-tracing-ui` 功能开关。

然后，进入 **Container Platform** 视图，导航至 **Observability** → **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 管理员 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 **Uninstall** 按钮。
- 在弹出的 **Uninstall "opentelemetry-operator"?** 窗口中点击 **Uninstall**。

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-prod-query
kubectl -n cpaas-system delete podmonitor jaeger-prod-monitor
kubectl -n cpaas-system delete jaeger jaeger-prod
kubectl -n cpaas-system delete rolebinding jaeger-prod-rb
kubectl -n cpaas-system delete role jaeger-prod-role
kubectl -n cpaas-system delete serviceaccount jaeger-prod-sa
kubectl -n cpaas-system delete secret jaeger-prod-oauth2-proxy
kubectl -n cpaas-system delete secret jaeger-prod-es-basic-auth
kubectl -n cpaas-system delete configmap jaeger-prod-oauth2-proxy
```

卸载 Jaeger Operator

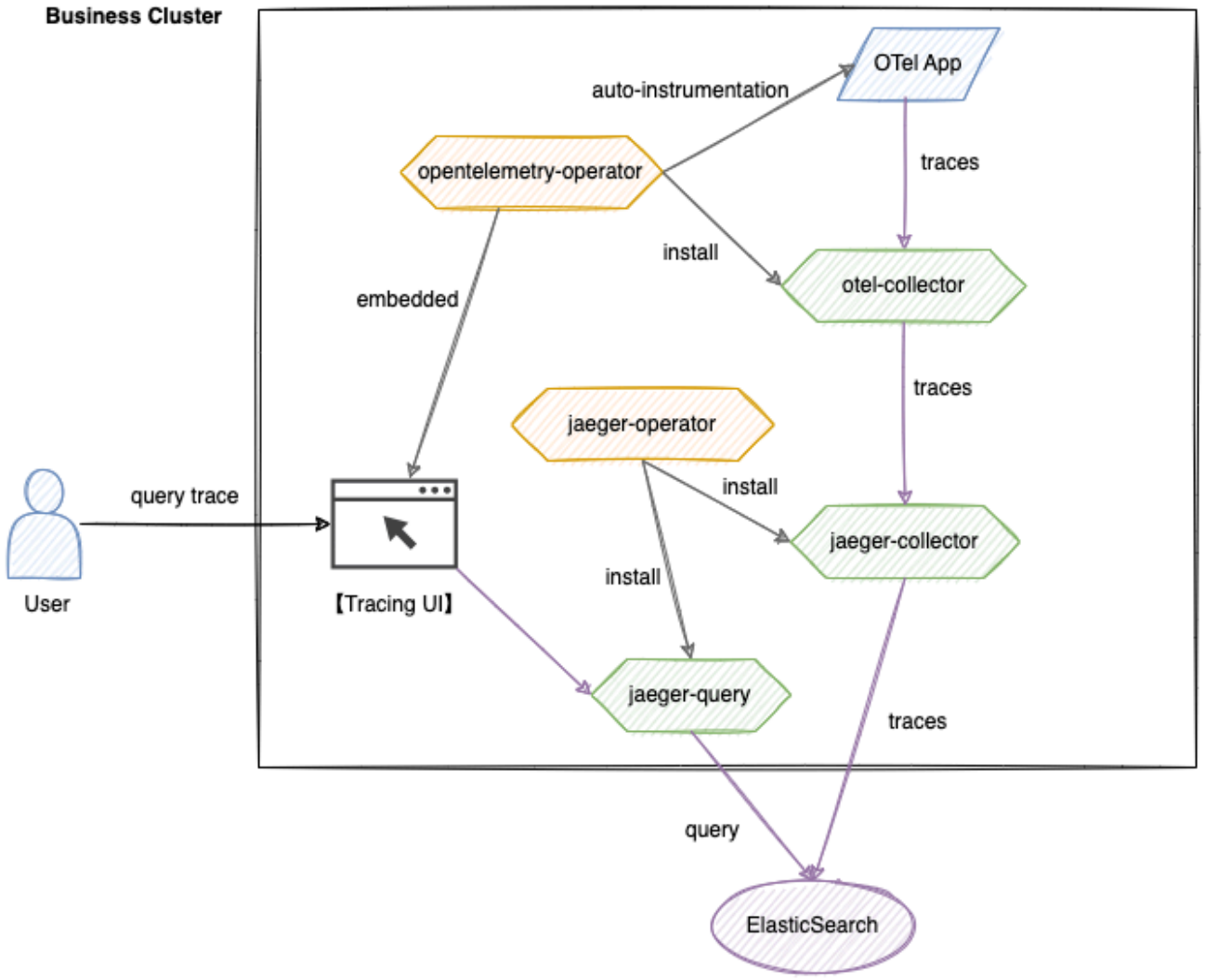
您可以通过 Web 控制台的 管理员 视图卸载 Jaeger Operator。

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 **Uninstall** 按钮。
- 在弹出的 **Uninstall "jaeger-operator"?** 窗口中点击 **Uninstall**。

架构

该架构基于 OpenTelemetry 和 Jaeger 技术栈，实现了分布式追踪的全生命周期管理。系统由数据采集、传输、存储、查询和可视化五个核心模块组成。



目录

核心组件

1. **OpenTelemetry** 系统

   - **opentelemetry-operator**
     一个集群级别的 Operator，负责部署和管理 otel-collector 组件，提供 OTel 自动注入能力。
   - **otel-collector**
     接收来自应用的追踪数据，对其进行过滤和批处理，然后转发给 jaeger-collector。
   - **Tracing UI**
     自研的可视化界面，集成了 jaeger-query API，支持多维度查询条件。

2. **Jaeger** 系统

   - **jaeger-operator**
     部署和管理 jaeger-collector 及 jaeger-query 组件。
   - **jaeger-collector**
     接收 otel-collector 转发和处理后的追踪数据，进行格式转换，并写入 Elasticsearch。
   - **jaeger-query**
     提供追踪查询 API，支持包括 TraceID 和标签在内的多条件检索。

3. 存储层

   - **Elasticsearch**
     一个分布式存储引擎，支持对海量 Span 数据的高效写入和检索。

数据流

- 写入流程
  `Application -> otel-collector -> jaeger-collector -> Elasticsearch`

  应用通过 SDK 或自动注入生成 Span 数据，otel-collector 对数据进行规范化处理，随后由 jaeger-collector 持久化写入 Elasticsearch。

- 查询流程

  `User -> Tracing UI -> jaeger-query -> Elasticsearch`

  用户通过 UI 提交查询条件，jaeger-query 从 Elasticsearch 中检索数据；UI 根据返回的数据进行可视化展示。

核心概念

目录

Telemetry

Telemetry 指系统及其行为发出的数据，包括 traces、metrics 和 logs。

OpenTelemetry

OpenTelemetry 是一个可观测性↗框架和工具包，旨在创建和管理诸如traces↗、metrics↗和logs↗等 telemetry 数据。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协同工作，包括开源工具如Jaeger↗和Prometheus↗，以及商业产品。

Span

Span 是分布式追踪的基本构建块，表示一个具体的操作或工作单元。每个 span 记录请求中的具体动作，帮助我们了解操作执行过程中发生的细节。

一个 span 包含名称、时间相关数据、结构化日志消息以及其他元数据（attributes），这些共同描绘了操作的完整图景。

Trace

Trace 记录请求（无论是来自原生应用还是终端用户）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 trace 由一个或多个 spans 组成。第一个 span 被称为根 span，代表请求从开始到结束的整个生命周期。根 span 之下的子 spans 提供了关于请求过程（或构成请求的各个步骤）的更详细上下文信息。

没有 traces，识别分布式系统中性能问题的根本原因将非常困难。traces 通过拆解请求在系统中的流动，使调试和理解分布式系统变得更加容易。

Instrumentation

为了实现可观测性，系统需要进行**Instrumentation**：即系统组件代码必须发出traces↗、metrics↗和logs↗。

使用 OpenTelemetry，你可以通过两种主要方式对代码进行 instrumentation：

1. 基于代码的解决方案↗：使用官方提供的大多数语言的 API 和 SDK↗
2. 零代码 instrumentation 解决方案↗

基于代码的解决方案能够从应用内部提供更深入和更丰富的 telemetry 数据。你可以使用 OpenTelemetry API 在应用中生成 telemetry 数据，这是对零代码 instrumentation 解决方案生成的 telemetry 数据的重要补充。

零代码 **instrumentation** 解决方案非常适合快速入门或当你无法修改需要采集 telemetry 数据的应用时使用。它们可以通过你使用的库或运行时环境提供丰富的 telemetry 数据。另一种理解方式是，它们提供关于应用边界（Edges）上发生事件的信息。

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智能体，能够接收、处理和导出 telemetry 数据。它支持接收多种格式的 telemetry 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。同时，它还支持在导出前对 telemetry 数据进行处理和过滤。

更多信息请参见 Collector↗。

Jaeger

Jaeger 是一个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 traces、分析性能瓶颈和排查异常。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，是云原生领域的重要可观测性工具。

Menu

操作指南

Query Tracing

Feature Overview

Main Features

Feature Advantages

Tracing Query

Query Result Analysis

查询 Trace 日志

功能概述

核心功能

前提条件

日志查询操作

Query Tracing

目录

Feature Overview

分布式追踪查询功能通过采集服务间调用的元数据信息，为微服务架构提供全链路追踪能力，帮助用户快速定位跨服务调用问题。该功能主要解决以下问题：

- 请求链路追踪：还原复杂分布式系统中的完整请求路径。
- 性能瓶颈分析：定位链路中耗时异常的调用节点。
- 故障根因定位：通过错误标记快速定位问题发生点。

适用场景包括：

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

核心价值：

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

Main Features

- 多维度查询：支持 TraceID、服务名、标签等 6 种查询条件组合。
- 可视化分析：通过时间序列瀑布图直观展示调用层级和时间分布。
- 精准定位：支持错误 Span 过滤及标签二次搜索。

Feature Advantages

- 快速定位问题：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：使用时间序列瀑布图直观展示调用关系，降低复杂度，提高故障分析效率。
- 灵活多样：支持简单查询与复杂组合，适应多种运维和开发场景。

Tracing Query

1 Step 1: Combine Query Conditions

提示：查询条件可以组合使用。您可以通过添加多个查询条件来细化查询。

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信息。
Service	发起/接收调用请求的服务（需选择或输入）。
Label	可通过输入标签（Tag）过滤查询结果，支持的标签包括 Span 详情中的标签。
Span Duration Greater Than	查询持续时间大于等于 *输入值*（毫秒）的 Span。
Only Search Error Spans	错误 Span 指 Tag 中 **error** 值为 `true` 的 Span。
Span Type	**Root Span**：搜索由配置的 **service** 发起的根 Span，适用于配置服务为整个调用请求发起方的场景。 **Service Entry Span**：搜索配置的 **service** 作为服务端被调用时生成的第一个 Span。
Maximum Query Count	最大可查询的 Span 数量，默认 200。 提示：出于性能考虑，平台单次最多展示 1000 个 Span。如满足查询条件的 Span 数量超过最大查询数，可通过细化查询条件或缩小时间范围分阶段查询。

2 Step 2: Execute Query

- 选择查询条件并输入对应值后，点击 **Add to Query Conditions** 按钮，当前条件将显示在 **Query Conditions** 结果区，并触发查询。
- 也可展开 **Common Query Conditions** 快速添加最近使用的查询条件。

Query Result Analysis

输入查询条件并搜索后，页面将生成查询结果区域。

Span List

查询结果区域左侧展示符合条件的 Span 列表及其基本信息，包括：服务名、被调用接口或请求处理方法、持续时间和开始时间。

Time-Series Waterfall Chart

查询结果区域右侧的时间序列瀑布图清晰展示单条追踪中 Span 之间的调用关系。时间序列瀑布图在追踪分析中的主要特点如下：

1. 自上而下展开：时间序列瀑布图中，各调用事件（Span）通常自图表顶部向下展开，每个横条代表一次服务调用或处理，位置一般反映操作的逻辑调用顺序。
2. 时间轴对齐：瀑布图横轴表示时间，每个横条长度代表该调用的持续时间，便于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，缩进越深表示该调用在链路中的调用深度越大。
4. 交互与详细数据展示：点击瀑布图中的横条可显示该调用的更多详细信息。

Span Details

点击时间序列瀑布图中某个 Span 行，可展开查看该 Span 的详细信息，包括：

- Service：Span 所属服务。
- Span Duration (ms)：Span 持续时间。
- URL：服务访问的 URL，对应 Span 标签中的 **http.url**。
- Tag：由键值对组成的 Span 标签信息，可用于高级搜索标签查询条件。点击标签旁按钮，可将当前标签条件添加至查询条件，实现更精准的查询结果。
- JSON：Span 的原始 JSON 结构，便于进一步查看其内部信息。

查询 **Trace** 日志

目录

功能概述

Trace 日志使用户能够通过唯一的 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，了解请求流程，并将业务日志与追踪上下文关联起来。

主要优势：

- 根因分析：识别分布式系统中跨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关联，实现端到端可视化。
- 高效过滤：通过 Pod 或 TraceID 过滤日志，聚焦相关数据。

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满足合规要求。

核心功能

- 基于 **TraceID** 查询：使用 TraceID 检索与特定追踪相关的所有日志。
- 以 **Pod** 为中心的过滤：查看参与追踪的特定 Pod 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：查看目标日志条目前后记录，进行深入分析。

前提条件

> **TIP**
>
> 在通过 TraceID 查询 Trace 日志之前，必须先对服务进行埋点，使业务日志中包含 TraceID。请参阅业务日志与 TraceID 关联指南进行配置。

默认行为：

- 显示整个追踪时长内的日志。
- 对于时长不足 1 分钟的追踪，查询追踪开始时间后 1 分钟内的日志。

日志查询操作

1 访问 **Trace** 日志

1. 查询 Trace 后，点击具体 Trace 查看详情。
2. 点击追踪可视化面板中的 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，选择目标时间段。

按查询条件

在 查询条件 文本框中输入关键词，根据特定内容过滤日志。

包含 **Trace ID**

选中 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。 切换日志字段在显示面板中的可见性。

查看日志上下文

1. 点击任意日志条目旁的 洞察。
2. 查看目标时间戳前后各 5 条日志。
3. 使用鼠标上下滚动加载更多日志。

Menu

实用指南

Java 应用中追踪的非侵入式集成

功能概述

使用场景

前置条件

操作步骤

操作结果

业务日志与 TraceID 关联

背景

在 Java 应用日志中添加 TraceID

在 Python 应用日志中添加 TraceID

验证方法

Java 应用中追踪的非侵入式集成

INFO

自动注入的 OpenTelemetry Java Agent↗ 支持 `Java 8+` 版本。

目录

功能概述

追踪是分布式系统可观测性的核心能力，能够完整记录系统内请求的调用路径和性能数据。本文介绍如何通过自动注入 OpenTelemetry Java Agent，实现 Java 应用的非侵入式追踪集成。

使用场景

Java 应用可集成于以下场景：

- 快速为 Java 应用添加追踪能力

- 避免修改应用源代码
- 使用 Kubernetes 部署服务
- 可视化服务间调用关系，分析性能瓶颈

前置条件

使用该功能前，请确保：

- 目标服务已部署在 Alauda Container Platform 上
- 服务使用的 JDK 版本为 Java 8 或更高
- 具备目标命名空间中 Deployment 的编辑权限
- 平台已完成追踪部署

操作步骤

对于需要集成到 Alauda Container Platform 追踪的 Java 应用，需进行如下适配：

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

Deployment 适配示例：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-java-deploy
spec:
  template:
    metadata:
      annotations:
        instrumentation.opentelemetry.io/inject-java: cpaas-system/acp-common-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，可通过标签关联或固定值设置。
3. 配置 `SERVICE_NAMESPACE` 环境变量，值为 `metadata.namespace`。

操作结果

Java 应用适配完成后：

- 新启动的 Java 应用 Pod 中若包含 `opentelemetry-auto-instrumentation-java` init 容器，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。

- 进入 **Observability** -> **Tracing** 页面，查看 Java 应用的追踪数据和时间线瀑布图。

TIP

本文将指导开发者如何在应用代码中集成获取 **TraceID**和将 **TraceID** 添加到应用日志的方法，适合具有一定开发经验的后端开发者。

业务日志与 **TraceID** 关联

目录

背景

- 为了将单次请求过程中自动发送的多个 span（不同模块/节点/服务调用）正确关联成一个完整的 trace，服务的 HTTP 请求头中会携带 TraceID 及其他用于关联 trace 的信息。
- 一个 trace 代表一次请求的调用过程，TraceID 是标识该请求的唯一 ID。日志中有了 TraceID，便可以将 trace 与应用日志关联起来。

基于以上背景，本文将说明如何从 HTTP 请求头中获取 TraceID 并将其添加到应用日志中，从而使您能够在平台上通过 TraceID 精准查询日志数据。

在 **Java** 应用日志中添加 **TraceID**

TIP

- 以下示例基于 **Spring Boot** 框架，使用 **Log4j** 和 **Logback** 进行说明。
- 您的应用需满足以下前置条件：
  - 日志库类型及版本需满足以下要求：

日志库	版本要求
Log4j 1	1.2+
Log4j 2	2.7+
Logback	1.0+

  - 应用已注入 Java Agent。

方法一：配置 `logging.pattern.level`

在应用配置中修改 `logging.pattern.level` 参数如下：

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

方法二：配置 `CONSOLE_LOG_PATTERN`

1. 修改 logback 配置文件如下。

TIP

此处以控制台输出为例，`%X{trace_id}` 表示从 MDC 中获取 key 为 `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 log 对象输出日志，示例如下：

```
@RestController
@Slf4j
public class ProviderController {

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

在 **Python** 应用日志中添加 **TraceID**

1. 在应用代码中添加如下代码，从请求头中获取 TraceID。示例代码如下，可根据需要调整：

> **TIP**
>
> **getForwardHeaders** 函数从请求头中获取 trace 信息，其中 `x-b3-traceid` 的值即为 TraceID。

```
    def getForwardHeaders(request):
        headers = {}
        incoming_headers = [
            'x-request-id',  # 所有应用均应传递 x-request-id，用于访问日志和一致的
trace/log 采样决策
            'x-b3-traceid',  # B3 trace 头，兼容 Zipkin、OpenCensusAgent 和 Stackdriver
配置
            '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. 在应用代码中添加如下代码，将获取到的 TraceID 添加到日志中。示例代码如下，可根据需要调整：

```
headers = getForwardHeaders(request)
tracing_section = ' [%(x-b3-traceid)s,%(x-b3-spanid)s] ' % headers
logging.info(tracing_section + "Oops, unexpected error happens.")
```

验证方法

1. 点击左侧导航栏的 **Tracing**。
2. 在查询条件中选择 TraceID，输入要查询的 TraceID，点击 **Add to query**。
3. 在下方展示的 trace 数据中，点击 TraceID 旁的 **View Log**。
4. 在 **Log Query** 页面，勾选 **Contain Trace ID**，系统将仅展示包含该 TraceID 的日志数据。

Menu

故障排除

无法查询到所需的**Tracing**

问题描述

根因分析

根因1的解决方案

根因2的解决方案

不完整的追踪数据

问题描述

根因分析

根因 1 的解决方案

根因 2 的解决方案

无法查询到所需的Tracing

目录

问题描述

在服务网格中查询Tracing时，可能会遇到无法获取目标Tracing的情况。

根因分析

1. Tracing采样率配置过低

当Tracing的采样率参数设置过低时，系统只会按比例采集Tracing数据。在请求量不足或低峰期时，可能导致采样数据低于可见阈值。

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刷新到可搜索状态存在10秒的延迟。查询最近生成的Tracing时，可能因数据尚未持久化而出现结果缺失。

该索引配置可以有效降低Elasticsearch的数据合并压力，提高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（tail sampling）。

根因**2**的解决方案

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

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

注意：设置为 `"null"` 会影响Elasticsearch的性能和查询速度。

不完整的追踪数据

目录

问题描述

追踪查询结果存在以下不完整数据的问题：

- 最近的查询（过去 30 分钟内）缺少部分 span。
- 超过 1 小时的追踪出现断连现象。

根因分析

1. 数据持久化延迟

Elasticsearch 的写入过程涉及内存缓冲区 → translog → segment 文件的顺序步骤，可能导致最新写入的数据存在可见性延迟。

2. 时间范围限制

默认情况下，`jaeger-query` 查询与追踪对应的 span 时，时间范围会延伸到 span 开始时间的前后各一小时。

例如，若某个 span 的开始时间为 `08:12:30`，结束时间为 `08:12:32`，则查询该追踪的时间范围为 `07:12:30` 至 `09:12:32`。

因此，如果追踪跨度超过 1 小时，通过该 span 查询可能无法得到完整的追踪。

根因 1 的解决方案

等待片刻后刷新页面，重新尝试查询。

根因 2 的解决方案

如果您的环境中追踪跨度较长，可以通过在 `jaeger-query` 启动时使用 `--es.asm.span-trace-query-time-adjustment-hours` 参数来调整单个追踪的查询时间范围。

该参数的默认值为 1 小时，您可以根据需要增加该值。

Menu

日志

关于 **Logging Service**

关于 Logging Service

Logging 模块是 ACP 平台可观测性套件的核心组件，提供全面的日志管理能力，实现高效且可靠的日志处理。

该模块提供四项关键的日志功能：

- 日志采集，实现对应用程序、容器和基础设施组件日志的自动收集
- 日志存储，基于 ElasticSearch 和 ClickHouse 后端，支持可扩展且持久的日志保存
- 日志查询，支持对大量日志数据的快速且灵活的搜索

通过与 Filebeat、ElasticSearch 和 ClickHouse 等强大的开源组件集成，该模块使组织能够高效处理海量日志，加快故障排查，确保合规性要求，并实时获得宝贵的运营洞察。

Note

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

Menu

事件

介绍

使用限制

Events

操作流程

事件概览

介绍

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化。它还提供存储、查询和可视化的能力。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。

基于从事件中识别出的根本原因，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，可以自动触发告警，通知相关人员及时干预，从而降低平台的运维风险。

目录

使用限制

该功能依赖于日志服务。请确保平台内已预先安装 ACP Log Essentails、ACP Log Collector 和 ACP Log Storage 插件。

Note

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

Events

目录

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（可选择或自定义时间范围），以及资源事件记录。

- 重要事件概览：该卡片展示选定时间范围内重要事件的原因及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：节点重启事件的资源数量为 20，表示在选定时间范围内，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，展示选定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，或展开视图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，如 Pod。
  - 命名空间：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`，该事件可忽略；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。

Menu

检查

介绍

介绍

使用限制

架构

架构

Inspection

Component Health Status

操作指南

Inspection

执行巡检

巡检配置

巡检报告说明

Component Health Status

Procedures to Operate

介绍

Inspection 模块是 ACP 平台可观测性套件的核心组件，提供自动化的检查和评估能力，实现对资源的全面监控和风险管理。

该模块提供四项关键的检查功能：

- 资源检查，对集群、节点、Pod、证书及其他平台资源进行自动评估，以识别风险和使用模式
- 实时监控，实时跟踪检查任务进度，立即了解资源的运行状态
- 可视化报告，直观展示检查结果，包括资源风险、使用信息和运行洞察
- 报告生成，支持以 PDF 或 Excel 格式下载检查报告，包含全面的分析和建议

通过将这些能力与基于角色的访问控制和自动评估算法相结合，帮助组织降低人工检查成本，主动识别资源异常，规避业务风险，并通过系统化的健康评估保持平台的最佳性能。

目录

使用限制

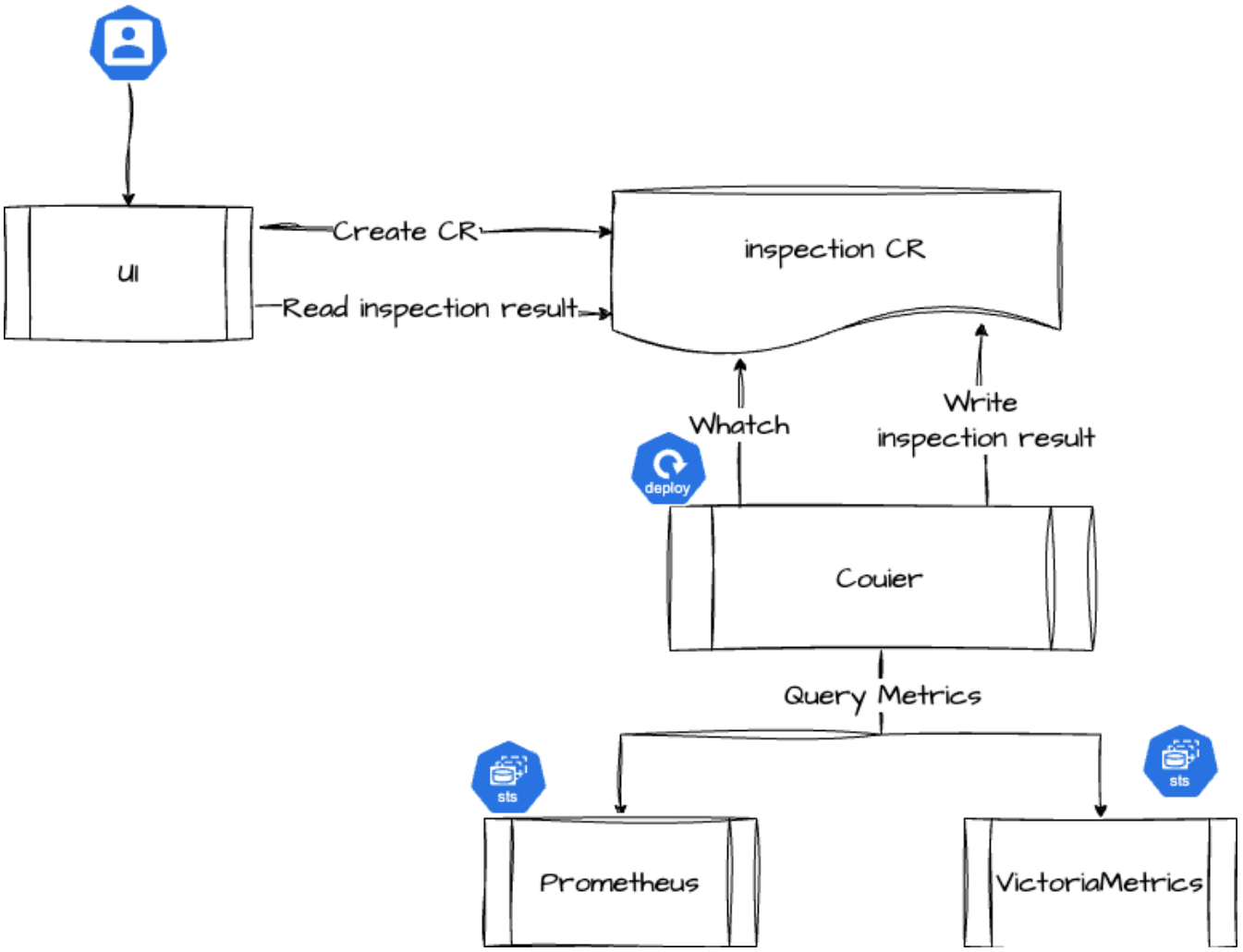
- 平台部分检查项依赖于集群已安装监控组件。请确保每个集群事先安装了 ACP Monitoring with Prometheus 插件或 ACP Monitoring with VictoriaMetrics 插件之一。
- 平台检查支持通过邮件发送检查结果。请确保邮件通知服务器配置已提前完成。

借助容器平台的检查功能，用户能够更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

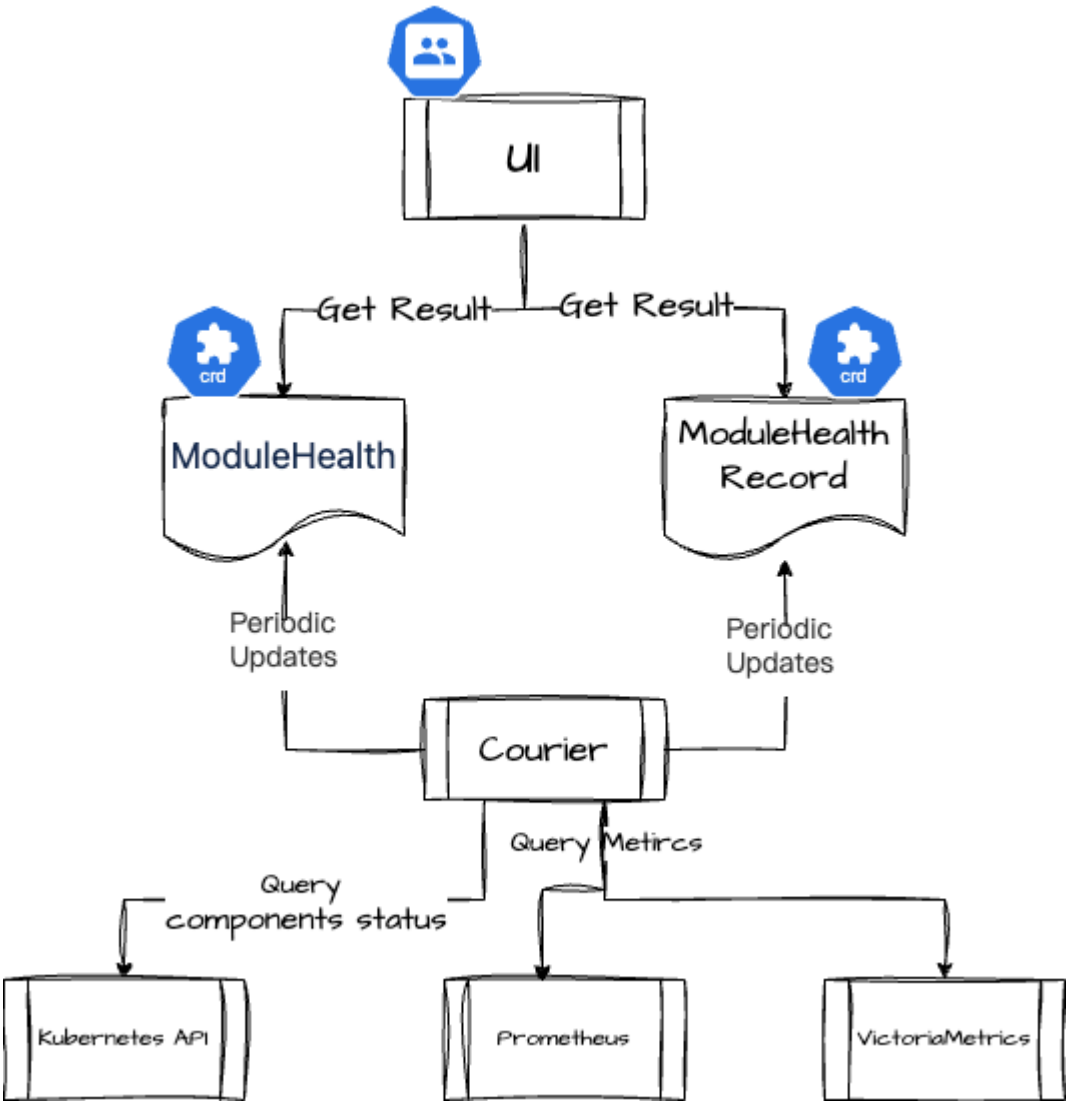
Inspection



Inspection 模块由平台组件 Courier 和监控组件共同提供，涉及以下业务流程：

- 创建 inspection 任务：平台向 `global` 集群提交 inspection 类型的 CR。
- 执行 inspection 任务：Courier 组件监控 inspection 类型 CR 的生成，并向各集群的监控组件查询与 inspection 相关的各类指标数据。
- 写入 inspection 结果：Courier 组件完成对各 inspection 项的评估后，将 inspection 结果写回对应的 inspection CR。
- 查看 inspection 结果：用户可通过平台查看 inspection 任务的状态和结果，数据来源于对应的 inspection CR。

Component Health Status



Component Health Status 由平台组件 Courier 和监控组件共同提供，涉及以下业务流程：

- 预定义组件监控列表：平台在 `global` 集群预定义了两类 CRD，用于定义需监控的组件列表及监控方式：
  - ModuleHealth：定义需监控的组件及监控方式。
  - ModuleHealthRecord：定义各集群对应组件的监控结果。
- 定期监控组件状态：Courier 会 watch ModuleHealth，检查指定功能，然后将 inspection 结果写入 ModuleHealth 和 ModuleHealthRecord 的 CR 资源。
- 组件状态判定：Courier 会请求 Kubernetes 和监控组件的数据，以判定组件的实际状态及存在的问题。
  - Kubernetes：检查组件是否安装及组件副本数是否正常。
  - Prometheus / VictoriaMetrics：基于各组件提供的指标，查询并判定组件是否能正常提供服务。

- 查看组件健康状态：用户可通过平台查看各组件的健康状态，数据来源于对应的 ModuleHealth 和 ModuleHealthRecord CR 资源。

Menu

操作指南

Inspection

执行巡检

巡检配置

巡检报告说明

Component Health Status

Procedures to Operate

Inspection

目录

执行巡检

1. 点击左侧导航栏中的 **Operation Center** > **Inspection** > **Basic Inspection**。

   提示：巡检页面展示最近一次巡检的巡检数据信息。巡检过程中，可实时查看已完成巡检的资源数据。

2. 在 Basic Inspection 页面，支持以下操作：

   - 执行巡检：点击页面右上角的 **Inspection** 按钮，对平台进行巡检。
   - 下载巡检报告：点击页面右上角的 **Download Report** 按钮，在弹出对话框中选择报告格式（PDF 和 Excel），点击下载，即可将对应格式的报告下载到本地。
     - PDF 格式的巡检报告不包含资源风险详情页数据；

- Excel 格式的巡检报告包含巡检的全部数据；
- 支持同时下载两种格式的报告。

巡检配置

巡检配置项	说明
定时巡检	自动任务执行时间规则，支持输入 Crontab 表达式。 提示：点击输入框展开平台预置的 触发规则模板，选择合适模板，简单修改即可快速设置触发规则。
巡检记录保留条数	保留的巡检记录条数。
邮件通知	选择邮件通知联系人。 注意：通知联系人必须配置了邮箱。
巡检报告名称	平台内置巡检通知模板用于通知联系人的名称。
巡检配置项	根据平台默认的证书、集群主机和 pod 巡检项，修改告警阈值或禁用巡检项。

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和结束时间。
- 巡检资源总数：最近一次巡检中巡检的资源总数（集群、节点、pod、证书）。
- 风险数：存在风险的资源数量，包括被判定为 **Fault** 和 **Warning** 的资源数。

资源风险巡检

在 资源风险巡检 页面，可查看 `global` 集群、自建集群、接入集群及这些集群下所有节点、pod 和证书的风险信息概览。

点击对应资源类型卡片上的 风险详情 按钮（**Cluster**、**Node**、**pod**、**Certificate**），进入该资源类型的风险详情页面。详情页中可查看该资源的最近一次巡检信息，以及故障和告警资源列表。

- 点击资源名称可跳转至资源详情页面。
- 点击列表中 **Name** 字段右侧的展开按钮，可展开故障和告警的判断条件及原因。

各资源风险状态判断标准（Fault、Warning）说明见下表。

注意：每种资源类型的故障和告警判断条件有多个，资源巡检数据满足任一判断条件即视为一条风险数据。

资源类型	巡检范围	故障判断条件	告警判断条件
Cluster	- `global` 集群 - 自建集群 - 接入集群	- 集群状态为 **Abnormal**； - apiserver 连接异常	- 集群规模（节点数/pod 数/指标数）增加后，监控组件资源配置未更新； - 日志数据量和日志采集频率增加后，日志组件资源配置未更新； - 集群 CPU 使用率超过 60%； - 集群内存使用率超过 60%； - 集群 ETCD 组件中任一 pod 处于非 Running 状态； - 集群中任一主机处于非 Ready 状态； - 集群中任意两个节点时间差超过 40 秒； - 集群 CPU 请求率（实际请求值/总量）超过 60%； - 集群内存请求率（实际请求值/总量）超过 80%； - 集群未安装监控组件；

资源类型	巡检范围	故障判断条件	告警判断条件
			- 集群监控组件异常； - 集群 **kube-controller-manager** 组件中任一 pod 处于非 Running 状态； - 集群 **kube-scheduler** 组件中任一 pod 处于非 Running 状态； - 集群 **kube-apiserver** 组件中任一 pod 处于非 Running 状态。
Node	- 所有控制节点 - 所有计算节点	- 节点状态为 **Abnormal**； - 节点上的 **node-exporter** 组件 pod 处于非 Running 状态； - 节点上的 **kubelet** 组件 pod 处于非 Running 状态。	- 节点 inode 空闲数少于 1000； - 节点 CPU 使用率超过 60%； - 节点内存使用率超过 60%； - 节点目录磁盘空间使用率超过 60%； - 节点系统负载超过 200%，且持续时间超过 15 分钟； - 过去一天内至少发生一次 NodeDeadlock（节点死锁）事件； - 过去一天内至少发生一次 NodeOOM（内存不足）事件； - 过去一天内至少发生一次 NodeTaskHung（任务挂起）事件； - 过去一天内至少发生一次 NodeCorruptDockerImage（Docker 镜像损坏）事件。
pod	所有 pod	- pod 状态为 **Error**； - pod 处于启动状态超过 5 分钟。	- Pod CPU 使用率超过 80%； - Pod 内存使用率超过 80%； - 过去 5 分钟内 Pod 重启次数大于等于 1 次。
Certificate	- Certmanager	证书状态为 **Expired**。	证书有效期少于 29 天。

资源类型	巡检范围	故障判断条件	告警判断条件
	证书 - Kubernetes 证书		

资源利用率巡检

点击 **Resource Utilization Inspection** 标签，进入 资源利用率巡检 页面。

在 资源利用率巡检 页面，可查看 `global` 集群、接入集群、自建集群的 CPU、内存、磁盘的总量、使用量及使用率，以及平台上集群、节点、pod、项目等资源数量。

- 资源使用统计：可查看 global、接入、自建集群的 CPU、内存、磁盘的总量及总使用率。
- 平台资源数量：可查看平台上运行的资源数量。

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。当您的账户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，以及与该功能相关的集群内组件的检测数据。这有助于您快速识别问题，提高平台的运维效率。

目录

Procedures to Operate

1. 导航至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；功能卡片显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，页面右侧将展开详细健康状态页面，您可以查看故障组件的详细问题信息。