Observability

Overview

Overview

Monitoring

Introduction

Install

Overview

Architectur

Concepts

Monitoring

Alarms

Notifications

Monitoring Dashboard

Guides

ML

How To

Distributed Tracing

Introduction

Usage Limitations

Install

Installing the Jaeger Operator

Deploying a Jaeger Instance

Architectur

Core Compone

Data Flow

Concepts

Telemetry

OpenTelemetry

Span

Trace

Instrumentation

OpenTelemetry Collector

Jaeger

Guides

How To

Troublesho

Logs

About Logging Service

Events

Introduction

Usage Limitations

Events

Operation Procedures

Event Overview

Inspection

Introduction

Usage Limitations

Architecture

Inspection

Component Health Status

Guides

Overview

The Observability module is a core feature of the ACP platform that provides comprehensive monitoring and observability capabilities for cloud-native applications.

This module integrates four essential observability pillars:

- **Synthetic monitoring (probe)** for proactive endpoint testing
- **Centralized logging** for unified log management and analysis
- **Real-time monitoring** for metrics collection and alerting
- **Distributed tracing** for end-to-end request tracking across microservices

By combining these capabilities into a single platform, it enables organizations to achieve complete visibility into application performance, rapidly diagnose issues, ensure system reliability, and optimize user experience across their entire technology stack.

Monitoring

Introduction

Introduction

Install

Install

Architecture

Monitoring Module Architecture

Overall Architecture Explanation

Monitoring System

Alerting System

Notification System

Monitoring Component Selectic

Important Notes

Component List

Architecture Comparison

Feature Comparison

Installation Scheme Suggestions

Monitor Co

Assumptions ar

Prometheus

VictoriaMetrics

Concepts

Concepts

Monitoring

Alarms

Notifications

Monitoring Dashboard

Guides

Management of Metrics

Viewing Metrics Exposed by Platform Cor

Viewing All Metrics Stored by Prometheus

Viewing All Built-in Metrics Defined by the

Integrating External Metrics

Management of Alert

Function Overview

Key Features

Functional Advantages

Creating Alert Policies via UI

Creating Resource Alerts via CLI

Creating Event Alerts via CLI

Creating Alert Policies via alert Templates

Managemer

Feature Overvie

Key Features

Notification Ser

Notification Cor

Notification Tem

Notification rule

Setting Silence for Alerts

Set Notification

Management of Monitoring Dashboards

Function Overview

Manage Dashboards

Manage Panels

Create Monitoring Dashboards via CLI

Common Functions and Variables

Managemer

Function Overv

Blackbox Monit

Blackbox Alerts

Customizing Bl

Create Blackbo

Reference Infor

How To

Backup and Restore of Prometl

Feature Overview

Use Cases

Prerequisites

Procedures to Operate

Operation Results

Learn More

Next Procedures

VictoriaMetrics Backup and Re

Function Overview

Use Cases

Prerequisites

Procedures

Operation Result

Learn More

Follow-up Actions

Collect Netv

Function Overv

Use Case

Prerequisites

Procedures to C

Operation Resu

Learn More

Subsequent Act

Introduction

The Monitoring module is a core component of the ACP platform's observability suite that provides comprehensive monitoring and alerting capabilities for platform administrators and operations teams.

This module delivers four essential monitoring capabilities:

- **Metrics collection** for real-time performance data gathering from clusters, nodes, applications, and containers
- **Dashboards** for intuitive visualization and analysis of system health and performance trends
- **Alerting** for proactive detection of issues through customizable rules and thresholds
- **Notifications** for timely delivery of alert information to operations personnel

By integrating these capabilities with open-source components like Prometheus and VictoriaMetrics, it enables organizations to maintain system reliability, prevent downtime, reduce operational costs, and ensure optimal performance across their entire infrastructure.

Install

TOC

Overview

The monitoring component serves as the infrastructure for monitoring, alerting, inspection, and health checking functions within the observability module. This document describes how to install the ACP Monitoring with Prometheus plugin or the ACP Monitoring with VictoriaMetrics plugin within a cluster.

Installation Preparation

> **INFO**
>
> Some components of the Monitoring are resource-intensive. We recommend running them on infra nodes and setting nodeSelector and tolerations to ensure they run only on those nodes. If you are evaluating the product and have not provisioned infra nodes, you can remove these settings so the components run on all nodes.
>
> For guidance on planning infra nodes, see Cluster Node Planning.

Before install the monitoring components, please ensure the following conditions are met:

- The appropriate monitoring component has been selected by referring to the Monitoring Component Selection Guide.
- When install in a workload cluster, ensure that the `global` cluster can access port 11780 of the workload cluster.
- If you need to use storage classes or persistent volume storage for monitoring data, please create the corresponding resources in the **Storage** section in advance.

Install the ACP Monitoring with Prometheus Plugin via console

Installation Procedures

1. Navigate to **App Store Management** > **Cluster Plugins** and select the target cluster.
2. Locate the **ACP Monitoring with Prometheus** plugin and click **Install**.

3. Configure the following parameters:

Parameter	Description
Scale Configuration	Supports three configurations: **Small Scale**, **Medium Scale**, and **Large Scale**: - Default values are set based on the recommended load test values of the platform - You can choose or customize quotas based on the actual cluster scale - Default values will be updated with platform versions; for fixed configurations, custom settings are recommended
Storage Type	- **LocalVolume**: Local storage with data stored on specified nodes - **StorageClass**: Automatically generates persistent volumes using a storage class - **PV**: Utilizes existing persistent volumes **Note**: Storage configuration cannot be modified after Installation
Replica Count	Sets the number of monitoring component pods **Note**: Prometheus supports only single-node installation
Parameter Configuration	Data parameters for the monitoring component can be adjusted as needed

4. Click **Install** to complete the installation.

Access Method

Once installation is complete, the components can be accessed at the following addresses (replace `<>` with actual values):

Component	Access Address
Thanos	`<platform_access_address>/clusters/<cluster>/prometheus`
Prometheus	`<platform_access_address>/clusters/<cluster>/prometheus-0`

Component	Access Address
Alertmanager	`<platform_access_address>/clusters/<cluster>/alertmanager`

Install the ACP Monitoring with Prometheus Plugin via YAML

1. Check available versions

Ensure the plugin has been published by checking for ModulePlugin and ModuleConfig resources, in the `global` cluster:

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

This indicates that the ModulePlugin `prometheus` exists in the cluster and version `v4.1.0` is published.

2. Create a ModuleInfo

Create a ModuleInfo resource to install the plugin without any configuration parameters:

```yaml
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
```

Reference for resources settings, example prometheus:

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

For more details, please refer to the Monitor Component Capacity Planning

YAML field reference (Prometheus):

Field path	Description
`metadata.labels.cpaas.io/cluster-name`	Target cluster name where the plugin is installed.
`metadata.labels.cpaas.io/module-name`	Must be `prometheus`.
`metadata.labels.cpaas.io/module-type`	Must be `plugin`.
`metadata.name`	ModuleInfo name (e.g., `<cluster>-prometheus`).
`spec.version`	Plugin version to install.
`spec.config.storage.type`	Storage type: `LocalVolume`, `StorageClass`, or `PV`.

Field path	Description
`spec.config.storage.capacity`	Storage size for Prometheus (Gi). Minimum 30 Gi recommended.
`spec.config.storage.nodes`	Node list when `storage.type=LocalVolume`. Up to 1 node supported.
`spec.config.storage.path`	LocalVolume path when `storage.type=LocalVolume`.
`spec.config.storage.storageClass`	StorageClass name when `storage.type=StorageClass`.
`spec.config.storage.pvSelectorK`	PV selector key when `storage.type=PV`.
`spec.config.storage.pvSelectorV`	PV selector value when `storage.type=PV`.
`spec.replicas`	Replica count; only applicable to `StorageClass` / `PV` types.
`spec.config.components.prometheus.retention`	Data retention days.
`spec.config.components.prometheus.scrapeInterval`	Scrape interval seconds; applies to ServiceMonitors without `interval`.
`spec.config.components.prometheus.scrapeTimeout`	Scrape timeout seconds; must be less than `scrapeInterval`.
`spec.config.components.prometheus.resources`	Resource settings for Prometheus.
`spec.config.components.nodeExporter.port`	Node Exporter port (default 9100).
`spec.config.components.nodeExporter.resources`	Resource settings for Node Exporter.

Field path	Description
`spec.config.components.alertmanager.resources`	Resource settings for Alertmanager.
`spec.config.components.kubeStateExporter.resources`	Resource settings for Kube State Exporter.
`spec.config.components.prometheusAdapter.resources`	Resource settings for Prometheus Adapter.
`spec.config.components.thanosQuery.resources`	Resource settings for Thanos Query.
`spec.config.size`	Monitoring scale: `Small`, `Medium`, or `Large`.
`spec.valuesOverride.ait/chart-kube-prometheus.global.nodeSelector`	Optional. The nodeselector for the Monitoring component. Use this parameter to select nodes for the monitoring component. This parameter is mainly used for infrastructure nodes.
`spec.valuesOverride.ait/chart-kube-prometheus.global.tolerations`	Optional. The tolerations for the Monitoring component. If the node selector selects tainted nodes, use this parameter to specify a taint toleration key, value, and effect. This parameter is mainly used for infrastructure nodes.

3. Verify installation

Since the ModuleInfo name changes upon creation, locate the resource via label to check the plugin status and version:

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

Field explanations:

- `NAME`: ModuleInfo resource name
- `CLUSTER`: Cluster where the plugin is installed
- `MODULE`: Plugin name
- `DISPLAY_NAME`: Display name of the plugin
- `STATUS`: Installation status; `Running` means successfully installed and running
- `TARGET_VERSION`: Intended installation version
- `CURRENT_VERSION`: Version before installation
- `NEW_VERSION`: Latest available version for installation

Install the ACP Monitoring with VictoriaMetrics Plugin via console

Prerequisites

- If only install the VictoriaMetrics agent, ensure that the VictoriaMetrics Center has been installed in another cluster.

Installation Procedures

1. Navigate to **App Store Management** > **Cluster Plugins** and select the target cluster.
2. Locate the **ACP Monitoring with VictoriaMetrics** plugin and click **Install**.
3. Configure the following parameters:

Parameter	Description
Scale Configuration	Supports three configurations: **Small Scale**, **Medium Scale**, and **Large Scale**: - Default values are set based on the recommended load test values of the platform - You can choose or customize quotas based on the actual cluster scale - Default values will be updated with platform versions; for fixed configurations, custom settings are recommended
Install Agent Only	- Off: Install the complete VictoriaMetrics component suite - On: Install only the VMAgent collection component, which relies on the VictoriaMetrics Center
VictoriaMetrics Center	Select the cluster where the complete VictoriaMetrics component has been installed
Storage Type	- **LocalVolume**: Local storage with data stored on specified nodes - **StorageClass**: Automatically generates persistent volumes using a storage class - **PV**: Utilizes existing persistent volumes
Replica Count	Sets the number of monitoring component pods: - LocalVolume storage type does not support multiple replicas - For other storage types, please refer to on-screen prompts for configuration
Parameter Configuration	Data parameters for the monitoring component can be adjusted **Note**: Data may temporarily exceed the retention period before being deleted

4. Click **Install** to complete the installation.

Install the ACP Monitoring with VictoriaMetrics Plugin via YAML

1. Check available versions

Ensure the plugin has been published by checking for ModulePlugin and ModuleConfig resources, in the `global` cluster:

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

This indicates that the ModulePlugin `victoriametrics` exists in the cluster and version `v4.1.0` is published.

2. Create a ModuleInfo

Create a ModuleInfo resource to install the plugin without any configuration parameters:

```yaml
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
```

Reference for resources settings, example prometheus:

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

For more details, please refer to the Monitor Component Capacity Planning

YAML field reference (VictoriaMetrics):

Field path	Description
`metadata.labels.cpaas.io/cluster-name`	Target cluster name where the plugin is installed.
`metadata.labels.cpaas.io/module-name`	Must be `victoriametrics`.

Field path	Description
`metadata.labels.cpaas.io/module-type`	Must be `plugin`.
`metadata.name`	ModuleInfo name (e.g., `<cluster>-victoriametrics`).
`spec.version`	Plugin version to install.
`spec.config.storage.type`	Storage type: `LocalVolume`, `StorageClass`, or `PV`.
`spec.config.storage.capacity`	Storage size for VictoriaMetrics (Gi). Minimum 30 Gi recommended.
`spec.config.storage.nodes`	Node list when `storage.type=LocalVolume`. Up to 1 node supported.
`spec.config.storage.path`	LocalVolume path when `storage.type=LocalVolume`.
`spec.config.storage.storageClass`	StorageClass name when `storage.type=StorageClass`.
`spec.config.storage.pvSelectorK`	PV selector key when `storage.type=PV`.
`spec.config.storage.pvSelectorV`	PV selector value when `storage.type=PV`.
`spec.replicas`	Replica count; LV does not support multiple replicas.
`spec.config.components.vmstorage.retention`	Data retention days for vmstorage.
`spec.config.components.vmagent.scrapeInterval`	Scrape interval seconds; applies to ServiceMonitors without `interval`.

Field path	Description
`spec.config.components.vmagent.scrapeTimeout`	Scrape timeout seconds; must be less than `scrapeInterval`.
`spec.config.components.vmstorage.resources`	Resource settings for vmstorage.
`spec.config.components.nodeExporter.port`	Node Exporter port (default 9100).
`spec.config.components.nodeExporter.resources`	Resource settings for Node Exporter.
`spec.config.components.alertmanager.resources`	Resource settings for Alertmanager.
`spec.config.components.kubeStateExporter.resources`	Resource settings for Kube State Exporter.
`spec.config.components.prometheusAdapter.resources`	Resource settings for Prometheus Adapter (used for HPA/custom metrics).
`spec.config.components.vmagent.resources`	Resource settings for vmagent.
`spec.config.size`	Monitoring scale: `Small`, `Medium`, or `Large`.
`spec.valuesOverride.ait/chart-victoriametrics.global.nodeSelector`	Optional. The nodeselector for the Monitoring component. Use this parameter to select nodes for the monitoring component. This parameter is mainly used for infrastructure nodes.
`spec.valuesOverride.ait/chart-victoriametrics.global.tolerations`	Optional. The tolerations for the Monitoring component. If the node selector selects tainted nodes, use this parameter to specify a taint toleration key, value, and effect. This

Field path	Description
	parameter is mainly used for infrastructure nodes.

3. Verify installation

Since the ModuleInfo name changes upon creation, locate the resource via label to check the plugin status and version:

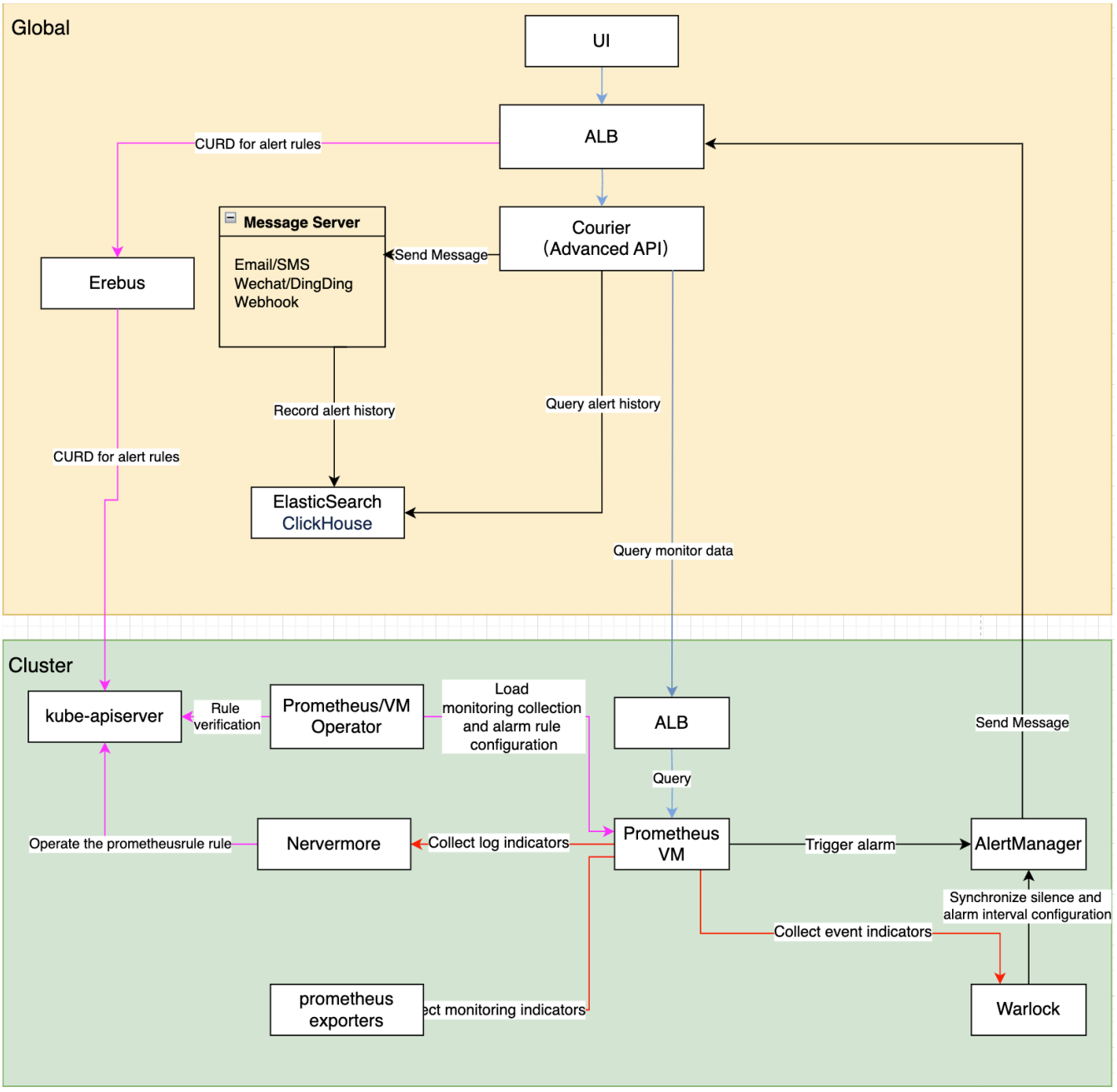
```
kubectl get moduleinfo -l cpaas.io/module-name=victoriametrics
NAME                                             CLUSTER         MODULE
DISPLAY_NAME     STATUS    TARGET_VERSION   CURRENT_VERSION   NEW_VERSION
global-e671599464a5b1717732c5ba36079795          global          victoria
metrics   victoriametrics  Running   v4.1.0           v4.1.0            v
4.1.0
```

Field explanations:

- `NAME`: ModuleInfo resource name
- `CLUSTER`: Cluster where the plugin is installed
- `MODULE`: Plugin name
- `DISPLAY_NAME`: Display name of the plugin
- `STATUS`: Installation status; `Running` means successfully installed and running
- `TARGET_VERSION`: Intended installation version
- `CURRENT_VERSION`: Version before installation
- `NEW_VERSION`: Latest available version for installation

Architecture

Monitoring Module Architecture



TOC

Overall Architecture Explanation

The monitoring system consists of the following core functional modules:

1. Monitoring System
   - Data Collection and Storage: Collecting and persisting monitoring metrics from multiple sources
   - Data Query and Visualization: Providing flexible query and visualization capabilities for monitoring data
2. Alerting System
   - Alert Rule Management: Configuring and managing alert policies
   - Alert Triggering and Notification: Evaluating alert rules and dispatching notifications
   - Real-time Alert Status: Providing a real-time view of the current alert status of the system
3. Notification System
   - Notification Configuration: Managing notification templates, contact groups, and policies
   - Notification Server: Managing the configuration of various notification channels

Monitoring System

Data Collection and Storage

1. Prometheus/VictoriaMetrics Operator Responsibilities:
   - Load and validate monitoring collection configurations
   - Load and validate alert rule configurations
   - Synchronize configurations to Prometheus/VictoriaMetrics instances
2. Sources of Monitoring Data:
   - Nevermore: Generates log-related metrics
   - Warlock: Generates event-related metrics
   - Prometheus/VictoriaMetrics: Discovers and collects various exporters' metrics via ServiceMonitor

Data Query and Visualization

1. Monitoring Data Query Process:
   - The browser initiates a query request (Path: `/platform/monitoring.alauda.io/v1beta1`)
   - ALB forwards the request to the Courier component
   - Courier API processes the query:
     - Built-in Metrics: Obtains PromQL through the indicators interface and queries
     - Custom Metrics: Directly forwards PromQL to the monitoring component
   - The monitoring dashboard retrieves data and displays it
2. Monitoring Dashboard Management Process:
   - Users access the `global` cluster ALB (Path: `/kubernetes/cluster_name/apis/ait.alauda.io/v1alpha2/MonitorDashboard`)
   - ALB forwards the request to the Erebus component

- Erebus routes the request to the target monitoring cluster
- The Warlock component is responsible for:
  - Validating the legality of the monitoring dashboard configuration
  - Managing the MonitorDashboard CR resource

Alerting System

Alert Rule Management

The alert rule configuration process:

1. Users access the `global` cluster ALB (Path: `/kubernetes/cluster_name/apis/monitoring.coreos.com/v1/prometheusrules`)
2. The request passes through ALB -> Erebus -> target cluster kube-apiserver
3. Responsibilities of each component:
   - Prometheus/VictoriaMetrics Operator:
     - Validating the legality of alert rules
     - Managing PrometheusRule CR
   - Nevermore: Listening for and processing log alert metrics
   - Warlock: Listening for and processing event alert metrics

Alert Processing Workflow

1. Alert Evaluation:
   - PrometheusRule/VMRule defines alert rules
   - Prometheus/VictoriaMetrics evaluates rules periodically
2. Alert Notification:
   - Alerts are sent to Alertmanager once triggered
   - Alertmanager -> ALB -> Courier API

- Courier API is responsible for dispatching notifications

3. Alert Storage:

- Alert history is stored in ElasticSearch/ClickHouse

Real-time Alert Status

1. Status Collection:

- The `global` cluster Courier generates metrics:
  - cpaas_active_alerts: Current active alerts
  - cpaas_active_silences: Current silence configurations
- Global Prometheus collects every 15 seconds

2. Status Display:

- The front-end queries and displays real-time status via Courier API

Notification System

Notification Configuration Management

The management process for notification templates, notification contact groups, and notification policies is as follows:

1. Users access the standard API of the `global` cluster via a browser

- Access path: `/apis/ait.alauda.io/v1beta1/namespaces/cpaas-system`

2. Managing related resources:

- 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 is responsible for:

- Validating the legality of notification templates
- Validating the legality of notification contact groups
- Validating the legality of notification policies

Notification Server Management

1. Users access the `global` cluster's ALB via a browser
   - Access path: `/kubernetes/global/api/v1/namespaces/cpaas-system/secrets`
2. Managing and submitting notification server configurations
   - Resource name: platform-email-server
3. Courier is responsible for:
   - Validating the legality of the notification server configuration

Monitoring Component Selection Guide

When installing cluster monitoring, the platform provides two monitoring components for you to choose from: VictoriaMetrics and Prometheus. This article will detail the characteristics and applicable scenarios of these two components, helping you make the most suitable choice.

TOC

Important Notes

- Only one of VictoriaMetrics or Prometheus can be selected when installing cluster monitoring components.
- Starting from version 3.18, VictoriaMetrics has been upgraded to Beta status, which meets production environment usage conditions.
- VictoriaMetrics is suitable for scenarios with high availability requirements and multi-cluster monitoring.
- Prometheus is suitable for single-cluster monitoring scenarios, especially for smaller scales.

Component List

Prometheus Related Components

Component Name	Function Description
Prometheus Server	Core server responsible for collecting, storing, and querying monitoring data
Exporters	Monitoring data collection components that expose monitoring metrics via HTTP interfaces
AlertManager	Alert management center, handling alert rules and notifications
PushGateway	Supports push mode for monitoring data, used for data transfer in special network environments

VictoriaMetrics Related Components

Component Name	Function Description
VMStorage	Monitoring data storage engine

Component Name	Function Description
VMInsert	Data writing component responsible for data distribution and storage
VMSelect	Query service component providing data querying capabilities
VMAlert	Alert rule evaluation and handling component
VMAgent	Monitoring metric collection component

Architecture Comparison

Prometheus Architecture

Prometheus is a mature open-source monitoring system and is the second graduated project of CNCF after Kubernetes. It has the following characteristics:

- Powerful data collection capabilities.
- Flexible query language PromQL.
- A comprehensive ecosystem.
- Supports cluster monitoring at a thousand-node scale.

VictoriaMetrics Architecture

VictoriaMetrics is a next-generation high-performance time series database and monitoring solution with the following advantages:

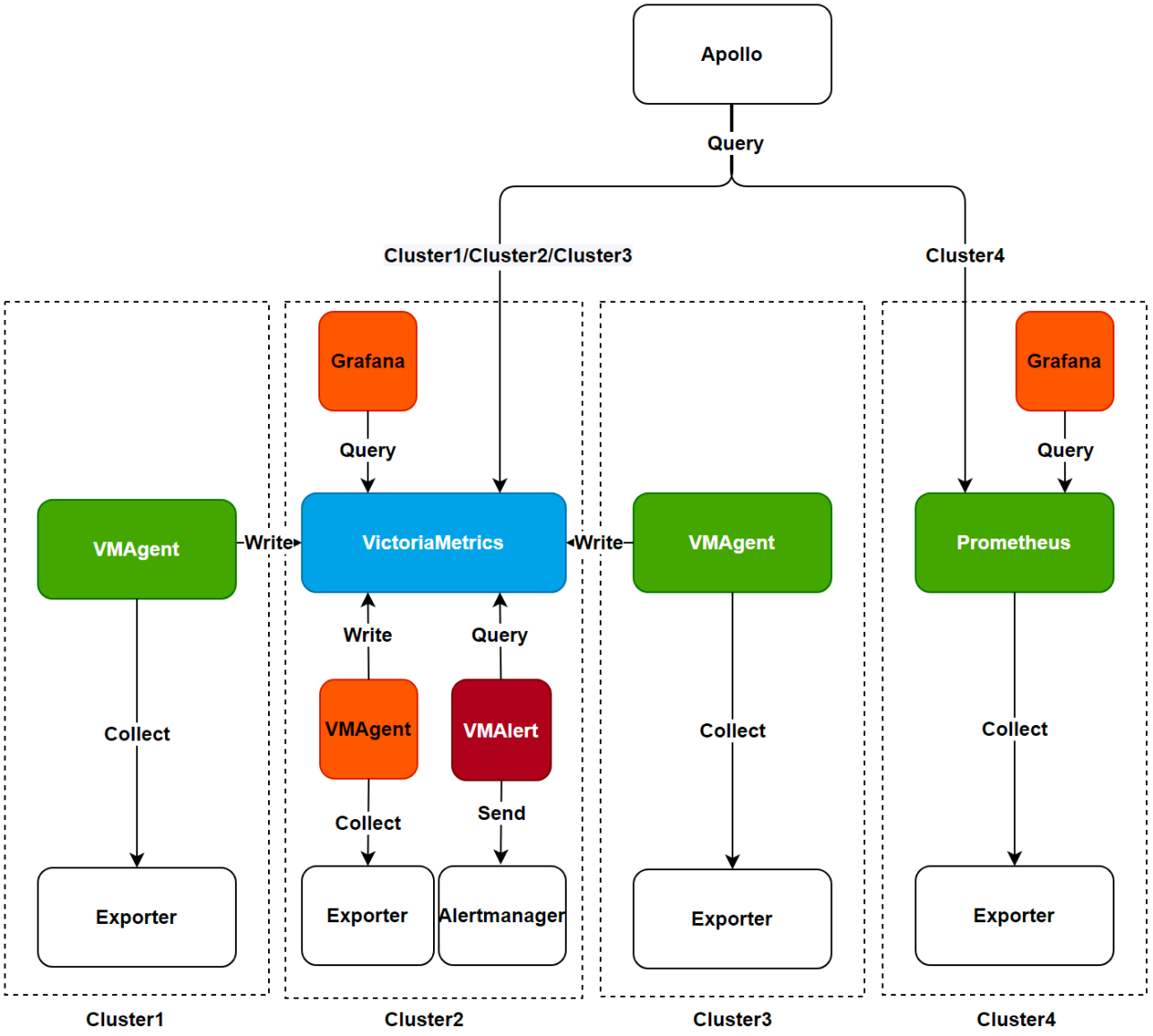
- Higher data compression ratio.
- Lower resource consumption.
- Native support for cluster high availability.
- Simpler operation and maintenance management.

Feature Comparison

Feature	Prometheus	VictoriaMetrics	Description
High Availability Installation	❌	✅	VictoriaMetrics supports true cluster high availability with better data consistency
Single Node Installation	✅	✅	Both support single-node installation mode
Long-term Data Storage	Requires remote storage	Natively supported	VictoriaMetrics is more suitable for long-term data storage
Resource Efficiency	Higher	Better	VictoriaMetrics has better resource utilization
Community Support	Very mature	Rapidly developing	Prometheus has a larger community ecosystem

Installation Scheme Suggestions

Monitoring Installation Architecture Overview



The above diagram shows the installation architecture and data flow of the monitoring components supported by the platform. The platform provides the following two installation methods for selection:

Note: When replacing monitoring components, please ensure that existing components are completely uninstalled, and monitoring data does not support cross-component migration.

Prometheus Installation Method

This method corresponds to the architecture of **cluster4** in the above diagram:

- Uses Prometheus components to collect and process monitoring data.
- Queries and displays data through the monitoring panel.
- Suitable for single-cluster scenarios.

VictoriaMetrics Installation Method

VictoriaMetrics supports the following two installation modes:

1. **Single Cluster Installation Mode**
   - Corresponds to the architecture of **cluster2** in the above diagram.
   - All VictoriaMetrics components are installed in the same cluster.
   - Uses VMAgent to collect data and write to VictoriaMetrics.
   - VMAlert is responsible for alert rule evaluation.
   - Queries and displays data through the monitoring panel. **Tip**: It is recommended to use this mode when data scale is below 1 million per second.
2. **Multi-Cluster Installation Mode**
   - Corresponds to the architecture of **cluster1/cluster2/cluster3** in the above diagram.
   - Installs VMAgent in the workload cluster as a data collection agent.
   - VMAgent writes data into VictoriaMetrics in the central monitoring cluster.
   - Supports unified monitoring management across multiple clusters. **Tip**: Ensure that VictoriaMetrics services are installed in the monitoring cluster before installing VMAgent.

Selection Recommendations

Scenarios Suitable for Using VictoriaMetrics

- **High Performance and Scalability Needs**: Suitable for monitoring scenarios that handle high-throughput data and long-term storage.
- **Cost-Effectiveness Considerations**: Need to optimize storage and computing resource costs.
- **High Availability Requirements**: Requires high availability assurance for monitoring components.
- **Multi-Cluster Management**: Requires unified management of monitoring data across multiple clusters.

Scenarios Suitable for Using Prometheus

- **Single Cluster with Small Scale**: Monitoring scale is small, with no high availability requirements.
- **Existing Prometheus Users**: Already have a complete Prometheus monitoring system.
- **Simple Stability Requirements**: Pursuing a simple and reliable monitoring solution.
- **Deep Ecosystem Integration**: Closely integrated with the Prometheus ecosystem, with high migration costs.

Monitor Component Capacity Planning

The monitor component is responsible for storing metrics data collected from one or more clusters in the platform. Therefore, you need to assess your monitor scale in advance and plan the resources needed for the monitor component according to the guidelines in this document.

TOC

Assumptions and Methodology

- Data in this document comes from controlled lab performance reports and is intended as a sizing baseline for production planning.
- Retention for disk examples is 7 days; adjust proportionally for other retention targets.

- Storage baseline matches the warning above (SSD, ~6000 IOPS, ~250MB/s read/write, independent mount).
- Test workloads exercised typical monitoring pages such as "acp ns overview page" and "platform region detail page".

Prometheus

Below are sizing recommendations by scale for Prometheus and related components (Thanos Query, Thanos Sidecar, etc.).

Small Scale — 10 worker nodes, 500 double-container Pods

- Metric ingestion rate: ~2800 samples/second

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
courier-api	courier	2	2C	4Gi	-	-
kube-prometheus-thanos-query	thanos-query	1	1C	1Gi	-	-
prometheus-kube-prometheus-0	prometheus	1	2C	8Gi	20G	~10G write over 7 days

Medium Scale — 50 worker nodes, 2000 double-container Pods

- Metric ingestion rate: ~7294 samples/second

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
courier-api	courier	2	4C	4Gi	-	-
kube-prometheus-thanos-query	thanos-query	1	2.5C	8Gi	-	-
prometheus-kube-prometheus-0	prometheus	1	4C	8Gi	40G	~30G write over 7 days

Large Scale — 500 worker nodes, 10000 double-container Pods

- Metric ingestion rate: ~41575 samples/second

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
courier-api	courier	2	6C	4Gi	-	-
kube-prometheus-thanos-query	thanos-query	1	2C	6Gi	-	In-fiel deplo may u replic
prometheus-kube-prometheus-0	prometheus	1	8C	20Gi	100G	Peak ~15Gi ~69G over 7

VictoriaMetrics

Below are sizing recommendations by scale for VictoriaMetrics components.

Small Scale — 10 worker nodes, 500 double-container Pods

- Metric ingestion rate: ~3274 samples/second

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	~1.5G write over 7 days

Medium Scale — 50 worker nodes, 2000 double-container Pods

- Metric ingestion rate: ~6940 samples/second

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	-	-

Component	Container	Replicas	CPU Limit	Memory Limit	Disk (if applicable)	Notes
vmstorage-cluster	vmstorage	1	2C	2Gi	10G	~2.6G write over 7 days

Large Scale — 500 worker nodes, 10000 double-container Pods

- Metric ingestion rate: ~34300 samples/second

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	~16.8G write over 7 days

Concepts

TOC

Monitoring

Metrics

Metrics are used to quantitatively describe the operating status of a system, and each metric consists of four basic elements:

- Metric Name: Used to identify the monitored object, such as `cpu_usage`
- Metric Value: Specific measurement value, such as `85.5`
- Timestamp: Records the time of measurement
- Labels: Used for multidimensional data classification, such as `{pod="nginx-1", namespace="default"}`

PromQL

PromQL is the query language for Prometheus, used to query and aggregate metric data from the monitoring system.

Built-in Indicators

The platform has preset a series of commonly used monitoring metrics based on long-term operational experience. You can directly use these metrics when configuring alarm rules or creating monitoring dashboards without additional configuration.

Exporter

The Exporter is a component for collecting monitoring data, with primary responsibilities including:

- Collecting raw monitoring data from the target system
- Transforming data into a standard time-series metric format
- Providing metric data for querying via HTTP interface

ServiceMonitor

ServiceMonitor is used to declaratively manage monitoring configurations and primarily defines:

- The selection criteria for monitoring targets
- Configuration of metric collection interfaces
- Execution parameters for collection tasks (intervals, timeouts, etc.)

Alarms

Alarm Rules

Alarm rules define the specific conditions for triggering alarms:

- Alarm Expression: Describes the conditions for triggering an alarm using PromQL statements
- Alarm Threshold: Explicit boundary values for trigger
- Duration: Duration for which the conditions must be continuously met
- Alarm Level: Distinguishes the severity of alarms (e.g., P0/P1/P2)

Alarm Policies

Alarm policies organize multiple alarm rules together for unified configuration:

- Alarm Targets: The target scope of the rules
- Notification Method: The channels for sending alarms
- Sending Interval: The time interval for repeated alarm notifications

Notifications

Notification Policies

Notification policies manage the rules for sending alarm messages:

- Recipients: Target users for alarm notifications
- Notification Channels: Supported message sending methods
- Notification Templates: Definition of message content format

Notification Templates

Notification templates customize the display format of alarm messages:

- Title Template: Format of the alarm message title
- Content Template: Organization of alarm details
- Variable Replacement: Supports dynamic data filling

Monitoring Dashboard

Dashboard

A dashboard is a collection of multiple related panels, providing an overall view of the system status. It supports flexible layout arrangements and can organize panels in rows or columns.

Panels

Panels are visual representations of monitoring data, supporting various display types.

Data Sources

The configuration of monitoring data sources. Currently, only the monitoring components of the current cluster are supported as data sources, and custom data sources are not supported for now.

Variables

Variables serve as placeholders for values and can be used in metric queries. Through the variable selector at the top of the dashboard, you can dynamically adjust query conditions,

allowing chart content to update in real-time.

Guides

Management of Metrics

- Viewing Metrics Exposed by Platform Com
- Viewing All Metrics Stored by Prometheus
- Viewing All Built-in Metrics Defined by the
- Integrating External Metrics

Management of Alert

- Function Overview
- Key Features
- Functional Advantages
- Creating Alert Policies via UI
- Creating Resource Alerts via CLI
- Creating Event Alerts via CLI
- Creating Alert Policies via alert Templates
- Setting Silence for Alerts

Managemer

- Feature Overvie
- Key Features
- Notification Ser
- Notification Cor
- Notification Tem
- Notification rule
- Set Notification

Management of Monitoring Dashboards

- Function Overview
- Manage Dashboards
- Manage Panels
- Create Monitoring Dashboards via CLI
- Common Functions and Variables

Managemer

- Function Overv
- Blackbox Monit
- Blackbox Alerts
- Customizing Bla
- Create Blackbo
- Reference Infor

Management of Metrics

The platform's monitoring system is based on the metrics collected by Prometheus / VictoriaMetrics. This document will guide you on how to manage these metrics.

TOC

Viewing Metrics Exposed by Platform Components

The monitoring method for the cluster components within the platform is to extract metrics exposed via `ServiceMonitor`. Metrics in the platform are publicly available through the

`/metrics` endpoint. You can view the exposed metrics of a specific component in the platform using the following example command:

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

Sample Output:

```
# 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 con
current reconciles per controller
# TYPE controller_runtime_max_concurrent_reconciles gauge
# HELP controller_runtime_reconcile_errors_total Total number of reconcil
iation errors per controller
# TYPE controller_runtime_reconcile_errors_total counter
# HELP controller_runtime_reconcile_time_seconds Length of time per recon
ciliation per controller
```

Viewing All Metrics Stored by Prometheus / VictoriaMetrics

You can view the list of available metrics in the cluster to help you write the PromQL you need based on these metrics.

Prerequisites

1. You have obtained your user Token
2. You have obtained the platform address

Procedures

> Run the following command to get the list of metrics using the `curl` command:

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

Sample Output:


```
{
  "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"]
}
```


Viewing All Built-in Metrics Defined by the Platform

To simplify user usage, the platform has built in a large number of commonly used metrics. You can directly use these metrics when configuring alerts or monitoring dashboards without needing to define them yourself. The following will introduce you to how to view these metrics.

Prerequisites

1. You have obtained your user Token
2. You have obtained the platform address

Procedures

Run the following command to get the list of metrics using the `curl` command:

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

Sample Output:

```
[
  {
  "alertEnabled": true, (1)
  "annotations": {
   "cn": "CPU utilization of containers in the compute component",
   "descriptionEN": "Cpu utilization for pods in workload",
   "descriptionZH": "CPU utilization of containers in the compute com
ponent",
   "displayNameEN": "CPU utilization of the pods",
   "displayNameZH": "CPU utilization of containers in the compute com
ponent",
   "en": "Cpu utilization for pods in workload",
   "features": "SupportDashboard", (2)
   "summaryEN": "CPU usage rate {{.externalLabels.comparison}}{{.exte
rnalLabels.threshold}} of Pod ({{.labels.pod}})",
   "summaryZH": "CPU usage rate {{.externalLabels.comparison}}{{.exte
rnalLabels.threshold}} of pod ({{.labels.pod}})"
  },
  "displayName": "CPU utilization of containers in the compute compon
ent",
  "kind": "workload",
  "multipleEnabled": true,  (3)
  "name": "workload.pod.cpu.utilization",
  "query": "avg by (kind,name,namespace,pod) (avg by (kind,name,names
pace,pod,container)(cpaas_advanced_container_cpu_usage_seconds_total_
irate5m{kind=~\"{{.kind}}\",name=~\"{{.name}}\",namespace=~\"{{.names
pace}}\",container!=\"\",container!=\"POD\"}) / avg by (kind,name,nam
espace,pod,container)(cpaas_advanced_kube_pod_container_resource_limi
ts{kind=~\"{{.kind}}\",name=~\"{{.name}}\",namespace=~\"{{.namespac
e}}\",resource=\"cpu\"}))", (4)
  "summary": "CPU usage rate {{.externalLabels.comparison}}{{.externa
lLabels.threshold}} of pod ({{.labels.pod}})",
  "type": "metric",
  "unit": "%",
  "legend": "{{.namespace}}/{{.pod}}",
  "variables": [ (5)
   "namespace",
   "name",
   "kind"
  ]
 }
]
```

1. Whether this metric supports being used for configuring alerts
2. Whether this metric supports being used in monitoring dashboards
3. Whether this metric supports being used when configuring alerts for multiple resources
4. The PromQL statement defined for the metric
5. The variables that can be used in the PromQL statement of the metric

Integrating External Metrics

In addition to the built-in metrics of the platform, you can also integrate metrics exposed by your applications or third-party applications via `ServiceMonitor` or `PodMonitor`. This section uses the Elasticsearch Exporter installed in pod form in the same cluster as an example for explanation.

Prerequisites

You have installed your application and exposed metrics through specified interfaces. In this document, we assume your application is installed in the `cpaas-system` namespace and has exposed the `http://<elasticsearch-exporter-ip>:9200/_prometheus/metrics` endpoint.

Procedures

1. Create a Service/Endpoint for the Exporter to expose metrics

```
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. Create a `ServiceMonitor` object to describe the metrics exposed by your application:

```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) To which Prometheus should the ServiceMonitor be synchronized; the operator will listen to the corresponding ServiceMonitor resource based on the serviceMonitorSelector configuration of the Prometheus CR. If the ServiceMonitor's labels do not match the serviceMonitorSelector configuration of the Prometheus CR, this ServiceMonitor will not be monitored by the operator.

(2) The operator will listen to which namespaces of ServiceMonitor based on the serviceMonitorNamespaceSelector configuration of the Prometheus CR; if the

ServiceMonitor is not in the serviceMonitorNamespaceSelector of the Prometheus CR, this ServiceMonitor will not be monitored by the operator.

(3) Metrics collected by Prometheus will add a job label, with the value being the service label value corresponding to jobLabel.

(4) The ServiceMonitor matches the corresponding Service based on the namespaceSelector configuration.

(5) The ServiceMonitor matches the Service based on the selector configuration.

(6) The ServiceMonitor matches the Service's port based on port configuration.

(7) The access path to the Exporter, default is /metrics.

(8) The interval at which Prometheus scrapes the Exporter metrics.

(9) If authentication is required to access the Exporter path, authentication information needs to be added; it also supports bearer token, tls authentication, and other methods.

3. Check if the ServiceMonitor is being monitored by Prometheus
   Access the UI of the monitoring component to check if the job `cpaas-elasticsearch-exporter` exists.

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

Management of Alert

TOC

Function Overview

The alert management function of the platform aims to help users comprehensively monitor and promptly detect system anomalies. By utilizing pre-installed system alerts and flexible custom alert capabilities, combined with standardized alert templates and a tiered management mechanism, it provides a complete alert solution for operation and maintenance personnel.

Whether it's platform administrators or business personnel, they can conveniently configure and manage alert policies within their respective permission scopes for effective monitoring of platform resources.

Key Features

- **Built-in System Alert Policies**: Rich alert rules are preset based on common fault diagnosis ideas for `global` clusters and workload clusters.
- **Custom Alert Rules**: Supports the creation of alert rules based on various data sources, including preset monitoring indicators, custom monitoring indicators, black-box monitoring items, platform log data, and platform event data.
- **Alert Template Management**: Supports the creation and management of standardized alert templates for quick application to similar resources.
- **Alert Notification Integration**: Supports the push of alert information to operation and maintenance personnel through various channels.
- **Alert View Isolation**: Distinguishes between platform management alerts and business alerts, ensuring that personnel in different roles focus on their respective alert information.
- **Real-time Alert Viewing**: Provides real-time alerts, offering concentrated displays of the number of resources currently experiencing alerts and detailed alert information.

- **Alert History Viewing**: Supports the viewing of historical alert records over a period, facilitating the analysis of recent monitoring alert conditions by operation and maintenance personnel and administrators.

Functional Advantages

- **Comprehensive Monitoring Coverage**: Supports monitoring of various resource types such as clusters, nodes, and computing components, and comes with rich built-in system alert policies that can be used without additional configuration.
- **Efficient Alert Management**: Standardized configurations through alert templates enhance operational efficiency, and the separation of alert views makes it easier for personnel in different roles to quickly locate relevant alerts.
- **Timely Problem Detection**: alert notifications are automatically triggered to ensure timely problem detection, supporting multi-channel alert pushing for proactive problem avoidance.
- **Robust Permission Management**: Strict access control for alert policies ensures that alert information is secure and manageable.

Creating Alert Policies via UI

Prerequisites

- A notification policy is configured (if you need to configure automatic alert notifications).
- Monitoring components are installed in the target cluster (required when creating alert policies using monitoring indicators).
- Log storage components and log collection components are installed in the target cluster (required when creating alert policies using logs and events).

Procedures

1. Navigate to **Operation and Maintenance Center** > **alerts** > **alert Policies**.
2. Click **Create Alert Policy**.
3. Configure basic information.

Selecting Alert Type

Resource Alert

- Alert types categorized by resource type (e.g., deployment status under a namespace).
- Resource selection description:
  - Defaults to "Any" if no parameter is selected, supporting automatic association with newly added resources.
  - When "Select All" is chosen, it only applies to the current resource.
  - When multiple namespaces are selected, resource names support regular expressions (e.g., `cert.*`).

Event Alert

- Alert types categorized by specific events (e.g., abnormal Pod status).
- By default, selects all resources under the specified resource and supports automatic association with newly added resources.

Configuring Alert Rules

Click **Add Alert Rule** and configure the following parameters based on the alert type:

Resource Alert Parameters

Parameter	Description
Expression	Monitoring metric algorithm in Prometheus format, e.g., `rate(node_network_receive_bytes{instance="$server",device!~"lo"}[5m])`
Metric Unit	Custom monitoring metric unit, can be entered manually or selected from platform preset units
Legend Parameter	Controls the name corresponding to the curve in the chart, formatted as `{{.LabelName}}`, e.g., `{{.hostname}}`
Time Range	Time window for log/event queries

Parameter	Description
Log Content	Query fields for log content (e.g., Error), where multiple query fields are linked by OR
Event Reason	Query fields for event reasons (Reason, e.g., BackOff, Pulling, Failed, etc.), where multiple query fields are linked by OR
Trigger Condition	Condition consisting of comparison operators, alert thresholds, and duration (optional). Determines if an alert is triggered based on the comparison of real-time values/log count/event count against the alert threshold, as well as the duration of real-time values within the alert threshold range.
alert Level	Divided into four levels: Critical, Serious, Warning, and Info. You can set a reasonable alert level according to the impact of the alert rules on business for the corresponding resources.

Event Alert Parameters

Parameter	Description
Time Range	Time window for event queries
Event Monitoring Item	Supports monitoring event levels or event reasons, where multiple fields are linked by OR
Trigger Condition	Based on event count for comparison judgement
alert Level	Same definition as resource alert levels

Other Configurations

1. Select one or more created notification policies.
2. Configure alert sending intervals.
   - Global: Use platform default configuration.
   - Custom: Different sending intervals can be set based on alert levels.
   - When "Do Not Repeat" is selected, notifications will only be sent when the alert is triggered and recovered.

Additional Notes

1. In the "More" options of the alert rule, labels and annotations can be set.
2. Please refer to the Prometheus Alerting Rules Documentation ↗ for configuring labels and annotations.
3. Note: Do not use the `$value` variable in labels, as this may cause alert exceptions.

Creating Resource Alerts via CLI

Prerequisites

- A notification policy is configured (if you need to configure automatic alert notifications).
- Monitoring components are installed in the target cluster (required when creating alert policies using monitoring indicators).
- Log storage components and log collection components are installed in the target cluster (required when creating alert policies using logs and events).

Procedures

1. Create a new YAML configuration file named `example-alerting-rule.yaml`.
2. Add PrometheusRule resources to the YAML file and submit it. The following example creates a new alert policy called policy:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global # The name of the cluster where the alert is located
    alert.cpaas.io/kind: Cluster # The type of resource,
    alert.cpaas.io/name: global # The resource object, supporting single, multiple (separated by |), or any (.*)
    alert.cpaas.io/namespace: cpaas-system # The namespace where the alert object is located, supporting single, multiple (separated by |), or any (.*)
    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 # The display name of the alert policy
  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 # alert rule name
      rules:
        - alert: cluster.pod.status.phase.not.running-tx1ob-e998f0b94854ee1eade5ae79279e005a
          annotations:
            alert_current_value: '{{ $value }}' # Notification of the current value, keep as default
          expr: (count(min by(pod)(kube_pod_container_status_ready{}) !=1) or on() vector(0))>2
          for: 30s # Duration
```

```
          labels:
            alert_cluster: global # The name of the cluster where the a
lert is located
            alert_for: 30s # Duration
            alert_indicator: cluster.pod.status.phase.not.running # The
name of the alert rule indicator (custom alert indicator name as custo
m)
            alert_indicator_aggregate_range: '30' # The aggregation tim
e for the alert rule, in seconds
            alert_indicator_blackbox_name: '' # Black-box monitoring it
em name
            alert_indicator_comparison: '>' # The comparison method for
the alert rule
            alert_indicator_query: '' # Query for the logs of the alert
rule (only for log alerts)
            alert_indicator_threshold: '2' # The threshold for the aler
t rule
            alert_indicator_unit: '' # The indicator unit for the alert
rule
            alert_involved_object_kind: Cluster # The type of the objec
t to which the alert rule belongs: Cluster|Node|Deployment|Daemonset|St
atefulset|Middleware|Microservice|Storage|VirtualMachine
            alert_involved_object_name: global # The name of the object
to which the alert rule belongs
            alert_involved_object_namespace: '' # The namespace of the
object to which the alert rule belongs
            alert_name: cluster.pod.status.phase.not.running-tx1ob # Th
e name of the alert rule
            alert_namespace: cpaas-system # The namespace where the ale
rt rule is located
            alert_project: cpaas-system # The project name of the objec
t to which the alert rule belongs
            alert_resource: policy # The name of the alert policy where
the alert rule is located
            alert_source: Platform # The data type of the alert policy
where the alert rule is located: Platform-Platform Data Business-Busine
ss Data
            severity: High # The severity level of the alert rule: Crit
ical-Critical, High-Serious, Medium-Warning, Low-Info
```

Creating Event Alerts via CLI

Prerequisites

- A notification policy is configured (if you need to configure automatic alert notifications).
- Monitoring components are installed in the target cluster (required when creating alert policies using monitoring indicators).
- Log storage components and log collection components are installed in the target cluster (required when creating alert policies using logs and events).

Procedures

1. Create a new YAML configuration file named `example-alerting-rule.yaml`.
2. Add PrometheusRule resources to the YAML file and submit it. The following example creates a new alert policy called policy2:

```
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":"Deploymen
t","operator":"=","namespaces":["*"]}]'
      # names: The resource name for the event alert; operator is ineff
ective if name is empty.
      # kind: The type of resource that triggers the event alert.
      # namespace: The namespace where the resource that triggers the e
vent alert belongs. An empty array indicates a non-namespaced resource;
when ns is ['*'], it indicates all namespaces.
      # operator: Selector =, !=, =~, !~
    alert.cpaas.io/kind: Event # The type of alert, Event (event alert)
    alert.cpaas.io/name: '' # Used for resource alerts; remains empty f
or event alerts
    alert.cpaas.io/namespace: cpaas-system
    alert.cpaas.io/notifications: '["acp-qwtest"]'
    alert.cpaas.io/repeat-config: '{"Critical":"never","High":"5m","Med
ium":"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-34c9a378e3b6dda8401c2d728994
ce2f
          # 6sial-34c9a378e3b6dda8401c2d728994ce2f can be customized to
```

```
ensure uniqueness
          annotations:
            alert_current_value: '{{ $value }}' # Notification of the c
urrent value, keep as default
          expr: round(((avg
            by(kind,namespace,name,reason)(increase(cpaas_event_count{n
amespace=~".*",id="policy2-cluster.event.count-6sial"}[300s])))
            + (avg
            by(kind,namespace,name,reason)(abs(increase(cpaas_event_cou
nt{namespace=~".*",id="policy2-cluster.event.count-6sial"}[300s]))))))
            / 2)>2
          # The id in the policy2 needs to be the name of the alert pol
icy; 6sial must match the preceding alert rule name
          for: 15s # Duration
          labels:
            alert_cluster: global # The name of the cluster where the a
lert is located
            alert_for: 15s # Duration
            alert_indicator: cluster.event.count # The name of the aler
t rule indicator (custom alert indicator name as custom)
            alert_indicator_aggregate_range: '300' # The aggregation ti
me for the alert rule, in seconds
            alert_indicator_blackbox_name: ''
            alert_indicator_comparison: '>' # The comparison method for
the alert rule
            alert_indicator_event_reason: ScalingReplicaSet # Event rea
son.
            alert_indicator_threshold: '2' # The threshold for the aler
t rule
            alert_indicator_unit: pieces # The indicator unit for the a
lert rule; remains unchanged for event alerts
            alert_involved_object_kind: Event
            alert_involved_object_options: Single
            alert_name: cluster.event.count-6sial # The name of the ale
rt rule
            alert_namespace: cpaas-system # The namespace where the ale
rt rule is located
            alert_project: cpaas-system # The project name of the objec
t to which the alert rule belongs
            alert_repeat_interval: 5m
            alert_resource: policy2 # The name of the alert policy wher
e the alert rule is located
            alert_source: Platform # The data type of the alert policy
where the alert rule is located: Platform-Platform Data Business-Busine
```

```
ss Data
            severity: High # The severity level of the alert rule: Crit
ical-Critical, High-Serious, Medium-Warning, Low-Info
```

Creating Alert Policies via alert Templates

alert templates are a combination of alert rules and notification policies targeted at similar resources. Through alert templates, it is easy and quick to create alert policies for clusters, nodes, or computing components on the platform.

Prerequisites

- A notification policy is configured (if you need to configure automatic alert notifications).
- Monitoring components are installed in the target cluster (required when creating alert policies using monitoring indicators).

Procedures

Creating Alert Template

1. In the left navigation bar, click **Operation and Maintenance Center** > **alerts** > **alert Templates**.
2. Click **Create alert Template**.
3. Configure the basic information of the alert template.
4. In the **alert Rules** section, click **Add alert Rule**, and follow the parameter descriptions below to add alert rules:

Parameter	Description
Expression	Monitoring metric algorithm in Prometheus format, e.g., `rate(node_network_receive_bytes{instance="$server",device!~"lo"}[5m])`
Metric Unit	Custom monitoring metric unit, can be entered manually or selected from platform preset units

Parameter	Description
Legend Parameter	Controls the name corresponding to the curve in the chart, formatted as `{{.LabelName}}`, e.g., `{{.hostname}}`
Time Range	Time window for log/event queries
Log Content	Query fields for log content (e.g., Error), where multiple query fields are linked by OR
Event Reason	Query fields for event reasons (Reason, e.g., BackOff, Pulling, Failed, etc.), where multiple query fields are linked by OR
Trigger Condition	Condition consisting of comparison operators, alert thresholds, and duration (optional).
alert Level	Divided into four levels: Critical, Serious, Warning, and Info. You can set a reasonable alert level according to the impact of the alert rules on business for the corresponding resources.

5. Click **Create**.

Creating Alert Policies Using alert Templates

1. In the left navigation bar, click **Operation and Maintenance Center** > **alerts** > **alert Policies**. **Tip**: You can switch the target cluster through the top navigation bar.
2. Click the expand button next to the **Create alert Policy** button > **Template Create alert Policy**.
3. Configure some parameters, referring to the descriptions below:

Parameter	Description
Template Name	The name of the alert template to use. The templates are categorized by cluster, node, and computing component. Upon selecting a template, you can view the alert rules, notification policies, and other information set within the alert template.
Resource Type	Select whether the template is an alert policy template for **Cluster**, **Node**, or **Computing Component**; the corresponding resource name will be displayed.

4. Click **Create**.

Setting Silence for Alerts

Supports silencing alerts for clusters, nodes, and computing components. By setting silence for specific alert policies, you can control that all rules under the alert policy do not send notification messages when triggered during the set silence period. Permanent silence and custom time silence can be set.

For example: When the platform is upgraded or maintained, many resources may show abnormal statuses, leading to numerous triggered alerts, which cause operation and maintenance personnel to frequently receive alert notifications before the upgrade or maintenance is completed. Setting silence for the alert policy can prevent this situation.

Note: When the silence status persists until the silence end time, the silence setting will be automatically cleared.

Setting via UI

1. In the left navigation bar, click **Operation and Maintenance Center** > **alerts** > **alert Policies**.
2. Click the operation button on the right side of the alert policy to be silenced > **Set Silence**.
3. Toggle **alert Silence** switch to open it.
   Tip: This switch controls whether the silence setting takes effect. To cancel silence, simply turn off the switch.
4. Configure relevant parameters according to the descriptions below:
   Tip: If no silence range or resource name is selected, it defaults to **Any**, meaning that subsequent **Delete/Add** resource actions will correspond to **Delete Silence/Add Silence** alert policies; if "Select All" is chosen, it will only apply to the currently selected resource range, and subsequent **Delete/Add** resource actions will not be processed.

Parameter	Description
Silence Range	The scope of resources where the silence setting takes effect.

Parameter	Description
Resource Name	The name of the resource object targeted by the silence setting.
Silence Time	The time range for alert silence. The alert will enter silence state at the start of the silence time, and if the alert policy remains in an alert state or triggers alerts after the silence end time, alert notifications will resume. **Permanent**: The silence setting will last until the alert policy is deleted. **Custom**: Custom settings for the start time and end time of silence, with the time interval not less than 5 minutes.

5. Click **Set**.

   Tip: From the moment silence is set until the start of silence, the silence status of the alert policy is considered **Silence Waiting**. During this period, when rules in the policy trigger alerts, notifications will be sent normally; after silence starts until it ends, the silence status of the alert policy is **Silencing**, and when rules in the policy trigger alerts, notifications will not be sent.

Setting via CLI

1. Specify the resource name of the alert policy you want to set silence for and execute the following command:

   ```
   kubectl edit PrometheusRule <TheNameOfThealertPolicyYouWantToSet>
   ```

2. Modify the resource as shown in the example to add silence annotations and submit.

```yaml
---
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"}]}}'
      # The silence configuration for node-level alert policies, including start time, end time, creator, etc.; if the silence range includes specific nodes, please append the resources.node information as shown above. If you need silence for all resources, you do not need the resources field.
    # 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"]}'
    # The silence configuration for workload-level alert policies, including start time, end time, creator, etc.; if the silence range includes specific workloads, please append name and namespace information as shown above. If you need silence for all resources, you do not need the name and namespace fields.
    # Setting the endsAt field to 2199-12-31T00:00:00Z indicates permanent silence.
    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:
  ## Exclude irrelevant information
```

Recommendations for Configuring Alert Rules

More alert rules do not always equate to better outcomes. Redundant or complex alert rules can lead to alert storms and increase your maintenance burden. It is recommended that you read the following guidelines before configuring alert rules to ensure that custom rules can achieve their intended purposes while remaining efficient.

- **Use the Fewest New Rules Possible**: Create only those rules that meet your specific requirements. By using the fewest number of rules, you can create a more manageable and centralized alert system in the monitoring environment.
- **Focus on Symptoms Rather than Causes**: Create rules that notify users of symptoms rather than the root causes of those symptoms. This ensures that when relevant symptoms occur, users can receive alerts and may investigate the root causes that triggered the alerts. Using this strategy can significantly reduce the total number of rules you need to create.
- **Plan and Assess Your Needs Before Making Changes**: First, clarify which symptoms are important and what actions you want users to take when these symptoms occur. Then evaluate existing rules to decide if you can modify them to achieve your objectives without creating new rules for each symptom. By modifying existing rules and carefully creating new ones, you can help simplify the alert system.
- **Provide Clear Alert Messages**: When you create alert messages, include descriptions of symptoms, possible causes, and recommended actions. The information included should be clear, concise, and provide troubleshooting procedures or links to additional relevant information. Doing so helps users quickly assess situations and respond appropriately.
- **Set Severity Levels Reasonably**: Assign severity levels to your rules to indicate how users should respond when symptoms trigger alerts. For instance, classify alerts with a severity level of Critical, signaling that immediate action is required from relevant personnel. By establishing severity levels, you can help users decide how to respond upon receiving alerts and ensure prompt responses to urgent issues.

Management of Notification

TOC

Feature Overview

With notifications, you can integrate the platform's monitoring and alerting features to promptly send pre-warning information to notification recipients, reminding relevant personnel to take necessary measures to resolve issues or avoid failures.

Key Features

- **Notification Server**: The notification server provides services for sending notification messages to notification contact groups on the platform, such as an email server.
- **Notification Contact Group**: A notification contact group is a set of notification recipients with similar logical characteristics, which can reduce your maintenance burden by allowing a categorization of entities that receive notification messages.
- **Notification Template**: A notification template is a standardized structure composed of custom content, content variables, and content format parameters. It is used to standardize the content and format of alert notification messages for notification strategies. For example, customizing the subject and content of email notifications.
- **Notification rule**: A notification rule is a collection of rules defining how to send notification messages to specific contacts. It is essential to use a notification rule for scenarios such as alerts, inspections, and login authentication that require notifying external services.

Notification Server

The notification server provides services for sending notification messages to recipients on the platform. The platform currently supports the following notification servers:

- **Corporate Communication Tool Server**: Supports integration with WeChat Work, DingTalk, and Feishu built-in applications for sending notifications to individuals.
- **Email Server**: Sends notifications via email using an email server.
- **Webhook Type Server**: Supports integration with corporate WeChat group bots, DingTalk group bots, Feishu group bots, or sending WebHooks to your designated server.

WARNING

Only one corporate communication tool server can be added.

Corporate Communication Tool Server

WeChat Work

1. Configure the notification server parameters as per the example below. Once parameters are filled in, switch to the `global` cluster in **Cluster Management** > **Resource Management** and create the resource object.

```
# WeChat Work corpId, corpSecret, agentId acquisition methods can be re
ferenced in the official documentation: https://developer.work.weixin.q
q.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            # Server's Chinese display name, en
coded in base64 by default
  displayNameEn: WeChat             # Server's English display name, en
coded in base64 by default
  corpId:                           # Corporate ID, encoded in base64 b
y default
  corpSecret:                       # Application secret, encoded in ba
se64 by default
  agentId:                          # Corporate application ID, encoded
in base64 by default
```

2. After the creation, you need to update the user's **WeChat Work ID** in the platform's **User Role Management** > **User Management** or in the user's **Personal Information** to ensure the user can receive messages normally.

DingTalk

1. Configure the notification server parameters as per the example below. Once parameters are filled in, switch to the `global` cluster in **Cluster Management** > **Resource Management** and create the resource object.

```yaml
# DingTalk appKey, appSecret, agentId acquisition method: 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                # Server's Chinese display name, encoded in base64 by default
  displayNameEn: DingTalk           # Server's English display name, encoded in base64 by default
  appKey:                           # Application key, encoded in base64 by default
  appSecret:                        # Application secret, encoded in base64 by default
  agentId:                          # Application agent_id, encoded in base64 by default
```

2. After the creation, you need to update the user's **DingTalk ID** in the platform's **User Role Management** > **User Management** or in the user's **Personal Information** to ensure the user can receive messages normally.

Feishu

1. Configure the notification server parameters as per the example below. Once parameters are filled in, switch to the `global` cluster in **Cluster Management** > **Resource Management** and create the resource object.

```
# Feishu appId, appSecret acquisition methods: 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                  # Server's Chinese display name, encoded in base64 by default
  displayNameEn: Feishu               # Server's English display name, encoded in base64 by default
  appId:                              # Application ID, encoded in base64 by default
  appSecret:                          # Application secret, encoded in base64 by default
```

2. After the creation, you need to update the user's **Feishu ID** in the platform's **User Role Management** > **User Management** or in the user's **Personal Information** to ensure the user can receive messages normally.

Email Server

1. In the left navigation bar, click **Platform Settings** > **Notification Server**.
2. Click **Configure Now**.
3. Refer to the following instructions to configure the relevant parameters.

Parameter	Description
Service Address	The address of the notification server supporting the SMTP protocol, e.g., `smtp.yeah.net`.
Port	The port number for the notification server. When **Use SSL** is checked, the SSL port number must be entered.

Parameter	Description
Server Configuration	**Use SSL**: Secure Socket Layer (SSL) is a standard security technology. The SSL switch is used to control whether to establish an encrypted link between the server and client. **Skip Insecure Verification**: The insecureSkipVerify switch is used to control whether to verify the client certificate and server hostname. If enabled, certificates and the consistency between the hostname in the certificate and the server hostname will not be verified.
Sender Email	The sender's email account in the notification server, used for sending notification emails.
Enable Authentication	If authentication is required, please configure the username and authorization code for the email server.

4. Click **OK**.

Webhook Type Server

Supports integration with corporate WeChat group bots, DingTalk group bots, Feishu group bots, or sending HTTP requests to your designated Webhook server.

Corporate WeChat Group Bot

1. In the left navigation bar, click **Cluster Management** > **Cluster**.
2. Click the operation button next to the `global` cluster > **CLI Tool**.
3. Execute the following command on the master node of the `global` cluster:

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

Tip: `dHJ1ZQo=` is the base64 encoded value of true; to disable, replace `dHJ1ZQo=` with `ZmFsc2UK`, which is the base64 encoded value of false.

DingTalk Group Bot

1. In the left navigation bar, click **Cluster Management** > **Cluster**.
2. Click the operation button next to the `global` cluster > **CLI Tool**.
3. Execute the following command on the master node of the `global` cluster:

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

Tip: `dHJ1ZQo=` is the base64 encoded value of true; to disable, replace `dHJ1ZQo=` with `ZmFsc2UK`, which is the base64 encoded value of false.

Feishu Group Bot

1. In the left navigation bar, click **Cluster Management** > **Cluster**.
2. Click the operation button next to the `global` cluster > **CLI Tool**.
3. Execute the following command on the master node of the `global` cluster:

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

Tip: `dHJ1ZQo=` is the base64 encoded value of true; to disable, replace `dHJ1ZQo=` with `ZmFsc2UK`, which is the base64 encoded value of false.

Webhook Server

1. In the left navigation bar, click **Cluster Management** > **Cluster**.
2. Click the operation button next to the `global` cluster > **CLI Tool**.
3. Execute the following command on the master node of the `global` cluster:

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

Tip: `dHJ1ZQo=` is the base64 encoded value of true; to disable, replace `dHJ1ZQo=` with `ZmFsc2UK`, which is the base64 encoded value of false.

Configure Custom Request Headers for Webhook Notifications

If the target Webhook endpoint requires custom HTTP request headers, create a Secret in the `cpaas-system` namespace and associate it with the notification contact group later.

1. In the left navigation bar, click **Cluster Management** > **Cluster**.
2. Click the operation button next to the `global` cluster > **CLI Tool**.
3. Create a YAML file similar to the following on the master node of the `global` cluster.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: webhook-header-secret
  namespace: cpaas-system
type: NotificationSender
data:
  X-Secret: <base64-encoded-header-value>
```

4. Save the YAML file as `webhook-header.yaml` and apply the resource.

```
kubectl apply -f webhook-header.yaml
```

INFO

1. Each key under `data` is used as an HTTP header name.
2. Each value under `data` must be base64 encoded before it is applied to the cluster.
3. If the endpoint requires multiple headers, add more key-value pairs under `data`.

Notification Contact Group

A notification contact group is a set of notification recipients with similar logical characteristics. For example, you can set an operations and maintenance team as a notification contact group

for easy selection and management when configuring notification strategies.

INFO

1. The platform supports various notification servers, and the corresponding configuration options for notification types will be displayed based on the notification server configuration.
2. If you need to use a Webhook type server as a notification recipient, you must configure the relevant URL in the notification contact group.
3. If the Webhook endpoint requires custom request headers, you must associate the notification contact group with a Secret of type `NotificationSender` in the `cpaas-system` namespace.

1. In the left navigation bar, click **Operations Center** > **Notifications**.
2. Switch to the **Notification Contact Group** tab.
3. Click **Create Notification Contact Group** and configure the relevant parameters as per the instructions below.

Parameter	Description
Email	Add an email to the entire notification contact group. The platform will send notifications to this email and all contacts' emails in the group.
Webhook URL/WeChat Group Bot/DingTalk Group Bot/Feishu Group Bot	Please fill in the corresponding notification method URL based on the configured notification server. Once configured, contacts in this group will be notified using this method.
Contact Configuration	Click **Add Contact** to add existing platform users to the contact group. Ensure the accuracy of the selected contacts' contact information (phone, email, interface callback) to avoid missing message notifications.

4. Click **Add**.

Associate a Notification Contact Group with a Header Secret

If the configured Webhook endpoint requires custom request headers, associate the contact group with the Secret created in Configure Custom Request Headers for Webhook Notifications.

1. In the left navigation bar, click **Cluster Management** > **Cluster**.
2. Click the operation button next to the `global` cluster > **CLI Tool**.
3. Edit the corresponding `NotificationGroup` resource on the master node of the `global` cluster.

```
kubectl edit notificationgroups.ait.alauda.io -n cpaas-system <notification-group-name>
```

4. Add the `cpaas.io/notification.webhook.config` annotation to `metadata.annotations`. Its value must be the name of the Secret created for the custom request headers.

```
apiVersion: ait.alauda.io/v1beta1
kind: NotificationGroup
metadata:
  name: <notification-group-name>
  namespace: cpaas-system
  annotations:
    cpaas.io/notification.webhook.config: webhook-header-secret
```

INFO

The Webhook URL is configured in the notification contact group, while custom request headers are configured through the referenced Secret.

Notification Template

A notification template is a standardized structure composed of custom content, content variables, and content format parameters. It is used to standardize the content and format of alert notification messages for notification strategies.

Platform administrators or operations personnel can set notification templates to customize the content and format of notification messages based on different alert notification methods, helping users quickly get critical alert information and improve operational efficiency.

> **INFO**
>
> The platform supports various notification servers, and the corresponding notification type templates will be displayed according to the notification server configuration. If no notification server is configured, the corresponding notification templates will not be displayed by default.

Create Notification Template

1. In the left navigation bar, click **Operations Center** > **Notifications**.
2. Switch to the **Notification Template** tab.
3. Click **Create Notification Template**.
4. In the **Basic Information** section, configure the following parameters.

Parameter	Description
Message Type	Select the type of message according to the purpose of the notification. **Alert Message**: Sends alert messages triggered by alert rules, in conjunction with the platform's alerting functionality; **Component Exception Message**: Sends notification information triggered by exceptions in certain components.

5. In the **Template Configuration** section, reference different template types to configure variables and content formatting parameters.

> **INFO**

1. The content of the template can only consist of variables, variable display names, and special formatting markup language supported by the platform. Variables and other elements can be freely combined as long as they comply with the syntax rules.
2. Only variables supported by the platform can be used in the template. You can modify variable display names and content formats, but you cannot modify the variable itself. Refer to Reference Variables, and Special Formatting Markup Language in Emails.
3. The platform provides default notification template content for various notification types based on actual operational scenarios, which can meet most notification message setting needs. If there are no special requirements, you may directly use the default template content.

6. Click **Create**.

Reference Variables

Variables are the keys of labels or annotations in notification messages (NotificationMessage), formatted as `{{.labelKey}}`. To facilitate users in quickly obtaining key information, custom display names can be assigned to variables; for example: `Alert Level: {{ .externalLabels.severity }}`.

When a notification rule sends notification messages to users based on a notification template, the variables in the template will reference the corresponding label values in the notification message (actual monitoring data). Ultimately, monitoring data will be sent to users in a standardized content format.

The platform provides the following basic variables by default:

Display Name	Variable	Description
Alert Status	`{{ .externalLabels.status }}`	For example: Alerting.
Alert Level	`{{ .externalLabels.severity }}`	For example: Critical.
Alert Cluster	`{{ .labels.alert_cluster }}`	For example: Cluster 1 where the alert occurred.

Display Name	Variable	Description
Alert Object	`{{ .externalLabels.object }}`	The type and name of the resource where the alert occurred, e.g., node 192.168.16.53.
rule Name	`{{ .labels.alert_resource }}`	The name of the alert rule, e.g., cpaas-node-rules.
Alert Description	`{{ .externalLabels.summary }}`	Description of the alert rule.
Trigger Value	`{{ .externalLabels.currentValue }}`	The monitored value that triggered the alert.
Alert Time	`{{ dateFormatWithZone .startsAt "2006-01-02 15:04:05" "Asia/Chongqing" }}`	The start time of the alert.
Recovery Time	`{{ dateFormatWithZone .endsAt "2006-01-02 15:04:05" "Asia/Chongqing" }}`	The end time of the alert.
Metric Name	`{{ .labels.alert_indicator }}`	Name of the monitoring metric.

Special Formatting Markup Language in Emails

In email notifications, common HTML format tags and their instructions are referenced in the table below:

Content Element	Tag	Description
Text	-	Supports input of Chinese/English text content.

Content Element	Tag	Description
Font	`<font color="#FF0000">Set Font Color</font>` `<b>Bold Font</b>`	Set font format.
Title	`<h1>Level 1 Title</h1>`, supports up to h6 (header 6).	Set title level.
Paragraph	`<p>Paragraph</p>`	Insert regular paragraph text.
Quote	`<q>Quote</q>`	Insert short quoted content.
Hyperlink	`<a href="//www.example.com">Hyperlink</a>`	Insert a hyperlink.

Notification rule

A notification rule is a collection of rules defining how to send notification messages to specific contacts. It is essential to use notification strategies for scenarios requiring notification to external services, such as alerts, inspections, and login authentication.

> **INFO**
>
> The platform supports various notification servers, and the notification modes corresponding to notification types will be displayed based on the notification server configuration. If no notification server is configured, the corresponding notification modes will not be displayed by default.

Prerequisites

To use the **Corporate Communication Tool Server** to notify contacts, users must first modify their contact information in **Personal Information** by entering their `WeChat Work ID`.

Operation Procedures

1. In the left navigation bar, click **Operations Center** > **Notifications**.
2. Click **Create Notification rule** and configure the relevant parameters as per the following instructions.

Parameter	Description
Notification Contact Group	A notification contact group is a logical set of notification recipients, which the platform will notify using the specified notification method.
Notification Recipients	Choose to add one or more notification recipients, and the platform will send notifications according to the recipients' **Personal Information** contact methods.
Notification Method	Supports multiple methods including **WeChat Work**, **DingTalk**, **Feishu**, **Corporate WeChat Group Bot**, **DingTalk Group Bot**, **Feishu Group Bot**, **WebHook URL**, and supports multiple selections. **Note**: Some parameters will be displayed after configuring the notification server.
Notification Template	Select the notification template to display notification information.

3. Click **Create**.

Set Notification Rule for Projects

The platform's notification strategies, notification templates, and notification contact groups are tenant-isolated. As a project administrator, you will not be able to view or use notification strategies, notification templates, or notification contact groups configured by other projects or platform administrators. Therefore, you need to refer to this document to configure suitable notification strategies for your project.

Prerequisites

1. You have contacted the platform administrator to complete the notification server setup.
2. If you need to notify through corporate communication tools, you also need to ensure that the contacts to be notified have correctly configured their communication tool IDs in **Personal Information**.

Operation Procedures

1. In the **Project Management** view, click ***Project Name***.
2. In the left navigation bar, click **Notifications**.
3. Switch to the **Notification Contact Group** tab, refer to Notification Contact Group to create a notification contact group.

> **TIP**
>
> If you do not need to manage notification contacts through a notification contact group or do not need to notify a webhook type notification server, you can skip this step.

4. Switch to the **Notification Template** tab, refer to Notification Template to create a notification template.
5. Switch to the **Notification rule** tab, refer to Notification rule to create a notification rule.

Management of Monitoring Dashboards

TOC

Function Overview

The platform provides powerful dashboard management functionality designed to replace traditional Grafana tools, offering users a more comprehensive and flexible monitoring experience. This feature aggregates various monitoring data from within the platform, presenting a unified monitoring view that significantly enhances your configuration efficiency.

Main Features

- Supports configuring custom monitoring dashboards for both business views and platform views.
- Enables viewing publicly shared dashboards configured in platform views from business views, with data isolated based on the namespace to which the business belongs.
- Supports managing panels within the dashboard, allowing users to add, delete, modify panels, zoom in/out panels, and move panels through drag-and-drop.
- Allows setting custom variables within the dashboard for filtering query data.
- Supports configuring groups within the dashboard for managing the panels. Groups can be displayed repeatedly based on custom variables.
- Supported panel types include: trend、step line chart、bar chart、horizontal bar chart、bar gauge chart、gauge chart、table、stat chart、XY chart、pie chart、text.
- One-click import feature for Grafana dashboards.

Advantages

- Supports user-customized monitoring scenarios without being constrained by predefined templates, truly achieving a personalized monitoring experience.

- Provides a rich array of visualization options, including line charts, bar charts, pie charts, and flexible layout and styling options.
- Integrates seamlessly with the platform's role permissions, allowing business views to define their own monitoring dashboards while ensuring data isolation.
- Deep integration with various functionalities of the container platform, enabling instant access to monitoring data for containers, networks, storage, etc., providing users with comprehensive performance observation and fault diagnosis.
- Fully compatible with Grafana dashboard JSON, allowing easy migration from Grafana for continued use.

Use Cases

- **IT Operations Management**: As part of the IT operations team, you can use the monitoring dashboards to unify the display and management of various performance metrics of the container platform, such as CPU, memory, network traffic, etc. By customizing monitoring reports and alert rules, you can promptly detect and pinpoint system issues, enhancing operational efficiency.
- **Application Performance Analysis**: For application developers and testers, monitoring dashboards offer various rich visualization options to intuitively display application running states and resource consumption. You can customize dedicated monitoring views tailored to different application scenarios to deeply analyze application performance bottlenecks and provide a basis for optimization.
- **Multi-Cluster Management**: For users managing multiple container clusters, monitoring dashboards can aggregate monitoring data from disparate clusters, allowing you to grasp the overall operational state of the system at a glance.
- **Fault Diagnosis**: When a system issue occurs, monitoring dashboards provide you with comprehensive performance data and analytical tools to quickly pinpoint the root cause of the problem. You can swiftly view fluctuations in relevant monitoring metrics based on alert information for in-depth fault analysis.

Prerequisites

Currently, monitoring dashboards only support viewing monitoring data collected by monitoring components installed in the platform. Therefore, you should prepare as follows

before configuring a monitoring dashboard:

- Ensure that the cluster for which you want to configure the monitoring dashboard has monitoring components installed, specifically the **ACP Monitor with Prometheus** or **ACP Monitor with VictoriaMetrics** plugin.
- Ensure that the data you wish to display on the dashboard has been collected by the monitoring components.

Relationship Between Monitoring Dashboards and Monitoring Components

- Monitoring dashboard resources are stored in the Kubernetes cluster. You can switch views between different clusters using the **Cluster** tab at the top.
- Monitoring dashboards depend on the monitoring components in the cluster for querying data sources. Therefore, before using monitoring dashboards, ensure that the current cluster has successfully installed monitoring components and that they are operating normally.
- The monitoring dashboard will default to requesting monitoring data from the corresponding cluster. If you install the **VictoriaMetrics** plugin in proxy mode in the cluster, we will request the storage cluster for you to query the corresponding data for this cluster without the need for special configuration.

Manage Dashboards

A dashboard is a collection composed of one or more panels, organized and arranged in one or more rows to provide a clear view of relevant information. These panels can query raw data from data sources and transform it into a series of visual effects supported by the platform.

Create a Dashboard

1. Click **Create Dashboard**, reference the following instructions to configure relevant parameters.

Parameter	Description
Folder	The folder where the dashboard resides; you can input or select an existing folder.
Label	Label for the monitoring dashboard; you can quickly find existing dashboards by filtering through the top labels during the switch.
Set as Main Dashboard	If enabled, this will set the current dashboard as the main dashboard upon successful creation; when re-entering the monitoring dashboard feature, the main dashboard data will be displayed by default.
Variables	Add variables when creating the dashboard to reference as metric parameters in the added panels, which can also be used as filters on the dashboard homepage.

2. After adding, click **Create** to finish creating the dashboard. Next, you need to **add variables**, **add panels**, and **add groups** to complete the overall layout design.

Import Dashboard

The platform supports direct import of Grafana JSON to convert it into a monitoring dashboard for display.

- Currently, only Grafana JSON of version V8+ is supported; lower versions will be prohibited from being imported.
- If any panels within the imported dashboard are not within the platform's supported scope, they may be displayed as **unsupported panel types**, but you can modify the panel's settings to achieve normal display.
- After importing the dashboard, you can perform any management actions as usual, which will not differ from panels created in the platform.

Add Variables

1. In the variable form area, click **Add**.

Query

Variables of type `Query` allow you to filter data based on the feature dimensions of time series. The query expression can be specified to dynamically calculate and generate query results.

Parameter	Description
Query Settings	When defining query settings, besides using PromQL to query time series, the platform also provides some common variables and functions. Reference Common Functions and Variables.
Regular Expression	By using regular expressions, you can filter out the desired values from the content returned by the variable queries. This makes each option name in the variable more expected. You can preview if the filtered values meet expectations in **Variable Value Preview**.
Selection Settings	- **Multiple Selection**: When selected from the top filters on the dashboard homepage, allows the selection of multiple options simultaneously. You need to reference this variable in the query expression of the panels to view the data corresponding to the variable value. - **All**: If checked, an option containing **All** will be enabled in the filter options to select all variable data.

Constant

Constant Variables are static variables with fixed values that remain unchanged throughout the dashboard, commonly used for storing environment identifiers, fixed thresholds, or configuration parameters that need to be referenced across multiple panels without displaying as filter options.

Parameter	Description
Constant Value	The value of the constant variable.

Custom

Custom Variables allow users to define a predefined list of static options that appear as dropdown filters on the dashboard, commonly used for manual selection of specific services, teams, or categories without requiring dynamic data queries.

Parameter	Description
Custom Settings	Enter option values separated by commas, using the format display_name : value for each option (e.g., Production : prod, Staging : stage, Development : dev), or simply list values directly if display name equals value.

Textbox

Textbox Variables are variables that allow users to enter text directly, commonly used for specifying specific values or parameters that do not require dynamic data queries.

Parameter	Description
Textbox Value	The default value of the textbox variable.

2. Click **OK** to add one or more variables.

Add Panels

Add multiple panels to the currently created monitoring dashboard to display data information for different resources.

Tip: You can customize the size of a panel by clicking the lower right corner; click anywhere on the panel to rearrange the order of the panels.

1. Click **Add Panel**, reference the following instructions to configure relevant parameters.

- **Panel Preview**: The area will dynamically display the data information corresponding to the added metrics.
- **Add Metric**: Configure the panel title and monitoring metrics in this area.
- **Adding Method**: Supports using built-in metrics or using natively customized metrics. Both methods will take the union and be effective simultaneously.
  - **Built-in Metrics**: Select commonly used metrics and legend parameters built into the platform to display the data information under the current panel.
    - **Note**: All metrics added to the panel must have a unified unit; it is not possible to add metrics with multiple units to one panel.

  - **Native**: Customize the metric unit, metric expression, and legend parameters. The metric expression follows PromQL syntax; for details, please refer to PromQL Official Documentation ↗.

- **Legend Parameters**: Control the names corresponding to the curves in the panels. Text or templates can be used:

  - Rule: The input value must be in the format `{{.xxxx}}`; for example, `{{.hostname}}` will replace it with the value corresponding to the hostname label returned by the expression.
  - **Tip**: If you input an incorrectly formatted legend parameter, the names corresponding to the curves in the panel will be displayed in their original format.

- **Instant Switch**: When the **Instant** switch is turned on, it will query instant values through Prometheus's Query interface and sort them, as in statistical charts and gauge charts. If off, it will use the `query_range` method to calculate, querying a series of data over a specific time period.
- **Panel Settings**: Supports selecting different panel types for visualizing metric data. Please refer to Manage Panels.

2. Click **Save** to complete adding the panels.
3. You can add one or more panels within the dashboard.
4. After adding the panels, you can use the following operations to ensure the display and size of the panels meet your expectations.

   - Click the lower right corner of the panel to customize its size.
   - Click anywhere on the panel to rearrange the order of the panels.
   - Click the **Edit** button to modify the panel settings.
   - Click the **Delete** button to delete the panel.
   - Click the **Copy** button to copy the panel.

5. After adjusting, click the **Save** button on the dashboard page to save your modifications.

Add Groups

Groups are logical dividers within the dashboard that can group panels together.

1. Click the **Add Panel** drop-down menu > **Add Group**, and reference the following instructions to configure relevant parameters.

- **Group**: The name of the group.
- **Repeat**: Supports disabling repeats or selecting variables for the current panels.
  - **Disable Repeat**: Do not select a variable, and use the default created group.
  - **Parameter Variables**: Select the variables created in the current panels, and the monitoring dashboard will generate a row of identical sub-groups for each corresponding value of the variable. Sub-groups do not support modifications, deletions, or moving of the panels.

2. After adding the group, you can perform the following operations on the group to manage the panel display within the dashboard.

- Groups can be collapsed or expanded to hide part of the content in the dashboard. Panels within collapsed groups will not send queries.
- Move the panel into the group to allow that panel to be managed by that group. The group will manage all panels between it and the next group.
- When a group is folded, you can also move all panels managed by that group together.
- The folding and unfolding of groups also constitutes an adjustment to the dashboard. If you want to maintain this state when reopening this dashboard next time, please click the **Save** button.

Switch Dashboards

Set the created custom monitoring dashboard as the main dashboard. When re-entering the monitoring dashboard feature, the main dashboard data will be displayed by default.

1. In the left navigation bar, click **Operations Center** > **Monitoring** > **Monitoring Dashboards**.
2. By default, the main monitoring dashboard is entered. Click **Switch Dashboard**.
3. You can find dashboards by filtering through labels or searching by name, and switch main dashboards via the **Main Dashboard** switch.

Other Operations

You can click the operation button on the right side of the dashboard page to perform actions on the dashboard as needed.

Operation	Description
YMAL	Opens the actual CR resource code of the dashboard stored in the Kubernetes cluster. You can modify all content in the dashboard by editing parameters in the YAML.
Export Expression	You can export the metrics and corresponding query expressions used in the current dashboard in CSV format.
Copy	Copies the current dashboard; you can edit the panels as needed and save it as a new dashboard.
Settings	Modifies the basic information of the current dashboard, such as changing labels and adding more variables.
Delete	Deletes the current monitoring dashboard.

Manage Panels

The platform provides various visualization methods to support different use cases. This chapter will mainly introduce these panel types, configuration options, and usage methods.

Panel Description

No.	Panel Name	Description	Suggested Use Cases
1	Trend Chart	Displays the trend of data over time via one or more line segments.	Shows trends over time, such as changes in CPU utilization, memory usage, etc.
2	Step Line Chart	Builds on the line chart by connecting data points with horizontal and vertical segments	Suitable for displaying the timestamps of discrete events, such as the number of alerts.

No.	Panel Name	Description	Suggested Use Cases
		to form a step-like structure.	
3	Bar Chart	Uses vertical rectangular bars to represent the magnitude of data, where the height of the bars represents value.	Bar charts are intuitive for comparing value differences, beneficial for discovering patterns and anomalies, suitable for scenarios focusing on value changes, such as the number of pods, number of nodes, etc.
4	Horizontal Bar Chart	Similar to the bar chart but uses horizontal rectangular bars to represent data.	When there are many data dimensions, horizontal bar charts can better utilize spatial layout and improve readability.
5	Gauge Chart	Uses half or ring shapes to represent the current value of an indicator and its proportion of the total.	Intuitively reflects the current status of key monitoring indicators, such as system CPU utilization and memory usage. It is recommended to use alert thresholds with color changes to indicate abnormal conditions.
6	Gauge Bar Chart	Uses vertical rectangular bars to display the current value of indicators and their proportion.	Intuitively reflects the current status of key indicators, such as target completion progress and system load. When multiple categories of the same indicator exist, the gauge bar chart is more recommended, such as available disk space or utilization.
7	Pie Chart	Uses sectors to display the proportional relationship of parts to the whole.	Suitable for demonstrating the composition of overall data across different dimensions, such as the proportions of 4XX, 3XX, and 2XX response codes over a period.

No.	Panel Name	Description	Suggested Use Cases
8	Table	Organizes data in a row-column format, making it easy to view and compare specific values.	Suitable for displaying structured multi-dimensional data, such as detailed information of nodes, detailed information of pods, etc.
9	Stat Chart	Displays the current value of a single key indicator, typically requiring textual explanation.	Suitable for showing real-time values of important monitoring indicators, such as numbers of pods, number of nodes, current alert count, etc.
10	Scatter Plot	Uses Cartesian coordinates to plot a series of data points, reflecting the correlation between two variables.	Suitable for analyzing relationships between two indicators, discovering patterns such as linear correlation and clustering through the distribution of data points, helping unearth relationships between metrics.
11	Text Card	Displays key textual information in a card format, usually containing a title and a brief description.	Suitable for presenting textual information, such as panel descriptions and troubleshooting explanations.

Panel Configuration Description

General Parameters

Parameter	Description
Basic Information	Select the appropriate panel type based on the selected metric data and add titles and descriptions; you can add one or more links, which can be quickly accessed by selecting the corresponding link name next to the title.
Standard Settings	Units used for native metric data. Additionally, gauge charts and gauge bars also support configuring the **Total Value** field, which will display as the

Parameter	Description
	percentage of **Current Value/Total Value** in the chart.
Tooltips	Tooltips are the display switch for real-time data when hovering over the panels and support selected sorting.
Threshold Parameters	Configure the threshold switch for the panels; when enabled, the threshold will be shown in selected colors in the panels, allowing for threshold sizing.
Value	Set the calculation method for values, such as the most recent value or minimal value. This configuration option is only applicable to stat charts and gauge charts.
Value Mapping	Redefine specified values, ranges,regex or special such as defining 100 as full load. This configuration option is only applicable to stat charts, tables, and gauge charts.

Special Parameters for Panels

Panel Type	Parameter	Description
Trend Chart	**Graph Style**	You can choose between a line chart or an area chart as the display style; line charts focus more on reflecting the trend changes of indicators, while area charts draw more attention to changes in total and partial proportions. Choose based on your actual needs.
Gauge Chart	**Gauge Chart Settings**	**Show Direction**: When you need to view multiple metrics in a single chart, you can set whether these metrics are arranged horizontally or vertically. **Unit Redefinition**: You can set independent units for each metric; if not set, the platform will display units from the **Standard Settings**.
Stat Chart	**Stat Chart Settings**	**Show Direction**: When you need to view multiple metrics in a single chart, you can set whether these metrics are arranged horizontally or vertically.

Panel Type	Parameter	Description
		Graph Mode: You can add a graph to the stat chart to display the trend of the metric over time.
Pie Chart	**Pie Chart Settings**	**Maximum Number of Slices**: You can set this parameter to reduce the number of slices in the pie chart to lessen the interference of categories with comparatively low proportions but high quantities. Excess slices will be merged and displayed as **Others**. **Label Display Fields**: You can set the fields displayed in the pie chart labels.
Pie Chart	**Graph Style**	You can choose either pie or donut as the display style.
Table	**Table Settings**	**Hide Columns**: You can reduce the number of columns in the table with this parameter to focus on some primary column information. **Column Alignment**: You can modify the alignment of data within the column using this parameter. **Display Name and Unit**: You can modify the column names and units used through this parameter.
Text Card	**Graph Style**	**Style**: You can choose to edit the content you wish to display in the text card in either a rich-text editing box or HTML.

Create Monitoring Dashboards via CLI

1. Create a new YAML configuration file named `example-dashboard.yaml`.
2. Add the MonitorDashboard resource to the YAML file and submit it. The following example creates a monitoring dashboard named demo-v2-dashboard1:

```
kind: MonitorDashboard
apiVersion: ait.alauda.io/v1alpha2
metadata:
  annotations:
    cpaas.io/dashboard.version: '3'
    cpaas.io/description: '{"zh":"描述信息","en":""}' # Description fiel
d
    cpaas.io/operator: admin
  labels:
    cpaas.io/dashboard.folder: demo-v2-folder1 # Folder
    cpaas.io/dashboard.is.home.dashboard: 'False' # Is it the main dash
board?
  name: demo-v2-dashboard1 # Name
  namespace: cpaas-system # Namespace (all management view creations wi
ll occur in this ns)
spec:
  body: # All information fields
    titleZh: 更新显示名称 # Built-in field for Chinese display name (this
field is created under the Chinese language)
    title: english_display_name # Built-in field for English display na
me (this field is created under the English language) Built-in dashboar
ds can set bilingual translations.
    templating: # Custom variables
      list:
        - hide: 0 # 0 means not hidden; 1 means only the label is hidde
n; 2 means both label and value are hidden
          label: 集群 # Built-in variable display name (label is set to
the appropriate name based on the language, e.g., cluster in English)
          name: cluster # Built-in variable name (unique)
          options: # Define dropdown options; if a query retrieves dat
a, it will use requested data; otherwise, it will use options. A defaul
t value can be set (generally only used for setting default values)
            - selected: false # Whether to default select
              text: global
              value: global
          type: custom # Custom variable type; currently, only built-in
(custom) and query are supported (Importing Grafana will support consta
nt custom interval (after import, it will be changed to a custom variab
le and will not support auto))

        - allValue: '' # Select all, passing options with the format xx
x|xxx|xxx; can set allValue for conversion (Grafana retrieves all data
for the current variable as xxx|xxx|xxx, adjustments will ensure consis
```

```
tency)
          current: null # Current value of the variable; if not set, de
faults to the first in the list
          definition: query_result(kube_namespace_labels) # Query expre
ssion for data retrieval
          hide: 0 # 0 means not hidden; 1 means only the label is hidde
n; 2 means both label and value are hidden
          includeAll: true # Whether to select all
          label: ns # Built-in variable display name
          multi: true # Whether multiple selections are allowed
          name: ns # Variable name (unique)
          options: []
          query: ''
          regex: /.*namespace=\"(.*?)\".*/ # Regex expression for extra
cting variable values
          sort: 2 # Sorting: 1 - ascending alphabetical order; 2 - desc
ending alphabetical order (only these two support temporarily); 3 - asc
ending numerical order; 4 - descending numerical order
          type: query # Custom variable type
    time: # Dashboard time
      from: now-30m # Start time
      to: now # End time
    repeat: '' # Row repeat configuration; chooses custom variable
    collapsed: 'false' # Row collapsed or expanded configuration
    description: '123' # Description (tooltip after title)
    targets: # Data sources
      - indicator: cluster.node.ready # Metric
        expr: sum (cpaas_pod_number{cluster=\"\"}>0) # PromQL expressio
n
        instant: false # Query mode true retrieves data at a specific t
ime
        legendFormat: '' # Legend
        range: true # Default querying range when retrieving data
        refId: 指标1 # Unique identifier for display name of data source
    gridPos: # Information on the dashboard's positional layout
      h: 8 # Height
      w: 12 # Width (width corresponds to 24 grid units)
      x: 0 # Horizontal position
      y: 0 # Vertical position
    panels: # Panel data
      title: 图表标题tab # Panel name
      type: table # Panel type; currently supports timeseries, barchar
t, stat, gauge, table, bargauge, row, text, pie (step chart, scatter pl
ot, bar chart, configurable through drawStyle attribute)
```

```
      id: a2239830-492f-4d27-98f3-cb7ecb77c56f # Unique identifier
      links: # Links
        - targetBlank: true # Open in a new tab
          title: '1' # Name
          url: '1' # URL address
      transformations: # Data transformations
        - id: 'organize' # Type organize; used for sorting, rearranging order, showing fields, whether to display
          options:
            excludeByName: # Hidden fields
              cluster_cpu_utilization: true
            indexByName: # Sort
              cluster_cpu_utilization: 0,
              Time: 1
            renameByName: # Rename
              Time: ''
              cluster_cpu_utilization: '222'
        - id: 'merge' # Merging data
          options:
      fieldConfig: # For defining panel properties and appearance
        defaults: # Default configuration
          custom: # Custom graphic attributes
            align: 'left' # Table alignment: left, center, right
            cellOptions: # Table threshold configuration
              type: color-text # Only supports text for threshold color settings
            spanNulls: false # true connects null values; false does not connect; number == 0 connects null values according to 0
            drawStyle: line # Panel types: line, bars for bar charts, points for point charts
            fillOpacity: 20 # Exists when drawStyle is area (currently does not support configuration, area defaults to 20)
            thresholdsStyle: # Configures how to display thresholds (currently only supports line)
              mode: line # Threshold display format (area not supported currently)
            lineInterpolation: 'stepBefore' # Step chart configuration; defaults to only supporting stepBefore (stepAfter will be supported later)
          decimals: 3 # Decimal points
          min: 0 # Minimum value (currently not supported for page configuration, only supports imports that have been adapted)
          max: 1 # Maximum value (page configuration only applies to stat gauge barGauge pie)
```

```
          unit: '%' # Unit
          mappings: # Value mapping configuration (currently only suppo
rts value and range types; special types supported on data)
            - options: # Value mapping rules
                '1': # Corresponding value
                  index: 0
                  text: 'Running' # Displayed as Running when value is
1
              type: value # Value mapping type
            - options: # Range mapping rules
                from: 2 # Range start value
                to: 3 # Range end value
                result: # Mapping result
                  index: 1
                  text: 'Error' # Values from 2 to 3 will display as Er
ror
              type: range # Mapping type for range
            - type: special # Mapping type for special scenarios
              options:
                match: null # nan null null+nan empty true false
                result:
                  text: xxx
                  index: 2
          thresholds: # Threshold configuration
            mode: absolute # Threshold configuration mode, absolute val
ue mode (currently only supports absolute and percentage mode; percenta
ge mode is not supported yet)
            steps: # Threshold steps
              - color: '#a7772f' # Threshold color
                value: '2' # Threshold value
              - color: '#007AF5' # Default value with no value is the B
ase
        overrides: # Override configuration
          - matcher:
              id: byName # Match based on field name
              options: node # Corresponding name
            properties: # Override configuration; id currently only sup
ports displayName unit
              - id: displayName # Display name override
                value: '1' # Overridden display name
              - id: unit # Unit override
                value: GB/s # Unit value
              - id: noValue # No value display
                value: No value display
```

```
      options:
        orientation: horizontal # Control the layout direction of panel
s; applies to gauge and barGauge (stat will be supported later)
        legend: # Legend configuration
          calcs: # Calculating methods (only displays when the legend p
osition is on the right)
            - latest # Currently only supports most recent value
          placement: right # Legend position (right or bottom; defaults
to bottom)
          placementRightTop: '' # Configuration for the upper right
          showLegend: true # Whether to display the legend
        tooltip: # Tooltips
          mode: multi # Mode dual selection (only multi-mode supported)
All data displayed when the mouse hovers over
          sort: asc # Sorting: asc or desc
        reduceOptions: # Value calculating method (used for aggregating
data)
          calcs: # Calculating methods (latest, minimum, maximum, avera
ge, sum)
            - latest
          limit: 3 # Pie limits the number of slices
        textMode: 'value' # Stat configuration; defines style for displ
aying metric value; options are auto, value, value_and_name, name, none
(currently not supported in the page configuration, but supported in im
ports)
        colorMode: 'value' # Stat configuration; defines color mode for
displaying metric values; options are none, value, background (defaults
to value; not supported in configuration but adapted in import)
        displayLabels: ['name', 'value', 'percent'] # Fields displayed
in pie chart labels
        pieType: 'pie' # Pie chart type; options are pie and donut
        mode: 'html' # Text chart type mode; options are html and richT
ext
        content: '<div>xxx</div>' # Content for text chart type
        footer:
          enablePagination: true # Table pagination enabled
```

Common Functions and Variables

Common Functions

When defining query settings, besides using PromQL to set queries, the platform provides some common functions as follows for your reference in customizing query settings.

Function	Purpose
label_names()	Returns all labels in Prometheus, e.g., label_names().
label_values(label)	Returns all selectable values for the label name in all monitored metrics in Prometheus, e.g., label_values(job).
label_values(metric, label)	Returns all selectable values for the label name in the specified metric in Prometheus, e.g., label_values(up, job).
metrics(metric)	Returns all metric names that satisfy the defined regex pattern in the metric field, e.g., metrics(cpaas_active).
query_result(query)	Returns the query result for the specified Prometheus query, e.g., query_result(up).

Common Variables

While defining query settings, you can combine common functions into variables to quickly define custom variables. Here are some common variable definitions available for your reference:

Variable Name	Query Function
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)`

Variable Name	Query Function
daemonset	`label_values(kube_daemonset_status_number_ready{namespace="$namespace"} daemonset)`
statefulset	`label_values(kube_statefulset_replicas{namespace="$namespace"}, statefulset)`
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)`

Variable Use Case One

Using the **query_result(query)** function to query the value: `node_load5`, and extract the IP.

1. In **Query Settings**, fill in `query_result(node_load5)`.
2. In the **Variable Value Preview** area, the preview example is `node_load5{container="node-exporter",endpoint="metrics",host_ip="192.168.178.182",instance="192.168.178.182:9100"}`.
3. In **Regular Expression**, fill in `/.*instance="(.*?):.*/` to filter the value.
4. In the **Variable Value Preview** area, the preview example is `192.168.176.163`.

Variable Use Case Two

1. Add the first variable: namespace, using the **query_result(query)** function to query the value: `kube_namespace_labels`, and extract the namespace.
   - **Query Settings**: `query_result(kube_namespace_labels)`.
   - **Variable Value Preview**: `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"}`.

- **Regular Expression**: `/.+namespace=\"(.*?)\".*/`.
- In the **Variable Value Preview** area, the preview example includes multiple namespaces such as `argocd`, `cpaas-system`, and more.

2. Add the second variable: deployment, and reference the variable created earlier:

- **Query Settings**: `kube_deployment_spec_replicas{namespace=~"$namespace"}`.
- **Regular Expression**: `/.+deployment="(.*?)",.*/`.

3. Add a panel to the current dashboard and reference the previously added variables, for example:

- Metric Name: **pod Memory Usage under Compute Components**.
- Key-Value Pair: `kind`: `Deployment`, `name`: `$deployment`, `namespace`: `$namespace`.

4. Once you have added the panels and saved them, you can view the corresponding panel information on the dashboard homepage.

Notes When Using Built-in Metrics

> **WARNING**
>
> The following metrics use custom variables `namespace`, `name`, and `kind`, which do not support **multiple selections** or selecting **all**.
>
> - `namespace` only supports selecting a specific namespace;
> - `name` only supports three types of computing components: `deployment`, `daemonset`, `statefulset`;
> - `kind` only supports specifying one of the types: `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`

Management of Probe

TOC

Function Overview

The probe feature of the platform is realized based on Blackbox Exporter, allowing users to probe the network via ICMP, TCP, or HTTP to quickly identify faults occurring on the platform.

Unlike white-box monitoring systems, which rely on various monitoring metrics already available on the platform, blackbox monitoring focuses on the outcomes. When white-box

monitoring cannot cover all factors affecting service availability, blackbox monitoring can swiftly detect faults and issue alerts based on those faults. For example, if an API endpoint is abnormal, blackbox monitoring can promptly expose such issues to users.

> **WARNING**
>
> The probe function does not support using ICMP to detect IPv6 addresses on nodes with kernel versions 3.10 and below. To use this scenario, please upgrade the kernel version on the node to 3.11 or higher.

Blackbox Monitoring

To create a blackbox monitoring item, you can choose the ICMP, TCP, or HTTP probing method to periodically probe the specified target address.

Prerequisites

The monitoring components must be installed in the cluster, and the monitoring components must be functioning properly.

Procedures for Operation

1. In the left navigation bar, click **Operations Center** > **Monitoring** > **Blackbox Monitoring**.
   Tip: Blackbox monitoring is a cluster-level feature. Click on the top navigation bar to switch between clusters.
2. Click **Create Blackbox Monitoring Item**.
3. Refer to the following instructions to configure the relevant parameters.

Parameter	Description
Probing Method	**ICMP**: Probes by pinging the domain name or IP address entered in the **Target Address** to check the server's availability. **TCP**: Probes the business port of the host by listening on the `<domain:port>` or `<IP:port>` specified in the **Target Address**.

Parameter	Description
	HTTP: Probes the URL entered in **Target Address** to check website connectivity. **Tip**: The HTTP probing method only supports GET requests by default; for POST requests, please refer to Customizing the BlackboxExporter Monitoring Module.
Probing Interval	The interval time for probing.
Target Address	The target address for probing, with a maximum of 128 characters. The input format for the target address varies by probing method: **ICMP**: A domain name or IP address, e.g., `10.165.94.31`. **TCP**: `<domain:port>` or `<IP:port>`, e.g., `172.19.155.133:8765`. **HTTP**: A URL that starts with http or https, e.g., `http://alauda.cn/`.

4. Click **Create**.
   Once created successfully, you can view the latest probing results in real time on the list page, and based on the blackbox monitoring items, you can create alert policies. When a fault is detected, an alert will be automatically triggered to notify the relevant personnel for resolution.

> **WARNING**
>
> After successfully creating the blackbox monitoring items, the system requires about 5 minutes to synchronize the configuration. During this synchronization period, probing will not occur and probing results cannot be viewed.

Blackbox Alerts

Prerequisites

- The monitoring components must be installed in the cluster, and the monitoring components must be functioning properly.

- The blackbox monitoring item must have been successfully created, and the system must have finished synchronizing the configuration so that probing results are visible on the blackbox monitoring page.

Procedures for Operation

1. In the left navigation bar, click **Operations Center** > **Alerts** > **Alert Policies**.
   Tip: Alert policies are a cluster-level feature. Click on the top navigation bar to switch between clusters. Please ensure you switch to the cluster where the blackbox monitoring item has just been configured.
2. Click **Create Alert Policy**.
3. Refer to the following instructions to configure the relevant parameters; for more parameter information, please refer to Create Alert Policies.

- **Alert Type**: Please select **Resource Alert**.
- **Resource Type**: Please select **Cluster**.
- Click **Add Alert Rule**.
  - **Alert Type**: Please select **Blackbox Alert**.
  - **Blackbox Monitoring Item**: Please select the desired blackbox monitoring item.
  - **Metric Name**: Please select the metric you wish to monitor and alert on. The current supported metrics by the platform are **Connectivity** and **HTTP Status Code**.
    - **Connectivity**: This metric can be selected for all blackbox monitoring items, where the trigger condition **“!= 1”** indicates that the target address of the blackbox monitoring item is unreachable.
    - **HTTP Status Code**: This metric can be selected when the probing method of the chosen blackbox monitoring item is **HTTP**. You can input a three-digit positive integer as the value for the trigger condition, for example, if the condition is set to **“> 299”**, it means alerts are fired when the response codes are 3XX, 4XX, or 5XX.
  - **Notification Policy**: Please select your pre-configured policy.
  - Click **Add**.

4. Click **Create**. After the alert policy submission, you can see this alert policy in the alert policy list.

Customizing BlackboxExporter Monitoring Module

You can also enhance the functionalities of blackbox monitoring by adding customized monitoring modules to the BlackboxExporter configuration file. For example, by adding the **http_post_2xx** module to the configuration file, when the probing method of blackbox monitoring is set to `HTTP`, it would then be able to probe the status of POST request methods.

The configuration file for blackbox monitoring is located within the namespace where the Prometheus component of the cluster is installed, with the default name being `cpaas-monitor-prometheus-blackbox-exporter`, which can be modified as needed based on the actual name.

> **TIP**
>
> This configuration file is a ConfigMap resource related to the namespace, which can be quickly viewed and updated through the platform's management feature, **Cluster Management > Resource Management**.

Procedures for Operation

1. Update the configuration file of blackbox monitoring by adding customizable monitoring modules to **key** `modules`.
   Taking the addition of the **http_post_2xx** module as an example:

```
blackbox.yaml: |
  modules:
    http_post_2xx:                    # HTTP POST probing module
      prober: http
      timeout: 5s
      http:
        method: POST                  # Request method for probing
        headers:
          Content-Type: application/json
        body: '{}'                    # Body content sent with the probe
```

For complete YAML examples of the blackbox monitoring configuration file, please refer to Reference Information.

2. Activate the configuration by choosing one of the following methods.

   - Restart the Blackbox Exporter Component **cpaas-monitor-prometheus-blackbox-exporter** by deleting its Pod.
   - Execute the following command to call the reload API and refresh the configuration file:

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

Create Blackbox Monitoring Items and Alerts via CLI

Prerequisites

- Notification policies must be configured (if you require alert automatic notifications).
- The target cluster must have monitoring components installed.

Procedures for Operation

1. Create a new YAML configuration file named `example-probe.yaml`.
2. Add the PrometheusRule resource to the YAML file and submit it. The following example creates a new alert policy named `prometheus-liveness`:

```yaml
apiVersion: monitoring.coreos.com/v1
kind: Probe
metadata:
  annotations:
    cpaas.io/creator: jhshi@alauda.io # Creator of the probe item
    cpaas.io/updated-at: '2021-05-25T08:08:45Z' # Last update time of t
he probe item
    cpaas.io/display-name: 'Prometheus prober' # Description of the pro
be item
  creationTimestamp: '2021-05-10T02:04:33Z' # Creation time of the prob
e item
  labels:
    prometheus: kube-prometheus # Label value used for prometheus's nam
e
  name: prometheus-liveness # Name of the probe item
  namespace: cpaas-system # Namespace used for the prometheus's namespa
ce
spec:
  jobName: prometheus-liveness # Name of the probe item
  prober:
    url: cpaas-monitor-prometheus-blackbox-exporter:9115 # URL for Blac
kbox metrics, retrieved from features
  module: http_2xx # Name of the probe item's module
  targets:
    staticConfig:
      static:
        - http://www.prometheus.io # Target address of the probe item
      labels:
        module: http_2xx # Name of the probe item's module
        prober: http # Probing method of the probe item
  interval: 30s # Probe interval for the probe item
  scrapeTimeout: 10s
```

3. Create a new YAML configuration file named `example-alerting-rule.yaml`.
4. Add the PrometheusRule resource to the YAML file and submit it. The following example creates a new alert policy named `policy`:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  annotations:
    alert.cpaas.io/cluster: global # Name of the cluster where the aler
t resides
    alert.cpaas.io/kind: Cluster # Resource type
    alert.cpaas.io/name: global # Name of the cluster where the blackbo
x monitoring item resides
    alert.cpaas.io/namespace: cpaas-system # Namespace used for the pro
metheus's namespace, keep defaults
    alert.cpaas.io/notifications: '["test"]'
    alert.cpaas.io/repeat-config: '{"Critical":"never","High":"5m","Med
ium":"5m","Low":"5m"}'
    alert.cpaas.io/rules.description: '{}'
    alert.cpaas.io/rules.disabled: '[]'
    alert.cpaas.io/subkind: ''
    cpaas.io/description: ''
    cpaas.io/display-name: policy # Display name of the alert policy
  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 # Name of the alert rules
      rules:
        - alert: cluster.blackbox.probe.success-y97ah-9833444d918cab96c
43e9ab6efc172cf
          annotations:
            alert_current_value: '{{ $value }}' # Current value for not
ification, keep default
          expr:
            max by (job, instance) (probe_success{job=~"test",
            instance=~"https://demo.at-servicecenter.com/"})!=1
            # Connectivity alert scenario, be sure to modify the blackb
ox monitoring item name and target address
```

```
          for: 30s # Duration
          labels:
            alert_cluster: global # Name of the cluster where the alert
resides
            alert_for: 30s # Duration
            alert_indicator: cluster.blackbox.probe.success # Keep unch
anged
            alert_indicator_aggregate_range: '0' # Keep unchanged
            alert_indicator_blackbox_instance: https://demo.at-servicec
enter.com/ # Blackbox monitoring target address
            alert_indicator_blackbox_name: test # Blackbox monitoring i
tem name
            alert_indicator_comparison: '!=' # Keep configuration uncha
nged for connectivity alerts
            alert_indicator_query: '' # Used for log alerts, no need to
configure this parameter
            alert_indicator_threshold: '1' # Threshold for the alert ru
le, 1 indicates connectivity, keep unchanged
            alert_indicator_unit: '' # Unit of the alert rule's metrics
            alert_involved_object_kind: Cluster # Keep unchanged for bl
ackbox alerts
            alert_involved_object_name: global # Cluster where the blac
kbox monitoring item resides
            alert_involved_object_namespace: '' # Namespace of the obje
ct to which the alert rule belongs
            alert_name: cluster.blackbox.probe.success-y97ah # Name of
the alert rule
            alert_namespace: cpaas-system # Namespace where the alert r
ule resides
            alert_project: cpaas-system # Name of the project of the ob
ject to which the alert rule belongs
            alert_resource: policy # Name of the alert policy where the
alert rule resides
            alert_source: Platform # Type of data for the alert rule: P
latform- platform data, Business- business data
            severity: High # Alert rule level: Critical- disaster, High
- serious, Medium- warning, Low- tip
        - alert: cluster.blackbox.http.status.code-235el-99b0095b6b6669
415043e14ae84f43bc
          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 status code alert scenario, be sure to modify the bl
ackbox monitoring item name and target address
          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-servicec
enter.com/
            alert_indicator_blackbox_name: test
            alert_indicator_comparison: '>'
            alert_indicator_query: ''
            alert_indicator_threshold: '299' # Threshold for alert rule
s, in HTTP status code alert scenarios, should be a three-digit number,
for example, statuses greater than 299 (3XX, 4XX, 5XX) indicate an erro
r
            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
```

Reference Information

A complete example of the YAML configuration file for blackbox monitoring is as follows:

```
apiVersion: v1
data:
  blackbox.yaml: |
    modules:
      http_2xx_example:               # Example of HTTP probing
        prober: http
        timeout: 5s                   # Timeout for probing
        http:
          valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
# The Version in the returned information, generally defaults
          valid_status_codes: []  # Defaults to 2xx
# Range of valid response codes; if the returned code is within this rang
e, it is considered a successful probe
          method: GET                 # Request method
          headers:                    # Request headers
            Host: vhost.example.com
            Accept-Language: en-US
            Origin: example.com
          no_follow_redirects: false  # Indicates whether to allow redire
ction
          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: # Verifies that no cookies are set
            - header: Set-Cookie
              allow_missing: true
              regexp: '.*'
          fail_if_header_not_matches:
            - header: Access-Control-Allow-Origin
              regexp: '(\*|example\.com)'
          tls_config:                  # TLS configuration for https requ
ests
            insecure_skip_verify: false
          preferred_ip_protocol: "ip4" # defaults to "ip6"
# Preferred IP protocol version
          ip_protocol_fallback: false  # No fallback to "ip6"
      http_post_2xx:                   # Example of HTTP probing with Bod
y
        prober: http
        timeout: 5s
```

```
        http:
          method: POST                # Request method for probing
          headers:
            Content-Type: application/json
          body: '{"username":"admin","password":"123456"}'
# Body carried during probing
      http_basic_auth_example:        # Example of probing with username
and password
        prober: http
        timeout: 5s
        http:
          method: POST
          headers:
            Host: "login.example.com"
          basic_auth:                 # Username and password to be adde
d during probing
            username: "username"
            password: "mysecret"
      http_custom_ca_example:
        prober: http
        http:
          method: GET
          tls_config:                 # Specify the root certificate to
use during probing
            ca_file: "/certs/my_cert.crt"
      http_gzip:
        prober: http
        http:
```

How To

Backup and Restore of Prometl	VictoriaMetrics Backup and Re(	Collect Netv
Feature Overview	Function Overview	Function Overv
Use Cases	Use Cases	Use Case
Prerequisites	Prerequisites	Prerequisites
Procedures to Operate	Procedures	Procedures to (
Operation Results	Operation Result	Operation Resu
Learn More	Learn More	Learn More
Next Procedures	Follow-up Actions	Subsequent Act

Backup and Restore of Prometheus Monitoring Data

TOC

Feature Overview

Prometheus monitoring data is stored in TSDB (Time Series Database) format, supporting backup and restore functionalities. The monitoring data is stored in a designated path within

the Prometheus container (specified by the configuration `storage.tsdb.path`, which defaults to `/prometheus`).

```
template:
  spec:
    containers:
      - args:
          - '--storage.tsdb.path=/prometheus' # Directory for storing mon
itoring data in the Prometheus container
```

Use Cases

- Retaining historical monitoring data during system migration
- Preventing data loss due to unexpected incidents
- Migrating monitoring data to a new Prometheus instance

Prerequisites

- The ACP Monitoring with Prometheus plugin has been installed (the name of the compute component is `prometheus-kube-prometheus-0`, and the type is `StatefulSet`)
- Administrator privileges for the cluster
- Ensure there is sufficient storage space at the target storage location

Procedures to Operate

Backup Data

Before starting the backup, please note: When Prometheus stores monitoring data, it first places the collected data into a cache and then periodically writes it to the storage directory. The following backup methods use the storage directory as the data source, so they do not include the data in the cache at the time of backup.

Method 1: Backup Storage Directory (Recommended)

1. Use the kubectl cp command to back up:

```
kubectl cp -n cpaas-system prometheus-kube-prometheus-0-0:/prometheus -
c prometheus <target storage path>
```

2. Backup from the storage backend (based on the type of storage selected during installation):

- **LocalVolume**: Copy from the `/cpaas/monitoring/prometheus` directory
- **PV**: Copy from the PV mount directory (it is recommended to set the PV's **persistentVolumeReclaimPolicy** to `Retain`)
- **StorageClass**: Copy from the PV mount directory

Method 2: Snapshot Backup

1. Enable Admin API:

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

Add the configuration:

```
spec:
  enableAdminAPI: true
```

Note: After updating and saving the configuration, the Prometheus Pod (Pod name: prometheus-kube-prometheus-0-0) will restart. Wait until all Pods are in Running status before proceeding with subsequent operations.

2. Create a snapshot:

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

Restore Data

1. Copy the backup data to the Prometheus container:

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

2. Move data into the specified directory:

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

Shortcut: When the storage type is **LocalVolume** during plugin installation, simply copy the backup data directly to the `/cpaas/monitoring/prometheus/prometheus-db/` directory of the node where the plugin is installed.

Operation Results

- After backup is complete, the complete TSDB format monitoring data can be seen at the target storage path
- After restoration is complete, Prometheus will automatically load the historical monitoring data

Learn More

TSDB Data Format Description

Example of TSDB format data structure:

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

Data Backup Considerations

- Backup data does not include the cached data at the time of backup
- It is recommended to perform data backups regularly
- When using PV storage, it is advisable to set the **persistentVolumeReclaimPolicy** to `Retain`

Next Procedures

- Verify whether the monitoring data has been correctly restored
- Regularly schedule data backup plans
- If using the snapshot backup method, you may opt to disable the Admin API

VictoriaMetrics Backup and Recovery of Monitoring Data

TOC

Function Overview

The backup and recovery feature for VictoriaMetrics monitoring data allows you to perform regular backups of monitoring data and recover data when necessary, ensuring the safety and availability of monitoring data.

Use Cases

- Regularly backing up monitoring data to prevent data loss
- Data migration during system migration
- Disaster recovery
- Reconstructing test environment data

Prerequisites

- The ACP Monitoring with VictoriaMetrics plugin has been installed in the cluster
- Ensure there is sufficient storage space for backups
- Have access to the VictoriaMetrics storage path

Procedures

1. Confirm Storage Path

The monitoring data of VictoriaMetrics is stored in the specified path of the container, which is indicated by the `-storageDataPath` parameter, defaulting to `/vm-data`.

Configuration example:

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

Note: The name of the computing component in the ACP Monitoring with VictoriaMetrics plugin is `vmstorage-cluster`, and its type is `StatefulSet`.

2. Execute Data Backup

Use vmbackup tool to perform data backup; please refer to the vmbackup official documentation↗ for detailed operations.

3. Execute Data Recovery

Use vmrestore tool to restore backup data; please refer to the vmrestore official documentation↗ for detailed operations.

Operation Result

After completing the backup, you will receive a complete backup file of the monitoring data. After executing the recovery operation, your monitoring data will be restored to the state it was in at the time of backup.

Learn More

- VictoriaMetrics official documentation↗
- Best Practices for Data Backup↗
- Troubleshooting Data Recovery↗

Follow-up Actions

- Verify the integrity of the backup data
- Set up a regular backup schedule
- Periodically test the recovery process
- Monitor the execution status of backup tasks

Collect Network Data from Custom-Named Network Interfaces

TOC

Function Overview

The platform supports collecting network data from custom-named network interfaces by modifying the configuration of the monitoring component, enabling you to view the network traffic information for these interfaces on the monitoring page.

Use Case

This is applicable when your nodes use custom-named network interfaces (not following the `eth.|en.|wl.*|ww.*` naming conventions) and require monitoring of these interfaces'

network traffic data in the platform.

Prerequisites

- A workload cluster has been created
- You have platform administrator permissions
- The naming conventions for the custom network interfaces are known

Procedures to Operate

1. Click the CLI tool icon in the top navigation bar of the platform.
2. Click **global**.
3. In the `global` cluster, find the moduleinfo resource name corresponding to your workload cluster:

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

   Example output:

   ```
   global-6448ef7f7e5e3924c1629fad826372e7     global     prometheus
   prometheus                          Running   v3.15.0-zz231204040711-9d
   1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz2312040407
   11-9d1fc12474c2
   ovn-0954f21f0359720e8c115804376b3e7e        ovn        prometheus
   prometheus                          Running   v3.15.0-zz231204040711-9d
   1fc12474c2   v3.15.0-zz231204040711-9d1fc12474c2   v3.15.0-zz2312040407
   11-9d1fc12474c2
   ```

4. Edit the moduleinfo resource of the workload cluster:

   ```
   kubectl edit moduleinfo <moduleinfo resource name of the workload clust
   er>
   ```

5. Add or modify the valuesOverride field:

```
spec:
  valuesOverride:# If this field does not exist, you need to add the va
luesOverride field under spec along with the parameters below
    ait/chart-cpaas-monitor:
      global:
        indicator:
          networkDevice: eth.*|em.*|en.*|wl.*|ww.*|[A-Z].*i|custom_inte
rface # Replace custom_interface with the custom regular expression to
ensure correct matching of the network interface name
```

6. After waiting for 10 minutes, check the network-related charts on the node's monitoring page to ensure the changes have taken effect.

Operation Results

Once the configuration is effective, you can view the following data of the custom-named network interfaces on the platform's monitoring page:

- Network traffic data
- Network throughput
- Network packet statistics

Learn More

- For more information on network monitoring, please refer to the Monitoring Architecture Documentation

Subsequent Actions

- Monitor the performance metrics of the custom network interfaces
- Set alert rules based on the monitoring data

Distributed Tracing

Introduction

Install

Architecture

Architecture

Core Components

Data Flow

Concepts

Concepts

Telemetry

OpenTelemetry

Span

Trace

Instrumentation

OpenTelemetry Collector

Jaeger

Guides

Query Tracing

Feature Overview

Main Features

Feature Advantages

Tracing Query

Query Result Analysis

Query Trace Logs

Feature Overview

Core Features

Prerequisites

Log Query Operations

How To

Non-Intrusive Integration of Tra

- Feature Overview
- Use Cases
- Prerequisites
- Steps to Operate
- Operation Results

Business Log Associated with the TraceID

- Background
- Adding TraceID to Java Application Logs
- Adding TraceID to Python Application Logs
- Verification Method

Troubleshooting

Unable to Query the Required T

- Problem Description
- Root Cause Analysis
- Solution for Root Cause 1
- Solution for Root Cause 2

Incomplete Tracing Data

- Problem Description
- Root Cause Analysis
- Solution for Root Cause 1
- Solution for Root Cause 2

Introduction

The Distributed Tracing module is a core component of the ACP platform's observability suite that provides end-to-end request tracking and analysis capabilities for distributed microservices architectures.

This module delivers four essential tracing capabilities:

- **Trace collection** for automated gathering of distributed request data through OpenTelemetry automatic injection or SDK integration
- **Trace storage** for scalable persistence of tracing data using Elasticsearch as the backend storage
- **Trace visualization** for multi-dimensional analysis through customized UI dashboards and service dependency mapping
- **Trace querying** for precise search and filtering using TraceID, service names, tags, and other rich search conditions

By integrating these capabilities with OpenTelemetry standards and open-source components, it enables organizations to quickly locate service anomalies, analyze performance bottlenecks, trace complete request lifecycles, and optimize distributed system performance across their microservices architecture.

TOC

Usage Limitations

When using tracing, the following constraints should be noted:

- **Balancing Sampling Strategies and Performance**
  - In high-load scenarios, the collection of tracing data may exert certain pressure on Elasticsearch's performance and storage; it is recommended to configure the sampling rate reasonably based on business conditions.

Install

> **WARNING**
>
> **This deployment document is only applicable to scenarios involving the integration of the container platform with the tracing system.**
>
> The **Tracing** component and the **Service Mesh** component are mutually exclusive. If you have already deployed the Service Mesh component, please uninstall it first.

This guide provides cluster administrators with the process of installing the tracing system on the Alauda Container Platform cluster.

Prerequisites:

- You have access to the Alauda Container Platform cluster with an account that has `platform-admin-system` permissions.
- You have the `kubectl` CLI installed.
- The `Elasticsearch` component is set up to store tracing data, including the access URL and `Basic Auth` information.

TOC

Installing the Jaeger Operator

Install the Jaeger Operator using the Web Console

You can install the Jaeger Operator from the Alauda Container Platform **Marketplace → OperatorHub** section where the available Operators are listed.

Steps

- In the **Administrator** view of the Web Console, select the **cluster** where you want to deploy the Jaeger Operator, then navigate to **Marketplace → OperatorHub**.
- Use the search box to search for `Alauda build of Jaeger` in the catalog. Click on the **Alauda build of Jaeger** title.
- Read the introductory information about the Operator on the **Alauda build of Jaeger** page. Click **Install**.
- On the **Install** page:
  - Select **Manual** for the **Upgrade Strategy**. For the `Manual` approval strategy, OLM will create update requests. As a cluster administrator, you must manually approve the OLM update requests to upgrade the Operator to the new version.
  - Select the **stable (Default)** channel.
  - Choose **Recommended** for **Installation Location**. Install the Operator in the recommended `jaeger-operator` namespace, so the Operator can monitor and be available in all namespaces within the cluster.

- Click **Install**.
- Verify that the **Status** displays as **Succeeded** to confirm the Jaeger Operator was installed correctly.
- Check that all components of the Jaeger Operator were successfully installed. Log into the cluster via terminal, and run the following command:

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

Example output

```
NAME                      DISPLAY          VERSION   REPLACES   PHASE
jaeger-operator.vx.x.0    Jaeger Operator  x.x.0                Succeed
ed
```

If the `PHASE` field shows `Succeeded`, it means the Operator and its components were installed successfully.

Deploying a Jaeger Instance

A Jaeger instance and its related resources can be installed with the `install-jaeger.sh` script, which takes three parameters:

- `--es-url`: The access URL for Elasticsearch.
- `--es-user-base64`: The `Basic Auth` username for Elasticsearch, encoded in base64.
- `--es-pass-base64`: The `Basic Auth` password for Elasticsearch, encoded in base64.

Copy the installation script from **DETAILS**, log into the cluster where you want to install it, save it as `install-jaeger.sh`, and execute it after granting execute permissions:

▶ **DETAILS**

Script execution example:

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

Script output example:

```
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
```

Installing the OpenTelemetry Operator

Install the OpenTelemetry Operator using the Web Console

You can install the OpenTelemetry Operator from the Alauda Container Platform **Marketplace** → **OperatorHub** section where the available Operators are listed.

Steps

- In the **Administrator** view of the Web Console, select the **cluster** where you want to deploy the OpenTelemetry Operator, then navigate to **Marketplace → OperatorHub**.
- Use the search box to search for `Alauda build of OpenTelemetry` in the catalog. Click on the **Alauda build of OpenTelemetry** title.
- Read the introductory information about the Operator on the **Alauda build of OpenTelemetry** page. Click **Install**.
- On the **Install** page:
  - Select **Manual** for the **Upgrade Strategy**. For the `Manual` approval strategy, OLM will create update requests. As a cluster administrator, you must manually approve the OLM update requests to upgrade the Operator to the new version.
  - Select the **alpha (Default)** channel.
  - Choose **Recommended** for **Installation Location**. Install the Operator in the recommended `opentelemetry-operator` namespace, so the Operator can monitor and be available in all namespaces within the cluster.
- Click **Install**.
- Verify that the **Status** displays as **Succeeded** to confirm the OpenTelemetry Operator was installed correctly.
- Check that all components of the OpenTelemetry Operator were successfully installed. Log into the cluster via terminal, and run the following command:

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

  Example output

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

  If the `PHASE` field shows `Succeeded`, it means the Operator and its components were installed successfully.

Deploying OpenTelemetry Instances

OpenTelemetry instances and their related resources can be installed using the `install-otel.sh` script.

Copy the installation script from **DETAILS**, log into the cluster where you want to install it, save it as `install-otel.sh`, and execute it after granting execute permissions:

▶ **DETAILS**

Script execution example:

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

Script output example:

```
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
```

Enable Feature Switch

The tracing system is currently in the **Alpha** phase and requires you to manually enable the `acp-tracing-ui` feature switch in the **Feature Switch** view.

Then, navigate to the **Container Platform** view, and go to **Observability** → **Tracing**, to view the tracing feature.

Uninstall Tracing

Deleting OpenTelemetry Instance

Log into the installed cluster and execute the following commands to delete the OpenTelemetry instance and its related resources.

```
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-adm
in
kubectl -n cpaas-system delete serviceaccount otel-collector
```

Uninstalling OpenTelemetry Operator

You can uninstall the OpenTelemetry Operator using the **Administrator** view in the Web Console.

Steps

- From **Marketplace** → **OperatorHub** → use the **search box** to search for `Alauda build of OpenTelemetry`.
- Click on the **Alauda build of OpenTelemetry** title to enter its details.
- On the **Alauda build of OpenTelemetry** details page, click the **Uninstall** button in the upper right corner.
- In the **Uninstall "opentelemetry-operator"?** window, click **Uninstall**.

Deleting Jaeger Instance

Log into the installed cluster and execute the following commands to delete the Jaeger instance and its related resources.

```
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
```

Uninstalling Jaeger Operator

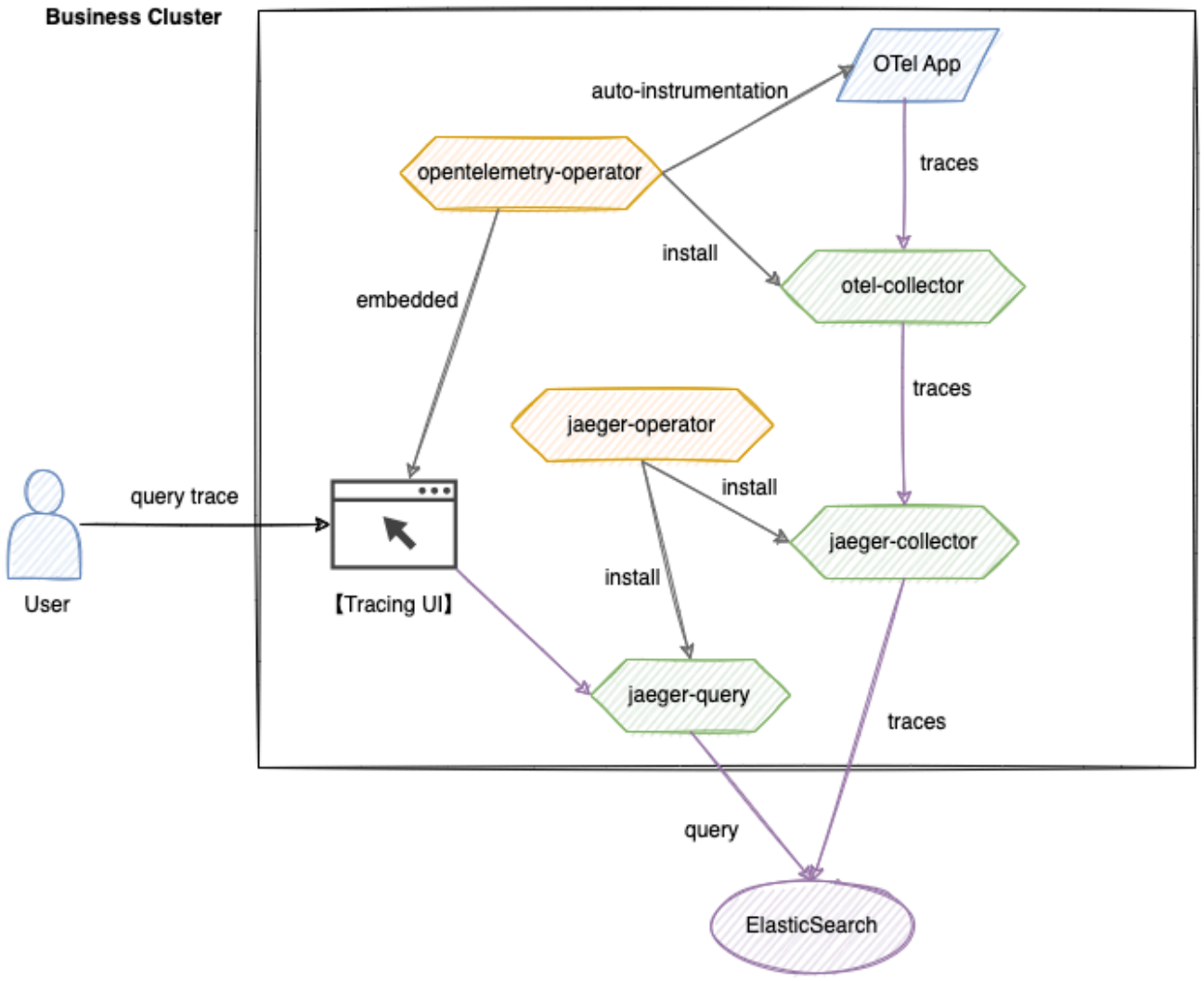
You can uninstall the Jaeger Operator using the **Administrator** view in the Web Console.

Steps

- From **Marketplace** → **OperatorHub** → use the **search box** to search for `Alauda build of Jaeger`.
- Click on the **Alauda build of Jaeger** title to enter its details.
- On the **Alauda build of Jaeger** details page, click the **Uninstall** button in the upper right corner.
- In the **Uninstall "jaeger-operator"?** window, click **Uninstall**.

Architecture

This architecture is built on the OpenTelemetry and Jaeger technology stack, achieving the full lifecycle management of distributed tracing. The system comprises five core modules: data collection, transmission, storage, querying, and visualization.



TOC

Core Components

1. **OpenTelemetry System**

   - **opentelemetry-operator**
     A cluster-level Operator responsible for deploying and managing the otel-collector component, providing OTel automatic injection capability.
   - **otel-collector**
     Receives tracing data from applications, filters and batches it, and then forwards it to the jaeger-collector.
   - **Tracing UI**
     A self-developed visualization interface that integrates with the jaeger-query API, supporting multi-dimensional query conditions.

2. **Jaeger System**

   - **jaeger-operator**
     Deploys and manages the jaeger-collector and jaeger-query components.
   - **jaeger-collector**
     Receives tracing data forwarded and processed by the otel-collector, performs format conversion, and writes it to Elasticsearch.
   - **jaeger-query**
     Provides a tracing query API, supporting multi-condition retrieval including TraceID and labels.

3. **Storage Layer**

   - **Elasticsearch**
     A distributed storage engine that supports efficient writing and retrieval of massive Span data.

Data Flow

- **Writing Process**

  `Application -> otel-collector -> jaeger-collector -> Elasticsearch`

  The application generates Span data via SDK or automatic injection, which is standardized by the otel-collector and subsequently persisted to Elasticsearch by the jaeger-collector.

- **Query Process**

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

  The user submits query conditions through the UI, and jaeger-query retrieves data from Elasticsearch; the UI visualizes the results based on the return data.

Concepts

TOC

Telemetry

Telemetry refers to the data emitted by systems and their behaviors, including traces, metrics, and logs.

OpenTelemetry

OpenTelemetry is an observability framework and toolkit designed to create and manage telemetry data such as traces, metrics, and logs. Importantly, OpenTelemetry is vendor-agnostic, meaning it can work with various observability backends, including open-source tools like Jaeger and Prometheus as well as commercial products.

Span

Span is the fundamental building block of distributed tracing, representing a specific operation or work unit. Each span records specific actions within a request, helping us understand the details of what occurred during the operation's execution.

A span contains a name, time-related data, structured log messages, and other metadata (attributes) that collectively illustrate the complete picture of the operation.

Trace

Trace records the path of a request (whether from an application or end-user) as it propagates through a multi-service architecture (such as microservices and serverless applications).

A trace consists of one or more spans. The first span is known as the root span, which represents the entire lifecycle of a request from start to finish. Child spans beneath the root span provide more detailed contextual information about the request process (or the various steps that constitute the request).

Without traces, identifying the root cause of performance issues in distributed systems would be quite challenging. Traces make it easier to debug and understand distributed systems by breaking down the flow of requests through them.

Instrumentation

To enable observability, a system needs to undergo **Instrumentation**: that is, the component code of the system must emit traces ↗, metrics ↗, and logs ↗.

With OpenTelemetry, you can instrument your code in two primary ways:

1. Code-based solutions ↗: Using the official APIs and SDKs for most languages ↗
2. Zero-instrumentation solutions ↗

Code-based solutions provide deeper insights and richer telemetry data from within your application. You can generate telemetry data in your application using the OpenTelemetry API, which is an important complement to the telemetry data generated by zero-instrumentation solutions.

Zero-instrumentation solutions are great for quickly getting started or when you cannot modify the application from which you need telemetry data. They can provide rich telemetry data via the libraries or runtime environment you are using. Another way to understand them is that they deliver information about events occurring at the boundaries (Edges) of the application.

These two solutions can be used simultaneously.

OpenTelemetry Collector

OpenTelemetry Collector is a vendor-agnostic agent that can receive, process, and export telemetry data. It supports receiving telemetry data in various formats (such as OTLP, Jaeger, Prometheus, and many commercial/proprietary tools) and sending that data to one or more backends. It also supports processing and filtering telemetry data before exporting.

For more information, see Collector↗.

Jaeger

Jaeger is an open-source **distributed tracing system**. It is designed to monitor and diagnose complex distributed systems based on microservices architecture, helping developers visualize request traces, analyze performance bottlenecks, and troubleshoot anomalies. Jaeger is compatible with the **OpenTracing** standard (now part of OpenTelemetry), supports multiple programming languages and storage backends, and is a key observability tool in the cloud-native space.

Guides

Query Tracing

- Feature Overview
- Main Features
- Feature Advantages
- Tracing Query
- Query Result Analysis

Query Trace Logs

- Feature Overview
- Core Features
- Prerequisites
- Log Query Operations

Query Tracing

TOC

Feature Overview

The distributed tracing query feature provides full-link tracing capabilities for microservices architecture by collecting metadata information of inter-service calls, helping users quickly locate cross-service call issues. This feature mainly addresses the following problems:

- **Request Link Tracing**: Restoring the complete request path in complex distributed systems.
- **Performance Bottleneck Analysis**: Identifying abnormal call nodes in terms of time consumption within the link.

- **Fault Root Cause Location**: Quickly locating the point of issue occurrence through error marking.

Applicable scenarios include:

- Rapidly locating abnormal services during production environment fault troubleshooting.
- Identifying high-latency call links during performance tuning.
- Validating inter-service call relationships after a new version release.

Core values:

- Enhancing observability of distributed systems.
- Reducing Mean Time to Recovery (MTTR).
- Optimizing inter-service call performance.

Main Features

- Multi-dimensional Querying: Supports 6 combinations of query conditions such as TraceID, service name, labels, etc.
- Visual Analysis: Intuitively displays call hierarchy and time distribution through time-series waterfall charts.
- Precise Location: Supports error Span filtering and secondary searches with labels.

Feature Advantages

- **Quick Problem Identification**: Narrowing down the inspection range through multi-dimensional query conditions accelerates problem location.
- **Visual Presentation**: Using time-series waterfall charts to intuitively display call relationships reduces complexity and enhances fault analysis efficiency.
- **Flexibility and Variety**: Supports both simple queries and complex combinations, adapting to various operation and development scenarios.

Tracing Query

1 Step 1: Combine Query Conditions

Tip: Query conditions can be combined for use. You can refine your query by adding multiple query conditions.

Query Condition	Description
TraceID	The unique identifier for the complete link, which can be used to query the specified tracing.
Service	The service that initiates/receives the call request (needs to be selected or input).
Label	You can filter the query results by entering labels (Tag), supported Tags include those in the Span details.
Span Duration Greater Than	Spans that have a duration greater than or equal to *input value* (ms).
Only Search Error Spans	Error Spans refer to Spans whose Tag value of **error** is `true`.
Span Type	**Root Span**: Searches for root Spans initiated by the configured **service**. This search mode is used when the configured service is the initiator of the entire call request. **Service Entry Span**: Searches for the first Span generated when the configured **service** is called as a server.
Maximum Query Count	The maximum number of Spans that can be queried, with a default of 200. **Tip**: For performance reasons, the platform can display a maximum of 1000 Spans at a time. If the number of Spans that meet the query conditions exceeds the **maximum query count**, you can refine the query conditions or narrow the time range for phased queries.

2 Step 2: Execute Query

> - Once you select the query conditions and enter the respective values, click the **Add to Query Conditions** button, and the current conditions will be displayed in the **Query Conditions** result area, triggering the query.
> - You can also expand **Common Query Conditions** to quickly add recently used search conditions.

Query Result Analysis

After entering the query conditions and searching, a query results area will be generated on the page.

Span List

The left side of the query results area displays a list of Spans that meet the conditions along with basic information about the Spans, including: service name, called interface or request processing method, duration, and start time.

Time-Series Waterfall Chart

The time-series waterfall chart on the right side of the query results area clearly displays the call relationships between Spans in a single tracing. The main features of using time-series waterfall charts in tracing analysis are as follows:

1. Top-to-bottom expansion: In the time-series waterfall chart, various call events (Spans) typically expand downwards from the top of the chart, with each horizontal bar representing a service call or process. The position generally reflects the logical calling order of operations.
2. Time axis alignment: The horizontal axis of the time-series waterfall chart represents time. The length of each bar indicates the duration of that call, allowing for an intuitive comparison of the time relationships between different calls.
3. Indentation description: Indentation indicates the hierarchical relationship of calls, with deeper indentation denoting greater call depth within that link.
4. Interactivity and detailed data display: Clicking on the bars in the time-series waterfall chart can display more detailed information about that call.

Span Details

By clicking on the row of the Span in the time-series waterfall chart, you can expand and view detailed information about the Span, including:

- Service: The service within the Span.
- Span Duration (ms): The duration of the Span.
- URL: The URL accessed by the service, corresponding to **http.url** in Span Tags.
- Tag: The label information of the Span composed of key-value pairs, which can be used for advanced search tag query conditions. By clicking the button next to the tag, you can add the current Tag condition to the query conditions for more precise query results.
- JSON: The original JSON structure of the Span, allowing for further inspection of its internal information.

Query Trace Logs

TOC

Feature Overview

Trace Logs enable users to query and analyze logs associated with a specific distributed trace using its unique TraceID. This feature helps developers and operators quickly locate issues, understand request flows, and correlate business logs with trace contexts.

Key Benefits:

- **Root Cause Analysis**: Identify errors and latency issues across microservices in distributed systems.
- **Context Correlation**: Link business logs to trace spans for end-to-end visibility.
- **Efficient Filtering**: Filter logs by Pods or TraceID to focus on relevant data.

Applicable Scenarios:

- Debugging cross-service transaction failures.
- Analyzing performance bottlenecks in complex workflows.
- Auditing request processing flows for compliance.

Core Features

- **TraceID-Based Query**: Retrieve all logs associated with a specific trace using its TraceID.
- **Pod-Centric Filtering**: View logs from specific Pods involved in the trace.
- **Log Export**: Export filtered log data in customizable formats.
- **Contextual Log Viewing**: Examine log records before/after a target entry for deeper analysis.

Prerequisites

> **TIP**
>
> Before querying trace logs by TraceID, you must first instrument your services to include TraceID in business logs. Follow the Business Log Correlation with TraceID Guide for configuration details.

Default Behavior:

- Displays logs from the entire trace duration.
- For traces shorter than 1 minute, queries logs within 1 minute after the trace start time.

Log Query Operations

1 Access Trace Logs

1. After querying traces, click on a specific trace to view its details.
2. Click the **View Log** tab in the trace visualization panel.

2 Filter Logs

By Pod Name

In the **Pod Name** selector, choose target Pod from the participating services list.

By Time Range

In the **Time Range** selector, choose target time range.

By Query Conditions

Enter keywords in the **Query Conditions** text box to filter logs based on specific content.

Contain Trace ID

Select the **Contain Trace ID** checkbox.

3 Advanced Operations

Export Logs

1. Click **Export**.
2. Select fields to include using column checkboxes.
3. Choose export format (JSON/CSV).

Customize Display Fields

Click **Set**. Toggle visibility of log fields in the display panel.

View Log Context

1. Click **Insight** next to any log entry.
2. Explore 5 preceding and succeeding logs around the target timestamp.
3. Scroll up/down with mouse to load more logs.

How To

Non-Intrusive Integration of Tracing in Java Applications

INFO

The automatically injected OpenTelemetry Java Agent ↗ supports `Java 8+` versions.

TOC

Feature Overview

Tracing is a core capability of observability in distributed systems, which can fully record the call paths and performance data of requests within the system. This article describes how to achieve non-intrusive integration of tracing in Java applications using the automatic injection of the OpenTelemetry Java Agent.

Use Cases

Java applications can be integrated for the following scenarios:

- Quickly adding tracing capabilities to Java applications
- Avoiding modifications to the application source code
- Deploying services with Kubernetes
- Visualizing service inter-call relationships and analyzing performance bottlenecks

Prerequisites

Before using this feature, ensure that:

- The target service is deployed on the Alauda Container Platform
- The service is using JDK version Java 8 or higher
- You have editing permissions for the Deployment in the target namespace
- The platform has completed tracing deployment

Steps to Operate

For a Java application that needs to be integrated into the Alauda Container Platform tracing, the following adaptations are required:

- Configure automatic injection annotations for the Java Deployment.
- Set the `SERVICE_NAME` environment variable.
- Set the `SERVICE_NAMESPACE` environment variable.

Example of Deployment adaptation:

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

(1) Choose `cpaas-system/acp-common-java` Instrumentation as the configuration for injecting the Java Agent.

(2) Configure the `SERVICE_NAME` environment variable, which can be associated through labels or fixed values.

(3) Configure the `SERVICE_NAMESPACE` environment variable, with its value as `metadata.namespace`.

Operation Results

After adapting the Java application:

- If the newly started Java application pod contains the `opentelemetry-auto-instrumentation-java` init container, it indicates that the injection was successful.
- Send test requests to the Java application.
- In the **Container Platform** view, select the **project**, **cluster**, and **namespace** where the Java application resides.
- Navigate to the **Observability** -> **Tracing** page to view the tracing data and timeline waterfall diagram of the Java application.

TIP

This article will guide developers on how to integrate methods for **getting TraceID** and **adding TraceID to application logs** in the application code, suitable for backend developers with some development experience.

Business Log Associated with the TraceID

TOC

Background

- To correctly associate multiple automatically sent spans (different modules/nodes/services called during a single request) into a single trace, the service's HTTP request headers will include TraceID and other information used for associating the trace.
- A trace represents the call process of a single request, and TraceID is the unique ID identifying this request. With the TraceID in the logs, the traceing can be associated with the application logs.

Based on the above background, this article will explain how to obtain the TraceID from the HTTP request headers and add it to application logs, allowing you to accurately query log data on the platform using TraceID.

Adding TraceID to Java Application Logs

TIP

- The following examples are based on the **Spring Boot** framework and use **Log4j** and **Logback** for illustration.
- Your application must meet the following prerequisites:
  - The type and version of the logging library must meet the following requirements:

Logging Library	Version Requirement
Log4j 1	1.2+
Log4j 2	2.7+
Logback	1.0+

  - The application has been injected with a Java Agent.

Method 1: Configure `logging.pattern.level`

Modify the `logging.pattern.level` parameter in your application configuration as follows:

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

Method 2: Configure `CONSOLE_LOG_PATTERN`

1. Modify the logback configuration file as follows.

> **TIP**
>
> The console output is used as an example here, where `%X{trace_id}` indicates the value of the key `trace_id` retrieved from MDC.

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

2. In the class where logs need to be output, add the `@Slf4j` annotation and use the log object to output logs, as shown below:

```
@RestController
@Slf4j
public class ProviderController {

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

Adding TraceID to Python Application Logs

1. In the application code, add the following code to retrieve the TraceID from the request headers. The example code is as follows and can be adjusted as needed:

> **TIP**
>
> The **getForwardHeaders** function retrieves trace information from the request headers, where the value of `x-b3-traceid` is the TraceID.

```
    def getForwardHeaders(request):
        headers = {}
        incoming_headers = [
            'x-request-id',  # All applications should pass x-request-id
for access logs and consistent tracing/log sampling decisions
            'x-b3-traceid',  # B3 trace header, compatible with Zipkin,
OpenCensusAgent, and Stackdriver configurations
            '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. In the application code, add the following code to include the retrieved TraceID in the logs. The example code is as follows and can be adjusted as needed:

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

Verification Method

1. Click on **Tracing** in the left navigation bar.
2. In the query criteria, select TraceID, enter the TraceID to query, and click **Add to query**.
3. In the displayed trace data below, click **View Log** next to the TraceID.
4. On the **Log Query** page, check **Contain Trace ID**; the system will only display log data containing the TraceID.

Troubleshooting

Unable to Query the Required Tracing

TOC

Problem Description

When querying the tracing in a service mesh, you may encounter situations where the target tracing cannot be retrieved.

Root Cause Analysis

1. Tracing Sampling Rate Configured Too Low

When the sampling rate parameter for the tracing is set too low, the system will only collect tracing data proportionally. During times of insufficient request volume or low-peak periods, this may lead to the sampled data being below the visibility threshold.

2. Elasticsearch Real-Time Limitations

The default configuration for Elasticsearch index is `"refresh_interval": "10s"`, which results in a delay of 10 seconds before data is refreshed from the memory buffer to a searchable state. When querying recently generated tracing, the results may be missing because the data has not yet been persisted.

This index configuration can effectively reduce the data merge pressure on Elasticsearch, improving indexing speed and the speed of the first query, but it also reduces the real-time nature of the data to some extent.

Solution for Root Cause 1

- Appropriately increase the sampling rate according to requirements.
- Use richer sampling methods, such as tail sampling.

Solution for Root Cause 2

Adjust the refresh interval through the `--es.asm.index-refresh-interval` startup parameter of `jaeger-collector`, with a default value of `10s`.

If the value of this parameter is `"null"`, there will be no configuration for the index's `refresh_interval`.

Note: Setting it to `"null"` will affect the performance and query speed of Elasticsearch.

Incomplete Tracing Data

TOC

Problem Description

The tracing query results exhibit the following issues of incomplete data:

- Recent queries (within the last 30 minutes) are missing some spans.
- Tracing older than 1 hour are experiencing disconnections.

Root Cause Analysis

1. Data Persistence Delay

The writing process in Elasticsearch requires a sequence of steps involving memory buffer → translog → segment files, which can result in visibility delays for the most recently written

data.

2. Time Range Limitation

By default, when `jaeger-query` queries spans corresponding to tracing, the time range extends one hour before and after the start time of the span.

For instance, if a span starts at `08:12:30` and ends at `08:12:32`, the time range for querying that tracing would be from `07:12:30` to `09:12:32`.

Therefore, if the tracing spans over 1 hour, querying through this span may not yield a complete tracing.

Solution for Root Cause 1

Wait a moment and refresh the page to try the query again.

Solution for Root Cause 2

If the tracing span in your environment is lengthy, you can adjust the query time range for a single tracing using the `--es.asm.span-trace-query-time-adjustment-hours` startup parameter in `jaeger-query`.

The default value of this parameter is `1` hour, and you can increase this value as needed.

Logs

About Logging Service

About Logging Service

The Logging module is a core component of the ACP platform's observability suite that provides comprehensive log management capabilities for efficient and reliable log processing.

This module delivers four essential logging capabilities:

- **Log collection** for automated gathering of logs from applications, containers, and infrastructure components
- **Log storage** for scalable and durable persistence using ElasticSearch and ClickHouse backends
- **Log querying** for fast and flexible search across large volumes of log data

By integrating these capabilities with powerful open-source components like Filebeat, ElasticSearch, and ClickHouse, it enables organizations to efficiently handle massive log volumes, accelerate troubleshooting, ensure compliance requirements, and gain valuable operational insights in real time.

> **Note**
>
> Because Logging Service releases on a different cadence from Alauda Container Platform, the Logging Service documentation is now available as a separate documentation set at Logging Service ↗.

Events

Introduction

The platform integrates with Kubernetes events, logging significant status changes and various operational state changes of Kubernetes resources. It also provides capabilities for storage, querying, and visualization. When abnormalities occur with resources such as clusters, nodes, or Pods, users can analyze events to determine specific causes.

Based on the root causes identified from the events, users can create alert policies for workloads. When the number of critical events reaches the alert threshold, alerts can be automatically triggered to notify relevant personnel for timely intervention, thereby reducing operational risks on the platform.

TOC

Usage Limitations

This feature relies on the logging service. Please ensure that the ACP Log Essentails , ACP Log Collector and ACP Log Storage plugins are installed within the platform beforehand.

Note

Because Logging Service releases on a different cadence from Alauda Container Platform, the Logging Service documentation is now available as a separate documentation set at Logging Service ↗.

Events

TOC

Operation Procedures

1. Click on **Operations Center** > **Events** in the left navigation bar.
   Tip: Switch the cluster to view events using the dropdown selection box in the top navigation bar.

Event Overview

The events page displays an overview of significant events that occurred in the last 30 minutes by default (you can choose or customize the time range), as well as records of resource events.

- **Significant Event Overview**: This card shows the reason for significant events and the number of resources that experienced such events within the selected time range.
  - **Note**: When the same resource experiences this type of event multiple times, the resource count will not accumulate.

  - For example: If the resource count for node restart events is 20, it indicates that within the selected time range, 20 resources encountered such events, and the same resource may have experienced it multiple times.

- **Resource Event Records**: Below the significant event overview area, all event records that meet the query conditions within the selected time range are displayed. You can filter for respective types of events by clicking on the significant event card, or you can expand the view and input query conditions to search. The query conditions are as follows:

  - **Resource Type**: The type of Kubernetes resource that experienced the event, e.g., Pod.
  - **Namespace**: The namespace of the Kubernetes resource where the event occurred.
  - **Event Reason**: The reason for the occurrence of the event.
  - **Event Level**: The significance of the event, such as Warning.
  - **Resource Name**: The name of the Kubernetes resource that experienced the event. Multiple names can be selected or entered.

TIP

- Click the view icon next to the resource name in the event record to view detailed information about the event in the pop-up **Event Details** dialog.
- The color of the icon to the left of the event reason indicates the event level. A green icon indicates that the level of this event is `Normal`, and this event can be ignored; an orange icon signifies that the level of this event is `Warning`, indicating that there is an anomaly with the resource and this event should be monitored to prevent incidents.

Inspection

Introduction

Introduction

Usage Limitations

Architecture

Architecture

Inspection

Component Health Status

Guides

Inspection

Execute Inspection

Inspection Configuration

Inspection Report Explanation

Component Health Status

Procedures to Operate

Introduction

The Inspection module is a core component of the ACP platform's observability suite that provides automated inspection and assessment capabilities for comprehensive resource monitoring and risk management.

This module delivers four essential inspection capabilities:

- **Resource inspection** for automated assessment of clusters, nodes, pods, certificates, and other platform resources to identify risks and usage patterns
- **Real-time monitoring** for live tracking of inspection task progress and immediate visibility into resource operational status
- **Visual reporting** for intuitive display of inspection results including resource risks, usage information, and operational insights
- **Report generation** for downloadable inspection reports in PDF or Excel formats with comprehensive analysis and recommendations

By integrating these capabilities with role-based access controls and automated assessment algorithms, it enables organizations to reduce manual inspection costs, proactively identify resource anomalies, mitigate business risks, and maintain optimal platform performance through systematic health assessments.

TOC

Usage Limitations

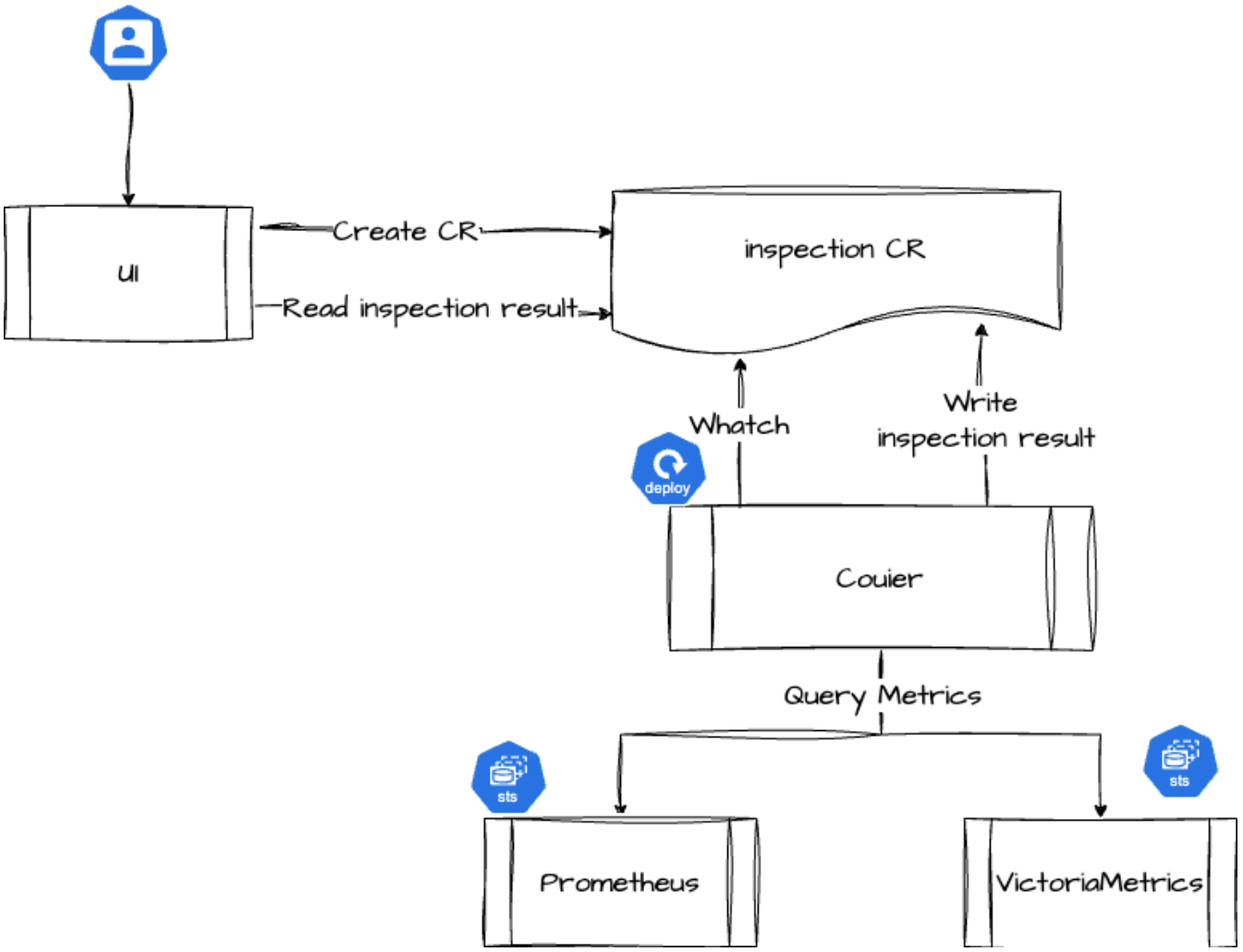
- Some inspection items on the platform depend on clusters having monitoring components installed. Please ensure that each cluster has either the ACP Monitoring with Prometheus plugin or the ACP Monitoring with VictoriaMetrics plugin installed in advance.
- The platform inspection supports sending inspection results via email. Please ensure that the email notification server configuration has been completed in advance.

With the container platform's inspection functionality, users can manage and maintain the container environment more efficiently, enhancing system stability and security.

Architecture

TOC

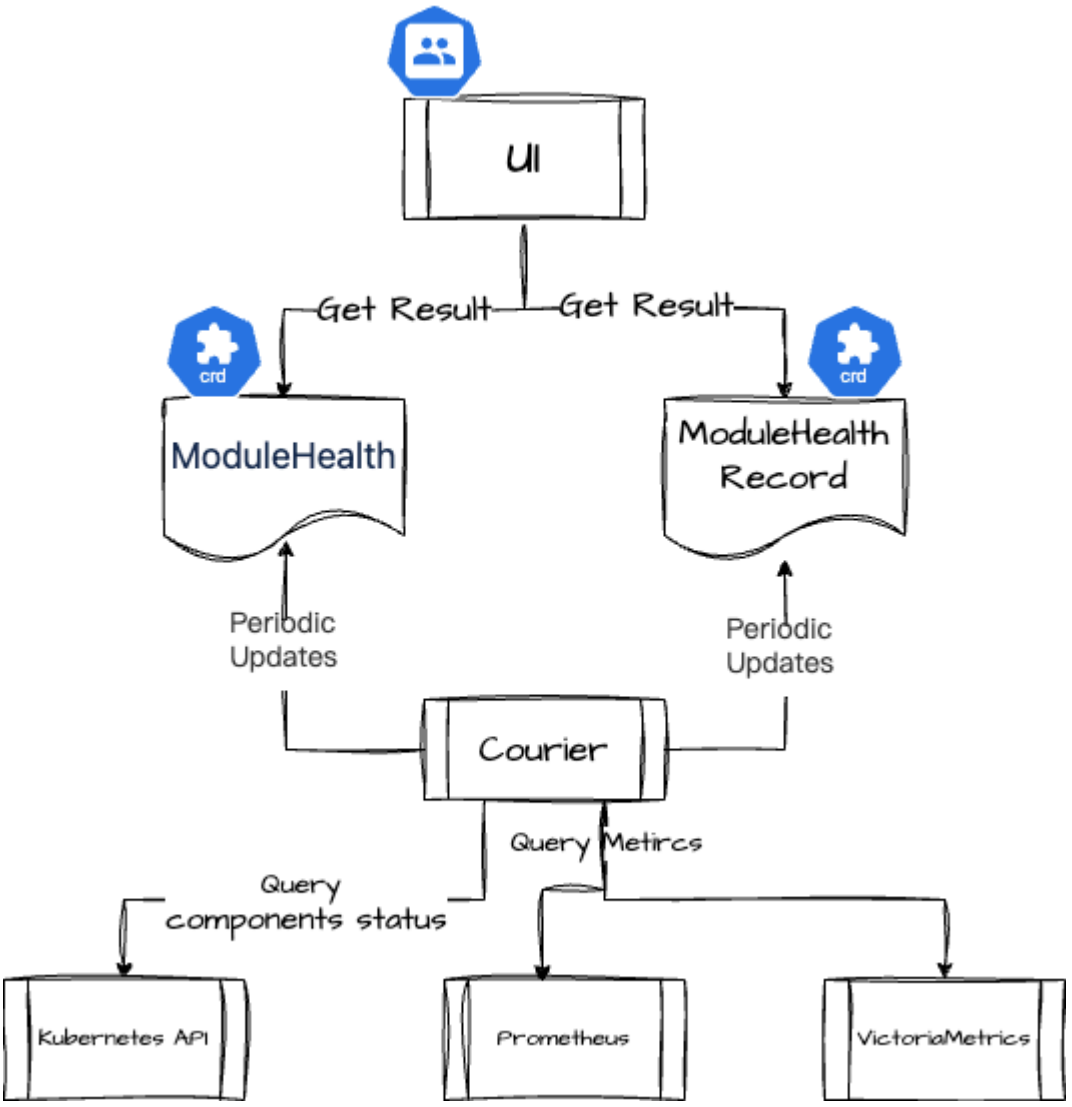
Inspection



The inspection module is jointly provided by the platform component Courier and the monitoring component, involving the following business processes:

- Create inspection task: The platform submits an inspection-type CR to the `global` cluster.
- Execute inspection task: The Courier component monitors the generation of inspection-type CRs and queries the monitoring components of each cluster for various metric data related to the inspection.
- Write inspection results: After the Courier component completes the evaluation of each inspection item, it will write the inspection results back into the corresponding inspection CR.
- View inspection results: Users can check the status and results of inspection tasks through the platform, where data will be obtained from the corresponding inspection CR.

Component Health Status



Component health status is jointly provided by the platform component Courier and the monitoring component, involving the following business processes:

- Predefined component monitoring list: The platform has predefined two types of CRDs in the `global` cluster to define the list of components to be monitored and the monitoring methods:
  - ModuleHealth: Defines the components that need to be monitored and the monitoring methods.
  - ModuleHealthRecord: Defines the monitoring results of the corresponding components in each cluster.
- Regularly monitor component status: Courier will watch ModuleHealth, check the specified functions, and then write the inspection results to the CR resources of ModuleHealth and ModuleHealthRecord.
- Component status determination: Courier will request data from Kubernetes and the monitoring components to determine the actual status of the components and any existing

issues.

  - Kubernetes: Checks whether the component is installed and whether the number of component replicas is normal.
  - Prometheus / VictoriaMetrics: Based on the metrics provided by each component, queries and determines whether the component can provide services normally.
- View component health status: Users can check the health status of each component through the platform, where data will be obtained from the corresponding CR resources of ModuleHealth and ModuleHealthRecord.

Guides

Inspection

Execute Inspection

Inspection Configuration

Inspection Report Explanation

Component Health Status

Procedures to Operate

Inspection

TOC

Execute Inspection

1. Click on **Operation Center** > **Inspection** > **Basic Inspection** in the left navigation bar.
   Tip: The inspection page displays the inspection data information from the most recent inspection. During the inspection process, you can view the resource data of completed inspections in real-time.
2. On the Basic Inspection page, the following actions are supported:
   - **Execute Inspection**: Click the **Inspection** button in the upper right corner of the page to perform an inspection on the platform.
   - **Download Inspection Report**: Click the **Download Report** button in the upper right corner of the page, select the report format (PDF and Excel) in the pop-up dialog, and click to download, which will download the corresponding format report to your local machine.

- The PDF format inspection report does not include resource risk details page data;
- The Excel format inspection report contains all data from the inspection;
- Supports simultaneous download of both formats of reports.

Inspection Configuration

Inspection Configuration	Description
Scheduled Inspection	Automated task execution timing rules, supporting input of Crontab expressions. **Tip**: Click the input box to expand the platform's preset **Trigger Rule Templates**, select the appropriate template, and quickly set the trigger rules with simple modifications.
Inspection Record Retention	The number of inspection records to retain.
Email Notification	Select email notification contacts. **Note**: Notification contacts must have email configured.
Inspection Report Name	The name that will be used by the platform's built-in inspection notification template to notify contacts.
Inspection Configuration Items	Modify the warning thresholds or disable inspection items according to the platform's default inspection items for certificates, cluster hosts, and pods.

Inspection Report Explanation

Most Recent Inspection

In the **Most Recent Inspection** information area, you can view relevant information from the most recent inspection:

- **Inspection Time**: The start and end time of the most recent inspection.

- **Total Number of Inspection Resources**: The total number of resources (clusters, nodes, pods, certificates) inspected in the most recent inspection.
- **Risks**: The number of resources at risk, including those classified as **Fault** and **Warning**.

Resource Risk Inspection

In the **Resource Risk Inspection** page, you can view an overview of risk information for `global` clusters, self-built clusters, accessed clusters, and all nodes, pods, and certificates under these clusters.

Click the **Risk Details** button on the card of the corresponding resource type (**Cluster**, **Node**, **pod**, **Certificate**) to enter the risk details page for that resource type. On the details page, you can view the most recent inspection information for the resource, as well as the list of resources with faults and warnings.

- Click on the resource name to jump to the resource details page.
- Click the expand button on the right side of the **Name** field in the list to expand the judgment conditions and reasons for faults and warnings.

For explanations of the risk status judgment criteria (Fault, Warning) for each resource, refer to the table below.

Note: There are multiple conditions used to judge the faults and warnings for each resource type; when the inspection data of the resource matches any one of the judgment conditions, it is considered a piece of risk data.

Resource Type	Inspection Scope	Fault Judgment Conditions	Warning Judgment Conditions
Cluster	- `global` cluster - Self-built cluster - Accessed cluster	- Cluster status is **Abnormal**; - apiserver connection is abnormal	- After the cluster scale (number of nodes/pods/mrtrics) increases, the monitoring component resource configuration has not been updated. - After the log data volume and log collection frequency increase, the log component resource configuration has not been updated.

Resource Type	Inspection Scope	Fault Judgment Conditions	Warning Judgment Conditions
			- Cluster CPU usage exceeds 60%; - Cluster memory usage exceeds 60%; - Any pod in the ETCD component of the cluster is in a non-Running state; - Any host in the cluster is in a non-Ready state; - The time difference between any two nodes in the cluster exceeds 40S; - The CPU request rate of the cluster (actual request value / total) exceeds 60%; - The memory request rate of the cluster (actual request value / total) exceeds 80%; - Monitoring components are not installed in the cluster; - Monitoring components of the cluster are abnormal; - Any pod in the **kube-controller-manager** component of the cluster is in a non-Running state; - Any pod in the **kube-scheduler** component of the cluster is in a non-Running state; - Any pod in the **kube-apiserver** component of the cluster is in a non-Running state.
Node	- All control nodes - All compute nodes	- Node status is **Abnormal**; - The **node-exporter** component's pod on the node is in a non-Running state;	- Node's inode free is less than 1000 - Node CPU usage exceeds 60%; - Node memory usage exceeds 60%; - Disk space usage of the node

Resource Type	Inspection Scope	Fault Judgment Conditions	Warning Judgment Conditions
		- The **kubelet** component's pod on the node is in a non-Running state.	directory exceeds 60%; - Node system load exceeds 200% and runtime exceeds 15 minutes; - At least one NodeDeadlock (node deadlock) event occurred in the past day; - At least one NodeOOM (out of memory) event occurred in the past day; - At least one NodeTaskHung (task hung) event occurred in the past day.
pod	All pods	- pod status is **Error**; - The pod has been in the starting state for more than 5 minutes.	- Pod CPU usage exceeds 80%; - Pod memory usage exceeds 80%; - The number of restarts of the Pod in the past 5 minutes is greater than or equal to 1.
Certificate	- Certmanager certificates - Kubernetes certificates	Certificate status is **Expired**.	Certificate's validity period is less than 29 days.

Resource Utilization Inspection

Click on the **Resource Utilization Inspection** tab to enter the **Resource Utilization Inspection** page.

In the **Resource Utilization Inspection** page, you can view the total amount, usage, and usage rate of CPU, memory, and disk of `global` clusters, accessed clusters, and self-built clusters, as well as the number of resources such as clusters, nodes, pods, and projects on the platform.

- **Resource Usage Statistics**: You can view the total amount and total usage rate of CPU, memory, and disk of global, accessed, and self-built clusters.

- **Platform Resource Quantity**: You can view the number of resources running on the platform.

Component Health Status

The platform health status page presents statistical data on the health status of features that have been installed on the platform. When your account has management or auditing permissions related to the platform, you can also view detailed health data for specific features, including: a list of clusters that do not have the feature installed, the health status of clusters that have the feature installed, and detection data for components within clusters associated with the feature. This can help you quickly identify issues and improve the operational efficiency of the platform.

TOC

Procedures to Operate

1. Navigate to the view page of installed products or the platform center (platform management, project management, operations center).
2. Click the question mark button at the top right corner of the navigation bar > **Platform Health Status**.
3. Check the feature card; the feature card displays the health status information of the feature. If there are abnormalities in the feature components, it will be reflected on the card as `fault`.
4. Click on the health/fault value on the feature card to expand the detailed health status page on the right side of the page, where you can view detailed issue information for the faulty

components.