# 腾讯云可观测平台

# Prometheus 监控

腾讯云

# 文档目录

# Prometheus 监控
# Prometheus 监控简介
# Prometheus 监控概述

最近更新时间：2025-01-21 15:32:41

Prometheus 监控服务（TencentCloud Managed Service for Prometheus，TMP）是基于开源 Prometheus 构建的高可用、全托管的服务，与腾讯云容器服务（TKE）高度集成，兼容开源生态丰富多样的应用组件，结合腾讯云可观测平台的告警功能和 Prometheus Alertmanager 能力，为您提供免搭建的高效运维能力，减少开发及运维成本。

## 开源 Prometheus 简介

Prometheus 是一个开源监控系统。与 Kubernetes 相似，Prometheus 受启发于 Google 的 Borgmon 监控系统，而 Kubernetes 也是从 Google 的 Borg 演变而来的。Prometheus 始于2012年，并由 SoundCloud 内部工程师开发，于2015年1月发布。2016年5月，其成为继 Kubernetes 之后第二个正式加入 Cloud Native Computing Foundation（CNCF） 基金会的项目。现在最常见的 Kubernetes 容器管理系统中，通常会搭配 Prometheus 进行监控。
Prometheus 具有如下特性：

- 自定义多维数据模型（时间序列数据由 Metric 和一组 Key/Value Label 组成）。
- 灵活而强大的查询语言 PromQL，可利用多维数据完成复杂的监控查询。
- 不依赖分布式存储，支持单主节点工作。
- 通过基于 HTTP 的 Pull 方式采集时序数据。
- 可通过 PushGateway 的方式来实现数据 Push 模式。
- 可通过动态的服务发现或者静态配置去获取要采集的目标服务器。
- 结合 Grafana 可方便地支持多种可视化图表及仪表盘。

## 产品功能

根据监控分层，Prometheus 监控服务覆盖了业务监控、应用层监控、中间件监控、系统层监控，结合腾讯云可观测平台告警和开源 Grafana 可以提供一站式全方位的监控体系，帮助业务快速发现和定位问题，减轻故障给业务带来的影响。

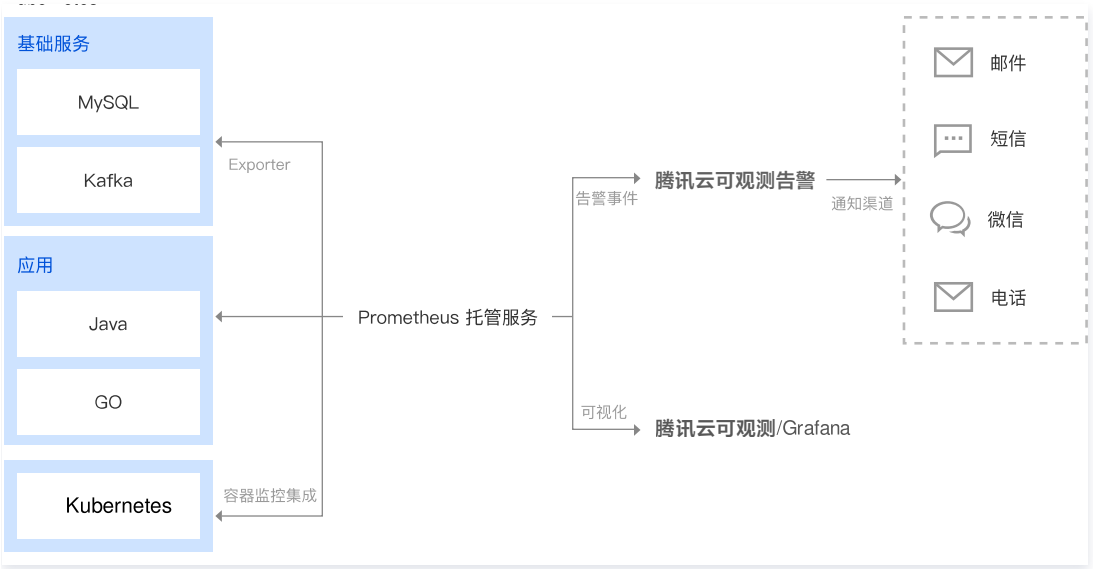
- 系统层监控：例如 CPU、Memory、Disk 和 Network 等。
- 中间组件层监控：例如 Kafka、MySQL 和 Redis 等。
- 应用层监控：应用服务监控，例如 JVM、HTTP 和 RPC 等。
- 业务监控：业务黄金指标，例如登录数和订单量等。

# Prometheus 应用场景

最近更新时间：2024-08-16 11:52:01

## 一体化监控场景

Prometheus 监控服务提供一站式开箱即用的 Prometheus 全托管服务，天然集成开源 Grafana 大盘和腾讯云可观测平台告警。支持基础服务、应用层、容器服务等监控场景。




## 应用服务监控场景

### 场景一

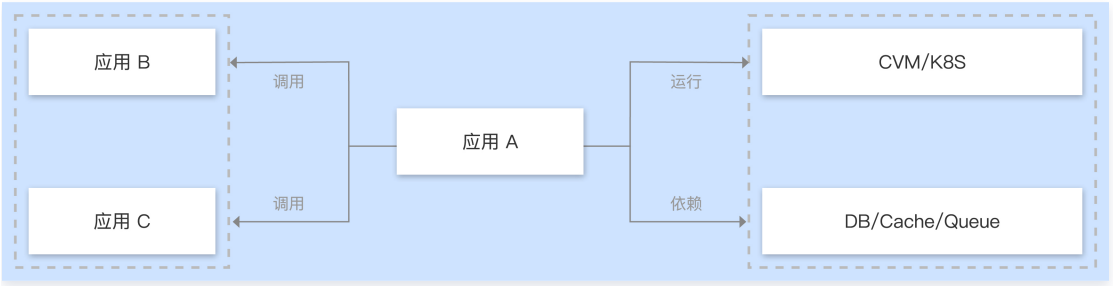
某应用提供了对外的接口服务，但无法了解该接口服务质量。Prometheus 监控服务可对开发语言进行集成，实时对接口的访问量/延时/成功率进行监控。

### 场景二

Prometheus 监控服务同时也会对服务进行异常检测，可了解该异常影响了哪些接口、发生在哪些主机，或者了解该异常是单机问题还是整个集群的共性问题。

### 场景三

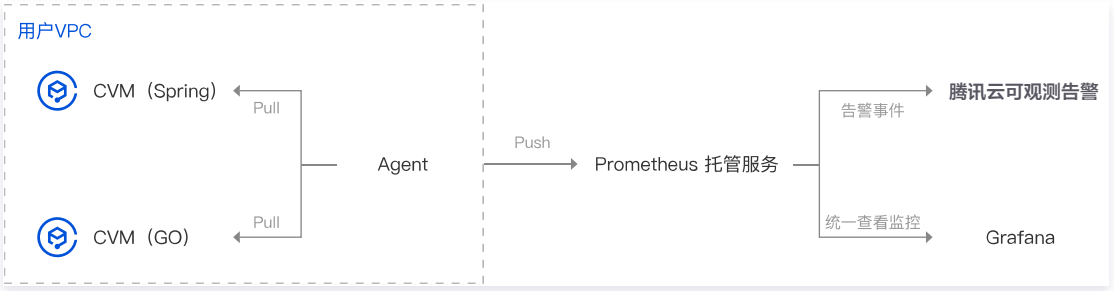
对于 Java 应用来说，可进行单机的 GC/内存/线程状态等监控，全方面地了解 JVM 内部的状态。




## 云服务器监控场景

当您的服务部署在 CVM 上时，几乎每次服务的扩缩容都要修改 Prometheus 的抓取配置。针对这类场景，结合腾讯云平台提供的标签能力和 Prometheus Agent 对腾讯云标签的发现能力，用户只需合理地对 CVM 关联标签即可方便地管理监控目标对象， 免去了需要不断手动更新配置的维护成本，例如：

1. 服务 A 同时部署在 2 台 CVM 上，并对其所在的 CVM 关联标签（服务名：A）。
2. 由于需要进行业务活动，原有 CVM 数量不满足业务活动需求，需再扩容 3 台 CVM，这时只需要对这 3 台 CVM 关联标签（服务名：A）。成功关联后，Agent 就会自动发现新增的3台 CVM，主动抓取监控指标。
3. 活动过后缩容下线 3 台 CVM，服务发现功能会自动感知服务下线，停止抓取监控指标。




## 自定义监控场景

您可以通过 Prometheus 监控服务自定义上报指标监控数据，对应用或者服务内部的一些状态进行监控，如请求处理数、下单数等，也可以对一些核心逻辑的处理耗时进行监控，如请求外部服务的耗时情况等。

# Prometheus 监控优势

最近更新时间：2024-10-22 14:10:22

## 基于开源自建 Prometheus，会遇到哪些问题?

使用开源 Prometheus 需要自行购买相关资源并部署系统（简称自建）。由于开源 Prometheus 自身的短板，自建 Prometheus 也给企业带来了不少困扰。

1. **对于中小企业，使用成本高**

   自建 Prometheus 的使用成本，包括机器资源成本和人力成本，最主要的是人力成本。其中人力成本又包括：

   - 前期调研成本
   - 中期搭建成本
   - 后期维护成本

由于中小企业的运维团队规模较小，一般不多于5人，有的甚至只有一两个人，要自建和维护一套 Prometheus 监控服务，显然非常吃力。

2. **对于大企业，可扩展性差，容易出现性能瓶颈**

   大企业或快速发展的中型企业，在业务发展初期自建 Prometheus 监控，但随着业务量高速的增长，意味着更多资源的投入，对监控也有了更高的要求。自建 Prometheus 就会开始暴露出可扩展性差、性能瓶颈的问题，使企业运维面临巨大的挑战。

**腾讯云 Prometheus 与自建 Prometheus 功能对比**

| 对比类型 | 具体功能 | 腾讯云 Prometheus 监控 | 自建 Prometheus 监控 |
| --- | --- | --- | --- |
| 数据集成 | 集成腾讯云容器服务 | 一键自动集成 | 手动接入，配置复杂 |
| | 跨 VPC/跨地域集成容器 | 自动支持 | 需自行做网络的打通 |
| | 集成基础云产品数据 | 一键安装 | 需自行安装 Exporter |
| | 常用监控组件集成 | 一键安装 | 需自行安装 Exporter |
| | 标签自动识别资源变化 | 支持 | 不支持 |
| 可视化 | 关联 Grafana 可视化 | 支持快速关联托管 Grafana 服务 | 需自行搭建 Grafana |
| | 预设 Dashboard 模板 | 支持 | 不支持 |
| 告警 | 告警通知渠道 | 可复用腾讯云可观测平台-告警管理的渠道 | 需自行搭建 |
| | 告警通知模板 | 支持 | 不支持 |
| 其他能力 | 健康巡检 | 支持 | 不支持 |
| | 预聚合 | 支持 | 不支持 |
| 高可用性 | 多副本 | 支持 | 不支持 |
| | 水平拓展 | 结合腾讯云自研的分片和调度技术，实现动态扩缩，满足用户的弹性需求，同时支持负载均衡 | 无法水平扩展 |
| | 数据存储 | 数据存储能力无上限 | 数据存储受限于本地磁盘大小 |
| 安全管理 | 数据安全 | 基于腾讯云安全体系，支持鉴权管理 | 不支持 |
| 成本 | 人力成本 | 一次性配置，免运维 | • 前期调研<br>• 中期搭建<br>• 后期维护 |
| | 资源成本 | • 按需使用<br>• 按量计费<br>• 容器核心基础指标免费 | • 固定费用<br>• 存在资源浪费的可能 |

# 基本概念

最近更新时间：2024-10-22 14:10:22

**本文汇总使用 Prometheus 监控服务过程中涉及的基本概念，方便您查询和了解相关概念。**

| 概念 | 说明 |
| --- | --- |
| Exporter | Exporter 是一个采集监控数据并通过 Prometheus 监控规范对外提供数据的组件。目前有上百个官方或者第三方 Exporter 可供使用，请参见 Exporter详情 。 |
| Job | 一组 Target 的配置集合。定义了抓取间隔，访问限制等作用于一组 Target 的抓取行为。 |
| Prometheus 实例 | Prometheus 监控服务提供的管理 Prometheus 监控数据采集和数据存储分析的逻辑单元。 |
| Prometheus 探针 | 部署在用户侧或者云产品侧 Kubernetes 集群。负责自动发现采集目标、采集指标和远程写到其他库。 |
| PromQL | Prometheus 监控服务的查询语言。支持瞬时查询和时间跨度查询，内置多种函数和操作符。可以对原始数据进行聚合、切片、预测和联合。 |
| Target | Prometheus Agent 要抓取的采集目标。采集目标暴露自身运行、业务指标，或者代理暴露监控对象的运行、业务指标。 |
| 告警规则 | Prometheus 监控 Alerting Rule 格式的告警配置。可以通过 PromQL 描述。 |
| 标签 | 描述指标的一组 Key-Value 值。 |
| 服务发现 | Prometheus 监控服务的功能特点之一，无需静态配置，可以自动发现采集目标。支持 Kubernetes SD、Consul、Eureka 等多种服务发现方式，支持通过 Service Monitor、Pod Monitor 的方式暴露采集目标。 |
| 预聚合 | Prometheus 监控服务 Recording Rule能 力。可以通过 PromQL 将原始数据加工成新的指标，提升查询效率。 |
| 集成中心 | 集成了 Prometheus 监控服务支持的所有服务，您可以根据页面指引安装对应的服务，成功安装后即可在监控面板查看监控数据。 |
| 告警策略 | 用于定义告警如何触发，如何发送。 |
| 云产品监控 | Prometheus 监控 服务集成了腾讯云云产品的监控数据。可一键安装 Agent 即可查看监控数据。 |
| 指标 | 采集目标暴露的、可以完整反映监控对象运行或者业务状态的一系列标签化数据。 |
| TPS | 每秒数据点的上报总数。它是衡量系统处理能力的重要指标。 |
| Series 上限 | 指标个数上限，Series 上限 = (单个指标 × 该指标的维度组合) × 指标个数。 |

# 相关限制

最近更新时间：2024-11-18 10:25:02

## 实例限制

每个付费实例的 Series 限制最大为450万，免费试用实例为200万。如需调整付费实例的限制可以 联系我们，在资源充足的情况下，我们将会为您合理地调整相关限制。

> **说明**
> 时间序列（时间线，Series）含义：由指标名和标签组成。相同的指标名和标签在时间序列中构成唯一的一条时间线。

## 数据上报限制

若您使用 Prometheus 监控服务上报监控数据，将会有下列指标（ `__name__` 唯一的 Series）限制。

- 上报时必须有指标名，即 `__name__` 标签，指标名必须符合规范，只支持英文字母开头，可包含 ASCII 字符、数字、下划线以及冒号，并必须符合正则表达式 `[a-zA-Z_:][a-zA-Z0-9_:]*` ，参见 Prometheus 官方文档 。
- 每个指标最多32个标签。
- 标签名必须符合规范，可包含 ASCII 字符、数字、下划线，并必须符合正则表达式 `[a-Za-Z_][a-Za-Z0-9_]*` 。标签名称开头为 `__` 仅供内部使用。
- 标签长度：标签名为1024字符，标签值名为 2048 字符。
- 同一个指标下，和标签的维度组合不能超过10万（ 在 histogram 有较多 bucket 的情况下，histogram 类型的指标不支持调整）。
- **每秒上报的数据点总量限制：付费实例不能超过300000，免费试用实例不能超过100000。**

> **说明**
> 标签的作用：Prometheus 中存储的数据为时间序列，是由指标名和一系列的标签（键值对）唯一标识的，不同的标签代表不同的时间序列，即通过指定标签查询指定数据。添加的标签越多，查询的维度越细。

## Prometheus 查询限制

为了保证查询效率及更好的用户体验，Prometheus 查询有如下限制：

- 单个查询涉及的 time series 不能超过100000。
- 单个查询涉及的数据量不能超过100MB。
- 对于查询频次没有限制，但是如果并发量超过 15 可能会有一定的排队延时（较多较慢的大查询可能会出现，一般情况不会出现。时间跨度超过两周的大查询出现延时的概率会相应升高）。

以上限制对告警规则和预聚合规则同样有效，建议根据业务场景来限制查询范围或者其他方式对大查询进行适当拆分，也可以采用先拆分后聚合的方式，例如对预聚合后的结果再次进行聚合。

## 告警与预聚合限制

告警相关的限制：

- 单实例告警规则上限：150。
- 单实例总告警数量上限：2000（超出则直接丢弃）。
- 单实例所有告警的标签、Annotation 等字段总的大小上限：20MiB（超出则直接丢弃）。

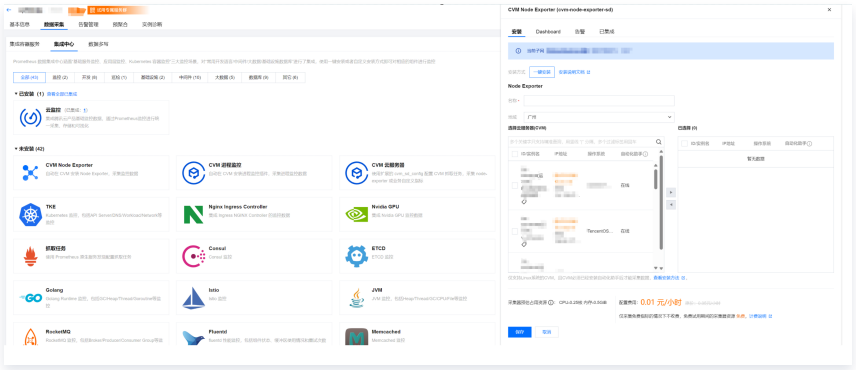
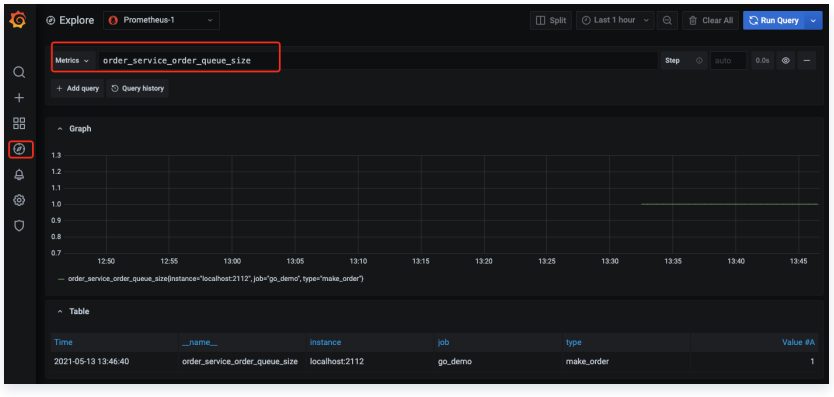
预聚合相关的限制：

- 单实例预聚合规则上限：150。
- 单预聚合规则分组中的预聚合规则上限：35。

# 功能特性

最近更新时间：2024-12-05 16:40:32

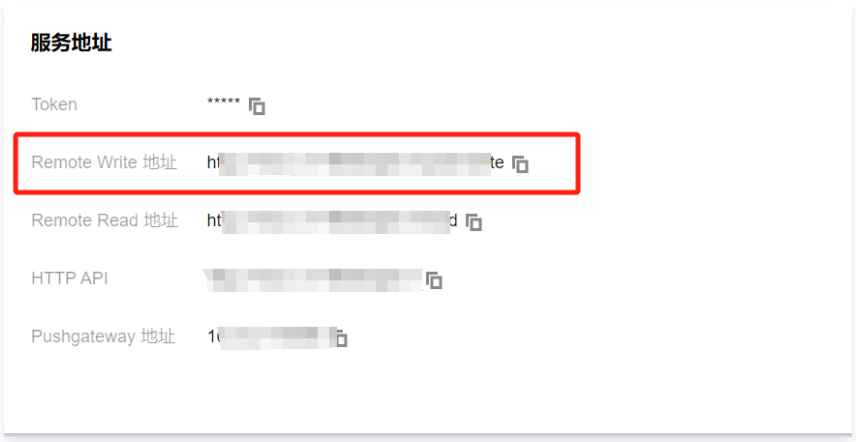
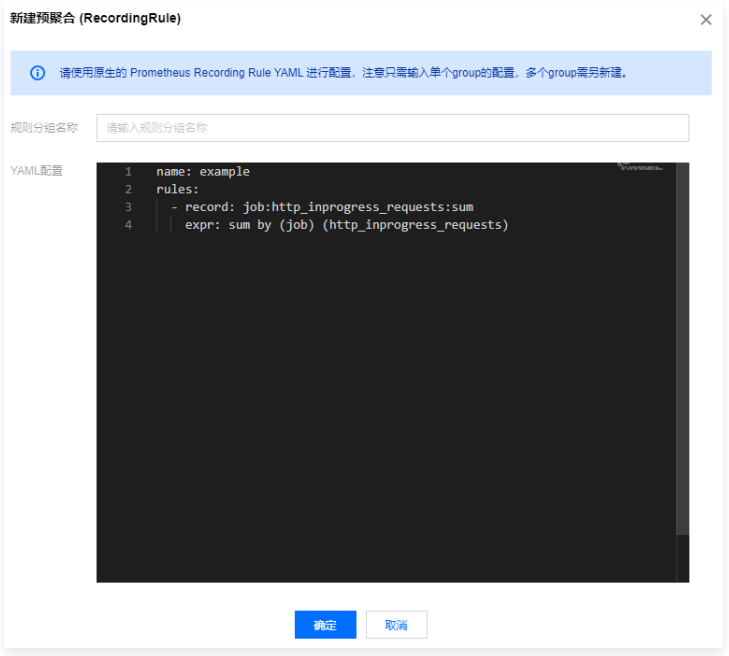
## 监控对象接入

| 功能 | 功能说明 | 控制台示例图 |
| --- | --- | --- |
| 创建 Prometheus 实例 | 支持创建多地域 Prometheus 实例。 |  |
| 集成中心 | 支持多种组件一键安装和自定义接入，成功安装后即可在 Grafana 中查看监控数据。 |  |
| 健康巡检 | 通过定期探测对应服务的连通性，来检测其服务的健康情况，帮助您实时地掌握服务的健康状况，及时发现异常，来提升服务的 SLA。 |  |
| 自定义监控 | 支持自定义上报指标监控数据，对业务内部核心指标进行监控，如请求数，下单数，请求外部服务的耗时情况等。 |  |

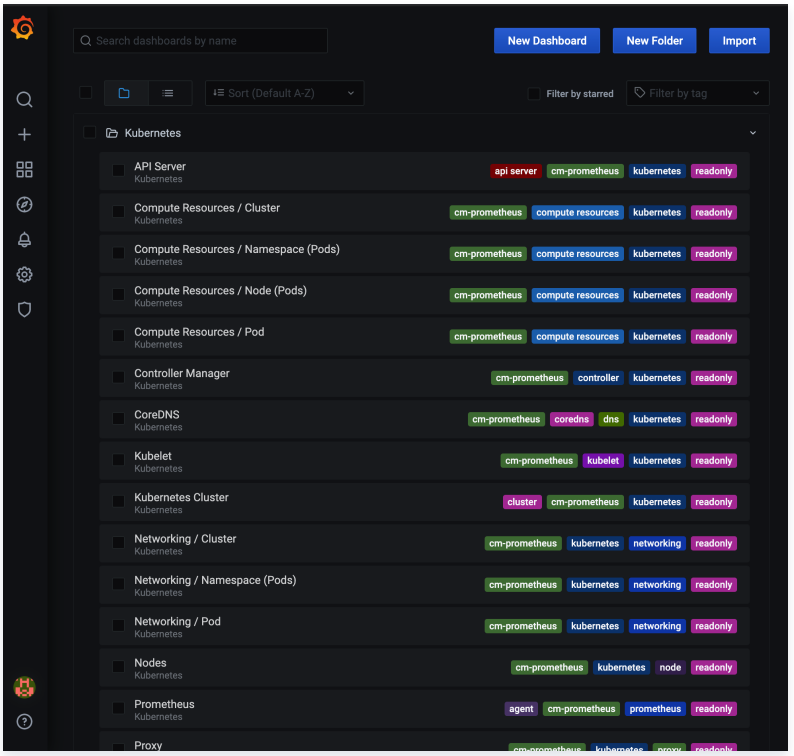
## 监控指标采集

| 功能 | 功能说明 | 控制台示例图 |
| --- | --- | --- |
| 数据采集 | 在数据采集页面，可以进行数据采集配置，并直观查看采集目标。 |  |
| targets | 支持通过 targets 可以直观查看正在被抓取的目标，以及抓取状态是否正常。 |  |

## 监控数据处理

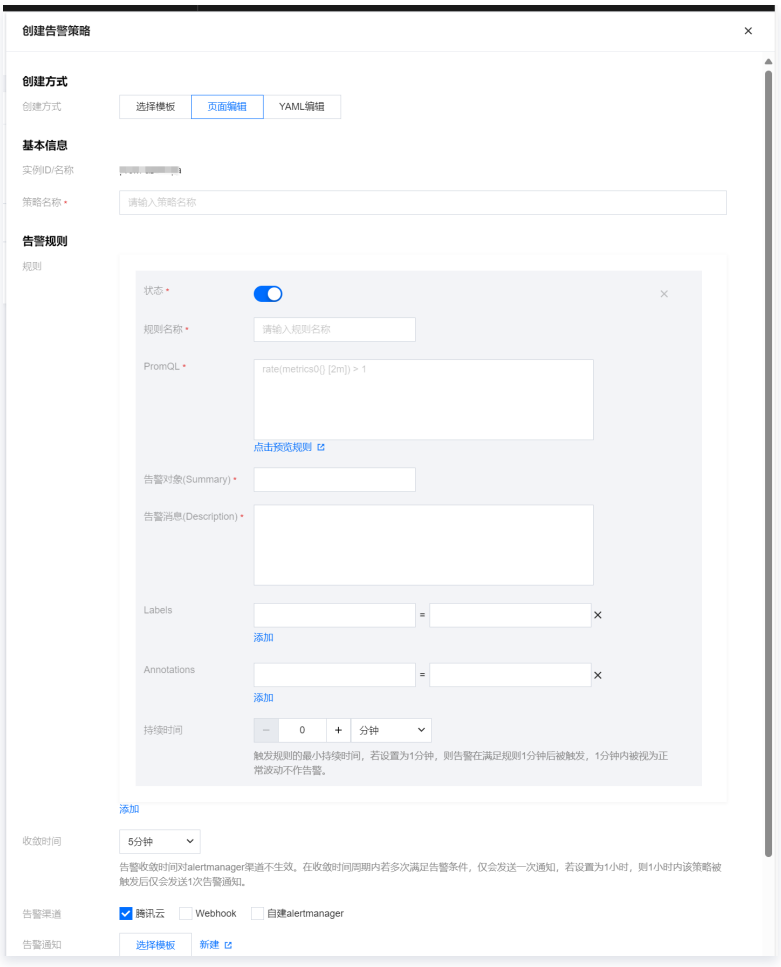
| 功能 | 功能说明 | 控制台示例图 |
| --- | --- | --- |
| 获取 Remote Write 地址 | Remote Write 功能支持作为远程数据库存储 Prometheus 监控服务的数据。您可以使用 Remote Write地址，将自建 Prometheus 的监控数据存储到 Prometheus 监控服务的实例中，实现远程存储，并可视化展示在同一 Grafana。 |  |
| 预聚合 | 对一些常用的指标或者计算相对复杂的指标进行提前计算，然后将这些数据存储到新的数据指标中，提前计算好的指标查询速度更快，可以解决用户配置以及查询慢的问题。 |  |

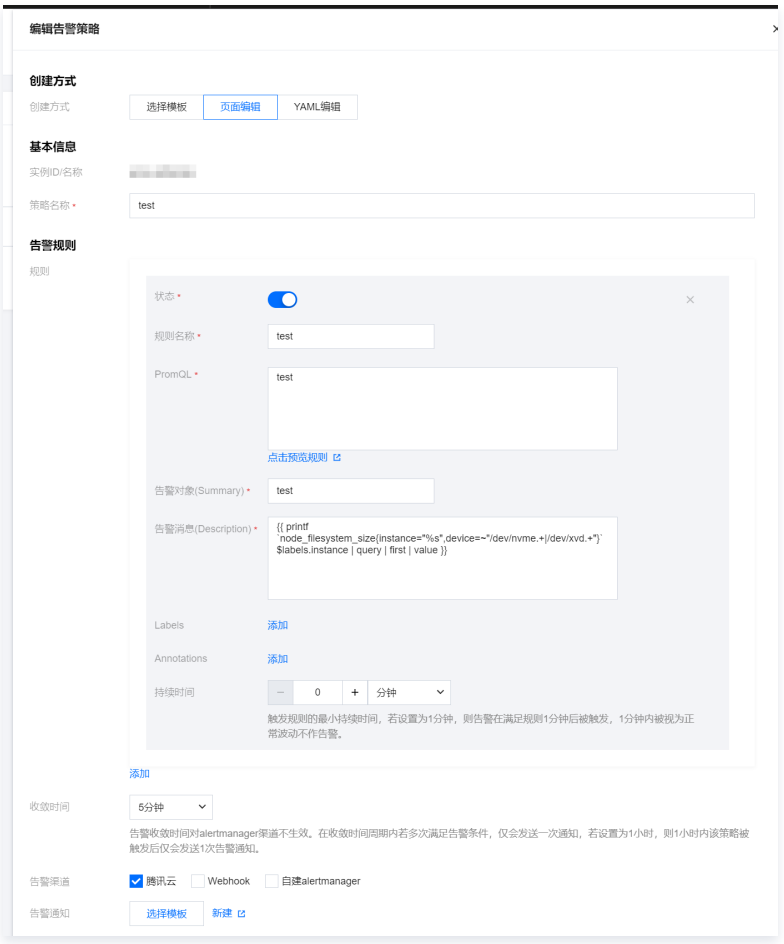
## 监控数据展示

| 功能 | 功能说明 | 控制台示例图 |
| --- | --- | --- |
| Grafana | 预置丰富的 Grafana 大盘，同时也支持自定义大盘。还预设了 Grafana 官网常用的插件， 您可以在控制台一键安装。 |  |
| 实例监控 | 支持 Prometheus 实例状态和使用情况监控，包括 Prometheus 实例存储情况、告警发送情况、Grafana 请求和仪表盘数量等，方便您实时了解 Prometheus 实例使用情况。 |  |

| | | |
|---|---|---|
| HTTP API | 获取 Prometheus 监控服务数据地址。您可以通过该地址将 Prometheus 实例的监控数据接入自建的 Grafana 大盘展示数据，也可以将 Prometheus 监控数据进行二次开发。 |  |

## 告警

| 功能 | 功能说明 | 控制台示例图 |
|---|---|---|
| 创建告警 | 预置多种报警规则，也支持针对特定监控对象自定义告警规则。TMP 集成了腾讯云可观测平台告警通知模板，在某些指标发生异常时及时通知您采取措施。 |  |

| | | |
|---|---|---|
| 管理告警 | 支持对告警策略执行开启、关闭、编辑、删除等操作，同时还支持其他实例告警一键导入。 |  |

# 开服地域

最近更新时间：2025-03-18 15:05:03

**说明：**
开服地域指的是物理的数据中心或服务器位置，可简单理解为一个服务或资源在哪个地区或国家开放，即在开放的地域为用户提供服务和资源。当资源在特定地域的数据中心创建成功后，通常就无法更换地域。

Prometheus 支持的地域和可用区如下：

| 地域 | 可用区 |
| --- | --- |
| 华南地区（广州）ap-guangzhou | 广州三区 ap-guangzhou-3 |
| | 广州四区 ap-guangzhou-4 |
| | 广州六区 ap-guangzhou-6 |
| | 广州七区 ap-guangzhou-7 |
| 华南地区（深圳金融）ap-shenzhen-fsi | 深圳金融一区 ap-shenzhen-fsi-1 |
| | 深圳金融二区 ap-shenzhen-fsi-2 |
| | 深圳金融三区 ap-shenzhen-fsi-3 |
| 华东地区（上海）ap-shanghai | 上海二区 ap-shanghai-2 |
| | 上海三区 ap-shanghai-3 |
| | 上海四区 ap-shanghai-4 |
| | 上海五区 ap-shanghai-5 |
| | 上海八区 ap-shanghai-8 |
| 华东地区（上海金融）ap-shanghai-fsi | 上海金融一区 ap-shanghai-fsi-1 |
| | 上海金融三区 ap-shanghai-fsi-3 |
| 华东地区（南京）ap-nanjing | 南京一区 ap-nanjing-1 |
| | 南京二区 ap-nanjing-2 |
| | 南京三区 ap-nanjing-3 |
| 华东地区（上海自动驾驶云）ap-shanghai-adc | 上海自动驾驶云一区 ap-shanghai-adc-1 |
| 华北地区（北京）ap-beijing | 北京三区 ap-beijing-3 |
| | 北京四区 ap-beijing-4 |
| | 北京五区 ap-beijing-5 |
| | 北京六区 ap-beijing-6 |
| | 北京七区 ap-beijing-7 |
| 华北地区（北京金融）ap-beijing-fsi | 北京金融一区 ap-beijing-fsi-1 |
| | 北京金融二区 ap-beijing-fsi-2 |
| 西南地区（成都）ap-chengdu | 成都一区 ap-chengdu-1 |
| | 成都二区 ap-chengdu-2 |

| | |
|---|---|
| 西南地区（重庆）ap-chongqing | 重庆一区 ap-chongqing-1 |
| 港澳台地区（中国香港）ap-hongkong | 中国香港一区 ap-hongkong-1 |
| | 中国香港二区 ap-hongkong-2 |
| | 中国香港三区 ap-hongkong-3 |
| 港澳台地区（中国台北）ap-taipei | 中国台北一区 ap-taipei-1 |
| 东南亚地区（新加坡）ap-singapore | 新加坡一区 ap-singapore-1 |
| | 新加坡二区 ap-singapore-2 |
| | 新加坡三区 ap-singapore-3 |
| | 新加坡四区 ap-singapore-4 |
| 美国西部地区（硅谷）na-siliconvalley | 硅谷一区 na-siliconvalley-1 |
| | 硅谷二区 na-siliconvalley-2 |
| 欧洲地区（法兰克福）eu-frankfurt | 法兰克福一区 eu-frankfurt-1 |
| 亚太地区（首尔）ap-seoul | 首尔一区 ap-seoul-1 |
| | 首尔二区 ap-seoul-2 |
| 亚太地区（曼谷）ap-bangkok | 曼谷一区 ap-bangkok-1 |
| | 曼谷二区 ap-bangkok-2 |
| 亚太地区（东京）ap-tokyo | 东京一区 ap-tokyo-1 |
| | 东京二区 ap-tokyo-2 |
| 美东地区（弗吉尼亚）na-ashburn | 弗吉尼亚一区 na-ashburn-1 |
| | 弗吉尼亚二区 na-ashburn-2 |
| 南美地区（圣保罗）sa-saopaulo | 圣保罗一区 sa-saopaulo-1 |
| 亚太东南地区（雅加达）ap-jakarta | 雅加达一区 ap-jakarta-1 |

# 接入指南
# 抓取配置说明

最近更新时间：2024-05-17 10:47:11

## 概述

Prometheus 主要通过 Pull 的方式来抓取目标服务暴露出来的监控接口，因此需要配置对应的抓取任务来请求监控数据并写入到 Prometheus 提供的存储中，目前 Prometheus 服务提供了如下几个任务的配置：

- 原生 Job 配置：提供 Prometheus 原生抓取 Job 的配置。
- 云服务器服务发现配置：提供腾讯云服务器实例的服务发现配置。
- Pod Monitor：在 K8S 生态下，基于 Prometheus Operator 来抓取 Pod 上对应的监控数据。
- Service Monitor：在 K8S 生态下，基于 Prometheus Operator 来抓取 Service 对应 Endpoints 上的监控数据。

> **说明**
> `[]` 中的配置项为可选。

## 原生 Job 配置

相应配置项说明如下：

```
# 抓取任务名称，同时会在对应抓取的指标中加了一个 label(job=job_name)
job_name: <job_name>

# 抓取任务时间间隔
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# 抓取请求超时时间
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# 抓取任务请求 URI 路径
[ metrics_path: <path> | default = /metrics ]

# 解决当抓取的 label 与后端 Prometheus 添加 label 冲突时的处理。
# true: 保留抓取到的 label，忽略与后端 Prometheus 冲突的 label；
# false: 对冲突的 label，把抓取的 label 前加上 exported_<original-label>，添加后端 Prometheus 增加的 label；
[ honor_labels: <boolean> | default = false ]

# 是否使用抓取到 target 上产生的时间。
# true: 如果 target 中有时间，使用 target 上的时间；
# false: 直接忽略 target 上的时间；
[ honor_timestamps: <boolean> | default = true ]

# 抓取协议：http 或者 https
[ scheme: <scheme> | default = http ]

# 抓取请求对应 URL 参数
params:
  [ <string>: [<string>, ...] ]

# 通过 basic auth 设置抓取请求头中 `Authorization` 的值，password/password_file 互斥，优先取 password_file 里面的值。
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]
```

```
# 通过 bearer token 设置抓取请求头中 `Authorization` bearer_token/bearer_token_file 互斥，优先取 bearer_token
里面的值。
[ bearer_token: <secret> ]

# 通过 bearer token 设置抓取请求头中 `Authorization` bearer_token/bearer_token_file 互斥，优先取 bearer_token
里面的值。
[ bearer_token_file: <filename> ]

# 抓取连接是否通过 TLS 安全通道，配置对应的 TLS 参数
tls_config:
  [ <tls_config> ]

# 通过代理服务来抓取 target 上的指标，填写对应的代理服务地址。
[ proxy_url: <string> ]

# 通过静态配置来指定 target，详见下面的说明。
static_configs:
  [ - <static_config> ... ]

# HTTP 服务发现配置。原生服务发现配置可参考 Prometheus 官方文档
http_sd_configs:
  [ - <http_sd_config> ... ]

# 在抓取数据之后，把 target 上对应的 label 通过 relabel 的机制进行改写，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
relabel_configs:
  [ - <relabel_config> ... ]

# 数据抓取之后，通过 relabel 机制进行改写 label 对应的值，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
metric_relabel_configs:
  [ - <relabel_config> ... ]

# 一次抓取数据点限制，0：不作限制，默认为 0
[ sample_limit: <int> | default = 0 ]

# 一次抓取 Target 限制，0：不作限制，默认为 0
[ target_limit: <int> | default = 0 ]
```

## static_config 配置

相应配置项说明如下：

```
# 指定对应 target host 的值，如ip:port。
targets:
  [ - '<host>' ]

# 在所有 target 上加上对应的 label，类似全局 label 的概念。
labels:
  [ <labelname>: <labelvalue> ... ]
```

示例：

```
job_name: prometheus
scrape_interval: 30s
static_configs:
- targets:
```

```
  - 127.0.0.1:9090
```

## 云服务器服务发现配置

> **注意：**
> 云服务器服务发现配置 cvm_sd_configs 不是 Prometheus 原生配置，不支持在集成中心或集成容器服务的抓取任务中使用。目前支持在集成中心的 CVM 云服务器集成中配置使用。以下简称 CVM 服务发现。

CVM 服务发现利用腾讯云 API 自动获取 CVM 实例列表，默认使用 CVM 的私网 IP。服务发现产生以下元标签，这些标签可以在 relabel 配置中使用。

| 标签 | 说明 |
| --- | --- |
| __meta_cvm_instance_id | 实例 ID |
| __meta_cvm_instance_name | 实例名 |
| __meta_cvm_instance_state | 实例状态 |
| __meta_cvm_instance_type | 实例机型 |
| __meta_cvm_OS | 实例操作系统 |
| __meta_cvm_private_ip | 私网 IP |
| __meta_cvm_public_ip | 公网 IP |
| __meta_cvm_vpc_id | 网络 ID |
| __meta_cvm_subnet_id | 子网 ID |
| __meta_cvm_tag_<tagkey> | 实例标签值 |
| __meta_cvm_region | 实例所在区域 |
| __meta_cvm_zone | 实例的可用区 |

### cvm_sd_configs 配置说明

```
# 腾讯云的地域，地域列表见文档
https://cloud.tencent.com/document/api/213/15692#.E5.9C.B0.E5.9F.9F.E5.88.97.E8.A1.A8。
region: <string>

# 自定义 endpoint。
[ endpoint: <string> ]

# 访问腾讯云 API 的凭证信息。如果不设置，取环境变量 TENCENT_CLOUD_SECRET_ID 和 TENCENT_CLOUD_SECRET_KEY 的值。
# 如使用集成中心的 CVM 抓取任务进行配置，则无需填写。
[ secret_id: <string> ]
[ secret_key: <secret> ]

# CVM 列表的刷新周期。
[ refresh_interval: <duration> | default = 60s ]

# 抓取 metrics 的端口。
ports:
  - [ <int> | default = 80 ]

# CVM 列表的过滤规则。支持的过滤条件见文档
https://cloud.tencent.com/document/api/213/15728#2.-.E8.BE.93.E5.85.A5.E5.8F.82.E6.95.B0。
filters:
  [ - name: <string>
      values: <string>, [...] ]
```

```
```

**说明**
使用集成中心的 CVM 云服务器配置 cvm_sd_configs 时，该集成自动使用服务预设角色授权确保安全性，无需您手动填写如下参数：secret_id、secret_key、endpoint。

**CVM 云服务器集成配置示例**

```
job_name: demo-monitor
cvm_sd_configs:
- region: ap-guangzhou
  ports:
  - 8080
  filters:
  - name: tag:service
    values:
    - demo
relabel_configs:
- source_labels: [__meta_cvm_instance_state]
  regex: RUNNING
  action: keep
- regex: __meta_cvm_tag_(.*)
  replacement: $1
  action: labelmap
- source_labels: [__meta_cvm_region]
  target_label: region
  action: replace
```

## Pod Monitor

**相应配置项说明如下：**

```
# Prometheus Operator CRD 版本
apiVersion: monitoring.coreos.com/v1
# 对应 K8S 的资源类型，这里面 Pod Monitor
kind: PodMonitor
# 对应 K8S 的 Metadata，这里只用关心 name，如果没有指定 jobLabel，对应抓取指标 label 中 job 的值为
<namespace>/<name>
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus  # namespace不固定，除kube-system下的任意namespace都可以
  labels:
      prom_id: prom-xxx
# 描述抓取目标 Pod 的选取及抓取任务的配置
spec:
  # 填写对应 Pod 的 label，pod monitor 会取对应的值作为 job label 的值。
  # 如果查看的是 Pod Yaml，取 pod.metadata.labels 中的值。
  # 如果查看的是 Deployment/Daemonset/Statefulset，取 spec.template.metadata.labels。
  [ jobLabel: string ]
  # 把对应 Pod 上的 Label 添加到 Target 的 Label 中
  [ podTargetLabels: []string ]
  # 一次抓取数据点限制，0：不作限制，默认为 0
  [ sampleLimit: uint64 ]
  # 一次抓取 Target 限制，0：不作限制，默认为 0
  [ targetLimit: uint64 ]
  # 配置需要抓取暴露的 Prometheus HTTP 接口，可以配置多个 Endpoint
  podMetricsEndpoints:
```

```
  [ - <endpoint_config> ... ] # 详见下面 endpoint 说明
  # 选择要监控 Pod 所在的 namespace，不填为选取所有 namespace
  [ namespaceSelector: ]
    # 是否选取所有 namespace
    [ any: bool ]
    # 需要选取 namespace 列表
    [ matchNames: []string ]
  # 填写要监控 Pod 的 Label 值，以定位目标 Pod  [K8S metav1.LabelSelector]
(https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.24/#labelselector-v1-meta)
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

**示例：**

```
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: redis-exporter # 填写一个唯一名称
    namespace: cm-prometheus  # namespace不固定，除kube-system下的任意namespace都可以
    labels:
        prom_id: prom-xxx # 配置您的实例 ID
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port  # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 Redis 实例 ID
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: ip
        replacement: '1.x.x.x' # 调整成对应的 Redis 实例 IP
    namespaceSelector:    # 选择要监控pod所在的namespace
      matchNames:
      - redis-test
    selector:     # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: redis-exporter
```

## Service Monitor

**相应配置项说明如下：**

```
# Prometheus Operator CRD 版本
apiVersion: monitoring.coreos.com/v1
# 对应 K8S 的资源类型，这里面 Service Monitor
```

```
kind: ServiceMonitor
# 对应 K8S 的 Metadata，这里只用关心 name，如果没有指定 jobLabel，对应抓取指标 label 中 job 的值为 Service 的名称。
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus   # namespace不固定，除kube-system下的任意namespace
  labels:
      prom_id: prom-xxx  # 配置您的实例 ID
# 描述抓取目标 Pod 的选取及抓取任务的配置
spec:
  # 填写对应 Pod 的 label(metadata/labels)，service monitor 会取对应的值作为 job label 的值
  [ jobLabel: string ]
  # 把对应 service 上的 Label 添加到 Target 的 Label 中
  [ targetLabels: []string ]
  # 把对应 Pod 上的 Label 添加到 Target 的 Label 中
  [ podTargetLabels: []string ]
  # 一次抓取数据点限制，0：不作限制，默认为 0
  [ sampleLimit: uint64 ]
  # 一次抓取 Target 限制，0：不作限制，默认为 0
  [ targetLimit: uint64 ]
  # 配置需要抓取暴露的 Prometheus HTTP 接口，可以配置多个 Endpoint
  endpoints:
  [ - <endpoint_config> ... ] # 详见下面 endpoint 说明
  # 选择要监控 Pod 所在的 namespace，不填为选取所有 namespace
  [ namespaceSelector: ]
    # 是否选取所有 namespace
    [ any: bool ]
    # 需要选取 namespace 列表
    [ matchNames: []string ]
  # 填写要监控 Pod 的 Label 值，以定位目标 Pod  [K8S metav1.LabelSelector](https://v1-
17.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#labelselector-v1-meta)
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

**示例：**

```
  apiVersion: monitoring.coreos.com/v1
  kind: ServiceMonitor
  metadata:
    name: go-demo     # 填写一个唯一名称
    namespace: cm-prometheus  # namespace不固定，除 kube-system 下的任意namespace
    labels:
        prom_id: prom-xxx    # 配置您的实例 ID
  spec:
    endpoints:
    - interval: 30s
      # 填写service yaml中Prometheus Exporter对应的Port的Name
      port: 8080-8080-tcp
      # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      path: /metrics
      relabelings:
      # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 app，
      # 我们通过 relabel 的 replace 动作把它替换成了 application
      - action: replace
        sourceLabels:  [__meta_kubernetes_pod_label_app]
        targetLabel: application
    # 选择要监控service所在的namespace
```

```
    namespaceSelector:
      matchNames:
      - golang-demo
    # 填写要监控service的Label值，以定位目标service
    selector:
      matchLabels:
        app: golang-app-demo
```

## endpoint_config 配置

**相应配置项说明如下：**

```
# 对应 port 的名称，这里需要注意不是对应的端口，默认：80，对应的取值如下：
# ServiceMonitor：对应 Service>spec/ports/name；
# PodMonitor：说明如下：
#   如果查看的是 Pod Yaml，取 pod.spec.containers.ports.name 中的值。
#   如果查看的是 Deployment/Daemonset/Statefulset，取 spec.template.spec.containers.ports.name。
[ port: string | default = 80]
# 抓取任务请求 URI 路径
[ path: string | default = /metrics ]
# 抓取协议：http 或者 https
[ scheme: string | default = http]
# 抓取请求对应 URL 参数
[ params: map[string][]string]
# 抓取任务间隔的时间
[ interval: string | default = 30s ]
# 抓取任务超时
[ scrapeTimeout: string | default = 30s]
# 抓取连接是否通过 TLS 安全通道，配置对应的 TLS 参数
[ tlsConfig: TLSConfig ]
# 通过对应的文件读取 bearer token 对应的值，放到抓取任务的 header 中
[ bearerTokenFile: string ]
# 通过对应的 K8S secret key 读取对应的 bearer token，注意 secret namespace 需要和 PodMonitor/ServiceMonitor 相同
[ bearerTokenSecret: string ]
# 解决当抓取的 label 与后端 Prometheus 添加 label 冲突时的处理。
# true：保留抓取到的 label，忽略与后端 Prometheus 冲突的 label；
# false：对冲突的 label，把抓取的 label 前加上 exported_<original-label>，添加后端 Prometheus 增加的 label；
[ honorLabels: bool | default = false ]
# 是否使用抓取到 target 上产生的时间。
# true：如果 target 中有时间，使用 target 上的时间；
# false：直接忽略 target 上的时间；
[ honorTimestamps: bool | default = true ]
# basic auth 的认证信息，username/password 填写对应 K8S secret key 的值，注意 secret namespace 需要和
PodMonitor/ServiceMonitor 相同。
[ basicAuth: BasicAuth ]
# 通过代理服务来抓取 target 上的指标，填写对应的代理服务地址。
[ proxyUrl: string ]
# 在抓取数据之前，把 target 上对应的 label 通过 relabel 的机制进行改写，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
relabelings:
[ - <relabel_config> ...]
# 数据抓取完成之后写入之前，通过 relabel 机制进行改写 label 对应的值，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
metricRelabelings:
[ - <relabel_config> ...]
```

## relabel_config/relabelings 配置

相应配置项说明如下：

```
# 从原始 labels 中取哪些 label 的值进行 relabel，取出来的值通过 separator 中的定义进行字符拼接。
# 如果是 PodMonitor/ServiceMonitor 对应的配置项为 sourceLabels 。
[ source_labels: '[' <labelname> [, ...] ']' ]
# 定义需要 relabel 的 label 值拼接的字符，默认为 ';'。
[ separator: <string> | default = ; ]

# action 为 replace/hashmod 时，通过 target_label 来指定对应 label name。
# 如果是 PodMonitor/ServiceMonitor 对应的配置项为 targetLabel 。
[ target_label: <labelname> ]

# 需要对 source labels 对应值进行正则匹配的表达式。
[ regex: <regex> | default = (.*) ]

# action 为 hashmod 时用到，根据 source label 对应值 md5 取模值。
[ modulus: <int> ]

# action 为 replace 的时候，通过 replacement 来定义当 regex 匹配之后需要替换的表达式，可以结合 regex 正规则表达式替
换。
[ replacement: <string> | default = $1 ]

# 基于 regex 匹配到的值进行相关的操作，对应的 action 如下，默认为 replace:
# replace: 如果 regex 匹配到，通过 replacement 中定义的值替换相应的值，并通过 target_label 设置并添加相应的 label
# keep: 如果 regex 没有匹配到，丢弃
# drop: 如果 regex 匹配到，丢弃
# hashmod: 通过 modulus 指定的值把 source label 对应的 md5 值取模，添加一个新的 label，label name 通过
target_label 指定
# labelmap: 如果 regex 匹配到，使用 replacement 替换对应的 label name
# labeldrop: 如果 regex 匹配到，删除对应的 label
# labelkeep: 如果 regex 没有匹配到，删除对应的 label
[ action: <relabel_action> | default = replace ]
```

# 自定义监控

最近更新时间：2024-10-31 18:11:52

## 操作场景

您可以通过 Prometheus 监控服务自定义上报指标监控数据，对应用或者服务内部的一些状态进行监控，如请求处理数，下单数等，也可以对一些核心逻辑的处理耗时进行监控，如请求外部服务的耗时情况等。
本文以 Go 这个语言为例，介绍如何通过 Prometheus 监控服务进行业务自定义指标上报，可视化及告警。

## 支持开发语言

Prometheus 开源社区官方 SDK：

- Go
- Java or Scala
- Python
- Ruby

其它第3方开发语言 SDK：

- Bash
- C
- C++
- Common Lisp
- Dart
- Elixir
- Erlang
- Haskell
- Lua for Nginx
- Lua for Tarantool
- .NET / C#
- Node.js
- Perl
- PHP
- R
- Rust

更多信息请参见此处

## 数据模型

Prometheus 具有多维分析的能力，数据模型有如下几部分组成。
`Metric Name`(指标名称) + `Labels`(标签) + `Timestamp`(时间戳) + `Value/Sample`(监控值/样品)

- Metric Name(指标名称)：监控对象的含义（例如，http_request_total – 表示当前系统接收到的 HTTP 请求总量）。
- 标签(label)：表示当前样本的特征维度，是一个 K/V 结构，通过这些维度 Prometheus 可以对样本数据进行过滤，聚合等。
- 时间戳(timestamp)：一个精确到毫秒的时间戳。
- 样本值(value)：一个 float64 的浮点型数据表示当前样本的值。

`Metric Name`(指标名称)/`Labels`(标签) 只能由 ASCII 字符、数字、下划线以及冒号组成，并必须符合正则表达式[a-zA-Z_:][a-zA-Z0-9_:]*。

- 更多 Data Model 说明
- Metric/Label 命名最佳实践

## 如何监控埋点

Prometheus 根据监控的不同场景提供了 `Counter` / `Gauge` / `Histogram` / `Summary` 四种指标类型，每种指标类型说明可参见下文。更多说明请参见 Prometheus 官网 METRIC TYPES 。
Prometheus 社区提供了多种开发语言的 SDK，每种语言的使用方法基本上类似，主要是开发语言语法上的区别，下面主要以 Go 作为例子如何上报自定义监控指标数据。

## Counter

计数类型，数据是单调递增的指标，服务重启之后会重置。可以用 Counter 来监控请求数/异常数/用户登录数/订单数等。
如何通过 `Counter` 来监控订单数：

```
package order

import (
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
)

// 定义需要监控 Counter 类型对象
var (
    opsProcessed = promauto.NewCounterVec(prometheus.CounterOpts{
        Name: "order_service_processed_orders_total",
        Help: "The total number of processed orders",
    }, []string{"status"}) // 处理状态
)

// 订单处理
func makeOrder() {
    opsProcessed.WithLabelValues("success").Inc() // 成功状态
    // opsProcessed.WithLabelValues("fail").Inc() // 失败状态

    // 下单的业务逻辑
}
```

例如，通过 rate() 函数获取订单的增长率：

```
rate(order_service_processed_orders_total[5m])
```

## Gauge

当前值，监控打点的时候可对其做加减。可以用 Gauge 来监控当前内存使用率 /CPU 使用率/当前线程数/队列个数等。
如何通过 `Gauge` 来监控订单队列大小：

```
package order

import (
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
)

// 定义需要监控 Gauge 类型对象
var (
    queueSize = promauto.NewGaugeVec(prometheus.GaugeOpts{
        Name: "order_service_order_queue_size",
        Help: "The size of order queue",
    }, []string{"type"})
)

type OrderQueue struct {
    queue chan string
}

func newOrderQueue() *OrderQueue  {
    return &OrderQueue{
        queue: make(chan string,100),
```

```
    }
}

// 产生订单消息
func (q *OrderQueue)produceOrder() {
    // 产生订单消息

    // 队列个数加1
    queueSize.WithLabelValues("make_order").Inc() // 下单队列
    // queueSize.WithLabelValues("cancel_order").Inc() // 取消订单队列
}

// 消费订单消息
func (q *OrderQueue)consumeOrder() {
    // 消费订单消息

    // 队列个数减1
    queueSize.WithLabelValues("make_order").Dec()
}
```

通过 Gauge 指标，直接查看订单每种类型队列的当前大小：

```
order_service_order_queue_size
```

## Histogram

直方图，Prometheus 会根据配置的 `Bucket` 来计算样本的分布情况，后期可以再加工，一般多用于耗时的监控，通过 Histogram 可以计算出P99/P95/P50等耗时，同时也可以监控处理的个数，如果用上 Histogram 就不需要再用 Counter 统计个数。可以用 Histogram 来监控接口响应时间/数据库访问耗时等。
Histogram 和 Summary 的使用方式类似，可以直接参考 Summary 的使用方式。

## Summary

摘要，和 Histogram 有一点类似，也是计算样本的分布情况，区别是 Summary 会在客户端计算出分布情况(P99/P95/Sum/Count)，因此也会更占客户端资源，后期不可再聚合计算处理，同样可以用 Summary 来监控接口响应时间/数据库访问耗时等。
如何通过 `Summary` 来监控订单处理耗时：

```
package order

import (
    "net/http"
    "time"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

// 定义需要监控 Summary 类型对象
var (
    opsProcessCost = promauto.NewSummaryVec(prometheus.SummaryOpts{
        Name: "order_service_process_order_duration",
        Help: "The order process duration",
    }, []string{"status"})
)

func makeOrder() {
    start := time.Now().UnixNano()
    // 下单逻辑处理结束，记录处理耗时
    defer opsProcessCost.WithLabelValues("success").Observe((float64)(time.Now().UnixNano() - start))
```

```
    // 下单的业务逻辑
    time.Sleep(time.Second) // 模拟处理耗时
}
```

通过 Summary 指标，直接查看下单处理平均耗时：

```
order_service_processed_order_duration_sum / order_service_processed_order_duration_count
```

## 暴露 Prometheus 指标

通过 `promhttp.Handler()` 把监控埋点数据暴露到 HTTP 服务上。

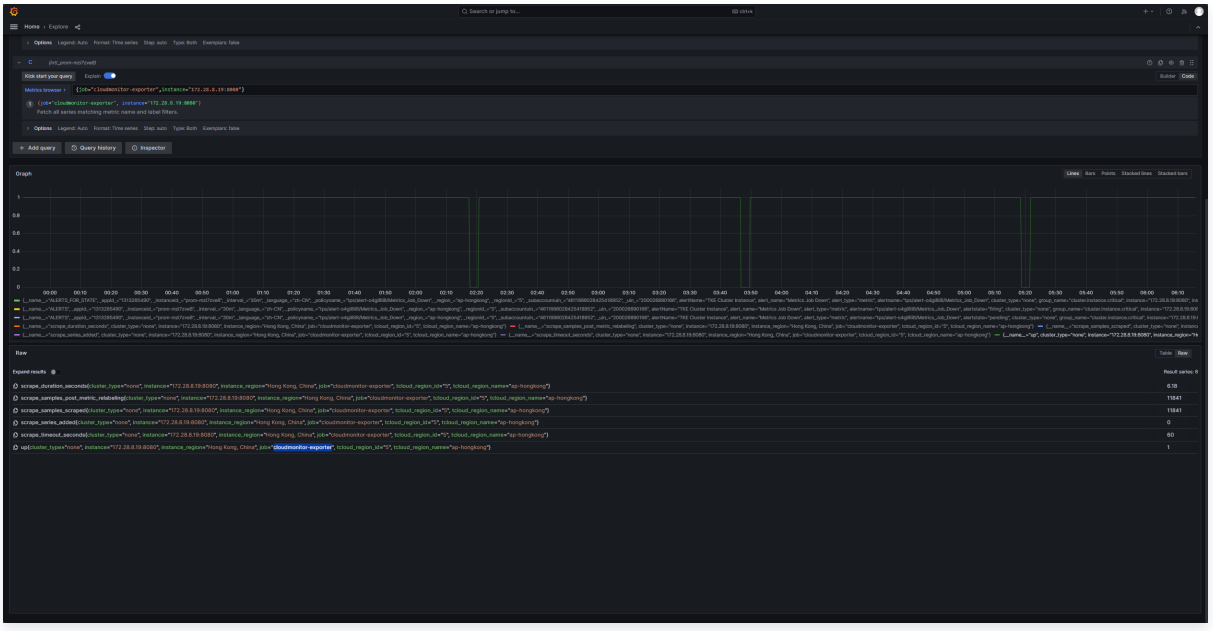
```
package main

import (
    "net/http"

    "github.com/prometheus/client_golang/prometheus/promhttp"
)

func main() {
        // 业务代码

        // 把 Prometheus 指标暴露在 HTTP 服务上
        http.Handle("/metrics", promhttp.Handler())

        // 业务代码
}
```

## 采集数据

完成相关业务自定义监控埋点之后，应用发布，即可通过 Prometheus 来抓取监控指标数据。详情请参见 Golang 接入 。

## 查看监控数据和告警

- 打开 Prometheus 监控服务自带的 Grafana，通过 `Explore` 来查看监控指标数据，如下图，也可以 自定义 Grafana 监控大盘 。



- 通过 `Prometheus` 和 腾讯云可观测平台 `告警管理` 的能力可以对自定义监控指标进行实时告警，详情请参见 告警介绍及使用 。

# EMR 接入
# Flink 接入

最近更新时间：2025-03-06 19:49:22

## 操作场景

在使用 Flink 过程中需要对 Flink 任务运行状态进行监控，以便了解 Flink 任务是否正常运行，排查 Flink 故障等。Prometheus 监控服务对 push gateway 做了集成，支持 Flink 写入 metrics，并提供了开箱即用的 Grafana 监控大盘。

## 前提条件

- 购买的腾讯云弹性 MapReduce（以下简称 EMR）产品包含 Flink 组件，并在实例上跑 Flink 任务。
- 使用与 EMR 相同的地域及私有网络 VPC 购买腾讯云 Prometheus 监控实例。

## 操作步骤

### 产品接入

#### 获取 PushGateway 访问地址

1. 登录 腾讯云可观测平台。
2. 进入 Prometheus 实例，根据 Pushgateway 接入 创建 Pushgateway。
3. 获取 PushGateway 地址。

#### 修改 Flink 配置

1. 进入 弹性 MapReduce，选择并进入对应的“实例”，选择**集群服务**页面。
2. 找到 **Flink** 配置项，选择**配置管理** > **flink-conf.yaml**，进入配置管理页面。
3. 在页面单击**编辑配置** > **新增配置项**，依次添加以下配置，更多信息请参见 官方文档。

| 配置名 | 默认 | 类型 | 描述 | 建议 |
| --- | --- | --- | --- | --- |
| metrics.reporter.promgateway.class | 无 | 字符串 | 实现 metrics 导出到 push gateway 的 Java 类名 | org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporterFactory |
| metrics.reporter.promgateway.jobName | 无 | 字符串 | push 任务名 | 指定方便理解的字符串 |
| metrics.reporter.promgateway.randomJobNameSuffix | true | 布尔 | 是否在任务名后添加随机字符串 | 需设置为 true，如果不添加，Flink 任务间 metrics 会相互覆盖 |
| metrics.reporter.promgateway.groupingKey | 无 | 字符串 | 添加到每个 metrics 的全局 label，格式为 k1=v1;k2=v2 | 添加 EMR 实例 ID 方便区分不同实例的数据，例如 instance_id=emr-xxx |
| metrics.reporter.promgateway.interval | 无 | 时间 | 推送 metrics 的时间间隔，例如30秒 | 建议设置在1分钟左右 |
| metrics.reporter.promgateway.hostUrl | 无 | 字符串 | push gateway 的服务地址 | 控制台上 prometheus 实例的服务地址 |
| metrics.reporter.promgateway.deleteOnShutdown | true | 布尔 | Flink 任务执行完后是否删除 push gateway 上对应的 metrics | 设置为 true |

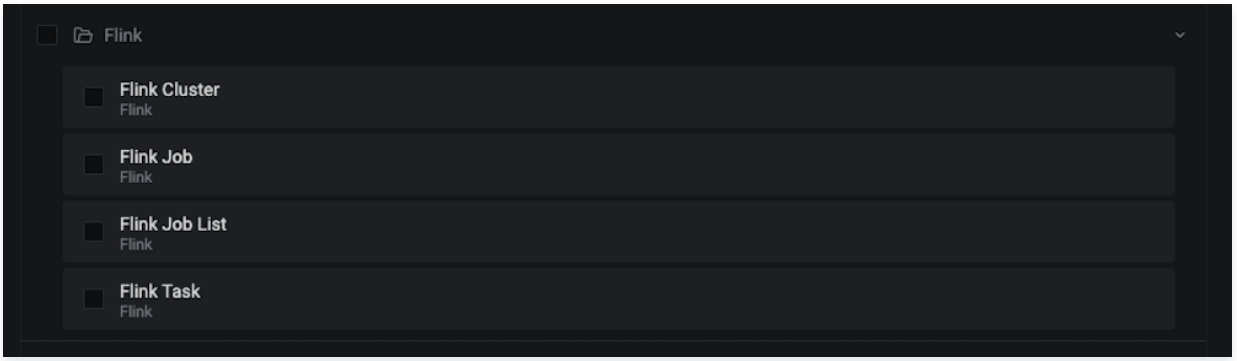
**配置示例如下：**

```
metrics.reporter.promgateway.class: org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporter
metrics.reporter.promgateway.jobName: my-job
```

```
metrics.reporter.promgateway.randomJobNameSuffix: true
metrics.reporter.promgateway.interval: 60 SECONDS
metrics.reporter.promgateway.groupingKey: instance_id=emr-xxxx
metrics.reporter.promgateway.host: pushgateway 的 ip
metrics.reporter.promgateway.port: 8080
```

### 查看监控

1. 登录 Prometheus 监控，选择并进入对应 Prometheus 实例。
2. 选择**数据采集 > 集成中心**，在**集成中心**中找到 **Flink** 监控，在 **Dashboard** 页面单击**安装/升级**即可开启 Flink 监控大盘。
3. 进入 Grafana，单击展开 Flink 监控面板。

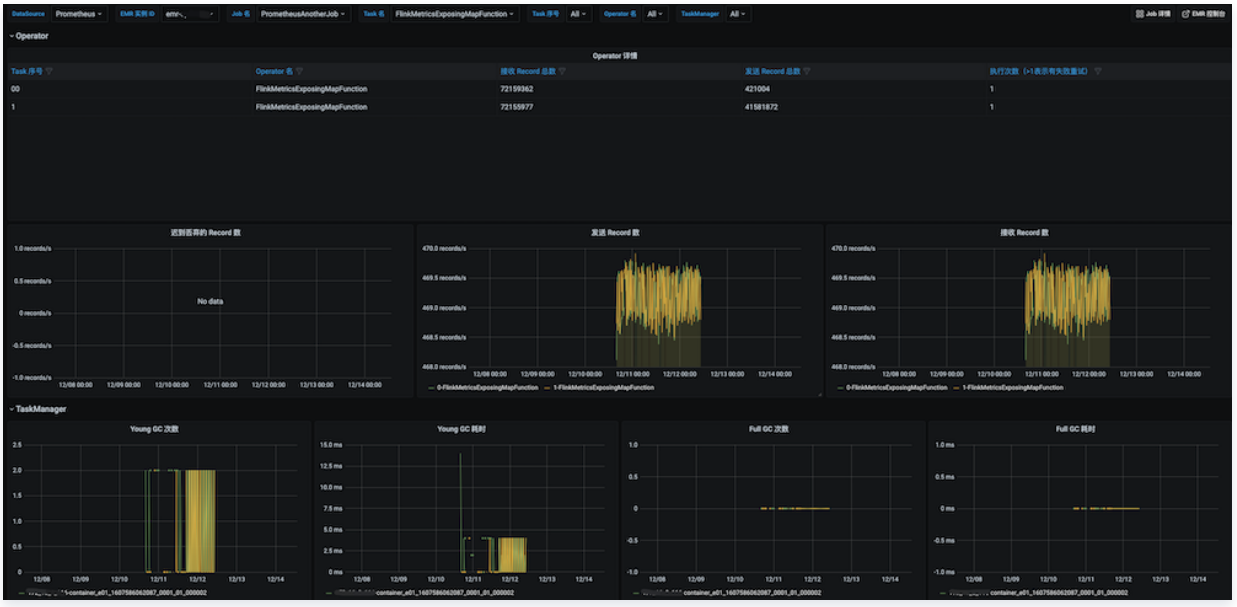



4. 单击 **Flink Job List** 查看监控。

5. 单击表格中的 **Job 名**或 **Job ID 列值**，查看 Job 监控详情。

6. 单击右上角的 **Flink 集群**，查看 Flink 集群监控。

7. 单击表格中的 **Task 名列值**，查看 Task 监控详情。



## 告警接入

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**，选择对应 Prometheus 实例进入管理页面。
3. 选择**告警管理 > 告警策略**，单击**新建告警策略**可以添加相应的告警策略，详情请参见 新建告警策略 。

# Prometheus 采集 EMR 组件

最近更新时间：2025-03-21 17:38:22

## 操作场景

在使用 腾讯云弹性 MapReduce（以下简称 EMR）产品过程中，需要将 EMR 监控指标上报到 Prometheus 监控服务，可参考下文指引。下文将为您介绍如何快速采集 EMR 监控指标。

## 采集 EMR on CVM 实例

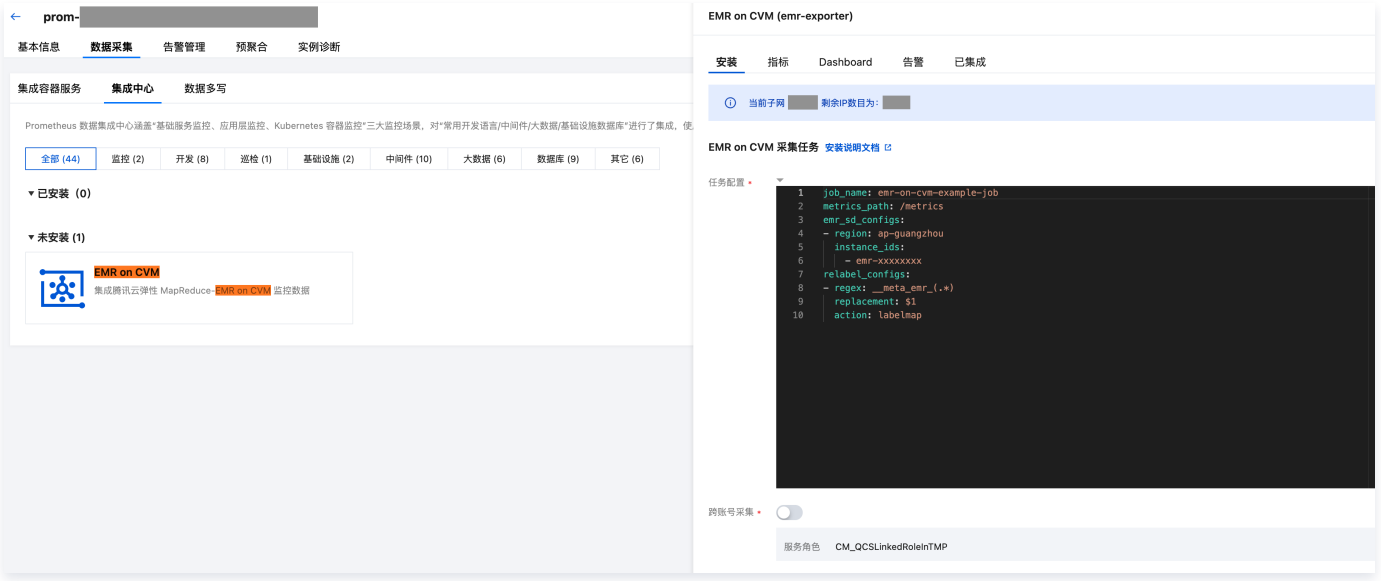
### 前提条件

使用与 EMR 相同的地域及私有网络 VPC，购买腾讯云 Prometheus 监控实例 。可查看 Prometheus 监控服务支持的地域 。

> **说明：**
> - Prometheus 监控服务 通过 EMR 节点内网地址采集监控指标，跨 VPC 采集需要先打通网络。
> - 在网络相通的前提下，支持 跨账号采集 。

### 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控服务**。
3. 在 Prometheus 实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，选择**数据采集 > 集成中心**。
5. 在集成中心找到并单击 **EMR on CVM**，即会弹出一个安装窗口，确认信息后单击**保存**即可。

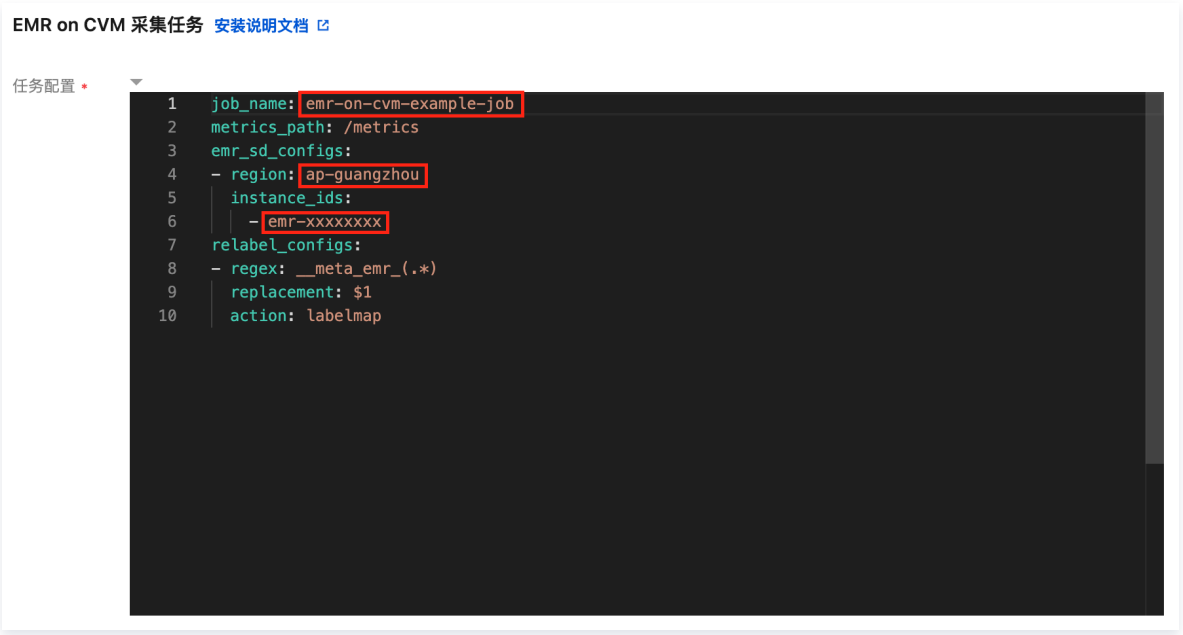



6. 进入 EMR 控制台 ，单击集群的 ID/名称，在**实例信息**页面，获取 EMR 集群所在地域、EMR 实例 ID。


7. 填写任务配置（Yaml 格式）。按下图红框依次填写任务名、EMR 集群所在地域、EMR 实例 ID。

> **说明：**
> - 地域填写格式可以参见 开服地域 的地域说明，例如：`ap-guangzhou` 。
> - 实例 ID 支持填写多个。

- relabel_configs 配置参见 抓取配置说明 。



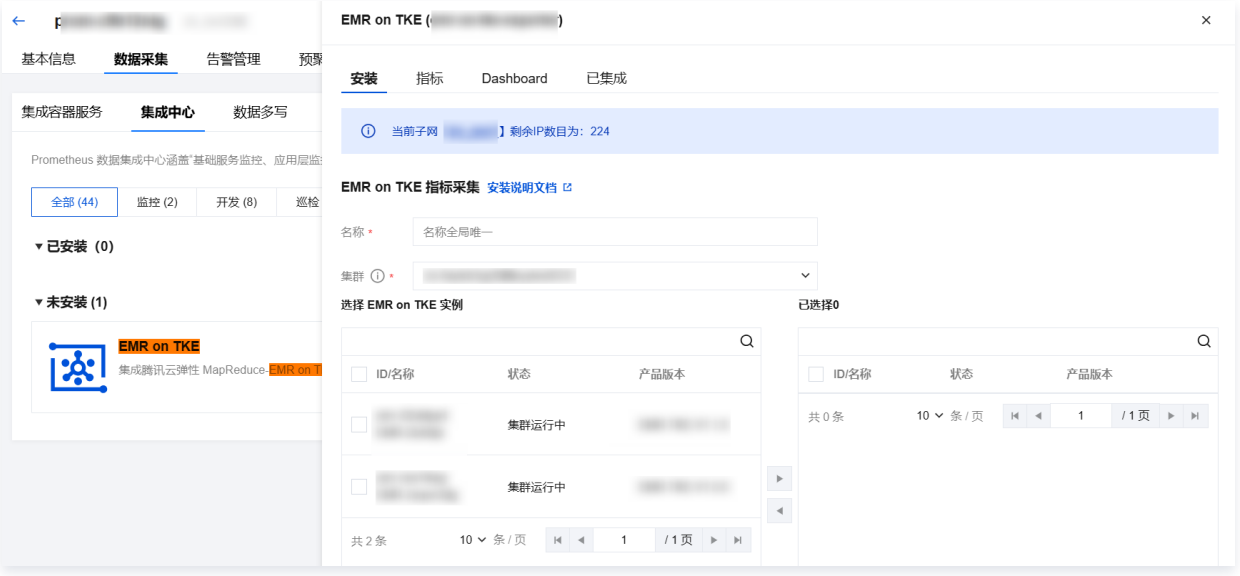

## 采集 EMR on TKE 实例

### 前提条件

Prometheus 实例关联 EMR on TKE 实例所在容器集群，请参见 关联集群 。

### 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控服务**。
3. 在 Prometheus 实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，选择**数据采集 > 集成中心**。
5. 在集成中心找到并单击 **EMR on TKE**，即会弹出一个安装窗口。




6. 选择 EMR on TKE 实例所在容器集群，会自动查找容器集群中的 EMR on TKE 实例，选择需要采集的实例移动到右侧框，单击**保存**即可。

## 采集 EMR Serverless 实例

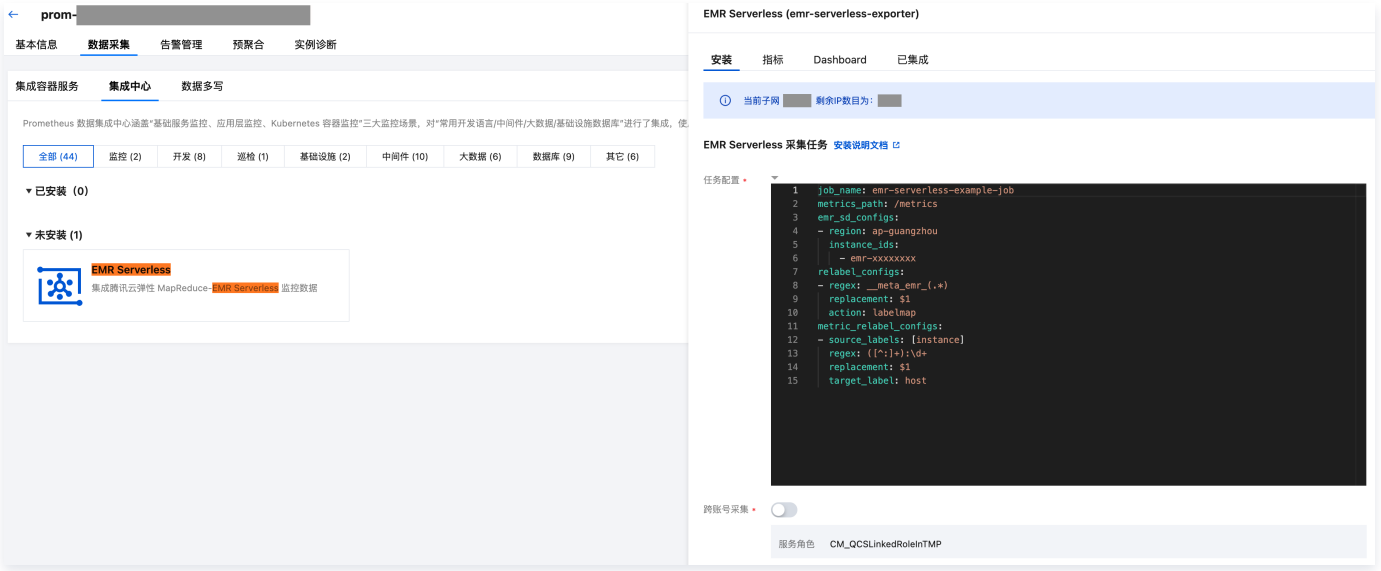
### 前提条件

使用与 EMR 相同的地域及私有网络 VPC，购买腾讯云 Prometheus 监控实例 。可查看 Prometheus 监控服务支持的地域 。

> **说明：**
> - Prometheus 监控服务 通过 EMR 节点内网地址采集监控指标，跨 VPC 采集需要先打通网络。
> - 在网络相通的前提下，支持 跨账号采集 。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控服务**。
3. 在 Prometheus 实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，选择**数据采集 > 集成中心**。
5. 在集成中心找到并单击 **EMR Serverless**，即会弹出一个安装窗口，确认信息后单击**保存**即可。




6. 进入 EMR 控制台 ，单击集群的 ID/名称，在**实例信息**页面，获取 EMR 集群所在地域和 EMR 实例 ID。


7. 填写任务配置（Yaml 格式）。按下图红框依次填写任务名、EMR 集群所在地域、EMR 实例 ID。

> **说明：**
> - 地域填写格式可以参见 开服地域 的地域说明，例如：`ap-guangzhou` 。
> - 实例 ID 支持填写多个。
> - relabel_configs 配置参见 抓取配置说明 。

EMR Serverless 采集任务 安装说明文档

任务配置 *

```
job_name: emr-serverless-example-job
metrics_path: /metrics
emr_sd_configs:
- region: ap-guangzhou
  instance_ids:
    - emr-xxxxxxxx
relabel_configs:
- regex: __meta_emr_(.*)
  replacement: $1
  action: labelmap
metric_relabel_configs:
- source_labels: [instance]
  regex: ([^:]+):\d+
  replacement: $1
  target_label: host
```

## 支持指标

Prometheus 监控服务支持所有 EMR 指标，详细指标列表请参见 EMR 集群监控指标 。

### 如何正确判断服务异常

- EMR on CVM 的 service_status 指标在服务异常或者用户手动停止时都会显示0，无法正确区分异常服务。因为 EMR 无法直接提供服务异常的指标，只能判断用户是否手动停止。Prometheus 监控服务根据服务是否手动停止的信息，新增 emr_additional_service_status 指标，与 service_status 指标组合可用于区分异常服务，值为 0 表示服务异常：

  `(service_status{} * on(instance_id, host, type) group_left() (emr_additional_service_status{} == 1))`
- EMR on TKE、EMR Serverless 不存在上述情况，可直接使用 service_status 指标，值为 0 表示服务异常。

# Java 应用接入
# Spring Boot 接入

最近更新时间：2025-02-25 10:55:52

## 操作场景

在使用 Spring Boot 作为开发框架时，需要监控应用的状态，例如 JVM/Spring MVC 等。Prometheus 监控服务基于 Spring Actuator 机制采集 JVM 等数据，结合配套提供的 Grafana Dashboard 可以方便地监控 Spring Boot 应用的状态。
本文档以在容器服务上部署 Spring Boot 应用为例，介绍如何通过 Prometheus 监控服务监控其状态。

## 前提条件

- 创建 腾讯云容器服务-托管版集群：在腾讯云容器服务中创建 Kubernetes 集群。
- 使用 容器镜像服务 管理应用镜像。
- 应用基于 Spring Boot 框架进行开发。

以下主要以 Maven 为例，说明整个集成。

## 操作步骤

⚠ 注意：

- Spring Boot 已提供 actuator 组件来对应用进行监控，简化了开发的使用成本，所以这里直接使用 actuator 为 Spring Boot 应用进行监控埋点，基于 Spring Boot 2.0及以上的版本，低版本会有配置上的差别需要注意。
- 若您使用 spring boot 1.5接入，接入时和2.0会有一定区别，需要注意如下几点：
  - 访问 `prometheus metrics` 的地址和2.0不一样，1.5默认的是 `/prometheus`，即 `http://localhost:8080/prometheus`。
  - 若报401错误则表示没有权限(Whitelabel Error Page)，1.5默认对 `management` 接口加了安全控制，需要修改 `management.security.enabled=false`。
  - 若项目中用 `bootstrap.yml` 来配置参数，在 `bootstrap.yml` 中修改 `management` 不起作用，需要在 `application.yml` 中修改，原因： spring boot 启动加载顺序有关。
  - `metric common tag` 不能通过 `yml` 来添加，可以通过代码加一个 `bean` 的方式实现，如下：

```
@Component
public class MetricCommonTag {
    @Autowired
    protected MeterRegistry meterRegistry;

    @PostConstruct
    public void initialize() {
        this.meterRegistry.config()
                .commonTags("region", "gz");  // key->value
    }
}
```

## 修改应用的依赖及配置

### 步骤1：修改 pom 依赖

项目中已经引用 `spring-boot-starter-web` 的基础上，在 `pom.xml` 文件中添加 `actuator/prometheus` Maven 依赖项。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
```

```
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

#### 步骤2：修改配置

编辑 `resources` 目录下的 `application.yml` 文件，修改 `actuator` 相关的配置来暴露 Prometheus 协议的指标数据。

```
server:
  tomcat:
    mbeanregistry:
      enabled: true
management:
  endpoints:
    web:
      exposure:
        include: prometheus  # 打开 Prometheus 的 Web 访问 Path
  metrics:
    # 在 Prometheus 中添加特别的 Labels
    # 必须加上对应的应用名，因为需要以应用的维度来查看对应的监控
    tags:
      application: spring-mvc-demo
    # 下面选项建议打开，以监控 http 请求的 P99/P95 等，具体的时间分布可以根据实际情况设置
    distribution:
      sla:
        http:
          server:
            requests: 1ms,5ms,10ms,50ms,100ms,200ms,500ms,1s,5s
      percentiles-histogram:
        http:
          server:
            requests: true # 开启 http server 的请求监控
```

#### 步骤3：本地验证

在项目当前目录下，运行 `mvn spring-boot:run` 之后，可以通过 `http://localhost:8080/actuator/prometheus` 访问到 Prometheus 协议的指标数据，说明相关的依赖配置已经正确。

> **说明：**
> 例子中配置默认配置，对应的端口和路径以实际项目为准。

### 将应用发布到腾讯云容器服务上

#### 步骤1：本地配置 Docker 镜像环境

如果本地之前未配置过 Docker 镜像环境，可以参考 Docker 镜像操作快速入门 进行配置，如果已经配置可以直接执行下一步。

#### 步骤2：打包及上传镜像

1. 在项目根目录下添加 `Dockerfile` ，您可以参考如下示例进行添加，在实际项目中需要修改 `Dockerfile` 。

```
FROM openjdk:21-jdk
WORKDIR /spring-mvc-demo
ADD target/spring-mvc-demo-*.jar /spring-mvc-demo/spring-mvc-demo.jar
CMD ["java","-jar","spring-boot-demo.jar"]
```

2. 打包镜像，在项目根目录下运行如下命令，在实际项目中需要替换对应的 `namespace` 、 `ImageName` 、 `镜像版本号` 。

```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
```

```
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
```

例如：

```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/prom_spring_demo/spring-mvc-demo:latest
docker push ccr.ccs.tencentyun.com/prom_spring_demo/spring-mvc-demo:latest
```

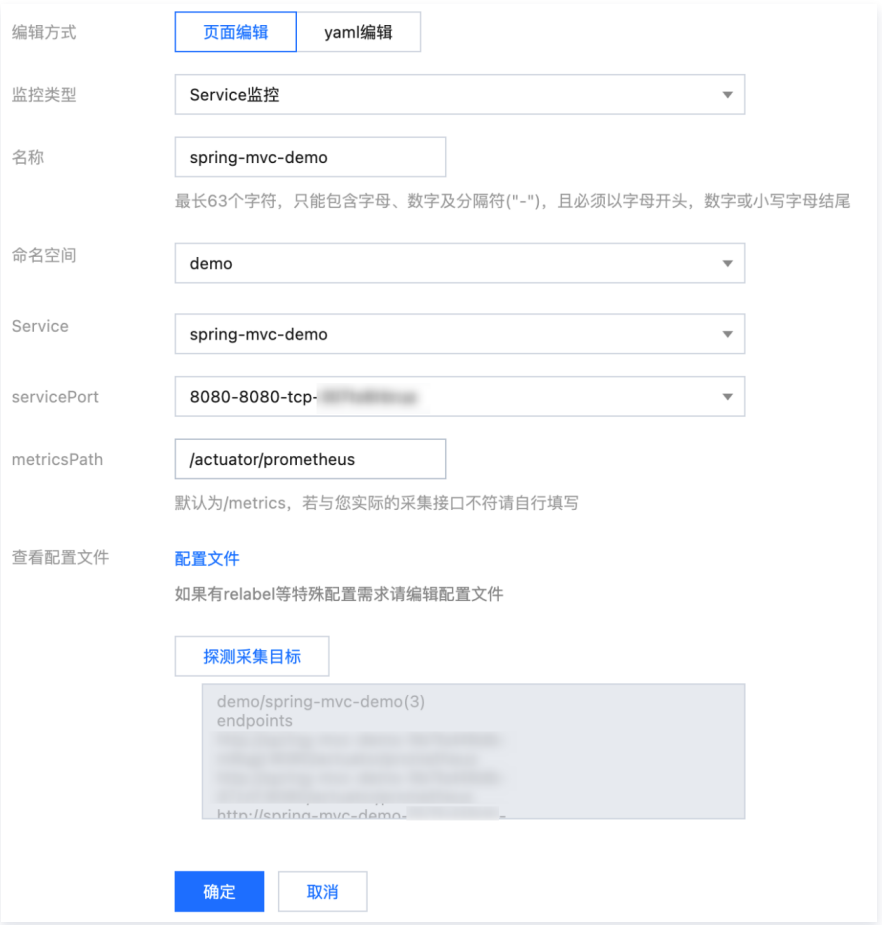
### 步骤3：应用部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中选择**集群**，进入集群列表选择需要部署的容器集群。
3. 选择**工作负载** > **Deployment**，在 **Deployment** 管理页面，选择对应的命名空间来进行部署服务，这里选择通过控制台的方式创建，同时打开 Service 访问方式，您也可以选择通过命令行的方式创建。


名称
spring-mvc-demo
最长63个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母开头，数字或小写字母结尾
描述
请输入描述信息，不超过1000个字符
命名空间
demo
Labels
新增
标签键名称不超过63个字符,仅支持英文、数字、'/'、'-',且不允许以('/')开头。支持使用前缀，更多说明查看详情
标签键值只能包含字母、数字及分隔符("-"、"_"、"."），且必须以字母、数字开头和结尾
数据卷（选填）
添加数据卷
为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储NFS、配置文件、PVC，还需挂载到容器的指定路径中。使用指引
实例内容器
spring-mv...
+ 添加容器
名称
spring-mvc-demo
最长63个字符，只能包含小写字母、数字及分隔符("-")，且不能以分隔符开头或结尾
镜像
prometheus-demo.tencentcloudcr.com/prometheus-demo/spring-mvc-der
选择镜像
镜像版本（Tag）
latest
选择镜像版本
镜像拉取策略
Always
IfNotPresent
Never
总是从远程拉取该镜像
环境变量
新增变量
变量名为空时，在变量名称中粘贴一行或多行key=value或key: value的键值对可以实现快速批量输入
CPU/内存限制
CPU限制
request 0.25 - limit 0.5 核
内存限制
request 256 - limit 1024 MiB
Request用于预分配资源,当集群中的节点没有request所要求的资源数量时,容器会创建失败。
Limit用于设置容器使用资源的最大上限,避免异常情况下节点资源消耗过多。
GPU 资源
卡数: 0 个
配置该工作负载使用的最少GPU资源，请确保集群内已有足够的GPU资源
容器端口
添加容器端口
显示高级设置
实例数量
手动调节
自动调节
直接设定实例数量
实例数量
1 个
镜像访问凭证
prom-demo
添加镜像访问凭证
请指定镜像访问凭证以拉取私有镜像，或参考为服务账号添加 ImagePullSecrets 实现免密拉取； 如无合适的访问凭证，请新建访问凭证
节点调度策略
不使用调度策略
自定义调度规则
可根据调度规则，将Pod调度到符合预期的Label的节点中。设置工作负载的调度规则指引
如果集群开启了注册节点能力，请注意不要将nginx-ingress调度到注册节点上，具体参考节点调度策略
容忍调度
不使用容忍调度
使用容忍调度
显示高级设置

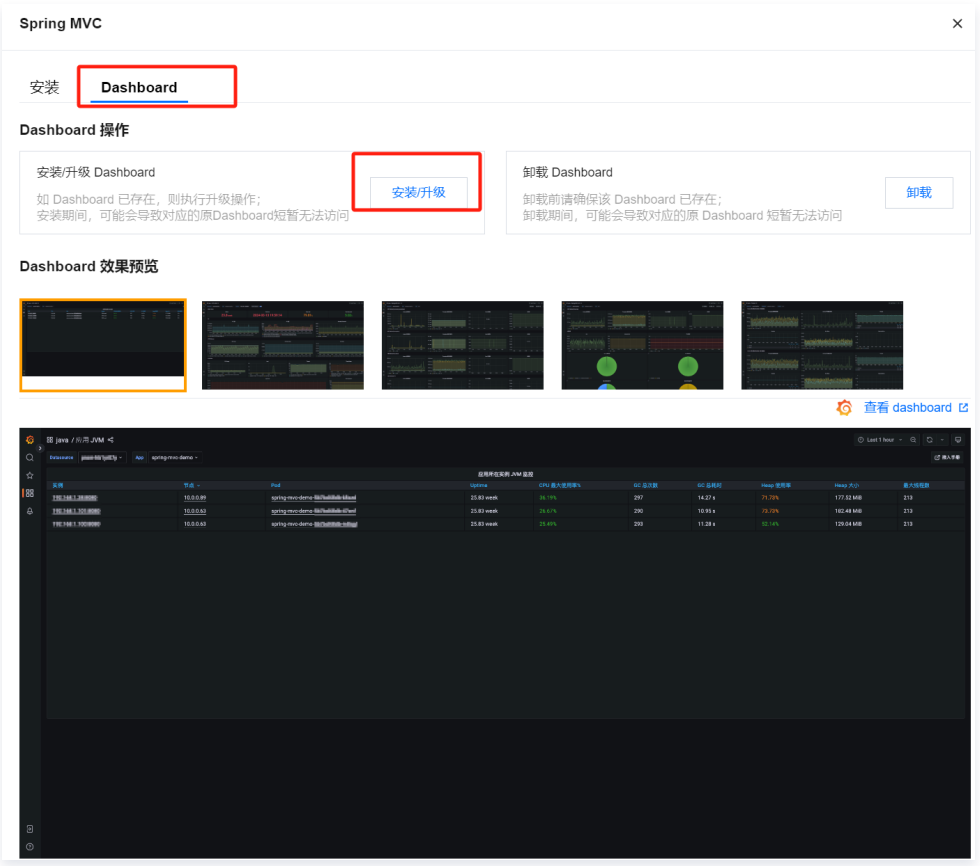

### 步骤4：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，进入到容器服务集成管理页面，选择服务部署所在的容器集群（该集群已经与 Prometheus 实例进行了关联）。
3. 在操作列单击**数据采集配置**，然后单击**新建自定义监控**，添加 ServiceMonitor 采集配置，具体如下：



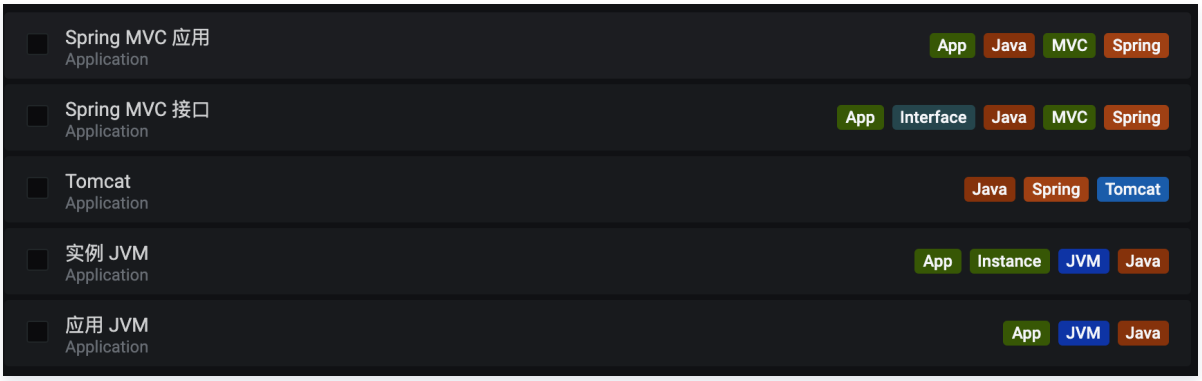

### 步骤5：安装 Dashboard

1. 在**集成中心**下找到并单击 **Spring MVC**。
2. 选择 **Dashboard**，单击 **Dashboard 操作**下的**安装/升级**。




### 步骤6：查看监控

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 打开 Prometheus 实例对应的 Grafana 地址，在 `Dashboards/Manage/Application` 下查看应用相关的监控大屏。
   - **Spring MVC 应用**：监控 MVC 的状态，例如请求耗时/请求量/成功率/异常分布等。
   - **Spring MVC 接口**：接口级监控，可以对应多个接口，方便定位是哪个接口出问题。
   - **Tomcat**：Tomcat 内部状态的监控大屏，例如线程使用情况等。
   - **应用 JVM**：从应用角度出发，查看该应用下所有实例是否有问题，当发现某个实例有问题时可以下钻到对应的实例监控。
   - **实例 JVM**：单实例 JVM 详细的监控数据。


Spring MVC 应用
Application
App Java MVC Spring
Spring MVC 接口
Application
App Interface Java MVC Spring
Tomcat
Application
Java Spring Tomcat
实例 JVM
Application
App Instance JVM Java
应用 JVM
Application
App JVM Java


RED Method(Overview)
Latency(请求耗时)
Throughput(每秒请求数)
Error(失败数)
成功率
P99 P95 P50
now now - 1d now - 7d
now
now
应用层
2XX
3XX/4XX/5XX
日志级别
2XX
5XX
debug info trace warn error
Top 10(请求量)
/hello/{name} 13.71 K
/actuator/prometheus 480
/**/favicon.ico 0
接口异常类型分布
NumberFormatException 344
接口异常实例分布
344
接口异常分布
/hello/{name} 344

# JVM 接入

最近更新时间：2025-03-03 10:05:03

## 操作场景

在使用 Java 作为开发语言时，需要监控 JVM 的性能。Prometheus 监控服务通过采集应用暴露出来的 JVM 监控数据，并提供了开箱即用的 Grafana 监控大盘。
本文介绍了通过 client_java 或 jmx_exporter 两种方式输出 JVM 指标，用 Prometheus 监控服务监控其状态。

> **说明：**
> 若已使用 Spring Boot 作为开发框架，请参见 Spring Boot 接入 。

## 前提条件

- 创建腾讯云容器服务 托管版集群 。
- 使用 容器镜像服务 管理应用镜像。

## 指标埋点

### client_java

client_java 是 Prometheus 官方提供的采集 SDK，提供简洁的 API 自定义指标埋点，还有开箱即用的 JVM 指标，是开发者接入 Prometheus 监控服务的首选方式。

#### 修改应用的依赖及配置

1. 修改 pom 依赖。在 `pom.xml` 文件中添加相关的 Maven 依赖项，1.x 版本做了重构和老版本已经不兼容，优先选择最新版本，示例如下：

```
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>prometheus-metrics-core</artifactId>
    <version>1.3.3</version>
</dependency>
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>prometheus-metrics-instrumentation-jvm</artifactId>
    <version>1.3.3</version>
</dependency>
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>prometheus-metrics-exporter-httpserver</artifactId>
    <version>1.3.3</version>
</dependency>
```

2. 修改代码。初始化并注册 Metrics，示例如下：

```
package org.example;

import io.prometheus.metrics.core.metrics.Counter;
import io.prometheus.metrics.exporter.httpserver.HTTPServer;
import io.prometheus.metrics.instrumentation.jvm.JvmMetrics;

import java.io.IOException;

public class Main {
    public static void main(String[] args) throws InterruptedException, IOException {
        JvmMetrics.builder().register(); // 初始化并注册 JVM 指标
```

```
        // 初始化并注册业务自定义指标
        Counter counter = Counter.builder()
                .name("my_count_total")
                .help("example counter")
                .labelNames("status")
                .register();
        counter.labelValues("ok").inc();
        counter.labelValues("ok").inc();
        counter.labelValues("error").inc();
        // 启动 metrics http server
        HTTPServer server = HTTPServer.builder()
                .port(9400)
                .buildAndStart();
        System.out.println("HTTPServer listening on port http://localhost:" + server.getPort() +
"/metrics")

        Thread.currentThread().join();
    }

}
```

3. 本地验证。本地启动之后，可以通过 `http://localhost:9400/metrics` 访问到 Prometheus 协议的指标数据。

```
http://localhost:9400/metrics
```

### 将应用发布到腾讯云容器服务上

1. 本地配置 Docker 镜像环境。如果本地之前未配置过 Docker 镜像环境，可以参见容器镜像服务 Docker 镜像操作快速入门 进行配置。若已配置请执行下一步。
2. 打包及上传镜像。

   2.1 在项目根目录下添加 `Dockerfile`，请根据实际项目进行修改。示例如下：

```
FROM openjdk:8-jdk
WORKDIR /java-demo
ADD target/java-demo-*.jar /java-demo/java-demo.jar
CMD ["java","-jar","java-demo.jar"]
```

   2.2 打包镜像，在项目根目录下运行如下命令，需要替换对应的 [namespace]、[ImageName] 和 [镜像版本号]。

```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
```

## jmx_exporter

jmx_exporter 是 Prometheus 官方 exporter，把 JVM 原生 MBeans 数据转换为 Prometheus 格式的指标并通过 HTTP 服务暴露出来。jmx_exporter 以 Java Agent 无代码侵入方式运行，但是只能暴露已经注册到 MBeans 上的指标，无法做业务自定义埋点。对于绝大部分的开发者，client_java 是最常用的接入手段。

### 准备 jmx_exporter 资源

1. 下载 jar 包。在项目 发布页 下载最新版本的 Java Agent Jar 包，这里以1.0.1为例。

```
wget
https://repo.maven.apache.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/1.0.1/jmx_prometheus_ja
vaagent-1.0.1.jar
```

2. 准备配置文件。此处使用最少可用配置，各个配置项的详细说明请参见 文档 。

```
rules:
- pattern: ".*"
```

3. 本地验证。jmx_exporter 启动格式是 `-javaagent:<jmx_exporter jar 路径>=< metric 暴露端口>:< jmx_exporter 配置文件路径>` 。

```
java -javaagent:./jmx_prometheus_javaagent-1.0.1.jar=9400:./jmx.yml -jar demo.jar
```

4. 正常启动后，访问 `http://localhost:9400/metrics` 会返回 jvm 开头的指标。

```
 http://localhost:9400/metrics
```

**将应用发布到腾讯云容器服务上**

1. 本地配置 Docker 镜像环境。如果本地之前未配置过 Docker 镜像环境，请参见容器镜像服务 Docker 镜像操作快速入门 进行配置。若已配置请执行下一步。
2. 打包及上传镜像。

   2.1 在项目根目录下添加 `Dockerfile` ，请根据实际项目进行修改。示例如下：

```
FROM openjdk:8-jdk
WORKDIR /java-demo
# 添加下载的 jmx_exporter jar 包
ADD ./jmx_prometheus_javaagent-1.0.1.jar /java-demo/jmx_prometheus_javaagent-1.0.1.jar
# 添加准备的 jmx_exporter 配置文件
ADD ./jmx.yml  /java-demo/jmx.yml
ADD ./target/application.jar /java-demo/java-demo.jar
# 通过 java agent 注入 jmx_exporter
CMD ["java", "-javaagent:/java-demo/jmx_prometheus_javaagent-1.0.1.jar=9400:/java-demo/jmx.yml", "-
jar", "java-demo.jar"]
```

   2.2 打包镜像，在项目根目录下运行如下命令，需要替换对应的 [namespace]、[ImageName] 和 [镜像版本号]。

```
docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
```

## 应用部署

1. 登录 容器服务控制台，在左侧菜单栏中集群页面，选择需要部署的容器集群。
2. 通过**工作负载 > Deployment** 进入 **Deployment** 管理页面，选择对应的**命名空间**进行部署服务，通过 YAML 来创建对应的 Deployment，YAML 配置如下。

> **说明：**
> 如需通过控制台创建，请参见 Spring Boot 接入 。

```
apiVersion: apps/v1
kind: Deployment
metadata:
    labels:
      k8s-app: java-demo
    name: java-demo
    namespace: java-demo
spec:
    replicas: 1
    selector:
      matchLabels:
        k8s-app: java-demo
```

```
    template:
      metadata:
        labels:
          k8s-app: java-demo
      spec:
       containers:
       - image: ccr.ccs.tencentyun.com/prometheus-demo/java-demo
         imagePullPolicy: Always
         name: java-demo
         ports:
          - containerPort: 9400
            name: metric-port
```

## 添加采集任务

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**，选择对应 Prometheus 实例进入管理页面。
3. 选择**数据采集 > 集成容器服务**，进入到容器服务集成管理页面。
4. 选择**数据采集配置 > 新增自定义监控 > yaml 编辑 > PodMonitors** 新增抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: java-demo
  namespace: java-demo
spec:
  namespaceSelector:
    matchNames:
    - java-demo
  podMetricsEndpoints:
  - interval: 15s
    path: /metrics
    port: metric-port
    relabelings:
    - action: replace
      sourceLabels:
      - __meta_kubernetes_pod_label_k8s_app
      targetLabel: application
  selector:
    matchLabels:
      k8s-app: java-demo
```

## 查看监控

1. 进入对应 Prometheus 实例，在**数据采集 > 集成中心**中找到 **JVM 监控**，安装对应的 Grafana Dashboard 即可开启 JVM 监控大盘。
2. 打开 Prometheus 实例对应的 Grafana 地址，在 `Dashboards` 下查看应用相关的监控大屏。
   - **应用 JVM**：从应用角度出发，查看该应用下所有实例是否存在异常，当发现某个实例有异常时，可以下钻到对应的实例监控。


   - **实例 JVM**：单实例 JVM 详细的监控数据。


Quick Facts
Start time
2024/11/12 14:24:05
Uptime
1.5 hour
Info
Oracle Corporation, OpenJDK Runtime Environment, 1.8.0_342-b07
Heap usage
1.76%
Non-Heap usage
2.43%
GC
GC Count
3.025 ops/s
3.020 ops/s
3.015 ops/s
3.010 ops/s
3.005 ops/s
3.000 ops/s
15:25
15:30
15:35
15:40
15:45
15:50
Copy
MarkSweepCompact
GC Average Cost
1.00000 ms
750.00000 µs
500.00000 µs
250.00000 µs
0 s
JVM Memory
JVM Heap
191 MiB
95.4 MiB
0 B
used
committed
max
JVM Non-Heap
572 MiB
381 MiB
191 MiB
0 B
JVM Total
954 MiB
763 MiB
572 MiB
381 MiB
191 MiB
0 B
Memory Pool[Metaspace]

# Golang 应用接入

最近更新时间：2025-03-18 15:05:03

Prometheus 提供了 官方版 Golang 库 用于采集并暴露监控数据，本文为您介绍如何使用官方版 Golang 库来暴露 Golang runtime 相关的数据，以及其它一些基本简单的示例，并使用 Prometheus监控服务来采集指标展示数据等。

## 添加指标

> **说明：**
> Golang Client API 相关的文档请参见 GoDoc 。

### 安装

通过 `go get` 命令来安装相关依赖，示例如下。

```
go get github.com/prometheus/client_golang/prometheus
go get github.com/prometheus/client_golang/prometheus/promauto
go get github.com/prometheus/client_golang/prometheus/promhttp
```

### 开始（运行时指标）

1. 准备一个 HTTP 服务，路径通常使用 `/metrics` 。可以直接使用 `prometheus/promhttp` 里提供的 `Handler` 函数。
   如下是一个简单的示例应用，通过 `http://localhost:2112/metrics` 暴露 Golang 应用的一些默认指标数据（包括运行时指标、进程相关指标以及构建相关的指标）。

   ```
   package main

   import (
           "net/http"

           "github.com/prometheus/client_golang/prometheus/promhttp"
   )

   func main() {
           http.Handle("/metrics", promhttp.Handler())
           http.ListenAndServe(":2112", nil)
   }
   ```

2. 执行以下命令启动应用。

   ```
   go run main.go
   ```

3. 执行以下命令，访问基础内置指标数据。

   ```
   curl http://localhost:2112/metrics
   ```

### 应用层面指标

1. 上述示例仅暴露了一些基础的内置指标。应用层面的指标还需要额外添加（后续我们将提供一些 SDK 方便接入）。如下示例暴露了一个名为 `myapp_processed_ops_total` 的 计数类型 指标，用于对目前已经完成的操作进行计数。如下每两秒操作一次，同时计数器加1。

   ```
   package main

   import (
           "net/http"
   ```

```
        "time"

        "github.com/prometheus/client_golang/prometheus"
        "github.com/prometheus/client_golang/prometheus/promauto"
        "github.com/prometheus/client_golang/prometheus/promhttp"
)

func recordMetrics() {
        go func() {
                for {
                        opsProcessed.Inc()
                        time.Sleep(2 * time.Second)
                }
        }()
}

var (
        opsProcessed = promauto.NewCounter(prometheus.CounterOpts{
                Name: "myapp_processed_ops_total",
                Help: "The total number of processed events",
        })
)

func main() {
        recordMetrics()

        http.Handle("/metrics", promhttp.Handler())
        http.ListenAndServe(":2112", nil)
}
```

2. 执行以下命令启动应用。

```
go run main.go
```

3. 执行以下命令，访问暴露的指标。

```
curl http://localhost:2112/metrics
```

从输出结果我们可以看到 `myapp_processed_ops_total` 计数器相关的信息，包括帮助文档、类型信息、指标名和当前值，如下所示。

```
# HELP myapp_processed_ops_total The total number of processed events
# TYPE myapp_processed_ops_total counter
myapp_processed_ops_total 666
```

## 使用 Prometheus 监控服务

### 步骤1：打包并部署应用

1. Golang 应用一般可以使用如下形式的 Dockerfile（按需修改）。

```
FROM golang:alpine AS builder
RUN apk add --no-cache ca-certificates \
        make \
        git
COPY . /go-build
RUN cd /go-build && \
        export GO111MODULE=on && \
        export GOPROXY=https://goproxy.io && \
```

```
        go build -o 'golang-exe' path/to/main/

FROM alpine
RUN apk add --no-cache tzdata
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs
COPY --from=builder /go-build/golang-exe /usr/bin/golang-exe
ENV TZ Asia/Shanghai
CMD ["golang-exe"]
```

2. 镜像可以使用 腾讯云的镜像仓库，或者使用其它公有或者自有镜像仓库。
3. 需要根据应用类型定义一个 Kubernetes 的资源，这里我们使用 Deployment，示例如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: golang-app-demo
  labels:
    app: golang-app-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: golang-app-demo
  template:
    metadata:
      labels:
        app: golang-app-demo
    spec:
      containers:
      - name: golang-exe-demo:v1
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

4. 同时需要 Kubernetes Service 做服务发现和负载均衡。

```
apiVersion: v1
kind: Service
metadata:
  name: golang-app-demo
spec:
  selector:
    app: golang-app-demo
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

> ⚠ **注意：**
> 必须添加一个 Label 来标明目前的应用，Label 名不一定为 `app`，但是必须有类似含义的 Label 存在，其它名字的 Label 我们可以在后面添加数据采集任务的时候做 relabel 来达成目的。

## 步骤2：添加数据采集任务

当服务运行起来之后，需要进行如下操作让腾讯云 Prometheus 监控服务发现并采集监控指标：

1. 登录 腾讯云可观测平台 Prometheus 控制台，单击新建的**实例 ID/名称**。
2. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：

3. 进入数据采集配置页面后，单击**新建自定义监控**，在弹出的页面中选择 **yaml 编辑**，如下图所示：


4. 通过此方式添加 `Service Monitor`，目前支持基于 `Labels` 发现对应的目标实例地址，因此可以对一些服务添加特定的 `K8S Labels`，可以使 `Labels` 下的服务都会被 Prometheus 服务自动识别出来，不需要再为每个服务添加采集任务，以上面的例子配置信息如下：

> **说明：**
> `port` 的取值为 `service yaml` 配置文件里的 `spec/ports/name` 对应的值。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: go-demo     # 填写一个唯一名称
  namespace: cm-prometheus   # namespace固定，不要修改
spec:
  endpoints:
  - interval: 30s
    # 填写service yaml中Prometheus Exporter对应的Port的Name
    port: 2112
    # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
```

```
    path: /metrics
    relabelings:
    # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 app，
    # 我们通过 relabel 的 replace 动作把它替换成了 application
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # 选择要监控service所在的namespace
  namespaceSelector:
    matchNames:
    - golang-demo
    # 填写要监控service的Label值，以定位目标service
  selector:
    matchLabels:
      app: golang-app-demo
```

> **注意：**
> 示例中名称为 application 的 Label 必须配置，否则无法使用我们提供一些其它的开箱即用的集成功能。更多高阶用法请参见 ServiceMonitor 或 PodMonitor 。

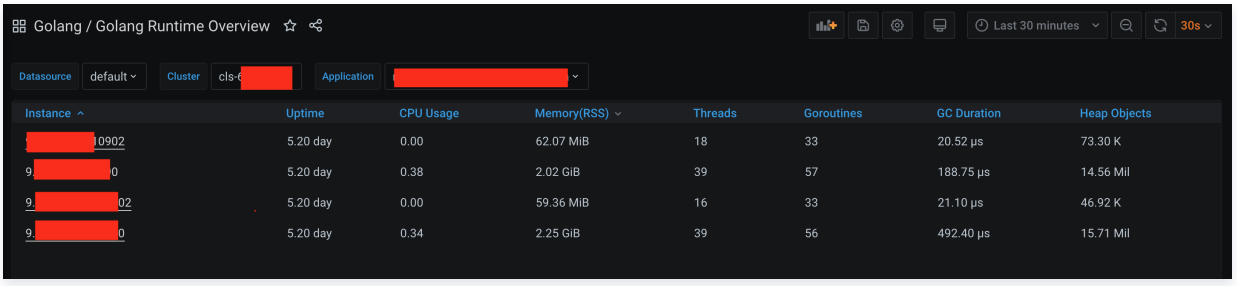
## 步骤3：查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 在 Prometheus 实例 列表，找到对应的 Prometheus 实例，单击实例 ID 右侧图标，打开 Grafana，输入您的账号密码，即可进行 Grafana 可视化大屏操作区。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到并单击 **Golang 监控**，选择 **Dashboard > Dashboard 操作 > Dashboard 安装/升级**，单击**安装/升级**，安装对应的 Grafana Dashboard。
3. 进入 Grafana，单击图表，展开监控面板，单击对应的监控图表名称即可查看监控数据。




## 总结

本文通过两个示例展示了如何将 Golang 相关的指标暴露给 Prometheus 监控服务，以及如何使用 Grafana 可视化的图表查看监控数据。文档只使用了计数类型 Counter 的指标，对于其它场景可能还需要 Gauge、Histogram 以及 Summary 类型的指标，请参见 指标类型 。
对于其它应用场景，我们会集成更多框架提供更多开箱即用的指标监控、可视化面板以及告警模板。

# Exporters 接入
# ElasticSearch Exporter 接入

最近更新时间：2025-02-18 09:21:32

## 操作场景

在使用 ElasticSearch 过程中需要对 ElasticSearch 运行状态进行监控，例如集群及索引状态等，Prometheus 监控服务提供了基于 Exporter 的方式来监控 ElasticSearch 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 ElasticSearch Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 ElasticSearch 是腾讯云 ElasticSearch Service，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装（推荐）

#### 前提条件

Prometheus 实例所在私有网络 VPC 与 ElasticSearch 网络相通。

#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集** > **集成中心**。
4. 在**集成中心**找到并单击 **ElasticSearch**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | ElasticSearch 的用户名称。 |
| 密码 | ElasticSearch 的密码。 |
| 地址 | ElasticSearch 的连接地址。 |
| 标签 | 给指标添加自定义 Label。 |
| Exporter 配置 | • 所有节点：勾选表示查询集群中所有节点的统计信息；不勾选表示仅查询连接的节点的统计信息。<br>• 索引状态：勾选表示查询集群中所有索引的统计信息。<br>• 索引配置：勾选表示查询集群中所有索引配置的统计信息。<br>• 分片：勾选表示查询集群中所有索引的统计信息，包括分片级别的统计信息（相当于勾选了索引状态）。<br>• 快照：勾选表示查询集群快照的统计信息。<br>• 集群配置：勾选表示查询集群配置的统计信息。 |

## 方式二：自定义安装

> **说明：**

> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

### 前提条件

在您执行操作前，请确认已满足以下条件：

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台，选择并进入对应的 Prometheus 实例，在**数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。详情可参见指引 关联集群。

### 操作步骤

#### 步骤1：部署 ElasticSearch Exporter

1. 登录 容器服务控制台。
2. 在左侧菜单栏中选择**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 使用 Secret 管理 ElasticSearch 连接串。

> **说明：**
> ElasticSearch 连接串的格式为 `<proto>://<user>:<password>@<host>:<port>`，例如 `http://admin:pass@localhost:9200`。

4.1 选择**工作负载 > Deployment**，进入 Deployment 页面。

4.2 在页面右上角单击 **YAML 创建**，创建 YAML 配置，配置说明如下：

使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 ElasticSearch Exporter 的时候直接使用 Secret Key，需要调整对应的 URI，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: es-secret-test
  namespace: es-demo
type: Opaque
stringData:
  esURI: you-guess  #对应 ElasticSearch 的 URI
```

5. 部署 ElasticSearch Exporter。

在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: es-exporter # 根据业务需要调整成对应的名称
  name: es-exporter # 根据业务需要调整成对应的名称
  namespace: es-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: es-exporter # 根据业务需要调整成对应的名称
  template:
    metadata:
      labels:
        k8s-app: es-exporter # 根据业务需要调整成对应的名称
    spec:
      containers:
```

```
      - env:
          - name: ES_URI
            valueFrom:
              secretKeyRef:
                name: es-secret-test # 对应上一步中的 Secret 的名称
                key: esURI # 对应上一步中的 Secret Key
          - name: ES_ALL
            value: "true"
        image: ccr.ccs.tencentyun.com/rig-agent/es-exporter:1.1.0
        imagePullPolicy: IfNotPresent
        name: es-exporter
        ports:
        - containerPort: 9114
          name: metric-port
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

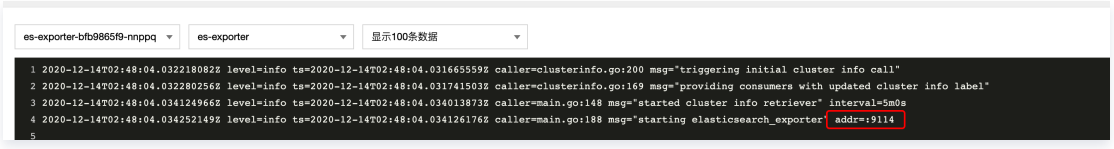
**说明**

上述示例通过 `ES_ALL` 采集了所有 ElasticSearch 的监控项，可以通过对应的参数进行调整，Exporter 更多详细的参数请参见 elasticsearch_exporter 。

6. 验证。

6.1 在 **Deployment** 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

6.2 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：




6.3 单击 **Pod 管理**页签进入 Pod 页面。

6.4 在右侧的操作项下单击**远程登录**登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 ElasticSearch 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
curl localhost:9114/metrics
```

执行结果如下图所示：

```
# HELP elasticsearch_breakers_estimated_size_bytes Estimated size in
# TYPE elasticsearch_breakers_estimated_size_bytes gauge
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",clus
2.0102643e+07
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",clus
1.9926654e+07
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",clus
1.9685163e+07
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clust
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clust
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clust
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_reques
0
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_reques
1167
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_reques
1167
elasticsearch_breakers_estimated_size_bytes{breaker="parent",cluster=
2.0102643e+07
```

**步骤2：添加采集任务**

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 在**数据采集 > 集成容器服务**页面选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: es-exporter      # 填写一个唯一名称
  namespace: cm-prometheus    # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  namespaceSelector:
    matchNames:
      - es-demo
  podMetricsEndpoints:
  - interval: 30s
    path: /metrics
    port: metric-port
  selector:
    matchLabels:
      k8s-app: es-exporter
```

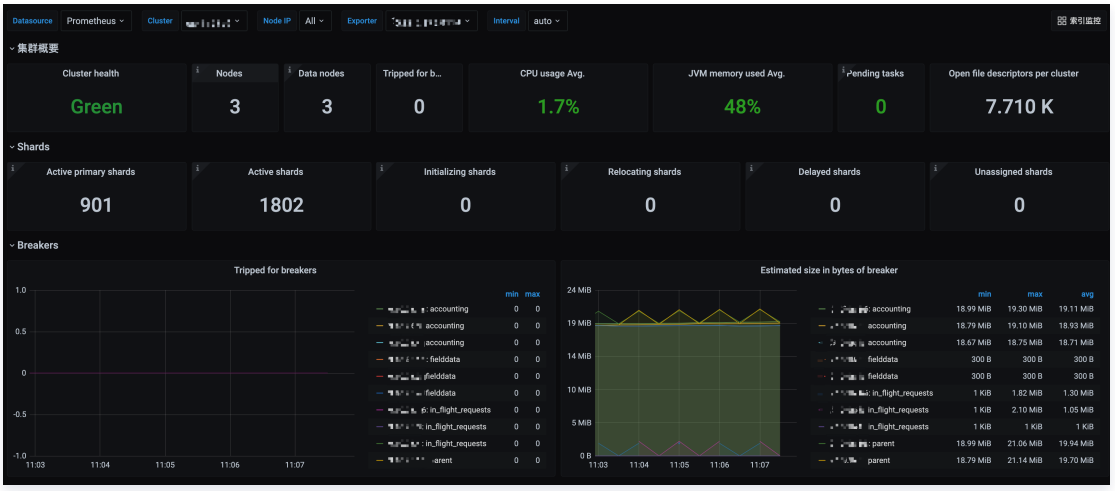
## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，进入集成中心页面。找到 ElasticSearch 监控，选择 **Dashboard > Dashboard 操作 > Dashboard 安装/升级**，单击**安装/升级**，安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中单击 Grafana 图标即可自动打开 Kafka 监控大盘，查看实例相关的监控数据，如下图所示：


## 配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**告警管理 > 告警配置**，单击**新建告警策略**添加相应的告警策略，详情请参见 新建告警策略 。

# Kafka Exporter 接入

最近更新时间：2024-12-09 18:41:52

## 操作场景

在使用 Kafka 过程中需要对 Kafka 运行状态进行监控，例如集群状态、消息消费情况是否有积压等， Prometheus 监控服务提供基于 Exporter 的方式来监控 Kafka 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 Kafka Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 Kafka 是腾讯云 消息队列 CKafka 版，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装(推荐)

#### 前提条件

- Prometheus 实例所在私有网络 VPC 与 Kafka 网络相通。
- 需在 Kafka 放通 Prometheus IPv4 地址的读权限，详细步骤可参见 配置 ACL 策略 。


#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集**，再点击**集成中心**。
4. 在集成中心找到并单击 **Kafka**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


**配置说明**

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 地址 | 填写 Kafka Broker 的连接地址。 |
| Kafka 版本 | 选填，部分特定版本必填，例如 0.10.2.0。 |
| 标签 | 给指标添加自定义 Label。 |
| topic 过滤正则 | 选填，不填默认采集全部的 topic。填写后只会采集符合正则的 topic。 |
| group 过滤正则 | 选填，不填默认采集全部的 group。填写后只会采集符合正则的 group。 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台 > 选择**对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。详情可参见 关联集群。
- 需在 Kafka 放通 Prometheus IPv4 地址的读权限，同上文 一键安装 的前提条件。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
5. 在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: kafka-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
  name: kafak-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
  namespace: kafka-demo # 集群的 namespace，exporter 会部署在该 namespace 下
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: kafka-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
  template:
    metadata:
      labels:
        k8s-app: kafka-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
    spec:
      containers:
      - args:
        - --kafka.server=x.x.x.x:9092 # 对应 Kafka 实例的地址信息
        image: ccr.ccs.tencentyun.com/rig-agent/kafka-exporter:v1.3.0
        imagePullPolicy: IfNotPresent
        name: kafka-exporter
        ports:
        - containerPort: 9121
          name: metric-port  # 这个名称在配置抓取任务的时候需要
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
```

```
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

> **说明：**
> Exporter 详细参数请参见 kafka_exporter 。

**步骤二：添加采集任务**

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 进入**集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 自定义监控 > 新建自定义监控 > YAML编辑**，来添加采集配置。
   监控类型选择 `PodMonitors`，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: kafka-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s # 采集间隔
    port: metric-port # 填写步骤一 yaml 中的 spec.template.spec.containers[0].ports[0].name
    path: /metrics # Exporter 的指标采集路径，默认填 /metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'ckafka-xxxxxx' # 调整成对应的 Kafka 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 Kafka 实例 IP
  namespaceSelector:
    matchNames:
    - kafka-demo # 填写 Exporter 所在的 namespace
  selector:  # 填写要监控 pod 的 Label 值，以定位目标 pod
    matchLabels:
      k8s-app: kafka-exporter
```

> **说明：**
> 由于 `Exporter` 和 `Kafka` 部署在不同的服务器上，因此建议通过 Prometheus Relabel 机制将 Kafka 实例的信息放到监控指标中，以便定位问题。

## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。

2. 选择**数据采集 > 集成中心**，在集成中心页面找到 Kafka 监控，选择 **Dashboard 操作 > Dashboard 安装/升级** 来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可自动打开 Kafka 监控大盘，查看实例相关的监控数据，如下图所示：



## 配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**告警管理**，可以添加相应的告警策略，详情请参见 新建告警策略 。

# Rabbitmq Exporter 接入

最近更新时间：2025-02-08 15:55:33

## 操作场景

RabbitMQ Exporter 是一个用于监控 RabbitMQ 消息队列监控系统的 Prometheus 插件，用于收集和暴露 RabbitMQ 的性能指标。通过 RabbitMQ Exporter，用户可以实时了解消息队列的运行状况，包括消息的传输速率、队列的大小、连接数等指标信息。腾讯云可观测平台 Prometheus 提供了与 RabbitMQ Exporter 集成及开箱即用的 Grafana 监控大盘。
从 RabbitMQ 3.7.0版本开始，RabbitMQ 内置了 Prometheus 插件，RabbitMQ 可以直接作为 Prometheus 采集目标，用于 RabbitMQ 的性能监控，而 RabbitMQ Exporter 导出指标可以作为内置采集的补充数据。

## 接入方式：RabbitMQ 内置导出

### 前提条件

RabbitMQ 版本不低于3.7.0。

### 启用 RabbitMQ 的 Prometheus 插件

#### 开源 RabbitMQ 开启

开启方式：

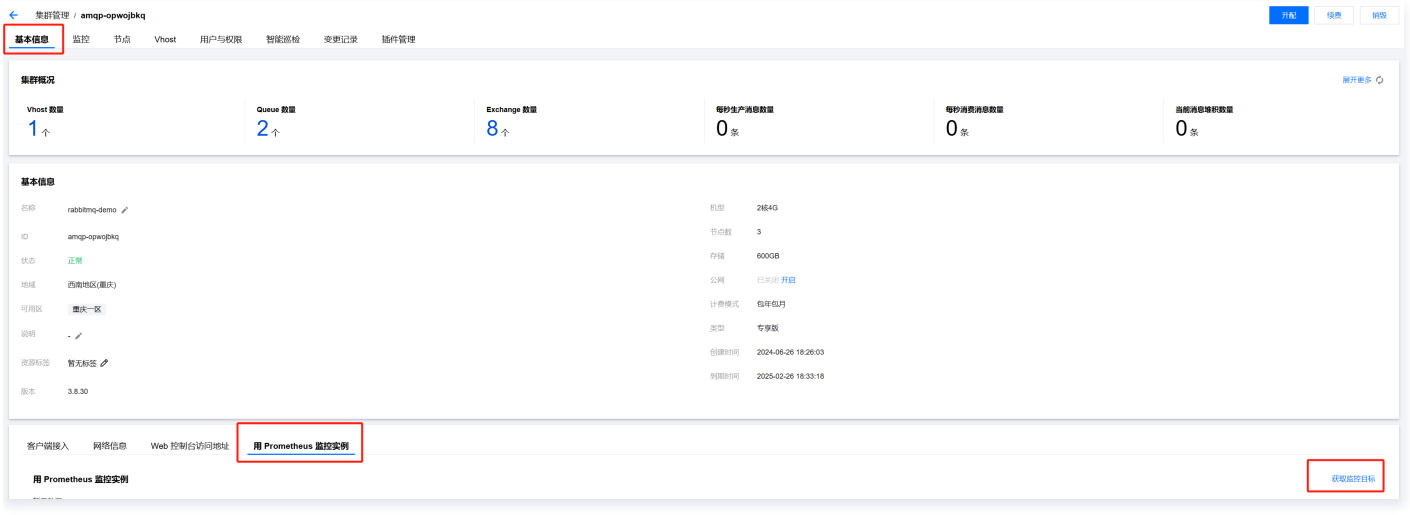
```
rabbitmq-plugins enable rabbitmq_prometheus
```

查看方式：

```
curl localhost:15692/metrics  # 默认开放为15692端口
```

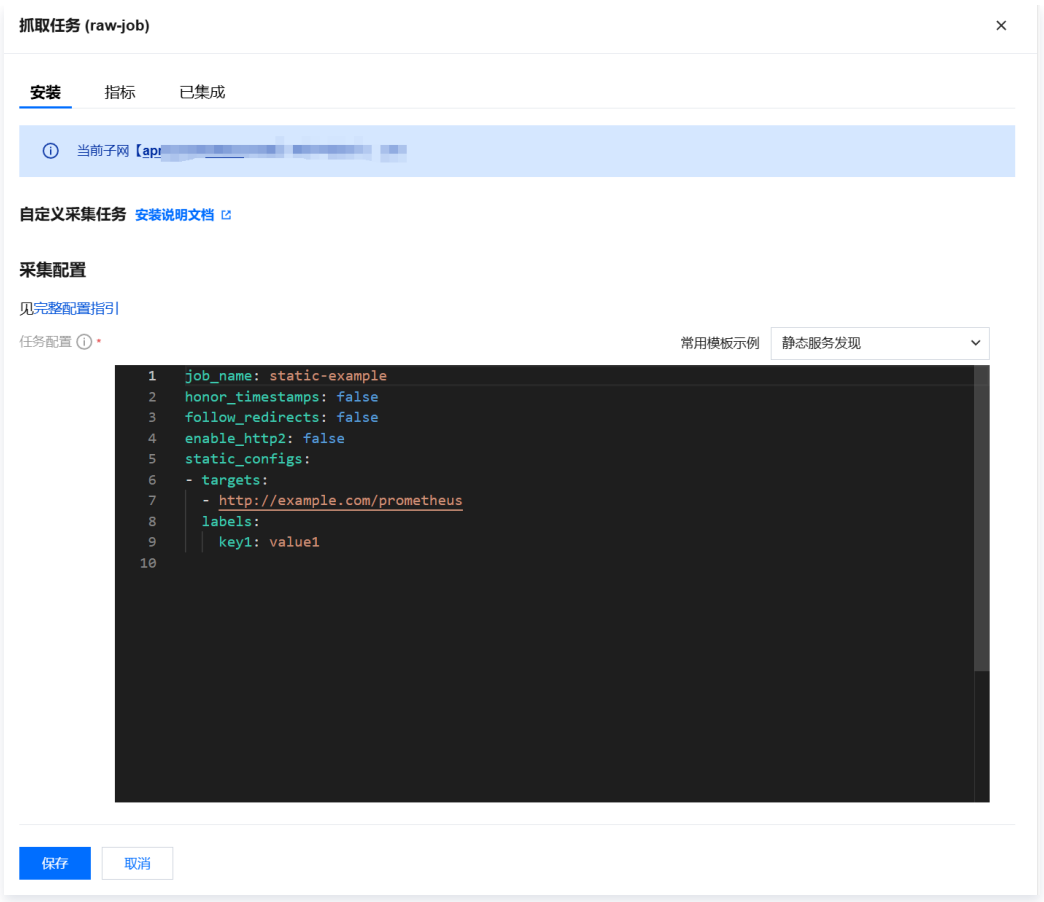
#### 腾讯云消息队列 RabbitMQ 版开启

在 腾讯云消息队列 RabbitMQ 版 中找到自己的集群，单击进入集群，在**基本信息 > 用 Prometheus 监控实例**下单击**获取监控目标**，即可得到拉取监控数据地址。

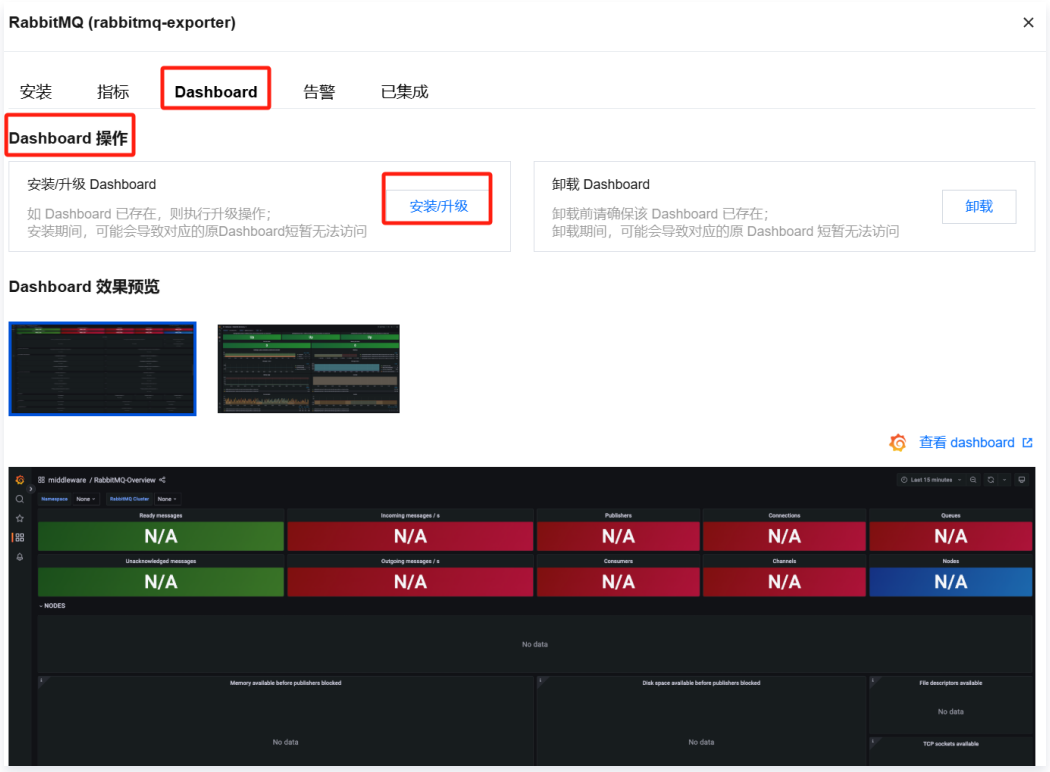



## 指标采集

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击**抓取任务**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。




5. 在集成中心找到并单击 RabbitMQ，选择 Dashboard > Dashboard 操作，然后单击安装/升级。

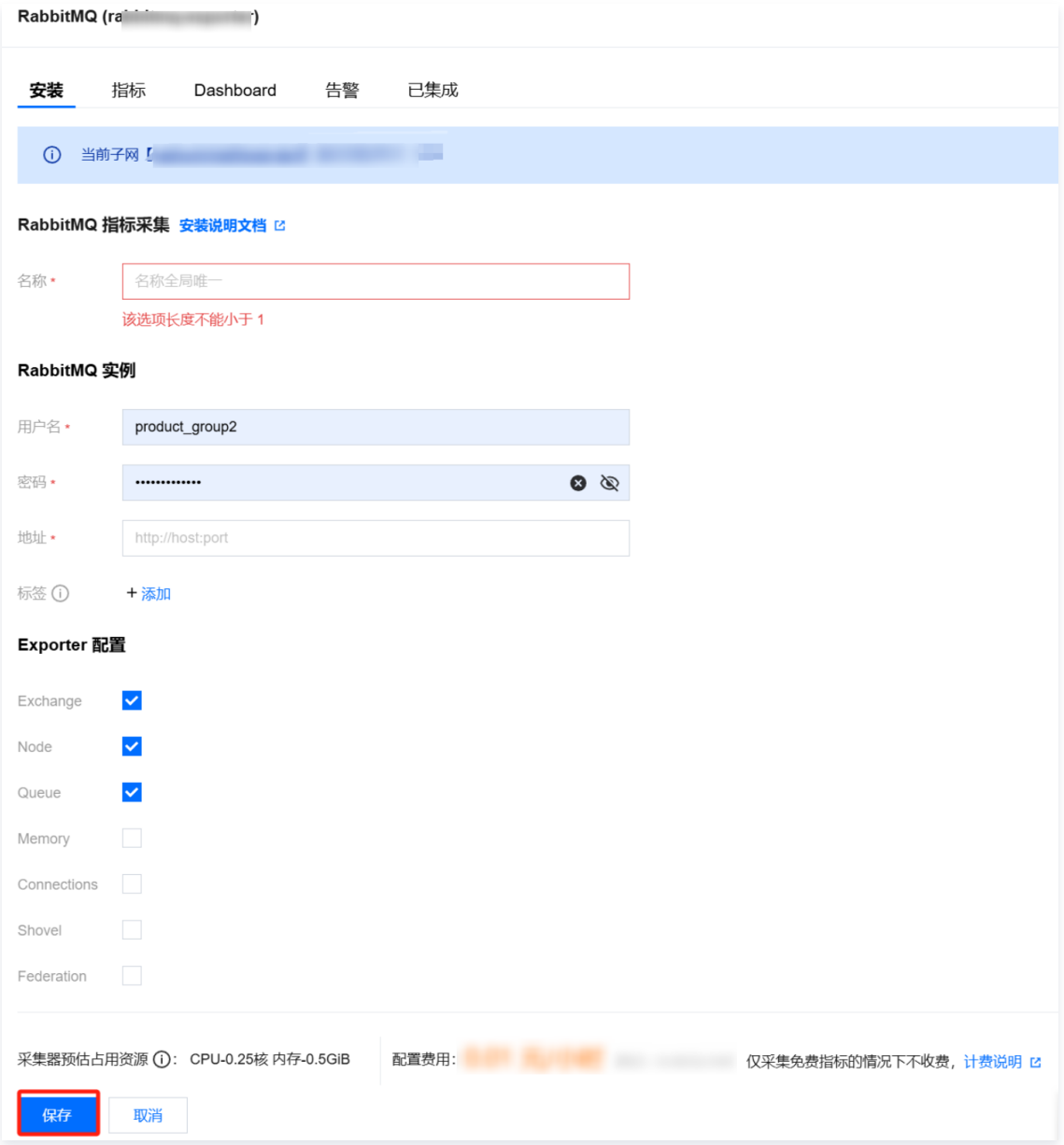



# 接入方式：开源 Exporter 导出

## 方式一：一键安装(推荐)

### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **RabbitMQ**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。




**配置说明**

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | RabbitMQ 用户名称。 |
| 密码 | RabbitMQ 用户密码。 |
| 地址 | RabbitMQ web 访问地址。 |
| 标签 | 给指标添加自定义 Label。 |
| Exporter 配置 | 需要采集的信息选项，包括 Exchange、Node、Queue、Memory、Connections、Shovel、Federation。 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台，选择对应的 Prometheus 实例，选择**数据采集** > **集成容器服务**，然后找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

### 操作步骤

#### 步骤1：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 RabbitMQ Exporter > 验证 步骤完成 Exporter 部署。

#### 步骤2：部署 RabbitMQ Exporter

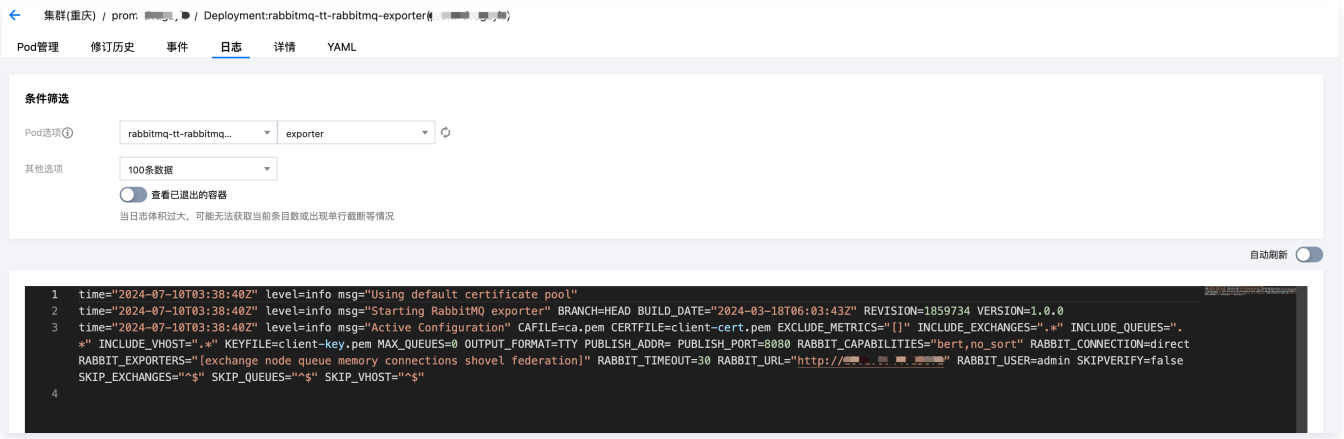
在 Deployment 管理页面，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
  name: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
  namespace: rabbitmq-demo # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
  template:
    metadata:
      labels:
        k8s-app: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
    spec:
      containers:
      - env:
        - name: RABBIT_URL
          value: http://10.1.2.3:15672  # 根据业务需要调整成对应的地址
        - name: RABBIT_USER
          value: test    # 根据业务进行调整
        - name: RABBIT_PASSWORD
          value: abc123  # 根据业务进行调整
        - name: RABBIT_EXPORTERS
          value: exchange,node,queue,memory,connections,shovel,federation  # 根据业务进行调整
        - name: PUBLISH_PORT
          value: "8080"
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:rabbitmq-exporter-1.0.0
        imagePullPolicy: IfNotPresent
        name: rabbitmq-exporter
        ports:
        - containerPort: 8080
          name: metric-port
        terminationMessagePath: /dev/termination-log
```

```
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```

**步骤3：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 RabbitMQ 指标：

```
curl localhost:8080/metrics
```

执行结果如下图所示：

```
rabbitmq_queue_messages_published_total{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_ram Total number of messages which are resident in ram.
# TYPE rabbitmq_queue_messages_ram gauge
rabbitmq_queue_messages_ram{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_ram{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_ready Number of messages ready to be delivered to clients.
# TYPE rabbitmq_queue_messages_ready gauge
rabbitmq_queue_messages_ready{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_ready{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_ready_global Number of messages ready to be delivered to clients.
# TYPE rabbitmq_queue_messages_ready_global gauge
rabbitmq_queue_messages_ready_global{cluster="amqp-opwojbkq"} 0
# HELP rabbitmq_queue_messages_ready_ram Number of messages from messages_ready which are resident in ram.
# TYPE rabbitmq_queue_messages_ready_ram gauge
rabbitmq_queue_messages_ready_ram{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_ready_ram{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_redelivered_total Count of subset of messages in deliver_get which had the redelivered flag set.
# TYPE rabbitmq_queue_messages_redelivered_total counter
rabbitmq_queue_messages_redelivered_total{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_redelivered_total{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_returned_total Count of messages returned to publisher as unroutable.
# TYPE rabbitmq_queue_messages_returned_total counter
rabbitmq_queue_messages_returned_total{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_returned_total{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_unacknowledged Number of messages delivered to clients but not yet acknowledged.
# TYPE rabbitmq_queue_messages_unacknowledged gauge
rabbitmq_queue_messages_unacknowledged{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_unacknowledged{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_unacknowledged_global Number of messages delivered to clients but not yet acknowledged.
# TYPE rabbitmq_queue_messages_unacknowledged_global gauge
rabbitmq_queue_messages_unacknowledged_global{cluster="amqp-opwojbkq"} 0
# HELP rabbitmq_queue_messages_unacknowledged_ram Number of messages from messages_unacknowledged which are resident in ram.
```

**步骤4：添加采集任务**

1. 登录 Prometheus 监控控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。

3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: rabbitmq-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # 填写 pod yaml 中 Prometheus Exporter 对应的 Port 的 Name
    path: /metrics  # 填写 Prometheus Exporter 对应的 Path 的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 RabbitMQ 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 RabbitMQ 实例 IP
  namespaceSelector:    # 选择要监控 pod 所在的 namespace
    matchNames:
    - rabbitmq-demo
  selector:  # 填写要监控 pod 的 Label 值，以定位目标 pod
    matchLabels:
      k8s-app: rabbitmq-exporter
```

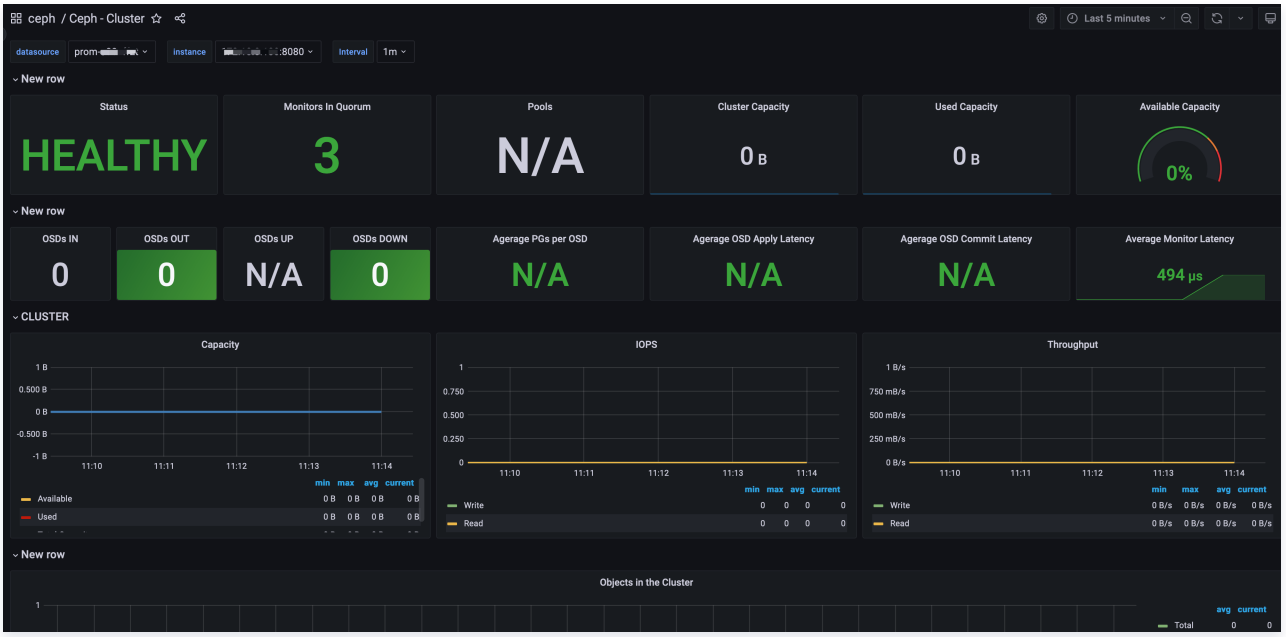
## 查看监控

### 前提条件

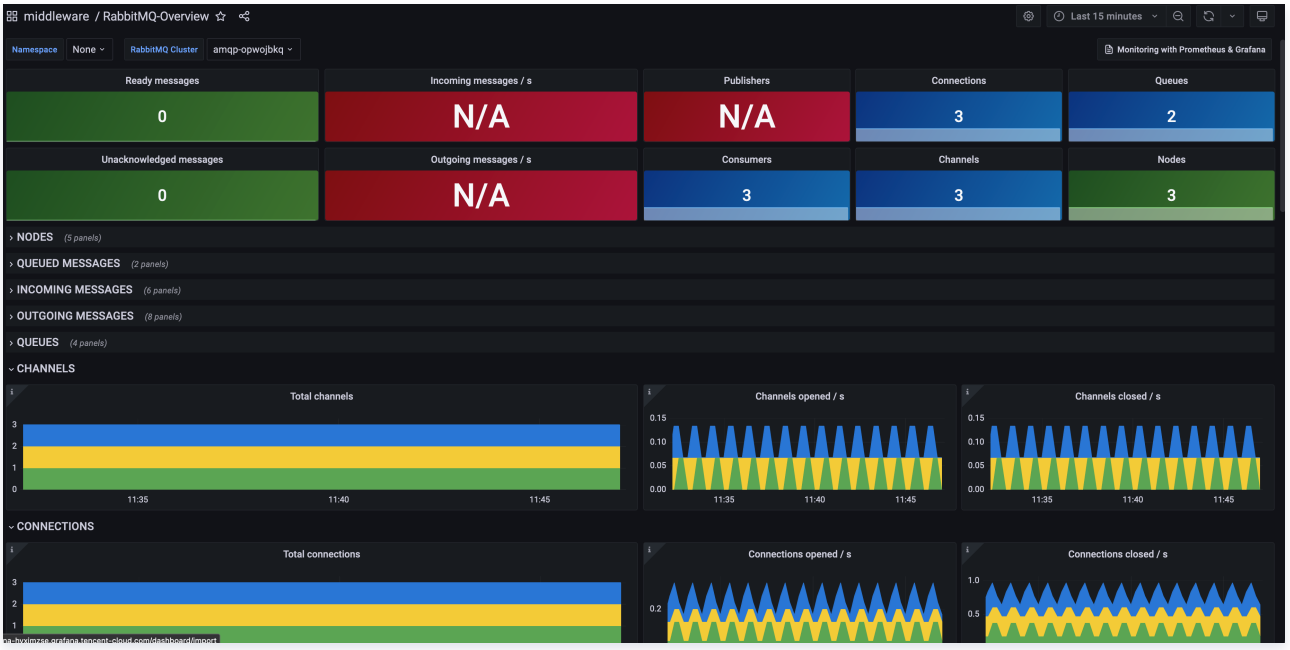
Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 Grafana 地址，打开并登录，然后在 middleware 文件夹中找到 RabbitMQ Monitoring 监控面板，查看实例相关监控数据，如下图所示：




3. RabbitMQ 内置监控可在 middleware 文件夹下的 RabbitMQ-Overview 面板中查看。




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

## 附录：RabbitMQ Exporter 环境变量配置

| 名称 | 默认 | 描述 |
| --- | --- | --- |
| RABBIT_URL | http://127.0.0.1:15672 | RabbitMQ 管理插件 URL。 |
| RABBIT_USER | guest | RabbitMQ 管理插件的用户名。 |
| RABBIT_PASSWORD | guest | RabbitMQ 管理插件的密码。 |
| RABBIT_CONNECTION | direct | RabbitMQ 连接方式，direct 或者 loadbalancer。 |
| RABBIT_USER_FILE | - | 带有用户名的文件位置（对 docker secrets 有用）。 |

| | | |
|---|---|---|
| RABBIT_PASSWORD_FILE | - | 带有密码的文件的位置（对 docker secrets 有用）。 |
| PUBLISH_PORT | 9419 | Exporter 导出指标端口，默认9419。 |
| PUBLISH_ADDR | "" | Exporter 导出指标监听地址，默认为""。 |
| OUTPUT_FORMAT | TTY | 日志输出格式，支持TTY和JSON。 |
| LOG_LEVEL | info | 日志级别。可能的值：“debug”、“info”、“warning”、“error”、“fatal”或“panic”，默认info。 |
| CAFILE | ca.pem | 访问管理插件的根证书路径。如果使用自签名证书则需要。如果文件不存在将被忽略。 |
| CERTFILE | client-cert.pem | 用于验证导出器真实性的客户端证书路径。如果文件不存在，将被忽略。 |
| KEYFILE | client-key.pem | 与证书一起使用以验证导出者真实性的私钥路径。如果文件不存在，将被忽略。 |
| SKIPVERIFY | false | true/0 将忽略管理插件的证书错误。 |
| SKIP_VHOST | ^$ | 正则表达式，匹配的虚拟主机名不会被导出。首先执行 INCLUDE_VHOST，然后执行 SKIP_VHOST。适用于队列和交换器。 |
| INCLUDE_VHOST | .* | 正则表达式虚拟主机过滤器。仅导出匹配的虚拟主机。适用于队列和交换机。 |
| INCLUDE_QUEUES | .* | 正则表达式队列过滤器。仅导出匹配的名称。 |
| SKIP_QUEUES | ^$ | 正则表达式，匹配的队列名称不会被导出（对于短暂的 rpc 队列很有用）。首先执行 INCLUDE，然后执行 SKIP。 |
| INCLUDE_EXCHANGES | .* | 正则表达式交换过滤器。（仅导出匹配的虚拟主机中的交换）。 |
| SKIP_EXCHANGES | ^$ | 正则表达式，匹配的交易所名称不会被导出。首先执行 INCLUDE，然后执行 SKIP。 |
| RABBIT_CAPABILITIES | bert,no_sort | 目标 RabbitMQ 服务器支持的扩展抓取功能的逗号分隔列表。 |
| RABBIT_EXPORTERS | exchange,node,queue | 已启用模块列表。可能的模块：连接、铲子、联合、交换、节点、队列、内存。 |
| RABBIT_TIMEOUT | 30 | 从管理插件检索数据的超时时间（秒）。 |
| MAX_QUEUES | 0 | 删除指标之前的最大队列数（如果设置为0则禁用）。 |
| EXCLUDE_METRICS | - | 要从导出中排除的指标名称。以逗号分隔。 |

# Fluentd/FluentBit 接入

最近更新时间：2025-02-26 16:14:02

Fluentd 是一个开源的数据收集器，旨在简化日志管理和分析的过程。它提供了一个统一的日志层，能够收集、转换并将数据发送到多种目的地。Fluentd 的设计哲学是提供简单、可靠和灵活的日志管理解决方案，使得它能够轻松地与各种数据源和最终存储系统集成。
Fluentd 的核心特点包括其插件架构，这使得它能够通过安装不同的插件来支持广泛的输入/输出源，包括文件、日志、数据库和 HTTP 源等。此外，Fluentd 还支持数据的实时处理，包括过滤、修改和丰富数据，以满足特定的日志管理需求。
使用 Fluentd 可以帮助组织减少日志管理的复杂性，提高数据处理的效率，便于日志数据的分析和监控。无论是在云环境还是在本地部署，Fluentd 都能提供灵活的日志管理解决方案，帮助企业更好地理解和利用他们的数据。
Fluentd 和 Fluent Bit 都可以用作聚合器或转发器，它们可以相互补充或用作独立解决方案。近年来，出于性能和兼容性原因，云提供商从 Fluentd 转向 Fluent Bit。Fluent Bit 现在被认为是下一代解决方案。
本文将为您介绍各类部署方式部署的 Fluentd 组件如何暴露 Prometheus 指标数据，并通过腾讯云可观测平台-Prometheus监控服务来采集指标并展示数据。Prometheus 监控服务支持提供预设的 Fluentd 监控面板，以方便您使用。

## Fluentd

### Fluentd 部署方式

Fluentd 在云环境下部署有很多方式，本文将介绍基于以下四种部署方式进行监控接入：

- 使用 Fluentd 镜像自定义部署
- 使用 Fluentd Daemonset 镜像进行部署
- 使用 Fluentd Operator 进行管理 Fluentd
- 使用 Logging Operator 部署管理 Fluentd

### Fluentd 镜像自定义部署

#### 部署流程

1. 获取或者自制安装符合业务使用插件的 Fluentd 镜像。
2. 创建 Fluentd 配置，进行输入、过滤、输出等插件的配置。
3. 使用 Fluentd 配置和镜像创建 Fluentd 负载。
4. 查看 Fluentd 运行情况。

#### 指标导出

##### 方式一：Prometheus 插件导出（推荐）

1. Fluentd 官方镜像是没有 Prometheus 插件的，需要安装，您可以使用以下 Dockerfile，来输出包含 Prometheus 插件 的 Fluentd 镜像。

```
FROM fluent/fluentd:latest

RUN fluent-gem install fluent-plugin-prometheus
## 对于 td-agent：
## RUN td-agent-gem install fluent-plugin-prometheus
```

若您已经有 DockerFile，则在其中加入安装 fluent-plugin-prometheus 插件命令即可。

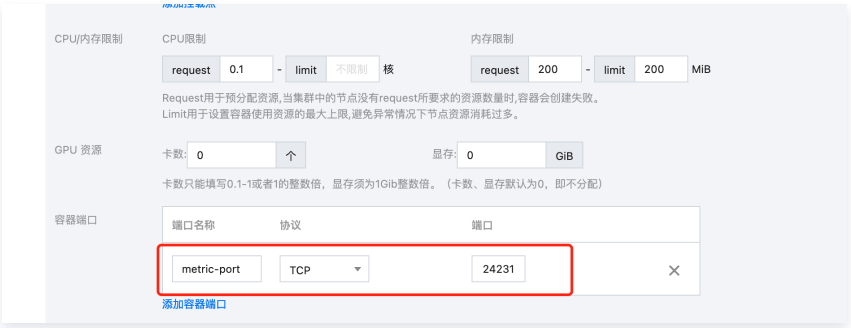
```
RUN fluent-gem install fluent-plugin-prometheus
## 对于 td-agent：
## RUN td-agent-gem install fluent-plugin-prometheus
```

2. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluentd 配置，在其中添加 prometheus 相关输入插件，导出 prometheus 指标。

```
# expose metrics in prometheus format

<source>
```

```
  @type prometheus
  bind 0.0.0.0
  port 24231
  metrics_path /metrics
</source>

<source>
  @type prometheus_output_monitor
  interval 10
  @id in_prometheus_output_monitor
</source>
```

3. 在**工作负载**中找到您的 Fluentd 负载，更新镜像为第一步中输出的镜像，然后再给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。




**方式二：Exporter 导出**

1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluentd 配置，在其中添加 monitor-agent 相关输入插件，导出监控数据。

```
<source>
    @type monitor_agent
    bind 0.0.0.0
    port 24220
</source>
```

2. 部署 fluentd-exporter 导出 prometheus 指标。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/instance: fluentd-demo # 根据业务需要调整成对应的名称，建议加上 Fluentd 实例的信息
    app.kubernetes.io/name: fluentd-exporter
  name: fluentd-demo-fluentd-exporter  # 根据业务需要调整成对应的名称，建议加上 Fluentd 实例的信息，
  namespace: cm-prometheus  # 根据业务调整命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/instance: fluentd-test
      app.kubernetes.io/name: fluentd-exporter
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
```

```
        app.kubernetes.io/instance: fluentd-demo # 根据业务需要调整成对应的名称，建议加上 fluentd 实例的信息
        app.kubernetes.io/job: fluentd-demo
        app.kubernetes.io/name: fluentd-exporter
    spec:
      containers:
      - args:
        - --telemetry.address=:8080  ## 指标开放到 8080端口
        - --scrape_uri=http://172.0.0.1:24220/api/plugins.json  ## 地址使用 Fluentd pod 实例 IP
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:fluentd_exporter-latest
        imagePullPolicy: IfNotPresent
        name: fluentd-exporter
        ports:
        - containerPort: 8080
          name: metric-port  # 这个名称在配置抓取任务的时候需要
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

## Fluentd Daemonset 镜像部署

### 部署流程

根据实际业务使用相应的 fluentd-k8s-daemonset，下面以输出到 elasticsearch 的 fluentd-daemonset-elasticsearch 为例，介绍其部署流程：

1. 在 fluentd-kubernetes-daemonset 项目的 配置生成模板目录 下找到配置文件的 erb 文件，根据业务进行配置的调整，然后生成各配置文件。或者使用 Docker 镜像目录 中已经生成出来的配置，根据业务调整得到最终配置。
2. 使用 fluentd-daemonset-elasticsearch.yaml（根据实际业务调整 host、端口、鉴权等信息）部署输出到 elasticsearch 的 fluentd。
3. 查看 fluentd 运行情况。

### 指标导出

Fluentd Daemonset 自带 Prometheus 插件，同时默认配置已进行 Prometheus 指标导出。

```
root@fluentd-94ttz:/fluentd/etc# cat fluent.conf

# AUTOMATICALLY GENERATED
# DO NOT EDIT THIS FILE DIRECTLY, USE /templates/conf/fluent.conf.erb

@include "#{ENV['FLUENTD_SYSTEMD_CONF'] || 'systemd'}.conf"
@include "#{ENV['FLUENTD_PROMETHEUS_CONF'] || 'prometheus'}.conf"
@include kubernetes.conf
@include conf.d/*.conf

<match **>
   @type elasticsearch
   @id out_es
   @log_level info
```

prometheus.conf 默认导出配置。

```
# AUTOMATICALLY GENERATED
# DO NOT EDIT THIS FILE DIRECTLY, USE /templates/conf/prometheus.conf.erb

# Prometheus metric exposed on 0.0.0.0:24231/metrics
```

```
<source>
  @type prometheus
  @id in_prometheus
  bind "#{ENV['FLUENTD_PROMETHEUS_BIND'] || '0.0.0.0'}"
  port "#{ENV['FLUENTD_PROMETHEUS_PORT'] || '24231'}"
  metrics_path "#{ENV['FLUENTD_PROMETHEUS_PATH'] || '/metrics'}"
</source>

<source>
  @type prometheus_output_monitor
  @id in_prometheus_output_monitor
</source>
```

## Fluentd Operator 部署管理

### 部署流程

1. 使用 Fluentd Operator 管理部署 Fluentd 之前需要先部署 Fluent Operator。部署方式有两种：一种为 yaml 部署，另一种为 helm 部署，详情请参见 开始安装 。
2. 创建 ClusterInput、ClusterFilter、ClusterOutput CR 进行输入、输出、过滤配置。
3. 创建 ClusterFluentdConfig CR，将上述组件关联进来。
4. 挂载 ClusterFluentdConfig 创建 Fluentd CR，Operator 会根据 CR 创建出 Fluentd。
5. 查看 Fluentd 运行情况。

### Fluentd 指标导出

由于目前 Fluentd Operator 只支持 Fluent Bit Prometheus 指标的导出，Fluentd 暂不支持，故而 Fluentd 需使用 Fluentd 镜像自定义部署指标导出 。

## Logging Operator 部署管理

### 部署流程

1. 使用 Logging Operator 管理部署 Fluentd 之前需要先部署 Logging Operator，详情请参见 示例 。
2. 创建 FluentbitAgent CR，Operator 会根据该 CR 进行 Fluentd 部署。
3. 创建 Flow、Output CR，进行 Fluentd 配置。
4. 查看 Fluentd 运行情况。

### 指标导出

Logging Operator 与 Prometheus 有良好的适配性，在 Logging CR 添加如下配置即可导出指标。

```
spec:
  fluentdSpec:
    metrics:
      serviceMonitor: true
  fluentbitSpec:
    metrics:
      serviceMonitor: true
```

导出之后配置 ServiceMonitor 即可完成指标采集。

## Fluent Bit

Fluent Bit 官方镜像本身就支持导出 Prometheus 指标能力，只需要按需开放即可。

### 镜像部署指标导出

### 方法一：HTTP Server 导出

1. 登录 腾讯云容器服务 ，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluent Bit 配置，在其中添加如下配置：

```
[SERVICE]
    HTTP_Server  On     # 若存在 HTTP_Server，但状态为Off，需改为On
    HTTP_Listen  0.0.0.0
    HTTP_PORT    2020   # 端口可自行调整

[INPUT]
    Name cpu

[OUTPUT]
    Name   stdout
    Match *
```

2. Prometheus 指标导出验证。

```
curl -s http://172.0.0.1:2020/api/v2/metrics/prometheus
```

指标输出：

```
# TYPE fluentbit_input_storage_chunks_down gauge
fluentbit_input_storage_chunks_down{name="fluentbit_metrics.1"} 0
# HELP fluentbit_input_storage_chunks_busy Total number of chunks in a busy state.
# TYPE fluentbit_input_storage_chunks_busy gauge
fluentbit_input_storage_chunks_busy{name="fluentbit_metrics.1"} 0
# HELP fluentbit_input_storage_chunks_busy_bytes Total number of bytes used by chunks in a busy state.
# TYPE fluentbit_input_storage_chunks_busy_bytes gauge
fluentbit_input_storage_chunks_busy_bytes{name="fluentbit_metrics.1"} 0
# HELP fluentbit_output_upstream_total_connections Total Connection count.
# TYPE fluentbit_output_upstream_total_connections gauge
fluentbit_output_upstream_total_connections{name="stdout.0"} 0
# HELP fluentbit_output_upstream_busy_connections Busy Connection count.
# TYPE fluentbit_output_upstream_busy_connections gauge
fluentbit_output_upstream_busy_connections{name="stdout.0"} 0
# HELP fluentbit_output_chunk_available_capacity_percent Available chunk capacity (percent)
# TYPE fluentbit_output_chunk_available_capacity_percent gauge
fluentbit_output_chunk_available_capacity_percent{name="stdout.0"} 100
# HELP fluentbit_output_upstream_total_connections Total Connection count.
# TYPE fluentbit_output_upstream_total_connections gauge
fluentbit_output_upstream_total_connections{name="prometheus_exporter.1"} 0
# HELP fluentbit_output_upstream_busy_connections Busy Connection count.
# TYPE fluentbit_output_upstream_busy_connections gauge
```

3. 在**工作负载**中找到您的 Fluent Bit 负载，给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。


## 方法二：Fluent Bit Metrics 导出

1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluent Bit 配置，在其中添加如下配置：

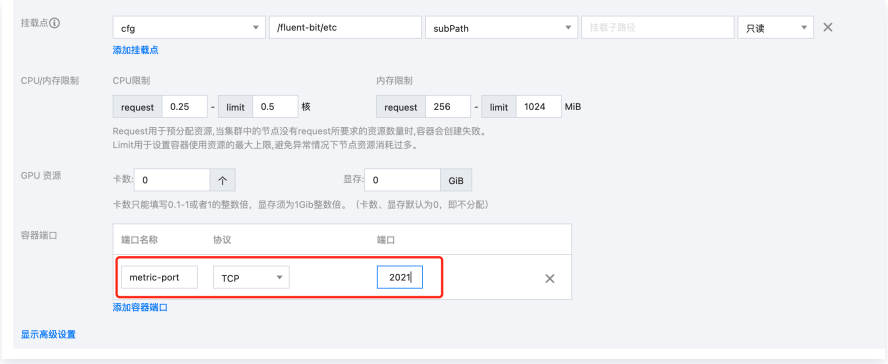
```
[SERVICE]
    flush           1
    log_level       info

[INPUT]
    name            fluentbit_metrics
    tag             internal_metrics
    scrape_interval 2
```

```
[OUTPUT]
    name            prometheus_exporter
    match           internal_metrics
    host            0.0.0.0
    port            2021
```

2. Prometheus 指标导出验证。

```
curl -s http://172.0.0.:2021/metrics
```

3. 在**工作负载**中找到您的 Fluent Bit 负载，给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。




## Fluentd Operator 部署指标导出

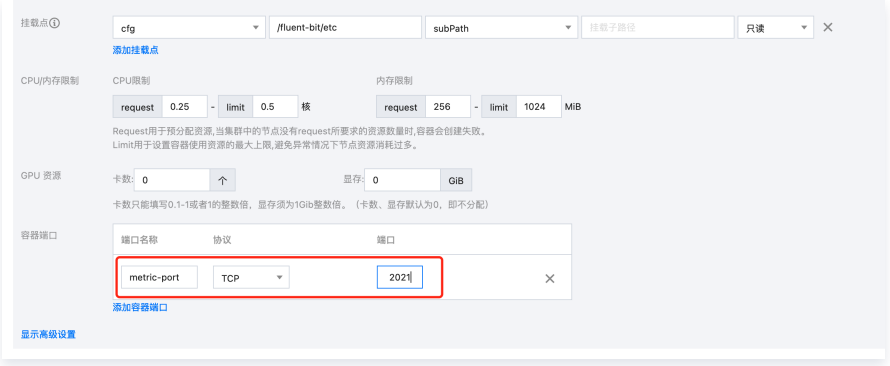
1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，添加如下 ClusterInput CR：

```
apiVersion: fluentbit.fluent.io/v1alpha2
kind: ClusterInput
metadata:
  name: fluentbit_metrics
  labels:
    fluentbit.fluent.io/enabled: "true"
    fluentbit.fluent.io/mode: "fluentbit"
spec:
  tag: internal_metrics
  scrape_interval: 2
  scrape_on_start: true
```

添加如下 ClusterOutput CR：

```
apiVersion: fluentbit.fluent.io/v1alpha2
kind: ClusterOutput
metadata:
  name: prometheus-exporter
  labels:
    fluentbit.fluent.io/enabled: "true"
    fluentbit.fluent.io/mode: "fluentbit"
spec:
  match: internal_metrics
  metricsExporter:
    host: "0.0.0.0"
    port: 2021
    addLabels:
      app: "fluentbit"
```

2. 将上述 CR Selector 到 ClusterFluentdConfig CR 中。
3. 在**工作负载**中找到您的 Fluent Bit 负载，给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。




## Logging Operator 部署指标导出

Logging Operator 与 Prometheus 有良好的适配性，在 Logging CR 添加如下配置即可导出指标。

```
spec:
  fluentdSpec:
    metrics:
      serviceMonitor: true
  fluentbitSpec:
    metrics:
      serviceMonitor: true
```

导出之后配置 ServiceMonitor 即可完成指标采集。

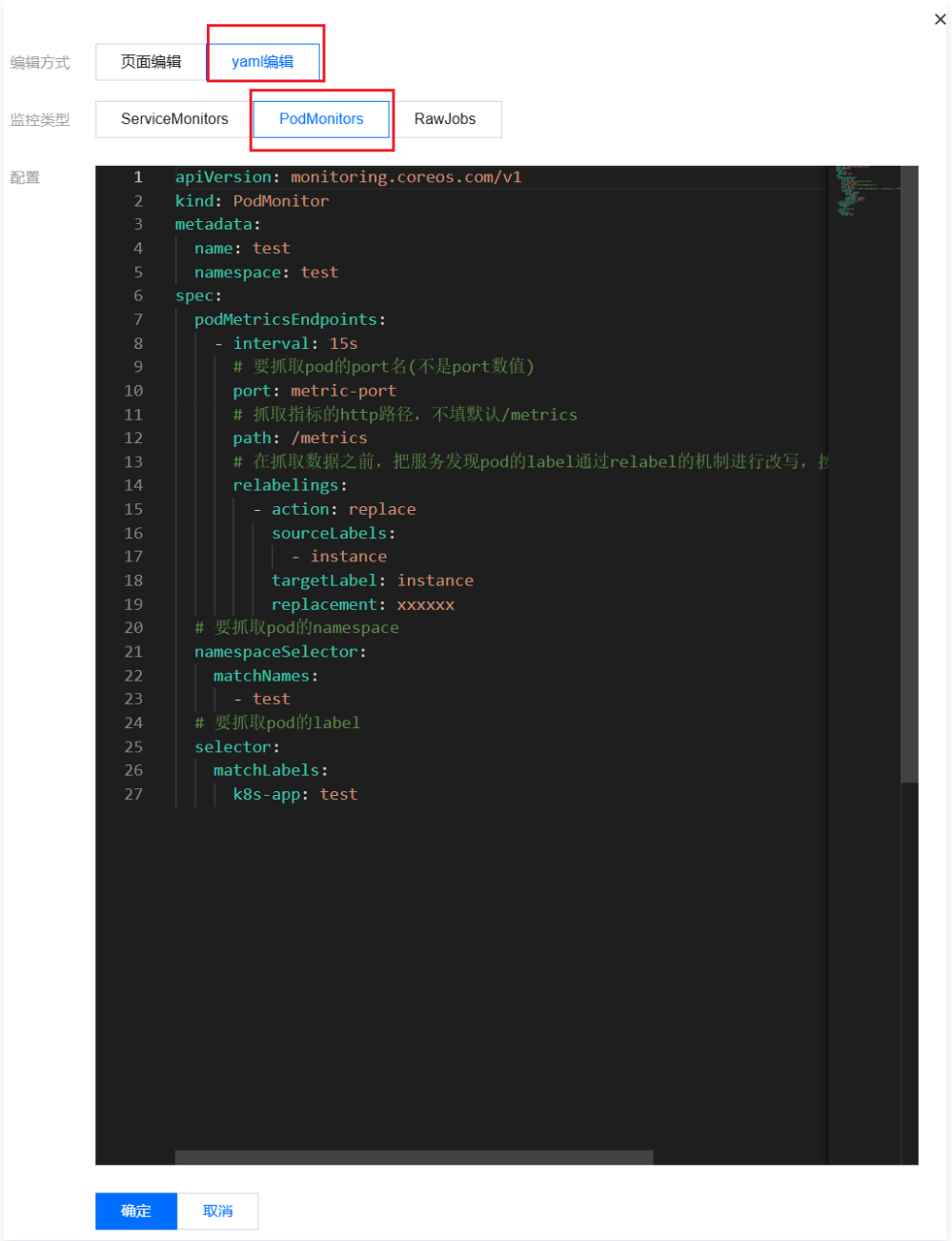
# 接入 Prometheus 监控

## Podmonitor 采集（推荐）

在 Fluentd 存在动态扩缩容场景下，为了能够动态服务发现，需要使用 Podmonitor 来进行指标采集：

1. 登录 腾讯云可观测平台 > Prometheus 控制台，单击新建的**实例 ID/名称**。
2. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：


3. 进入数据采集配置页面后，单击**新建自定义监控**，在弹出的页面中选择 **yaml编辑**，监控类型选择 **PodMonitors**，如下图所示：




4. **通过此方式添加 `Pod Monitor`，目前支持基于 `Labels` 发现对应的目标实例地址，因此可以对一些负载添加特定的 `K8S Labels`，可以使 `Labels` 下负载的 Pod 都会被 Prometheus 服务自动识别出来，不需要再为每个服务添加采集任务，以上面的例子配置信息如下：**

> **说明：**
> `port` 的取值为 `pod yaml` 配置文件里的 `spec/ports/name` 对应的值。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: fluentd-demo     # 填写一个唯一名称
  namespace: cm-prometheus   # 根据业务调整命名空间
spec:
  podMetricsEndpoints:
  - interval: 15s
    # 要抓取pod的port名（不是port数值）
    port: metric-port
    # 抓取指标的http路径，不填默认/metrics，Fluent Bit 方式一为 /api/v2/metrics/prometheus
```

```
    path: /metrics
    # 在抓取数据之前，把服务发现pod的label通过relabel的机制进行改写，按顺序执行多个relabel规则
    # 根据业务实际使用进行配置，也可以不进行relabel配置
    relabelings:
      - action: replace
        sourceLabels:
          - instance
        targetLabel: instance
        replacement: xxxxxx
  # 选择要监控pod所在的namespace
  namespaceSelector:
    matchNames:
    - fluentd-demo
  # 填写要监控pod的Label值，以定位目标pod
  selector:
    matchLabels:
      app: fluentd-demo
```

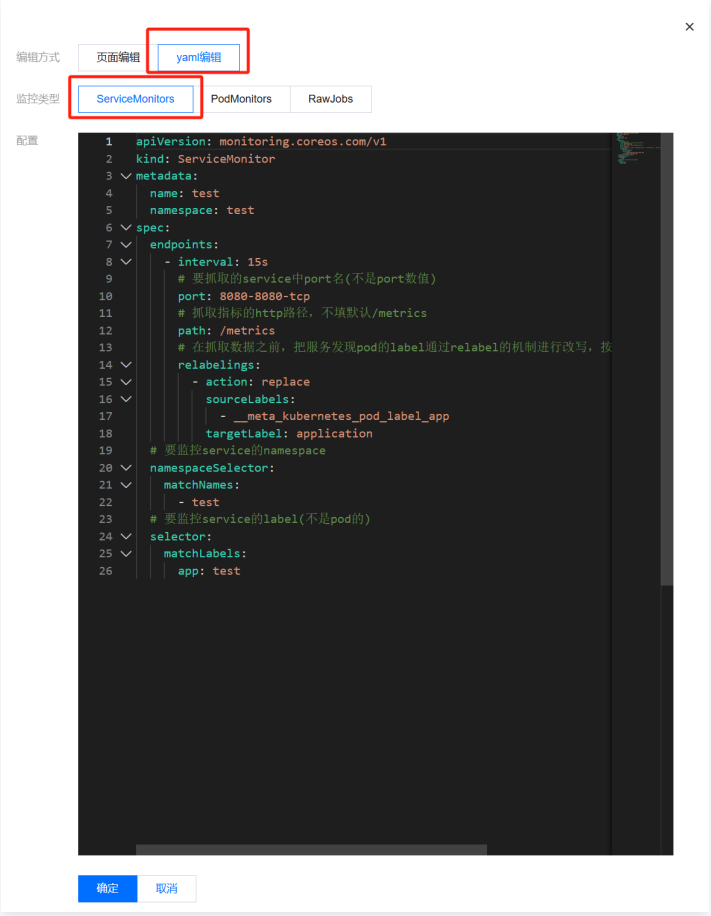
> **注意：**
> 示例中名称为 application 的 Label 必须配置，否则无法使用我们提供一些其它的开箱即用的集成功能。更多高阶用法请参见 ServiceMonitor 或 PodMonitor 。

## ServiceMonitor 采集（Logging Operator 使用）

Logging Operator 部署 Fluent Bit 场景中，指标导出是使用 ServiceMonitor 的方式，需要使用 ServiceMonitor 进行指标采集：

1. 登录 腾讯云可观测平台 > Prometheus 控制台，单击新建的**实例 ID/名称**。
2. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：


3. 进入数据采集配置页面后，单击**新建自定义监控**，在弹出的页面中选择 **yaml 编辑**，监控类型选择 **ServiceMonitors**，如下图所示：




4. 通过此方式添加 `Service Monitor`，目前支持基于 `Labels` 发现对应的目标实例地址，因此可以对一些服务添加特定的 `K8S Labels`，可以使 `Labels` 下的服务都会被 Prometheus 服务自动识别出来，不需要再为每个服务添加采集任务，以上面的例子配置信息如下：

> **说明：**
> `port` 的取值为 `service yaml` 配置文件里的 `spec/ports/name` 对应的值。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: fluent-bit-demo     # 填写一个唯一名称
  namespace: cm-prometheus  # 根据业务调整命名空间
spec:
  endpoints:
  - interval: 30s
    # 填写service yaml中Prometheus Exporter对应的Port的Name
    port: metrics
    # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    path: /metrics
    relabelings:
    # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 app，
    # 我们通过 relabel 的 replace 动作把它替换成了 application
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # 选择要监控service所在的namespace
  namespaceSelector:
    matchNames:
    - fluent-bit-demo
    # 填写要监控service的Label值，以定位目标service
  selector:
```

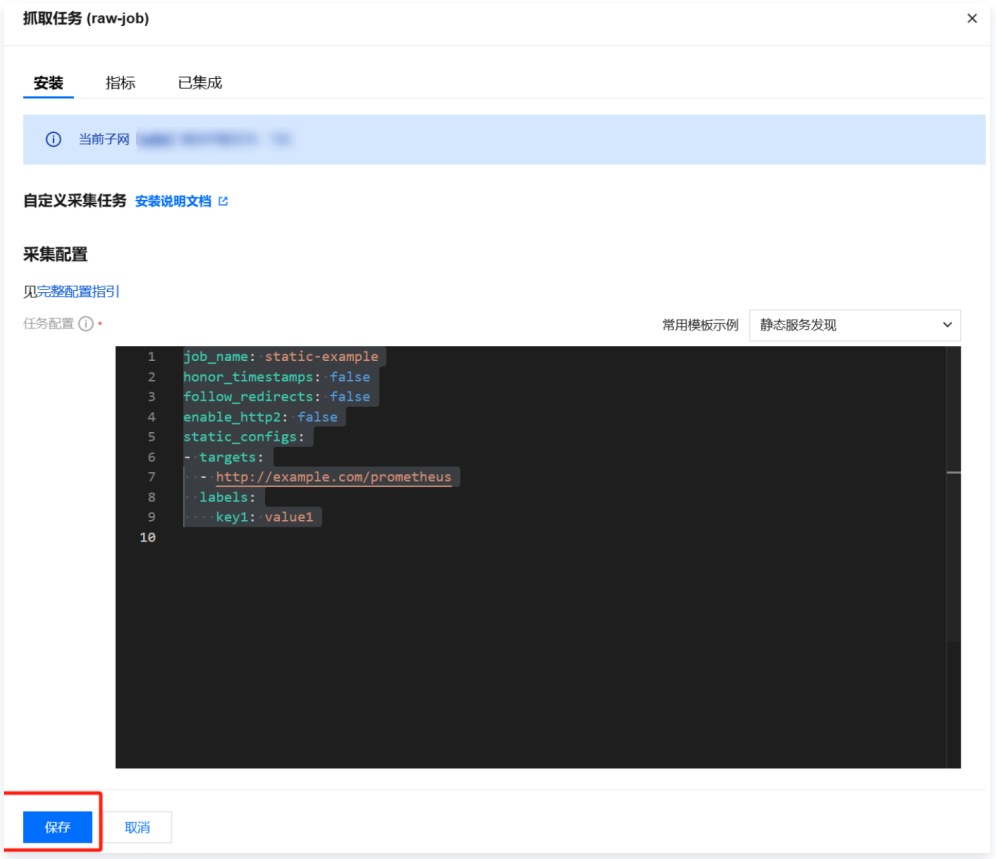
```
    matchLabels:
      app: fluent-bit-app-demo
```

> **注意：**
> 示例中名称为 application 的 Label 必须配置，否则无法使用我们提供一些其它的开箱即用的集成功能。更多高阶用法请参见 ServiceMonitor 或 PodMonitor 。

## Rawjob 采集

Rawjob 采集需要配置 Fluentd 服务 pod ip，仅能在组件稳定为固定的 pod 时使用，动态扩缩容的情况下需要使用 Podmonitor 或者 Servicemonitor 来保证动态的服务发现。Rawjob 采集配置方式：

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击**抓取任务**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。




## 查看监控

1. 在 Prometheus 实例 列表，找到对应的 Prometheus 实例，单击实例ID 右侧图标，打开您的专属 Grafana，输入您的账号密码，即可进行 Grafana 可视化大屏操作区。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到并单击 **Fluentd** 监控，选择 **Dashboard > Dashboard 操作 > Dashboard 安装/升级**，单击**安装/升级**安装对应的 Grafana Dashboard。
3. 进入 Grafana，单击图表，展开监控面板，单击对应的监控图表名称即可查看监控数据。

# MongoDB Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

在使用 MongoDB 过程中需要对 MongoDB 运行状态进行监控，以便了解 MongoDB 服务是否运行正常，排查 MongoDB 故障问题原因， Prometheus 监控服务提供了基于 Exporter 的方式来监控 MongoDB 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 MongoDB Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 MongoDB 是腾讯云 云数据库 MongoDB 版，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **MongoDB**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | MongoDB 的用户名称。 |

| | |
|---|---|
| 密码 | MongoDB 的密码。 |
| 地址 | MongoDB 的连接地址。 |
| 标签 | 给指标添加自定义 Label。 |
| Exporter 配置 | • database：启用数据库指标的收集。<br>• collection：启用集合指标的收集。<br>• topmetrics：启用数据库表头指标信息的收集。<br>• indexusage：启用索引使用统计信息的收集。<br>• connpoolstats：收集 MongoDB 连接池统计信息。 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 MongoDB 连接串 > 部署 MongoDB Exporter > 验证 步骤完成 Exporter 部署。

#### 使用 Secret 管理 MongoDB 连接串

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 MongoDB  Exporter 的时候直接使用 Secret Key，需要调整对应的 URI，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
    name: mongodb-secret-test
    namespace: mongodb-test
type: Opaque
stringData:
    datasource: "mongodb://{user}:{passwd}@{host1}:{port1},{host2}:{port2},{host3}:{port3}/admin"   # 对
应连接URI
```

#### 部署 MongoDB Exporter

在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
```

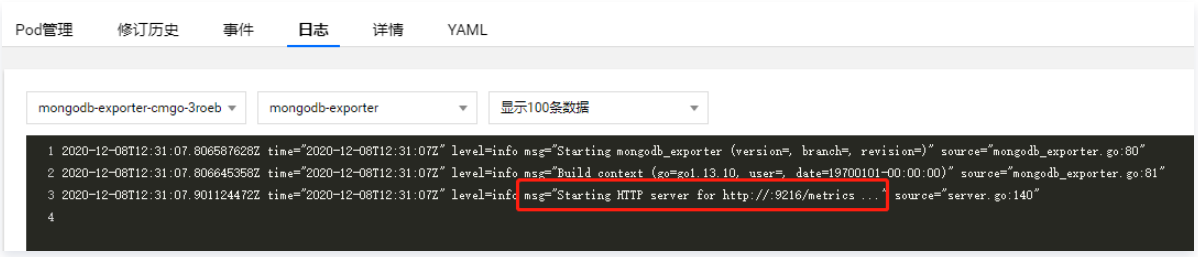
```
    k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
  name: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
  namespace: mongodb-test
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
  template:
    metadata:
      labels:
        k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
    spec:
      containers:
        - args:
            - --collect.database       # 启用采集 Database metrics
            - --collect.collection     # 启用采集 Collection metrics
            - --collect.topmetrics     # 启用采集 table top metrics
            - --collect.indexusage     # 启用采集 per index usage stats
            - --collect.connpoolstats  # 启动采集 MongoDB connpoolstats
          env:
            - name: MONGODB_URI
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret-test
                  key: datasource
          image: ccr.ccs.tencentyun.com/rig-agent/mongodb-exporter:0.10.0
          imagePullPolicy: IfNotPresent
          name: mongodb-exporter
          ports:
            - containerPort: 9216
              name: metric-port  # 这个名称在配置抓取任务的时候需要
          securityContext:
            privileged: false
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
        - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: { }
      terminationGracePeriodSeconds: 30
```

**说明：**
Exporter 详细参数请参见 mongodb_exporter 。

**验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

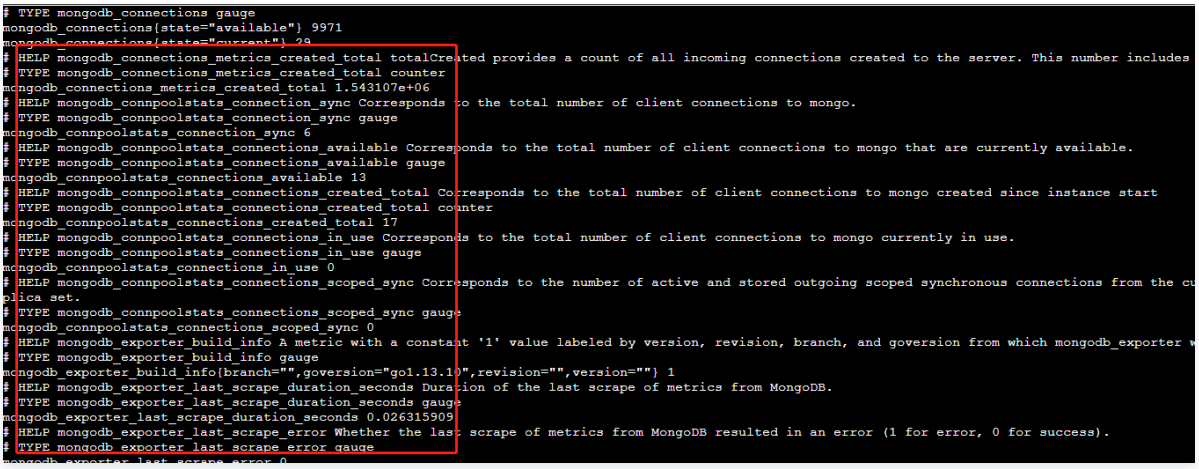
2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：




3. 单击 **Pod 管理**页签，进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，登录 Pod，在命令行中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 MongoDB 指标，若发现未能得到对应的数据，请检查一下连接 URI 是否正确，具体如下：

```
wget 127.0.0.1:9216/metrics
cat metrics
```

命令执行结果如下图所示：




**步骤二：添加采集任务**

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: mongodb-exporter # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port   # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'cmgo-xxxxxxxx' # 调整成对应的 MongoDB 实例 ID
  namespaceSelector:  # 选择要监控pod所在的namespace
    matchNames:
```

```
    - mongodb-test
  selector: # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: mongodb-exporter
```

**说明：**
由于 `Exporter` 和 `MongoDB` 部署在不同的服务器上，因此建议通过 Prometheus Relabel 机制将 MongoDB 实例的信息放到监控指标中，以便定位问题。

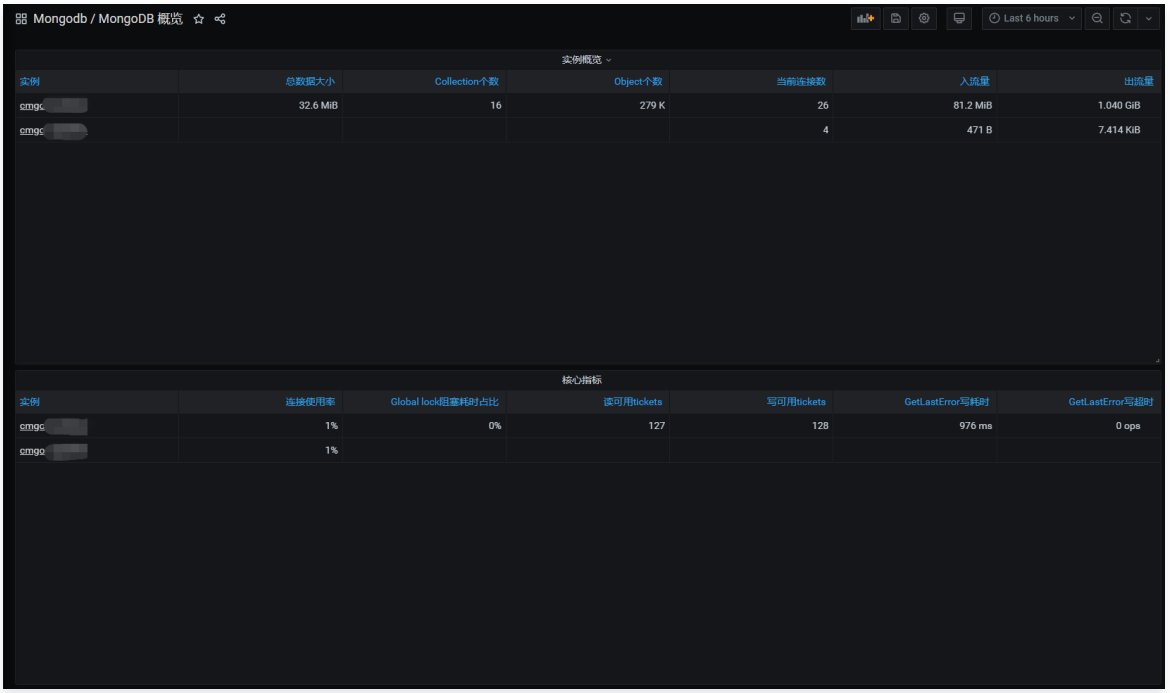
## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，进入集成中心页面。找到 `MongoDB` 监控，安装对应的 Grafana Dashboard 即可开启 `MongoDB` 监控大盘，查看实例相关的监控数据，如下图所示：
   - **MongoDB 概览**：以实例的维度查看实例状态，例如文档个数、连接使用率、读写耗时等，可单击实例跳转到该实例详情。



- **MongoDB 详情**：可以查看某个实例的详细状态，例如元数据概览、核心指标、命令操作、请求流量、读写 Top 等。


> **说明：**
>
> 每个图表可以单击左侧的 进行查看说明。

## 配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击告警策略，可以添加相应的告警策略，详情请参见 新建告警策略 。

### 常见问题

#### 客户端报错：client checkout connect timeout，该如何处理？

可能是连接池使用率达到100%，导致创建连接失败。可以通过 Grafana 大盘 **MongoDB 详情/核心指标/连接使用率**指标排查。


#### 写入不断超时，该如何处理？

需检查 Cache 使用率是否过高、Transactions 可用个数是否为0，可以通过 Grafana 大盘 **MongoDB 详情/核心指标/ WiredTiger Transactions 可用个数| WiredTiger Cache 使用率| GetLastError 写耗时| GetLastError 写超时**指标排查。

# PostgreSQL Exporter 接入

最近更新时间：2025-04-24 18:57:12

## 操作场景

在使用 PostgreSQL 过程中都需要对 PostgreSQL 运行状态进行监控，以便了解 PostgreSQL 服务是否运行正常，排查 PostgreSQL 故障问题原因，Prometheus 监控服务提供了基于 Exporter 的方式来监控 PostgreSQL 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 PostgreSQL Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 PostgreSQL 是腾讯云 云数据库 PostgreSQL，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装（推荐）

#### 操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在**集成中心**找到并单击 **PostgreSQL**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | PostgreSQL 的用户名称。 |
| 密码 | PostgreSQL 的密码。 |

| | |
|---|---|
| 地址 | PostgreSQL 的连接地址。 |
| 标签 | 给指标添加自定义 Label。 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台，选择对应的 Prometheus 实例，选择**数据采集** > **集成容器服务**，然后找到对应容器集群完成关联集群操作。可参见指引 关联集群。

### 操作步骤

#### 步骤1：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中选择**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 PostgreSQL 密码 > 部署 PostgreSQL Exporter > 获取指标 步骤完成 Exporter 部署。

#### 使用 Secret 管理 PostgreSQL 密码

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 PostgreSQL Exporter 的时候直接使用 Secret Key，需要调整对应的 `password`，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
    name: postgres-test
type: Opaque
stringData:
    username: postgres
    password: you-guess #对应 PostgreSQL 密码
```

#### 部署 PostgreSQL Exporter

在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下（**复制下面的内容，可根据实际业务调整相应的参数**）。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test # 根据业务需要调整成对应的名称，建议加上 PG 实例的信息
  namespace: postgres-test # 根据业务需要调整成对应的名称，建议加上 PG 实例的信息
  labels:
    app: postgres
    app.kubernetes.io/name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
```

```
      app.kubernetes.io/name: postgresql
  template:
    metadata:
      labels:
        app: postgres
        app.kubernetes.io/name: postgresql
    spec:
      containers:
      - name: postgres-exporter
        image: ccr.ccs.tencentyun.com/rig-agent/postgres-exporter:v0.8.0
        args:
          - "--web.listen-address=:9187" # export 开启的端口
          - "--log.level=debug" # 日志级别
        env:
          - name: DATA_SOURCE_USER
            valueFrom:
              secretKeyRef:
                name: postgres-test # 对应上一步中的 Secret 的名称
                key: username  # 对应上一步中的 Secret Key
          - name: DATA_SOURCE_PASS
            valueFrom:
              secretKeyRef:
                name: postgres-test # 对应上一步中的 Secret 的名称
                key: password  # 对应上一步中的 Secret Key
          - name: DATA_SOURCE_URI
            value: "x.x.x.x:5432/postgres?sslmode=disable"  # 对应的连接信息
        ports:
        - name: http-metrics
          containerPort: 9187
```

**说明：**
上述示例将 Secret 中的用户名密码传给了环境变量 `DATA_SOURCE_USER` 和 `DATA_SOURCE_PASS`，使用户无法查看到明文的用户名密码。您还可以用 `DATA_SOURCE_USER_FILE` / `DATA_SOURCE_PASS_FILE` 从文件读取用户名密码，或使用 `DATA_SOURCE_NAME` 将用户名密码也放在连接串里，例如 `postgresql://login:password@hostname:port/dbname` 。

**参数说明**

`DATA_SOURCE_URI` / `DATA_SOURCE_NAME` 连接串 query 部分（ `?` 之后）支持的参数如下（最新的以 GoDoc 为准）。

| 参数 | 参数说明 |
| --- | --- |
| sslmode | 是否使用 SSL，支持的值如下：<br>• – disable：不使用 SSL。<br>• – require：总是使用（跳过验证）。<br>• – verify-ca：总是使用（检查服务端提供的证书是不是由一个可信的 CA 签发）。<br>• – verify-full：总是使用（检查服务端提供的证书是不是由一个可信的 CA 签发，并且检查 hostname 是不是被证书所匹配）。 |
| fallback_application_name | 一个备选的 application_name。 |
| connect_timeout | 最大连接等待时间，单位秒。0值等于无限大。 |
| sslcert | 证书文件路径。文件数据格式必须是 PEM。 |
| sslkey | 私钥文件路径。文件数据格式必须是 PEM。 |
| sslrootcert | root 证书文件路径。文件数据格式必须是 PEM。 |

另外 Exporter 支持其他参数，如下说明（详情请参见 README ）。

| 参数 | 参数说明 | 环境变量 |
| --- | --- | --- |
| --web.listen-address | 监听地址，默认9487。 | PG_EXPORTER_WEB_LISTEN_ADDRESS |
| --web.telemetry-path | 暴露指标的路径，默认 /metrics。 | PG_EXPORTER_WEB_TELEMETRY_PATH |
| --extend.query-path | 指定一个包含自定义查询语句的 YAML 文件，参见 queries.yaml 。 | PG_EXPORTER_EXTEND_QUERY_PATH |
| --disable-default-metrics | 只使用通过 queries.yaml 提供的指标。 | PG_EXPORTER_DISABLE_DEFAULT_METRICS |
| --disable-settings-metrics | 不抓取 pg_settings 相关的指标。 | PG_EXPORTER_DISABLE_SETTINGS_METRICS |
| --auto-discover-databases | 是否自动发现 Postgres 实例上的数据库。 | PG_EXPORTER_AUTO_DISCOVER_DATABASES |
| --dumpmaps | 打印内部的指标信息，除了 debug 不要使用，方便排查自定义 queries 相关的问题。 | - |
| --constantLabels | 自定义标签，通过 key=value 的形式提供，多个标签对使用 , 分隔。 | PG_EXPORTER_CONSTANT_LABELS |
| --exclude-databases | 需要排除的数据库，仅在 --auto-discover-databases 开启的情况下有效 | PG_EXPORTER_EXCLUDE_DATABASES |
| --log.level | 日志级别 debug/info/warn/error/fatal。 | PG_EXPORTER_LOG_LEVEL |

**获取指标**

通过 `curl http://exporter:9187/metrics` 无法获取 `Postgres` 实例运行时间。我们可以通过自定义一个 `queries.yaml` 来获取该指标：

1. 创建一个包含 `queries.yaml` 的 ConfigMap 。
2. 将 ConfigMap 作为 Volume 挂载到 Exporter 某个目录下面。
3. 通过 `--extend.query-path` 来使用 ConfigMap，将上述的 Secret 以及 Deployment 进行汇总，汇总后的 YAML 如下所示。

```
# 注意：以下 document 创建一个名为 postgres-test 的 Namespace，仅作参考
apiVersion: v1
kind: Namespace
metadata:
  name: postgres-test

# 以下 document 创建一个包含用户名密码的 Secret
---
apiVersion: v1
kind: Secret
metadata:
  name: postgres-test-secret
  namespace: postgres-test
type: Opaque
stringData:
  username: postgres
  password: you-guess

# 以下 document 创建一个包含自定义指标的 queries.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-test-configmap
  namespace: postgres-test
data:
  queries.yaml: |
```

```
    pg_postmaster:
      query: "SELECT pg_postmaster_start_time as start_time_seconds from pg_postmaster_start_time()"
      master: true
      metrics:
        - start_time_seconds:
            usage: "GAUGE"
            description: "Time at which postmaster started"

# 以下 document 挂载了 Secret 和 ConfigMap ，定义了部署 Exporter 相关的镜像等参数
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test
  namespace: postgres-test
  labels:
    app: postgres
    app.kubernetes.io/name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
      app.kubernetes.io/name: postgresql
  template:
    metadata:
      labels:
        app: postgres
        app.kubernetes.io/name: postgresql
    spec:
      containers:
        - name: postgres-exporter
          image: wrouesnel/postgres_exporter:latest
          args:
            - "--web.listen-address=:9187"
            - "--extend.query-path=/etc/config/queries.yaml"
            - "--log.level=debug"
          env:
            - name: DATA_SOURCE_USER
              valueFrom:
                secretKeyRef:
                  name: postgres-test-secret
                  key: username
            - name: DATA_SOURCE_PASS
              valueFrom:
                secretKeyRef:
                  name: postgres-test-secret
                  key: password
            - name: DATA_SOURCE_URI
              value: "x.x.x.x:5432/postgres?sslmode=disable"
          ports:
            - name: http-metrics
              containerPort: 9187
          volumeMounts:
            - name: config-volume
              mountPath: /etc/config
      volumes:
        - name: config-volume
          configMap:
            name: postgres-test-configmap
```

```
```

4. 执行 `curl http://exporter:9187/metrics` 命令后，即可通过自定义的 `queries.yaml` 查询到 Postgres 实例启动时间指标。示例如下：

```
# HELP pg_postmaster_start_time_seconds Time at which postmaster started
# TYPE pg_postmaster_start_time_seconds gauge
pg_postmaster_start_time_seconds{server="x.x.x.x:5432"} 1.605061592e+09
```

**步骤2：添加采集任务**

当 Exporter 运行起来之后，需要进行以下操作配置 Prometheus 监控服务发现并采集监控指标：

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: postgres-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  namespaceSelector:
    matchNames:
    - postgres-test
  podMetricsEndpoints:
  - interval: 30s
    path: /metrics
    port: http-metrics # 前面 Exporter 那个 Container 的端口名
    relabelings:
    - action: labeldrop
      regex: __meta_kubernetes_pod_label_(pod_|statefulset_|deployment_|controller_)(.+)
    - action: replace
      regex: (.*)
      replacement: postgres-xxxxxx
      sourceLabels:
      - instance
      targetLabel: instance
  selector:
    matchLabels:
      app: postgres
```

> **说明：**
> 更多高阶用法请参见 ServiceMonitor 和 PodMonitor。

## 查看监控

> **说明：**
> 需要使用上述 获取指标 配置来获取 Postgres 实例的启动时间。

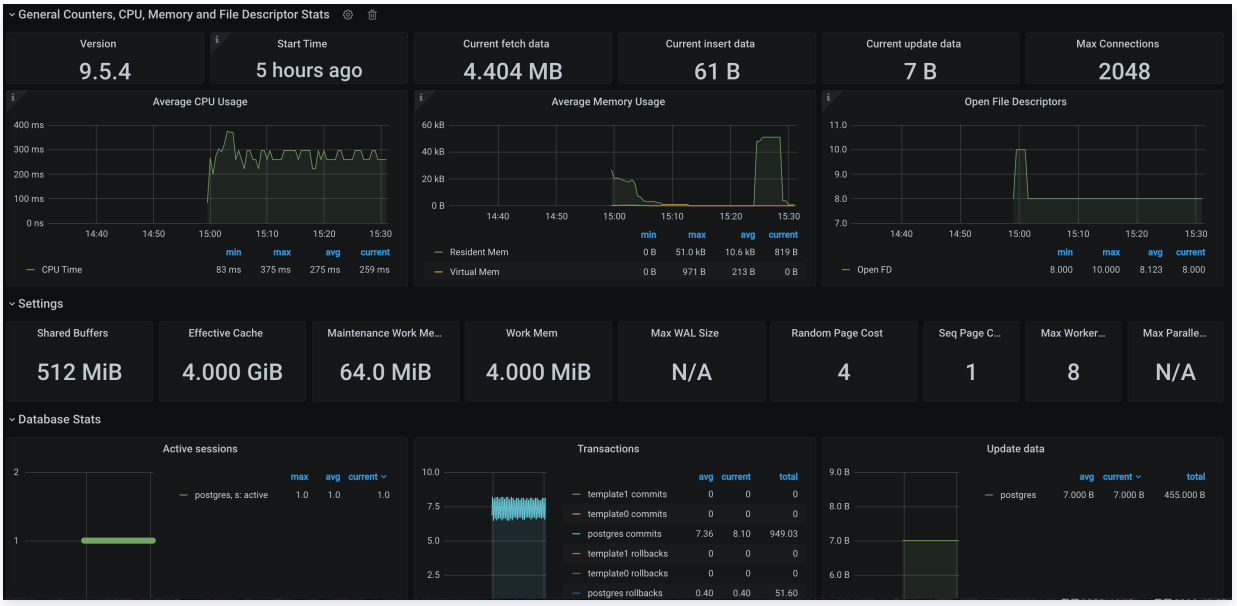
### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。

2. 选择**数据采集** > **集成中心**，在集成中心页面找到 PostgreSQL 监控，选择 **Dashboard 操作** > **Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 单击 实例 ID 右侧图标，打开您的专属 Grafana，输入您的账号密码，即可进行 Grafana 可视化大屏操作区。
4. 进入 Grafana，单击图标，展开监控面板，单击对应的监控图标名称即可查看监控数据。




## 配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 在**告警管理** > **告警策略**页面，可以添加相应的告警策略，详情请参见 新建告警策略 。

> **说明：**
> 后续 Prometheus 监控服务将提供更多 PostgreSQL 相关的告警模板。

# Nginx Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

Nginx 通过 stub_status 页面暴露了部分监控指标。Nginx Prometheus Exporter 会采集单个 Nginx 实例指标，并将其转化为 Prometheus 可用的监控数据， 最终通过 HTTP 协议暴露给 Prometheus 服务进行采集。我们可以通过 Exporter 上报重点关注的监控指标，用于异常报警和大盘展示。

## 前提条件

## 开启 NGINX stub_status 功能

> **说明：**
> 1. 下述例子为 Nginx 部署在容器服务中，其他部署方式登录及配置修改方式进行对应调整即可。
> 2. 容器服务相关操作可参见 容器服务 相关文档。

因为 Nginx Prometheus Exporter 是通过 Nginx 的 stub_status 模块对其进行监控，所以需要确保 Nginx 服务打开了 stub_status 模块，具体步骤如下：

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**，找到业务 Nginx 服务所在集群，进入集群，找到业务 Nginx 服务。
3. 登录到业务 Nginx 服务，执行以下命令检查 Nginx 是否已经开启了该模块：

```
nginx -V 2>&1 | grep -o with-http_stub_status_module
```

   - 如果在终端中输出 `with-http_stub_status_module` ，则说明 Nginx 已启用 stub_status 模块。
   - 如果未输出任何结果，则可以使用 `--with-http_stub_status_module` 参数从源码重新配置编译一个 Nginx。示例如下：

```
./configure \
… \      ## 之前编译nginx所需的命令
--with-http_stub_status_module
make
sudo make install
```

4. 若未添加 Nginx 服务相关 ConfigMap，可登录到业务 Nginx 服务，将配置目录（官方镜像为 `/etc/nginx/conf.d` ）下的 default.conf 配置信息进行拷贝，创建 ConfigMap，将配置信息添加到该 ConfigMap 中，ConfigMap 相关操作指引见 ConfigMap 管理 。
5. 确认 stub_status 模块启用之后，在 ConfigMap 的default.conf 中添加如下配置。示例如下：

```
server {
 listen 8080;  # 根据业务情况进行调整
 listen [::]:8080; # 根据业务情况进行调整
 server_name localhost; # 根据业务情况进行调整
 location = /stub_status { # 具体路径可根据业务情况进行调整
     stub_status;
 }
}
```

ConfigMap 中配置示例如下：


6. 配置修改完成之后，找到业务 Nginx 服务，点击**更多 > 重新部署**，完成配置的重新加载。非容器服务环境重加载命令：

```
nginx -t
nginx -s reload
```

7. 完成上述步骤之后，登录业务 Nginx 服务，执行如下命令，即可查看 Nginx 上次启动后工作状态的统计结果。

```
curl http://localhost:8080/stub_status  ## 根据配置文件中配置内容进行相应调整
```

```
Active connections: 45
server accepts handled requests
1056958 1156958 4491319
Reading: 0 Writing: 25 Waiting : 7
```

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **Nginx**，找到后单击它即会弹出一个安装窗口。
5. 在弹出窗口的安装页面，填写指标采集名称、地址、路径等信息，并单击**保存**。


### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 地址 | Nginx 服务的连接地址。 |
| 路径 | Nginx 服务的服务状态路径，在配置中指定 |
| 用户名 | Nginx 服务 HTTP 验证用的用户名称。 |
| 密码 | Nginx 服务 HTTP 验证用的密码。 |
| 标签 | 给指标添加自定义 Label。 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台 > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。

3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 Nginx Exporter > 验证 步骤完成 Exporter 部署。

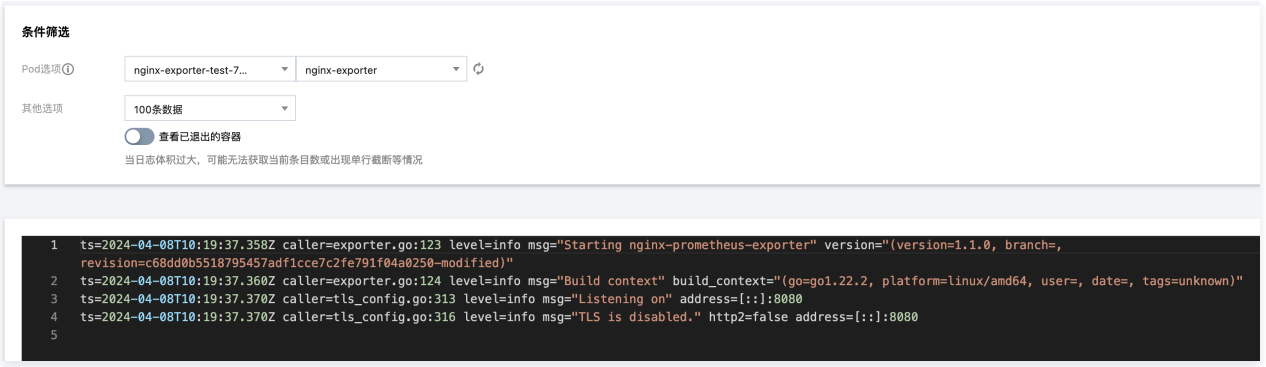
**步骤二：部署 Nginx Exporter**

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter，配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
  name: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
  namespace: nginx-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
  template:
    metadata:
      labels:
        k8s-app: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
    spec:
      containers:
      - args:
        - --web.listen-address=:8080
        - --nginx.scrape-uri=http://127.0.0.1:8080/stub_status  # 根据业务需要调整成 Nginx 实例对应地址
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:nginx-exporter-v1.1.0
        name: nginx-exporter
        ports:
        - containerPort: 9113
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

**步骤三：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 Nginx 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:8080/metrics
```

执行结果如下图所示：

```
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 2
# HELP nginx_connections_handled Handled client connections
# TYPE nginx_connections_handled counter
nginx_connections_handled 128
# HELP nginx_connections_reading Connections where NGINX is reading the request header
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
# HELP nginx_connections_waiting Idle client connections
# TYPE nginx_connections_waiting gauge
nginx_connections_waiting 1
# HELP nginx_connections_writing Connections where NGINX is writing the response back to the client
# TYPE nginx_connections_writing gauge
nginx_connections_writing 1
# HELP nginx_exporter_build_info A metric with a constant '1' value labeled by version, revision, branch, goversion from which nginx_exporter was built, and the goos and goarch for the build.
# TYPE nginx_exporter_build_info gauge
nginx_exporter_build_info{branch="",goarch="amd64",goos="linux",goversion="go1.22.2",revision="c68dd0b5518795457adf1cce7c2fe791f04a0250-modified",tags="unknown",version="1.1.0"} 1
# HELP nginx_http_requests_total Total http requests
# TYPE nginx_http_requests_total counter
nginx_http_requests_total 29564
# HELP nginx_up Status of the last metric scrape
# TYPE nginx_up gauge
nginx_up 1
```

**步骤四：添加采集任务**

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: nginx-exporter   # 填写一个唯一名称
  namespace: cm-prometheus   # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics   # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
```

```
        replacement: 'crs-xxxxxx' # 调整成对应的 Nginx 实例信息
    namespaceSelector:    # 选择要监控pod所在的namespace
      matchNames:
      - nginx-demo
    selector:  # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: nginx-exporter
```

## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 middleware 文件夹中找到 nginx 实例监控面板，查看实例相关监控数据，如下图所示：


## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

# 附录：Nginx Exporter 采集参数说明

### 全局配置参数

| 名称 | 描述 |
| --- | --- |
| web.telemetry-path | 指标暴露路径，默认 `/metrics` 。 |
| nginx.scrape-uri | nginx 指标抓取 url，默认 `http://127.0.0.1:8080/stub_status` 。 |
| [no-]nginx.plus | 是否启用 NGINX Plus，默认是启用的。 |
| [no-]nginx.ssl-verify | 是否验证 ssl 证书。 |
| nginx.ssl-ca-cert | ssl 证书路径。 |
| nginx.ssl-client-cert | ssl 证书路径。 |

| | |
|---|---|
| nginx.ssl-client-key | ssl 证书路径。 |
| nginx.timeout | nginx 指标抓取超时。 |
| prometheus.const-label | 将在每个指标中使用的标签。格式为 label=value，允许重复多次。 |
| [no-]web.systemd-socket | 使用 systemd 套接字监听器代替端口监听器（仅限 Linux）。 |
| web.listen-address | 监听地址，默认：9113。 |
| web.config.file | 配置文件的路径，可以启用 TLS 或身份验证（实验性参数）。 |
| log.level | 日志级别，默认 info。 |
| log.format | 日志消息的输出格式，取值范围：[logfmt，json]，默认 logfmt。 |
| version | 打印版本信息。 |

# Redis Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

在使用数据库 Redis 过程中需要对 Redis 运行状态进行监控，以便了解 Redis 服务是否运行正常，排查 Redis 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Redis 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控 Redis。

> **说明：**
> 如果需要监控的 Redis 是腾讯云 云数据库 Redis，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Redis**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 地址 | Redis 的连接地址。 |
| 密码 | Redis 的密码。 |

| 标签 | 给指标添加自定义 Label。 |
| --- | --- |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务中找到对应容器集群完成关联集群操作。可参见指引 关联集群。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 Redis 密码 > 部署 Redis Exporter > 验证 步骤完成 Exporter 部署。

#### 使用 Secret 管理 Redis 密码

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 Redis Exporter 的时候直接使用 Secret Key，需要调整对应的 `password`，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
    name: redis-secret-test
    namespace: redis-test
type: Opaque
stringData:
    password: you-guess  #对应 Redis 密码
```

#### 部署 Redis Exporter

在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下：

> **说明：**
> 更多 Exporter 详细参数介绍请参见 redis_exporter。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-redis-
exporter
  name: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-redis-exporter
  namespace: redis-test # 选择一个适合的 namespace 来部署 exporter，如果没有需要新建一个 namespace
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      k8s-app: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-redis-
exporter
  template:
    metadata:
      labels:
        k8s-app: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-redis-
exporter
    spec:
      containers:
      - env:
        - name: REDIS_ADDR
          value: ip:port # 对应 Redis 的 ip:port
        - name: REDIS_PASSWORD
          valueFrom:
            secretKeyRef:
              name: redis-secret-test
              key: password
        image: ccr.ccs.tencentyun.com/rig-agent/redis-exporter:v1.32.0
        imagePullPolicy: IfNotPresent
        name: redis-exporter
        ports:
        - containerPort: 9121
          name: metric-port  # 这个名称在配置抓取任务的时候需要
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

**验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：


3. 单击 **Pod 管理**页签，进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录** Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 Redis 指标。如发现未能得到对应的数据，请检查一下 `REDIS_ADDR` 和 `REDIS_PASSWORD` 是否正确。示例如下：

```
curl http://localhost:9121/metrics
```

命令执行结果如下图所示：

```
# TYPE redis_keyspace_hits_total counter
redis_keyspace_hits_total 29916
# HELP redis_keyspace_misses_total keyspace_misses_total metric
# TYPE redis_keyspace_misses_total counter
redis_keyspace_misses_total 29
# HELP redis_last_slow_execution_duration_seconds The amount of time needed for last slow execution, in sec
# TYPE redis_last_slow_execution_duration_seconds gauge
redis_last_slow_execution_duration_seconds 0.011276
# HELP redis_latency_spike_duration_seconds Length of the last latency spike in seconds
# TYPE redis_latency_spike_duration_seconds gauge
redis_latency_spike_duration_seconds{event_name="command"} 0.011
redis_latency_spike_duration_seconds{event_name="fast-command"} 0.022
# HELP redis_latency_spike_last When the latency spike last occurred
# TYPE redis_latency_spike_last gauge
redis_latency_spike_last{event_name="command"} 1.604752448e+09
redis_latency_spike_last{event_name="fast-command"} 1.604738646e+09
# HELP redis_latest_fork_seconds latest_fork_seconds metric
# TYPE redis_latest_fork_seconds gauge
redis_latest_fork_seconds 0
# HELP redis_lazyfree_pending_objects lazyfree_pending_objects metric
# TYPE redis_lazyfree_pending_objects gauge
redis_lazyfree_pending_objects 0
# HELP redis_loading_dump_file loading_dump_file metric
# TYPE redis_loading_dump_file gauge
redis_loading_dump_file 0
# HELP redis_master_repl_offset master_repl_offset metric
# TYPE redis_master_repl_offset gauge
redis_master_repl_offset 2.37644710082e+11
# HELP redis_mem_fragmentation_ratio mem_fragmentation_ratio metric
# TYPE redis_mem_fragmentation_ratio gauge
redis_mem_fragmentation_ratio 1.43
# HELP redis_memory_max_bytes memory_max_bytes metric
# TYPE redis_memory_max_bytes gauge
redis_memory_max_bytes 1.2884901888e+10
# HELP redis_memory_used_bytes memory_used_bytes metric
# TYPE redis_memory_used_bytes gauge
redis_memory_used_bytes 1.1479248e+07
```

**步骤二：添加采集任务**

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port  # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Redis 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 Redis 实例 IP
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - redis-test
```

```
  selector:     # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: redis-exporter
```

> **说明：**
> 由于 `Exporter` 和 `Redis` 部署在不同的服务器上，因此建议通过 Prometheus Relabel 机制将 Redis 实例的信息放到监控指标中，以方便定位问题。

## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，进入集成中心页面。找到 Redis 监控，安装对应的 Grafana Dashboard 即可开启 Redis 监控大盘，查看实例相关的监控数据，如下图所示：


## 配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**告警管理 > 告警策略**，可以添加相应的告警策略，详情请参见 新建告警策略 。

# Aerospike Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

Aerospike Exporter 是一个用于 Aerospike 数据库的 Prometheus 指标导出工具，允许用户监视和收集 Aerospike 数据库的性能指标和统计信息。它可以帮助用户实时监控 Aerospike 集群的健康状况、性能表现和负载情况，有助于进行故障排除、性能优化和规划容量。通过将这些指标导出到 Prometheus，用户可以利用 Prometheus 的强大功能进行数据可视化、报警和分析。腾讯云可观测平台 Prometheus 提供了 Aerospike Exporter 集成及开箱即用的 Grafana 监控大盘。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Aerospike**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 域名 | Aerospike 数据库域名。 |
| 地址 | Aerospike 数据库端口。 |

| | |
|---|---|
| 用户名 | Aerospike 数据库用户名称。 |
| 密码 | Aerospike 数据库密码。 |
| 标签 | 给指标添加自定义 Label。 |

## 方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务中找到对应容器集群完成关联集群操作。可参见指引 关联集群。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 Exporter 配置 > 部署 Aerospike Exporter > 验证 步骤完成 Exporter 部署。

#### 步骤二：部署 Exporter 配置

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter，配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: aerospike-secret-test   # 根据业务需要调整成相应名称
  namespace: aerospike-demo    # 根据业务需要调整到相应命名空间
type: Opaque
stringData:
  ape.toml: |-
      [Agent]
          # metrics server timeout in seconds
          timeout = 30

          # support system statistics also
          refresh_system_stats = true

          # prometheus binding port
          bind = ":8080"  # 暴露指标端口

      [Aerospike]
          db_host = "127.0.0.1"    # 根据业务需要调整成对应的 IP 或域名
          db_port = 3000    # 根据业务需要调整成对应的端口
          user = "admin"  # 根据业务需要调整成对应的用户名
          password = "admin"  # 根据业务需要调整成对应的密码

          # timeout for sending commands to the server node in seconds
          timeout = 30
  gauge_stats_list.toml: |-
      # This file represents a list of metrics which are treated as Gauges while exporting to Prometheus
```

```
or some other Observability tool.
      # to know more about these stats, please visit https://docs.aerospike.com

      #
      # SETS: below section define all Sets stats which are treated as Gauges
      #
      sets_gauge_stats = [

        "device_data_bytes",
        "index_populating",
        "memory_data_bytes",
        "objects",
        "sindexes",
        "tombstones",
        "truncate_lut",

      # 7.0 changes
        "data_used_bytes",
        "truncating",
      ]

      #
      # XDR: below section define all XDR stats which are treated as Gauges
      #
      xdr_gauge_stats = [
        "compression_ratio",
        "in_progress",
        "in_queue",
        "lag",
        "lap_us",
        "latency_ms",
        "nodes",
        "recoveries_pending",
        "throughput",
        "uncompressed_pct",
      ]

      #
      # Sindex: below section define all Sindex stats which are treated as Gauges
      #
      sindex_gauge_stats = [
        "entries_per_bval",
        "entries_per_rec",
        "entries",
        "histogram",                    # removed in server6.0
        "ibtr_memory_used",             # removed in server6.0
        "keys", # removed in server6.0
        "load_pct",
        "load_time",
        "loadtime",                     # removed in server6.0
        "memory_used",                  # deprecated in server6.3 version and replaced by used_bytes
        "nbtr_memory_used",             # removed in server6.0
        "query_basic_avg_rec_count",    # removed in server6.0
        "used_bytes",                   # added in server6.3 represents memory used by data (aka
memory_used)
      ]

      #
      # Node: below section define all Node stats which are treated as Gauges
      #
```

```
        node_gauge_stats = [
          "batch_index_proto_compression_ratio",
          "batch_index_proto_uncompressed_pct",
          "batch_index_queue",
          "batch_index_unused_buffers",
          "client_connections",
          "cluster_clock_skew_ms",
          "cluster_clock_skew_stop_writes_sec",
          "cluster_integrity",
          "cluster_is_member",
          "cluster_max_compatibility_id",
          "cluster_min_compatibility_id",
          "cluster_size",
          "fabric_bulk_recv_rate",
          "fabric_bulk_send_rate",
          "fabric_connections",
          "fabric_ctrl_recv_rate",
          "fabric_ctrl_send_rate",
          "fabric_meta_recv_rate",
          "fabric_meta_send_rate",
          "fabric_rw_recv_rate",
          "fabric_rw_send_rate",
          "failed_best_practices",
          "heap_active_kbytes",
          "heap_allocated_kbytes",
          "heap_efficiency_pct",
          "heap_mapped_kbytes",
          "heap_site_count",
          "heartbeat_connections",
          "info_queue",
          "migrate_partitions_remaining",
          "objects",
          "process_cpu_pct",
          "proxy_in_progress",
          "queries_active",
          "rw_in_progress",
          "scans_active",
          "sindex_gc_list_creation_time",
          "sindex_gc_list_deletion_time",
          "system_free_mem_pct",
          "system_kernel_cpu_pct",
          "system_total_cpu_pct",
          "system_user_cpu_pct",
          "threads_detached",
          "threads_joinable",
          "threads_pool_active",
          "threads_pool_total",
          "time_since_rebalance",
          "tombstones",
          "tree_gc_queue",
          "tsvc_queue",
          #
          # 4.x XDR stats
          "dlog_free_pct",
          "dlog_used_objects",
          "xdr_active_failed_node_sessions",
          "xdr_active_link_down_sessions",
          "xdr_global_lastshiptime",
          "xdr_read_active_avg_pct",
          "xdr_read_idle_avg_pct",
          "xdr_read_latency_avg",
```

```
    "xdr_read_reqq_used_pct",
    "xdr_read_reqq_used",
    "xdr_read_respq_used",
    "xdr_read_txnq_used_pct",
    "xdr_read_txnq_used",
    "xdr_ship_compression_avg_pct",
    "xdr_ship_inflight_objects",
    "xdr_ship_latency_avg",
    "xdr_ship_outstanding_objects",
    "xdr_throughput",
    "xdr_timelag",
]

#
# Namespace: below section define all Namespace stats which are treated as Gauges
#
namespace_gauge_stats =[
    "appeals_rx_active",
    "appeals_tx_active",
    "appeals_tx_remaining",
    "available_bin_names",
    "cache_read_pct",
    "clock_skew_stop_writes",
    "dead_partitions",
    "defrag_q",
    "device_available_pct",
    "device_compression_ratio",
    "device_free_pct",
    "device_total_bytes",
    "device_used_bytes",
    "effective_is_quiesced",
    "effective_prefer_uniform_balance",
    "effective_replication_factor",
    "evict_ttl",
    "hwm_breached",
    "index_flash_alloc_bytes",
    "index_flash_alloc_pct",
    "index_flash_used_bytes",
    "index_flash_used_pct",
    "index_pmem_used_bytes",
    "index_pmem_used_pct",
    "master_objects",
    "master_tombstones",
    "memory_free_pct",
    "memory_used_bytes",
    "memory_used_data_bytes",
    "memory_used_index_bytes",
    "memory_used_set_index_bytes",
    "memory_used_sindex_bytes",
    "migrate_rx_instances",
    "migrate_rx_partitions_active",
    "migrate_rx_partitions_initial",
    "migrate_rx_partitions_remaining",
    "migrate_signals_active",
    "migrate_signals_remaining",
    "migrate_tx_instances",
    "migrate_tx_partitions_active",
    "migrate_tx_partitions_imbalance",
    "migrate_tx_partitions_initial",
    "migrate_tx_partitions_lead_remaining",
    "migrate_tx_partitions_remaining",
```

```
        "n_nodes_quiesced",
        "non_expirable_objects",
        "non_replica_objects",
        "non_replica_tombstones",
        "ns_cluster_size",
        "nsup_cycle_deleted_pct",
        "nsup_cycle_duration",
        "nsup_cycle_sleep_pct",
        "objects",
        "pending_quiesce",
        "pmem_available_pct",
        "pmem_compression_ratio",
        "pmem_free_pct",
        "pmem_total_bytes",
        "pmem_used_bytes",
        "prole_objects",
        "prole_tombstones",
        "query_aggr_avg_rec_count",
        "query_basic_avg_rec_count",
        "query_proto_compression_ratio",
        "query_proto_uncompressed_pct",
        "record_proto_compression_ratio",
        "record_proto_uncompressed_pct",
        "scan_proto_compression_ratio",
        "scan_proto_uncompressed_pct",
        "shadow_write_q",
        "stop_writes",
        "storage-engine.device.defrag_q",
        "storage-engine.device.free_wblocks",
        "storage-engine.device.shadow_write_q",
        "storage-engine.device.used_bytes",
        "storage-engine.device.write_q",
        "storage-engine.device.age",
        "storage-engine.file.defrag_q",
        "storage-engine.file.free_wblocks",
        "storage-engine.file.shadow_write_q",
        "storage-engine.file.used_bytes",
        "storage-engine.file.write_q",
        "storage-engine.file.age",
        "storage-engine.stripe.defrag_q",
        "storage-engine.stripe.free_wblocks",
        "storage-engine.stripe.shadow_write_q",
        "storage-engine.stripe.used_bytes",
        "storage-engine.stripe.write_q",
        "storage-engine.stripe.age",
        "storage-engine.stripe.backing_write_q",
        "migrate_fresh_partitions",
        "tombstones",
        "truncate_lut",
        "unavailable_partitions",
        "unreplicated_records",
        "write_q",
        "xdr_bin_cemeteries",
        "xdr_tombstones",

    # added in 7.0
        "data_avail_pct",
        "data_compression_ratio",
        "data_total_bytes",
        "data_used_bytes",
        "data_used_pct",
```

```
        "index_mounts_used_pct",
        "index_used_bytes",
        "indexes_memory_used_pct",
        "set_index_used_bytes",
        "sindex_mounts_used_pct",
        "sindex_used_bytes",
        "truncating",

      ]

      # System Info Gauge metrics list
      #
      system_info_gauge_stats = [
        "",
      ]
```

**步骤三：部署 Aerospike Exporter**

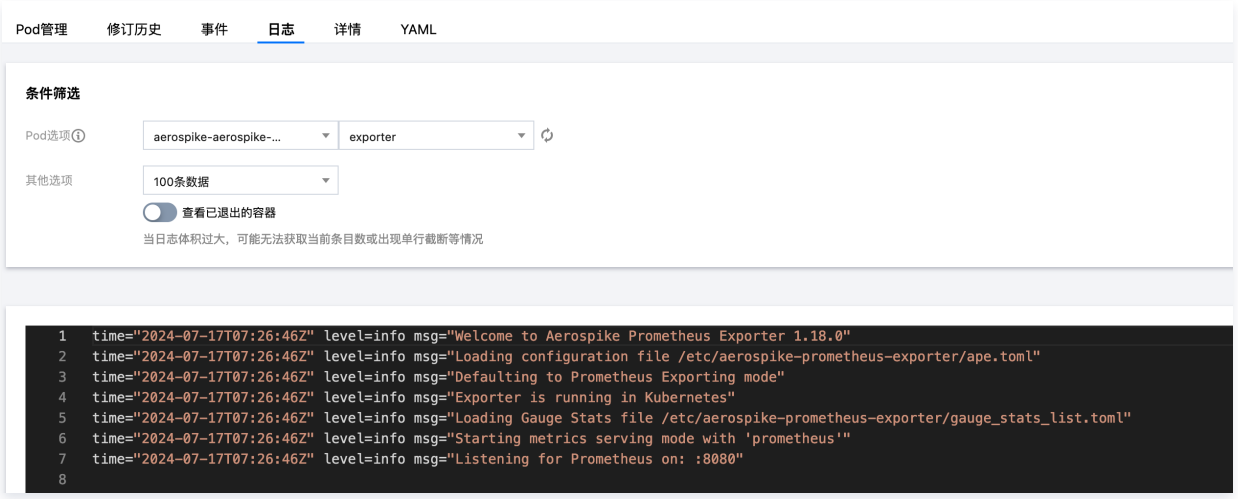
1. 在左侧菜单中选择**工作负载 > Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
  name: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
  namespace: aerospike-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
  template:
    metadata:
      labels:
        k8s-app: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
    spec:
      volumes:
      - name: sec
        secret:
          defaultMode: 420
          secretName: aerospike-secret-test # 对应 步骤二 配置名称
      containers:
      - name: aerospike-exporter
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:aerospike-exporter-1.18.0
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 8080  # 对应 步骤二 配置中的指标导出端口
            name: metrics
        livenessProbe:
          tcpSocket:
            port: metrics
        readinessProbe:
          tcpSocket:
            port: metrics
        volumeMounts:
          - mountPath: /etc/aerospike-prometheus-exporter
```

```
            name: sec
            readOnly: true
```

**步骤四：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，无报错信息输出即可，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 Aerospike 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:8080/metrics
```

执行结果如下图所示：


**步骤五：添加采集任务**

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: aerospike-exporter   # 填写一个唯一名称
  namespace: cm-prometheus   # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
```

```
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Aerospike 实例 ID
  namespaceSelector:    # 选择要监控 aerospike exporter pod 所在的 namespace
    matchNames:
    - aerospike-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: aerospike-exporter
```

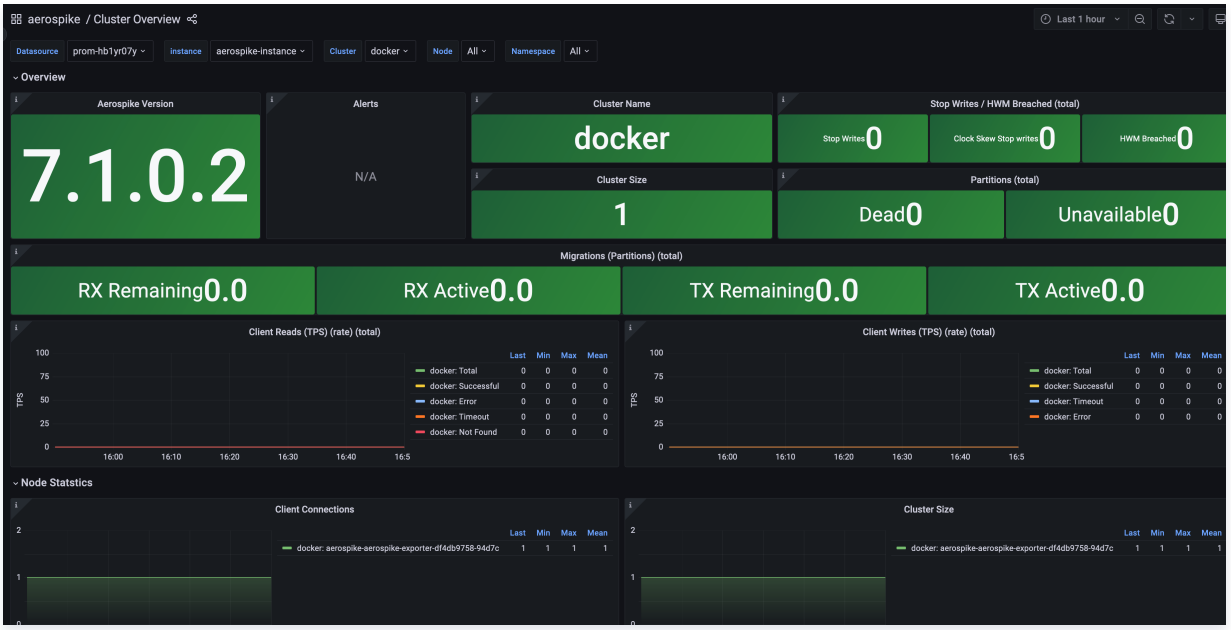
## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 aerospike 文件夹中找到 Aerospike 实例相关监控面板，查看实例相关监控数据，如下图所示：




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

# 附录：Aerospike Exporter 配置文件主要配置项

## Agent 配置项

| 名称 | 描述 |
| --- | --- |

| | |
|---|---|
| bind | 指标导出端口，默认":9145" |
| cert_file | 签名用证书文件 |
| key_file | 签名用证书文件 |
| root_ca | 签名用证书文件 |
| basic_auth_username | http auth 验证用户名 |
| basic_auth_password | http auth 验证密码 |
| timeout | 指标拉取超时 |
| labels | 自定义标签值 |
| refresh_system_stats | 支持系统数据统计 |

## Aerospike 配置项

| 名称 | 描述 |
|---|---|
| db_host | Aerospike 数据库域名或 IP |
| db_port | Aerospike 数据库服务端口 |
| auth_mode | Aerospike 校验模式，默认 internal，取值有 "external"，"internal"，"pki"，"" |
| user | Aerospike 数据库用户名 |
| password | Aerospike 数据库密码 |
| timeout | Aerospike 数据库连接超时 |

# MySQL Exporter 接入

最近更新时间：2025-02-19 19:46:16

## 操作场景

MySQL Exporter 是社区专门为采集 MySQL/MariaDB 数据库监控指标而设计开发，通过 Exporter 上报核心的数据库指标，用于异常报警和监控大盘展示，腾讯云可观测平台 Prometheus 提供了与 MySQL Exporter 集成及开箱即用的 Grafana 监控大盘。
目前，Exporter 支持高于5.6版本的 MySQL 和高于10.1版本的 MariaDB。在 MySQL/MariaDB 低于5.6版本时，部分监控指标可能无法被采集。

> **说明：**
> 如果需要监控的 MySQL 是腾讯云 云数据库 MySQL，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装（推荐）

#### 操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **MySQL**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。 |

| | |
|---|---|
| | • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | MySQL 的用户名称。 |
| 密码 | MySQL 的密码。 |
| 地址 | MySQL 的连接地址。 |
| 标签 | 给指标添加自定义 Label。 |
| Exporter 配置 | 参数描述可能存在偏差，具体可参考 官方文档 。<br>• auto_increment.columns：从 information_schema 采集 auto_increment 列和最大值。<br>• binlog_size：采集所有已注册的 binlog 文件的当前大小。<br>• engine_innodb_status：从 SHOW ENGINE INNODB STATUS 采集。<br>• engine_tokudb_status：从 SHOW ENGINE TOKUDB STATUS 采集。<br>• global_status：从 SHOW GLOBAL STATUS 采集，默认为 true。<br>• global_variables：从 SHOW GLOBAL VARIABLES 采集，默认为 true。<br>• info_schema.clientstats：如果使用 userstat=1 运行，则设置为 true 以采集客户端统计信息。<br>• info_schema.innodb_metrics：从 information_schema.innodb_metrics 采集指标。<br>• info_schema.innodb_tablespaces：从 information_schema.innodb_sys_tablespaces 采集指标。<br>• info_schema.innodb_cmp：从 information_schema.innodb_cmp 采集 InnoDB 压缩表指标，默认为 true。<br>• info_schema.innodb_cmpmem：从 information_schema.innodb_cmpmem 采集 InnoDB 缓冲池压缩指标(默认为 true)。<br>• info_schema.processlist：从 information_schema.processlist 采集线程状态计数。<br>• info_schema.query_response_time：如果 query_response_time_stats 为 ON，则采集查询响应时间分布(默认为 true)。<br>• info_schema.tables：从 information_schema.tables 采集指标。<br>• info_schema.tables.databases：用于收集表统计信息的数据库列表，逗号分隔，默认是 '*' 表示所有数据库。<br>• info_schema.tablestats：如果使用 userstat=1 运行，则设置为 true 以采集表统计信息。<br>• info_schema.schemastats：如果使用 userstat=1 运行，则设置为 true 以采集架构统计信息。<br>• info_schema.userstats：如果使用 userstat=1 运行，则设置为 true 以采集用户统计信息。<br>• perf_schema.eventsstatements：从 performance_schema.events_statements_summary_by_digest 采集指标。<br>• perf_schema.eventsstatements.digest_text_limit：标准化语句文本的最大长度，默认值为120。<br>• perf_schema.eventsstatements.limit：按响应时间限制事件语句摘要的数量，默认值为250。<br>• perf_schema.eventsstatements.timelimit：限制 'last_seen' 事件语句的最大时间，单位为秒，默认值为86400。<br>• perf_schema.eventsstatementssum：从 performance_schema.events_statements_summary_by_digest 汇总采集指标。<br>• perf_schema.eventswaits：从 performance_schema.events_waits_summary_global_by_event_name 采集指标。<br>• perf_schema.file_events：从 performance_schema.file_summary_by_event_name 采集指标。<br>• perf_schema.file_instances：从 performance_schema.file_summary_by_instance采集指标。<br>• perf_schema.indexiowaits：从 performance_schema.table_io_waits_summary_by_index_usage 采集指标。<br>• perf_schema.tableiowaits：从 performance_schema.table_io_waits_summary_by_table 采集指标。<br>• perf_schema.tablelocks：从 performance_schema.table_lock_waits_summary_by_table 采集指标。<br>• perf_schema.replication_group_member_stats：从 performance_schema.replication_group_member_stats 采集指标。<br>• perf_schema.replication_applier_status_by_worker：从 performance_schema.replication_applier_status_by_worker 采集指标。<br>• slave_status：从 SHOW SLAVE STATUS 采集，默认为 true。<br>• slave_hosts：从 SHOW SLAVE HOSTS 采集。<br>• heartbeat：从 heartbeat 采集。<br>• heartbeat.database：收集 heartbeat 数据的数据库，默认值为 heartbeat。<br>• heartbeat.table：收集 heartbeat 数据的表，默认值为 heartbeat。 |

### 方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

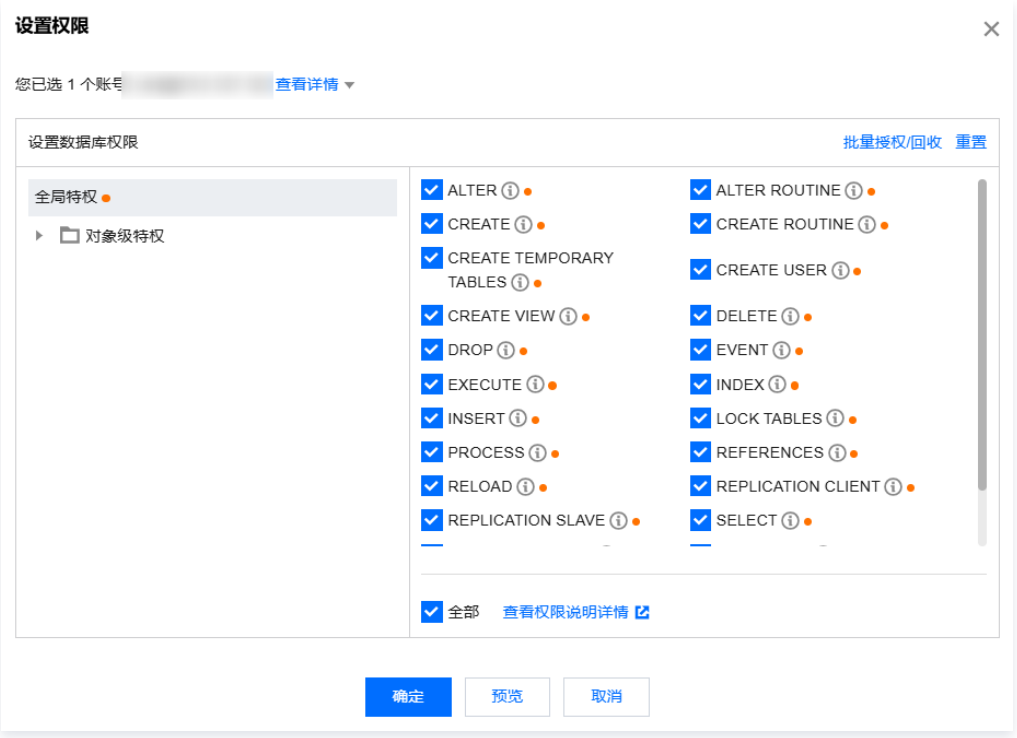
- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台，选择并进入对应的 Prometheus 实例，在**数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

### 操作步骤

#### 步骤1：数据库授权

因为 MySQL Exporter 是通过查询数据库中状态数据来对其进行监控，所以需要为对应的数据库实例进行授权。账号和密码需根据实际情况而定，授权步骤如下：

1. 登录 云数据库 MySQL 控制台。
2. 在实例列表页面单击需要授权的数据库名称，进入数据库详情页。
3. 选择**数据库管理 > 账号管理**，进入账号管理页面，根据业务实际需要创建监控建立的账号。
4. 单击账号右侧操作项下的**修改权限**，修改对应权限。示例如下图所示：




您也可以在您的云服务器中通过执行以下命令进行授权。

```
CREATE USER 'exporter'@'ip' IDENTIFIED BY 'XXXXXXXX' WITH MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter'@'ip';
```

> **说明**
> 建议为该用户设置最大连接数限制，以避免因监控数据抓取对数据库带来影响。但并非所有的数据库版本中都可以生效，例如 MariaDB 10.1 版本不支持最大连接数设置，则无法生效。详情请参见 MariaDB 说明 。

#### 步骤2：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏选择**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。

4. 使用 Secret 管理 MySQL 连接串。

4.1 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。

4.2 在页面右上角单击 **YAML 创建**，创建 YAML 配置，配置说明如下：

使用 Kubernetes 的 Secret 来管理连接串，并对连接串进行加密处理，在启动 MySQL Exporter 的时候直接使用 Secret Key，需要调整对应的**连接串**，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret-test
  namespace: mysql-demo
type: Opaque
stringData:
  datasource: "user:password@tcp(ip:port)/"  #对应 MySQL 连接串信息
```

5. 部署 MySQL Exporter。

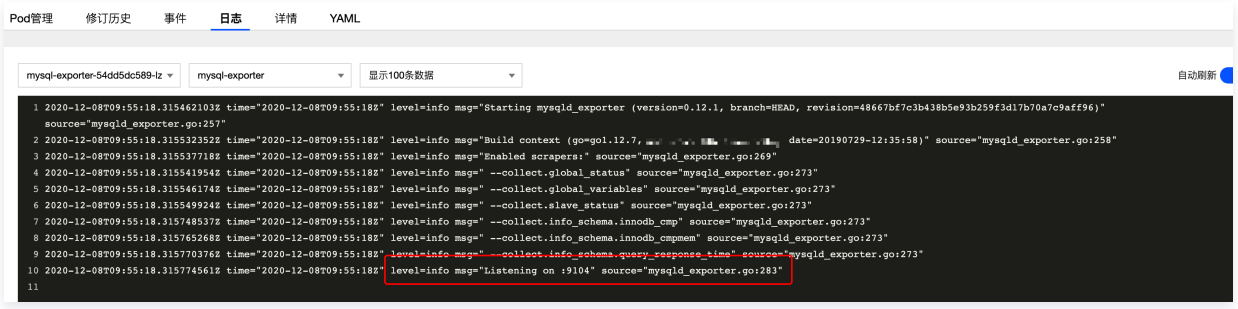
在 Deployment 管理页面，选择对应的命名空间来进行部署服务，可以通过 控制台的方式 创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
  name: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
  namespace: mysql-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
  template:
    metadata:
      labels:
        k8s-app: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
    spec:
      containers:
      - env:
        - name: DATA_SOURCE_NAME
          valueFrom:
            secretKeyRef:
              name: mysql-secret-test # 对应上一步中的 Secret 的名称
              key: datasource # 对应上一步中的 Secret Key
        image: ccr.ccs.tencentyun.com/rig-agent/mysqld-exporter:v0.12.1
        imagePullPolicy: IfNotPresent
        name: mysql-exporter
        ports:
        - containerPort: 9104
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

6. 验证。

6.1 在 **Deployment** 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

6.2 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：

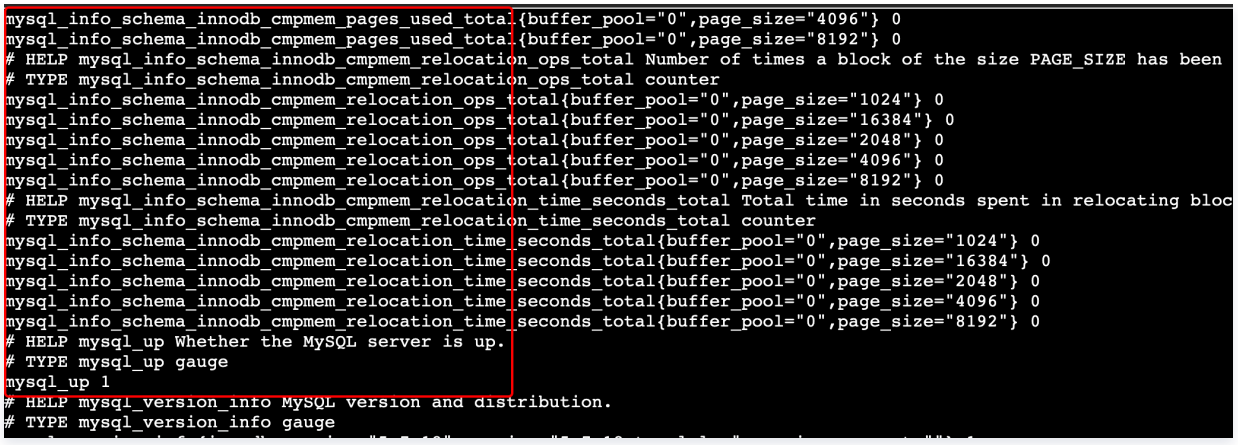



6.3 单击 **Pod 管理**页签进入 Pod 页面。

6.4 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令，可以正常得到对应的 MySQL 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -O- localhost:9104/metrics
```

执行结果如下图所示：




**步骤2：添加采集任务**

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 **PodMonitors** 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: mysql-exporter   # 填写一个唯一名称
  namespace: cm-prometheus   # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
```

```
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 MySQL 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 MySQL 实例 IP
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - mysql-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: mysql-exporter
```

## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，进入集成中心页面，找到并单击 **MySQL**，选择 **Dashboard > Dashboard 操作**下的**安装/升级 Dashboard**，单击**安装/升级**安装对应的 Grafana Dashboard。
3. 选择**已集成**，在已集成列表中单击 Grafana 图标即可自动打开 MySQL 监控大盘，查看实例相关的监控数据，如下图所示：



## 配置告警

腾讯云 Prometheus 托管服务内置了部分 MySQL 数据库的报警策略模板，可根据业务实际的情况调整对应的阈值来添加告警策略。详情请参见 新建告警策略。

## 附录：MySQL Exporter 采集参数说明

MySQL Exporter 使用各种 `Collector` 来控制采集数据的启停，具体参数如下：

| 名称 | MySQL 版本 | 描述 |
| --- | --- | --- |
| collect.auto_increment.columns | 5.1 | 在 information_schema 中采集 auto_increment 和最大值。 |
| collect.binlog_size | 5.1 | 采集所有注册的 binlog 文件大小。 |
| collect.engine_innodb_status | 5.1 | 从 SHOW ENGINE INNODB STATUS 中采集状态数据。 |
| collect.engine_tokudb_status | 5.6 | 从 SHOW ENGINE TOKUDB STATUS 中采集状态数据。 |
| collect.global_status | 5.1 | 从 SHOW GLOBAL STATUS（默认开启）中采集状态数据。 |
| collect.global_variables | 5.1 | 从 SHOW GLOBAL VARIABLES（默认开启）中采集状态数据。 |
| collect.info_schema.clientstats | 5.5 | 如果设置了 userstat=1，设置成 true 来开启用户端数据采集。 |
| collect.info_schema.innodb_metrics | 5.6 | 从 information_schema.innodb_metrics 中采集监控数据。 |
| collect.info_schema.innodb_tablespaces | 5.7 | 从 information_schema.innodb_sys_tablespaces 中采集监控数据。 |
| collect.info_schema.innodb_cmp | 5.5 | 从 information_schema.innodb_cmp 中采集 InnoDB 压缩表的监控数据。 |
| collect.info_schema.innodb_cmpmem | 5.5 | 从 information_schema.innodb_cmpmem 中采集 InnoDB buffer pool compression 的监控数据。 |
| collect.info_schema.processlist | 5.1 | 从 information_schema.processlist 中采集线程状态计数的监控数据。 |
| collect.info_schema.processlist.min_time | 5.1 | 线程可以被统计所维持的状态的最小时间。（默认：0） |
| collect.info_schema.query_response_time | 5.5 | 如果 query_response_time_stats 被设置成 ON，采集查询相应时间的分布。 |
| collect.info_schema.replica_host | 5.6 | 从 information_schema.replica_host_status 中采集状态数据。 |
| collect.info_schema.tables | 5.1 | 从 information_schema.tables 中采集状态数据。 |
| collect.info_schema.tables.databases | 5.1 | 设置需要采集表状态的数据库，或者设置成 '`*`' 来采集所有的。 |
| collect.info_schema.tablestats | 5.1 | 如果设置了 userstat=1，设置成 true 来采集表统计数据。 |
| collect.info_schema.schemastats | 5.1 | 如果设置了 userstat=1，设置成 true 来采集 schema 统计数据。 |
| collect.info_schema.userstats | 5.1 | 如果设置了 userstat=1，设置成 true 来采集用户统计数据。 |
| collect.perf_schema.eventsstatements | 5.6 | 从 performance_schema.events_statements_summary_by_digest 中采集监控数据。 |
| collect.perf_schema.eventsstatements.digest_text_limit | 5.6 | 设置正常文本语句的最大长度。（默认：120） |
| collect.perf_schema.eventsstatements.limit | 5.6 | 事件语句的限制数量。（默认：250） |
| collect.perf_schema.eventsstatements.timelimit | 5.6 | 限制事件语句 'last_seen' 可以保持多久，单位为秒。（默认：86400） |

| | | |
|---|---|---|
| collect.perf_schema.eventsstatementssum | 5.7 | 从 performance_schema.events_statements_summary_by_digest summed 中采集监控数据。 |
| collect.perf_schema.eventswaits | 5.5 | 从 performance_schema.events_waits_summary_global_by_event_name 中采集监控数据。 |
| collect.perf_schema.file_events | 5.6 | 从 performance_schema.file_summary_by_event_name 中采集监控数据。 |
| collect.perf_schema.file_instances | 5.5 | 从 performance_schema.file_summary_by_instance 中采集监控数据。 |
| collect.perf_schema.indexiowaits | 5.6 | 从 performance_schema.table_io_waits_summary_by_index_usage 中采集监控数据。 |
| collect.perf_schema.tableiowaits | 5.6 | 从 performance_schema.table_io_waits_summary_by_table 中采集监控数据。 |
| collect.perf_schema.tablelocks | 5.6 | 从 performance_schema.table_lock_waits_summary_by_table 中采集监控数据。 |
| collect.perf_schema.replication_group_members | 5.7 | 从 performance_schema.replication_group_members 中采集监控数据。 |
| collect.perf_schema.replication_group_member_stats | 5.7 | 从 from performance_schema.replication_group_member_stats 中采集监控数据。 |
| collect.perf_schema.replication_applier_status_by_worker | 5.7 | 从 performance_schema.replication_applier_status_by_worker 中采集监控数据。 |
| collect.slave_status | 5.1 | 从 SHOW SLAVE STATUS（默认开启）中采集监控数据。 |
| collect.slave_hosts | 5.1 | 从 SHOW SLAVE HOSTS 中采集监控数据。 |
| collect.heartbeat | 5.1 | 从 heartbeat 中采集监控数据。 |
| collect.heartbeat.database | 5.1 | 数据库心跳检测的数据源。（默认：heartbeat） |
| collect.heartbeat.table | 5.1 | 表心跳检测的数据源。（默认：heartbeat） |
| collect.heartbeat.utc | 5.1 | 对当前的数据库服务器使用 UTC 时间戳（`pt-heartbeat` is called with `--utc`）。（默认：false） |

### 全局配置参数

| 名称 | 描述 |
|---|---|
| config.my-cnf | 用来读取数据库认证信息的配置文件 `.my.cnf` 位置。（默认：`~/.my.cnf`） |
| log.level | 日志级别。（默认：info） |
| exporter.lock_wait_timeout | 为链接设置 lock_wait_timeout（单位：秒）以避免对元数据的锁时间太长。（默认：2） |
| exporter.log_slow_filter | 添加 log_slow_filter 以避免抓取的慢查询被记录。<br>**说明：**<br>不支持 Oracle MySQL。 |
| web.listen-address | web 端口监听地址。 |
| web.telemetry-path | metrics 接口路径。 |

| | |
|---|---|
| version | 打印版本信息。 |

## heartbeat 心跳检测

如果开启 `collect.heartbeat`，mysqld_exporter 会通过心跳检测机制抓取复制延迟数据。

# SQL Server Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

Microsoft SQL Server Exporter 是社区专门为采集 Microsoft SQL Server（简称：MSSQL）数据库监控指标而设计开发，通过 MSSQL Exporter 上报性能和健康状况等的指标数据，用于监控大盘展示和异常报警。腾讯云可观测平台 Prometheus 提供了与 MSSQL Exporter 集成及开箱即用的 Grafana 监控大盘。

> **说明：**
> 如果需要监控的 MSSQL 是腾讯云 云数据库 SQL Server，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **MSSQL**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


#### 配置说明

| 参数 | 说明 |
| --- | --- |

| | |
|---|---|
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | MSSQL 的用户名称。 |
| 密码 | MSSQL 的密码。 |
| 域名 | MSSQL 的服务域名。 |
| 地址 | MSSQL 的服务端口。 |
| 标签 | 给指标添加自定义 Label。 |

## 方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 **Prometheus 监控服务控制台** > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 MSSQL Exporter > 验证 步骤完成 Exporter 部署。

#### 步骤二：部署 MSSQL Exporter

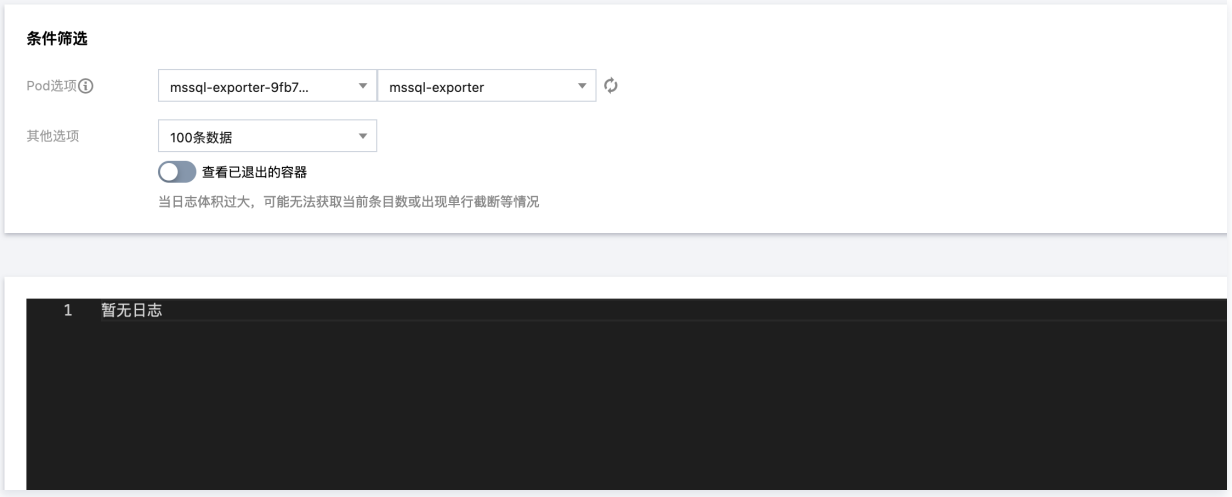
1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
  name: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
  namespace: mssql-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
  template:
    metadata:
      labels:
        k8s-app: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
    spec:
      containers:
      - env:
        - name: SERVER
          value: "127.0.0.1"  # 根据业务需要调整成对应的 MSSQL 的域名
```

```
      - name: PORT
        value: "1433"  # 根据业务需要调整成对应的 MSSQL 的端口
      - name: USERNAME
        value: user  # 根据业务需要调整成对应的 MSSQL 的用户名
      - name: PASSWORD
        value: "123456"  # 根据业务需要调整成对应的 MSSQL 的密码
      - name: EXPOSE
        value: "4000"  # 暴露指标端口，根据业务需要调整成对应的端口
      image: ccr.ccs.tencentyun.com/rig-agent/common-image:mssql-exporter-v1.3.0
      imagePullPolicy: IfNotPresent
      name: mssql-exporter
      ports:
      - containerPort: 4000  # 开放上述环境变量 EXPOSE 所对应的端口
        name: metric-port
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```

**步骤三：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，无报错信息输出即可，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 MSSQL 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:4000/metrics
```

执行结果如下图所示：

```
# HELP mssql_batch_requests Number of Transact-SQL command batches received per second. This statistic is affected by all constraints (such as I/O, number of users, cachesize, complexity of requests, and so on). High batch requests
 mean good throughput
# TYPE mssql_batch_requests gauge
mssql_batch_requests 5109759

# HELP mssql_transactions Number of transactions started for the database per second. Transactions/sec does not count XTP-only transactions (transactions started by a natively compiled stored procedure.)
# TYPE mssql_transactions gauge
mssql_transactions{database="grafana"} 205975
mssql_transactions{database="tempdb"} 4671923
mssql_transactions{database="Monitor"} 5715012
mssql_transactions{database="msdb"} 2009624
mssql_transactions{database="model_msdb"} 4747
mssql_transactions{database="model"} 16631
mssql_transactions{database="model_replicatedmaster"} 4746
mssql_transactions{database="mssqlsystemresource"} 220183
mssql_transactions{database="master"} 533839

# HELP mssql_page_fault_count Number of page faults since last restart
# TYPE mssql_page_fault_count gauge
mssql_page_fault_count 7440578

# HELP mssql_memory_utilization_percentage Percentage of memory utilization
# TYPE mssql_memory_utilization_percentage gauge
mssql_memory_utilization_percentage 93
```

**步骤四：添加采集任务**

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: mssql-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 MSSQL 实例 ID
  namespaceSelector:    # 选择要监控 mssql exporter pod所在的namespace
    matchNames:
    - mssql-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: mssql-exporter
```

## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 database 文件夹中找到 mssql 实例监控面板，查看实例相关监控数据，如下图所示：


## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

## 附录：MSSQL Exporter 环境变量配置

| 名称 | 描述 |
| --- | --- |
| SERVER | 必需，MSSQL 服务 IP 或 域名 |
| PORT | MSSQL 服务端口，默认1433 |
| USERNAME | 必需，MSSQL 服务用户名 |
| PASSWORD | 必需，MSSQL 服务密码 |
| ENCRYPT | 强制加密设置，默认为true |
| TRUST_SERVER_CERTIFICATE | 是否信任服务器的证书设置，默认为true |
| DEBUG | 逗号分割的启用日志列表，可选 currently supports app 和 metrics |

# Oracle DB Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

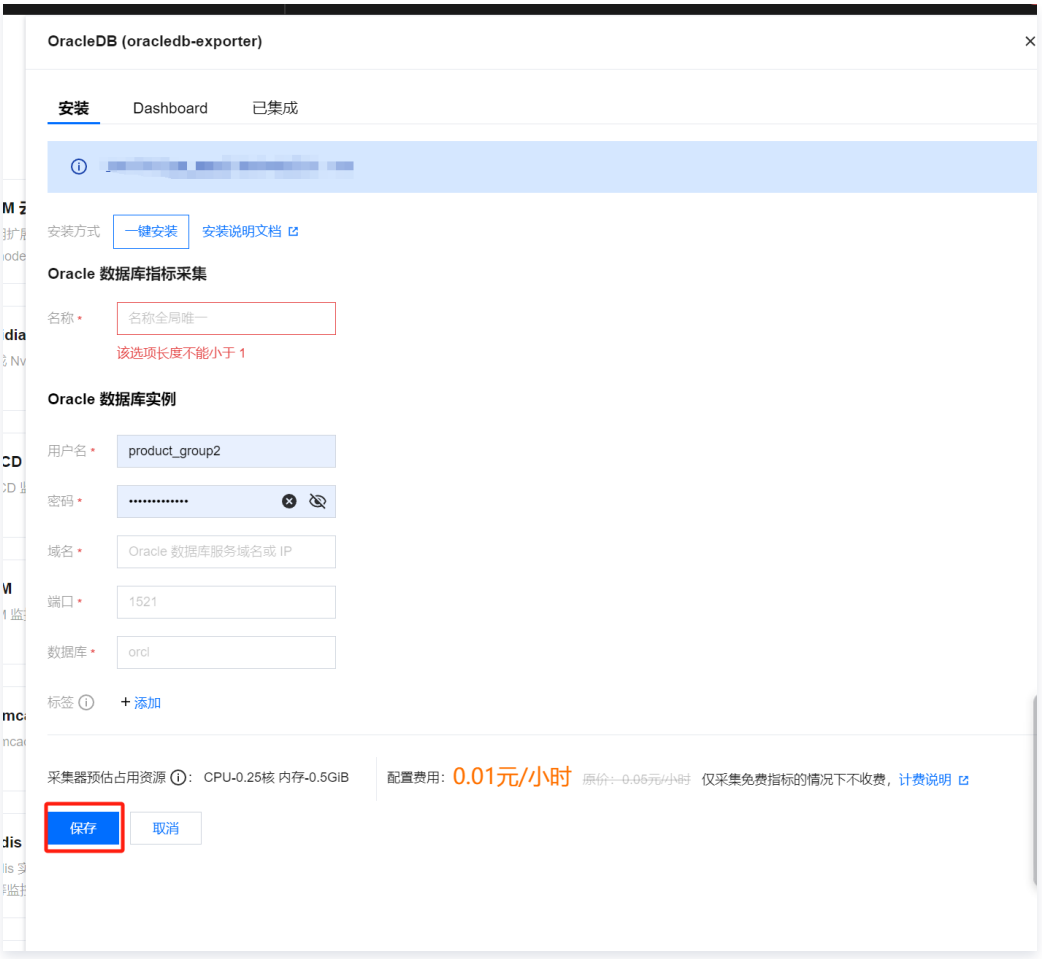
OracleDB Exporter 用于从 Oracle 数据库中抓取指标，并将其通过 Prometheus 指标的方式向外暴露的开源组件，通过该 Exporter 上报的性能、负载及健康状况等指标数据，用于监控大盘展示和异常报警。腾讯云可观测平台 Prometheus 提供了与 OracleDB Exporter 集成及开箱即用的 Grafana 监控大盘。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **OracleDB**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。




#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |

| | |
|---|---|
| 用户名 | OracleDB 的用户名称。 |
| 密码 | OracleDB 的密码。 |
| 域名 | OracleDB 的服务域名。 |
| 端口 | OracleDB 的服务端口。 |
| 数据库 | OracleDB 的数据库名称。 |
| 标签 | 给指标添加自定义 Label。 |

## 方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间。
- 在 Prometheus 监控控制台 > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 OracleDB 连接串 > 部署 OracleDB Exporter > 验证 步骤完成 Exporter 部署。

#### 步骤二：使用 Secret 管理 OracleDB 连接串

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理连接串，并对连接串进行加密处理，在启动 OracleDB Exporter 的时候直接使用 Secret Key，需要调整对应的**连接串**，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: oracledb-secret-test  # 根据业务需要调整成对应的名称
  namespace: oracledb-demo  # 根据业务需要调整成对应的命名空间
type: Opaque
stringData:
  datasource: "oracle://test:123456/127.0.0.1:1521/ORCLPDB1"  # 对应 OracleDB 连接串信息
  # test为用户名，123456为用户密码，127.0.0.1为数据库IP或者域名，1521为数据库端口，ORCLPDB1为数据库名称
```

#### 步骤三：部署 OracleDB Exporter

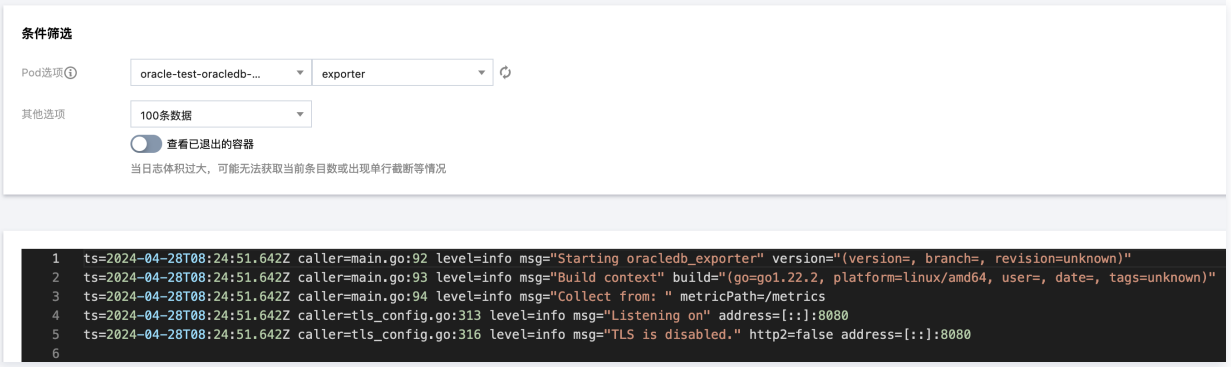
1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
```

```
  name: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
  namespace: oracledb-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
  template:
    metadata:
      labels:
        k8s-app: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
    spec:
      containers:
      - args:
        - --web.listen-address=:8080
        envFrom:
        - secretRef:
            name: oracledb-secret-test  # 上一步创建出来的Secret名称
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:oracledb-exporter-v0.6.0-alpine
        imagePullPolicy: IfNotPresent
        name: oracledb-exporter
        ports:
        - containerPort: 8080
          name: metrics
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

**步骤四：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，无报错信息输出即可，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 OracleDB 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:8080/metrics
```

执行结果如下图所示：

```
# HELP oracledb_tablespace_bytes Generic counter metric of tablespaces bytes in Oracle.
# TYPE oracledb_tablespace_bytes gauge
oracledb_tablespace_bytes{tablespace="SYSAUX",type="PERMANENT"} 5.0397184e+08
oracledb_tablespace_bytes{tablespace="SYSTEM",type="PERMANENT"} 2.56049152e+08
oracledb_tablespace_bytes{tablespace="TEMP",type="TEMPORARY"} 0
oracledb_tablespace_bytes{tablespace="USERS",type="PERMANENT"} 1.048576e+06
# HELP oracledb_tablespace_free Generic counter metric of tablespaces free bytes in Oracle.
# TYPE oracledb_tablespace_free gauge
oracledb_tablespace_free{tablespace="SYSAUX",type="PERMANENT"} 9.05609216e+09
oracledb_tablespace_free{tablespace="SYSTEM",type="PERMANENT"} 9.031385088e+09
oracledb_tablespace_free{tablespace="TEMP",type="TEMPORARY"} 9.046269952e+09
oracledb_tablespace_free{tablespace="USERS",type="PERMANENT"} 9.029484544e+09
# HELP oracledb_tablespace_max_bytes Generic counter metric of tablespaces max bytes in Oracle.
# TYPE oracledb_tablespace_max_bytes gauge
oracledb_tablespace_max_bytes{tablespace="SYSAUX",type="PERMANENT"} 9.560064e+09
oracledb_tablespace_max_bytes{tablespace="SYSTEM",type="PERMANENT"} 9.28743424e+09
oracledb_tablespace_max_bytes{tablespace="TEMP",type="TEMPORARY"} 9.046269952e+09
oracledb_tablespace_max_bytes{tablespace="USERS",type="PERMANENT"} 9.03053312e+09
# HELP oracledb_tablespace_used_percent Gauge metric showing as a percentage of how much of the tablespace has been used.
# TYPE oracledb_tablespace_used_percent gauge
oracledb_tablespace_used_percent{tablespace="SYSAUX",type="PERMANENT"} 5.271636675235647
oracledb_tablespace_used_percent{tablespace="SYSTEM",type="PERMANENT"} 2.756941749285538
oracledb_tablespace_used_percent{tablespace="TEMP",type="TEMPORARY"} 0
oracledb_tablespace_used_percent{tablespace="USERS",type="PERMANENT"} 0.011611451794332161
```

**步骤五：添加采集任务**

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: oracledb-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 OracleDB 实例 ID
  namespaceSelector:    # 选择要监控 oracledb exporter pod所在的namespace
    matchNames:
    - oracledb-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: oracledb-exporter
```

## 查看监控

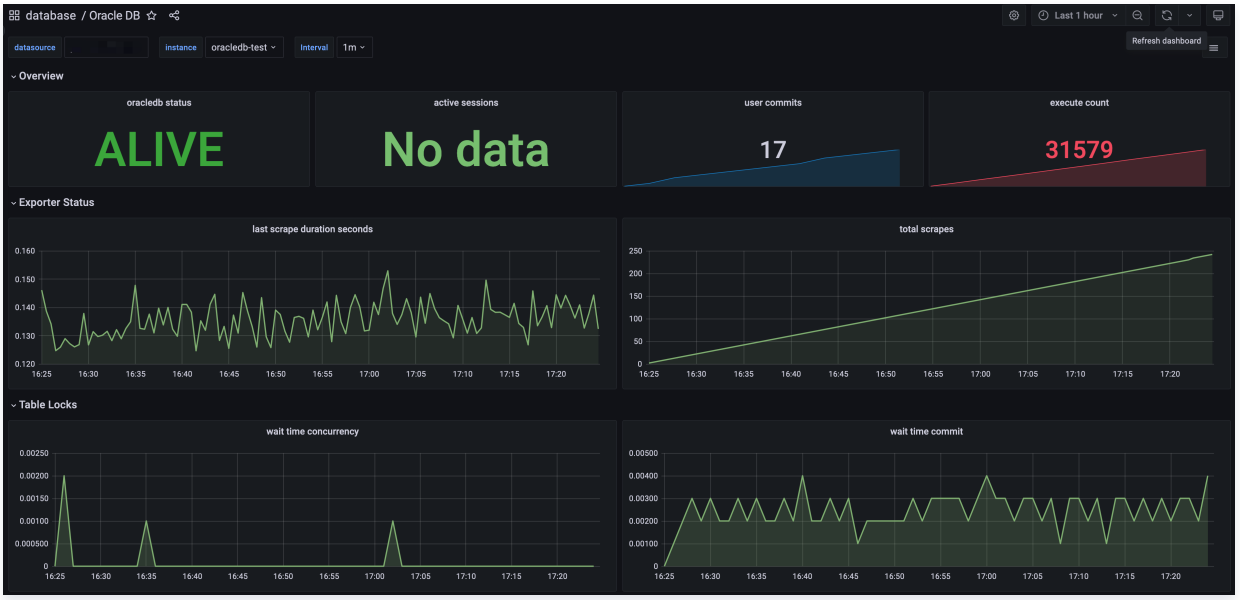
### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。

2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 database 文件夹中找到 Oracle DB 实例监控面板，查看实例相关监控数据，如下图所示：




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

## 附录：OracleDB Exporter 环境变量配置

| 名称 | 描述 |
| --- | --- |
| web.telemetry-path | 指标暴露路径，默认 `/metrics` 。 |
| web.systemd-socket | 使用 systemd 套接字监听器代替端口监听器（仅限 Linux）。 |
| web.listen-address | 监听地址，默认：9161。 |
| web.config.file | 配置文件的路径，可以启用 TLS 或身份验证。 |
| log.level | 日志级别，可选值列表[debug, info, warn, error, fatal]。 |
| log.format | 日志消息的输出格式，示例 logger:syslog?appname=bob&local=7 或 logger:stdout?json=true，默认 stderr。 |
| custom.metrics | 自定义指标配置路径。 |
| default.metrics | 默认指标配置路径。 |
| database.maxIdleConns | 最大空闲连接数，默认0。 |
| database.maxOpenConns | 最大打开连接数，默认0。 |
| database.dsn | 数据库 dsn 串。 |
| database.dsnFile | 读取 dsn 串的文件。 |
| query.timeout | 采集查询超时，默认5s。 |
| scrape.interval | 抓取间隔设置。 |

# Consul Exporter 接入

最近更新时间：2024-10-24 16:17:22

## 操作场景

在使用 Consul 过程中需要对 Consul 运行状态进行监控，以便了解 Consul 服务是否运行正常，排查 Consul 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Consul 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务 Consul。

## 操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Consul**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


## 配置说明

| 名称 | 描述 |
| --- | --- |
| 名称 | 每个集成需要一个唯一名称 |
| 地址 | 要采集的 Consul 实例的地址和端口 |
| 标签 | 增加具有业务含义的标签，会自动添加到 Prometheus 的 Label 中 |

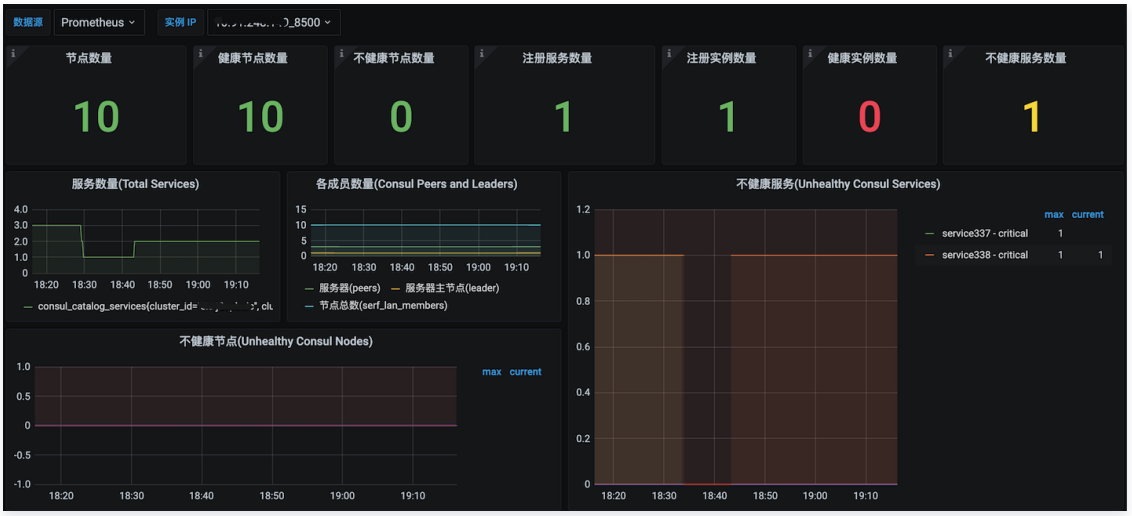
## 查看监控

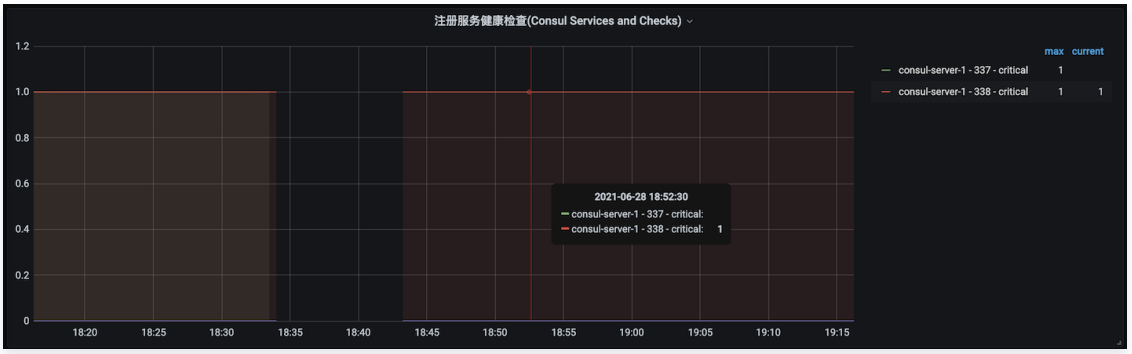
### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台 ，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，在集成中心页面找到 Consul 监控，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可通过监控大盘清晰看到如下监控状态：
   - Consul 集群节点状态
   - Consul 上注册服务的状态


数据源
Prometheus
实例 IP
节点数量
10
健康节点数量
10
不健康节点数量
0
注册服务数量
1
注册实例数量
1
健康实例数量
0
不健康服务数量
1
服务数量(Total Services)
各成员数量(Consul Peers and Leaders)
服务器(peers)
服务器主节点(leader)
节点总数(serf_lan_members)
不健康服务(Unhealthy Consul Services)
max current
service337 - critical 1
service338 - critical 1 1
不健康节点(Unhealthy Consul Nodes)
max current


注册服务健康检查(Consul Services and Checks)
max current
consul-server-1 - 337 - critical 1
consul-server-1 - 338 - critical 1 1
2021-06-28 18:52:30
consul-server-1 - 337 - critical:
consul-server-1 - 338 - critical: 1

# Ingress NGINX Controller Exporter 接入

最近更新时间：2024-10-24 16:17:22

## 操作场景

在使用 Ingress NGINX Controller 过程中需要对 Ingress NGINX Controller 运行状态进行监控，以便了解 Ingress NGINX Controller 服务是否运行正常，排查 Ingress NGINX Controller 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Ingress NGINX Controller 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务 Ingress NGINX Controller。

## 操作步骤

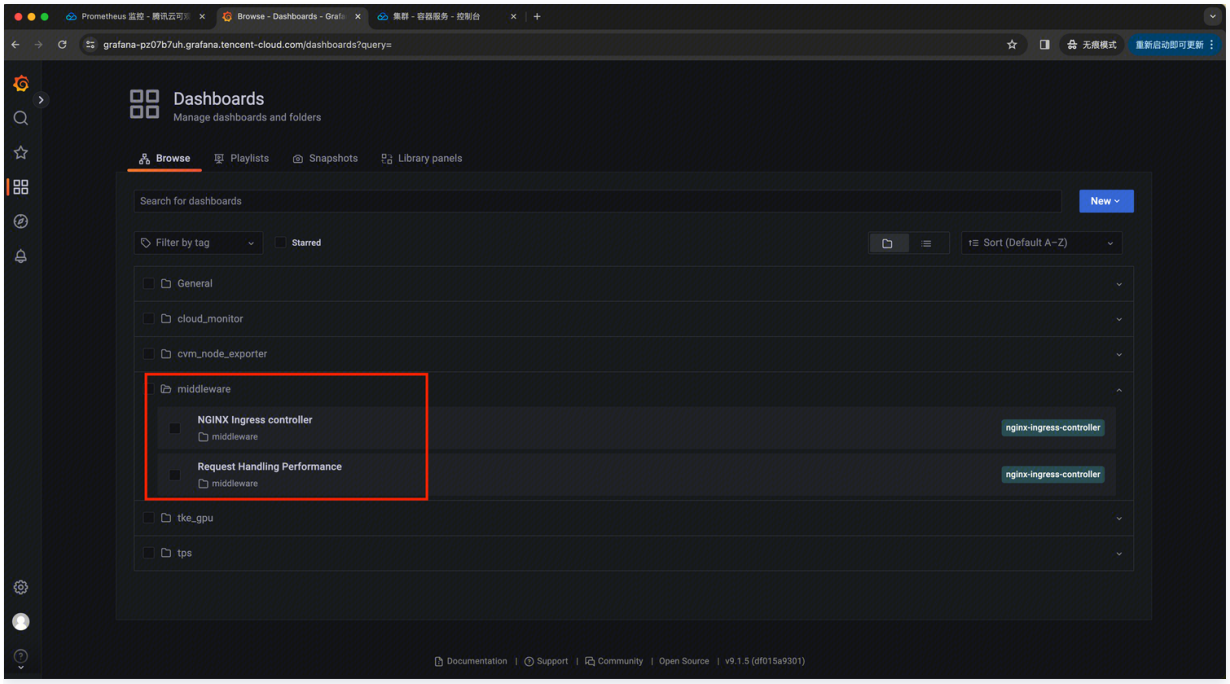
1. 登录 Prometheus 控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Ingress NGINX Controller**，即会弹出一个安装窗口，在安装页面填写集成名称，选取待监控的 nginx-ingress controller 所在的集群以及它的实例名，然后单击**保存**。


## 配置说明

| 名称 | 描述 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 集群 | 选取待监控的 nginx-ingress controller 所在的集群 |
| 实例 | 选取待监控的 nginx-ingress controller 所在的实例名 |

## 查看监控

待部署成功后（1分钟之内），在 Prometheus 相关联的 Grafana 里，找到 nginx-ingress 相关面板，即可观察上述 nginx-ingress controller 的 dashboards。


Dashboards
Manage dashboards and folders
Browse
Playlists
Snapshots
Library panels
Search for dashboards
New
Filter by tag
Starred
Sort (Default A–Z)
General
cloud_monitor
cvm_node_exporter
middleware
NGINX Ingress controller
middleware
nginx-ingress-controller
Request Handling Performance
middleware
nginx-ingress-controller
tke_gpu
tps
Documentation
Support
Community
Open Source
v9.1.5 (df015a9301)


middleware / NGINX Ingress controller
Controller Request Volume
0.000554 ops/s
Controller Connections
3.47
Controller Success Rate (non-4i5xx responses)
0%
Config Reloads
0
Last Config Failed
N/A
Ingress Request Volume
Ingress Success Rate (non-4i5xx responses)
Network I/O pressure
Average Memory Usage
Average CPU Usage
Ingress Percentile Response Times and Transfer Rates
Ingress
P50 Latency
P90 Latency
P99 Latency
IN
OUT
lyla-route
NaN
NaN
NaN
0.00 B/s
0.00 B/s
Ingress Percentile Response Times (Ingress Namespaces)
Ingress Request Latency Heatmap (Ingress Namespaces)


middleware / Request Handling Performance
Request Latency Percentiles
Upstream Response Latency Percentiles
Request Rate by Method and Path
Median Upstream Response Time by Method and Path
Response Error Rate by Method and Path
Upstream Response Time by Method and Path
Response Error Rate by Method and Path
Average Response Size by Method and Path

# TKE GPU Exporter 接入

最近更新时间：2025-06-11 10:35:21

## 操作场景

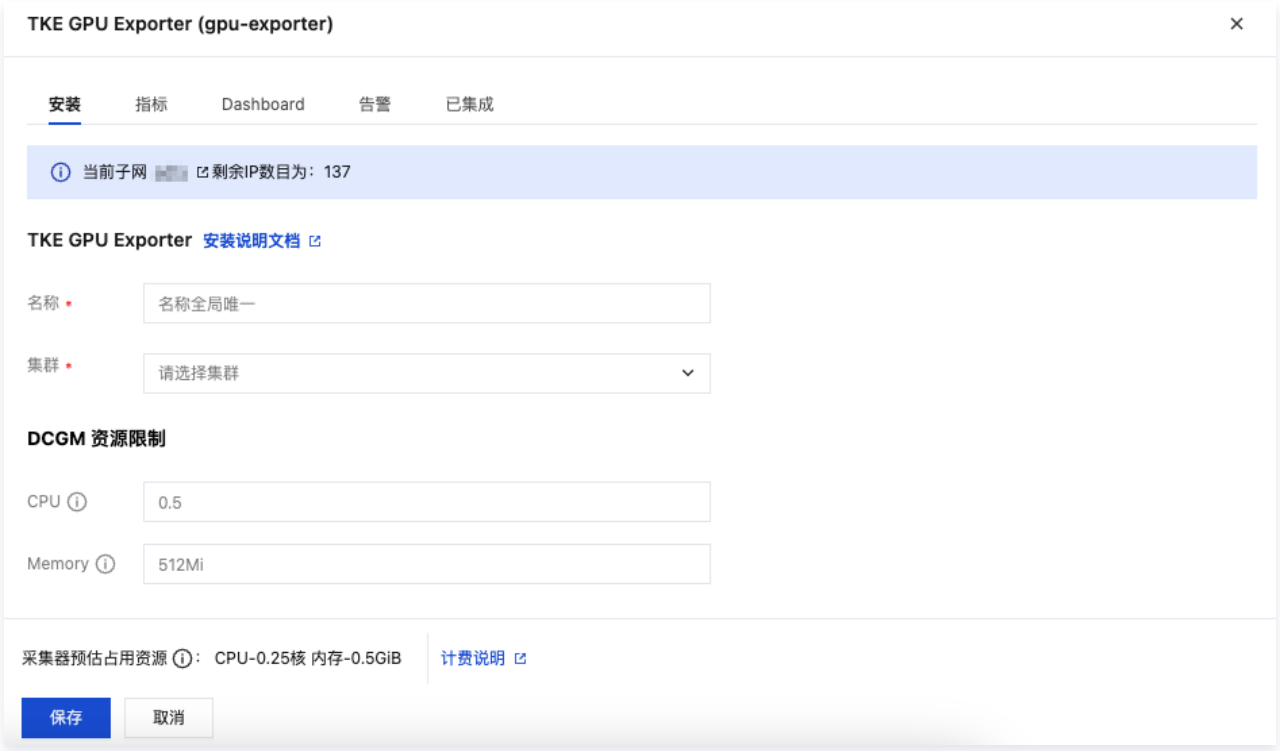
在使用 TKE GPU 资源过程中需要对资源使用状态进行监控，以便了解 GPU 服务是否运行正常，排查 GPU 资源故障。Prometheus 监控服务提供基于 Exporter 的方式来监控容器化 GPU 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务部署 TKE GPU Exporter 以及实现 TKE GPU Exporter 告警接入等操作。

## 前提条件

Prometheus 实例已成功关联待监控的 TKE 集群，步骤参考 集成容器服务 。

## 操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **TKE GPU Exporter**，即会弹出一个安装窗口，在安装页面填写集成名称，选取待监控的 GPU 所在的集群，然后单击**保存**。




## 配置说明

| 名称 | 描述 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 集群 | 选取待监控的 GPU 所在的集群。 |
| CPU | DCGM Exporter 工作负载的 CPU 限制，以 CPU 核心数为单位，需符合 Kubernetes 资源限制格式，具体支持的格式如下：<br>• 小数或整数，例如0.5表示0.5个 CPU 核心。<br>• 毫核值，整数后加单位m，例如500m表示500毫核（等同于0.5核心）。 |
| Memory | DCGM Exporter 工作负载的内存限制，以字节为单位，需符合 Kubernetes 资源限制格式，具体支持的格式如下：<br>• 纯整数，例如1048576表示1048576字节。 |

- 整数后加十进制单位，支持 P、T、G、M、k，例如100k表示100,000字节。
- 整数后加二进制单位，支持Pi、Ti、Gi、Mi、Ki，例如256Mi表示约268,435,456字节。

## 查看监控

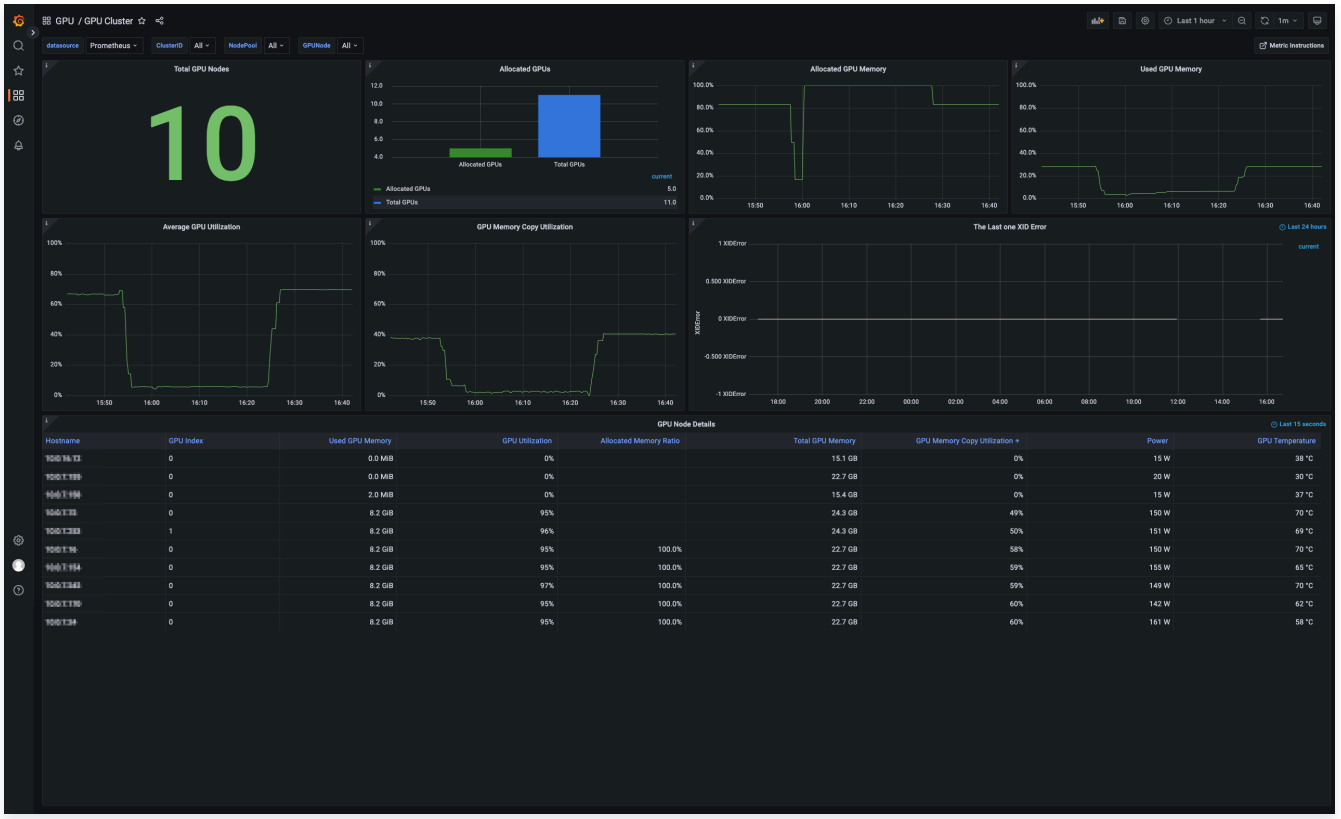
### 前提条件

Prometheus 实例已绑定 Grafana 实例。

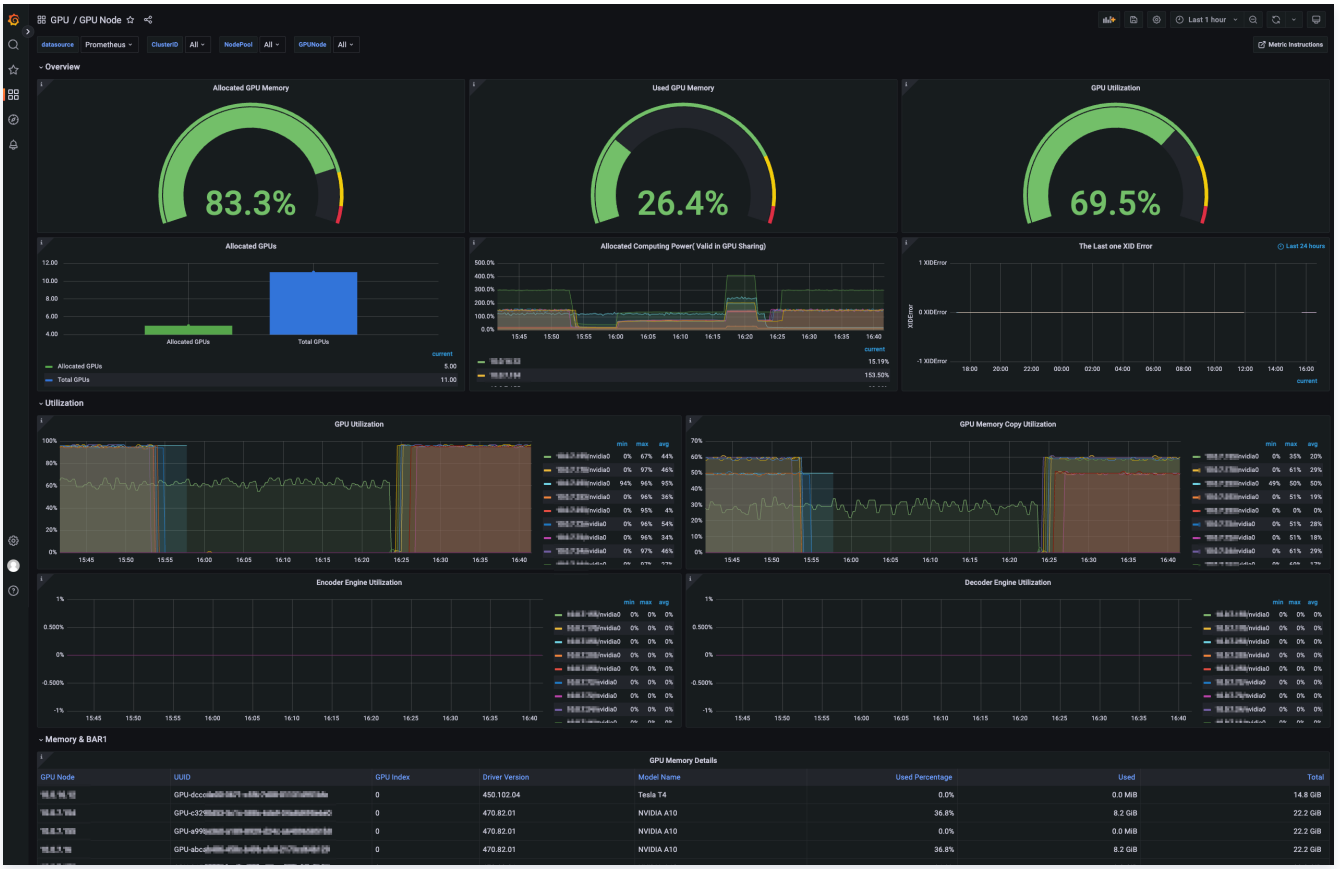
### 操作步骤

登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。

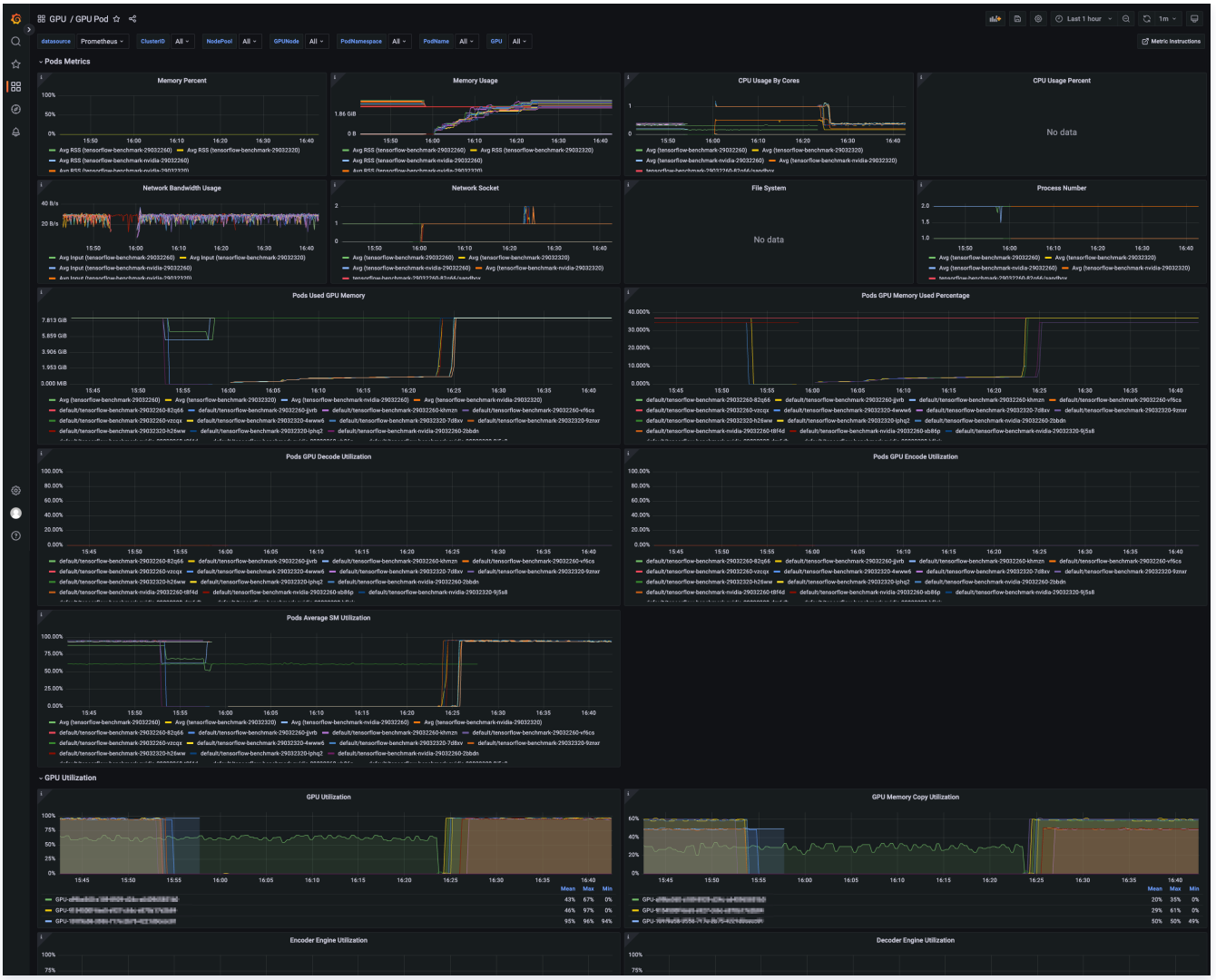
1. 选择**数据采集 > 集成中心**，进入集成中心页面。找到 **TKE GPU Exporter** 集成，选择 **Dashboard > Dashboard 操作 > 安装/升级 Dashboard**，单击**安装/升级**，安装对应的 Grafana Dashboard。
2. 选择查看已集成，在已集成列表中单击 Grafana 图标即可自动进入 GPU 监控大盘文件夹，查看实例相关的监控数据，如下图所示：
   - GPU Cluster：以集群的维度查看 GPU 节点状态，例如节点个数、GPU 使用率、GPU 内存、GPU 内存使用率等。




   - GPU Node：以节点池的维度查看 GPU 节点状态，例如节点状态概览、GPU 卡信息、内存使用率、温度与能耗信息等。




- GPU Pod：以 pod 的维度查看 GPU 节点状态，例如使用 GPU 节点的 pod CPU 内存利用率、网络带宽与文件系统监控、GPU 使用率、GPU 内存使用率等。




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，在集成中心页面找到 TKE GPU Exporter 集成，选择**告警 > 告警模板**，单击**展开规则列表**，可查看 TKE GPU 预设告警策略。


**单击配置，选择通知模板并保存即可将预设 GPU 告警策略添加至当前 Prometheus 实例。同时也可根据业务实际的情况来添加自定义告警策略。详情请参见** 新建告警策略 。


配置告警
策略名称
TKE GPU Exporter
收敛时间
5分钟
告警收敛时间对alertmanager渠道不生效。在收敛时间周期内若多次满足告警条件，仅会发送一次通知，若设置为1小时，则1小时内该策略被触发后仅会发送1次告警通知。
告警渠道
腾讯云
Webhook
自建alertmanager
告警通知
选择模板
新建
已选择 0 个通知模板，还可以选择 3 个
通知模板名称
包含操作
操作
当前通知模板列表为空，您可以选择相应的通知模板
请选择告警通知模板
保存
取消

# Memcached Exporter 接入

最近更新时间：2024-10-31 18:11:52

## 操作场景

在使用 Memcached 过程中需要对 Memcached 运行状态进行监控，以便了解 Memcached 服务是否运行正常，排查 Memcached 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Memcached 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务 Memcached。

## 操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心选择 **Memcached** 单击安装进行集成。

## 配置说明


| 名称 | 描述 |
| --- | --- |
| 名称 | 每个集成需要一个唯一名称 |
| 地址 | 要采集的 Memcached 实例的地址和端口 |
| 标签 | 增加具有业务含义的标签，会自动添加到 Prometheus 的 Label 中 |

## 查看监控

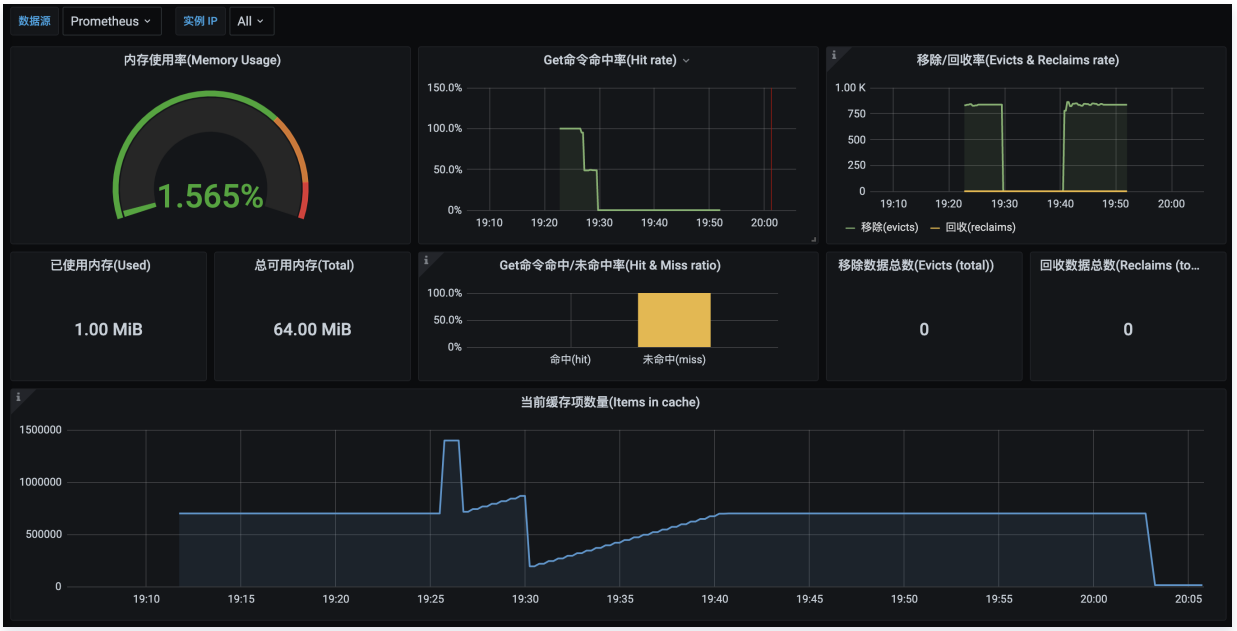
### 前提条件

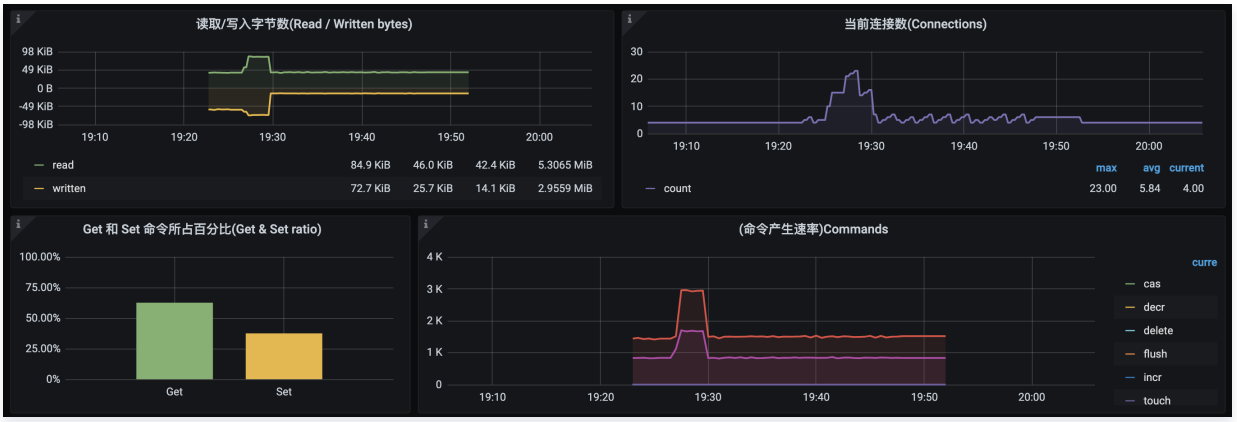
Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台 ，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，在集成中心页面找到 Memcached 监控，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可通过监控大盘清晰看到如下监控状态：
   - 内存使用率，同时显示已使用的内存和内存总量。
   - 当前的 Get 命令的命中率，同时显示服务运行期间 Get 命令命中和未命中的比例。
   - Memcached 移除旧数据和回收过期数据的速率，同时显示服务运行期间移除和回收的数据总数。
   - Memcached 存储的数据总量。
   - 从网络中读取和写入的字节数。

- 当前打开的连接数。
- 服务运行期间 Get 命令和 Set 命令的比例。
- 当前各命令的产生速率。



# Apache Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

Apache Exporter 是一种用于收集和公开 Apache HTTP 服务器指标的工具，Exporter 上报的核心指标，用于异常报警和监控大盘展示。腾讯云可观测平台 Prometheus 提供了 Apache Exporter 集成与开箱即用的 Grafana 监控大盘。

> **说明：**
> 为了保证 Exporter 能够采集数据，需要确保 Apache 服务已打开 服务状态 相关内容。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **Apache**，找到后单击它即会弹出一个安装窗口。
5. 在弹出窗口的安装页面，填写指标采集名称、地址、路径等信息，并单击**保存**。


#### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 地址 | Apache HTTP 服务的连接地址。 |

| | |
|---|---|
| 路径 | Apache HTTP 服务的服务状态路径，默认为 `/server-status` |
| 用户名 | Apache HTTP 服务的用户名称。 |
| 密码 | Apache HTTP 服务的密码。 |
| 标签 | 给指标添加自定义 Label。 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台 > **选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

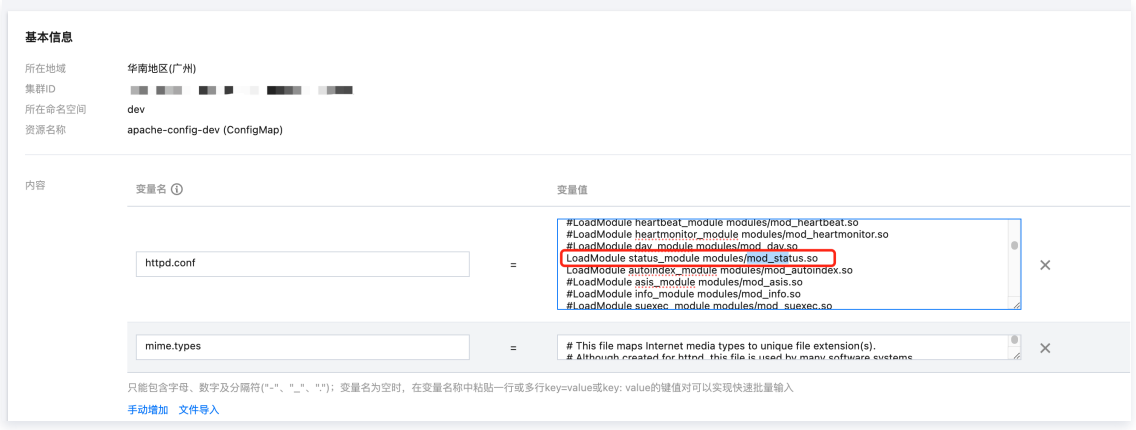
### 操作步骤

#### 步骤一：Apache 服务开启 mod_status 模块

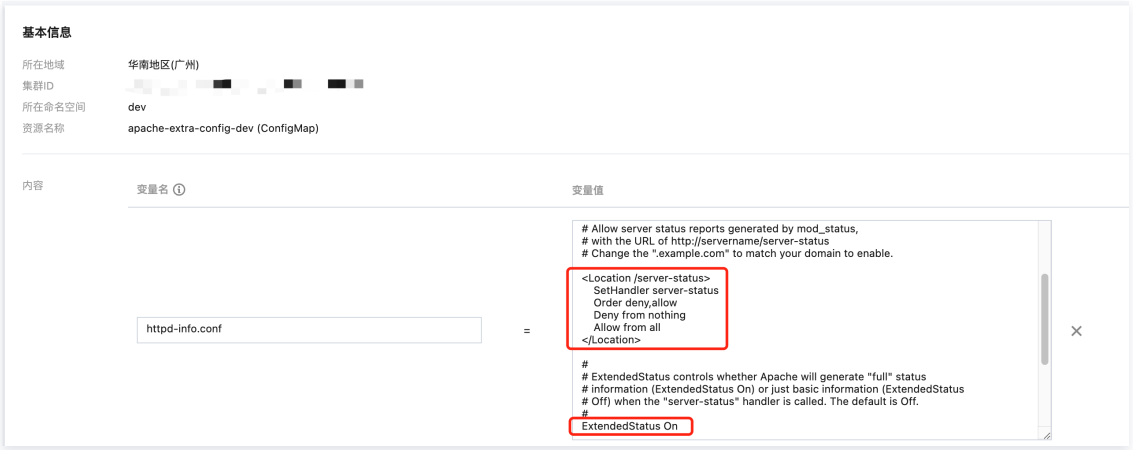
> **说明：**
> 容器服务相关操作可参见 容器服务 相关文档。

因为 Apache Exporter 是通过 Apache 的 mod_status 模块对其进行监控，所以需要确保 Apache 服务打开了 mod_status 模块，具体步骤如下：

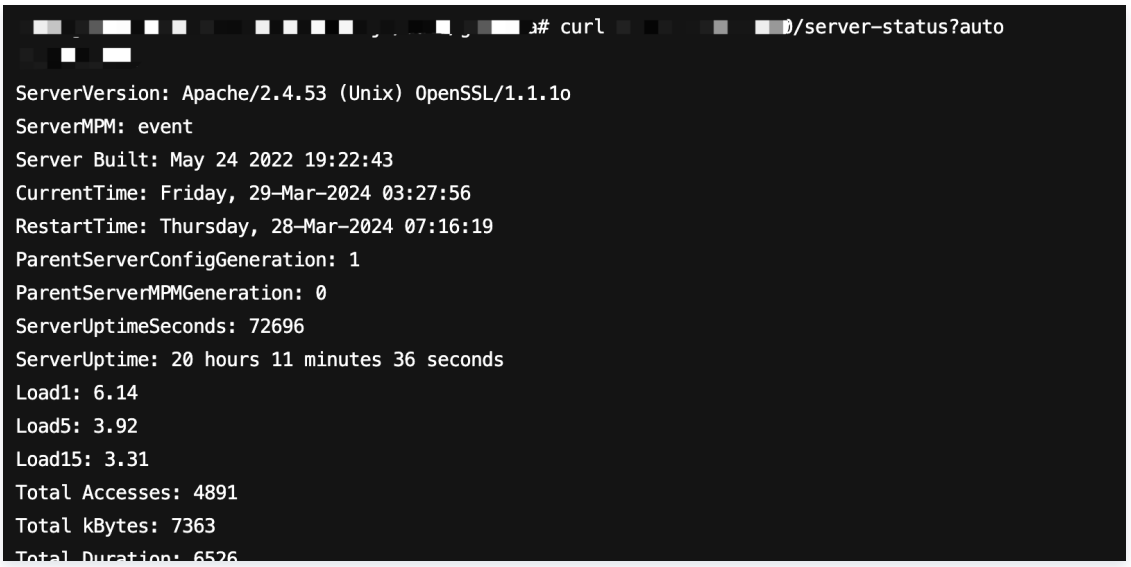
1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**，找到业务 Apache 服务所在集群，进入集群，找到业务 Apache 服务。
3. 若未添加 Apache 服务相关 ConfigMap，登录到业务 Apache 服务，将配置目录下的 httpd.conf、mime.types、extra/httpd-info.conf 等配置信息进行拷贝，创建 ConfigMap，将配置信息添加到该 ConfigMap 中，ConfigMap 相关操作指引见 ConfigMap 管理 。
4. 在 httpd.conf 中将这一行 LoadModule status_module modules/mod_status.so 注释去掉（即去掉最前面的#）。若服务存在 extra 相关配置，则在 httpd.conf 中放开相应配置，方法为去掉相关配置的注释。示例如下图所示：




5. 修改 httpd-info.conf 为业务需要的规则，并放开 ExtendedStatus，若不存在 extra 相关配置，则在 httpd.conf 中修改即可。示例如下图所示：




6. 验证mod_status模块是否开放成功，访问服务的 /server-status 相关接口，若正常返回数据即开放成功。示例如下图所示：




**步骤二：Exporter 部署**

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 Apache Exporter > 验证 步骤完成 Exporter 部署。

**步骤三：部署 Apache Exporter**

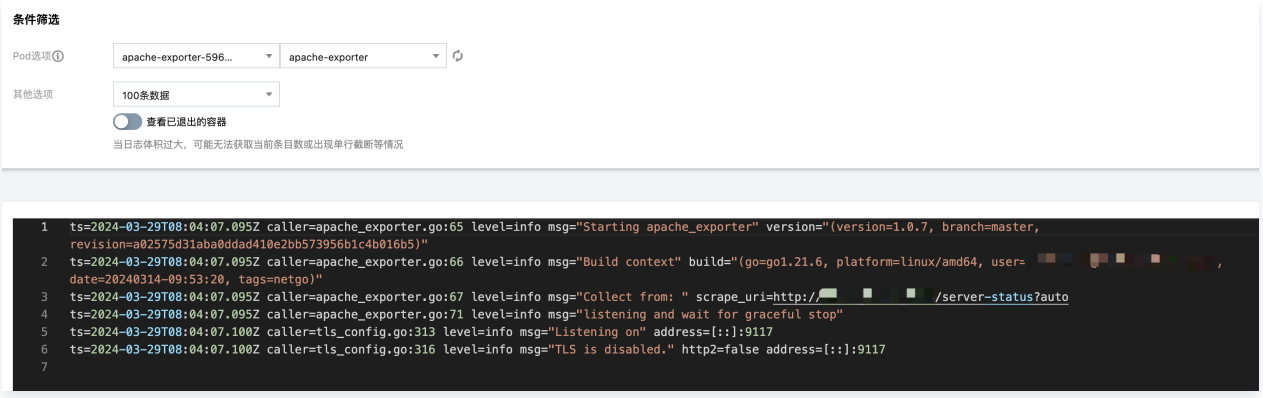
1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter，配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
  name: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
  namespace: apache-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
```

```
  template:
    metadata:
      labels:
        k8s-app: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
    spec:
      containers:
      - args:
        - --web.listen-address=:9117
        - --scrape_uri=http://192.1.1.2:8080/server-status?auto # 根据业务需要调整成 Apache 实例对应地址
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:apache-exporter-v1.0.7
        name: apache-exporter
        ports:
        - containerPort: 9117
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

**步骤四：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 Apache 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
curl localhost:9117/metrics
```

执行结果如下图所示：

```
# HELP apache_accesses_total Current total apache accesses (*)
# TYPE apache_accesses_total counter
apache_accesses_total 6031
# HELP apache_connections Apache connection statuses
# TYPE apache_connections gauge
apache_connections{state="closing"} 1
apache_connections{state="keepalive"} 0
apache_connections{state="total"} 1
apache_connections{state="writing"} 0
# HELP apache_cpu_time_ms_total Apache CPU time
# TYPE apache_cpu_time_ms_total counter
apache_cpu_time_ms_total{type="system"} 3670
apache_cpu_time_ms_total{type="user"} 4019.9999999999995
# HELP apache_cpuload The current percentage CPU used by each worker and in total by all workers combined (*)
# TYPE apache_cpuload gauge
apache_cpuload 0.00858278
# HELP apache_duration_ms_total Total duration of all registered requests in ms
# TYPE apache_duration_ms_total counter
apache_duration_ms_total 8188
# HELP apache_exporter_build_info A metric with a constant '1' value labeled by version, revision, branch, goversion from which apache_exporter was built, and the goos and goarch for the build.
# TYPE apache_exporter_build_info gauge
apache_exporter_build_info{branch="master",goarch="amd64",goos="linux",goversion="go1.21.6",revision="a02575d31aba0ddad410e2bb573956b1c4b016b5",tags="netgo",version="1.0.7"} 1
# HELP apache_generation Apache restart generation
# TYPE apache_generation gauge
apache_generation{type="config"} 1
apache_generation{type="mpm"} 0
# HELP apache_info Apache version information
# TYPE apache_info gauge
apache_info{mpm="event",version="Apache/2.4.53 (Unix) OpenSSL/1.1.1o"} 1
```

**步骤五：添加采集任务**

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: apache-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Apache 实例信息
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - apache-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: apache-exporter
```

## 查看监控

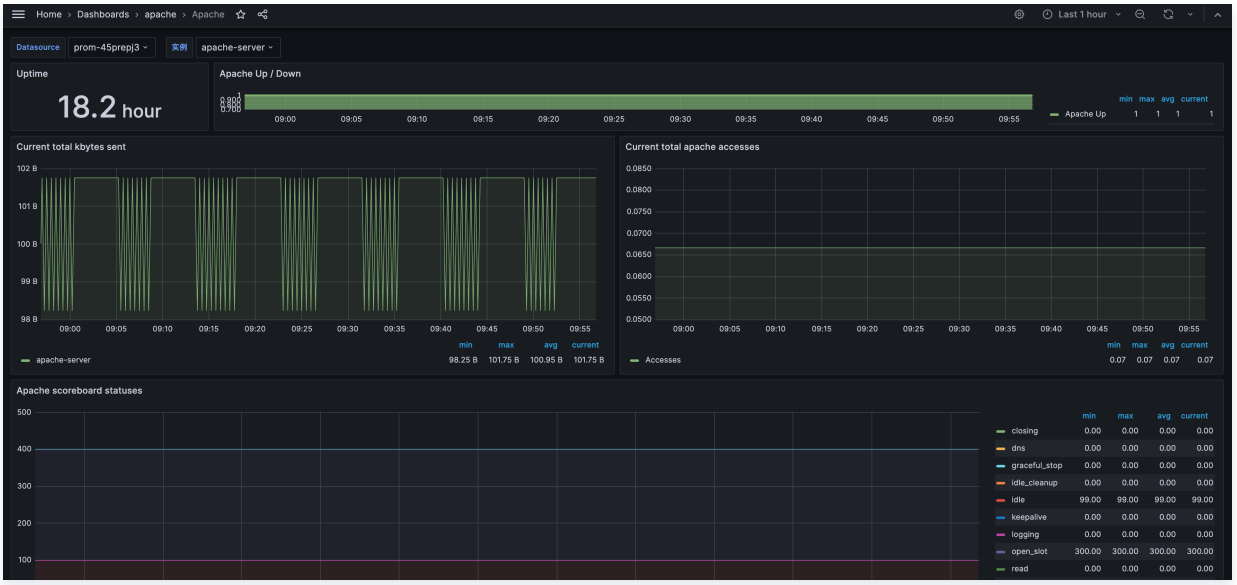
### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。

2. **单击数据采集 > 集成中心，进入集成中心页面。找到 Apache 监控，安装对应的 Grafana Dashboard 即可开启 Apache 监控大盘，查看实例相关的监控数据，如下图所示：**




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

# 附录：Apache Exporter 采集参数说明

## 全局配置参数

| 名称 | 描述 |
| --- | --- |
| telemetry.endpoint | 指标暴露路径，默认 `/metrics` 。 |
| scrape_uri | apache 服务状态页面 url，默认 `http://localhost/server-status/?auto` 。 |
| host_override | 覆盖 HTTP 主机标头，空字符串为没有覆盖。 |
| [no-]insecure | 如果使用 https，则忽略服务器证书。 |
| custom_headers | 将自定义标头添加到 Exporter。 |
| [no-]web.systemd-socket | 使用 systemd 套接字监听器代替端口监听器（仅限 Linux）。 |
| web.listen-address | 监听地址，默认：9117。 |
| web.config.file | 配置文件的路径，可以启用 TLS 或身份验证（实验性参数）。 |
| log.level | 日志级别，默认 info。 |
| log.format | 日志消息的输出格式，取值范围：[logfmt，json]，默认 logfmt。 |
| version | 打印版本信息。 |

# Ceph Exporter 接入

最近更新时间：2024-12-09 18:41:53

## 操作场景

Ceph Exporter 是一个用于 Prometheus 监控系统的插件，用于收集和暴露 Ceph 分布式存储集群的性能指标。它允许用户监视 Ceph 集群的健康状况、性能指标和状态信息，帮助管理员及时发现问题并进行相应的调整和优化。腾讯云可观测平台 Prometheus 提供了与 Ceph Exporter 集成及开箱即用的 Grafana 监控大盘。
Ceph Luminous 12.2.1 之前的版本需要使用该 Exporter 导出指标；Ceph Luminous 12.2.1 的 mgr 中自带了 Prometheus 插件，内置了 Prometheus ceph exporter，可以使用 Ceph mgr 内置的 exporter 作为 Prometheus 的采集目标，故而 Ceph Luminous 12.2.1 及之后的版本更建议使用自带的指标导出。

## 接入方式：Mgr 原生导出

### 前提

Ceph 版本不低于 Luminous 12.2.1。

### 启用 Ceph 的 Prometheus 插件

**查看是否已经开启：**

```
ceph mgr module ls
```

若输出列表中包含 "prometheus" 则说明已开启 prometheus 插件，若不存在则需要开启该插件。
**开启 prometheus 插件：**

```
ceph mgr module enable prometheus
```

**验证：**

```
ceph mgr module ls
```

输出列表中包含 "prometheus"，说明已开启 prometheus 插件。然后登录 mgr pod 验证拉取指标：

```
curl 127.0.0.1:9283/metrics ## 默认为9283端口，按业务实际端口进行调整
```

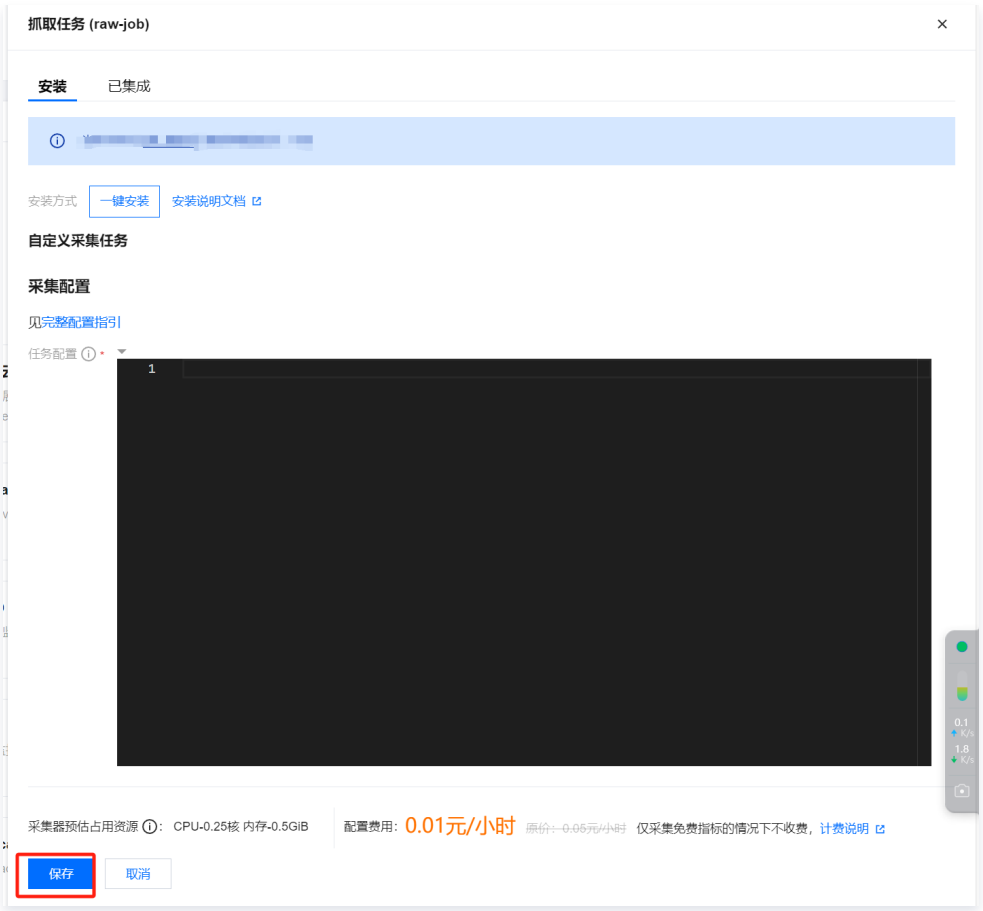
即可看到拉取的指标信息：

```
[root@roo                                    eph]# curl 127.0.0.1:9283/metrics

# HELP ceph_health_status Cluster health status
# TYPE ceph_health_status untyped
ceph_health_status 1.0
# HELP ceph_mon_quorum_status Monitors in quorum
# TYPE ceph_mon_quorum_status gauge
ceph_mon_quorum_status{ceph_daemon="mon.a"} 1.0
ceph_mon_quorum_status{ceph_daemon="mon.b"} 1.0
ceph_mon_quorum_status{ceph_daemon="mon.c"} 1.0
# HELP ceph_fs_metadata FS Metadata
# TYPE ceph_fs_metadata untyped
# HELP ceph_mds_metadata MDS Metadata
# TYPE ceph_mds_metadata untyped
# HELP ceph_mon_metadata MON Metadata
# TYPE ceph_mon_metadata untyped
ceph_mon_metadata{ceph_daemon="mon.a",hostname="172.      ",public_addr="172.      ",rank="0",ceph_version="ceph version 17.2.3 (dff484dfc9e19a9819f375586300b3b79d80034d) quincy (stable)"} 1.0
ceph_mon_metadata{ceph_daemon="mon.b",hostname="172.      ",public_addr="172.      ",rank="1",ceph_version="ceph version 17.2.3 (dff484dfc9e19a9819f375586300b3b79d80034d) quincy (stable)"} 1.0
ceph_mon_metadata{ceph_daemon="mon.c",hostname="172.      ",public_addr="172.      ",rank="2",ceph_version="ceph version 17.2.3 (dff484dfc9e19a9819f375586300b3b79d80034d) quincy (stable)"} 1.0
```

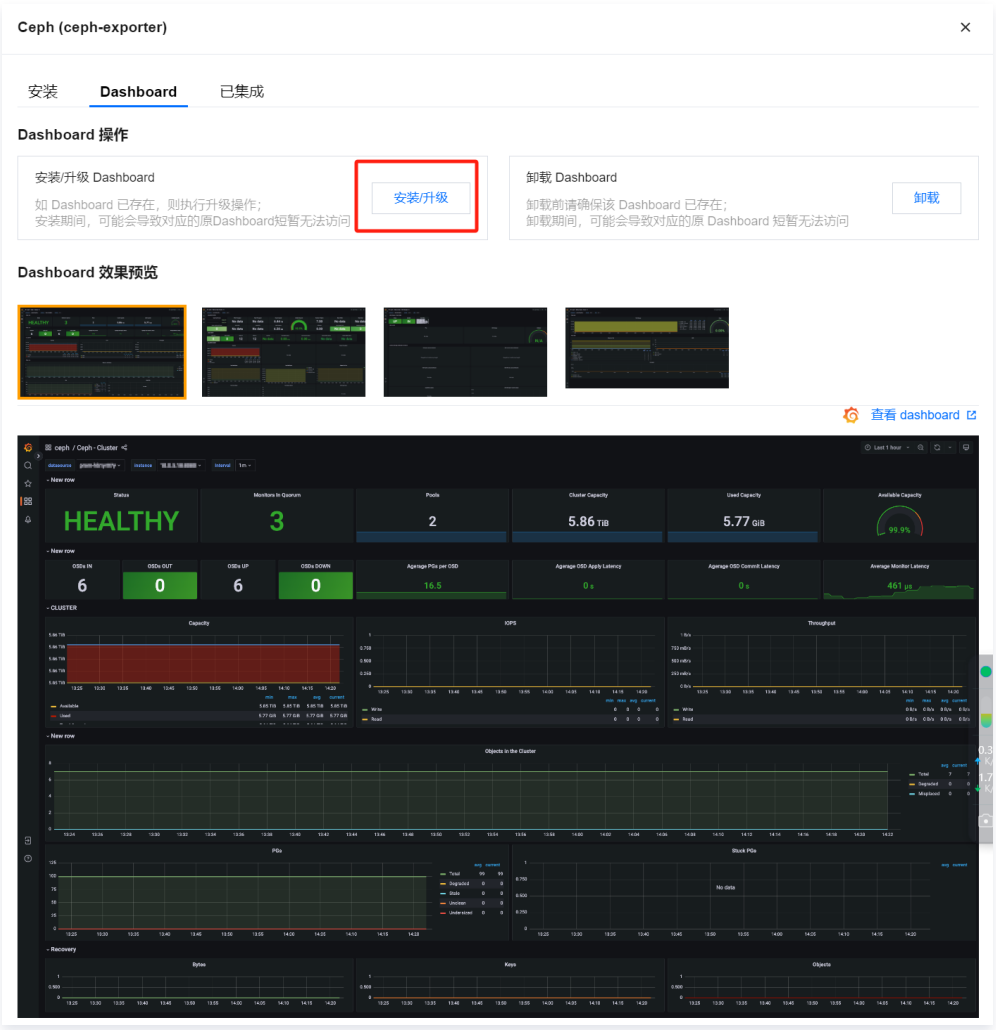
### 指标采集

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。

3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击**抓取任务**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。




5. 在集成中心找到并单击 **ceph**，即会弹出一个窗口，选择 **Dashboard > 安装/升级**。




## 接入方式：开源 Exporter 导出

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **Ceph**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。


### 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 用户名 | Ceph 的用户名称。 |
| 秘钥 | 上述用户名对应的秘钥。 |
| 集群 | Ceph 的集群名称。 |
| 标签 | 给指标添加自定义 Label。 |
| ceph 配置 | 连接 Ceph 所需的 ceph.conf 配置文件 |

## 方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

### 前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台 > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

### 操作步骤

#### 步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 创建配置及鉴权用 Secret > 部署 Ceph Exporter > 验证 步骤完成 Exporter 部署。

#### 步骤二：创建配置及鉴权用 Secret

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. Base64编码生成秘钥环与 ceph.conf。

```
ceph.conf:
[global]
mon_host = 172.18.0.2:6789,172.18.0.3:6789,172.18.0.4:6789
## 该换行不可省略，是必需内容

keyring:
[client.admin]
key = AJKs8dsdfJkEEHxASDanasiKasfdJLYE5G3UAw==
## 该换行不可省略，是必需内容
```

Base64编码后：

```
ceph.conf:
W2dsb2JhbF0KbW9uX2hvc3QgPSAxNzIuMTguMC4yOjY3ODksMTcyLjE4LjAuMzo2Nzg5LDE3Mi4xOC4wLjQ6Njc4OQo=
keyring:
W2NsaWVudC5hZG1pbl0Ka2V5ID0gQUpLczhkc2RmSmtFRUh4QVNEYW5hc2lLYXNmZEpMWUU1RzNVQXc9PQo=
```

3. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置：

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret-test  # 根据业务需要调整成对应的名称
  namespace: ceph-demo  # 根据业务需要调整成对应的命名空间
type: Opaque
data:
  ## ceph.client.user.keyring 中的 user 为鉴权用的用户名，请根据实际业务进行替换，例如
ceph.client.admin.keyring
  ceph.client.user.keyring:
W2NsaWVudC5hZG1pbl0Ka2V5ID0gQUpLczhkc2RmSmtFRUh4QVNEYW5hc2lLYXNmZEpMWUU1RzNVQXc9PQo=
  ceph.conf:
W2dsb2JhbF0KbW9uX2hvc3QgPSAxNzIuMTguMC4yOjY3ODksMTcyLjE4LjAuMzo2Nzg5LDE3Mi4xOC4wLjQ6Njc4OQo=
```

#### 步骤三：部署 Ceph Exporter

在 Deployment 管理页面，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

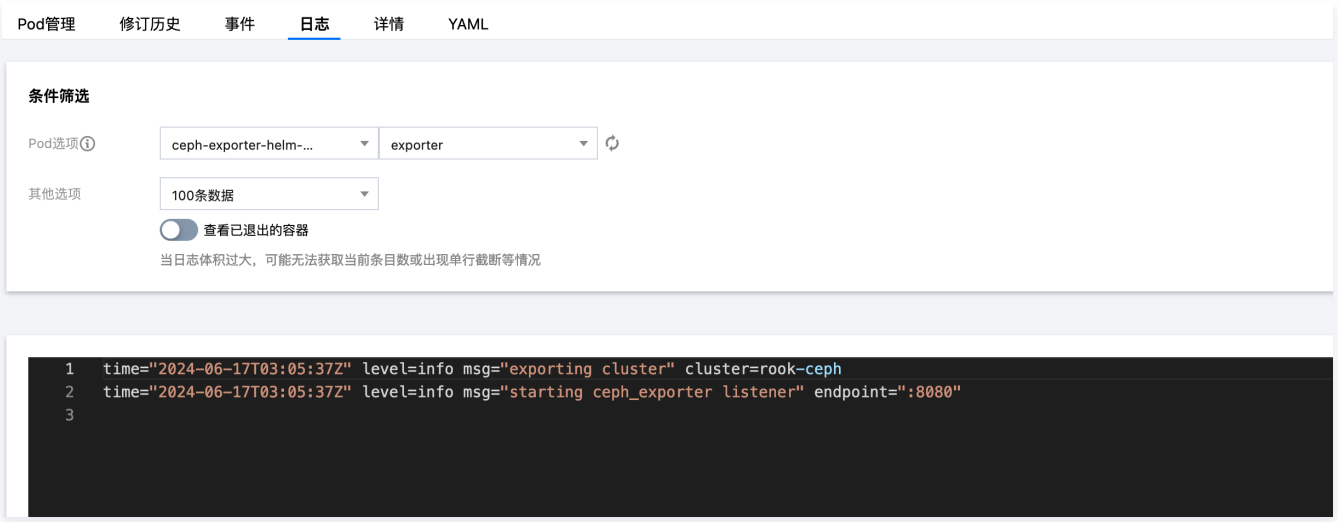
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 Ceph 实例的信息
  name: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 Ceph 实例的信息
  namespace: ceph-demo # 根据业务需要调整成对应的命名空间
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 Ceph 实例的信息
  template:
    metadata:
      labels:
        k8s-app: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 ceph 实例的信息
    spec:
      containers:
      - env:
        - name: CEPH_CLUSTER
          value: ceph  # 根据业务需要调整成相应的集群名称
        - name: CEPH_USER
          value: admin # 根据业务需要调整成相应的用户名
        - name: TELEMETRY_ADDR
          value: :8080
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:ceph-exporter-v4.2.3
        imagePullPolicy: IfNotPresent
        name: ceph-exporter
        ports:
        - containerPort: 8080
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /etc/ceph
          name: conf
          readOnly: true
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      volumes:
      - name: conf
        secret:
          defaultMode: 420
          secretName: ceph-secret-test  # 上面步骤中创建的 Secret 的名称
```

**步骤四：验证**

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：




3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 Ceph 指标。如发现未能得到对应的数据，请检查**秘钥**和 **ceph.conf** 配置是否正确，具体如下：

```
curl localhost:8080/metrics
```

执行结果如下图所示：

```
# HELP ceph_down_pgs No. of PGs in the cluster in down state
# TYPE ceph_down_pgs gauge
ceph_down_pgs{cluster="rook-ceph"} 0
# HELP ceph_features Counts of current client features, parsed from `ceph features`
# TYPE ceph_features gauge
ceph_features{cluster="rook-ceph",daemon="client",features="0x3f01cfbf7ffdffff",release="luminous"} 6
ceph_features{cluster="rook-ceph",daemon="mgr",features="0x3f01cfbf7ffdffff",release="luminous"} 2
ceph_features{cluster="rook-ceph",daemon="mon",features="0x3f01cfbf7ffdffff",release="luminous"} 3
# HELP ceph_forced_backfill_pgs No. of PGs in the cluster with forced_backfill state
# TYPE ceph_forced_backfill_pgs gauge
ceph_forced_backfill_pgs{cluster="rook-ceph"} 0
# HELP ceph_forced_recovery_pgs No. of PGs in the cluster with forced_recovery state
# TYPE ceph_forced_recovery_pgs gauge
ceph_forced_recovery_pgs{cluster="rook-ceph"} 0
# HELP ceph_health_status Health status of Cluster, can vary only between 3 states (err:2, warn:1, ok:0)
# TYPE ceph_health_status gauge
ceph_health_status{cluster="rook-ceph"} 1
# HELP ceph_health_status_interp Health status of Cluster, can vary only between 4 states (err:3, critical_warn:2, soft_warn:1, ok:0)
# TYPE ceph_health_status_interp gauge
ceph_health_status_interp{cluster="rook-ceph"} 1
# HELP ceph_incomplete_pgs No. of PGs in the cluster in incomplete state
# TYPE ceph_incomplete_pgs gauge
ceph_incomplete_pgs{cluster="rook-ceph"} 0
```

**步骤五：添加采集任务**

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: ceph-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
```

```
  - interval: 30s
    port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Ceph 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 Ceph 实例 IP
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - ceph-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: ceph-exporter
```

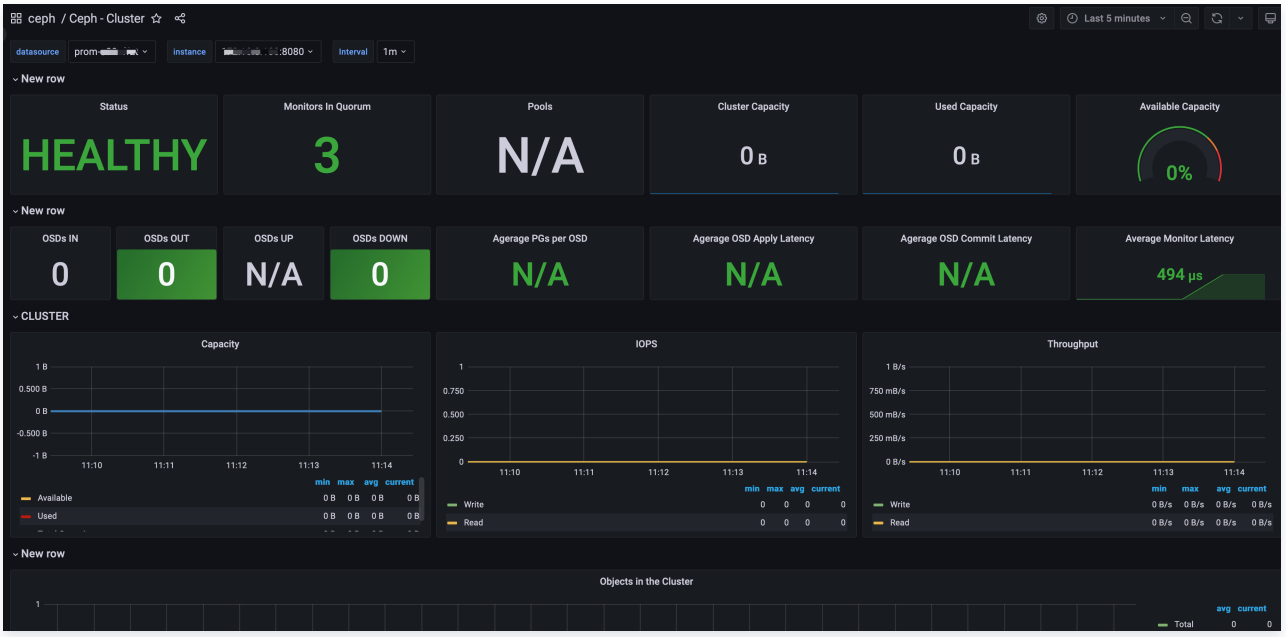
## 查看监控

### 前提条件

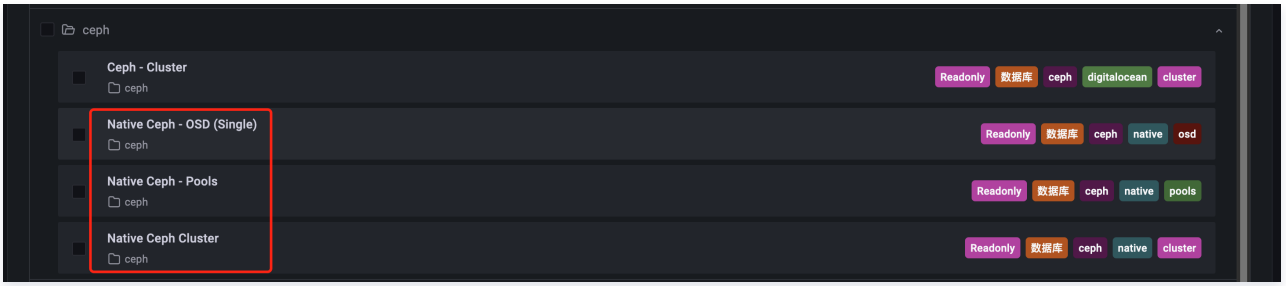
Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

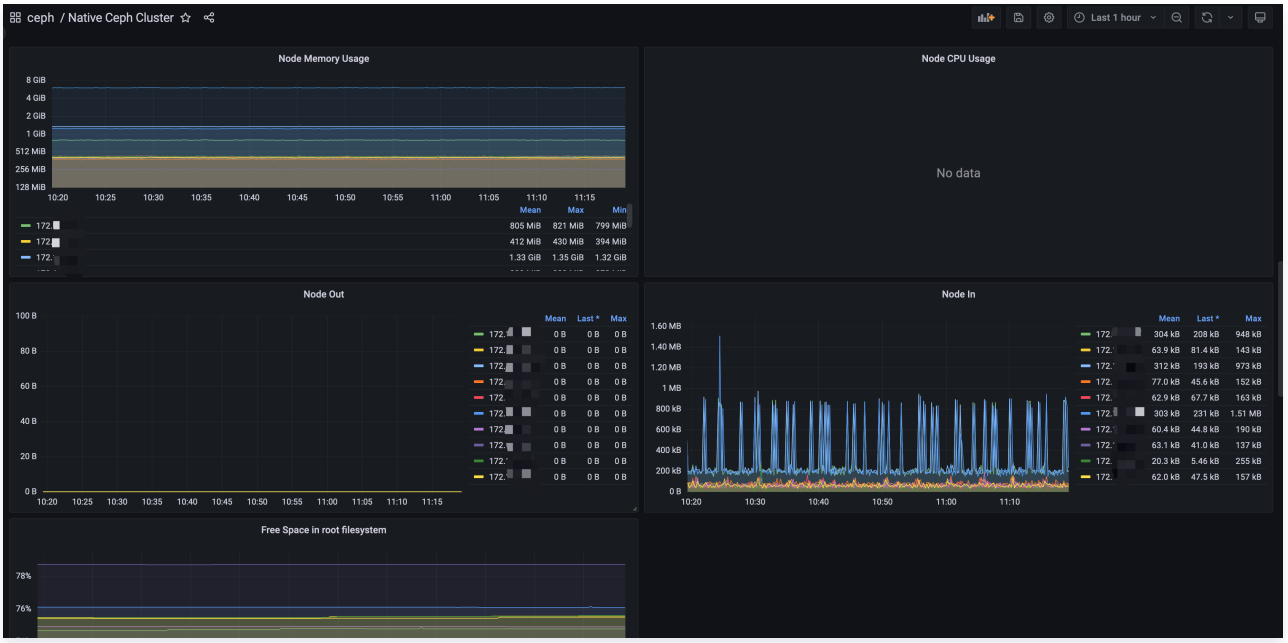
1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例基本信息页面，找到绑定的 Grafana 地址，打开并登录，然后在 ceph 文件夹中找到 Ceph – Cluster 监控面板，查看实例相关监控数据，如下图所示：




3. mgr 原生监控可在如下三个面板中查看：




4. Native Ceph Cluster 面板展示：




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

## 附录：Ceph Exporter 环境变量配置

| 名称 | 描述 |
| --- | --- |
| TELEMETRY_ADDR | 暴露指标地址，默认 *:9128。 |
| TELEMETRY_PATH | 暴露指标路径，默认 /metrics。 |
| EXPORTER_CONFIG | exporter 配置路径，默认 `/etc/ceph/exporter.yml` ，存在该配置时，CEPH_CLUSTER、CEPH_CONFIG、CEPH_USER 等信息都将去其中获取。 |
| RGW_MODE | 启用从 RGW 收集统计数据，取值如下：<br>• 0：禁用（默认值）<br>• 1：启用<br>• 2：后台 |
| CEPH_CLUSTER | Ceph 集群名称，默认 ceph。 |
| CEPH_CONFIG | ceph.conf 配置位置，默认 `/etc/ceph/ceph.conf` 。 |
| CEPH_USER | Ceph 连接集群用户名，默认 admin。 |
| CEPH_RADOS_OP_TIMEOUT | Ceph osd 与 mon 操作超时时长，默认30s。 |
| LOG_LEVEL | 日志等级，取值范围：[trace, debug, info, warn, error, fatal, panic]，默认 info。 |

| | |
|---|---|
| TLS_CERT_FILE_PATH | tls 验证文件路径 |
| TLS_KEY_FILE_PATH | tls 验证文件路径 |

# 其他 Exporter 接入

最近更新时间：2024-08-16 11:52:01

## 操作场景

Prometheus 监控服务目前已经提供了常用基础组件的集成方式，以及开箱即用的监控大屏，由于兼容原生 Prometheus，所以您也可以安装社区其他的 Exporter。

## 操作方式

如果您所使用的基础组件还没有提供相应的集成方式，可以参考如下方式进行集成，以及自定义监控大屏来满足相应的监控需求。

- 开源社区 Exporter 列表
- MySQL 的集成方式

# 健康巡检

最近更新时间：2024-11-08 10:51:21

## 操作场景

通过定期探测对应服务的连通性，来检测其服务的健康情况，帮助您实时的掌握服务的健康状况，及时发现异常，来提升服务的 SLA。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 在实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，单击**数据采集 > 集成中心**。
5. 在集成中心选择**健康巡检** ，单击它即会弹出一个安装窗口。


## 探测说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 每个探测任务需要一个唯一名称，对应 Grafana 监控大屏中的探测分组 |
| 探测方式 | 目前支持如下几种探测方式：<br>• http_get<br>• http_post<br>• tcp<br>• ssh |

| | |
|---|---|
| | • ping |
| 探测目标 | 被探测服务对应地址 |
| 标签 | 增加具有业务含义的标签，会自动添加到 Prometheus 的 Label 中 |

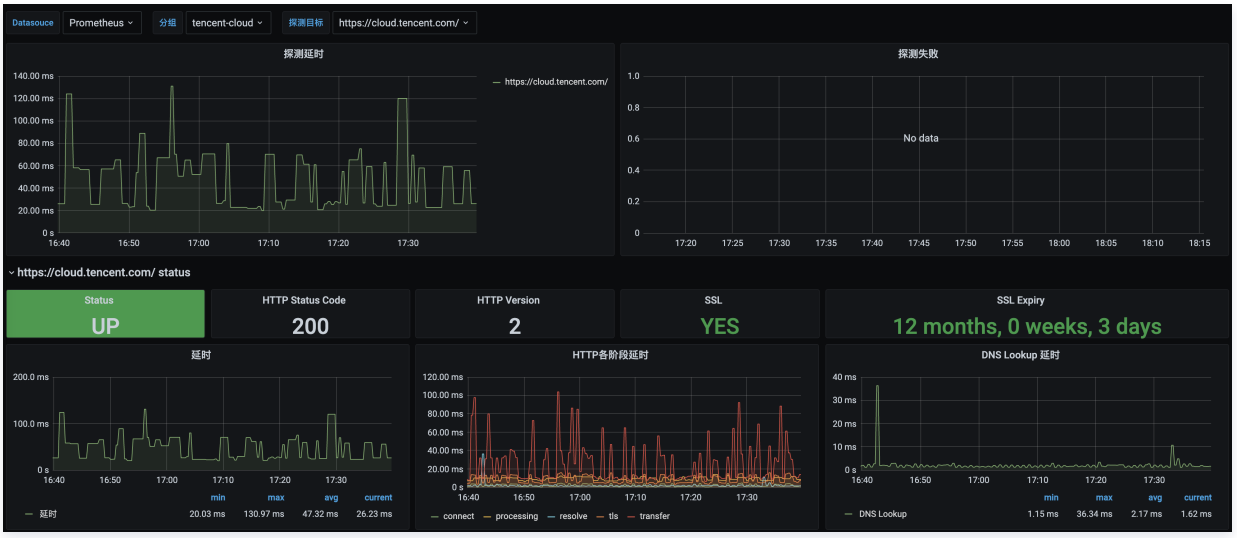
## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，在集成中心页面找到**健康巡检**，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可通过监控大盘清晰看到如下监控状态：
   - 服务访问的延时，是否健康。
   - 服务访问各阶段处理的延时。
   - 如果是 HTTPS 可以监控证书的过期时间。
   - 以及各种探测类型的状态。



# 云监控

最近更新时间：2025-06-09 17:12:22

## 操作场景

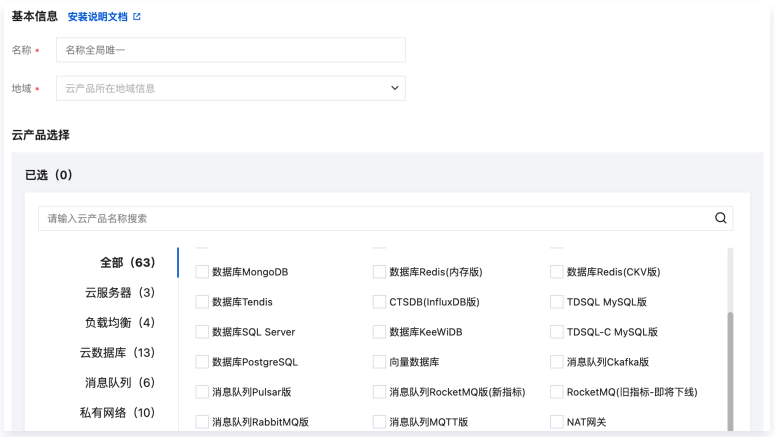
Prometheus 监控服务-云监控模块集成腾讯云产品基础监控数据，通过 Prometheus 监控进行统一采集、存储和可视化。

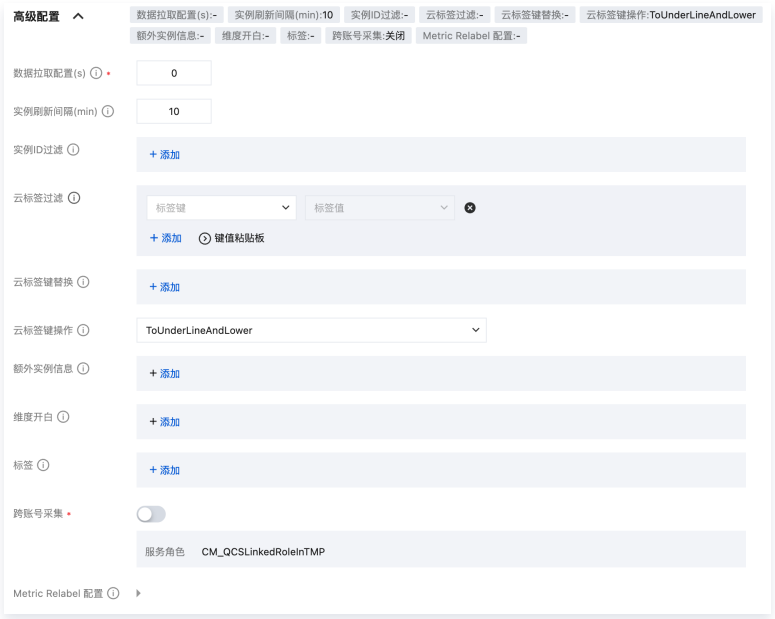
**说明：**

- 数据采集间隔：1分钟。目前不支持更小的采集间隔。
- 监控数据粒度：1分钟。如果指标不支持1分钟粒度，则选择5分钟粒度。
- 集成的监控数据包含云产品的标签数据（部分云产品不支持），标签键必须符合正则表达式 `[a-Za-Z_][a-Za-Z0-9_]*`，否则会被过滤。
- 不支持多地域。如果云产品分布在多个地域，需要安装多个集成。

## 操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心单击**云监控**，默认进入安装页面。定义集成名称、选择对应的云产品和进行 Exporter 配置。




## 配置说明

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 地域 | 必填，云产品所在地域。如果云产品不区分地域，则填写任意地域。 |
| 云产品选择 | 勾选想要采集的云产品。 |
| 数据拉取配置 | 单位秒。若设置为0，将忽略原始数据的时间戳；若设置大于0，将上报原始数据的时间戳，由于云产品监控数据上报到基础监控存在一定的延迟，该延迟将会体现在最新的数据上。<br>拉取数据范围：(当前时间 - 数据采集延迟 - 固定的时间间隔, 当前时间 - 数据采集延迟)。 |
| 实例刷新间隔 | 单位分钟，最小值为10。每隔一个实例刷新间隔，集成会重新拉取云产品实例信息。如果修改了实例名、云标签或者增删实例，会在一个实例刷新间隔内更新监控数据。 |
| 实例 ID 过滤 | 选填。不填默认采集主账号下所有实例的数据。键值对形式填写，键是集成定义的云产品 唯一 ID，值是逗号分隔的云产品实例 ID。填写键值对的云产品，只会采集填写的实例。 |
| 云标签过滤 | 选填。键值对形式填写，一个标签键可以对应多个标签值，以 \| 分割。不同的标签键取交集，同一标签键下的多个标签值取并集。对于支持云标签过滤的产品，如果同时配置了实例 ID 过滤，该产品的云标签过滤将不会生效。 |
| 云标签键替换 | 选填。将不合法的云产品标签键替换为合法值，例如将中文名转换成自定义的英文名。 |
| 云标签键操作 | 集成默认将标签键的大写字母转换成下划线+小写字母。支持对云产品标签键的转换操作：<br>• ToUnderLineAndLower：默认操作。<br>• ToLower：表示全转成小写字母。<br>• NoOperation：表示不做转换。 |
| 额外实例信息 | 添加额外的实例信息。Project 表示添加项目 ID 和项目名称标签，只有部分云产品支持。 |
| 维度开白 | 选填。部分云产品的维度存在指标名称相同、功能需要开白等问题，默认不采集，可通过该配置开启采集。<br>• lb_public:listener：负载均衡（公网）-监听器维度。<br>• lb_public:target：负载均衡（公网）-后端服务器维度。<br>• lb_public:domain：负载均衡（公网）-转发规则域名维度。<br>• lb_private:listener：负载均衡（内网）-监听器维度。<br>• lb_private:target：负载均衡（内网）-后端服务器维度。<br>• lb_private:domain：负载均衡（内网）-转发规则域名维度。<br>• apigw_cloudnative:node：云原生 API 网关-节点维度。<br>• scf_v2:version：云函数-版本维度。<br>• vbc:qosid：云联网-调度队列维度。<br>• ces:node：Elasticsearch-节点维度。 |
| 标签 | 选填。可以给集成采集到的指标添加额外的自定义标签。 |
| 跨账号采集 | 开启后可填写 跨账号采集 配置。<br>• 本账号角色：自定义角色，用于获取本账号临时密钥。<br>• 目标账号角色：自定义角色，用于获取目标账号临时密钥。<br>• 目标账号 uin：目标账号的主账号 ID。 |
| Metric Relabel 配置 | 选填。Prometheus Operator 原生的 metricRelabelings 配置。配置方式与 Prometheus 抓取配置的 metric_relabel_configs 相同，只有部分字段命名方式不同。 |

## Metric Relabel 配置示例

下面是常用的 metricRelabelings 示例：

```
metricRelabelings:
- action: labeldrop  # 去掉名为 labelA 的 label。regex是正则表达式，多个正则表达式用 | 分隔
  regex: labelA
```

```
- regex: ins-(.*) # 新增一个名为 id 的 label，其值通过名为 instance_id 的 label 的值经过正则处理后得到。例如
instance_id="ins-a"，新得到的 id="a"
  replacement: $1
  sourceLabels:
  - instance_id
  targetLabel: id
- targetLabel: region # 新增一个 region="ap-guangzhou" 的 label
  replacement: ap-guangzhou
- action: drop # 去掉名为 metricA 或 metricB 的指标
  sourceLabels:
  - __name__
  regex: metricA|metricB
```

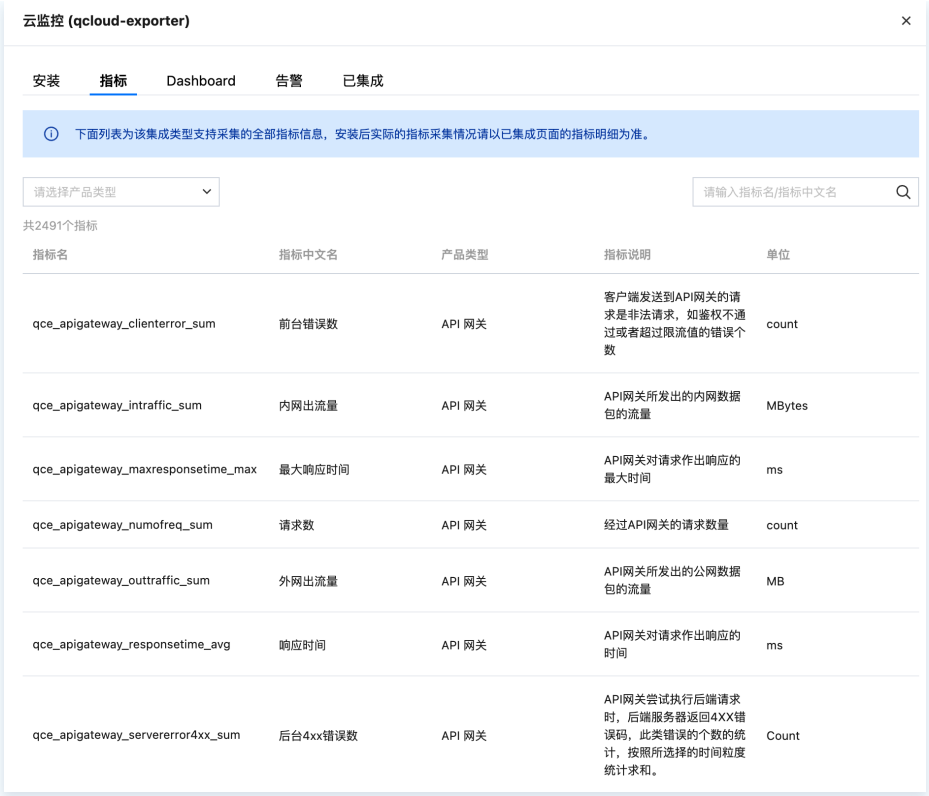
## 支持的云产品

| 云产品/指标文档 | 是否支持采集云标签 | 唯一 ID | 补充说明 |
| --- | --- | --- | --- |
| 云服务器 | 是 | cvm | 仅支持实例维度指标。 |
| 云服务器（内网） | 是 | sdn_vm | - |
| 云硬盘 | 是 | cbs | - |
| 负载均衡（公网） | 是 | lb_public | 默认采集实例维度指标，如需监听器、转发规则域名或后端服务器维度指标，请自行添加维度开白配置。不同维度的指标名相同，可以通过 monitor_view 标签来区分维度：<br>• 实例维度：instance；<br>• 监听器维度：listener；<br>• 后端服务器维度：target；<br>• 转发规则域名维度：domain。 |
| 负载均衡（内网） | 是 | lb_private | 默认采集实例维度指标，如需监听器或转发规则域名维度指标，请自行添加维度开白配置。不同维度的指标名相同，可以通过 monitor_view 标签来区分维度：<br>• 实例维度：instance；<br>• 监听器维度：listener；<br>• 后端服务器维度：target；<br>• 转发规则域名维度：domain。 |
| 负载均衡（四层独占集群） | 是 | tgw_set | 该产品是负载均衡开白产品，指标随时可能变化，不提供对外指标文档。 |
| 负载均衡（七层独占集群） | 是 | tgw_set_l7 | 该产品是负载均衡开白产品，指标随时可能变化，不提供对外指标文档。 |
| 云数据库 MongoDB | 是 | cmongo | - |
| 云数据库 MySQL（CDB） | 是 | cdb | - |
| 云数据库 Redis（CKV 版） | 是 | redis | - |
| 云数据库 Redis（内存版） | 是 | redis_mem | 支持实例维度和节点维度指标。 |
| 云数据库 Tendis | 是 | tendis | - |
| CTSDB（InfluxDB 版） | 是 | xstor | 该产品是 CTSDB 开白产品，指标随时可能变化，不提供对外指标文档。 |
| 云数据库 MariaDB | 是 | mariadb | 仅支持实例维度指标。 |
| 云数据库 PostgreSQL | 是 | postgres | - |
| TDSQL MySQL 版 | 是 | tdmysql | 仅支持实例维度指标。 |

| | | | |
|---|---|---|---|
| TDSQL-C MySQL 版 | 是 | cynosdb_mysql | 仅支持实例维度指标。 |
| 云数据库 SQL Server | 是 | sqlserver | 仅支持实例维度指标。 |
| 云数据库 KeeWiDB | 是 | keewidb | - |
| 向量数据库 | 是 | vecdb | - |
| NAT 网关 | 是 | nat_gateway | - |
| NAT 实例监控丢包率 | 是 | vpc_gw_detect | NAT 网关指标。 |
| 消息队列 Ckafka 版 | 是 | ckafka | 不支持 broker_ip 维度指标。 |
| 消息队列 Pulsar 版 | 是 | tdmq_pulsar | - |
| 消息队列 RocketMQ 版 | 是 | rocketmq | 指标文档和唯一 ID 对应新指标，使用旧指标的用户建议尽快切换。 |
| 消息队列 RabbitMQ 版 | 是 | rabbitmq | - |
| 消息队列 MQTT 版 | 是 | mqtt | - |
| 弹性公网 IP | 是 | lb | - |
| VPN 网关 | 是 | vpngw | - |
| VPN 通道 | 是 | vpnx | - |
| 网络探测 | 不支持标签 | vpc_net_detect | - |
| 私有网络-跨可用区流量 | 是 | sdn_az | 指标不支持1分钟粒度，默认拉取5分钟粒度数据。 |
| 私有网络-私有连接 | 是 | private_link | - |
| CDN（国内域名） | 是 | cdn | 不区分地域。仅支持域名维度指标。 |
| CDN（国外域名） | 是 | ov_cdn | 不区分地域。 |
| COS | 是 | cos | 存储相关指标延迟过高（2小时左右），不会保留数据的原始时间戳。存储相关指标不支持1分钟粒度，默认拉取5分钟粒度数据。 |
| 专线接入-物理专线 | 是 | dc | 不区分地域。 |
| 专线接入-专用通道 | 是 | dcx | 不区分地域。 |
| 专线接入-专线网关 | 是 | dcg | 同私有网络/网络连接/专线网关。 |
| 轻量应用服务器 | 是 | lighthouse | - |
| 云原生 API 网关 | 是 | apigw_cloudnative | 默认采集实例维度和公网负载均衡维度指标，如需节点维度指标，请自行添加维度开白配置。实例维度与节点维度的指标名相同，可以通过 monitor_view 标签来区分维度：<br>• 实例维度：gateway；<br>• 公网负载均衡维度：loadbalancer；<br>• 节点维度：node。 |
| Nacos | 是 | nacos | - |
| Zookeeper | 是 | zookeeper | - |
| Elasticsearch | 是 | ces | 默认采集实例维度指标，如需节点维度指标，请自行添加维度开白配置。可以通过 monitor_view 标签来区分维度：<br>• 实例维度：instance；<br>• 节点维度：node。 |

| | | | |
|---|---|---|---|
| 流计算 Oceanus | 是 | tstream | 需要在 流计算控制台 > **成员管理**中将服务角色 CM_QCSLinkedRoleInTMP 设为访客角色。 |
| 数据湖计算 DLC | 是 | dlc | 支持内部存储桶、Spark 引擎、Presto 引擎、网关 ID、Spark 作业、引擎维度指标。可以通过 monitor_view 标签来区分维度：<br>• 内部存储桶：bucket<br>• Spark 引擎：spark_engine<br>• Presto 引擎：presto_engine<br>• 网关 ID：gateway<br>• Spark 作业：spark_job<br>• 引擎（包括 Spark 和 Presto 引擎）：engine |
| 腾讯云数据仓库 TCHouse-C | 是 | cdwch | - |
| 腾讯云数据仓库 TCHouse-D | 是 | cdwdrs | - |
| 数据传输服务 | 是 | dts | 不支持 Kafka 相关维度指标。 |
| 云联网 | 是 | vbc | 如需调度队列维度指标，请自行添加维度开白配置。 |
| 全球应用加速 | 是 | gaap | - |
| EdgeOne（七层） | 是 | edgeone_l7 | 仅支持【子域名-站点】维度指标。 |
| Web 应用防火墙 | 是 | waf | - |
| 文件存储 | 是 | cfs | 目前未采集元数据、快照相关指标。 |
| 数据加速器 GooseFSx | 是 | goosefsx | - |
| 共享带宽包 | 是 | bwp | - |
| 云函数 | 是 | scf_v2 | 默认采集别名维度指标，如需版本维度指标，请自行添加维度开白配置。别名维度与版本维度的指标名相同，可以通过 monitor_view 标签来区分维度：<br>• 别名维度：alias。<br>• 版本维度：version。 |
| 云点播（VOD） | 是 | vod | 不区分地域。目前 FluxHitRate、RequestsHitRate、BackOriginBandwidth、BackOriginFlux 四个指标未对外，无法采集。 |
| 日志服务（CLS）-日志主题 | 是 | cls | - |
| 数据万象 | 是 | ci | - |
| API 网关 | 是 | apigateway | 仅支持 API 维度指标。 |
| TI-ONE（任务式建模） | 是 | ti_traintask | - |
| TI-ONE（Notebook） | 是 | ti_notebook | - |
| TI-ONE（在线服务） | 是 | ti_model | - |

**说明：**
为了区分不同云产品的指标，云监控集成对云产品指标名（指标文档中的指标英文名）做了转换。指标页中提供了云监控集成支持采集的指标信息，方便用户直接查看和使用。




## 跨账号采集

> **注意：**
> 不支持跨站采集（国内站账号与国际站账号不能互相采集）。

**场景：**账号 A 跨账号采集 账号 B 的监控数据。
**配置填写：**

- 在账号 A 下的 Prometheus 监控服务实例中创建云监控集成。
- 开启**跨账号采集**。
- **本账号角色**选择账号 A 创建的自定义角色。
- **目标账号角色**填入账号 B 创建的自定义角色。
- **目标账号 uin** 填入账号 B 的主账号 ID。


## 简要流程图


## 自定义角色

### 账号 A 创建自定义角色

1. 在 策略 页面，通过策略语法创建 自定义策略，添加 sts:AssumeRole 权限，该权限用于扮演账号 B 的角色。策略语法如下：

```
{
 "version": "2.0",
 "statement": [
 {
     "effect": "allow",
     "action": ["sts:AssumeRole"],
     "resource": ["*"]
 }
 ]
}
```

> **注意：**
> 如果需要限制权限，例如只能扮演账号 B 的自定义角色，可以将 resource 修改为 "qcs::cam::uin/[账号 B 主账号 ID]:roleName/[账号 B 自定义角色]"。

2. 在 角色 列表页面，单击**新建角色**。
3. 在弹出的选择角色载体窗口，选择**腾讯云产品服务**，进入角色信息填写页面。
4. 勾选**云服务器（cvm）**作为角色载体，使用案例选择**云服务器**，单击**下一步**。
5. 在策略列表内，勾选第1步创建的策略为角色配置策略，单击**下一步**。
6. 标记角色的标签键和标签值，可不填，单击**下一步**。
7. 输入您的角色名称，单击**完成**后即完成自定义角色创建。

### 账号 B 创建自定义角色

1. 在角色列表页面，单击**新建角色**。
2. 在弹出的选择角色载体信息窗口，选择**腾讯云账户**作为角色载体，进入角色信息填写页面。
3. 在输入角色载体信息页面，**云账号类型**选择**其他主账号**，**账号 ID** 填写账号 A 主账号 ID，其它可不填，单击**下一步**。
4. 在策略列表内，勾选预设策略 **ReadOnlyAccess** 为角色配置策略，单击**下一步**。
5. 标记角色的标签键和标签值，可不填，单击**下一步**。
6. 输入您的角色名称，单击**完成**后即完成自定义角色创建。

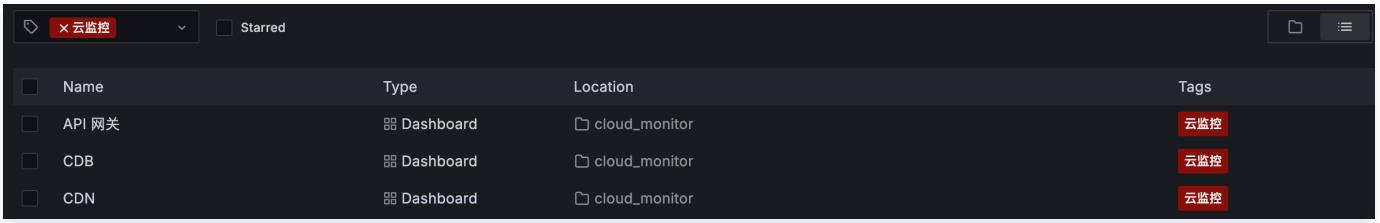
## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，在集成中心页面，找到并单击**云监控**，选择 **Dashboard > Dashboard 操作**下的**安装/升级 Dashboard**，单击**安装/升级**安装对应的 Grafana Dashboard。
3. 选择**已集成**，在已集成列表中单击 Grafana 图标即可自动打开云监控集成大盘列表，选择对应云产品大盘，查看实例相关的监控数据，如下图所示：





## 常见问题

### “数据拉取配置”该怎么配置?

- 若配置为0，Prometheus 会使用当前时间戳，覆盖数据的原始时间戳。

  使用场景：保证数据时间戳的实时性，以最大限度保证 Prometheus 能及时发出告警。
- 若配置为某个大于0的值 x：
  - 只要是大于0的值，Prometheus 就会保留数据的原始时间戳。

    使用场景：与云产品控制台监控页的时间戳保持一致。
  - 延迟拉取数据的时间窗口（延迟量等于 x）。

    问题背景：为了兼容云产品监控数据上报链路的时延，Prometheus 默认以 `(now-固定时延，now)` 的时间范围拉取数据。
    使用场景：若个别产品上报链路时延过大，此处需设置 x，使得拉取数据的时间范围变为：`(now-固定时延-x，now-x)`，以保证在这个延迟的窗口内，能更大限度地拉取到数据。

### 数据偶尔会产生1-2分钟的断点?

- **数据拉取配置**为0：一般不会产生断点。产生断点时请 提交工单 。
- **数据拉取配置**非0：查看集成日志，如果没有明显报错，则可能是延迟波动导致的断点。此时可观察指标正常时的延迟，将**数据拉取配置**重新设置为该延迟大小，例如3分钟的延迟，就设置为180，如果断点情况未能改善，请 提交工单 。

> ⚠ **注意：**
> - 重新设置**数据拉取配置**会影响当前集成中的所有云产品，如果只是个别云产品数据有断点，建议单独新建一个集成。
> - **延迟波动**：部分云产品监控数据，其延迟并不是稳定的。当延迟突然变低时，意味着一分钟内落盘了多个数据点，而云监控集成每一分钟只会采集最新的数据点，这就会导致断点。

### Targets 显示有问题?

- 无采集对象：刚创建的集成需要等待几分钟才能展示正确的 targets。
- (1/2)down：集成采用滚动更新，在新 pod 成功运行之前会继续采集旧 pod，期间就会显示两个 targets。

### 某个云产品没采集到指标?

1. 在**已集成**下，查看如下信息：
   - 查看**实例信息**是否含有该云产品，没有则说明未勾选该云产品。

- 确定 Targets 是 up 状态。
- 查看**指标明细**中是否有该云产品指标，若有则等待一分钟后再查询。


2. 确定所选地域下有该云产品实例。
3. 查看是否配置了**实例 ID 过滤**或**云标签过滤**，确定对应配置能获取到该云产品实例。
4. 查看是否配置了 **Metric Relabel 配置**，确定对应配置没有过滤该云产品指标。

## 更新动态

云监控集成每次保存，都会将集成更新为最新版本。下面是集成每次版本更新的主要时间节点与内容，可以用来评估集成更新的影响。

| 时间 | 更新内容 |
| --- | --- |
| 2025年5月 | Elasticsearch 支持节点维度指标，新增标签 monitor_view；TI-ONE 新增 gpu、gpu_type、app_id、sub_uin、sub_uin_name 标签。<br>注意：gpu 表示 GPU 卡资源，单位为1单位的 gpu_type；gpu_type 表示 GPU 类型；存量 Elasticsearch 和 TI-ONE 实例更新后会产生新时间线。 |
| 2025年4月 | 支持数据湖计算 DLC、数据加速器 GooseFSx；负载均衡（内网）支持后端服务器维度、清理重复标签 vpcid、loadbalancerid、loadbalancerport；云服务器新增标签 gpu_type，仅影响 GPU 云服务器。<br>注意：gpu_type 表示 GPU 类型，存量 GPU 云服务器实例更新后会产生新时间线；存量负载均衡（内网）实例更新后会产生新时间线。 |
| 2025年3月 | 支持 NAT 实例监控丢包率；云服务器支持 HCCPNV6e、HCCPNV5b、HCCSA4机型的 vRDMA 指标；云联网支持网络实例维度。 |
| 2025年2月 | 云数据库 Tendis 新增标签 node_zone_id。<br>注意：node_zone_id 表示节点可用区 ID，存量实例更新后会产生新时间线。 |
| 2025年1月 | 支持数据万象、消息队列 MQTT 版；文件存储支持 Turbo 文件系统指标。<br>注意：文件系统新增标签 protocol，表示文件系统协议，存量实例更新后会产生新时间线。 |
| 2024年12月 | 云联网支持服务等级维度和调度队列维度。 |
| 2024年11月 | 支持流计算 Oceanus、腾讯云数据仓库 TCHouse-C。 |
| 2024年10月 | 云数据库 MongoDB 支持 mongos 节点维度。 |
| 2024年9月 | 支持向量数据库；云产品全量支持采集云标签和云标签过滤；支持跨账号采集；负载均衡支持转发规则域名维度。 |
| 2024年8月 | 支持私有网络-跨可用区流量、私有网络-私有连接；COS 支持5分钟粒度的存储相关指标；TDSQL-C MySQL 版支持集群 ID 过滤。 |
| 2024年7月 | 支持消息队列 Pulsar 版；云函数支持版本维度。<br>注意：云函数新增固定标签 monitor_view，存量实例更新后会产生新时间线。 |
| 2024年6月 | 支持负载均衡（七层独占集群）、腾讯云数据仓库 TCHouse-D、云服务器（内网）；负载均衡（内网）支持监听器维度；云原生 API 网关支持公网负载均衡维度。<br>注意：负载均衡（内网）新增固定标签 monitor_view，存量实例更新后会产生新时间线。 |
| 2024年5月 | 支持云数据库 KeeWiDB；负载均衡（公网）支持后端服务器维度。 |

| | |
|---|---|
| 2024年3月 | 支持云原生 API 网关、消息队列 RabbitMQ 版、EdgeOne（七层）；支持配置云标签键替换和云标签键转换规则。 |
| 2023年12月 | 支持消息队列 RocketMQ 版 v5、CTSDB（InfluxDB 版）、TI-ONE、云数据库 Tendis、弹性公网 Ipv6。 |
| 2023年11月 | 支持网络探测；负载均衡（公网）支持监听器维度、云数据库 MySQL 支持代理节点维度。 |
| 2023年9月 | 支持日志服务、API 网关、负载均衡（四层独占集群）。 |
| 2023年8月 | 支持 CDN（国外域名）、云点播、云函数（别名）、消息队列 RocketMQ 版。 |
| 2023年7月 | 支持采集域名型负载均衡（公网）；支持采集 TDSQL-C MySQL 版云标签。 |
| 2023年6月 | 支持共享带宽包。云数据库 Redis（内存版）Redis 节点维度添加节点类型标签。 |
| 2023年5月 | 支持采集消息队列 CKafka 版云标签。 |
| 2023年4月 | 支持文件存储。 |
| 2023年3月 | 支持云硬盘。 |
| 2023年2月 | 支持负载均衡（内网）、Web 应用防火墙。 |
| 2022年11月 | 支持全球应用加速、云联网、数据传输服务。 |
| 2022年9月 | 支持专线网关；支持采集云产品标签。 |
| 2022年7月 | 支持 Zookeeper、Nacos、COS、CDN（国内域名）。 |
| 2022年6月 | 云监控集成上线。 |

# 非腾讯云主机监控

最近更新时间：2024-10-10 17:00:21

## 背景

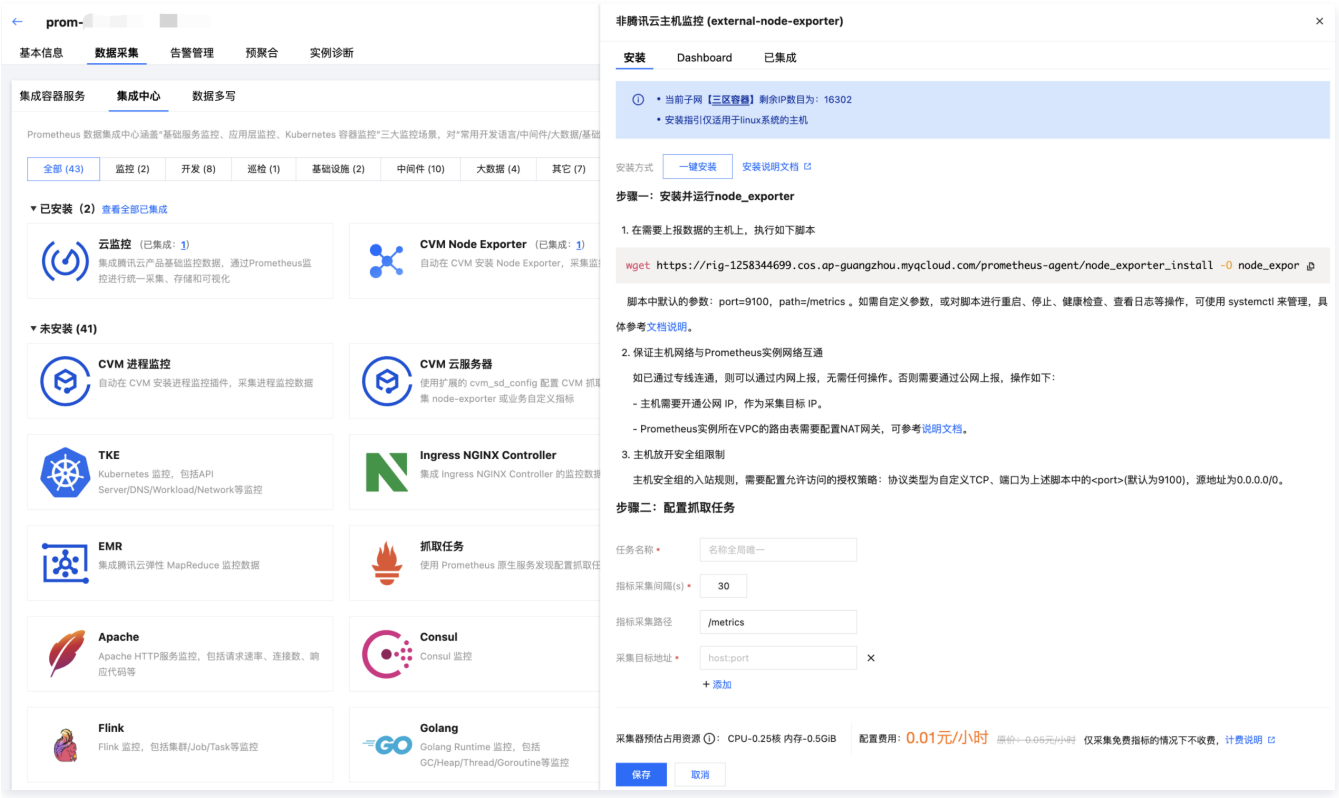
本文主要引导用户如何快速采集非腾讯云主机的监控数据，降低用户配置成本。

## 接入方式

### 方式一：一键安装(推荐)

#### 操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，单击**数据采集 > 集成中心**。
4. 在集成中心找到并单击**非腾讯云主机监控**，即会弹出一个安装窗口。




### 步骤一：安装并运行 node_exporter

1. **在需要上报数据的主机上执行以下脚本：**

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter_install -O
node_exporter_install && chmod +x
node_exporter_install && ./node_exporter_install
```

执行脚本会自动触发以下动作：下载 node exporter、运行 node exporter、检查数据上报、完成（数据成功暴露在9100端口）。
脚本执行结果示例如下：


> **说明：**
> 脚本中默认的参数：port=9100，path=/metrics ，如需自定义参数或对脚本进行重启、停止、健康检查、查看日志等操作，可使用 systemctl 来管理。

**自定义参数：**

- 修改 port，执行脚本语句替换为：

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter_install -O node_exporter_install && chmod +x node_exporter_install && ./node_exporter_install --web.listen-address=":9100"
```

- 修改 path，执行脚本语句替换为：

```
 wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter_install -O node_exporter_install && chmod +x node_exporter_install && ./node_exporter_install --web.telemetry-path="/metrics"
```

> **说明：**
> 更多自定义参数的配置指引可参考 文档说明 。

**常用的脚本管理操作：**

- 重启：

```
systemctl restart node_exporter
```

- 停止：

```
systemctl stop node_exporter
```

- 状态检查：

```
systemctl status node_exporter
```

- 日志查看：

```
journalctl -u node_exporter
```

**2. 保证主机网络与 Prometheus 实例内网互通**

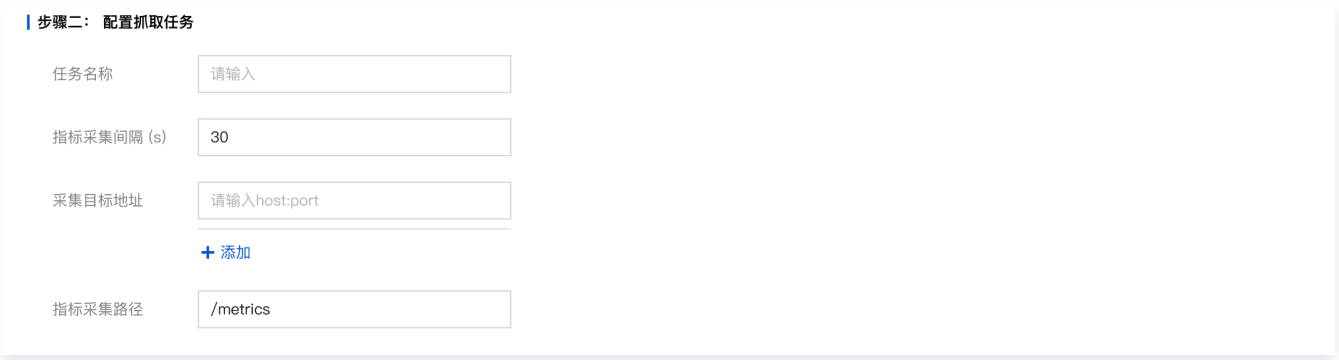
如已通过专线连通，则可以通过内网上报，无需任何操作。否则需要通过公网上报，操作如下：

- 主机需要开通公网 IP，作为采集目标 IP。
- Prometheus 实例所在 VPC 的路由表需要配置 NAT 网关，可参考 TKE Serverless 集群如何放通外网 。

**3. 主动放开安全组限制**

主机安全组的入站规则，需要配置允许访问的授权策略：协议类型为自定义 TCP、端口为上述脚本中的<port>，源地址为0.0.0.0/0。

### 步骤二：配置抓取任务




| 参数 | 说明 |
| --- | --- |
| 任务名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 指标采集间隔(s) | 输入指标采集间隔，单位s。 |
| 采集目标地址 | 输入采集目标地址，格式：host:port，支持添加多个。 |
| 指标采集路径 | 输入指标采集路径，默认为/metrics。 |

## 方式二：自定义安装

上述步骤一中脚本安装的方式，还可以替换为自定义安装，参考下述指引。

### 1. 下载安装 node_exporter：

在需要上报数据的主机上，下载并安装 node_exporter，您可以点击进入 Prometheus 开源官网下载地址 node_exporter ，也可以直接执行下列命令：

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter -O node_exporter
```

目录为当前文件夹：

```
[root@VM-0-46-tencentos ~]# ls
node_exporter
[root@VM-0-46-tencentos ~]#
```

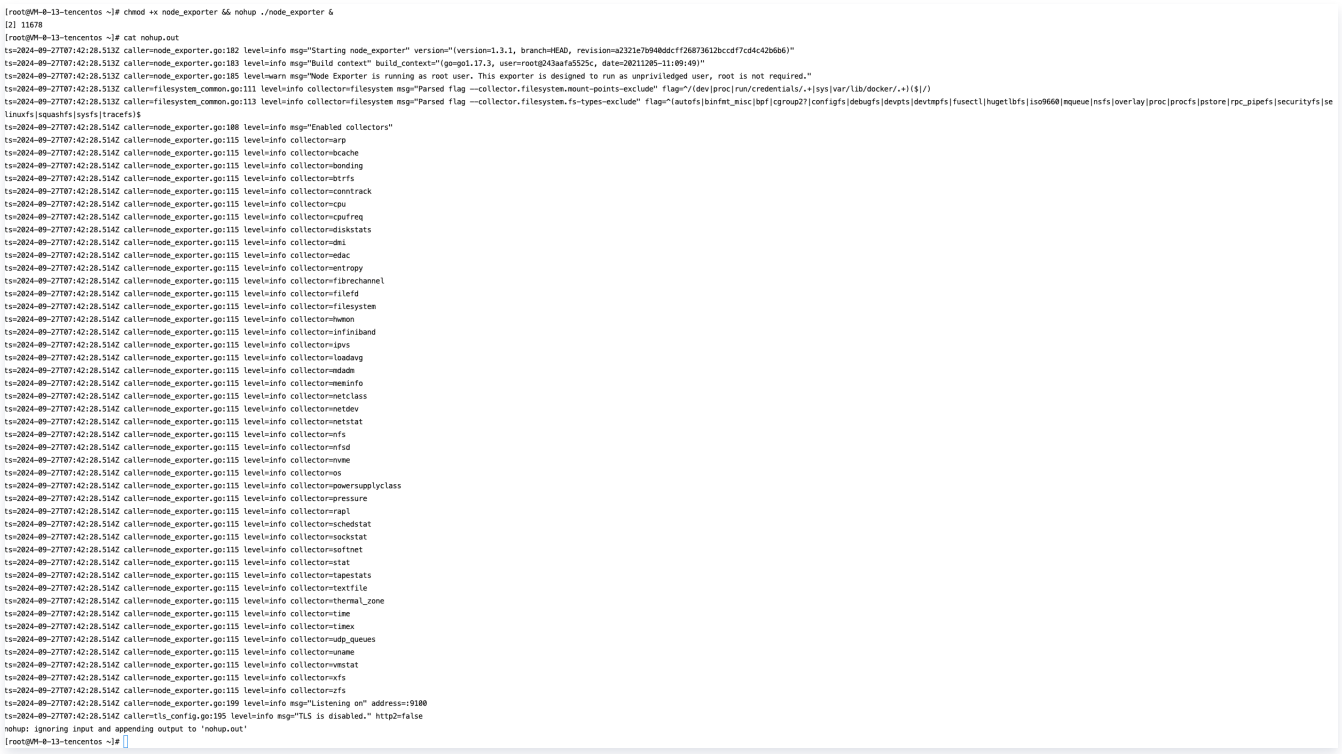
### 2. 运行 node_exporter 采集基础监控数据：

- 赋予权限，执行 node_exporter并查看日志。

```
chmod +x node_exporter && nohup ./node_exporter &
```

```
cat nohup.out
```

如下图所示即为执行成功：




- 可通过下列命令，查看暴露在9100端口的监控数据：

```
curl 127.0.0.1:9100/metrics
```

如下图为执行命令后看到的暴露出来的指标监控数据：


完成上述操作后，需在页面中配置抓取任务，参考方式一中的 配置描述 。

# 查看监控

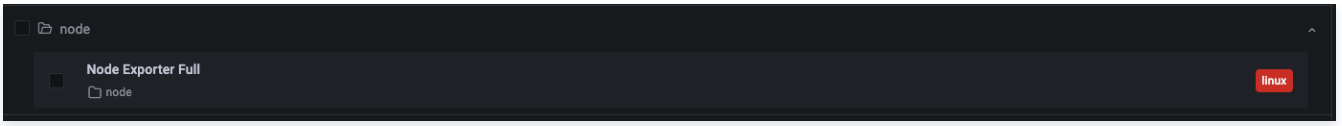
## 前提条件

Prometheus 实例已绑定 Grafana 实例。

## 操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。

2. 选择**数据采集 > 集成中心**，在集成中心页面找到非腾讯云主机监控卡片并点击弹出集成页面，选择 **Dashboard > Dashboard 安装/升级** 来安装对应的 Grafana Dashboard。
3. 打开 Prometheus 实例关联的Grafana实例地址，在 `Dashboards` 页面查看相关的监控大盘。





## 配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**告警管理**，可以添加相应的告警策略，详情请参见 新建告警策略 。

# 通过 Remote Read 读取云托管 Prometheus 实例数据

最近更新时间：2024-08-16 11:52:01

## 操作场景

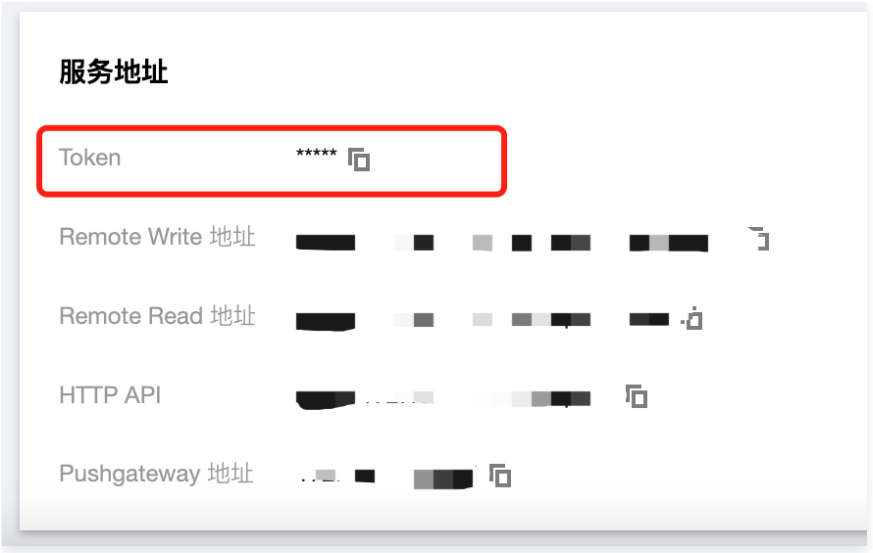
Prometheus 提供了 Remote read 接口，该接口支持将一系列 Prometheus 协议的数据源组织为单一数据源查询。本文介绍如何使用自建 Prometheus 通过 Remote read 读取云托管 Prometheus 实例的数据。

## Remote Read 配置

推荐配置 `prometheus.yml` 如下：

```
remote_read:
  - url: 'http://prom_ip:prom_port/api/v1/read'
    read_recent: true
    basic_auth:
      username: app_id
      password: token
```

推荐使用 Basic Auth 方式访问云托管 Prometheus 实例，username 为账号 AppID ，password 为 Prometheus 控制台 > 基本信息 > 服务地址中获取的 Token。




## 注意事项

- **配置 Remote read 的 Prometheus 需谨慎配置 `global:external_labels` **：
  external_labels 会被附加在 Remote read 的查询条件中，不正确的 label 可能导致查询不到需要的数据。
  `filter_external_labels: false` 配置项可以避免将 external_labels 加入查询条件（v2.34 版本以上支持）。
- **避免出现相同的 series**：
  对于完全相同的两个 series，Prometheus 会在查询合并时在每个时间点在随机一个 series 取值组成新的 series 作为查询结果，这会导致查询结果不准确。
  在 Prometheus 的设计理念中不存在多副本冗余存储的情况，所以不会对这种场景提供支持。

## Remote read 完整配置项

> **说明：**
> `[]` 中的配置项为可选项（本文展示 Prometheus:v2.40 版本配置，低版本可能缺少部分配置项，详见 prometheus官方文档 ）

```
# remote read 目标 prometheus 实例的 api 地址
url: <string>
```

```
# 标识一个唯一的 remote read 配置名称
[ name: <string> ]

# 查询 promql 中必须包含以下 label 过滤条件才会进行 remote read 查询
required_matchers:
  [ <labelname>: <labelvalue> ... ]

# remote read 查询超时时间
[ remote_timeout: <duration> | default = 1m ]

# 自定义 remote read 请求中附带的 headers，无法覆盖 prometheus 原本添加的 headers
headers:
  [ <string>: <string> ... ]

# 在本地有完整数据存储的时间范围是否进行 remote read 查询
[ read_recent: <boolean> | default = false ]

# 为每个 remote read 请求添加 Authorization header，password password_file 二选一
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# 自定义 Authorization header 配置
authorization:
  # 认证类型
  [ type: <string> | default: Bearer ]
  # 认证密钥，credentials credentials_file 二选一
  [ credentials: <secret> ]
  # 密钥从文件中获取
  [ credentials_file: <filename> ]

# OAuth2.0认证，不能与 basic_auth authorization 同时使用
oauth2:
  [ <oauth2> ]

# TLS 配置
tls_config:
  [ <tls_config> ]

# 代理 URL
[ proxy_url: <string> ]

# 查询请求是否接受3XX 跳转
[ follow_redirects: <boolean> | default = true ]

# 是否启用 HTTP2
[ enable_http2: <bool> | default: true ]

# 是否在 remote read 时附加 external_labels
[ filter_external_labels: <boolean> | default = true ]
```

# Agent 自助接入

最近更新时间：2024-07-19 14:57:51

## 使用场景

采集自建 IDC 上的服务，需要部署 Agent 并管理采集配置，上报监控数据到云上 Prometheus 监控服务。对于云上服务推荐使用 集成中心，集成中心会托管 Agent，提供多种中间件和抓取任务的自动化集成。

## 获取 Prometheus 实例访问配置

1. 前往 Prometheus 监控控制台，选择对应的实例 ID/名称，在**基本信息 > 服务地址**页面，获取 Remote Write 地址和 Token。


2. 在 账号信息 页面获取 APPID。

## 确认所在网络环境和云上实例联通

根据获取的 RemoteWrite 地址，执行如下命令，如果网络联通返回信息会包含 `401 Unauthorized` 。

```
curl -v -X POST  ${RemoteWriteURL}
```

## 安装并启动 vmagent

vmagent 占用较少的资源且兼容 Prometheus 采集配置和 Remote Write 协议而得到广泛使用。本文只介绍 vmagent 常用启动选项，通过 Systemd 或 Docker 来管理 vmagent，更多详细信息可以参考 官方文档。

### 常用启动选项

- -promscrape.noStaleMarkers：如果采集目标消失，默认会生成所有关联指标的 stale marker 并写到远程存储。设置该选项禁止该行为，能减少内存的占用。
- -loggerTimezone：日志中时间的时区，例如 `Asia/Shanghai, Europe/Berlin` 或者 `Local` (默认是 "UTC")。
- -remoteWrite.tmpDataPath：采集后待写出数据临时存储的文件路径。
- -remoteWrite.url：数据写向远程存储的 URL。
- -remoteWrite.basicAuth.username：远程存储 -remoteWrite.url 对应的 basic auth 用户名。
- -remoteWrite.basicAuth.password：远程存储 -remoteWrite.url 对应的 basic auth 密码。
- -promscrape.config：采集配置的路径，可以是文件路径或者 HTTP URL，更多详情 参考文档 。
- -promscrape.configCheckInterval：检查 -promscrape.config 配置变化的时间间隔，关于配置更新可以 参考文档 。

### 通过 Docker 管理

1. 在 vmagent 发布页面 选择镜像版本，推荐使用 latest。
2. 替换脚本中的 Prometheus 实例信息，启动 vmagent。

```
mkdir /etc/prometheus
touch /etc/prometheus/scrape-config.yaml
docker run -d --name vmagent --restart always --net host -v /etc/prometheus:/etc/prometheus
victoriametrics/vmagent:latest \
-promscrape.noStaleMarkers \
-loggerTimezone=Local \
-remoteWrite.url="${RemoteWriteURL}" \
-remoteWrite.basicAuth.username="${APPID}" \
-remoteWrite.basicAuth.password='${Token}' \
-remoteWrite.tmpDataPath=/var/lib/vmagent  \
-promscrape.config=/etc/prometheus/scrape-config.yaml \
-promscrape.configCheckInterval=5s
```

3. 查看 vmagent 日志

```
docker ps
docker logs vmagent
```

如果正常启动，执行如下命令会返回 `OK` 。

```
curl localhost:8429/health
```

## 通过 Systemd 管理

1. 在 vmagent 发布页面，根据操作系统和 CPU 架构，下载对应 vmutils-* 压缩包并解压。
2. 替换脚本中的 Prometheus 实例访问信息，启动 vmagent。

```
mkdir /etc/prometheus
touch /etc/prometheus/scrape-config.yaml
cat >/usr/lib/systemd/system/vmagent.service <<EOF
[Unit]
Description=VictoriaMetrics Agent
After=network.target

[Service]
LimitNOFILE=10240
ExecStart=/usr/bin/vmagent \
-promscrape.noStaleMarkers \
-loggerTimezone=Local   \
-remoteWrite.url="${RemoteWriteURL}" \
-remoteWrite.basicAuth.username="${APPID}" \
-remoteWrite.basicAuth.password="${Token}" \
-remoteWrite.tmpDataPath=/var/lib/vmagent \
-promscrape.config=/etc/prometheus/scrape-config.yaml \
-promscrape.configCheckInterval=5s
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
systemctl enable vmagent
systemctl start vmagent
sleep 3
systemctl status vmagent
```

3. 查看日志

```
journalctl -u vmagent
```

如果正常启动，执行如下命令会返回 `OK` 。

```
curl localhost:8429/health
```

## 管理配置

### 修改配置文件

编辑采集配置文件 `/etc/prometheus/scrape-config.yaml` 添加/更新/删除采集任务，关于 Prometheus 采集任务配置可以参考 官方文档。

```
global:
   scrape_interval: 30s
scrape_configs:
   - job_name: agent-monitor
     static_configs:
        - targets:
             - localhost:8429
```

修改配置后，要过选项 `-promscrape.configCheckInterval` 设置时间才会生效。

### 查看监控目标信息

执行如下命令，查看采集目标，可以查看配置是否生效并符合预期。

```
curl localhost:8429/api/v1/targets
```

# Pushgateway 接入

最近更新时间：2024-12-16 15:49:33

## 使用场景

PushGateway 是 Prometheus 生态中的一个重要一员，它允许任何客户端向其 Push 符合规范的自定义监控指标，再结合 Prometheus 统一收集监控。Prometheus Pushgateway 用于接收短期任务的指标数据，这些任务通过服务发现的监控系统无法直接监控。Pushgateway 允许临时性的工作（例如批处理作业）将指标推送到一个中央位置，而无需直接暴露其指标。这些数据可以被 Prometheus 服务器拉取和进行持久化存储。

> **注意：**
> Pushgateway 并不能将 Prometheus 转换为基于推送的监控系统，仅用作特殊场景中的指标暂存组件。同时 Pushgateway 不对暂存的指标做持久化保证，重启 Pushgateway 会导致暂存的指标丢失。实际场景请参见 Prometheus 官方文档 示例判断是否应该使用 Pushgateway。

## 一键安装

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 在实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，单击**数据采集 > 集成中心**。
5. 在集成中心搜索 **Pushgateway**，找到后单击它即会弹出一个安装窗口。
6. 在弹出窗口的安装页面，根据提示填写相关信息，并单击**保存**。


**配置说明**

| 参数 | 说明 |
| --- | --- |
| 名称 | 集成名称，命名规范如下：<br>• 名称具有唯一性。<br>• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。 |
| 采集超时 | Pushgateway 采集超时，时间格式，不能大于采集间隔。 |

| | |
|---|---|
| 采集间隔 | Pushgateway 采集间隔，时间格式。 |
| CPU/核 | Pushgateway CPU 核数限制，不能大于64。 |
| 内存/Gi | Pushgateway 内存限制，配置时需带上单位Gi，不能大于512Gi。 |

7. 在已集成列表中获取 Pushgateway 地址信息。


## 数据推送

### 基本概念

- 分组：以 job 和上报时指定的 label 确定的一组指标。
- 分组路径：分组指标上报请求在 pushgateway 中的对应请求路径，格式为：`/metrics/job/<JOB_NAME>{/<LABEL_NAME>/<LABEL_VALUE>}`。

### 开发准备

golang

```
go get github.com/prometheus/client_golang/prometheus
```

python

```
pip install prometheus-client
```

java

```
        <dependency>
            <groupId>io.prometheus</groupId>
            <artifactId>prometheus-metrics-core</artifactId>
            <version>1.3.3</version> <!-- 推荐使用最新版本参考：
https://mvnrepository.com/artifact/io.prometheus/prometheus-metrics-core -->
        </dependency>
        <dependency>
            <groupId>io.prometheus</groupId>
            <artifactId>prometheus-metrics-instrumentation-jvm</artifactId>
            <version>1.3.3</version>
        </dependency>
        <dependency>
            <groupId>io.prometheus</groupId>
            <artifactId>prometheus-metrics-exporter-pushgateway</artifactId>
            <version>1.3.3</version>
        </dependency>
```

其他语言和更详细信息请参见 官方文档。

### PUT 请求

PUT 请求用来向 pushgateway 上报一个完整的分组，pushgateway 会使用 PUT 请求体中的指标更新整个分组（请求体中不存在的指标会被删除）。请求体为空的 PUT 请求会使 pushgateway 删除整个分组。

golang

```golang
package main

import (
    "log"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/push"

    // 配合后续代码debug使用
    // "github.com/prometheus/common/expfmt"
)

// 定义并注册指标
var (
    serverRequestTotal = prometheus.NewCounterVec(prometheus.CounterOpts{
        Name: "http_server_request_total",
        Help: "total count of http server requests",
    }, []string{"path"})

    metricsRegistry = prometheus.NewRegistry()
)

func init() {
    metricsRegistry.MustRegister(serverRequestTotal)
}

func main() {
    url := "YOUR-PUSHGATEWAY-ADDRESS"

    serverRequestTotal.WithLabelValues("api/v1/list").Inc()

    if err := push.New(url, "tcloud_test_job").
        Collector(metricsRegistry).
        // 自定义分组label
        Grouping("provider", "tcloud").
        Grouping("product", "tmp").
        // 发送text/plain格式请求，便于debug
        // Format(expfmt.NewFormat(expfmt.TypeTextPlain)).
        // 发送PUT请求
        Push(); err != nil {
        log.Fatalf("Could not push completion time to Pushgateway: %v", err)
    }
}
```

python

```python
from prometheus_client import CollectorRegistry, Counter, push_to_gateway, pushadd_to_gateway,
delete_from_gateway

registry = CollectorRegistry()
counter = Counter('http_server_request_total', 'total count of http server request', ['path'],
registry=registry)
address = "YOUR-PUSHGATEWAY-ADDRESS"
job = "tcloud_test_job"
```

```
grouping_key = {"product": "tmp", "provider": "tcloud"}

if __name__ == "__main__":
    counter.labels(path='api/v1/list').inc()
    push_to_gateway(address, job=job, grouping_key=grouping_key, registry=registry)
```

java

```
package org.example;

import java.io.IOException;

import io.prometheus.metrics.core.metrics.Counter;
import io.prometheus.metrics.exporter.pushgateway.PushGateway;
import io.prometheus.metrics.instrumentation.jvm.JvmMetrics;
import io.prometheus.metrics.model.registry.PrometheusRegistry;


public class Main {
    private static PrometheusRegistry registry = new PrometheusRegistry();
    private static String address = "YOUR-PUSHGATEWAY-ADDRESS";
    private static String job = "tcloud_test_job";
    private static PushGateway pushGateway = PushGateway.builder().address(address).groupingKey("product",
"tmp").groupingKey("provider", "tcloud").job(job).registry(registry).build();
    private static Counter counter =
Counter.builder().name("http_request_total").labelNames("path").help("total count of http server
request").register(registry);


    public static void main(String[] args) throws IOException {
        JvmMetrics.builder().register(registry);
        counter.labelValues("api/v1/list").inc();
        pushGateway.push();
    }
}
```

shell

```
cat <<EOF | curl -X PUT --data-binary @-
http://*.*.*.*:8080/metrics/job/some_job/some_group_key1/value1/some_group_key2/value2
# TYPE some_metric counter
some_metric{label="val1"} 42
# TYPE another_metric gauge
# HELP another_metric Just an example.
another_metric 2398.283
EOF
```

## POST 请求

POST 请求用来向 pushgateway 上报分组中需要修改的部分指标，pushgateway 仅会更新 POST 请求体中存在的指标，其他指标不变。
请求体为空的 POST 请求仅会更新 pushgateway 中记录的上报时间。

golang

```
// 整体逻辑同PUT请求示例
err := push.New(url, "tcloud_test_job").
       Collector(metricsRegistry).
```

```
        // 自定义分组label
        Grouping("provider", "tcloud").
        Grouping("product", "tmp").
        // 发送POST请求
        Add()
```

**python**

```
if __name__ == "__main__":
    counter.labels(path='api/v1/list').inc()
    pushadd_to_gateway(address, job=job, grouping_key=grouping_key, registry=registry)
```

**java**

```
public static void main(String[] args) throws IOException {
    counter.labelValues("api/v1/list").inc();
    pushGateway.pushAdd();
}
```

**shell**

```
cat <<EOF | curl -X POST --data-binary @-
http://*.*.*.*:8080/metrics/job/some_job/some_group_key1/value1/some_group_key2/value2
# TYPE some_metric counter
some_metric{label="val1"} 42
# TYPE another_metric gauge
# HELP another_metric Just an example.
another_metric 2398.283
EOF
```

## DELETE 请求

DELETE 请求用来删除 pushgateway 中的整个分组。

**golang**

```
// 整体逻辑同PUT请求示例
err := push.New(url, "tcloud_test_job").
        // 自定义分组label
        Grouping("provider", "tcloud").
        Grouping("product", "tmp").
        // 发送DELETE请求
        Delete()
```

**python**

```
if __name__ == "__main__":
    delete_from_gateway(address, job=job, grouping_key=grouping_key)
```

**java**

```
public static void main(String[] args) throws IOException {
    pushGateway.delete();
```

```
}
```

shell

```
curl -X DELETE http://*.*.*.*:8080/metrics/job/some_job/some_group_key1/value1/some_group_key2/value2
```

## 管理 API

pushgateway 集成支持原生 pushgateway 的管理 API。可使用以下命令管理 pushgateway 自带的 API：

```
PUT /api/v1/admin/wipe
```

> **注意：**
> wipe API 会清空 pushgateway 中的全部上报指标。

## 接入建议

因为 pushgateway 自身实现和设计逻辑，仅建议短期任务或特殊场景（存在网络隔离，仅支持主动推送指标等）使用 pushgateway 接入。可以根据实际需要参考以下建议：

- **每个上报端使用单独分组**
  多个上报端同时上报指标到同一个 pushgateway 时，建议附带单独的分组 label，例如 `instance="instanceXX", instance="ip:port"` 等，防止在 pushgateway 中指标相互覆盖。
- **（持续上报场景）上报结束后删除 pushgateway 中对应指标**
  在持续使用 pushgateway 上报的场景中，上报端离线前应主动删除上报的分组，防止最后一次上报的数据一直保留在 pushgateway 中被反复采集。

# Docker 接入

最近更新时间：2024-06-20 17:35:41

## 操作场景

Docker Daemon 是 Docker 的核心组件之一，它是一个持续运行的后台进程，负责管理 Docker 容器的创建、运行和监控。目前其支持了 Prometheus 指标导出能力，可以通过配置将其开启。腾讯云可观测平台 Prometheus 监控服务通过抓取任务采集 Docker 暴露的 Daemon 监控数据，并提供了开箱即用的 Grafana 监控大盘。

## 操作步骤

### 前提条件

docker 版本不低于 17.04.0。

### 开启 Prometheus 监控

#### 配置 docker daemon

要将 Docker Daemon 配置为 Prometheus 目标，需要在 daemon.json 配置文件中添加 metrics-address 配置项，同时由于该功能在某些版本中属于实验性功能，可能还需要添加 experimental 配置项，以下为各系统中 daemon.json 配置默认位置，若配置不存在，则需要创建该配置：

- Linux：`/etc/docker/daemon.json`
- Windows 服务器：`C:\ProgramData\docker\config\daemon.json`
- Docker Desktop：打开 Docker Desktop 设置并选择 Docker Engine 来编辑文件。

如果文件为空，将下面内容放入即可：

```
{
  "metrics-addr" : "127.0.0.1:9323",  ## 集群节点上的docker导出指标时，127.0.0.1替换为节点ip，端口按实际业务调整
  "experimental" : true
}
```

如果文件非空，添加字段 metrics-address 和 experimental，并确保文件仍然是有效的 JSON 格式。
保存文件或配置，重启 Docker：

```
systemctl restart docker
或者
systemctl restart dockerd
```

> ⚠ **注意：**
> 除了最后一行，每一行都需要用逗号结尾。

#### 验证

登录集群或者 docker 容器中 pod 验证拉取指标：

```
curl 127.0.0.1:9323/metrics ## 使用上述配置中的地址端口
```

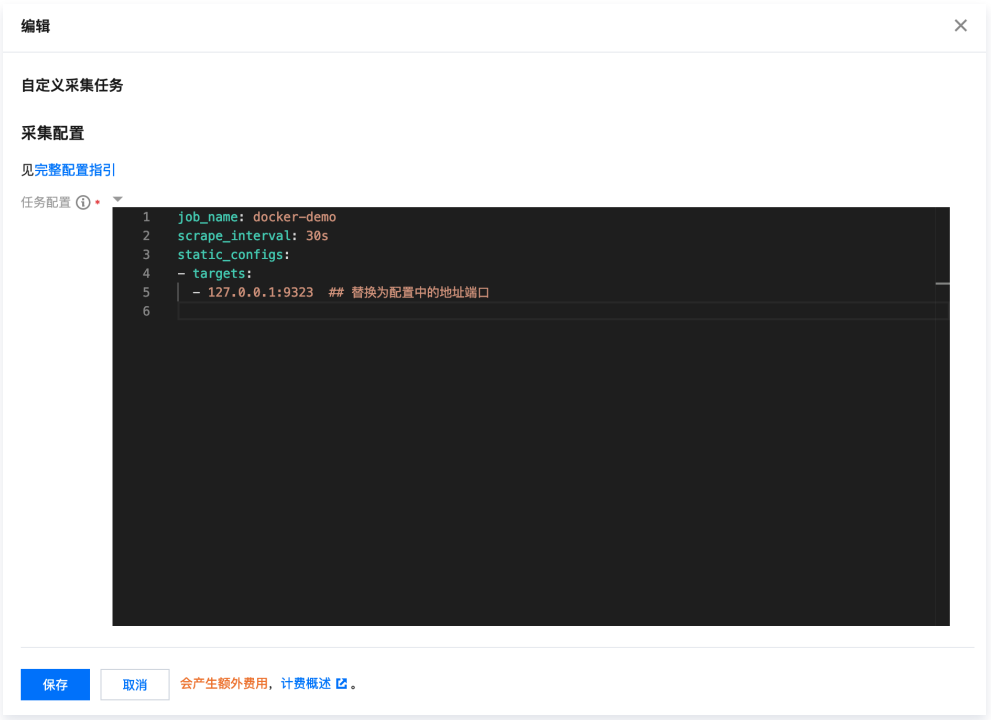
即可看到拉取的指标信息：

```
# HELP engine_daemon_engine_cpus_cpus The number of cpus that the host system of the engine has
# TYPE engine_daemon_engine_cpus_cpus gauge
engine_daemon_engine_cpus_cpus 2
# HELP engine_daemon_engine_info The information related to the engine and the OS it is running on
# TYPE engine_daemon_engine_info gauge
engine_daemon_engine_info{architecture="x86_64",commit="562656c364",daemon_id="FPJ5:VKVG:WAMF:LKPE:FDRQ:345D:HJXM:KQXA:YXKO:ZNQU:LE5N:APQS",graphdriver="overlay2",kernel="5.4.119-1
e="linux",version="v19.3.9-tke.1"} 1
# HELP engine_daemon_engine_memory_bytes The number of bytes of memory that the host system of the engine has
# TYPE engine_daemon_engine_memory_bytes gauge
engine_daemon_engine_memory_bytes 1.8052096e+09
# HELP engine_daemon_events_subscribers_total The number of current subscribers to events
# TYPE engine_daemon_events_subscribers_total gauge
engine_daemon_events_subscribers_total 0
# HELP engine_daemon_events_total The number of events logged
# TYPE engine_daemon_events_total counter
engine_daemon_events_total 28
# HELP engine_daemon_health_checks_failed_total The total number of failed health checks
# TYPE engine_daemon_health_checks_failed_total counter
engine_daemon_health_checks_failed_total 0
# HELP engine_daemon_health_checks_total The total number of health checks
# TYPE engine_daemon_health_checks_total counter
engine_daemon_health_checks_total 0
# HELP engine_daemon_network_actions_seconds The number of seconds it takes to process each network action
# TYPE engine_daemon_network_actions_seconds histogram
engine_daemon_network_actions_seconds_bucket{action="allocate",le="0.005"} 0
engine_daemon_network_actions_seconds_bucket{action="allocate",le="0.01"} 0
engine_daemon_network_actions_seconds_bucket{action="allocate",le="0.025"} 2
```

## 指标采集

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索**抓取任务**，单击**一键安装**。


5. 在新建页，填写指标采集名称和地址等信息，并单击**保存**。




6. 在集成中心搜索 docker，单击 **Dashboard 安装/升级**。

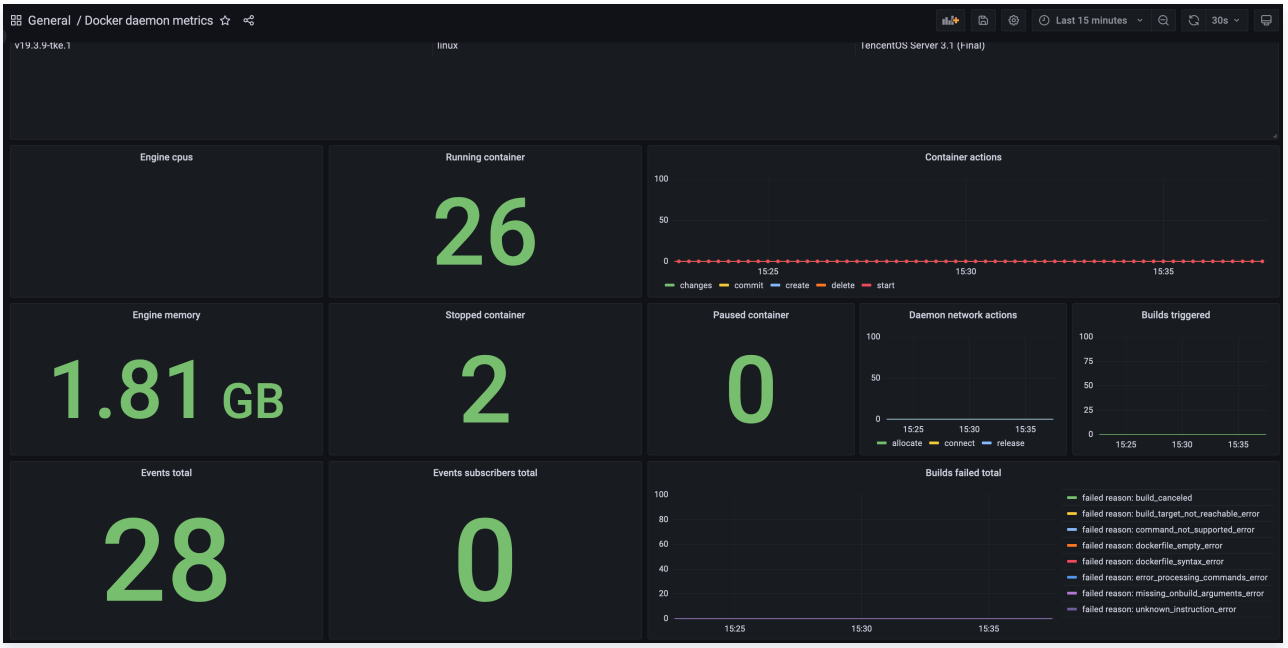

## 查看监控

### 前提条件

Prometheus 实例已绑定 Grafana 实例。

### 操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 others 文件夹中找到 **Docker daemon metrics** 监控面板，查看实例相关监控数据，如下图所示：




## 配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

# TKE 集群内安装组件说明

最近更新时间：2025-01-06 10:24:53

本文介绍 Prometheus 监控服务在 集成容器服务 过程中，在用户 TKE 集群内安装的各个组件的功能，使用权限和占用资源。

## proxy-agent

### 组件介绍

由于 TKE 集群有独立的网络环境，proxy-agent 部署在集群内为集群外的采集组件提供访问代理。外部采集组件一方面通过 proxy-agent 服务发现集群内的资源，另一方面通过 proxy-agent 抓取指标并写到 Prometheus 实例的时序存储中。

### 部署在集群内的资源对象

| Namespace | kubernetes 对象名称 | 类型 | 资源量 | 说明 |
| --- | --- | --- | --- | --- |
| <Prometheus 实例 ID> | proxy-agent | Deployment | 0.25C256Mi*2 | 采集代理 |
| | <Prometheus 实例 ID> | ServiceAccount | - | 权限载体 |
| | | ClusterRole | - | 采集权限相关 |
| | | Role | - | 外部集群额外管理权限 |
| | <Prometheus 实例 ID>-crb | ClusterRoleBinding | - | 采集权限相关 |
| | <Prometheus 实例 ID>-rb | RoleBinding | - | 外部集群额外管理权限 |

### 组件权限说明

#### 权限场景

| 功能 | 涉及对象 | 涉及操作权限 |
| --- | --- | --- |
| 采集配置管理 | scrapeconfigs,servicemonitors,podmonitors,probes,configmaps,secrets,namespaces | get/list/watch |
| 服务发现 | services,endpoints,nodes,pods,ingresses | get/list/watch |
| 部分系统组件指标抓取 | nodes/metrics,nodes/proxy,pods/proxy | get/list/watch |
| 带 RBAC 鉴权的指标抓取 | /metrics,/metrics/cadvisor | get |

#### 外部 Kubernetes 集群额外权限场景

| 功能 | 涉及对象 | 涉及操作权限 |
| --- | --- | --- |
| 采集配置管理 | scrapeconfigs,servicemonitors,podmonitors, probes | *(all) |
| 管理采集专用 namespace | <Prometheus 实例 ID>/* | *(all) |

#### 权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prom-instance
rules:
  - apiGroups:
      - monitoring.coreos.com
    resources:
      - scrapeconfigs
      - servicemonitors
```

```
      - podmonitors
      - probes
      - prometheuses
      - prometheusrules
    verbs:
      - get
      - list
      - watch
    # 外部 Kubernetes 集群使用
    # - *
  - apiGroups:
      - ""
    resources:
      - namespaces
      - configmaps
      - secrets
      - nodes
      - services
      - endpoints
      - pods
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups: [ "" ]
    resources:
      - nodes/metrics
      - nodes/proxy
      - pods/proxy
    verbs:
      - get
      - list
      - watch
  - nonResourceURLs: [ "/metrics", "/metrics/cadvisor" ]
    verbs:
      - get
---
# 外部 Kubernetes 集群使用
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: prom-instance
  namespace: prom-instance
rules:
  - apiGroups: [ "", "extensions", "apps" ]
    resources: [ "*" ]
    verbs: [ "*" ]
```

## tke-kube-state-metrics

### 组件介绍

tke-kube-state-metrics 使用开源组件 kube-state-metrics，监听集群的 API server，生成集群内各种对象的状态指标。

### 部署在集群内的资源对象

| Namespace | kubernetes 对象名称 | 类型 | 资源量 | 说明 |
| --- | --- | --- | --- | --- |
| kube-system | tke-kube-state-metrics | Statefulset | 0.5C512Mi | 采集程序 |
| | | ServiceAccount | - | 权限载体 |
| | | ClusterRole | - | 采集权限相关 |
| | | ClusterRoleBinding | - | 采集权限相关 |
| | | Service | - | 采集程序对应服务，供服务发现使用 |
| | | Role | - | 分片采集权限相关 |
| | | RoleBinding | - | 分片采集权限相关 |
| | kube-state-metrics | ServiceMonitor | - | 采集配置 |

### 组件权限说明

#### 权限场景

| 功能 | 涉及对象 | 涉及操作权限 |
| --- | --- | --- |
| 监听集群内各种资源的状态 | 绝大部分 Kubernetes 资源 | list/watch |
| 获取采集 Pod 所在分片序号 | statefulsets,pods | get |

#### 权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tke-kube-state-metrics
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - secrets
      - nodes
      - pods
      - services
      - serviceaccounts
      - resourcequotas
      - replicationcontrollers
      - limitranges
      - persistentvolumeclaims
      - persistentvolumes
      - namespaces
      - endpoints
    verbs:
      - list
      - watch
  - apiGroups:
      - apps
    resources:
      - statefulsets
      - daemonsets
```

```
      - deployments
      - replicasets
    verbs:
      - list
      - watch
  - apiGroups:
      - batch
    resources:
      - cronjobs
      - jobs
    verbs:
      - list
      - watch
  - apiGroups:
      - autoscaling
    resources:
      - horizontalpodautoscalers
    verbs:
      - list
      - watch
  - apiGroups:
      - authentication.k8s.io
    resources:
      - tokenreviews
    verbs:
      - create
  - apiGroups:
      - authorization.k8s.io
    resources:
      - subjectaccessreviews
    verbs:
      - create
  - apiGroups:
      - policy
    resources:
      - poddisruptionbudgets
    verbs:
      - list
      - watch
  - apiGroups:
      - certificates.k8s.io
    resources:
      - certificatesigningrequests
    verbs:
      - list
      - watch
  - apiGroups:
      - storage.k8s.io
    resources:
      - storageclasses
      - volumeattachments
    verbs:
      - list
      - watch
  - apiGroups:
      - admissionregistration.k8s.io
    resources:
      - mutatingwebhookconfigurations
      - validatingwebhookconfigurations
    verbs:
      - list
```

```
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - networkpolicies
      - ingresses
    verbs:
      - list
      - watch
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs:
      - list
      - watch
  - apiGroups:
      - rbac.authorization.k8s.io
    resources:
      - clusterrolebindings
      - clusterroles
      - rolebindings
      - roles
    verbs:
      - list
      - watch
---
kind: Role
metadata:
  name: tke-kube-state-metrics
  namespace: kube-system
rules:
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - get
  - apiGroups:
      - apps
    resourceNames:
      - tke-kube-state-metrics
    resources:
      - statefulsets
    verbs:
      - get
```

# tke-node-exporter

## 组件介绍

tke-node-exporter 使用开源项目 node_exporter，部署在集群内的每个 Node 上，用来采集硬件和类Unix操作系统指标。

## 部署在集群内的资源

| Namespace | kubernetes 对象名称 | 类型 | 资源量 | 说明 |
| --- | --- | --- | --- | --- |
| kube-system | tke-node-exporter | DaemonSet | 0.1C180Mi*node数量 | 采集程序 |
| | | Service | - | 采集程序对应服务，供服务发现使用 |

| | | | | |
|---|---|---|---|---|
| | node-exporter | ServiceMonitor | - | 采集配置 |

## 组件权限说明

该组件不使用任何集群权限。

# 安全组开放说明

最近更新时间：2024-08-15 18:01:01

## 概述

本文介绍 Prometheus 监控服务 在 集成容器服务 过程中，托管集群和用户集群安全组需要开放的端口说明。同时介绍绑定出现安全组相关问题时的解决方式。

## 托管集群

托管集群安全组由 Prometheus 监控服务创建，一般情况下不需要修改。

### 安全组

| 规则 | 协议端口 | 策略 |
| --- | --- | --- |
| 入站规则 | TCP:9093,9090,10901,10902,9990,3000,8080,8008 | 允许 |
| 入站规则 | TCP:8100-8200 | 允许 |
| 出站规则 | ALL | 允许 |

### 端口说明

| 端口 | 作用 | 备注 |
| --- | --- | --- |
| TCP:8008 | proxy-server 监听 proxy-agent 连接端口 | - |
| TCP:8080 | 集群内部接口调用端口 | - |
| TCP:3000 | grafana 代理端口 | - |
| TCP:9990 | cm-notify 同步端口 | 即将下线 |
| TCP:10901,10902 | thanos sidecar 监听地址 | - |
| TCP:9090 | 配置 reload 端口，采集数据查询接口 | - |
| TCP:9093 | 告警用端口 | - |
| TCP:8100-8200 | proxy-server 监听采集用端口 | 由于采集端口范围为100个，所以关联集群最多不能超过100个 |

## 查看方式

登录 Prometheus 监控，选择实例的 ID/名称 > **实例诊断**，采集诊断选择**集成中心**，在**采集架构图**中就可以看到托管集群安全组，点击安全组可通过超链接跳转至安全组界面，即可查看托管集群安全组。

## 用户集群

用户集群安全组在用户创建节点时指定，若不指定则为默认安全组。

### 安全组

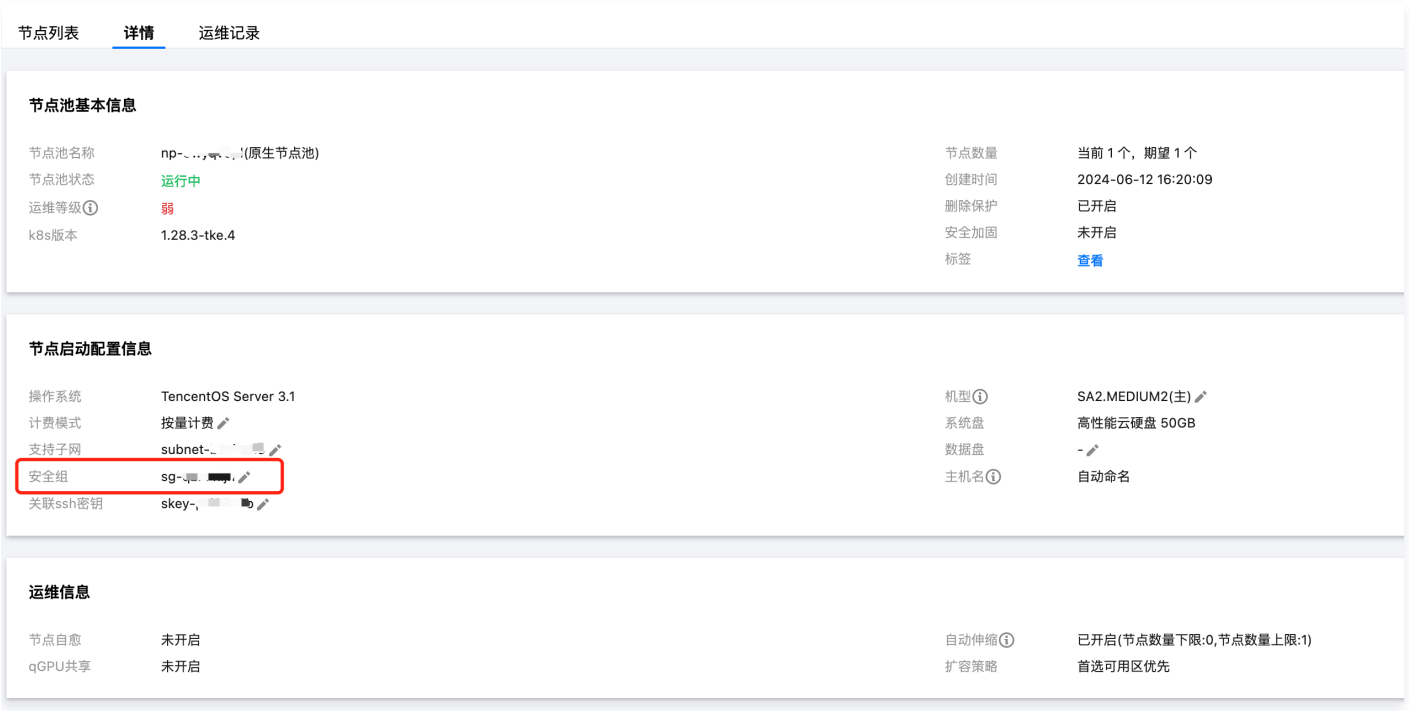
| 规则 | 集群类型 | 协议端口 | 策略 | 说明 |
| --- | --- | --- | --- | --- |
| 出站规则 | - | TCP:8008 | 允许 | 保证 proxy-agent 与 proxy-server 能够建立连接 |
| 入站规则 | 标准集群 | - | - | 标准集群无需开放端口 |
| 入站规则 | 独立集群 | TCP:9092,8180,443,10249,9100,60002,10252,10257,10259,10251等 | 允许 | 独立集群需额外开放 master 节点相关端口，保证 proxy-agent 能够拉取 master 节点相关监控数据 |

### 查看方式

登录 Prometheus 监控，选择实例的 ID/名称 > **数据采集**，点击集群 ID/名称即可跳转到该集群容器服务界面。

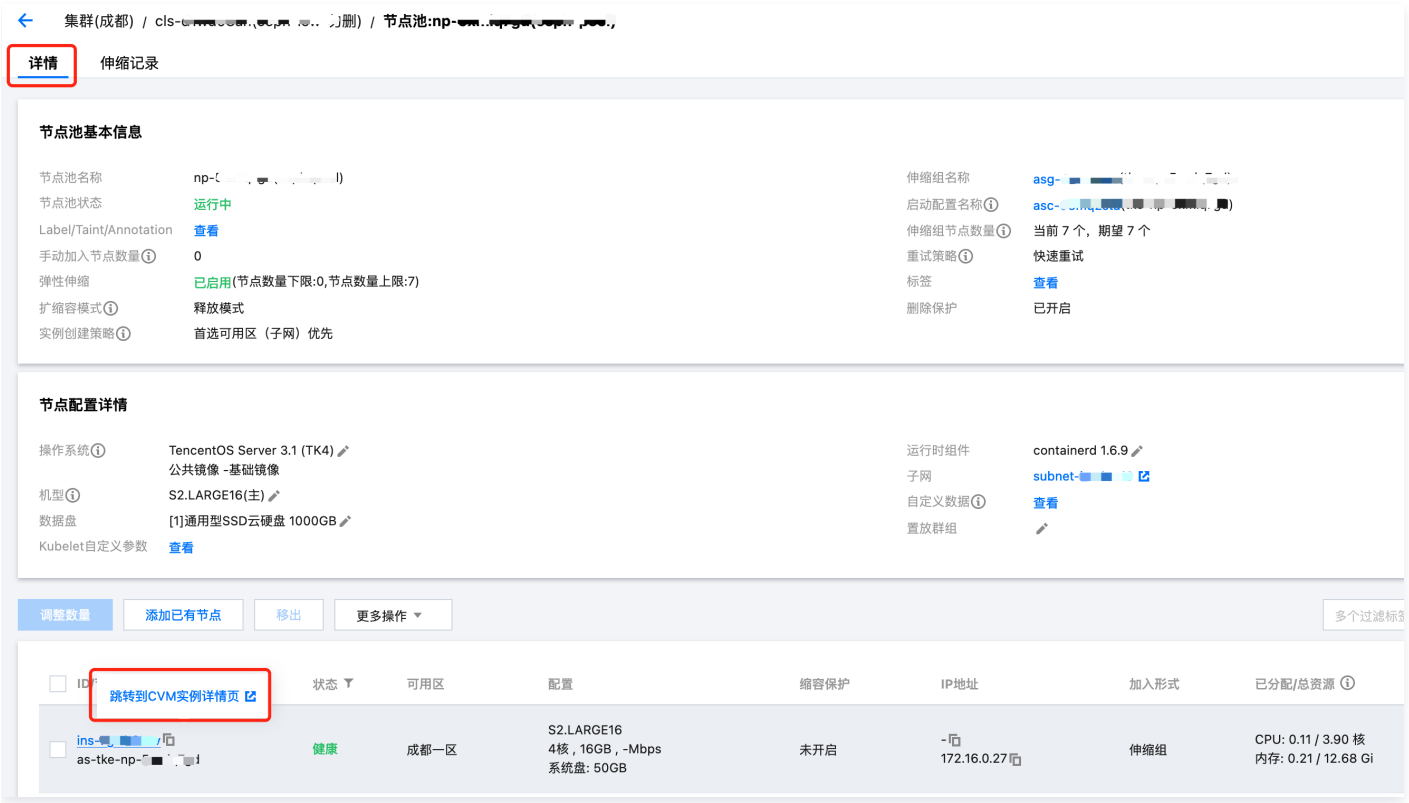
### 原生节点

点击**节点管理** > **Worker 节点** > **节点池**，点击节点池 id，在**详情**页即可看到安全组，在**腾讯云安全组**按安全组 id 进行搜索，查看具体规则即可。




### 普通节点

点击**节点管理** > **Worker 节点** > **节点池**，点击节点池 id，在**详情**页将鼠标放在节点 ID 上点击**跳转到 CVM 实例详情页**：




跳转至实例详情页面后，点击**安全组**，即可查看具体安全组信息：


## 超级节点

点击**节点管理** > **Worker 节点** > **节点池**，点击节点池 id，在**节点池基本信息**中即可查看安全组：


## 相关问题

### 问题描述

绑定状态异常，“安装 tmp-agent CR”步骤显示“context deadline exceeded”：

| | | | | |
|---|---|---|---|---|
| 安装 proxy-server 服务 | 完成 | 2024-06-12 11:04:52 | 2024-06-12 11:04:52 | N/A |
| 用户集群安装 proxy-agent | 完成 | 2024-06-12 11:04:52 | 2024-06-12 11:04:57 | N/A |
| 安装基础采集配置 | 完成 | 2024-06-12 11:04:55 | 2024-06-12 11:05:00 | N/A |
| 安装 tmp-agent CR | 异常 | 2024-06-12 11:04:57 | | {Reason:get resourceInformer failed,Object:TMPAgent/prom-[redacted]/tke-cls-[redacted] Message:failed to sync cache for ServiceMonitor informer}: context deadline exceeded |

用户集群内安装组件说明

确定 刷新

### 问题处理

#### vpc 是否相同，或者打通

1. 点击用户集群链接，打开关联集群，查看集群节点网络（即 vpcid）：


2. 在 Prometheus 实例的**基本信息**页面，点击**所属网络**，打开后即可查看集群网络：


3. 对比 vpcid，若不一致，查看 vpc 是否通过云联网打通。若云联网没有打通，则需要关联云联网，打通两边的 vpc，或者在关联集群时勾选**创建公网 CLB**。若云联网已打通，仍不能关联成功，需查看云联网是否达到带宽上限，若已达到则需要调大云联网带宽上限。

- 关联云联网：


- 勾选创建公网 CLB：


- **升级云联网带宽。进入 私有网络 > 云联网 页面，选择实例 ID > 带宽管理 > 升级带宽：**


**安全组是否放通**

1. 查看用户集群安全组，查看方式见 用户集群安全组查看方式，查看规则是否与要求一致；
2. 若用户集群为独立集群，查看 Master&Etcd 安全组信息，点击**节点管理 > Master&Etcd > 节点池**，点击节点池 id，并将鼠标放在节点 ID 上，点击**跳转到CVM实例详情页**，然后在 CVM 的**安全组**页面即可查看具体安全组信息：


检查安全组规则是否符合要求。

# 批量安装 Node Exporter

最近更新时间：2024-11-04 09:19:51

Prometheus 监控服务支持使用 CVM Node Exporter 采集云服务器（Cloud Virtual Machine，CVM）实例上的指标。目前已支持用户在控制台批量选择 CVM 实例并安装 Node Exporter。您可以参考下文进行操作。

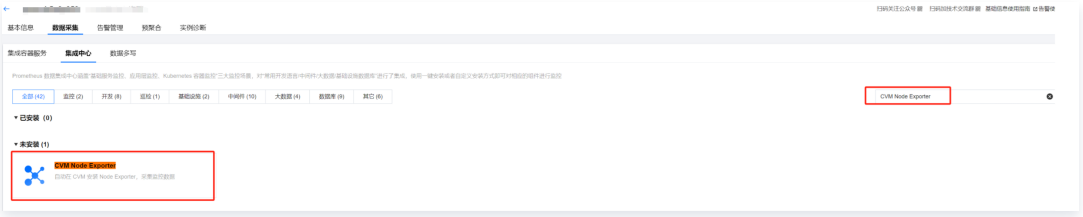
## 前提条件

CVM 已 安装腾讯云自动化助手（TencentCloud Automation Tools，TAT）。

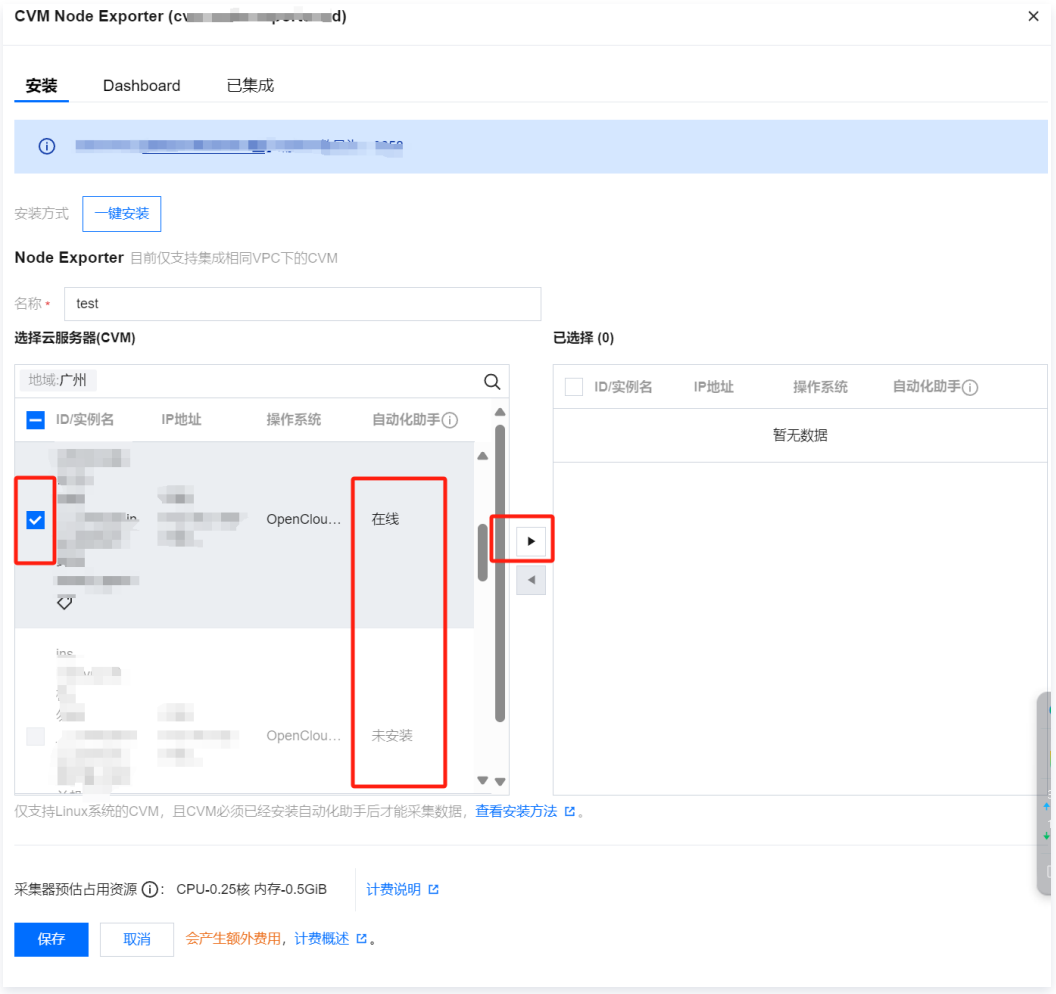
## 操作步骤

> **说明**
> 目前仅支持集成相同 VPC 下的 CVM。

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，单击**数据采集 > 集成中心**。
4. 在集成中心搜索 **CVM Node Exporter**，找到后单击它即会弹出一个安装窗口。




5. 进入 Node Exporter 安装页面，勾选已安装腾讯云自动化助手的 CVM 实例，并单击右侧 按钮，完成后单击**保存**。
   您可以在实例列表的**自动化助手**列中可查看自动化助手安装状态。




6. 保存成功后，等待安装。如下图，若运行状态为已部署，则安装成功。

# CVM 进程监控

最近更新时间：2025-01-16 17:21:32

Prometheus 监控服务支持使用 CVM Node Exporter 采集云服务器（Cloud Virtual Machine，CVM）实例上的指标。目前已支持用户在控制台批量选择 CVM 实例并安装 Node Exporter。您可以参考下文进行操作。

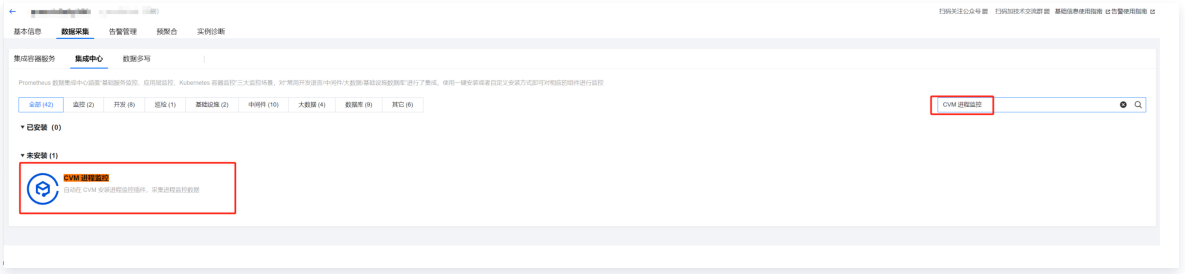
## 前提条件

CVM 已安装 腾讯云自动化助手（TencentCloud Automation Tools，TAT）。

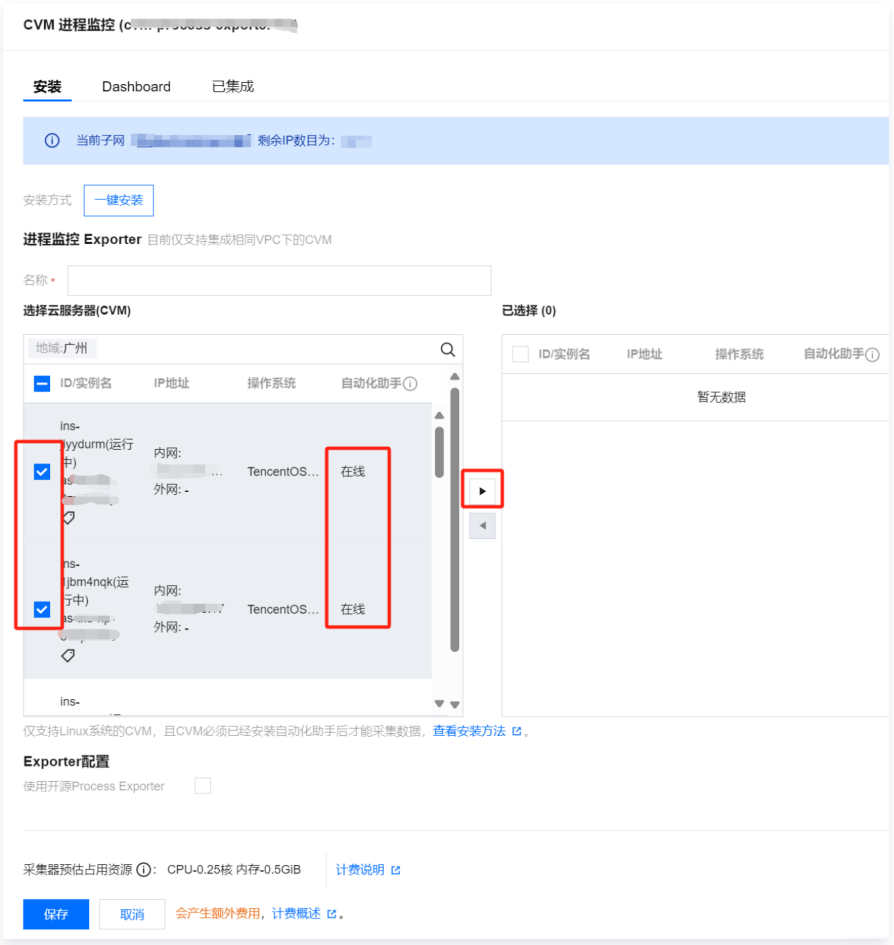
## 操作步骤

> **说明：**
> 目前仅支持集成相同 VPC 下的 CVM。

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **CVM 进程监控**，找到后单击它即会弹出一个安装窗口。




5. 进入 CVM 进程监控安装页面，勾选已安装腾讯云自动化助手的 CVM 实例，并单击右侧 ▶ 按钮，完成后单击**保存**。
   您可以在实例列表的**自动化助手**列中可查看自动化助手安装状态。

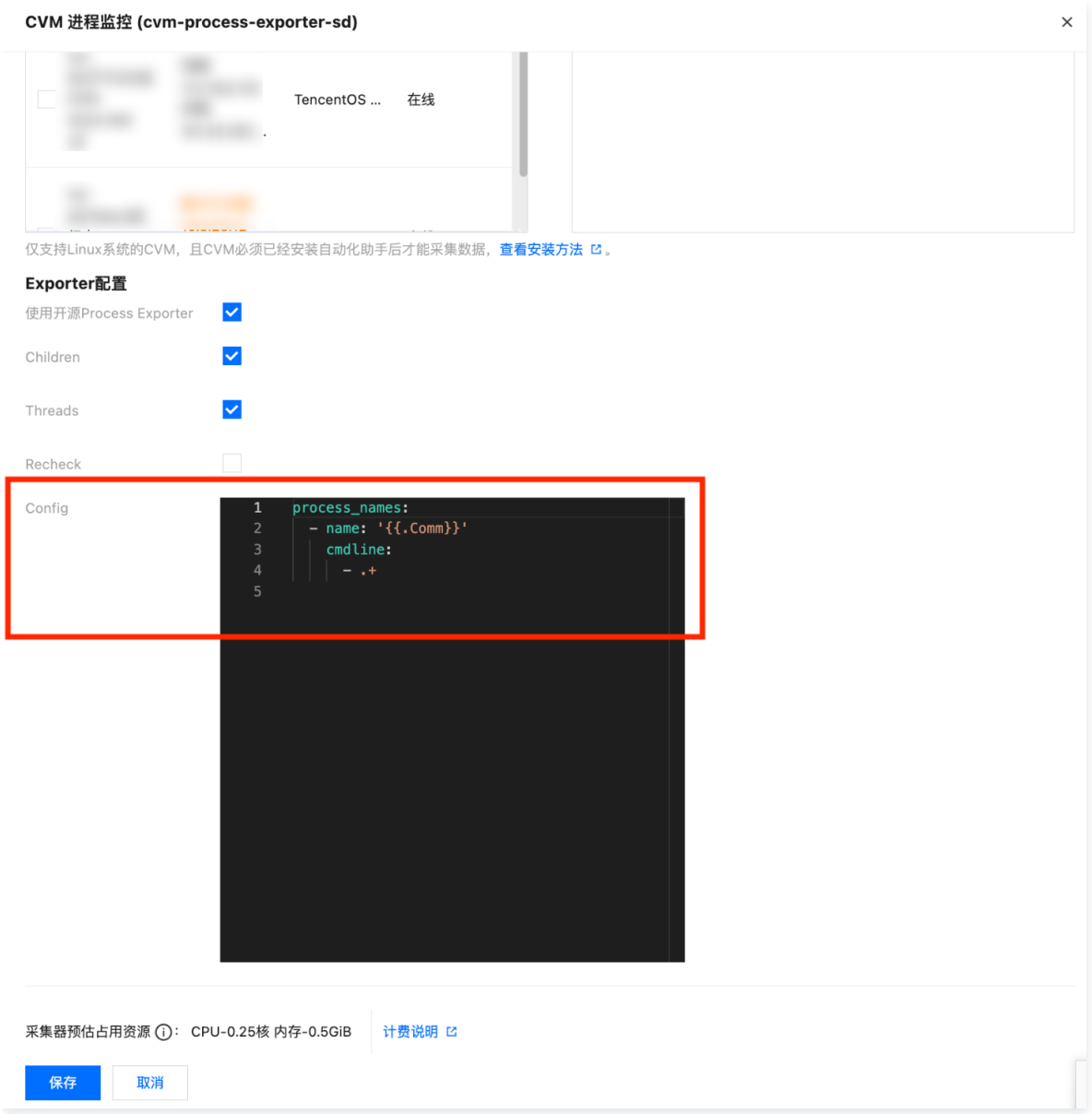



6. 为了指标的完备性及配置的灵活性，建议勾选**使用开源 Process Exporter**，开源 exporter 详情参见 process-exporter 。


7. 保存成功后，等待安装。如下图，若运行状态为已部署，则安装成功。


## 自定义进程标识

若您使用的是开源 exporter（见上述操作步骤 6），则可通过 `groupname` 标签，提取想要监控的进程的标识。
在默认配置下，`groupname` 的值是进程执行程序的基本名称（不包含路径和扩展名），如下图所示：




该默认配置可根据需要修改。例如要想将完整命令提取作为 `groupname`，可以将开源 exporter 的配置部分改为：

```
process_names:
  - name: "{{.Matches.fullCmd}}"
    cmdline:
    - (?P<fullCmd>.*)
```

**修改后的效果：**

例如一条进程执行命令为 `/usr/local/bin/python3.8 /my/path/test.py` 。

- 使用默认配置时，提取出的 `groupname` 标签的值为 `python3.8` 。
- 如果按上述配置，改为提取完整命令，则 `groupname` 标签的值为 `/usr/local/bin/python3.8 /my/path/test.py` 。

**用法示例：**如果想针对该进程告警，则 PromQL 里可以用 `groupname` 将它筛选出来：

`namedprocess_namegroup_num_procs{groupname="/usr/local/bin/python3.8 /my/path/test.py"}==0` 。

# 控制台操作指南
# 实例
# 创建实例

最近更新时间：2024-07-05 11:48:11

本文以自定义配置方式为例，指导您如何创建 Prometheus 实例。

## 前提条件

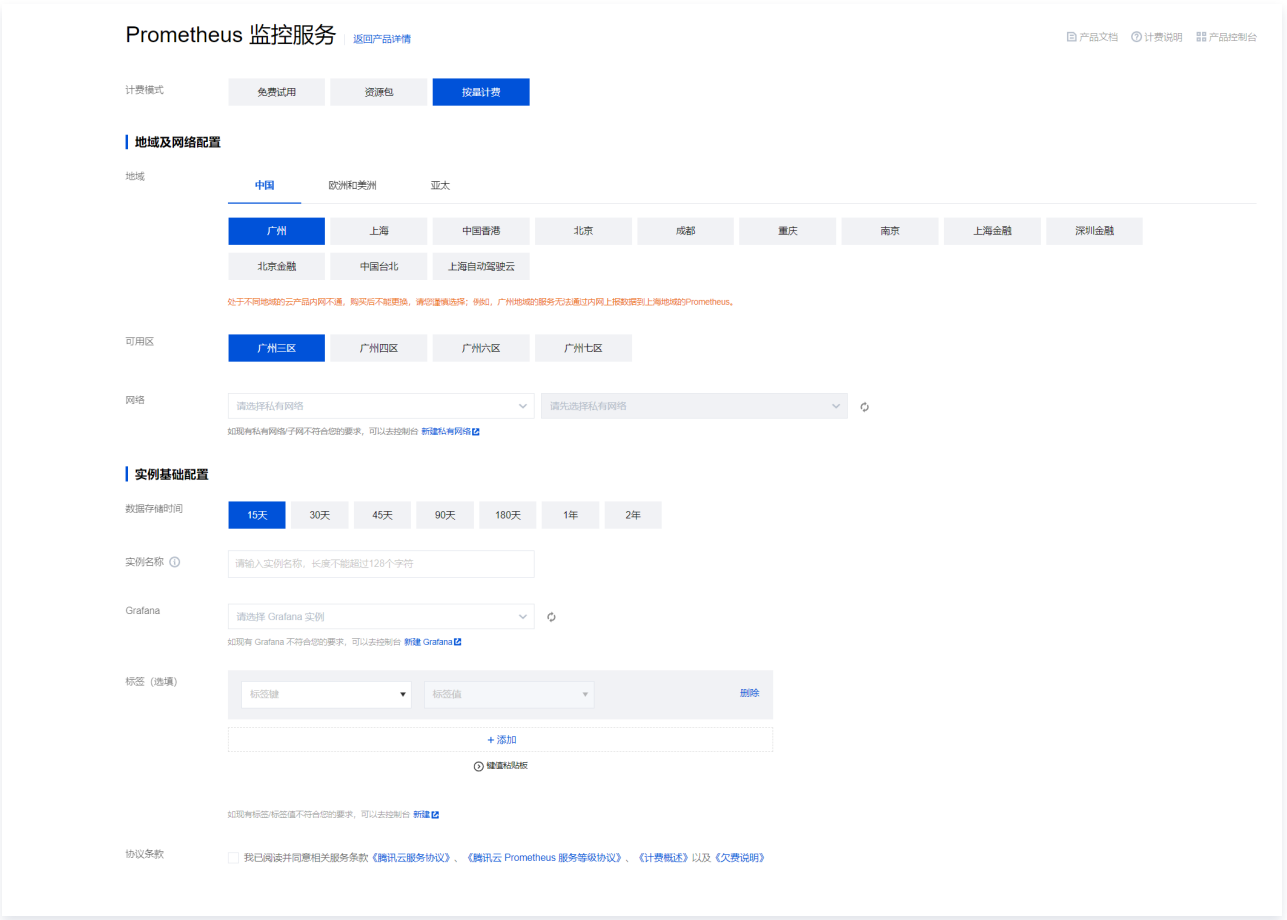
在创建 Prometheus 实例前，您需要完成以下工作：

- 注册腾讯云账号，并完成 实名认证 。
- 需要在目标地域 创建一个私有网络，并且在私有网络下的目标可用区 创建一个子网 。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 在 Prometheus 监控页面单击**新建**，根据页面提示，配置以下信息：

| 类别 | 必选/可选 | 配置说明 |
| --- | --- | --- |
| 计费模式 | 必选 | 目前支持资源包和按量付费模式，新用户支持免费试用15天。资源包具有较高的折扣优惠，目前资源包支持境内外多个城市，您可以自行选择购买。 |
| 地域 | 必选 | 处于不同地域的云产品内网不同，购买后不能更换，请您谨慎选择；例如，广州地域的服务无法通过内网上报数据到上海地域的 Prometheus。 |
| 可用区 | 必选 | 根据您实际需求选择。Prometheus 存储层目前并不区分可用区，同地域购买不同可用区的实例并不能作为多可用区灾备方案（建议使用云联网等方式上报到其它地域作为替代）。 |
| 网络 | 必选 | 表示在腾讯云上构建的逻辑隔离的网络空间，一个私有网络由至少一个子网组成。系统会为您在每个地域提供默认私有网络和子网。如现有的私有网络/子网不符合您的要求，详情请参见 新建私有网络 和 新建子网 进行创建。选择后只展示在同一私有网络的 Grafana 。 |
| 套餐包选择（仅套餐包类型可选） | 必选 | 支持基础版年包、升级版年包和豪华版年包。具体价格以购买页为准。 |
| 数据存储时间 | 必选 | 资源包只能选择存储时长为30天和180天的时长；按量付费不同的存储时长价格有差异，详情请参见 按量付费产品定价 。 |
| 实例名称 | 必选 | 用户自定义 Prometheus 实例名称。 |
| Grafana | 可选 | 选择对应的 Grafana 实例（仅展示与所选网络同一 VPC 的 Grafana 实例，若没有符合的 Grafana ，请参见 操作指引 创建） |
| 标签 | 可选 | 设置标签之后可以用于从不同维度对资源分类管理。具体可参见 标签说明。 |
| 到期或用完处理方式（仅套餐包类型可选） | 可选 | 勾选后，资源包到期或资源包内任一种用量额度用完后，自动转为按量付费。 |




4. 配置完后，勾选服务条款并单击**立即购买**支付即可。

# 搜索实例

最近更新时间：2024-07-05 11:48:11

默认情况下，Prometheus 监控服务控制台展示的是当前地域下 Prometheus 实例。为了帮助用户快速搜索出当前地域下的 Prometheus 实例，腾讯云提供搜索功能，目前可通过实例 ID、实例名称、实例状态、可用区、IPv4 地址、标签等资源属性维度进行过滤。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 在搜索框中，根据实际需求，输入需搜索的内容，单击进行搜索。


4. 支持用户根据不同的维度来过滤实例列表，目前支持以下几种维度：

| 过滤项 | 说明 |
| --- | --- |
| 实例 ID | 支持多个实例 ID 过滤，每个实例 ID 仅支持完全匹配过滤，支持直接输入实例 ID 来快速过滤。 |
| 实例名称 | 不支持多个名称过滤，支持模糊匹配过滤。 |
| 实例状态 | 支持用户多个状态选择过滤，同时也支持在列表头进行快速过滤。 |
| 可用区 | 支持用户多个可用区过滤，切换地域之后显示该地域下的 Prometheus 对应的可用区，同时也支持在列表头进行快速过滤。 |
| IPv4地址 | 支持用户通过 IPv4 地址进行精准查询。 |
| 标签 | 支持多个标签组合来过滤实例，同时也支持直接单击实例列表中对应的标签值来进行过滤。 |

# 修改实例名称

最近更新时间：2024-08-16 11:52:01

为了方便用户在 Prometheus 监控服务控制台上进行 Prometheus 实例管理，可快速辨识出 Prometheus 实例的名字，腾讯云支持给每台实例命名，并且可随时更改，立即生效。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在实例列表中，选择需要被修改实例名称的 Prometheus 实例，单击右侧的**更多** > **实例配置** > **修改名称**。
4. 在弹出的“**修改名称**”窗口中，输入新实例名称，单击**确定**即可。

# 销毁实例

最近更新时间：2024-10-31 18:11:52

当您不再使用某个实例时，可以对实例进行销毁，被销毁的实例会处于停服状态。对于停服状态的实例，您可以根据不同场景和需求进行重建实例。

## 相关影响

当实例进入停服状态时，实例数据相关影响如下：

- **IP**：对应的 IP 地址还会保留，但不对外提供正常的服务。
- **Grafana**：无法再通过对应的域名访问 Grafana。
- **数据**：对应的实例数据还会保留相应天数，具体天数以控制台销毁过程中的确认信息为准。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在实例列表中，选择需要被销毁的 Prometheus 实例，单击右侧的**更多** > **实例状态** > **销毁/退还**。


4. 由于销毁操作属于高危操作，需在弹出的**销毁**窗口中，完成销毁操作步骤，单击**确定**即可。

## 强制释放

当您实例进行销毁/退还处理，实例7天内处于停服状态，在此期间的数据将继续保留。实例仍存在 Prometheus 监控服务列表中，您可以执行强制释放，释放后将不会在列表中展示，释放后该实例下所有数据将被清除且不可恢复。

1. 在实例列表中，选择需要被销毁的 Prometheus 实例，单击右侧的**更多** > **实例状态** > **强制释放**。
2. 由于释放操作属于高危操作，实例下所有数据将被清除且不可恢复。请您确认后在弹出的**强制释放**窗口中，完成释放操作步骤，单击**确定**即可。

# 重建实例

最近更新时间：2024-08-16 11:52:01

如果您需要对停服或者异常状态的实例进行重建，您可以在控制台上对相应的实例进行重建操作。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在实例列表中，选择需要被销毁的 Prometheus 实例，单击右侧的**更多** > **实例状态** > **重建**。
4. 在弹出的“重建操作须知”窗口中，单击**确定**即可。


> **说明：**
> 如果当前实例状态为运行中，无法进行重建操作。

# 修改存储时长

最近更新时间：2024-10-31 18:11:52

> **说明：**
> 资源包不支持修改存储时长。

Prometheus 支持您在创建实例后修改数据存储时长。存储时长越长，实例单价越高，您可以参见 按量计费说明 进行合理调整。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在实例列表中，找到需要被修改的 Prometheus 实例，单击右侧的**更多 > 实例配置 > 修改存储时长**。


4. 在弹框中选择需要修改的存储时长，单击**确定**即可。


> **说明：**
> - 修改成功后，第二天0点开始，新采集的数据将按照新的存储时长存储和新的计费单价进行计费。
> - 历史数据的存储时长仍按照修改前的存储时长存储。

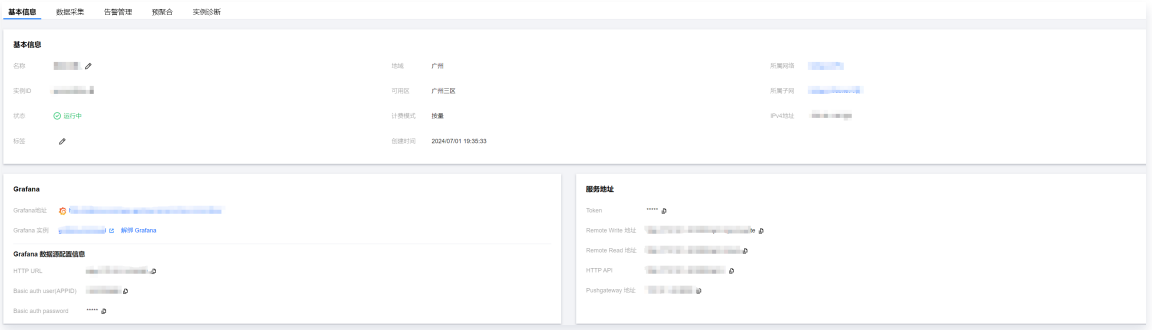
# 查看实例基本信息

最近更新时间：2024-07-05 11:48:11

为了方便用户查看 Prometheus 实例基本信息，您可以在实例列表页面选择对应的实例，进入实例管理页查看实例的基本信息。

## 操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在实例列表中，选择需要查看的 Prometheus 实例，单击实例的 ID/名称。




4. 在基本信息页支持如下操作：
   - 修改实例名称。
   - 编辑实例对应的标签。
   - 绑定 / 解绑 Grafana 实例。

# 容器监控
# 集成容器服务

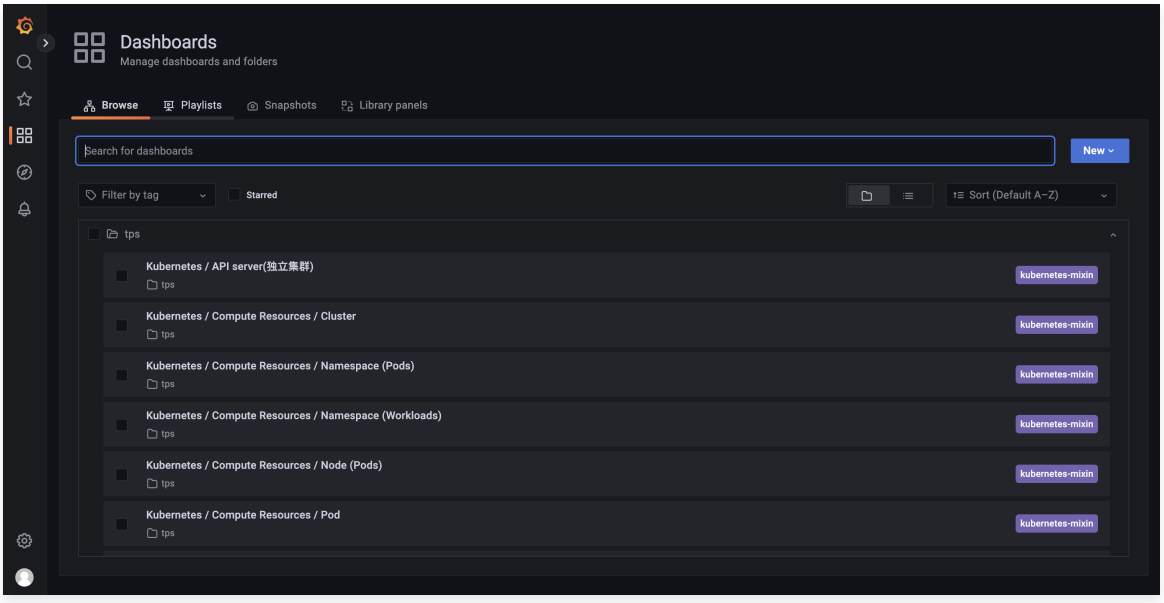
最近更新时间：2024-11-22 16:53:02

集成容器服务后即可对腾讯云容器服务业务场景进行监控。本文将为您介绍如何集成容器服务。
腾讯云容器服务（Tencent Kubernetes Engine，TKE）是基于原生 Kubernetes 提供以容器为核心的解决方案，解决用户开发、测试及运维过程的环境问题、帮助用户降低成本，提高效率。而 Kubernetes 是一款由 Google 开发的开源的容器编排工具，在 Google 已使用超过15年。作为容器领域实施的标准，Kubernetes 可以极大地简化应用的管理和部署复杂度。通过与容器服务集成，可以大大简化用户通过 Prometheus 来监控 Kubernetes 状态及其运行在上面的服务。

> **说明：**
> 为保证正常运行，存量实例在编辑采集配置和新关联集群时会自动更新组件版本，更新过程中可能会造成已关联的集群数据断点。

## 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 实例列表中，单击新建的**实例 ID/名称** 。
3. 进入 Prometheus 管理中心，在顶部导航栏中单击**数据采集**。
4. 在集成容器服务页面进行下列操作：
   - 关联集群：将集群和 Prometheus 实例关联，参见指引 关联集群 。
   - 数据采集配置：支持通过控制台新增或 Yaml 文件配置两种方式，创建新的数据采集规则来监控您的业务数据，参见指引 数据采集配置 。
   - 精简监控指标：选择需要上报的指标，避免不必要的费用支出，参见指引 精简监控指标 。
5. 完成以上操作后，在 Prometheus 实例列表点击对应的 Grafana 图标，查看您容器服务的监控数据。




## 关联集群

> **注意：**
> 关联集群成功后将在集群中安装监控数据采集插件，该插件在解除关联的同时会被删除。当前支持跨 VPC 关联，支持在同一个监控实例内监控不同地域不同 VPC 下的集群。

### 前提条件

- 已登录 腾讯云可观测平台控制台 ，并创建集群。
- 已创建 Prometheus 实例 。

## 操作步骤

### 关联腾讯云上 Kubernetes 集群

1. 登录 腾讯云可观测平台控制台，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要关联集群操作的实例名称，进入该实例详情页。
3. 在顶部导航栏中单击**数据采集 > 集成容器服务 > 关联集群**。
4. 在弹出的“关联集群”窗口，选择相关集群。


- **集群类型**：容器服务的标准集群、 弹性集群、注册集群、边缘集群。
- **跨 VPC 关联**：开启后支持在同一个监控实例内监控不同地域不同VPC下的集群。
  - **创建公网 CLB**：若您的实例所在的 VPC 与想要关联集群网络互通则无需创建；若您的实例所在的 VPC 与想要关联的集群网络不互通，则必须勾选创建公网 CLB，否则无法进行跨 VPC 集群的数据采集。例如：若您实例所在的 VPC 与想要关联集群所在的 VPC 已经通过 云联网 打通，则不需要创建公网 CLB。

- **集群所在地域**：选择集群所在地域。
- **集群**：选择需要关联的集群，支持多选。
- **全局标记**：用于给每个监控指标打上相同的键值对。

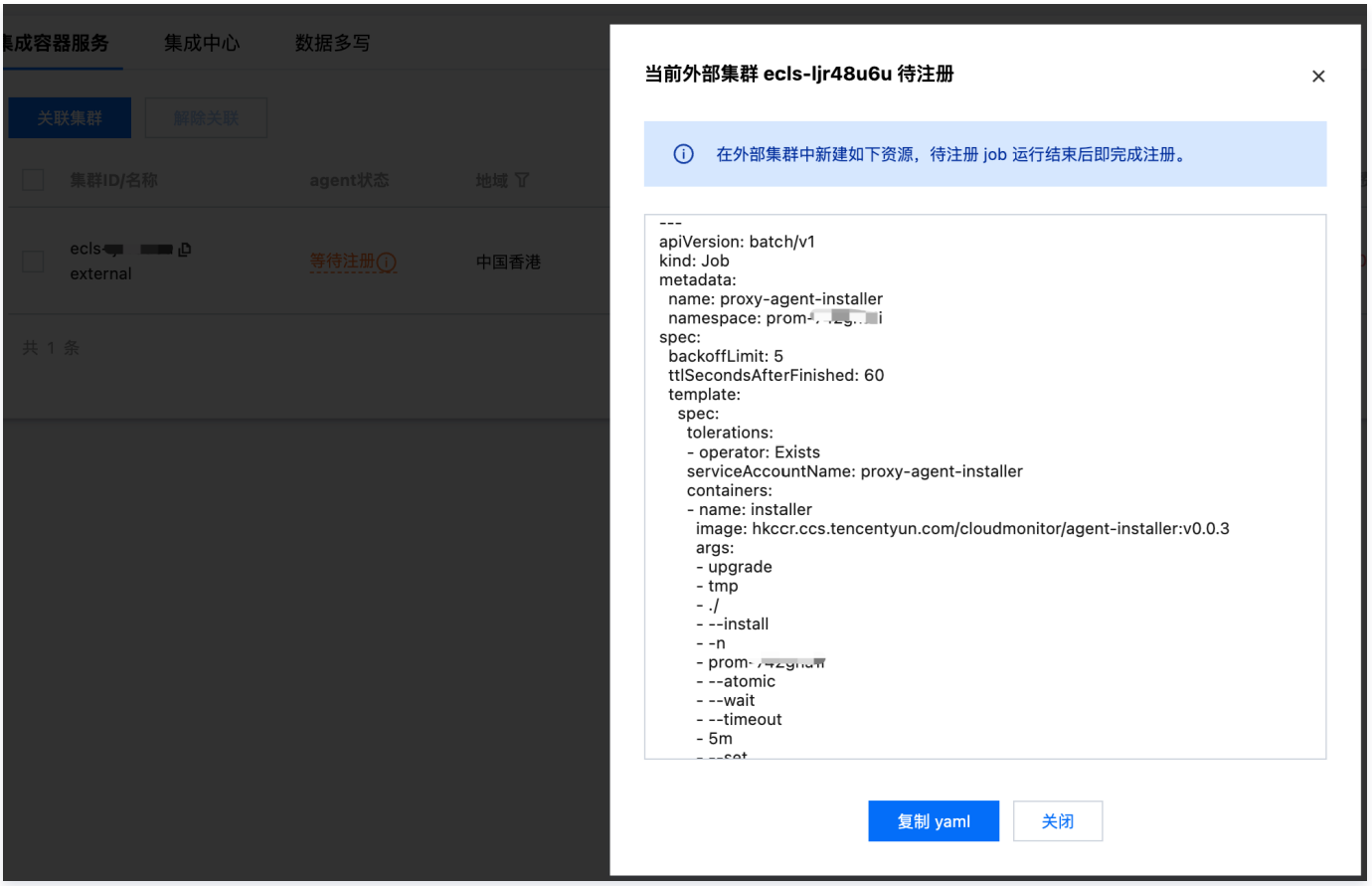
5. 单击**确定**即可将所选集群和当前监控实例关联。

### 关联外部 Kubernetes 集群

1. 登录 腾讯云可观测平台控制台，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要关联集群操作的实例名称，进入该实例详情页。
3. 在顶部导航栏中单击**数据采集 > 集成容器服务 > 关联集群**。
4. 在弹出的“关联集群”窗口，选择外部集群。


- **外部集群名称**：给集群取个名称。
- **外部集群所在地域**：选择要注册集群所在地域或临近地域。
- **采集方式**：如果外部集群已经和公有云 VPC 打通选择内网接入，否则选择公网。公网接入需要付额外的公网 CLB 费用。
- **全局标记**：用于给每个监控指标打上相同的键值对。

5. 单击**确定**开始初始化，会生成一个集群 ID。
6. 注册外部集群。




- 初始化成功后，点击**等待注册**弹出 Job 安装 yaml。Job 执行 helm 命令安装采集相关组件，组件和权限详细说明请参见 TKE 集群内安装组件说明 。Job 主要参数如下：

| 参数 | 说明 |
| --- | --- |
| timeout | helm 安装等待的超时时间，超时未完成会自动回滚 |
| proxyAgent.enabled | 是否安装 proxy-agent，必须设置为 true 否则注册不成功 |
| proxyAgent.instanceId | 实例 ID |
| proxyAgent.instanceToken | 实例 Token |
| proxyAgent.clusterId | 集群 ID |
| proxyAgent.clusterType | 集群类型 |
| proxyAgent.serverAddress | 实例为采集注册提供的公网地址，用户集群只有能正常访问这个地址才能注册成功 |
| proxyAgent.image | proxy-agent 镜像 |
| kubeStateMetrics.enabled | 是否安装 kube-state-metrics 组件，如果不需要或者已安装可以设置为 false |
| kubeStateMetrics.image | kube-state-metrics 镜像 |
| nodeExporter.enabled | 是否安装 node-exporter 组件，如果不需要或者已安装可以设置为 false |
| nodeExporter.image | node-exporter 镜像 |

- 创建并查看 Job 运行结果。

```
# 安装注册任务
kubectl apply -f <yaml>
# 设置 namespace
export KUBE_NS=<实例 ID>
```

```
# 查看 Job 执行状态
kubectl get job proxy-agent-installer -n ${KUBE_NS}
# 查看 Job POD，有失败可以查看出错日志
kubectl get pods -l job-name=proxy-agent-installer  -n ${KUBE_NS}
```

- 查看 proxy-agent 日志，正常注册日志应该包含 conn is active，否则用户要检查集群内能否正常访问 proxyAgent.serverAddress 指定的 IP:Port。

```
# 获取 proxy-agent pod 名
export KUBE_POD=`kubectl  get pods -l k8s-app=proxy-agent -n ${KUBE_NS}|sed '1d'|head -1|awk '{print $1}'`
# 查看 proxy-agent 日志
kubectl logs ${KUBE_POD} -n ${KUBE_NS}
```

7. 注册成功后等待1 – 2分钟，控制台上 **agent 状态**会变成**运行中**。接下来就可以像云上 Kubernetes 集群一样操作外部集群。

## 解除关联

1. 登录 腾讯云可观测平台控制台，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择解除关联的实例名称，进入该实例详情页。
3. 在**数据采集 > 集成容器服务**页面，单击实例右侧的**更多 > 解除关联**。
4. 在弹出的“解除关联集群”窗口，单击**确定**即可解除关联。

> **说明：**
> 如需在容器服务控制台关联 Prometheus 集群，请参见 容器服务关联 Prometheus 。

# 容器服务关联 Prometheus

最近更新时间：2024-11-27 11:45:23

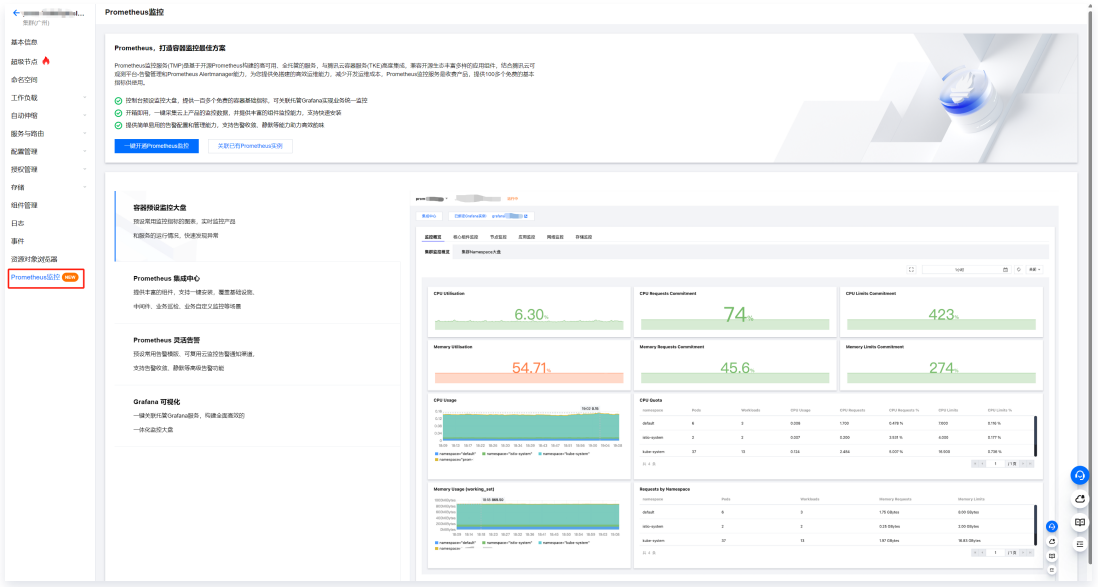
容器场景监控是 Prometheus 监控的主要用户场景，本文将为您介绍集群如何在容器场景监控下关联 Prometheus 监控，且关联 Prometheus 监控后，可以直接在集群页面查看监控数据图表，大大提高了用户的使用体验。

## 前提条件

账户下已有集群且暂未关联 Prometheus 监控。

## 操作步骤

1. 登录 容器服务控制台 。
2. 在左侧导航栏中选择**集群**，单击一个**集群名称/ID** 进入集群详情页。
3. 在集群详情页的左侧导航栏选择 **Prometheus 监控**，如下图所示。




4. 在弹出的 Prometheus 监控页面用户可选择 一键开通 Prometheus 监控 或者 关联已有 Prometheus 实例 。

## 方式一：一键开通 Prometheus 监控

> **说明：**
> 一键开通仅支持默认的按量付费计费模式和默认的存储时间15天。如需创建其他规格的 Prometheus 实例，请前往 Prometheus 控制台 创建实例 。

在弹出的 Prometheus 监控页面选择**一键开通 Prometheus 监控**，选择子网并勾选相关协议后点击**确定**。


## 方式二：关联已有 Prometheus 实例

在弹出的 Prometheus 监控页面选择**关联 Prometheus 实例**，会出现以下几种情况，请根据自身情况选择对应处理方案。

- 该账号下没有任何 Prometheus 实例，请按照下图提示去 新建 Prometheus 实例 。

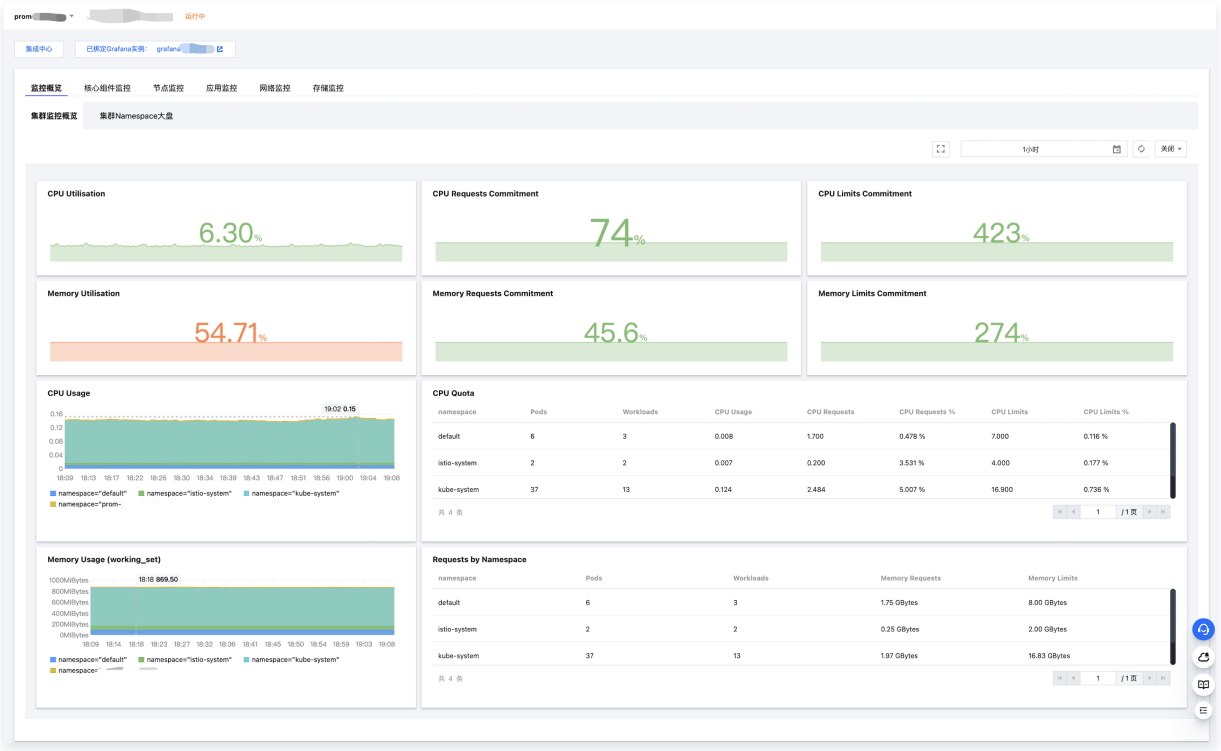

- 该账号下有 Prometheus 实例，但当前 VPC 下无实例，用户可选择 创建实例 或开启**跨 VPC 关联**。


- 该账号当前 VPC 下有 Prometheus 实例，客户选择当前 VPC 内的 Prometheus 实例。如下图所示：


- **用户选择开启跨 VPC 关联，会展示创建公网 CLB 实例功能，同时，可选择的实例列表是当前账号下的所有 Prometheus 实例。如下图：**


## 容器指标监控图表

1. **当成功关联 Prometheus 监控以后，展示容器常用指标的监控图表。**

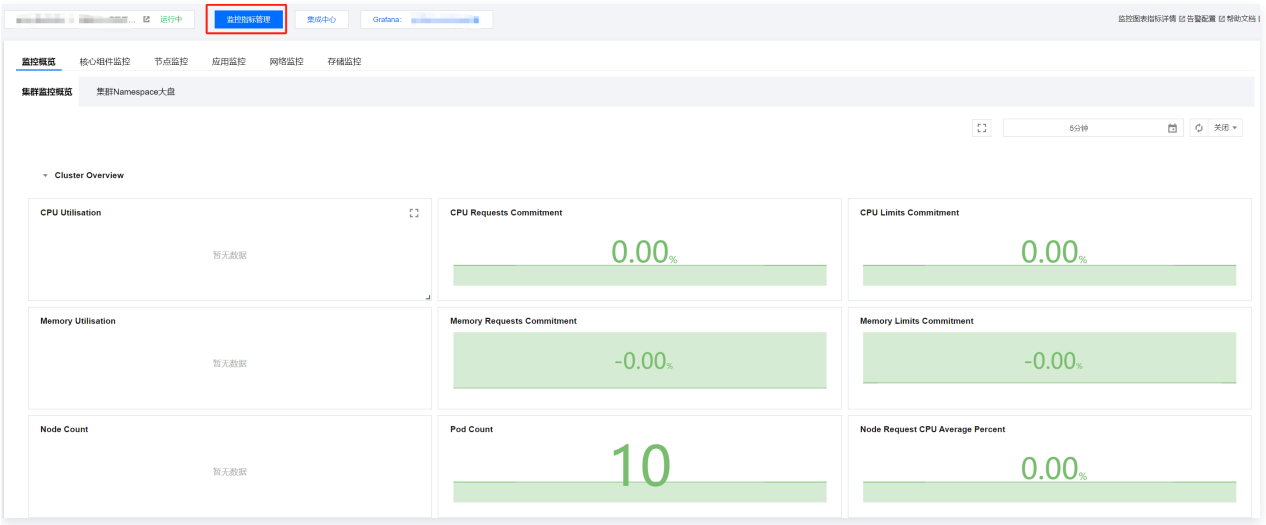



2. **若需要监控容器中运行的其他组件，可以点击集成中心，跳转到 Prometheus 监控的 集成中心 安装其他类型的监控组件。若当前图表不满足需求，可以点击绑定 Grafana 服务，在 绑定 Grafana 服务 后前往 Grafana平台配置更多指标的监控图表。**

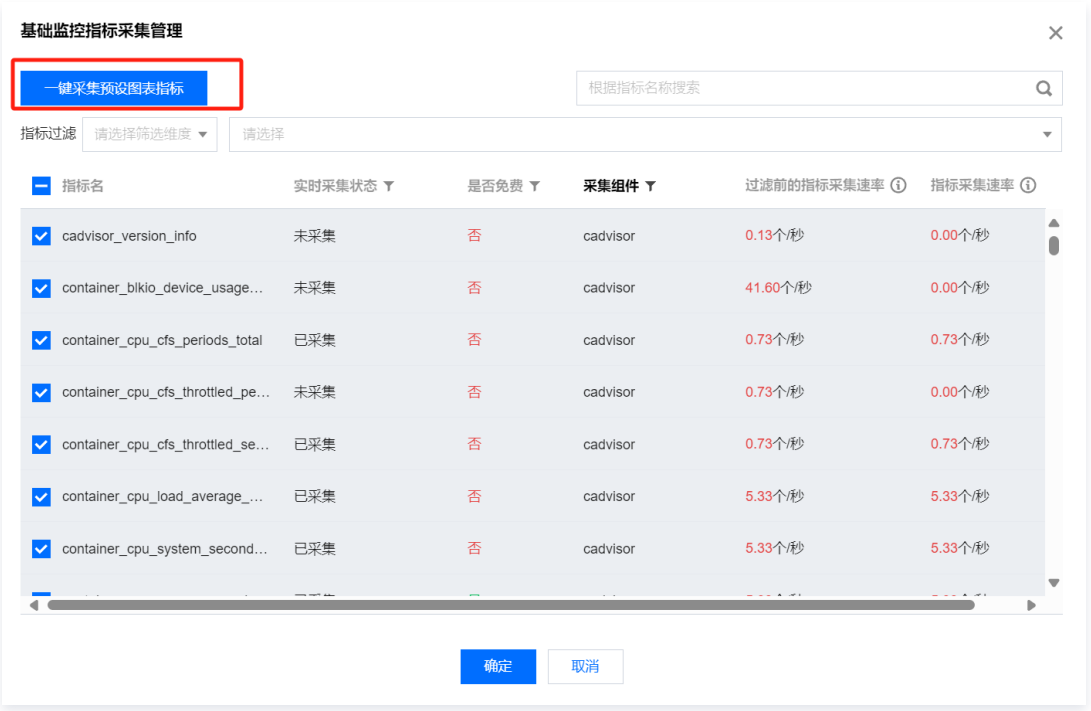
> **说明：**
> 如需在 Prometheus 控制台关联容器集群，可参考 集成容器服务 。

## 配置指标数据

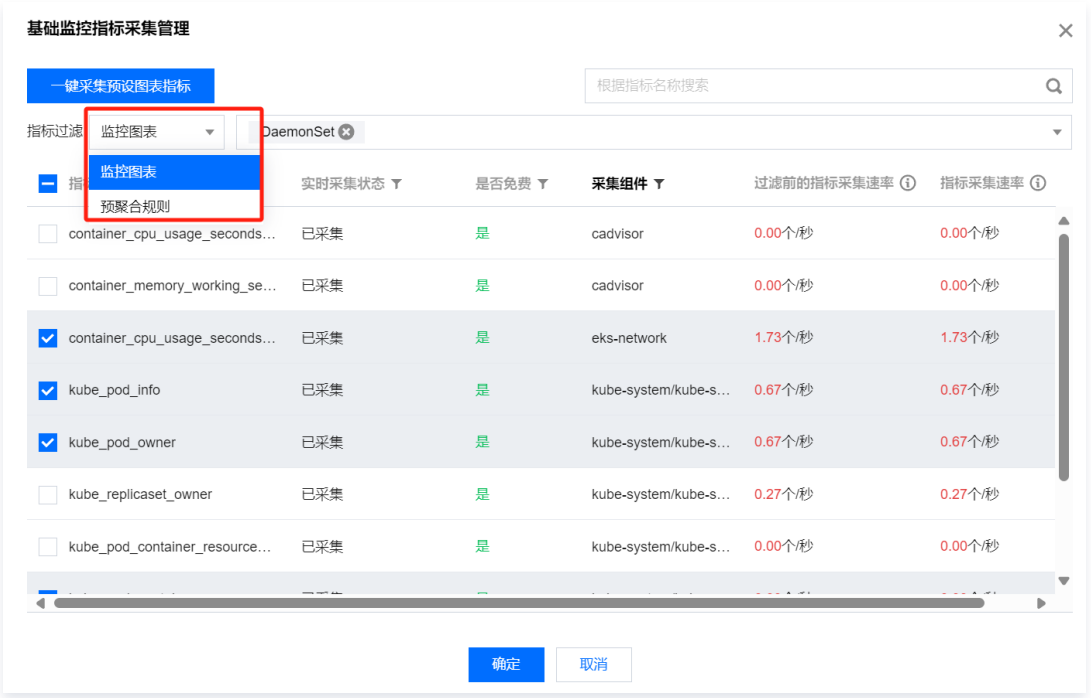
进入监控图表页面后，我们会发现部分图表有数据，部分图表显示**暂无数据**。这是由于免费的指标会自动接入，免费指标可参考 按量付费免费指标；而无数据的指标可以参考 容器监控图表 。配置指标的步骤如下：
点击监控图表页面上方的**监控指标管理**，即可跳转到基础监控指标采集管理页面。




- 若需要所有内嵌图表都显示有数据，可以在弹出的方框中选择**一键采集预设图表指标**后会再弹出一个确认窗口，点击**确定**将会采集 Prometheus 监控预设容器图表相关的全部指标。




- **若只需要采集部分指标，可以通过监控图表和预聚合规则进行指标筛选，勾选自己需要的指标后再点击确定即可。如下图所示：**




> **说明：**
> 由于有些指标属于预聚合指标，此类指标需要添加预聚合规则，若默认预聚合规则是关闭状态，我们需要前往 Prometheus 控制台，进入**实例 ID > 预聚合 > 开启默认预聚合规则**。

# 集成中心

最近更新时间：2025-01-15 11:56:12

Prometheus 监控服务对常用的开发语言/中间件/大数据/基础设施数据库进行了集成，支持一键安装和自定义安装方式，用户只需根据指引即可对相应的组件进行监控，同时提供了开箱即用的 Grafana 监控大盘。集成中心涵盖了基础服务监控、组件监控、应用层监控、业务监控等监控场景，方便您快速接入并使用。

## 支持服务列表

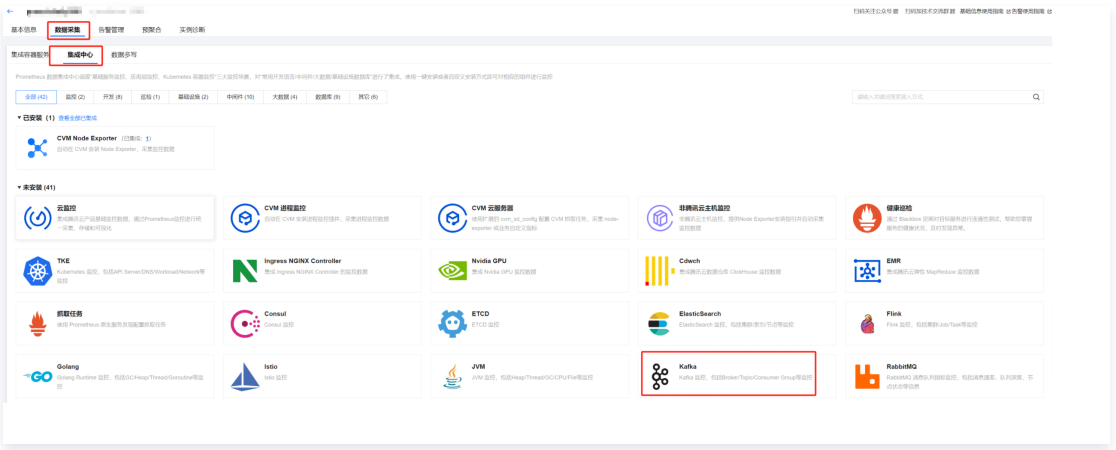
| 服务类型 | 服务名称 | 监控项 | 是否支持一键安装 | 接入文档 |
| --- | --- | --- | --- | --- |
| 监控 | 云监控 | 集成腾讯云产品基础监控数据，通过 Prometheus 监控进行统一采集、存储和可视化 | 支持 | 云监控 |
| | PTS | 云压测监控，提供压测任务 RPS、响应时间、错误率、压测节点内存/CPU 等监控 | 不支持 | 使用 Prometheus 观测性能压测指标 |
| 开发 | CVM Node Exporter | 自动在 CVM 安装 Node Exporter，采集监控数据 | 支持 | 批量安装 Node Exporter |
| | CVM 进程监控 | 自动在 CVM 安装进程监控插件，采集进程监控数据 | 支持 | CVM 进程监控 |
| | CVM 云服务器 | 使用扩展的 cvm_sd_config 配置 CVM 抓取任务，采集 node-exporter 或业务自定义指标 | 支持 | 云服务器场景下自定义接入 |
| | 非腾讯云主机监控 | 非腾讯云主机监控，提供 Node Exporter 安装指引并自动采集监控数据 | 支持 | 非腾讯云主机监控 |
| | 抓取任务 | 使用原生 static_config 配置抓取任务 | 支持 | 抓取配置说明 |
| | Golang | 包括 GC/Heap/Thread/Goroutine 等监控 | 不支持 | Golang 应用接入 |
| | JVM | 包括 Heap/Thread/GC/CPU/File 等监控 | 不支持 | JVM 接入 |
| | Spring MVC | 包括 HTTP 接口/异常/JVM 等监控 | 不支持 | Spring Boot 接入 |
| 巡检 | 健康巡检 | 通过 Blackbox 定期对目标服务进行连通性测试，帮助您掌握服务的健康状况，及时发现异常 | 支持 | 健康巡检 |
| 基础设施 | TKE | Kubernetes 监控，包括 API Server/DNS/Workload/Network 等监控 | 不支持 | 集成容器服务 |
| | Nvidia GPU | 集成 Nvidia GPU 监控数据 | 支持 | Nvidia Gpu Exporter 接入 |
| 中间件 | Ingress NGINX Controller | Ingress NGINX Controller 监控 | 支持 | Ingress NGINX Controller Exporter 接入 |
| | Consul | Consul 监控 | 支持 | Consul Exporter 接入 |
| | Etcd | Etcd 监控 | 不支持 | 配置 Prometheus 监控 |
| | Istio | Istio 监控 | 不支持 | - |
| | Kafka | 包括 Broker/Topic/Consumer Group 等监控 | 支持 | Kafka Exporter 接入 |
| | RabbitMQ | RabbitMQ 消息队列指标监控，包括消息速率、队列深度、节点状态等信息 | 支持 | Rabbitmq Exporter 接入 |
| | RocketMQ | RocketMQ 监控，包括 Broker/Producer/Consumer Group 等监控 | 支持 | - |
| | Fluentd | fluentd 性能监控，包括组件状态、缓冲区使用情况和重试次数等 | 不支持 | Fluentd/FluentBit 接入 |

| | | | | |
|---|---|---|---|---|
| | Nginx | Nginx 服务指标监控，包括健康状况、性能和负载情况等 | 支持 | Nginx Exporter 接入 |
| | Kong | Kong 监控 | 不支持 | 使用 Prometheus 监控 |
| 大数据 | Cdwch | 集成腾讯云数据仓库 ClickHouse 监控数据 | 支持 | - |
| | EMR | 集成腾讯云弹性 MapReduce 监控数据 | 支持 | Prometheus 采集 EMR 组件 |
| | ElasticSearch | 包括集群/索引/节点等监控 | 支持 | ElasticSearch Exporter 接入 |
| | Flink | 包括集群/Job/Task 等监控 | 不支持 | Flink 接入 |
| 数据库 | 云数据库 Memcached | Memcached 监控 | 支持 | Memcached Exporter 接入 |
| | 云数据库 MongoDB | 包括文档数/读写性能/网络流量等 | 支持 | MongoDB Exporter 接入 |
| | 云数据库 MySQL | 包括网络/连接数/慢查询等 | 支持 | MySQL Exporter 接入 |
| | MSSQL | Microsoft SQL Server 指标监控，包括 SQL Server 性能和健康状况等 | 支持 | SQL Server Exporter 接入 |
| | 云数据库 PostgreSQL | 包括 CPU/Memory/事务/Lock/读写等监控 | 支持 | PostgreSQL Exporter 接入 |
| | 云数据库 Redis | 包括内存使用率/连接数/命令执行情况等监控 | 支持 | Redis Exporter 接入 |
| | Aerospike | aerospike 数据库指标监控，包括集群的健康状况、性能表现和负载情况 | 支持 | Aerospike Exporter 接入 |
| | Ceph | Ceph 监控，包括集群、OSD、Pools 的状态、性能等 | 支持 | Ceph Exporter 接入 |
| | OracleDB | Oracle 数据库指标监控，包括数据库性能、负载、健康状况等 | 支持 | Oracle DB Exporter 接入 |
| 其它 | Docker | Docker daemon 监控，包括 docker、container、engine 等信息 | 不支持 | Docker 接入 |
| | Pushgateway | Prometheus Pushgateway，用于接收 Prometheus 服务发现无法直接监控的短期任务指标数据 | 支持 | Pushgateway 接入 |
| | Thanos Sidecar | Thanos Sidecar 仅支持查询当前实例功能，不支持写入功能，可以供 Thanos Query 查询实例数据。 | 支持 | - |
| | Apache | Apache HTTP 服务监控，包括请求速率、连接数、响应代码等 | 支持 | Apache Exporter 接入 |
| | Graphite | Graphite 指标转换成 Prometheus 指标 | 支持 | - |
| | 免鉴权代理 | 无鉴权代理，安装后可获得无需 basic auth 鉴权的 prometheus 内网地址 | 支持 | - |

## 一键安装

部分服务支持一键安装 Agent，详情请参见 支持服务列表。

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心选择支持一键安装的服务，下图以 Kafka 为例。




5. 单击 **Kafka** 即会弹出一个安装窗口，填写指标采集名称和地址等信息，并单击**保存**即可。


## 自定义安装

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心选择对应的服务，下图以 **CVM 云服务器**为例。单击它即会弹出一个窗口，单击**安装说明文档**查看接入指引（仅支持部分产品），接入成功后即可实时监控对应的服务。您还可以单击 **Dashboard 操作**，安装或升级该服务的 Grafana Dashboard。


## Metric Relabel 配置示例

在支持 Metric Relabel 配置的集成中，可以添加 Prometheus 原生的 metric_relabel_configs 配置，下面是常用的 metric_relabe_configs 示例：

```
metric_relabel_configs:
- action: labeldrop  # 去掉名为 labelA 的 label。regex是正则表达式，多个正则表达式用 | 分隔
  regex: labelA
- regex: ins-(.*) # 新增一个名为 id 的 label，其值通过名为 instance_id 的 label 的值经过正则处理后得到。例如
instance_id="ins-a"，新得到的 id="a"
  replacement: $1
  source_labels:
  - instance_id
  target_label: id
- target_label: region # 新增一个 region="ap-guangzhou" 的 label
  replacement: ap-guangzhou
- action: drop # 去掉名为 metricA 或 metricB 的指标
  source_labels:
```

```
- __name__
regex: metricA|metricB
```

## 数据多写

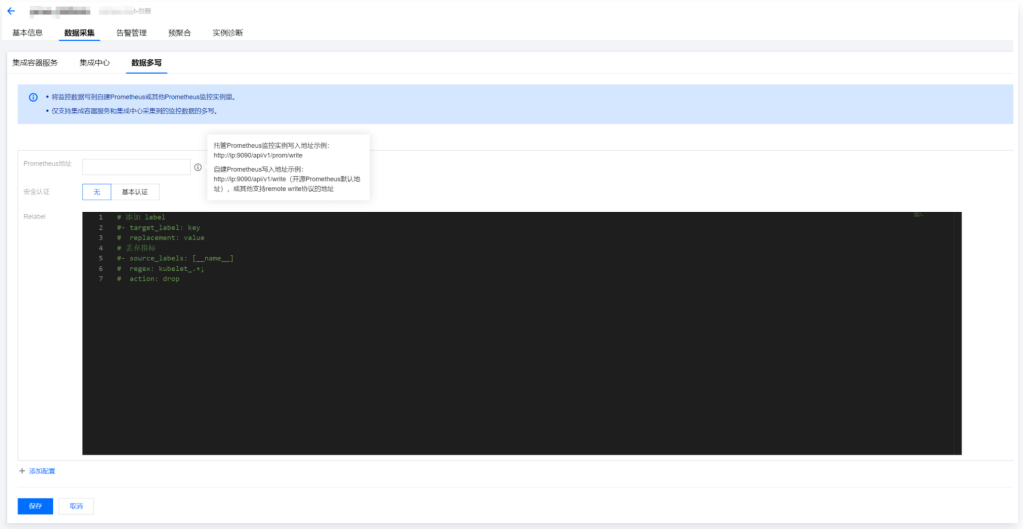
最近更新时间：2024-05-21 17:49:21

数据多写功能支持您将监控数据写到自建 Prometheus 或其他 Prometheus 监控实例中。

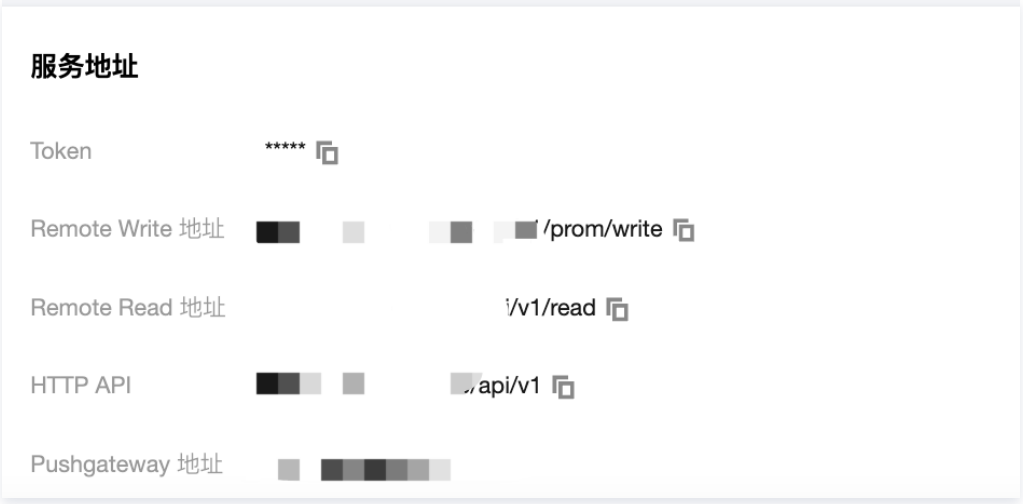
> **说明**
> 目前仅支持集成容器服务和集成中心采集到的监控数据的多写(通过预聚合规则计算得到的指标不支持多写)，其它暂不支持。

## 操作指南

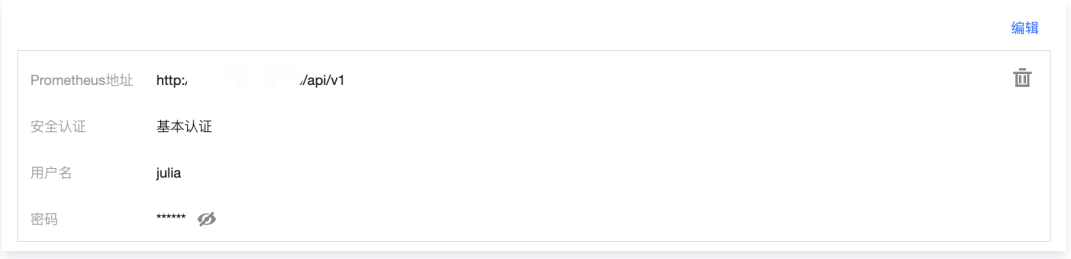
1. 登录 腾讯云可观测平台 。
2. 在左侧导航栏中单击 **Prometheus 监控**。
3. 单击对应的实例名称，进入实例详情页，在顶部导航栏点击**数据采集** > **数据多写**。
4. 在数据多写子窗口单击**添加**，参见下列描述配置数据多写信息。



- Prometheus 地址：在实例基本信息中获取 HTTP API 地址。




- 安全验证：是否开启安全认证，开启后需要自定义用户名和密码。



# 预聚合
# 预聚合概述

最近更新时间：2024-10-31 18:11:52

预聚合 (Recording Rule) 可以让我们对一些常用的指标或者计算相对复杂的指标进行提前计算，然后将这些数据存储到新的数据指标中，查询这些计算好的数据将比查询原始的数据更快更便捷。这对于 Dashboard 场景非常适用，可以解决用户配置以及查询慢的问题。
预聚合以规则组 (Rule Group) 的形式存在，相同组中的规则以一定的间隔顺序执行。聚合规则的名字必须符合 相应的 Prometheus 规范 。
通常一个规则文件如下：

```
groups:
  [ - <rule_group> ]
```

## 规则组

```
# 规则组名称，在同一文件中必须唯一
name: <string>

# 规则探测周期.
[ interval: <duration> | default = global.evaluation_interval ]

rules:
  [ - <rule> ... ]
```

## 规则

预聚合的语法如下：

```
# 生成的新的指标名称，必须是一个有效的指标名称
record: <string>

# PromQL 表达式，每次计算的数据都会存储到新的指标名称 'record' 中
expr: <string>

# 在要存储的数据中所要添加或者覆盖的标签
labels:
  [ <labelname>: <labelvalue> ]
```

## 推荐命名格式

预聚合规则命名的推荐格式：`level:metric:operations` 。

- level：表示聚合级别，以及规则的输出标签。
- metric：是指标名称。
- operations：应用于指标的操作列表。

例如：

```
- record: instance_path:requests:rate5m
  expr: rate(requests_total{job="myjob"}[5m])

- record: path:requests:rate5m
  expr: sum without (instance)(instance_path:requests:rate5m{job="myjob"})
```

# 规则管理

最近更新时间：2024-10-31 18:11:52

## 操作场景

用户可以通过 Prometheus 监控服务控制台来管理 Prometheus 预聚合规则，以解决原生 Prometheus 需要修改配置文件的不便利性。

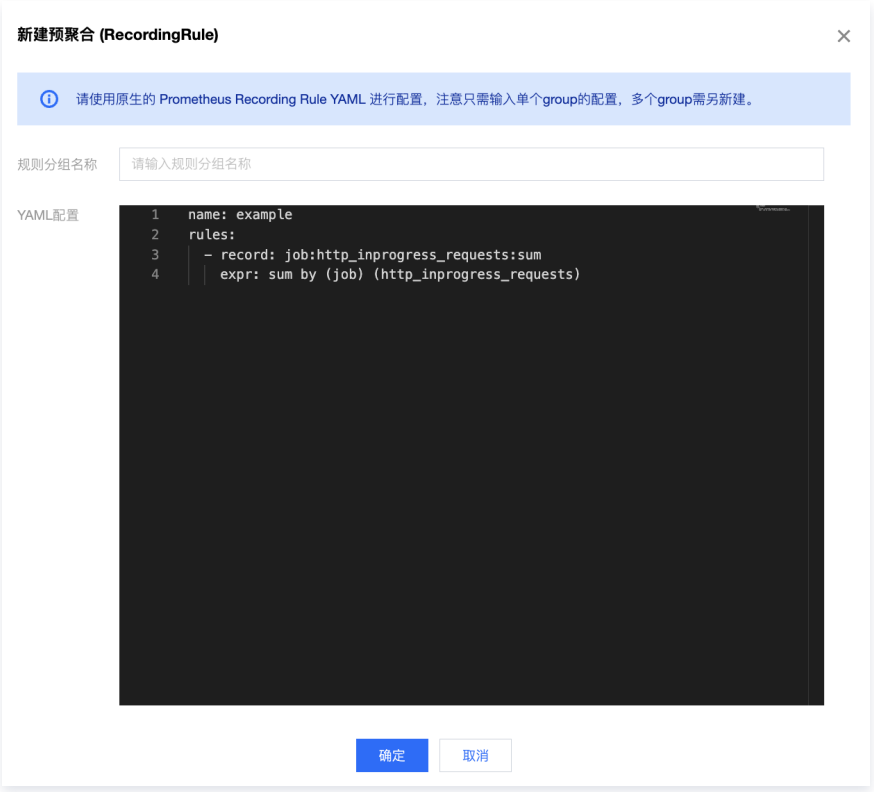
## 准备工作

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 创建 Prometheus 托管实例，详情请参见 创建实例 。
4. 通过实例列表进入到 Prometheus 实例的管理页面。
5. Prometheus 预聚合规则 (Recording Rule)，详情请参见 预聚合概述 。

## 操作步骤

### 新建规则

1. 在实例管理页面，选择需要预聚合操作的实例名称，进入该实例详情页。
2. 在顶部导航栏中单击**预聚合** > **新建预聚合**打开规则新建页面，根据具体实际需求调整规则的表达式及需要聚合生成的新指标名，如下图，具体术语请参见 预聚合概述 。




以下为一个简单预聚合规则例子：

```
name: example
rules:
  - record: job:http_inprogress_requests:sum
    expr: sum by (job) (http_inprogress_requests)
```

3. 单击**确定**即可。

### 管理规则

在规则列表中，可以对相应的规则进行临时的**禁用**或者对**未开启**的规则重新开启。禁用之后，规则将停止工作，相关的预聚合的指标将停止采集。

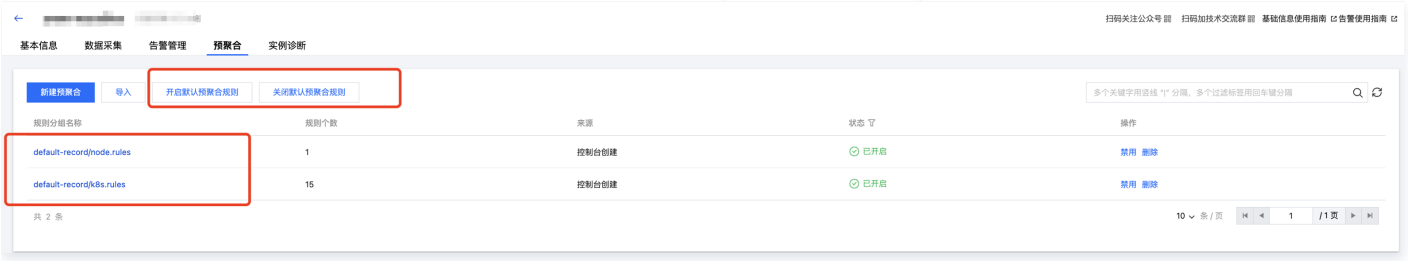
### 删除规则

1. 对一些不再使用的规则，可以进行删除操作。
2. 在列表中选择需要删除的规则，弹框确认之后，规则将会被删除，同时相关的规则将停止工作。

# 默认预聚合规则列表

最近更新时间：2024-08-13 16:10:51

**开启默认预聚合规则的情况下，会自动生成默认预聚合规则指标。默认预聚合规则指标用于常用监控指标的预设 Dashboard 图表展示与告警规则模板，正常计费。默认预聚合规则的启用状态是 Prometheus 实例维度的，可前往 Prometheus 监控 的实例详情页，在预聚合页面修改启用状态。**




**默认预聚合规则指标列表如下：**

| metric（指标名称） | 容器控制台预设 Dashboard | Grafana 平台预设 Dashboard | 告警规则模板 |
| --- | --- | --- | --- |
| :node_memory_MemAvailable_bytes:sum | 集群监控概览-Memory Utilisation | Kubernetes / Compute Resources / Cluster | - |
| node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 集群监控概览-CPU Usage | Kubernetes / Compute Resources / Cluster<br>Kubernetes / Compute Resources / Workload<br>Kubernetes / Compute Resources / Pod<br>Kubernetes / Compute Resources / Node (Pods)<br>Kubernetes / Compute Resources / Namespace (Workloads)<br>Kubernetes / Compute Resources / Namespace (Pods)<br>Kubernetes / Networking / Workload<br>Kubernetes / Networking / Namespace (Workload) | - |
| | 集群监控概览-CPU Quota | | - |
| | 集群 Namespace大盘-CPU Quota | | - |
| | 节点 Pod 监控-CPU Usage | | - |
| | 节点 Pod 监控-CPU Quota | | - |
| | 工作负载监控概览-CPU Usage | | - |
| | 工作负载监控概览-CPU Quota | | - |
| | DaemonSet-CPU Usage | | - |
| | DaemonSet-CPU Quota | | - |
| | 集群 Pod 监控-CPU Quota | | - |
| | StatefulSet-CPU Usage | | - |
| | StatefulSet-CPU Quota | | - |
| | Deployment-CPU Usage | | - |
| | Deployment-CPU Quota | | - |
| node_namespace_pod_container:container_memory_working_set_bytes | 节点 Pod 监控-Memory Usage | | - |
| | 节点 Pod 监控-Memory Quota | | - |
| node_namespace_pod_container:container_memory_rss | - | | - |
| node_namespace_pod_container:container_memory_cache | - | | - |
| node_namespace_pod_container:container_memory_swap | - | | - |

| | | | |
|---|---|---|---|
| namespace_workload_pod:kube_pod_owner:relabel | 集群监控概览-CPU Quota | | - |
| | 集群监控概览-Memory Requests | | - |
| | 工作负载监控概览-CPU Usage | | - |
| | 工作负载监控概览-CPU Quota | | - |
| | 工作负载监控概览-Memory Usage | | - |
| | 工作负载监控概览-Memory Quota | | - |
| | DaemonSet-CPU Usage | | - |
| | DaemonSet-CPU Quota | | - |
| | DaemonSet-Memory Usage | | - |
| | DaemonSet-Memory Quota | | - |
| | 命名空间工作负载网络监控-Current Rate of Bytes Received | | - |
| | 命名空间工作负载网络监控-Current Rate of Bytes Transmitted | | - |
| | 命名空间工作负载网络监控-Current Status | | - |
| | 命名空间工作负载网络监控-Average Rate of Bytes Received | | - |
| | 命名空间工作负载网络监控-Average Rate of Bytes Transmitted | | - |
| | 命名空间工作负载网络监控-Receive Bandwidth | | - |
| | 命名空间工作负载网络监控-Transmit Bandwidth | | - |
| | 命名空间工作负载网络监控-Rate of Received Packets | | - |
| | 命名空间工作负载网络监控-Rate of Transmitted Packets | | - |
| | 命名空间工作负载网络监控-Rate of Received Packets Dropped | | - |
| | 命名空间工作负载网络监控-Rate of Transmitted Packets Dropped | | - |
| | 工作负载网络监控-Current Rate of Bytes Received | | - |
| | 工作负载网络监控-Current Rate of Bytes Transmitted | | - |
| | 工作负载网络监控-Average Rate of Bytes Received | | - |
| | 工作负载网络监控-Average Rate of Bytes Transmitted | | - |
| | 工作负载网络监控-Receive Bandwidth | | - |
| | 工作负载网络监控-Transmit Bandwidth | | - |

| | | | |
|---|---|---|---|
| | 工作负载网络监控-Rate of Received Packets | | - |
| | 工作负载网络监控-Rate of Transmitted Packets | | - |
| | 工作负载网络监控-Rate of Received Packets Dropped | | - |
| | 工作负载网络监控-Rate of Transmitted Packets Dropped | | - |
| | StatefulSet-CPU Usage | | - |
| | StatefulSet-CPU Quota | | - |
| | StatefulSet-Memory Usage | | - |
| | StatefulSet-Memory Quota | | - |
| | Deployment-CPU Usage | | - |
| | Deployment-CPU Quota | | - |
| | Deployment-Memory Usage | | - |
| | Deployment-Memory Quota | | - |
| namespace:kube_pod_container_resource_requests_memory_bytes:sum | - | - | Kubernetes 资源 |
| namespace:kube_pod_container_resource_requests_cpu_cores:sum | - | - | Kubernetes 资源 |

# 实例诊断

最近更新时间：2024-12-05 16:40:32

## 背景

为了提升客户在使用 Prometheus 监控采集端的体验，现提供了新的采集端架构，升级后的采集端新架构支持实例诊断、系统健康检查，并提升了采集 Agent 资源利用率和指标采集稳定性。本文将指引用户通过控制台实例诊断页面将旧采集架构升级到新采集架构，并获取关于当前实例采集和存储的细节信息，以获得更优质的使用体验。

## 操作步骤

### 升级新架构

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 实例列表中，单击**实例 ID/名称** 。
3. 进入 Prometheus 管理中心，在顶部导航栏中单击**实例诊断**，并选择对应的**采集集群**。

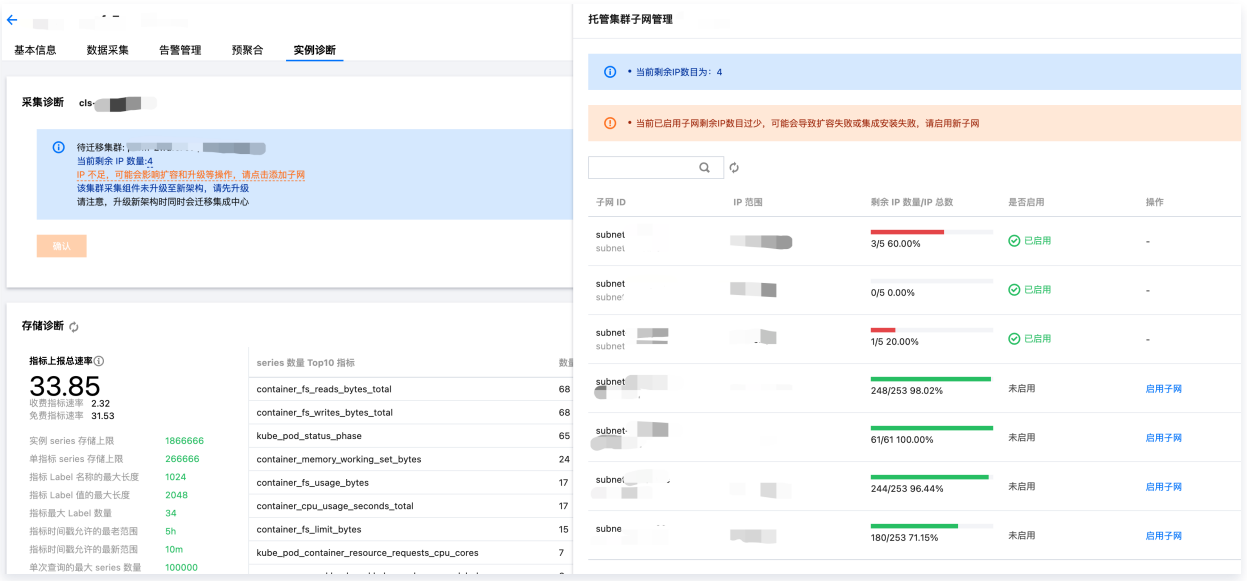
- 点击**确认**即可升级新架构。


> **注意：**
> 升级过程预计需要5分钟，可能存在1-2分钟的指标断点。

- 若实例的托管集群中 IP 数量少于10个会出现风险提示，为防止由于 IP 数量不足问题导致组件无法正常升级，请根据下文指引增加可用 IP 数量。


### 添加子网

1. 在实例诊断页面中点击子网信息，进入**托管集群子网管理**页面。




2. 页面当前 VPC 内已经添加到托管集群内的子网和未添加的子网，可根据规划点击**启用子网**启用剩余 IP 足够的子网。
3. 若没有可用子网，请先 添加子网 后再在当前页面中启用。

## 实例诊断

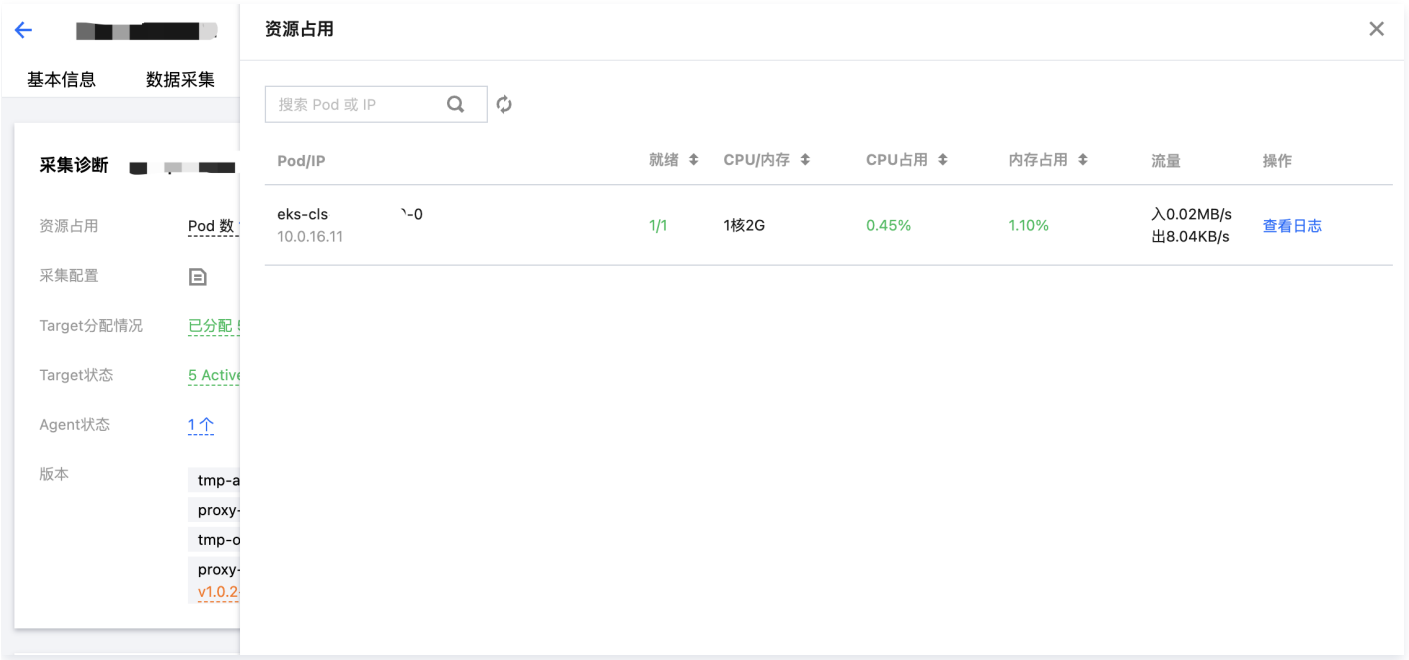
架构升级后，实例诊断页面包括采集和存储两个部分的内容，有助于用户了解 Prometheus 采集和存储的运行情况，从而更加快速地定位存在的问题。

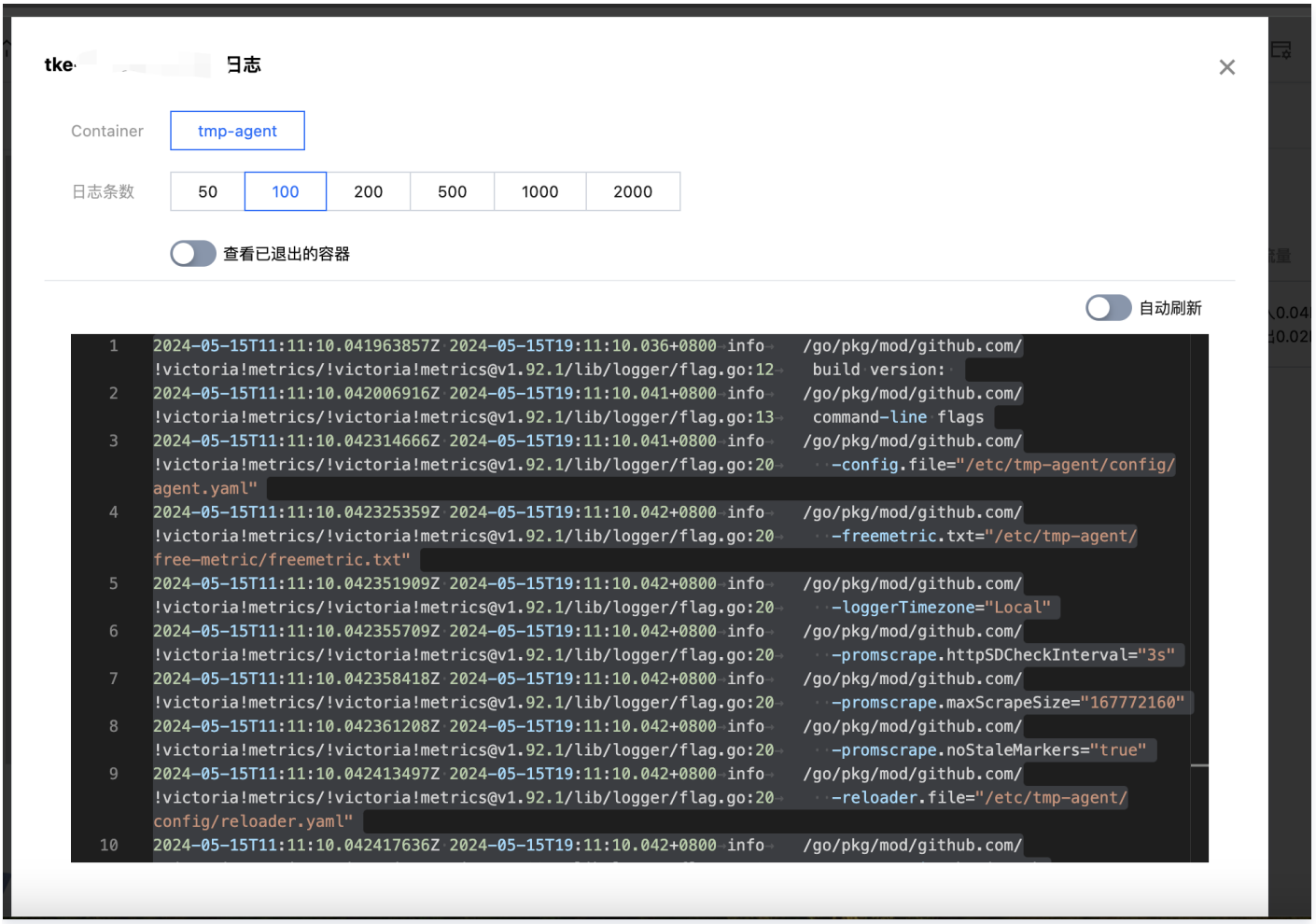
## 采集诊断

如图所示，采集诊断包括对应采集的资源占用、采集配置、Target 分配情况、Target 状态、Agent 状态、组件版本以及采集的架构图。


### 资源占用

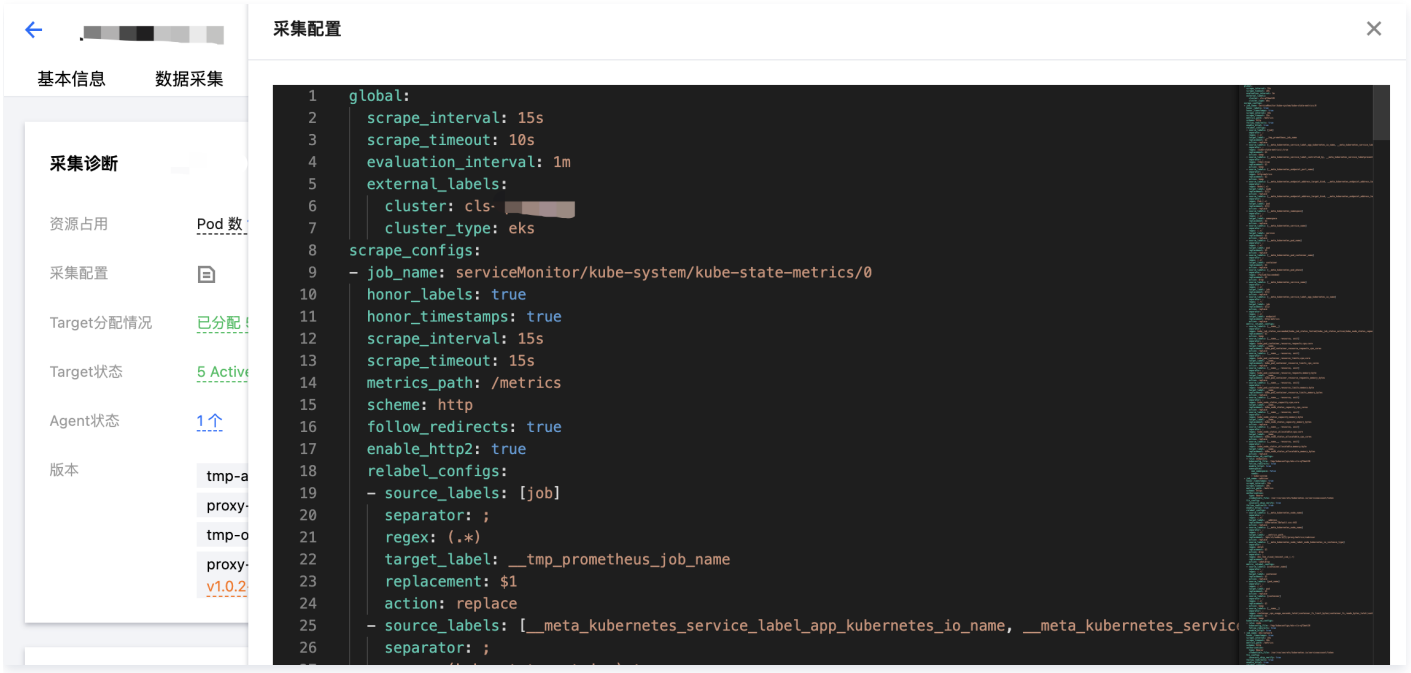
采集分片的资源占用情况，展示了包括 CPU、内存上限和占用以及出入流量等信息；点击**查看日志**可以看到该 Pod 的日志，便于查看运行细节和排查异常问题。



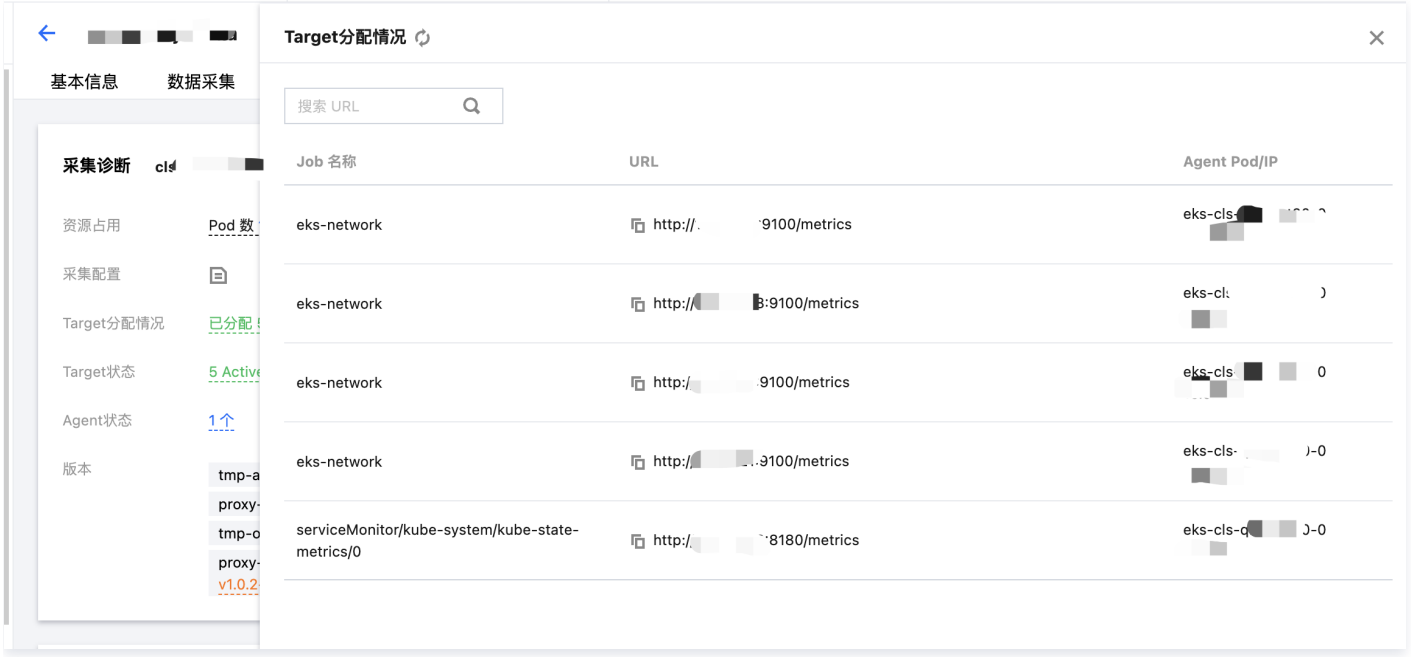

### 采集配置

展示了当前采集的具体采集配置。



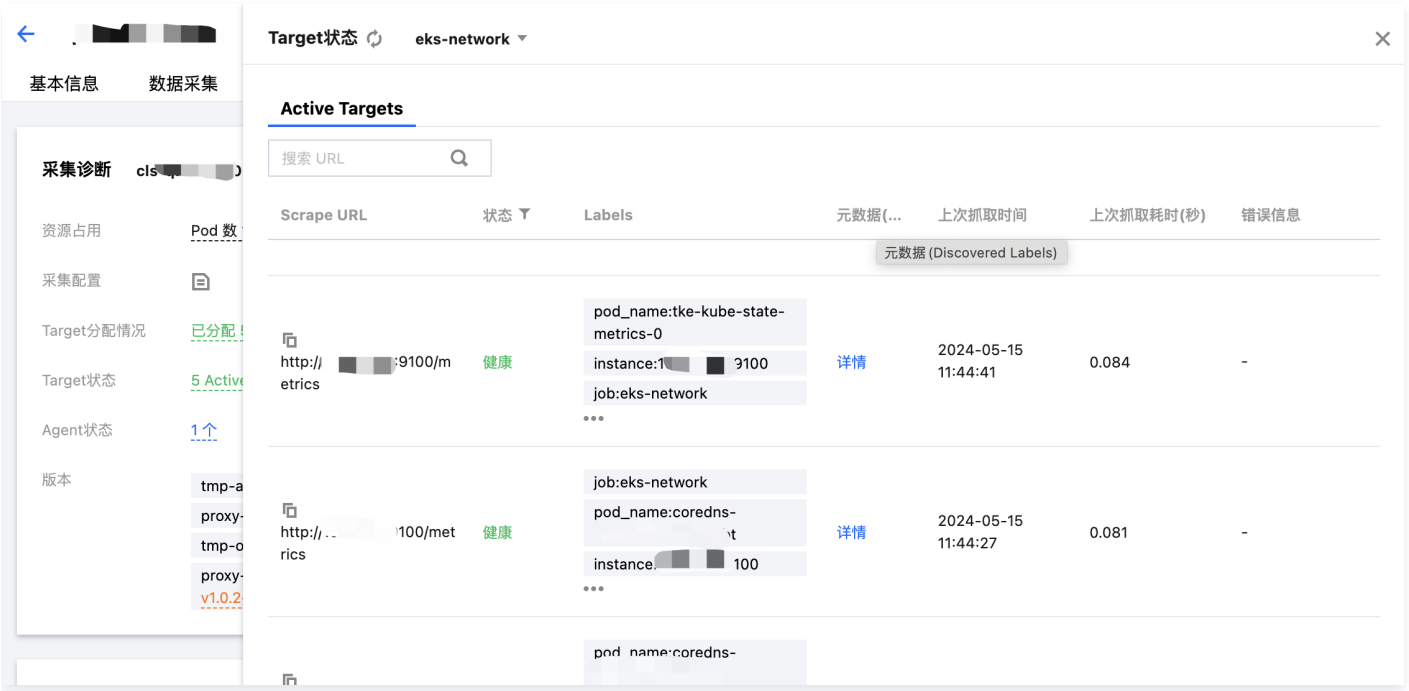

## Target 分配情况

展示了采集目标 Target 的 URL、Target 所属的采集任务 Job 名称，以及目前由哪个采集分片进行采集。



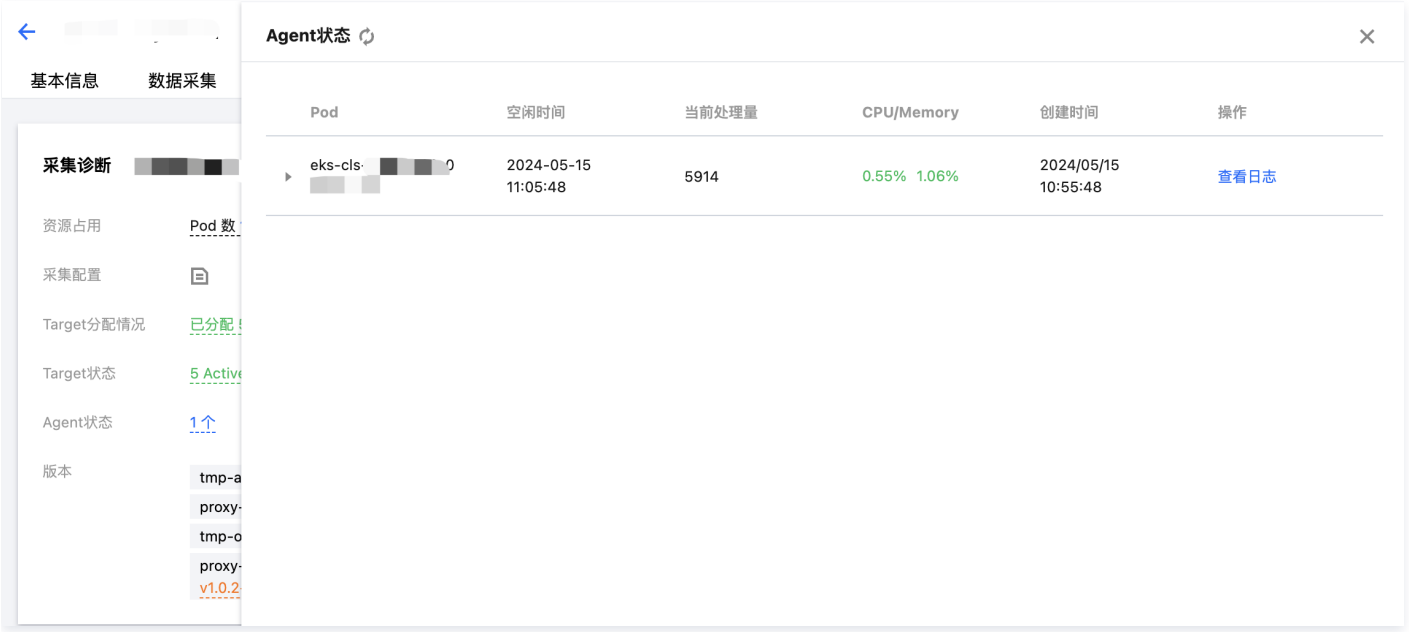

## Target 状态

可根据采集任务 Job 筛选活跃的采集目标 Targets，并获取对应 Target 的状态、Labels、Discovered Labels 等信息。Target 状态在正常情况下是“健康”，若为“异常”且已经有过上次抓取时间，可根据最右边的错误信息进行排查，常见的问题可能为 Target 本身不健康、权限配置错误、网络错误等等。




## Agent 状态

展示了采集分片 Agent 的运行状态；点击**查看日志**可以看到该 Pod 的日志以了解运行细节和排查异常问题。




## 组件版本

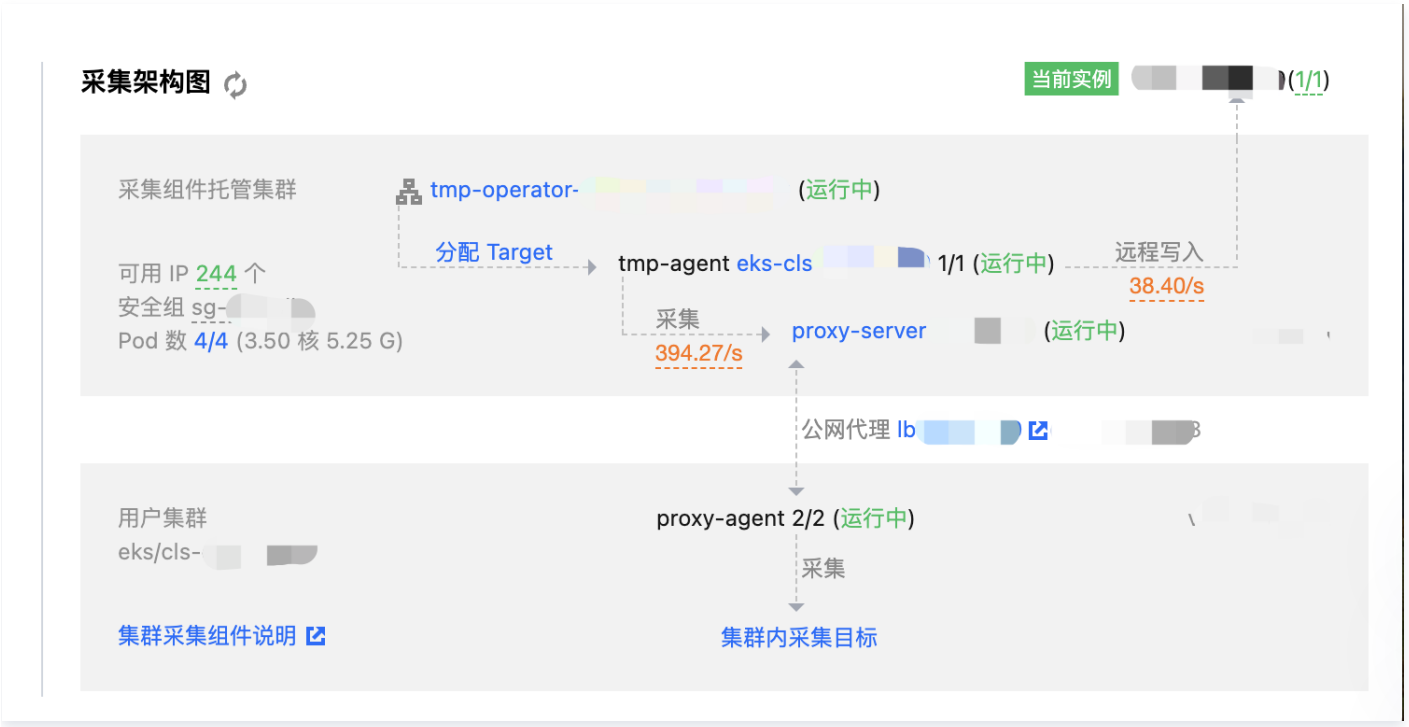
当前采集的组件版本信息，点击进入**组件版本**页面，包括 IP 数量检查和各组件版本的当前版本、最新版本和组件描述、升级描述等信息。


**说明：**

- 采集组件请尽量保持**最新版本**，可通过对应组件的**升级**进行升级；
- 组件 tmp-operator、tmp-agent、proxy-agent 正常情况下能够无影响升级；
- 组件 proxy-server 在升级过程中会出现采集断点，断点时长为 eks 拉起组件 Pod 的时间，并且影响范围是整个 Prometheus 实例的采集（包括集成中心和容器集群），请谨慎操作。

### 采集架构图

通过采集架构图可以了解当前的采集架构信息。




**采集组件托管集群：**

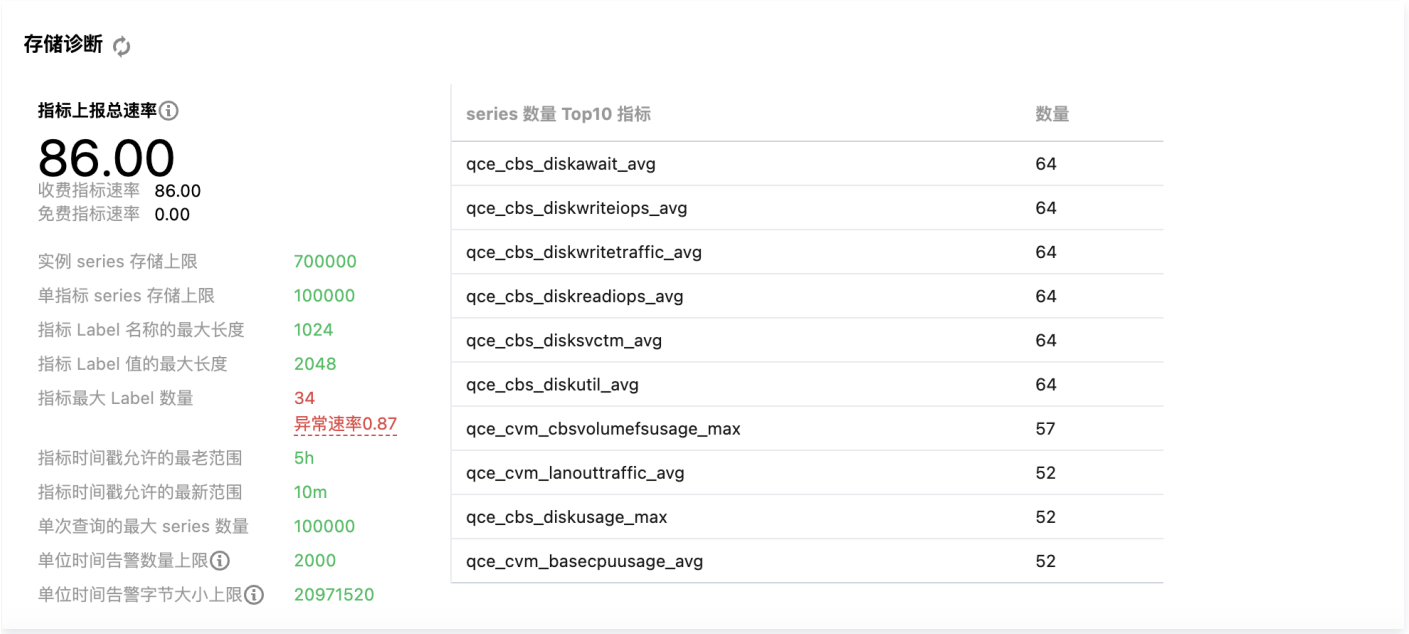
- 可用 IP 数量，当 IP 数量不足时会出现 **IP 数量不足**的提示，点击进入**托管集群子网管理**页面，具体操作可参考前面 添加子网；
- 托管集群的安全组，及其正常运行的放通要求，采集出现的部分网络问题可能是因为安全组未放通；
- 当前采集相关的 Pod 数量及资源使用情况；
- 采集调度组件 tmp-operator 的运行状态；
- 采集目标 Target 分配情况；
- 采集分片组件 tmp-agent 的运行状态；
- 指标采集速率及入带宽；
- 代理组件 proxy-server 的运行状态；
- 指标写入速率及出带宽；
- 写入目标状态，包括当前实例对应的存储和用户配置的数据多写。

**用户集群：**

- 公网代理 CLB 的状态（若开启）；
- 代理组件 proxy-agent 的运行状态；
- 集群内采集目标 Target 的状态。

## 存储诊断

如图所示，存储诊断包括了存储相关的状态和限制，具体限制可参考 相关限制 。




### 参数说明

| 参数 | 说明 |
| --- | --- |
| 指标上报总速率 | 包括免费指标和收费指标。 |
| 实例 series 存储上限 | 活跃的 series 数量超过该值会因为超限丢弃。 |
| 单指标 series 存储上限 | 相同指标名不同 label 是不同的 series，超过该值会因为超限丢弃。 |
| 指标标签 Label 名称的最大长度 | 超过该值会因为不合法而丢弃。 |
| 指标最大 Label 数量 | 超过该值会因为不合法而丢弃。 |
| 指标时间戳允许的最老范围 | 单条 series 中可以接受的最老时间戳（不允许乱序）。 |
| 指标时间戳允许的最新范围 | 单条 series 中可以接受的最新时间戳（不允许乱序）。 |

| | |
|---|---|
| 单次查询的最大 series 数量 | 查询数据时最大涉及的 series 数量，建议缩短范围查询 Range Query 的时间，或在合适的场景使用即时查询 Instant Query。 |
| 单位时间告警数量上限 | 5分钟内触发告警数量上限。 |
| 单位时间告警字节大小上限 | 5分钟内触发告警的标签、Annotation 等字段总的大小上限。 |
| series 数量 Top10 指标 | 同一指标名称，不同标签 Label 的键值都是不同的 series，存储收到单指标 series 存储上限限制，数量过大会产生高基数问题。 |

# 归档存储

最近更新时间：2025-05-16 14:57:42

归档存储是一种数据存储解决方案，专门用于长期保存不常访问的数据，它通常用于存储那些需要保留但不需要频繁访问的信息。为了满足用户对数据的长期保留和业务需求，并在最大程度上为用户减少成本，Prometheus 特推出了归档存储功能，下面将详细介绍该功能的使用方法。

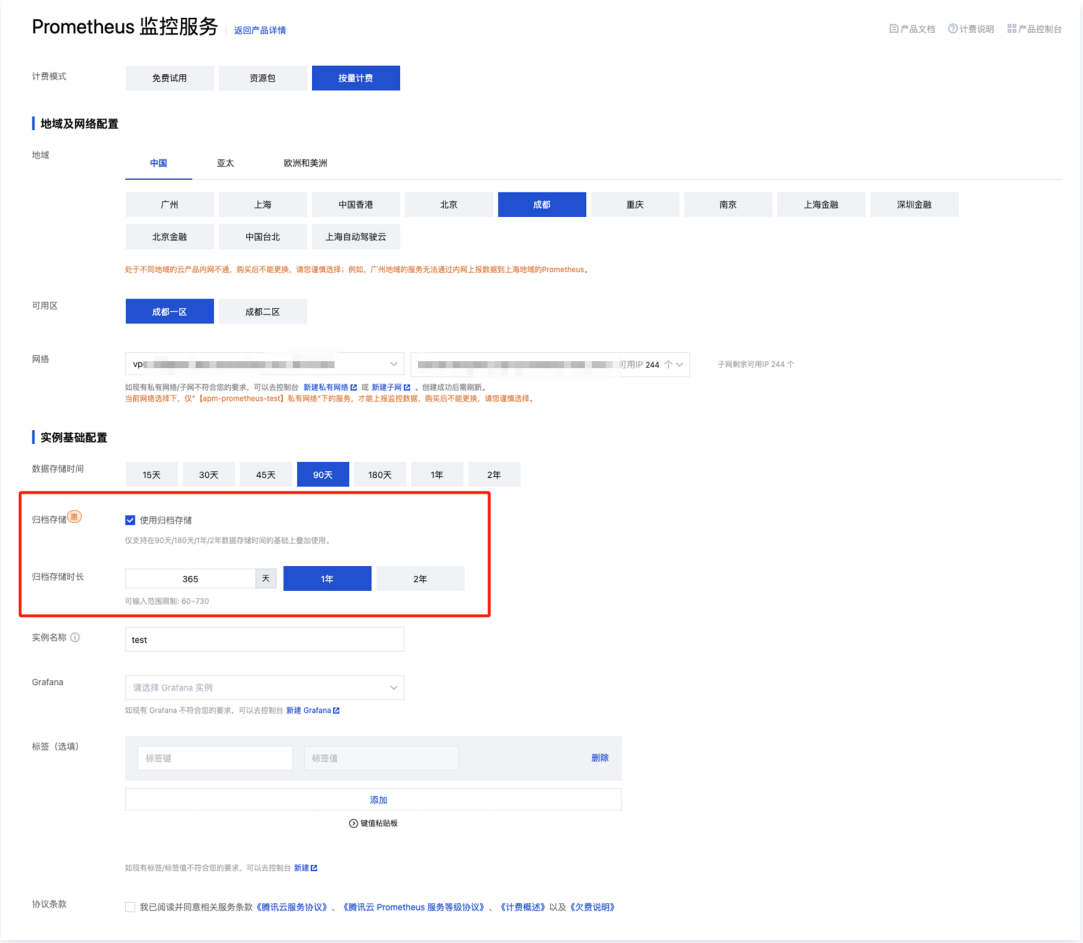
## 使用限制

- **规格限制**：规格为存储90天、180天、1年、2年的实例，才允许额外配置归档存储。
- **存储时间限制**：归档存储的时间范围，支持自定义输入，可配置的最小值为60天，最大值为2年。
- **查询频率限制**：查询的数据范围若包含归档存储数据，系统会限制查询请求频率为每秒最多3次。在无查询的情况下，系统会“冷却”并积累可用额度，最多可累积15次查询配额（约等于5秒的空闲时间），因此，您可以在短时间内一次性发起最多15个查询。（例子：若前5秒无查询请求，第6秒可以一次性发起15个查询；之后将恢复为每秒最多3次的速率。）

## 操作步骤

### 方法一：在实例购买页配置

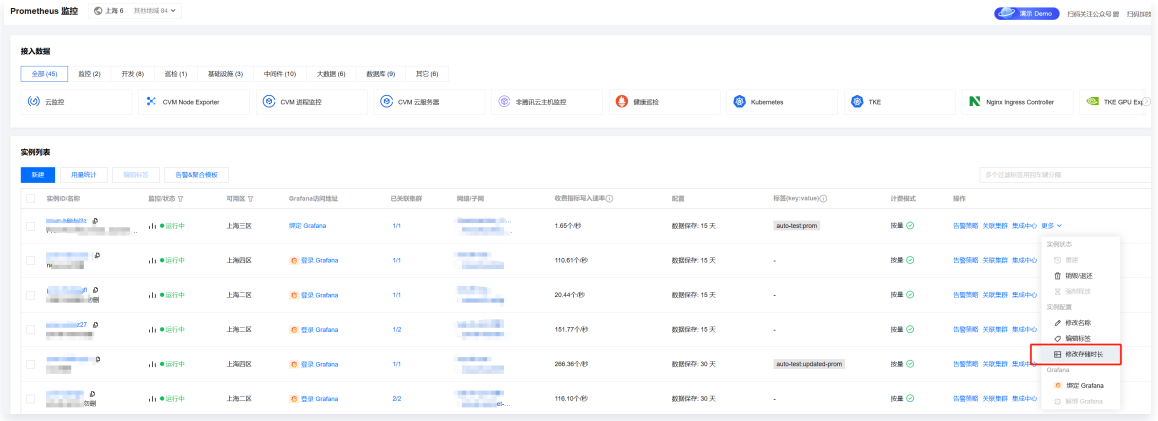
1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在 Prometheus 监控页面单击**新建**，若**数据存储时间**选择了90天、180天、1年、2年，则在该配置项下方有**归档存储**项。



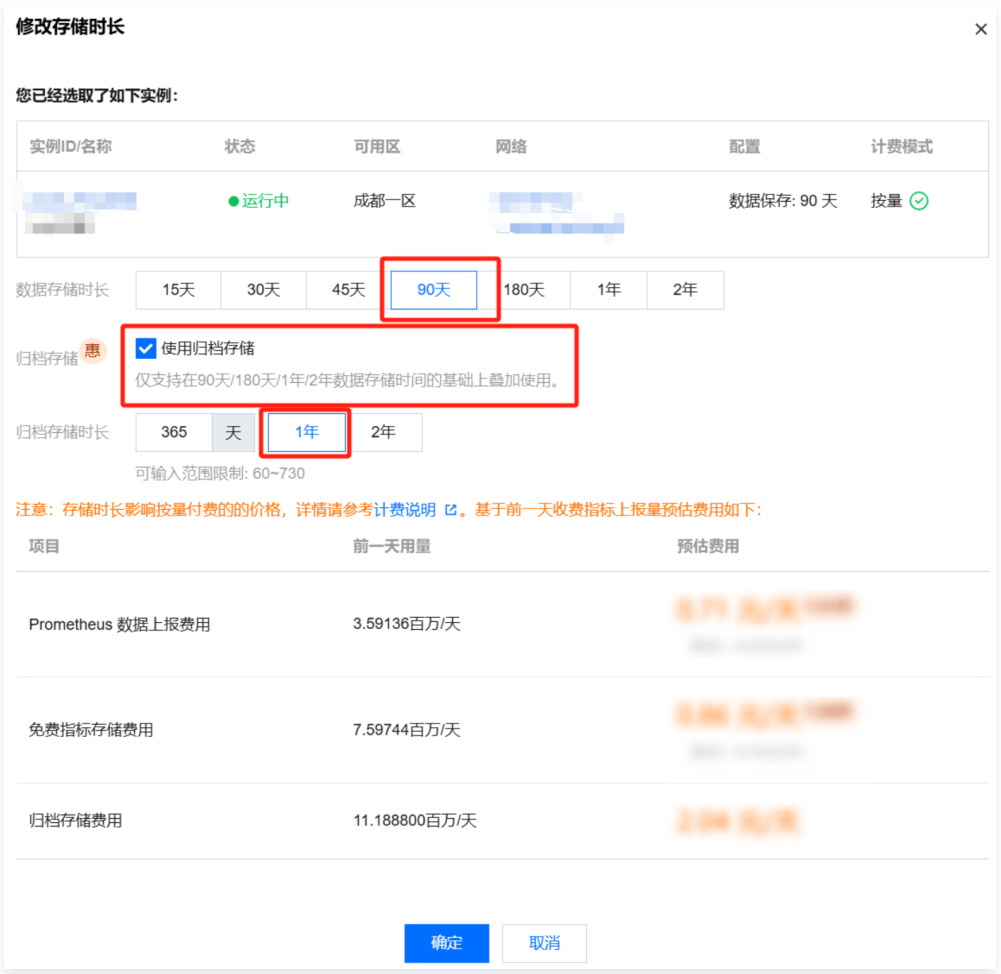

4. 归档存储支持自定义输入存储时长，输入范围限制为60 – 730天。其他配置项按照页面提示填写信息，配置完后，勾选服务条款并单击**立即购买**支付即可。

### 方法二：在实例列表页配置

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 在实例列表中，选择需要修改存储时长的 Prometheus 实例，单击右侧的**更多** > **修改存储时长**。

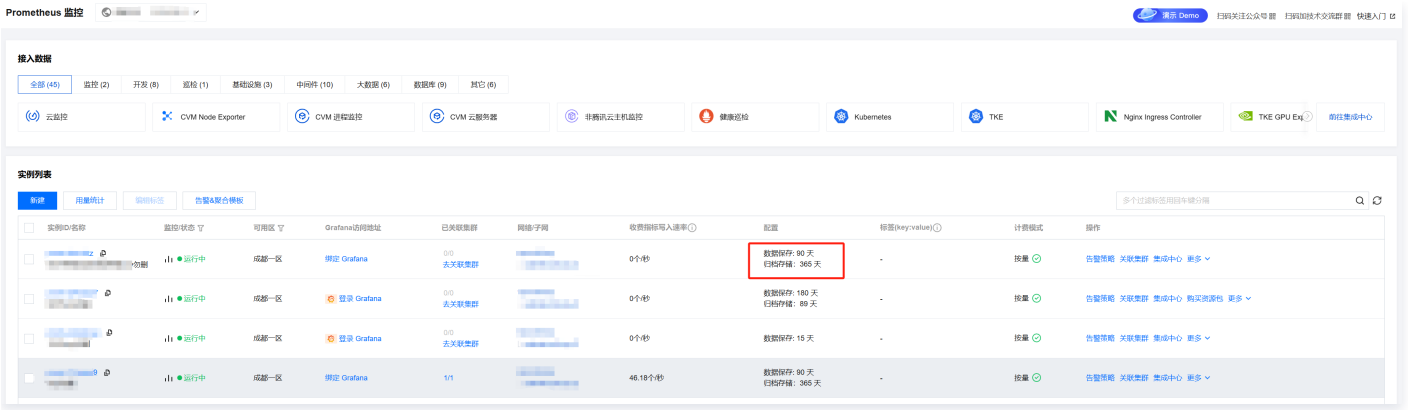



4. 当数据存储时长修改为90天、180天、1年、2年时，可勾选下方的**归档存储**，并自定义输入归档存储时长，修改完单击**确定**即可，如下图所示。




## 配置结果展示

按照上面任意一个方法配置成功后，返回实例列表页可以看到配置下方增加了一个归档存储天数展示，如下图所示：




## 附加说明

**正常存储与增加归档存储的对比如下：**

| 存储方式 | 价格 | 查询频率限制 | 参考文档 |
| --- | --- | --- | --- |
| 数据存储 | 高 | 无 | 按量付费介绍 |
| 数据存储+归档存储 | 低 | 有 | 归档存储付费介绍 |

# 告警策略
# 告警策略概述

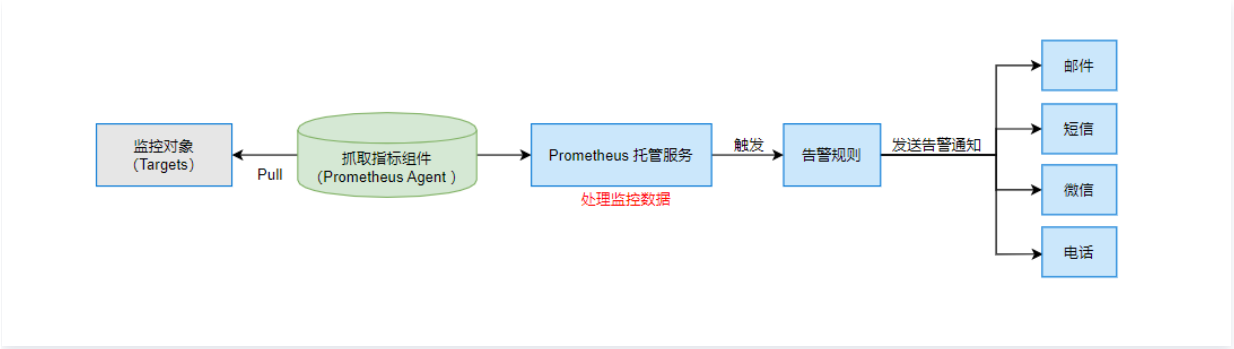
最近更新时间：2024-07-05 11:48:11

Prometheus 监控服务支持您基于 Prometheus 的表达式设定告警条件。在指标达到告警条件时，通过邮件、微信、短信、电话告警渠道通知您采取措施。

> **说明：**
> Prometheus 监控服务告警结合了腾讯云可观测平台告警能力和 Prometheus 开源生态告警能力，使告警更精准，更合理。

## 特性

- 支持 Alertmanager 的收敛、静默等特性，使告警合理。
- 支持指标数据进行聚合和告警规则进行周期性检测、计算，使告警更快速、更便捷。
- 告警规则支持 PromQL 进行定义，使告警更加灵活。
- 使用腾讯云可观测平台告警通知模板，支持邮件、短信、电话多种接收方式。




## 组成部分

| 术语 | 说明 |
| --- | --- |
| 策略名称 | 用户对告警策略命名。 |
| 告警规则 | 包含告警触发条件和持续时间，告警规则要由 PromQL 进行定义。 |
| 告警对象 | 自定义告警标题。 |
| 告警消息 | 自定义告警内容。 |
| 标签 | 用户指定要附加到告警上的一组附加标签。 |
| 注释 | 自定义告警附加消息。 |
| 通知模板 | 包含模板名称、通知类型、接收对象接收渠道，用户自定义接收方式和收敛方式。 |

# 告警规则说明

最近更新时间：2024-08-16 11:52:01

告警规则允许我们基于 Prometheus 的表达式设定告警条件，实时监控服务的状态，及时通知触达服务异常情况。

## 如何定义一个告警规则

在 Prometheus 中，告警规则和聚合规则的定义非常类似，一个告警规则的示例可能如下：

```
groups:
- name: example
  rules:
  - alert: HighRequestLatency
    expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5
    for: 10m
    labels:
      severity: page
    annotations:
      summary: High request latency
```

在告警规则文件中，我们可以将一组相关的规则设置定义在一个 group 下。在每一个 group 中我们可以定义多个告警规则 rule。一条告警规则主要由以下几部分组成：

- alert：告警规则的名称。
- expr：基于 PromQL 的表达式告警触发条件，用于计算是否有时间序列满足该条件。
- for：评估等待时间，可选参数。用于表示只有当触发条件持续一段时间后才发送告警。在等待期间新产生告警的状态为 pending。
- labels：自定义标签，允许用户指定要附加到告警上的一组附加标签。
- annotations：用于指定一组附加信息，例如用于描述告警详细信息的文字等，annotations 的内容在告警产生时会一同作为参数发送到 Alertmanager。

## 模板

通常情况，在告警规则文件的 annotations 中使用 summary 描述告警的概要信息，description 用于描述告警的详细信息。同时 Alertmanager 的 UI 也会根据这两个标签值，显示告警信息。为使告警信息具有更好的可读性，Prometheus 支持模板化 label 和 annotations 的中标签的值。
通过 `$labels.<labelname>` 变量可以访问当前告警实例中指定标签的值。$value 则可以获取当前 PromQL 表达式计算的样本值。

```
# To insert a firing element's label values:
{{ $labels.<labelname> }}
# To insert the numeric expression value of the firing element:
{{ $value }}
```

例如，可以通过模板化优化 summary 以及 description 的内容的可读性：

```
groups:
- name: example
  rules:

  # Alert for any instance that is unreachable for >5 minutes.
  - alert: InstanceDown
    expr: up == 0
    for: 5m
    labels:
      severity: page
    annotations:
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5
minutes."
```

```
  # Alert for any instance that has a median request latency >1s.
  - alert: APIHighRequestLatency
    expr: api_http_request_latencies_second{quantile="0.5"} > 1
    for: 10m
    annotations:
      summary: "High request latency on {{ $labels.instance }}"
      description: "{{ $labels.instance }} has a median request latency above 1s (current value: {{ $value
}}s)"
```

# 新建告警策略

最近更新时间：2025-05-30 14:13:32

本文指导您在 Prometheus 监控服务控制台中创建告警策略，在某些指标发生异常时及时通知您采取措施。

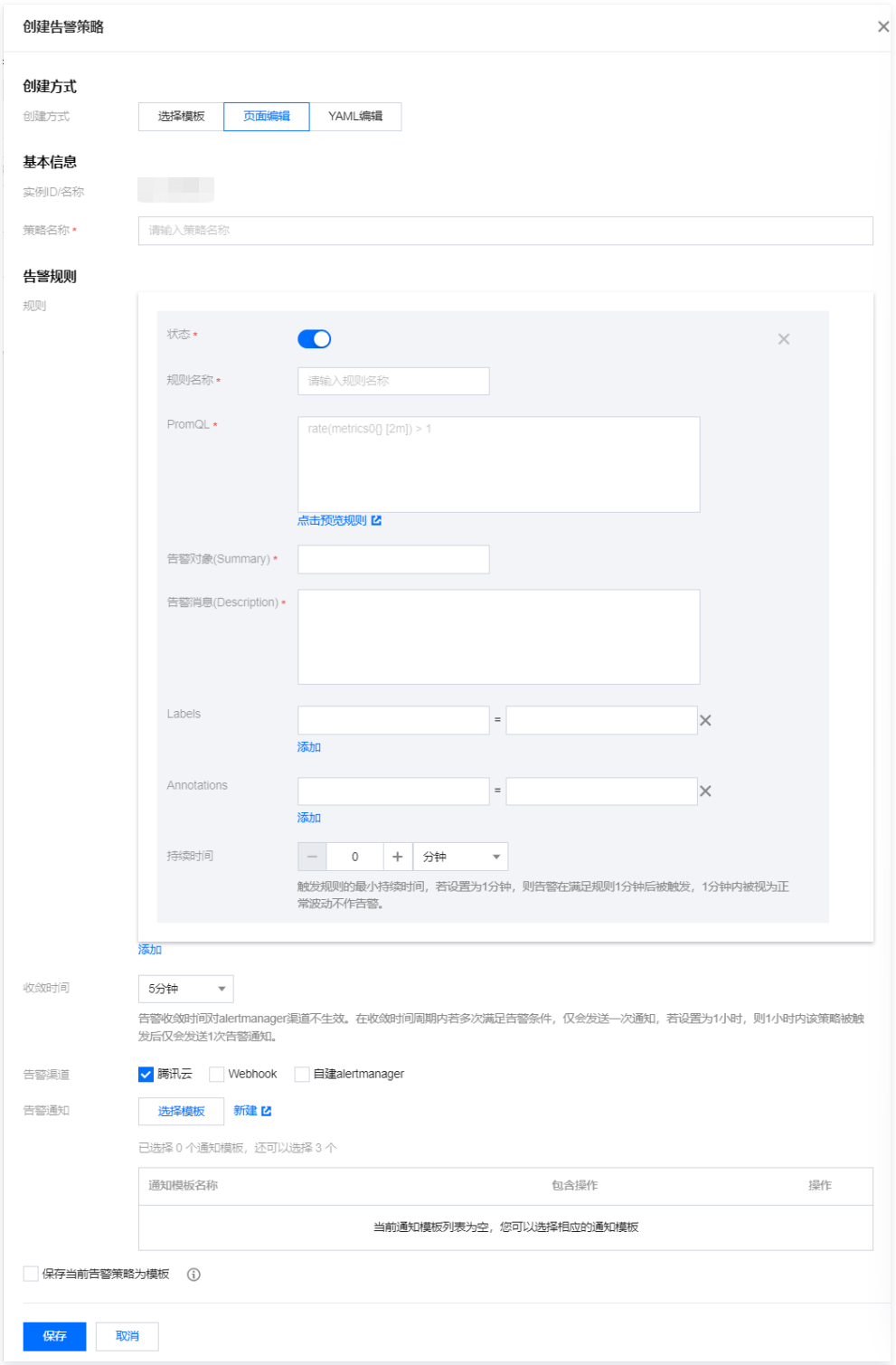
## 前提条件

- 已创建 Prometheus 实例 。
- 已 关联集群 或已安装 集成中心 的数据集成。

## 操作步骤

详细的告警规则说明，请参见 告警规则说明 。

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例，单击顶部菜单栏的**告警管理** ，进入**告警策略**页面。
3. 在告警策略页中单击**新建告警策略**，在弹框中配置告警策略信息。
4. 创建方式：有**选择模板**、**页面编辑**、**YAML 编辑**三种创建方式，下面以**页面编辑**示例。
   - **实例 ID/名称**： 默认显示，无需填写。
   - **策略名称**：自定义名称。
   - **规则**：告警规则组，支持添加多个规则。
     - **规则名称**：自定义名称。
     - **PromQL**：可使用默认模板也可自定义，表示基于 PromQL 的表达式告警触发条件，用于计算是否有时间序列满足该条件。
     - **告警对象(Summary)**：允许用户自定义告警标题，支持通过 {{ $labels.<labelname> }} 变量访问当前告警实例中指定标签的值，详情请参见 模板说明 。
     - **告警消息(Description)**：可使用默认模板也可自定义，允许用户自定义告警内容。支持通过 $labels.<labelname> 变量访问当前告警实例中指定标签的值，{{ $value }} 则可以获取当前 PromQL 表达式计算的样本值，详情请参见 模板说明 。
     - **Labels**：标签，可使用默认模板也可自定义，表示允许用户指定要附加到告警上的一组附加标签，可根据接收到告警的标签匹配相应的处理方式。
     - **Annotations**：注释，可使用默认模板也可自定义，表示允许用户定义告警附加消息。
     - **持续时间**：可使用默认模板也可自定义，表示当触发条件持续多少时间后才发送告警。
   - **收敛时间**：默认每5分钟告警一次，您也可以自定义告警通知频率。
   - **告警渠道**：可选择腾讯云告警渠道、webhook 告警渠道或自建 alertmanager 告警渠道。
   - **告警通知**：支持自定义告警通知模板，包含模板名称、通知类型、接收对象接收渠道等，详情请参见 通知模板 。
   - **保存当前告警策略为模板**：支持将当前页面配置的告警策略内容保存为模板，后续可引用该模板快速创建告警。




## 模板说明

Prometheus 支持模板化 label 和 annotations 中标签的值，通过 {{ $labels.<labelname> }} 变量可以访问当前告警实例中指定标签的值，其中 labelname 为指定标签的名称。{{ $value }} 则可以获取当前 PromQL 表达式计算的样本值，可通过模板函数对 {{ $value }} 进行多种格式转换，以提升数值的可读性，以下是常见的转换方式及示例。

- 基本数值格式化

```
{{ $value }}               // 原始值（如 123.456）
{{ printf "%.2f" $value }} // 保留两位小数（如 123.46）
{{ printf "%.0f" $value }} // 取整（如 123）
{{ printf "%.1e" $value }} // 科学计数法（如 1.2e+02）
```

```
{{ printf "%.2f%%" $value }} // 转换为百分比（如 123.46%）
{{ $value | printf "%.2f%%" }} // 等价写法
```

- 数据单位转换

```
// 字节单位（B, KB, MB, GB 等）
{{ $value | humanize1024 }} // 自动转换字节单位（如 1048576 → "1.0 MB"）
{{ $value | humanize1024 | printf "%.2f" }} // 保留两位小数（如 "1.00 MB"）

// 十进制单位（k, M, G 等）
{{ $value | humanize }} // 自动转换十进制单位（如 1000000 → "1.0 M"）

// 时间单位（s, ms, µs 等）
{{ $value | humanizeDuration }} // 自动转换时间单位（如 0.001 → "1 ms"）

// 百分比格式化转换
{{ $value | humanizePercentage }} // 自动转换为百分比格式（如 0.12345 → "12.3%"）
```

例如，某比率类指标取值范围为 [0,1]，在告警内容中可使用 {{ $value | humanizePercentage }} 转换为百分比格式。如果某比率类指标取值范围为 [0,100]，在告警内容中可使用 {{ printf "%.2f" $value }} 转换为保留两位小数的百分比格式。与此同时，如果指标包含一个名为 instance 的标签表示实例的 IP，在告警内容和告警对象中可使用 {{ $labels.instance }} 来获取标签的值。对应控制台中告警规则的配置如下：


收到的告警通知中，告警对象将附带触发告警的实例 IP，告警消息中将附带触发告警的实例 IP 和当前样本值，具体信息如下：

| 告警状态 State | 告警对象 Summary | 告警消息 Description |
| --- | --- | --- |
| Active | 实例10.0.0.[redacted] CPU使用率过高 | 主机 CPU 使用率过高。实例ID: 10.0.0.[redacted]，当前值: 1.228%。 |
| Active | 实例10.0.0.[redacted] CPU使用率过高 | 主机 CPU 使用率过高。实例ID: 10.0.0.[redacted]，当前值: 4.633%。 |

# 暂停告警策略

最近更新时间：2024-10-31 18:11:52

本文指导您在 Prometheus 监控服务控制台中，如何暂停目标实例下的告警策略。

## 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例，单击顶部导航栏的**告警管理**，进入**告警策略**页面。
3. 在告警策略页面，找到需要关闭的告警策略，在操作列表中单击**全部暂停**。
4. 在弹框中单击**确定**即可。

# 策略类型说明（旧）

最近更新时间：2024-04-26 14:33:11

该策略说明仅适用于旧的包年包月类型的 Prometheus 实例。按量付费类型和资源包类型的 Prometheus 实例，请参考控制台的模板管理中的默认模板内容。
Prometheus 监控服务为 Kubernetes 集群预设了 Master 组件、Kubelet、资源使用、工作负载 和 节点 报警模板。

## Kubernetes Master 组件

非托管集群提供如下指标：

| 策略名称 | 策略表达式 | 持续时间 | 策略描述 |
| --- | --- | --- | --- |
| 客户端访问 APIServer 出错 | (sum(rate(rest_client_requests_total{code=~"5.."}[5m])) by (cluster,instance, job) /sum(rate(rest_client_requests_total[5m])) by (cluster,instance, job))> 0.01 | 15m | 客户端访问 APIServer 出错率大于1% |
| 客户端访问 APIServer 证书快过期 | apiserver_client_certificate_expiration_seconds_count{job="apiserver"} > 0 and on(job) histogram_quantile(0.01, sum by (cluster, job, le) (rate(apiserver_client_certificate_expiration_seconds_bucket{job="apiserver"}[5m]))) < 86400 | 无 | 访问 APIServer 的客户端证书将在24小时后过期 |
| 聚合 API 出错 | sum by(cluster, name, namespace) (increase(aggregator_unavailable_apiservice_count[5m])) > 2 | 无 | 聚合 API 最近5分钟报错 |
| 聚合 API 可用性低 | (1 - max by(name, namespace, cluster) (avg_over_time(aggregator_unavailable_apiservice[5m]))) * 100 < 90 | 5m | 聚合 API 服务最近5分钟可用性低于90% |
| APIServer 故障 | absent(sum(up{job="apiserver"}) by (cluster) > 0) | 5m | APIServer 从采集目标中消失 |
| Scheduler 故障 | absent(sum(up{job="kube-scheduler"}) by (cluster) > 0) | 15m | Scheduler 从采集目标中消失 |
| Controller Manager 故障 | absent(sum(up{job="kube-controller-manager"}) by (cluster) > 0) | 15m | Controller Manager 从采集目标中消失 |
| Kubernetes 核心组件版本不一致 | count by (cluster) (count by (cluster, gitVersion) (label_replace(kubernetes_build_info{job!~"kube-dns\|coredns"},"gitVersion","$1","gitVersion","(v[0-9]*.[0-9]*).*"))) > 1 | 15m | Kubernetes 核心组件运行了不同的版本 |

## Kubelet

| 策略名称 | 策略表达式 | 持续时间 | 策略描述 |
| --- | --- | --- | --- |
| Node 状态异常 | kube_node_status_condition{job=~".*kube-state-metrics",condition="Ready",status="true"} == 0 | 15m | Node 状态异常持续15m |
| Node 不可达 | kube_node_spec_taint{job=~".*kube-state-metrics",key="node.kubernetes.io/unreachable",effect="NoSchedule"} == 1 | 15m | Node 不可达，上面的工作负载会重新调度 |
| Node 上运行太多 pod | count by(cluster, node) ((kube_pod_status_phase{job=~".*kube-state-metrics",phase="Running"} == 1) * on(instance,pod,namespace,cluster) group_left(node) topk by(instance,pod,namespace,cluster) (1, kube_pod_info{job="prometheus-kube-state-metrics"})) /max by(cluster, node) (kube_node_status_capacity_pods{job=~".*kube-state-metrics"} != 1) > 0.95/td> | 15m | Node 上运行 pod 量快达到上限 |
| Node 状态抖动 | sum(changes(kube_node_status_condition{status="true",condition="Ready"}[15m])) by (cluster, node) > 2 | 15m | Node 状态在正常和异常之间抖 |

| | | | |
|---|---|---|---|
| | | | 动 |
| Kubelet 的客户端证书快过期 | kubelet_certificate_manager_client_ttl_seconds < 86400 | 无 | Kubelet 客户端证书将在24小时后过期 |
| Kubelet 的服务端证书快过期 | kubelet_certificate_manager_server_ttl_seconds < 86400 | 无 | Kubelet 服务端证书将在24小时后过期 |
| Kubelet 客户端证书续签出错 | increase(kubelet_certificate_manager_client_expiration_renew_errors[5m]) > 0 | 15m | Kubelet 续签客户端证书出错 |
| Kubelet 服务端证书续签出错 | increase(kubelet_server_expiration_renew_errors[5m]) > 0 | 15m | Kubelet 续签服务端证书出错 |
| PLEG 耗时高 | histogram_quantile(0.99, sum(rate(kubelet_pleg_relist_duration_seconds_bucket[5m])) by (cluster, instance, le) * on(instance, cluster) group_left(node) kubelet_node_name{job="kubelet"}) >= 10 | 5m | PLEG 操作耗时的99分位数超过10秒 |
| Pod 启动耗时高 | histogram_quantile(0.99, sum(rate(kubelet_pod_worker_duration_seconds_bucket{job="kubelet"}[5m])) by (cluster, instance, le)) * on(cluster, instance) group_left(node) kubelet_node_name{job="kubelet"} > 60 | 15m | Pod 启动耗时的99分位数值超过60秒 |
| Kubelet 故障 | absent(sum(up{job="kubelet"}) by (cluster) > 0) | 15m | Kubelet 从采集目标消失 |

## Kubernetes 资源使用

| 策略名称 | 策略表达式 | 持续时间 | 策略描述 |
|---|---|---|---|
| 集群 CPU 资源过载 | sum by (cluster) ( max by (cluster, namespace, pod, container) ( kube_pod_container_resource_requests_cpu_cores{job=~".*kube-state-metrics"} ) * on(cluster, namespace, pod) group_left() max by (cluster, namespace, pod) ( kube_pod_status_phase{phase=~"Pending\|Running"} == 1 )) /sum by (cluster) (kube_node_status_allocatable_cpu_cores) >(count by (cluster) (kube_node_status_allocatable_cpu_cores)-1) / count by (cluster) (kube_node_status_allocatable_cpu_cores) | 5m | 集群内 Pod 申请的 CPU 总量过多，已无法容忍 Node 挂掉 |
| 集群内存资源过载 | sum by (cluster) ( max by (cluster, namespace, pod, container) ( kube_pod_container_resource_requests_memory_bytes{job=~".*kube-state-metrics"} ) * on(cluster, namespace, pod) group_left() max by (cluster, namespace, pod) ( kube_pod_status_phase{phase=~"Pending\|Running"} == 1 )) /sum by (cluster) (kube_node_status_allocatable_memory_bytes) >(count by (cluster) (kube_node_status_allocatable_memory_bytes)-1) /count by (cluster) (kube_node_status_allocatable_memory_bytes) | 5m | 集群内 Pod 申请的内存总量过多，已无法容忍 Node 挂掉 |
| 集群 CPU 配额过载 | sum by (cluster) (kube_resourcequota{job=~".*kube-state-metrics", type="hard", resource="cpu"}) /sum by (cluster) (kube_node_status_allocatable_cpu_cores) > 1.5 | 5m | 集群内 CPU 配额超过可分配 CPU 总量 |
| 集群内存配额过载 | sum by (cluster) (kube_resourcequota{job=~".*kube-state-metrics", type="hard", resource="memory"}) /sum by (cluster) (kube_node_status_allocatable_memory_bytes) > 1.5 | 5m | 集群内内存配额超过可分配内存总量 |
| 配额资源快使用完 | sum by (cluster, namespace, resource) (kube_resourcequota{job=~".*kube-state-metrics", type="used"}) /sum by (cluster, namespace, resource) (kube_resourcequota{job=~".*kube-state-metrics", type="hard"} > 0) >= 0.9 | 15m | 配额资源使用率超过 90% |

| | | | |
|---|---|---|---|
| CPU 执行周期受限占比高 | sum(increase(container_cpu_cfs_throttled_periods_total{container!="", }[5m])) by (cluster, container, pod, namespace) /sum(increase(container_cpu_cfs_periods_total{}[5m])) by (cluster, container, pod, namespace) > ( 25 / 100 ) | 15m | CPU 执行周期受到限制的占比高 |
| Pod 的 CPU 使用率高 | sum(rate(container_cpu_usage_seconds_total{job="kubelet", metrics_path="/metrics/cadvisor", image!="", container!="POD"}[5m])) by (cluster, namespace, pod, container) / sum(kube_pod_container_resource_limits_cpu_cores) by (cluster, namespace, pod, container) > 0.75 | 15m | Pod 的 CPU 使用率超过75% |
| Pod 的内存使用率高 | sum(container_memory_working_set_bytes{job="kubelet", metrics_path="/metrics/cadvisor", image!="", container!="POD"}) by (cluster, namespace, pod, container) /sum(kube_pod_container_resource_limits_memory_bytes) by (cluster, namespace, pod, container) > 0.75 | 15m | Pod 的内存使用率超过 75% |

## Kubernetes 工作负载

| 策略名称 | 策略表达式 | 持续时间 | 策略描述 |
|---|---|---|---|
| Pod 频繁重启 | increase(kube_pod_container_status_restarts_total{job=~".*kube-state-metrics"}[5m]) > 0 | 15m | Pod 最近5m频繁重启 |
| Pod 状态异常 | sum by (namespace, pod, cluster) ( max by(namespace, pod, cluster) ( kube_pod_status_phase{job=~".*kube-state-metrics", phase=~"Pending\|Unknown"} ) * on(namespace, pod, cluster) group_left(owner_kind) topk by(namespace, pod) ( 1, max by(namespace, pod, owner_kind, cluster) (kube_pod_owner{owner_kind!="Job"}) )) > 0 | 15m | Pod处于NotReady状态超过15分钟 |
| 容器状态异常 | sum by (namespace, pod, container, cluster) (kube_pod_container_status_waiting_reason{job=~".*kube-state-metrics"}) > 0 | 1h | 容器长时间处于 Waiting 状态 |
| Deployment 部署版本不匹配 | kube_deployment_status_observed_generation{job=~".*kube-state-metrics"} !=kube_deployment_metadata_generation{job=~".*kube-state-metrics"} | 15m | 部署版本和设置版本不一致，表示deployment 变更没有生效 |
| Deployment 副本数不匹配 | ( kube_deployment_spec_replicas{job=~".*kube-state-metrics"} != kube_deployment_status_replicas_available{job=~".*kube-state-metrics"}) and ( changes(kube_deployment_status_replicas_updated{job=~".*kube-state-metrics"}[5m]) == 0) | 15m | 实际副本数和设置副本数不一致 |
| Statefulset 部署版本不匹配 | ( kube_statefulset_status_replicas_ready{job=~".*kube-state-metrics"} != kube_statefulset_status_replicas{job=~".*kube-state-metrics"}) and ( changes(kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"}[5m]) == 0) | 15m | 部署版本和设置版本不一致，表示statefulset 变更没有生效 |
| Statefulset 副本数不匹配 | kube_statefulset_status_observed_generation{job=~".*kube-state-metrics"} !=kube_statefulset_metadata_generation{job=~".*kube-state-metrics"} | 15m | 实际副本数和设置副本数不一致 |
| Statefulset 更新未生效 | ( max without (revision) ( kube_statefulset_status_current_revision{job=~".*kube-state-metrics"}unless kube_statefulset_status_update_revision{job=~".*kube-state-metrics"} ) * ( kube_statefulset_replicas{job=~".*kube-state-metrics"} != kube_statefulset_status_replicas_updated{job=~".*kube- | 15m | Statefulset 部分 pod 没有更新 |

| | | | |
|---|---|---|---|
| | state-metrics"} )) and ( changes(kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"}[5m]) == 0) | | |
| Daemonset 变更卡住 | ( ( kube_daemonset_status_current_number_scheduled{job=~".*kube-state-metrics"} != kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"} ) or ( kube_daemonset_status_number_misscheduled{job=~".*kube-state-metrics"} != 0 ) or ( kube_daemonset_updated_number_scheduled{job=~".*kube-state-metrics"} != kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"} ) or ( kube_daemonset_status_number_available{job=~".*kube-state-metrics"} != kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"} )) and ( changes(kube_daemonset_updated_number_scheduled{job=~".*kube-state-metrics"}[5m]) == 0) | 15m | Daemonset 变更超过15分钟 |
| Daemonset 部分 node 未调度 | kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"} - kube_daemonset_status_current_number_scheduled{job=~".*kube-state-metrics"} > 0 | 10m | Daemonset 在部分 node 未被调度 |
| Daemonset 部分 node 被错误调度 | kube_daemonset_status_number_misscheduled{job=~".*kube-state-metrics"} > 0 | 15m | Daemonset 被错误调度到一些 node |
| Job 运行太久 | kube_job_spec_completions{job=~".*kube-state-metrics"} - kube_job_status_succeeded{job=~".*kube-state-metrics"} > 0 | 12h | Job 执行时间超过12小时 |
| Job 执行失败 | kube_job_failed{job=~".*kube-state-metrics"} > 0 | 15m | Job 执行失败 |
| 副本数和 HPA 不匹配 | (kube_hpa_status_desired_replicas{job=~".*kube-state-metrics"} !=kube_hpa_status_current_replicas{job=~".*kube-state-metrics"}) andchanges(kube_hpa_status_current_replicas[15m]) == 0 | 15m | 实际副本数和 HPA 设置的不一致 |
| 副本数达到 HPA 最大值 | kube_hpa_status_current_replicas{job=~".*kube-state-metrics"} ==kube_hpa_spec_max_replicas{job=~".*kube-state-metrics"} | 15m | 实际副本数达到 HPA 配置的最大值 |
| PersistentVolume 状态异常 | kube_persistentvolume_status_phase{phase=~"Failed\|Pending",job=~".*kube-state-metrics"} > 0 | 15m | PersistentVolume 处于 Failed 或 Pending状态 |

## Kubernetes 节点

| 策略名称 | 策略表达式 | 持续时间 | 策略描述 |
|---|---|---|---|
| 文件系统空间快耗尽 | ( node_filesystem_avail_bytes{job="node-exporter",fstype!=""} / node_filesystem_size_bytes{job="node-exporter",fstype!=""} * 100 < 3and node_filesystem_readonly{job="node-exporter",fstype!=""} == 0) | 1h | 文件系统空间预计在4小时后使用完 |

| | | | |
|---|---|---|---|
| 文件系统空间使用率高 | ( node_filesystem_avail_bytes{job="node-exporter",fstype!=""} / node_filesystem_size_bytes{job="node-exporter",fstype!=""} * 100 < 15and predict_linear(node_filesystem_avail_bytes{job="node-exporter",fstype!=""}[6h], 4*60*60) < 0and node_filesystem_readonly{job="node-exporter",fstype!=""} == 0) | 1h | 文件系统可用空间低于5% |
| 文件系统 inode 快耗尽 | ( node_filesystem_files_free{job="node-exporter",fstype!=""} / node_filesystem_files{job="node-exporter",fstype!=""} * 100 < 3and node_filesystem_readonly{job="node-exporter",fstype!=""} == 0) | 1h | 文件系统 inode 预计在4小时后使用完 |
| 文件系统 inode 使用率高 | ( node_filesystem_files_free{job="node-exporter",fstype!=""} / node_filesystem_files{job="node-exporter",fstype!=""} * 100 < 20and predict_linear(node_filesystem_files_free{job="node-exporter",fstype!=""}[6h], 4*60*60) < 0and node_filesystem_readonly{job="node-exporter",fstype!=""} == 0) | 1h | 文件系统可用 inode 低于3% |
| 网卡状态不稳定 | changes(node_network_up{job="node-exporter",device!~"veth.+"}[2m]) > 2 | 2m | 网卡状态不稳定，在 up 和 down 间频繁变化 |
| 网卡接收出错 | increase(node_network_receive_errs_total[2m]) > 10 | 1h | 网卡接收数据出错 |
| 网卡发送出错 | increase(node_network_transmit_errs_total[2m]) > 10 | 1h | 网卡发送数据出错 |
| 机器时钟未同步 | min_over_time(node_timex_sync_status[5m]) == 0 | 10m | 机器时间最近未同步，检查 NTP 是否正常配置 |
| 机器时钟漂移 | ( node_timex_offset_seconds > 0.05and deriv(node_timex_offset_seconds[5m]) >= 0)or( node_timex_offset_seconds < -0.05and deriv(node_timex_offset_seconds[5m]) <= 0) | 10m | 机器时间漂移超过300秒，检查 NTP 是否正常配置 |

# 通知模板

最近更新时间：2024-11-06 18:22:52

本文指导您在告警模块中创建通知模板。

## 应用场景

- 多个策略一键复用模板，减少用户重复配置用户通知。
- 个性化配置用户通知方式。例如：白天可以配置告警接收渠道为邮件、短信、微信。夜间可以配置告警接收渠道为电话。

## 前提条件

- 查看通知模板：子账号需拥有 MONITOR 读权限。
- 创建、编辑通知模板：子账号需拥有 MONITOR 写权限。

> **说明：**
> 详情可参见 访问权限 进行子账号授权。

## 相关限制

| 功能 | 限制 |
| --- | --- |
| 用户通知 | 最多可添加五项 |
| 接口回调 | 最多填写三个公网可访问的 URL |

## 操作步骤

### 新建通知模板

1. 登录 腾讯云可观测平台，选择**告警管理 > 告警配置 > 通知模板**。
2. 单击**新建通知模板**，在“新建通知模板”填写信息。

- 模板名称：自定义模板名称。
- 通知类型：
  - 告警触发：告警触发时发送通知。
  - 告警恢复：告警恢复时发送通知。
- 用户通知：
  - 接收对象：可选用户/用户组/值班表，如需创建告警接收组请参见 创建接收组 。
  - 通知周期：周一 – 周日，可自行勾选。
  - 通知时段：定义接收告警时间段。
  - 接收渠道：支持邮箱、短信、微信、电话、企业微信五种告警渠道。您还可以根据不同的用户维度，设置不同的告警接收渠道和通知时段。
    电话告警配置说明：
    - 拨打类型：
      - 同时拨打：支持接收对象下的所有用户同时接收电话告警。
      - 轮询拨打：对接收对象下的用户轮询进行电话告警。
        - 轮询次数：在无有效触达时，对所有接收人逐一轮询拨打的最多次数。
        - 轮询顺序：电话告警按照接收人顺序轮询拨打，上下拖动接收人调整拨打顺序。
        - 轮询间隔：电话告警按照接收人顺序轮询拨打的时间间隔。
    - 触达通知：电话成功接收或轮询结束发送信息给所有接收人。短信需计算额度。
    - 电话确认：开启后用户需要在接通电话后按1确认已接通，否则当前电话告警电话会再次触达。
- 接口回调：填写公网可访问到的url作为回调接口地址，腾讯云可观测平台将及时把告警信息推送到该地址，当 HTTP 返回 200为验证成功。告警回调字段说明请参见 告警回调说明 。

> **说明：**

- 回调地址保存后自动验证一次您的 URL，验证超时时间为5s；当用户创建的告警策略被触发或被恢复均会通过接口回调推送告警消息，此告警消息最多推送三次， 每次请求的超时时间为5s。
- 当用户创建的告警策略被触发或恢复时，均会通过接口回调推送告警消息。接口回调也支持重复告警。
- 告警回调 API 出方向 IP 为动态随机分配，无法将具体的 IP 信息提供给您，但 IP 端口固定为80端口，建议您根据80端口在安全组上配置加全放通策略。


## 默认通知模板

系统自动为您创建默认通知模板，模板内容如下：

| 功能 | 默认配置 |
| --- | --- |
| 模板名称 | 系统预设通知模板 |
| 通知类型 | 告警触发，告警恢复 |
| 告警接收人 | 主账号管理员 |
| 通知时间段 | 00:00:00 – 23:59:59（全天） |
| 接收渠道 | 邮件、短信 |

## 删除模板

> **说明：**
> 默认通知模板不支持删除。

1. 找到需要删除的模板名称，在操作列中单击**删除**。
2. 在弹框中确认删除即可。

## 复制模板

1. 找到需要复制的模板名称，在操作列中单击**复制**。
2. 在跳转页中修改信息或直接单击**完成**即可。

# 查看告警指标图表

最近更新时间：2024-09-04 18:28:21

本文将为您介绍在触发 Prometheus 告警后，如何快速查看告警指标图表。

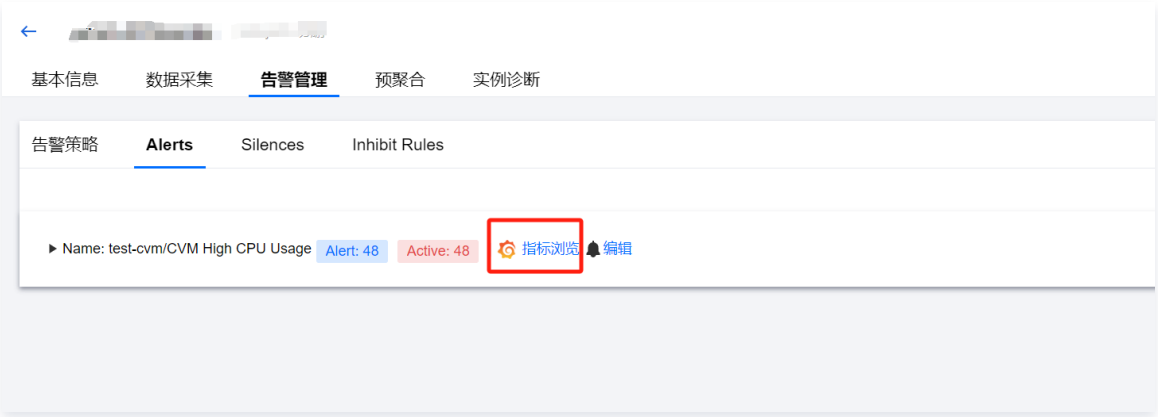
## 前提条件：

当前 Prometheus 实例已绑定 Grafana 服务，绑定指引可参考 Grafana 服务 。

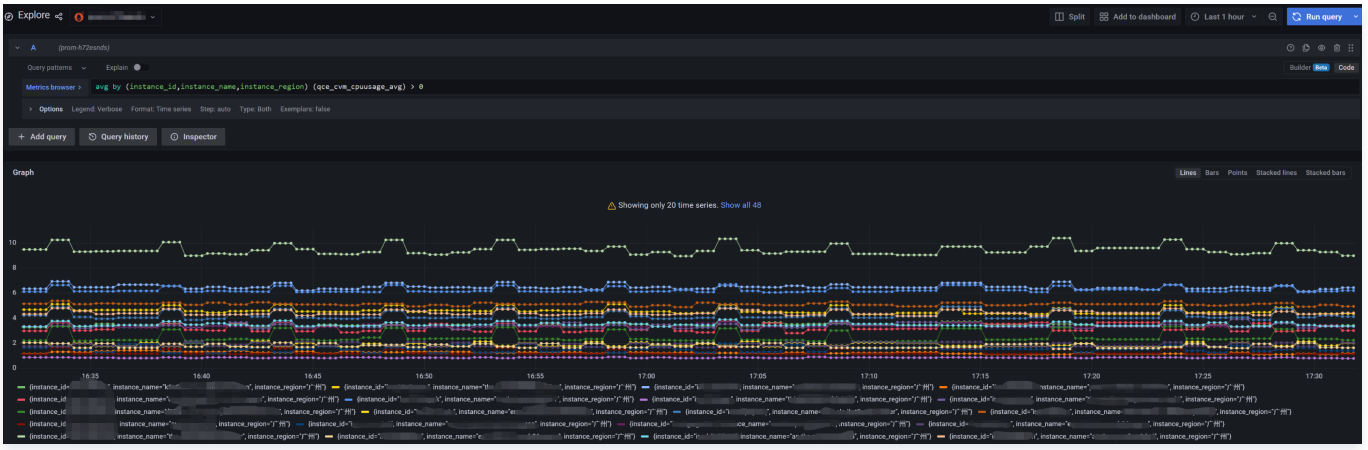
## 操作步骤

### 方法一：通过 Prometheus 控制台查看

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 实例列表中，单击**实例 ID/名称** 。
3. 进入 Prometheus 管理中心，在顶部导航栏中单击**告警管理 > Alerts** ，再单击 Alerts 页面中的**指标浏览**。

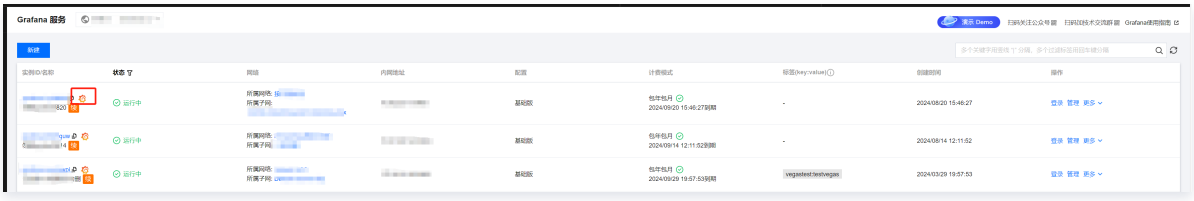



4. 单击**指标浏览**后会跳转到 Grafana 平台页面，展示与 Prometheus 告警相关的指标数据图表，如下图所示：

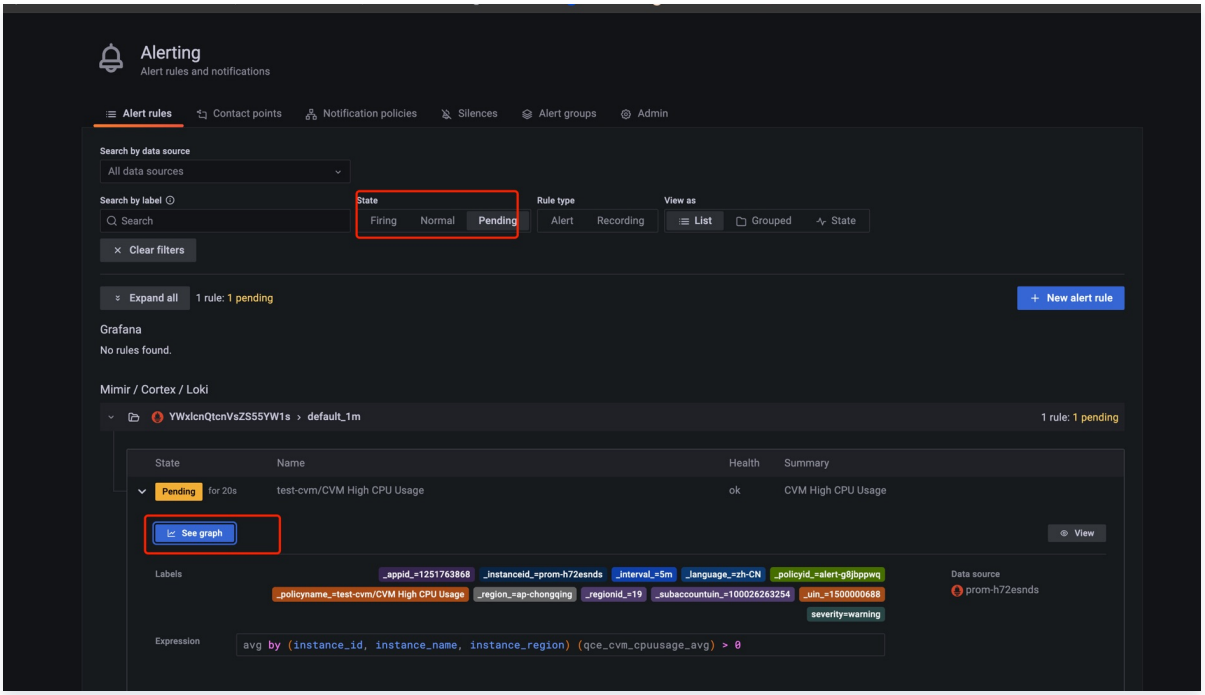


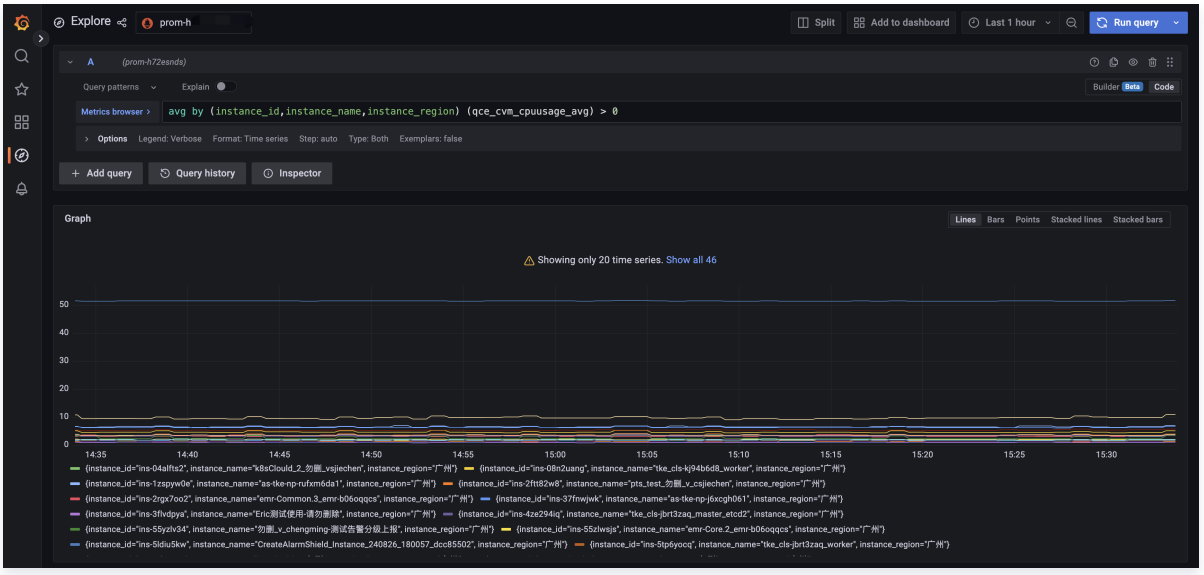

### 方法二：通过 Grafana 平台查看

1. 登录 Grafana 服务控制台 。
2. 单击需要查看实例右侧的 



3. 输入账号密码登录之后点击左侧导航栏的 进入Alerting 页面，再点击 See graph 即可查看与 Prometheus 告警相关的指标数据图表。



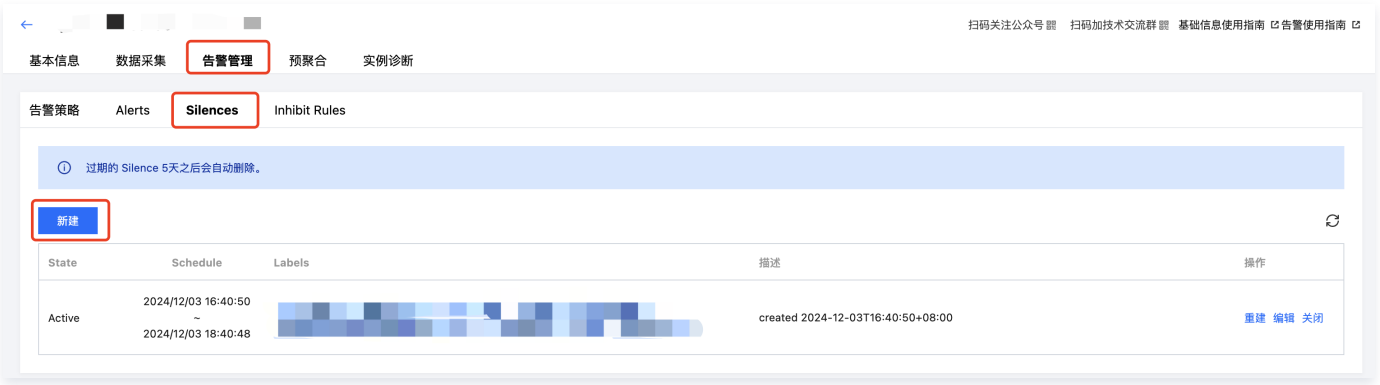
# 告警静默

最近更新时间：2024-12-18 10:02:33

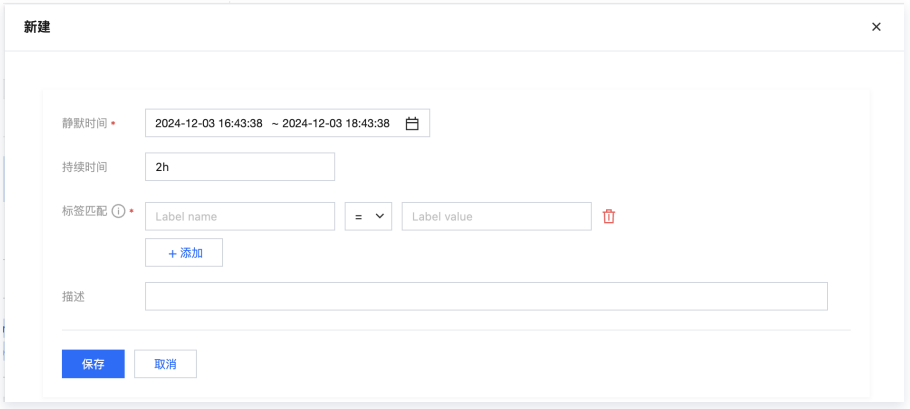
告警静默一般用于处理已知问题或在系统维护期间，暂时关闭特定警报的通知，或者对于频繁触发但不需要关注的警报，使用静默来减少干扰。告警静默需要设置静默策略，对满足匹配规则的告警进行静音处理，不发送告警通知。

## 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 实例列表中，单击**实例 ID/名称** 。
3. 进入 Prometheus 管理中心，在顶部导航栏中选择**告警管理 > Silences**，单击**新建**。




4. 跳转到新建页面后，根据页面提示配置抑制规则，配置完单击**保存**即可。




## 参数说明

| 参数 | 说明 |
| --- | --- |
| 静默时间 | 必填项，静默规则生效的起止时间。 |
| 持续时间 | 静默持续的时长，即静默时间的截止时间减去开始时间。 |
| 标签匹配 | 必填项，满足标签匹配规则的告警将被静默，输入标签名称、条件、标签值。可添加多个规则，满足所有匹配规则的告警信息才会被静默。 |
| 描述 | 对于静默规则的说明。 |

> **注意：**
> 在静默期间，符合匹配规则的告警消息和告警恢复消息都会被静默。

## 示例

### 使用场景：系统维护期间静默相关告警

### 场景描述

某个业务系统，设置了高负载时的告警，例如当 CPU 使用率超过某个阈值。为了进行定期的系统维护，需要在某个时间段内进行服务器升级，期间虽然会出现一些高负载，但已知这些不会影响用户体验，因此希望静音相关的告警。

假设需要静音的告警：

- 策略名称：HighCPUUsage。
- 相关实例：“server-”开头的实例。

则可以按照如下方式配置静默规则：

- 静默时间：2024-12-03 20:00:00~2024-12-03 22:00:00
- 持续时间：2h
- 标签匹配：
  - alertname=HighCPUUsage
  - instance=~server-.*
- 描述：系统维护，静默 CPU 高负载的告警。

### 整体效果

在2024-12-03 20:00:00 到 2024-12-03 22:00:00 的时间段内，所有匹配标签规则的告警（即“server-”开头实例的 CPU 高负载告警）将不会触发通知，从而避免了在维护期间的告警干扰。

# 告警抑制

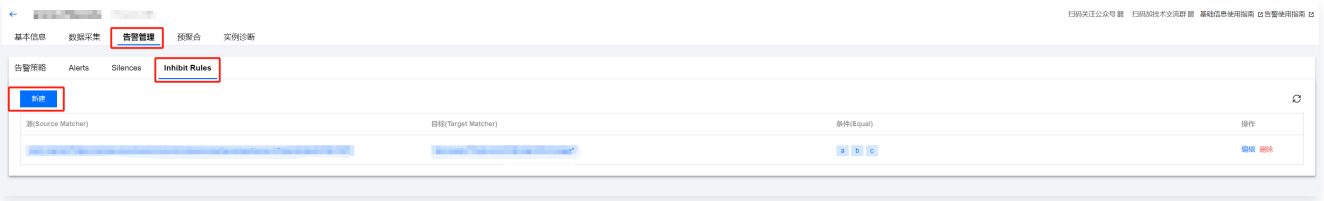
最近更新时间：2024-08-22 16:23:41

## 前言

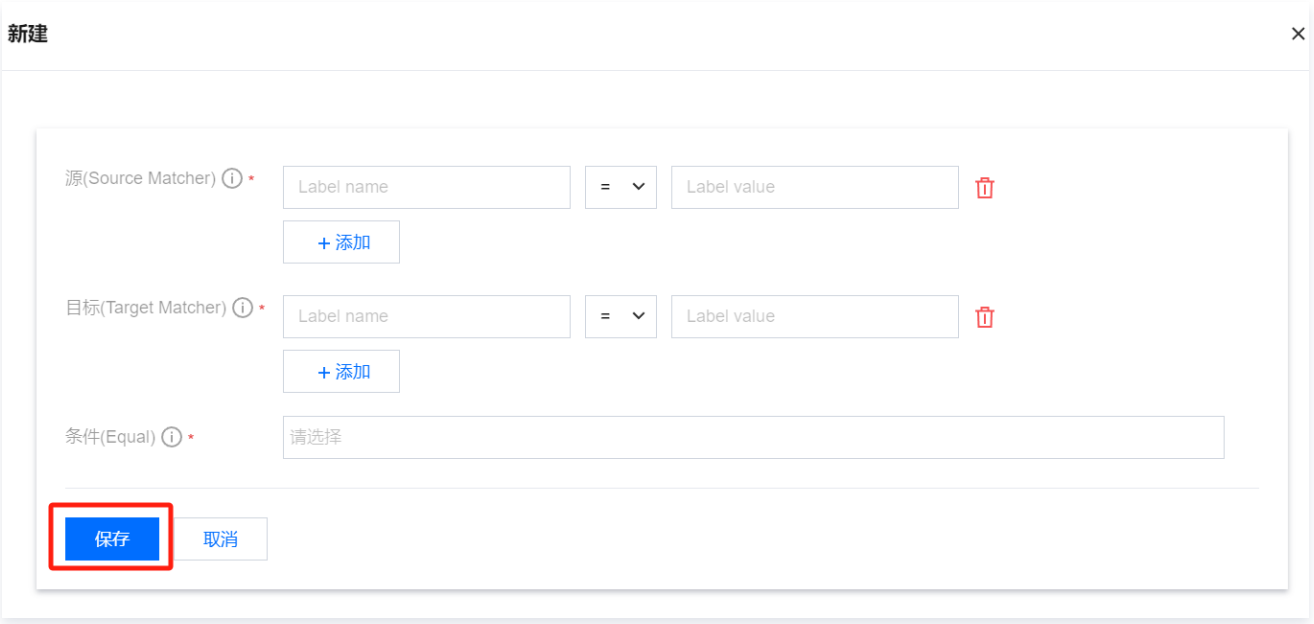
为了避免由于相同问题导致的成百上千的相似告警通知带来额外的运维工作量，我们增加了告警抑制功能。告警抑制指的是若某种类型的告警被触发，则抑制与之相关的其他相似告警。例如：如果告警内容是某个集群无法访问，则可以配置 Inhibition 规则，静默与该集群相关的所有其他告警。

## 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 实例列表中，单击**实例 ID/名称** 。
3. 进入 Prometheus 管理中心，在顶部导航栏中单击**告警管理 > Inhibit Rules > 新建**。




4. 跳转到新建页面后，根据页面提示配置抑制规则，配置完点击**保存**即可。




## 参数说明

| 参数 | 说明 |
| --- | --- |
| 源(Source Matcher) | 触发的告警，选择标签名称、条件、标签值。 |
| 目标(Target Matcher) | 需要被静默的告警，选择标签名称、条件、标签值。 |
| 条件(Equal) | 目标和源告警对于匹配条件中的标签名称必须具有相同的标签值，选择标签名称。 |

**说明：**

- **Inhibition 规则设置**：当存在满足某种规则的告警（源）时，抑制规则会静默满足另一种规则的告警（目标）。目标和源告警对于匹配条件中的标签名称必须具有相同的标签值。
- 为了防止警报自我抑制，与规则的目标端和源端都匹配的告警不能被与目标端和源端都匹配的其他告警（包括其自身）抑制。因此，建议告警的源和目标规则设计上要确保不会有任何告警同时匹配源规则和目标规则。

## 示例

### 使用场景：服务器 CPU 高负载告警

### 场景描述：

在一个监控系统中，配置了两个告警：

- 告警 A：CPU 负载超过90%。
- 告警 B：系统响应时间超过500ms。

这两个告警都是由于同一原因引起的，即服务器 CPU 高负载，导致系统性能下降。

**告警 A 的策略规则如下：**

```
alert: HighCPUUsage
expr: avg(rate(cpu_usage_seconds_total[5m])) by (instance) > 0.9
```

**告警 B 的策略规则如下：**

```
alert: HighResponseTime
expr: avg(response_time_seconds) by (instance) > 0.5
```

**则 Inhibition 规则配置方式如下：**

- 源：alert=HighCPUUsage
- 目标：alert=HighResponseTime
- 匹配条件：instance

### 整体效果：

- cpu_usage_seconds_total 指标在5分钟内的平均速率为95%，该指标的标签 instance=instanceX，则会触发告警 A，发送告警通知；
- response_time_seconds 指标的平均值为0.8s，该指标的标签 instance=instanceX，则会触发告警 B，但由于匹配上了 Inhibition 规则，所以不会发送告警通知。

# 标签管理
# 标签示例

最近更新时间：2024-10-22 14:10:22

## 简介

标签是腾讯云提供的用于标识云上资源的标记，是一个键-值对（Key-Value）。
您可以根据各种维度（例如业务、用途、负责人等）使用标签对 Prometheus 监控资源进行分类管理，通过标签非常方便地筛选过滤出对应的资源。标签键值对会严格按字符串进行解析匹配，腾讯云不会使用您设定的标签，标签仅用于您对资源的管理。
以下通过一个具体的案例来介绍标签的使用。

## 案例背景

某公司在腾讯云上拥有10个 Prometheus 监控服务实例，分属电商、游戏、文娱三个部门，服务于营销活动、游戏 A、游戏 B、后期制作等业务，三个部门对应的运维负责人为张三、李四、王五。

## 设置标签

为了方便管理，该公司使用标签分类管理对应的 Prometheus 监控服务资源，定义了下述标签键/值。

| 标签键 | 标签值 |
| --- | --- |
| 部门 | 电商、游戏、文娱 |
| 业务 | 营销活动、游戏 A、游戏 B、后期制作 |
| 运维负责人 | 张三、李四、王五 |

将这些标签键/值绑定到 Prometheus 实例上，资源与标签键/值的关系如下表所示：

| 实例 ID | 部门 | 业务 | 运维负责人 |
| --- | --- | --- | --- |
| prom-1jqwv1 | 电商 | 营销活动 | 王五 |
| prom-1jqwv12 | 电商 | 营销活动 | 王五 |
| prom-1jqwv13 | 游戏 | 游戏 A | 张三 |
| prom-1jqwv13 | 游戏 | 游戏 B | 张三 |
| prom-1jqwv14 | 游戏 | 游戏 B | 张三 |
| prom-1jqwv15 | 游戏 | 游戏 B | 李四 |
| prom-1jqwv16 | 游戏 | 游戏 B | 李四 |
| prom-1jqwv17 | 游戏 | 游戏 B | 李四 |
| prom-1jqwv18 | 文娱 | 后期制作 | 王五 |
| prom-1jqwv19 | 文娱 | 后期制作 | 王五 |
| prom-1jqwv110 | 文娱 | 后期制作 | 王五 |

## 使用标签

- 筛选出王五负责的 Prometheus 监控服务实例
  按照筛选规则筛选出运维负责人为“王五”的 Prometheus 资源即可，具体筛选办法请参见 使用标签 。
- 筛选出游戏部门中李四负责的 Prometheus 监控服务实例
  按照筛选规则筛选出部门为“游戏”、运维负责人为“李四”的 Prometheus 资源即可，具体筛选办法请参见 使用标签 。

# 使用标签

最近更新时间：2024-10-21 22:06:01

本文指导您在 Prometheus 监控服务控制台中，根据标签对实例进行资源筛选，过滤出对应的资源。

## 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表页顶部，选择地域。
3. 在实例列表右上角的搜索框，单击空白处，单击**标签**，弹出标签过滤选择框，如下图所示：


4. 在标签过滤选择框中选择对应的条件，单击**确定**之后进行过滤。
5. 如果需要调整对应的标签条件，单击搜索框里的**标签**，对后面的标签内容进行编辑。
6. 同时也支持直接单击实例列表中对应的标签值来进行过滤，如下图所示：

# 编辑标签

最近更新时间：2024-10-22 14:10:22

本文指导您在 Prometheus 监控服务控制台中，对目标实例进行标签的编辑操作。

## 操作步骤

### 对单个实例编辑标签

1. 登录 Prometheus 监控服务控制台 。
2. 在实例的管理页面，选择需要编辑标签的实例，选择**更多 > 实例配置 > 编辑标签**，如下图所示：


3. 在弹出的窗口中，根据实际需求进行添加、修改或者删除标签：

# 使用限制

最近更新时间：2024-10-21 22:06:01

标签是一个键-值对（Key-Value），您可以在 Prometheus 监控服务控制台，通过对 Prometheus 实例设置标签实现资源的分类管理。通过标签，可以非常方便筛选过滤出对应的资源。

## 数量限制

每个云资源允许的最大标签数是50。

## 标签键限制

- qcloud、tencent、project 开头为系统预留标签键，禁止创建。
- 只能为字母、数字、空格或汉字，支持 `+-=._:/@()[]（）【】,;><` 。
- 标签键长度最大为127个字符。

## 标签值限制

- 只支持字母、数字、空格、汉字或特殊符号。
- 标签值最大长度为255个字符。

# 访问控制
# 访问控制概述

最近更新时间：2024-10-21 22:06:01

如果您在腾讯云中使用到了 Prometheus 监控服务，该服务由不同的人管理，但都共享您的云账号密钥，将存在以下问题：

- 您的密钥由多人共享，泄密风险高。
- 您无法限制其他人的访问权限，易产生误操作造成安全风险。

此时，您就可以通过子账号实现不同的人管理不同的服务，来规避以上的问题。默认情况下，子账号无使用 Prometheus 权限。因此，我们需要创建策略来允许子账号使用他们所需要资源的权限。

## 简介

访问管理（Cloud Access Management，CAM）是腾讯云提供的一套 Web 服务，它主要用于帮助客户安全管理腾讯云账户下的资源的访问权限。通过 CAM，您可以创建、管理和销毁用户（组），并通过身份管理和策略管理控制哪些人可以使用哪些腾讯云资源。
当您使用 CAM 时，可以将策略与一个用户或一组用户关联起来，策略能够授权或者拒绝用户使用指定资源完成指定任务。有关 CAM 策略的更多相关基本信息，请参见 策略语法 。有关 CAM 策略的更多相关使用信息，请参见 策略 。
若您无需对子账号进行 Prometheus 相关资源的访问管理，您可以跳过此章节。跳过这些部分不会影响您对文档中其余部分的理解和使用。

## 入门

CAM 策略必须授权使用一个或多个 Prometheus 操作或者必须拒绝使用一个或多个 Prometheus 操作。同时还必须指定可以用于操作的资源（可以是全部资源，某些操作也可以是部分资源），策略还可以包含操作资源所设置的条件。
Prometheus 监控服务部分 API 操作不支持资源级权限，意味着对于该类 API 操作，您无法在使用该类操作的时候指定某个具体的资源来使用，而必须指定全部资源来使用。

# 策略设置

最近更新时间：2024-10-21 22:06:01

## 概述

访问策略可用于授予访问 Prometheus 监控服务实例的权限。访问策略使用基于 JSON 的访问策略语言。您可以通过访问策略语言授权指定委托人（principal）对指定的 Prometheus 监控服务资源执行指定的操作。
访问策略语言描述了策略的基本元素和用法，有关策略语言的说明可参见 CAM 策略管理 。

## 访问策略中的元素

访问策略语言包含以下基本意义的元素：

- **语句（statement）**：描述一条或多条权限的详细信息。该元素包括效力、操作、资源、条件等多个其他元素的权限或权限集合。一条策略有且仅有一个语句元素。
- **效力（effect）**：描述声明产生的结果是“允许”还是“显式拒绝”，包括 allow 和 deny 两种情况。该元素是必填项。
- **操作（action）**：描述允许或拒绝的操作。操作可以是 API（以 name 前缀描述）或者功能集（一组特定的 API，以 permid 前缀描述）。该元素是必填项。
- **资源（resource）**：描述授权的具体数据。资源是用六段式描述。每款产品的资源定义详情会有所区别。该元素是必填项。
- **条件（condition）**：描述策略生效的约束条件。条件包括操作符、操作键和操作值组成。条件值可包括时间、IP 地址等信息。有些服务允许您在条件中指定其他值。该元素是选填项。

## 元素用法

### 指定效力

如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。同时，也可以显式拒绝（deny）对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也无法访问。下面是指定允许效力的示例：

```
"effect" : "allow"
```

### 指定操作

腾讯云可观测平台定义了可在策略中指定一类控制台的操作，指定的操作按照操作性质分为读取部分接口（monitor:Describe*）和全部接口（monitor:*）。指定允许操作的示例如下：

```
"action": [
  "name/monitor:Describe*"
]
```

### 指定资源

资源（resource）元素描述一个或多个操作对象，如腾讯云 Prometheus 监控服务资源等。所有资源均可采用下述的六段式描述方式。

```
qcs:project_id:service_type:region:account:resource
```

参数说明如下：

| 参数 | 描述 | 是否必选 |
| --- | --- | --- |
| qcs | 是 qcloud service 的简称，表示是腾讯云的云服务 | 是 |
| project_id | 描述项目信息，仅为了兼容 CAM 早期逻辑，一般不填 | 否 |
| service_type | 产品简称，这里为 monitor | 是 |
| region | 描述地域信息 | 是 |
| account | 描述资源拥有者的主账号信息，即主账号的 ID，表示为 `uin/${OwnerUin}`，如 uin/100000000001 | 是 |

| | | |
|---|---|---|
| resource | 描述具体资源详情，前缀为 instance | 是 |

下面是一个 Prometheus 监控服务的六段式信息：

```
"resource":["qcs::monitor:ap-guangzhou:uin/100000000001:prom-instance/prom-73jingds"]
```

### 指定条件

访问策略语言可使您在授予权限时指定条件。主要是用于设置标签鉴权，标签条件只对绑定了该标签的集群生效，标签策略示例如下：

```
"condition": {
    "for_any_value:string_equal": {
        "qcs:tag": [
            "testkey&testvalue"
        ]
    }
}
```

这个语句含义是策略包含标签 key 为 testkey，value 为 testvalue 的资源。

## 实际案例

### 基于标签

如下案例中，策略的含义是允许访问 UIN 为1250000000下面的实例 ID 为 prom-73jingds 的资源，以及绑定了标签键为 testkey，标签值为 testvalue 的资源（如果该实例未属于标签 testkey& testvalue ，则仍不允许访问）。

```
{
   "version": "2.0",
    "statement": [
        {
            "action": [
                "name/monitor:*"
            ],
            "condition": {
               "for_any_value:string_equal": {
                    "qcs:tag": [
                        "testkey&testvalue"
                    ]
                }
            },
            "effect": "allow",
            "resource": [
                "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-73jingds"
            ]
        }
    ]
}
```

### 基于资源

基于资源 ID，分配指定资源的读写权限，主账号 ID 为1250000000：

- 配置广州地域（资源）只读权限。
  示例：为子用户分配， instance1(prom-73jingds)、instance2(prom-65jidfafk) 资源的只读权限。

```
{
  "version": "2.0",
  "statement": [{
      "effect": "allow",
```

```
        "action": [
            "name/monitor:Describe*"
        ],
        "resource": [
            "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-73jingds",
            "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-65jidfafk"
        ],
        "condition": []
    }]
}
```

- **配置广州地域（资源）部分读写权限。**
  **示例：为子用户分配，instance1(prom-73jingds) 资源的删除操作权限。**

```
{
  "version": "2.0",
  "statement": [{
        "effect": "allow",
        "resource": [
            "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-73jingds"
        ],
        "action": [
            "name/monitor:TerminatePrometheusInstances"
        ]
    }]
}
```

# 策略授予

最近更新时间：2024-10-21 22:06:01

## Prometheus 自定义策略

如果预设策略无法满足需求，可进入 访问管理控制台 > 策略， 单击**新建自定义策略**创建自定义策略。


创建自定义策略的方法可参见 策略设置 。

## 策略授权

已设置好的策略可以通过关联用户组或者子用户来授予权限。


## 自定义策略可授权资源类型

资源级权限指的是能够指定用户对哪些资源具有执行操作的能力。可以针对支持资源级权限的 Prometheus 服务操作，您可以控制何时允许用户执行操作或是允许用户使用特定资源。访问管理 CAM 中可授权的资源类型如下：

| 资源类型 | 授权策略中的资源描述方法 |
| --- | --- |
| Prometheus 监控服务 | `qcs::monitor:$region:$account:prom-instance/*`<br>`qcs::monitor:$region:$account:prom-instance/$instanceId` |

下表将介绍当前支持资源级权限的 Prometheus 监控服务 API 操作，设置策略时，action 填入 API 操作名称就可以对单独 API 进行控制，设置 action 也可以使用 `*` 作为通配符。

### 支持资源级授权的 API 列表

| API 操作 | API 描述 |
| --- | --- |
| DescribePrometheusInstances | 列出用户所有的 Prometheus 服务实例 |
| TerminatePrometheusInstances | 销毁 Prometheus 服务实例 |
| RecreatePrometheusInstance | 重新安装 Prometheus 服务实例 |
| ModifyPrometheusInstanceAttributes | 修改 Prometheus 实例相关属性 |
| ChangeGrafanaAdminPassword | 修改 Grafana Admin 密码 |
| UpgradeGrafanaDashboard | 升级 Grafana Dashboard |

| | |
|---|---|
| DescribePrometheusKubeClusters | 列出 Prometheus 可集成的 Kubernetes 集群列表 |
| InstallPrometheusAgent | 安装 Prometheus Agent |
| UninstallPrometheusAgent | 卸载 Prometheus Agent |
| DescribeServiceDiscovery | 列出 Prometheus 服务发现列表 |
| CreateServiceDiscovery | 创建 Prometheus 服务发现 |
| UpdateServiceDiscovery | 更新 Prometheus 服务发现 |
| DeleteServiceDiscovery | 删除 Prometheus 服务发现 |
| DescribePrometheusKubeBasicMonitor | 查询基础监控状态 |
| EnablePrometheusKubeBasicMonitor | 开启基础监控 |
| DisablePrometheusKubeBasicMonitor | 关闭基础监控 |
| DescribePrometheusAgentRuntime | 获取 Prometheus Agent 运行时状态 |
| DescribePrometheusJobTargets | 列出 Prometheus 指标抓取任务的状态信息 |
| DescribeRecordingRules | 查询预聚合规则 |
| CreateRecordingRule | 创建预聚合规则 |
| UpdateRecordingRule | 更新预聚合规则 |
| DeleteRecordingRules | 删除预聚合规则 |
| DescribeAlertRules | 报警规则查询 |
| DeleteAlertRules | 删除报警规则 |
| UpdateAlertRuleState | 更新报警策略状态 |
| CreateAlertRule | 创建告警规则 |
| UpdateAlertRule | 更新报警规则 |
| CreateAlarmNotice | 创建通知模板 |
| DeleteAlarmNotices | 删除告警通知模板（批量） |
| ModifyAlarmNotice | 修改通知模板 |
| DescribeAlarmNotices | 查询通知模板列表 |
| DescribeAlarmNotice | 查询单个通知模板的详情 |
| DescribeAlarmNoticeCallbacks | 查询账号下所有回调URL列表 |

### 不支持资源级授权的 API 列表

针对不支持资源级权限的 Prometheus 监控服务 API 操作，您仍可以向用户授予使用该操作的权限，但策略语句的资源（resource）元素必须指定为 `*`。

| API 操作 | API 描述 |
|---|---|
| CreatePrometheusInstance | 创建 Prometheus 服务实例 |

# 相关角色权限说明

最近更新时间：2025-04-24 18:57:12

在使用 Prometheus 监控服务过程中，为了能够使用相关云资源，会遇到多种需要进行服务授权的场景。在使用该服务的过程中主要涉及 TKE_QCSLinkedRoleInPrometheusService、TKE_QCSRole 和 CM_QCSLinkedRoleInTMP 三个服务角色。

- TKE_QCSLinkedRoleInPrometheusService 角色是用于授权 Prometheus 访问容器服务，默认关联的预设策略如下：

  QcloudAccessForTKELinkedRoleInPrometheusService：该策略仅用于腾讯云容器服务（TKE）访问其他云服务资源。包含对象存储（COS）相关操作权限。
- TKE_QCSRole 角色是用于授权容器服务，默认关联的预设策略如下：
  - QcloudAccessForTKERole：该策略用于腾讯云容器服务（TKE）访问云资源。
  - QcloudAccessForTKERoleInOpsManagement：该策略用于腾讯云容器服务（TKE）访问其他云服务资源。包含日志服务（CLS）相关操作权限。
- CM_QCSLinkedRoleInTMP 角色是用于授权 Prometheus 获取云资源信息，默认关联的预设策略如下：

  QcloudAccessForCMLinkedRoleInTMP：该策略用于 monitor 访问其他云服务资源。

## 操作场景

为了采集腾讯云容器服务（TKE）或集成中心其他组件监控的数据，Prometheus 服务实例需要访问相关 API 服务，需要您的授权委托，才能正常访问。此角色无需主动寻找配置，在未授权的情况下，使用相关功能时会自动弹出授权界面。

## 授权步骤

### 主账号授权

进入 Prometheus 实例购买页，填写完基本信息单击**立即购买**后将会出现授权提示框，进行 TKE_QCSLinkedRoleInPrometheusService 和 TKE_QCSRole 两个角色的授权，如下所示。单击**同意授权**即可授权成功。


若您需要使用集成中心的**云监控**、**CVM Node Exporter**、**CVM 进程监控**、**CVM 云服务器**、**EMR**、**Cdwch** 的一键安装功能，则需进行 CM_QCSLinkedRoleInTMP 角色的授权，如下图所示：


**单击提示框的点击授权，即会弹出如下窗口，单击同意授权后即可授权成功。**


**说明：**
**此次授权只会出现一次，如果您已授权，则不再出现该授权提示框。**

## 子账号授权

**主账号完成上述授权操作，成功创建了角色后，子账号无权限访问角色，需主账号对子账号授予 PassRole 权限，子账号才能正常使用，否则会提示失败。**
**在授予子账号 PassRole 权限时，请确保您的子账号有以下权限：**

| 权限说明 | 授予策略 |
| --- | --- |
| 需授予子账号访问 CAM 权限，主账号授予子账号的 PassRole 权限才会生效 | QcloudCamReadOnlyAccess<br>或 QcloudCamFullAccess |
| 云监控策略依赖于云产品策略，因此授予子账号 PassRole 权限前，需确保子账号可在 TKE 下正常访问 TKE 资源 | 详情请参见 容器服务（TKE）权限管理 |

**为确保上述权限授予成功，请参考以下步骤授予子账号 cam:PassRole 权限。**
**1. 使用主账号或具有管理权限的子账号在 访问管理 > 策略 > 新建自定义策略里边按策略语法创建，语法示例如下：**

```
{
 "version": "2.0",
 "statement": [
     {
         "effect": "allow",
         "action": "cam:PassRole",
         "resource":
"qcs::cam::uin/${OwnerUin}:role/tencentcloudServiceRoleName/TKE_QCSLinkedRoleInPrometheusService"
     },
     {
         "effect": "allow",
         "action": "cam:PassRole",
         "resource": "qcs::cam::uin/${OwnerUin}:roleName/TKE_QCSRole"
```

```
        }
    ]
}
```

为控制权限，示例语法限制了 cam:PassRole 仅对 TKE_QCSLinkedRoleInPrometheusService 和 TKE_QCSRole 服务角色生效，您可根据需要增减生效的角色。

2. 新建完后，参见 授权管理 在自定义策略下关联子账号即可。

# Grafana 服务

最近更新时间：2024-10-31 18:11:52

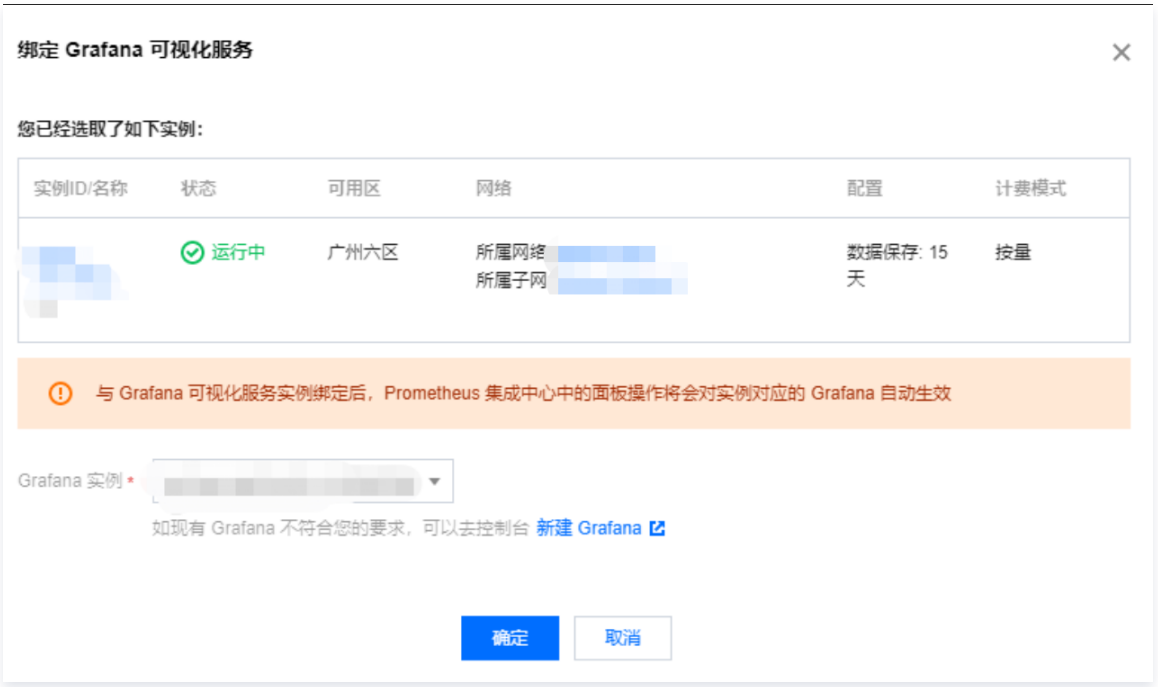
Prometheus 监控服务与 Grafana 服务 高度集成，一个 Grafana 实例可同时被多个 Prometheus 实例绑定，用于实现 Prometheus 数据的统一可视化。

## 操作步骤

Prometheus 监控服务支持您在 创建实例 时绑定 Grafana 实例，若您在购买 Prometheus 时未绑定实例可参考下列操作指引绑定。

## 绑定 Grafana 实例

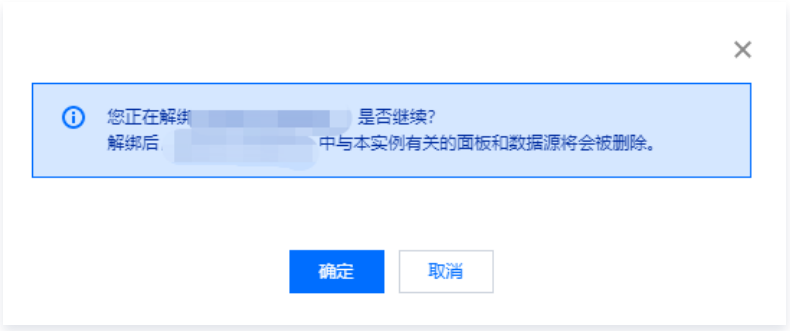
1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 列表中找对应的 Prometheus 实例，在操作列单击**更多 > Grafana > 绑定 Grafana**，或在 Grafana 访问地址列点击**绑定 Grafana** 。
3. 在弹框中选择 Grafana 实例，并单击**确定**即可。




**说明：**
仅支持选择与 Prometheus 实例同一VPC（私有网络）的 Grafana 实例，若没有符合的 Grafana ，请参见 操作指引 创建。

## 解绑 Grafana 实例

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 列表中找对应的 Prometheus 实例，在操作列单击**更多 > Grafana > 解绑 Grafana**。
3. 在弹框中单击**确定**解绑即可。



## 登录 Grafana 实例

1. 登录 Prometheus 监控服务控制台 。
2. 在 Prometheus 列表中找对应的 Prometheus 实例，单击 Grafana 访问地址列的**登录 Grafana** 功能。


3. 进入 Grafana 界面输入账号和密码即可登录，也可以选择 SSO 登录。

> **说明：**
> 更多 Grafana 操作请参见 Grafana 服务，如 Grafana 配置、图片渲染等操作。

# API 使用指南
# API 概览

最近更新时间：2024-10-21 22:06:01

## 支持的 API

Prometheus 监控服务所有支持的 API 与开源 Prometheus 提供的 API 有相同的参数输入和响应数据格式，未在此列出的 API 默认都不支持，或者部分 API 不是所有的版本都支持，以当前文档为准，所有 API 均为 HTTP 协议：

| API | 说明 | 使用方式 |
| --- | --- | --- |
| /api/v1/query | 查询某一时刻的数据 | 推荐使用 Grafana，可使用 HTTP 相关工具，部分 SDK 提供实现 |
| /api/v1/query_range | 查询时间范围类的数据 | 推荐使用 Grafana，可使用 HTTP 相关工具，部分 SDK 提供实现 |
| /api/v1/series | 查询 series | 推荐使用 Grafana，可使用 HTTP 相关工具，部分 SDK 提供实现 |
| /api/v1/labels | 查询标签名 | 推荐使用 Grafana，可使用 HTTP 相关工具，部分 SDK 提供实现 |
| /api/v1/label/{label_name}/values | 查询标签名多对应的值 | 推荐使用 Grafana，可使用 HTTP 相关工具，部分 SDK 提供实现 |
| /api/v1/prom/write | remote write 上报数据 | 常用方式为使用相关 Agent，例如 Prometheus |
| /metrics/{job}/{label-pairs*} | Pushgateway 上报数据 | 开源 SDK |

**说明：**
Prometheus 监控服务不直接提供 SDK 实现，需使用开源 Prometheus 实现，各语言的实现请参见：开源指引 。
Pushgateway 协议在这里仅作为 remote write 协议的补充，不支持删除操作，可以理解为我们提供的删除接口是空的不执行任何逻辑，特殊需求可自行部署 PushGateway 。

## 认证方法

以上提供的所有的 API 都需要认证，且所有的认证方式都支持下面两种。

## Basic Auth（推荐）

Basic Auth 兼容原生 Prometheus Query 的认证方式，用户名为用户的 APPID，进入控制台基本信息里面的 Token 即为密码。
了解更多：https://swagger.io/docs/specification/authentication/basic-authentication/

## Bearer Token

Bearer Token 随着实例创建而生成，进入控制台查看基本信息里面的 Token。
了解更多：https://swagger.io/docs/specification/authentication/bearer-authentication/

## API 响应及相关状态码

数据上报请求无固定的响应格式，程序只需关注状态码即可，错误响应最好记录详细的响应信息。
查询请求的响应格式为 JSON，基本结构如下：

```
{
  "status": "success" | "error",
  "data": <data>,

  // 当 status 状态为 error 时，下面的数据将被返回。
  "errorType": "<string>",
  "error": "<string>",

  // 当执行请求时有警告信息时，该字段将被填充返回。
  "warnings": ["<string>"]
```

```
}
```

相关状态码注解，错误相关的详细信息会包含在 HTTP 响应体内：

| 状态码 | 请求类型 | 概述 |
| --- | --- | --- |
| 400 | 查询 / 数据上报 | 请求参数错误 / 数据上报时如果 series 达到上限可能会出现此错误 |
| 401 | 查询 / 数据上报 | 认证失败 |
| 404 | 查询 / 数据上报 | API 不存在 |
| 422 | 查询 | 查询请求的表达式无法执行(RFC4918) |
| 429 | 数据上报 | 数据上报时 samples 速率达到上限（对于基础版，如果自监控上体现出余量还较多，Agent 会自动重新上报数据，这种情况理论上不会丢失数据，平均下来不超过限制） |
| 500 | 查询 / 数据上报 | 内部错误，频繁出现请联系我们 |
| 503 | 查询 / 数据上报 | 服务在启动、重建或升级中 / 查询请求被中止或者超时 |

# 数据写入

最近更新时间：2024-12-10 14:53:22

## 操作场景

在一些场景下（例如：Flink job 的生命周期可能很短，来不及等待 Prometheus 拉取其指标），我们需要将指标数据直接写入 Prometheus。此时可以使用 Remote Write 或者 Pushgateway 的方式。

## Remote Write

```
POST /api/v1/prom/write
```

Remote Write 是 Prometheus 标准的协议，更多介绍可以 访问这里 。使用 remote write 我们可以把 VPC 内其他 Prometheus 的数据写入到腾讯云托管的 Prometheus 服务中，对于数据的稳定性提升和迁移，都不失为一种不错的方案。

## Pushgateway

虽然 Prometheus 采集指标建议以 Pull 模式拉取为主，但是有些场景我们仍然需要使用 Push 推送的方式，详情请参见 官方文档 。详细使用参见 Pushgateway 接入 。

# 监控数据查询

最近更新时间：2024-08-16 11:52:01

## 操作场景

当我们有数据查询需求时，可以通过查询 API 请求监控数据。

## APPID/Token 获取方式

托管 Prometheus API 使用需要通过 `APPID` + `Token` 的方式进行鉴权访问。

- APPID 获取地址 。
- `Token` 从对应 Prometheus 实例的基本信息中获取。

## 查询 API 接口

```
GET /api/v1/query
POST /api/v1/query
```

## 查询参数

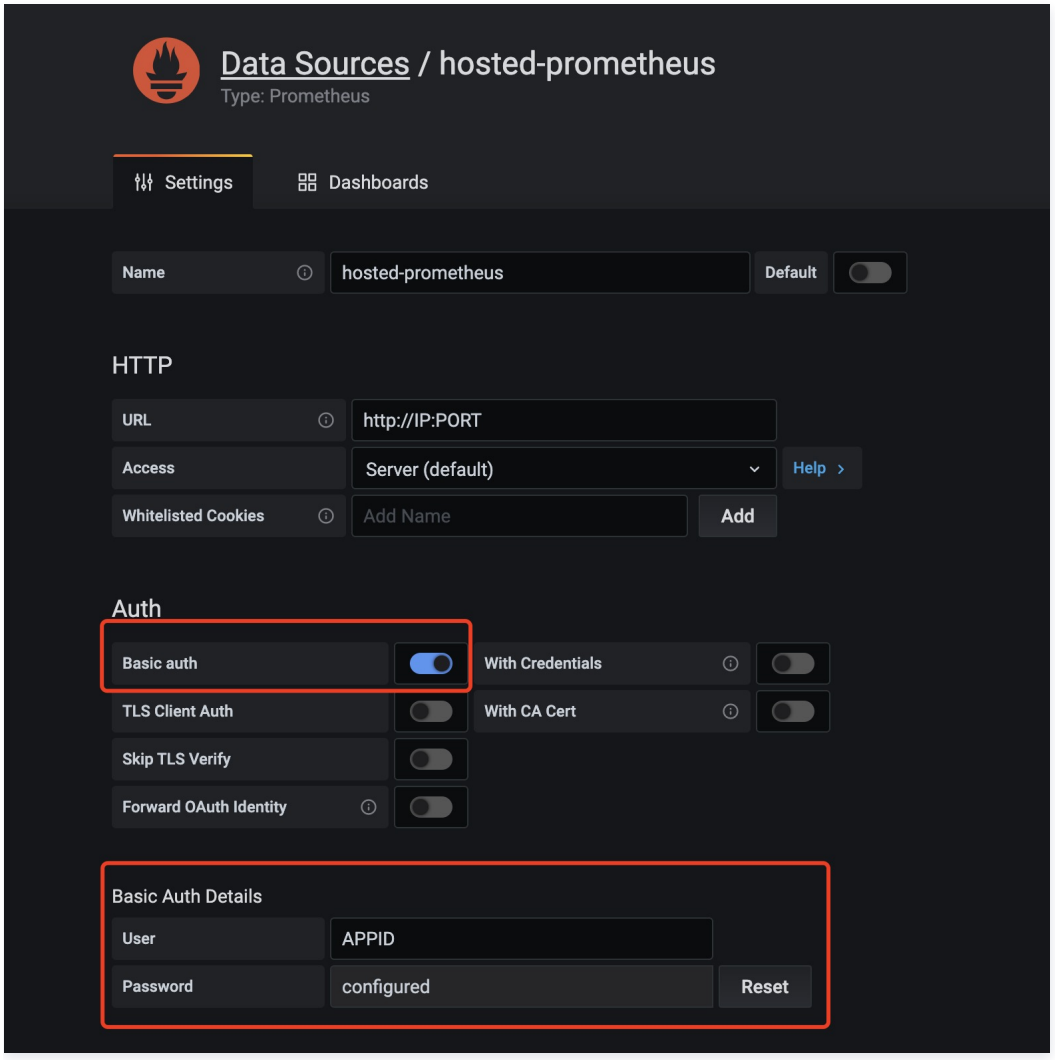
- query=<string>: Prometheus：查询表达式。
- time=<rfc3339 | unix_timestamp>：时间戳， 可选。
- timeout=<duration>：检测超时时间，可选。 默认由 `-query.timeout` 参数指定。

根据时间范围查询我们需要的数据是我们面临的最多的场景，这时我们需要用到 `/api/v1/query_range` 接口，示例如下：

```
$ curl -u "appid:token" 'http://IP:PORT/api/v1/query_range?query=up&start=2015-07-
01T20:10:30.781Z&end=2015-07-01T20:11:00.781Z&step=15s'
{
   "status" : "success",
   "data" : {
      "resultType" : "matrix",
      "result" : [
         {
            "metric" : {
               "__name__" : "up",
               "job" : "prometheus",
               "instance" : "localhost:9090"
            },
            "values" : [
               [ 1435781430.781, "1" ],
               [ 1435781445.781, "1" ],
               [ 1435781460.781, "1" ]
                  ]
         }
      ]
   }
}
```

## 自建 Grafana 添加数据源

我们可以通过自己部署的 Grafana 添加托管的 Prometheus 为数据源，方便我们在自己的 Grafana 中查看数据，前提是需要保证它们在同一 VPC 内，保证网络是可以互相访问的。
开启 BasicAuth 认证方法，并填写相应的认证信息即可，如下图配置。


Data Sources / hosted-prometheus
Type: Prometheus
Settings
Dashboards
Name
hosted-prometheus
Default
HTTP
URL
http://IP:PORT
Access
Server (default)
Help
Whitelisted Cookies
Add Name
Add
Auth
Basic auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity
Basic Auth Details
User
APPID
Password
configured
Reset

# 容器服务指标
# 按量付费免费指标

最近更新时间：2024-08-30 11:46:11

**按量付费模式部分指标基础免费存储15天，存储时长超过15天的实例，将按照超出的天数，收取免费指标的存储费用。**

| 所属配置文件 | 指标名 |
| --- | --- |
| node-exporter | node_boot_time_seconds |
| node-exporter | node_context_switches_total |
| node-exporter | node_cpu_seconds_total |
| node-exporter | node_disk_io_now |
| node-exporter | node_disk_io_time_seconds_total |
| node-exporter | node_disk_io_time_weighted_seconds_total |
| node-exporter | node_disk_read_bytes_total |
| node-exporter | node_disk_read_time_seconds_total |
| node-exporter | node_disk_reads_completed_total |
| node-exporter | node_disk_write_time_seconds_total |
| node-exporter | node_disk_writes_completed_total |
| node-exporter | node_disk_written_bytes_total |
| node-exporter | node_filefd_allocated |
| node-exporter | node_filesystem_avail_bytes |
| node-exporter | node_filesystem_free_bytes |
| node-exporter | node_filesystem_size_bytes |
| node-exporter | node_load1 |
| node-exporter | node_load15 |
| node-exporter | node_load5 |
| node-exporter | node_memory_Buffers_bytes |
| node-exporter | node_memory_Cached_bytes |
| node-exporter | node_memory_MemAvailable_bytes |
| node-exporter | node_memory_MemFree_bytes |
| node-exporter | node_memory_MemTotal_bytes |
| node-exporter | node_netstat_TcpExt_ListenDrops |
| node-exporter | node_netstat_Tcp_ActiveOpens |
| node-exporter | node_netstat_Tcp_CurrEstab |
| node-exporter | node_netstat_Tcp_InSegs |
| node-exporter | node_netstat_Tcp_OutSegs |

| | |
|---|---|
| node-exporter | node_netstat_Tcp_PassiveOpens |
| node-exporter | node_network_receive_bytes_total |
| node-exporter | node_network_transmit_bytes_total |
| node-exporter | node_sockstat_TCP_alloc |
| node-exporter | node_sockstat_TCP_inuse |
| node-exporter | node_sockstat_TCP_tw |
| node-exporter | node_sockstat_UDP_inuse |
| node-exporter | node_sockstat_sockets_used |
| node-exporter | node_uname_info |
| node-exporter | process_cpu_seconds_total |
| node-exporter | process_resident_memory_bytes |
| node-exporter | up |
| node-exporter | scrape_duration_seconds |
| node-exporter | scrape_timeout_seconds |
| node-exporter | scrape_series_added |
| node-exporter | scrape_samples_scraped |
| node-exporter | scrape_samples_post_metric_relabeling |
| cadvisor | container_cpu_usage_seconds_total |
| cadvisor | container_fs_limit_bytes |
| cadvisor | container_fs_reads_bytes_total |
| cadvisor | container_fs_usage_bytes |
| cadvisor | container_fs_writes_bytes_total |
| cadvisor | container_memory_working_set_bytes |
| cadvisor | container_network_receive_bytes_total |
| cadvisor | container_network_receive_packets_dropped_total |
| cadvisor | container_network_receive_packets_total |
| cadvisor | container_network_transmit_bytes_total |
| cadvisor | container_network_transmit_packets_dropped_total |
| cadvisor | container_network_transmit_packets_total |
| cadvisor | machine_cpu_cores |
| cadvisor | machine_memory_bytes |
| cadvisor | up |
| cadvisor | scrape_duration_seconds |
| cadvisor | scrape_timeout_seconds |
| cadvisor | scrape_series_added |

| | |
|---|---|
| cadvisor | scrape_samples_scraped |
| cadvisor | scrape_samples_post_metric_relabeling |
| eks-network | container_cpu_usage_seconds_total |
| eks-network | container_memory_working_set_bytes |
| eks-network | container_fs_limit_bytes |
| eks-network | container_fs_reads_bytes_total |
| eks-network | container_fs_usage_bytes |
| eks-network | container_fs_writes_bytes_total |
| eks-network | container_network_receive_bytes_total |
| eks-network | container_network_receive_packets_dropped_total |
| eks-network | container_network_receive_packets_total |
| eks-network | container_network_transmit_bytes_total |
| eks-network | container_network_transmit_packets_dropped_total |
| eks-network | container_network_transmit_packets_total |
| eks-network | up |
| eks-network | scrape_duration_seconds |
| eks-network | scrape_timeout_seconds |
| eks-network | scrape_series_added |
| eks-network | scrape_samples_scraped |
| eks-network | scrape_samples_post_metric_relabeling |
| kubelet | kubelet_cgroup_manager_duration_seconds_count |
| kubelet | kubelet_node_config_error |
| kubelet | kubelet_node_name |
| kubelet | kubelet_pleg_relist_duration_seconds_bucket |
| kubelet | kubelet_pleg_relist_duration_seconds_count |
| kubelet | kubelet_pleg_relist_interval_seconds_bucket |
| kubelet | kubelet_pod_start_duration_seconds_count |
| kubelet | kubelet_pod_worker_duration_seconds_count |
| kubelet | kubelet_running_containers |
| kubelet | kubelet_running_pods |
| kubelet | kubelet_runtime_operations_duration_seconds_bucket |
| kubelet | kubelet_runtime_operations_errors_total |
| kubelet | kubelet_runtime_operations_total |
| kubelet | process_cpu_seconds_total |
| kubelet | process_resident_memory_bytes |

| | |
|---|---|
| kubelet | rest_client_request_duration_seconds_bucket |
| kubelet | rest_client_requests_total |
| kubelet | storage_operation_duration_seconds_bucket |
| kubelet | storage_operation_duration_seconds_count |
| kubelet | storage_operation_errors_total |
| kubelet | volume_manager_total_volumes |
| kubelet | workqueue_adds_total |
| kubelet | workqueue_depth |
| kubelet | workqueue_queue_duration_seconds_bucket |
| kubelet | up |
| kubelet | scrape_duration_seconds |
| kubelet | scrape_timeout_seconds |
| kubelet | scrape_series_added |
| kubelet | scrape_samples_scraped |
| kubelet | scrape_samples_post_metric_relabeling |
| kube-state-metrics | kube_job_status_succeeded |
| kube-state-metrics | kube_job_status_failed |
| kube-state-metrics | kube_job_status_active |
| kube-state-metrics | kube_node_status_capacity_cpu_cores |
| kube-state-metrics | kube_node_status_capacity_memory_bytes |
| kube-state-metrics | kube_node_status_allocatable_cpu_cores |
| kube-state-metrics | kube_node_status_allocatable_memory_bytes |
| kube-state-metrics | kube_pod_info |
| kube-state-metrics | kube_pod_owner |
| kube-state-metrics | kube_pod_status_phase |
| kube-state-metrics | kube_pod_container_status_waiting |
| kube-state-metrics | kube_pod_container_status_running |
| kube-state-metrics | kube_pod_container_status_terminated |
| kube-state-metrics | kube_pod_container_status_restarts_total |
| kube-state-metrics | kube_pod_container_resource_requests_cpu_cores |
| kube-state-metrics | kube_pod_container_resource_requests_memory_bytes |
| kube-state-metrics | kube_pod_container_resource_limits_cpu_cores |
| kube-state-metrics | kube_pod_container_resource_limits_memory_bytes |
| kube-state-metrics | kube_replicaset_owner |
| kube-state-metrics | kube_statefulset_status_replicas |

| | |
|---|---|
| kube-state-metrics | kube_pod_container_resource_requests |
| kube-state-metrics | kube_pod_container_resource_limits |
| kube-state-metrics | kube_node_status_capacity |
| kube-state-metrics | kube_node_status_allocatable |
| kube-state-metrics | up |
| kube-state-metrics | scrape_duration_seconds |
| kube-state-metrics | scrape_timeout_seconds |
| kube-state-metrics | scrape_series_added |
| kube-state-metrics | scrape_samples_scraped |
| kube-state-metrics | scrape_samples_post_metric_relabeling |
| kube-controller-manager | rest_client_request_duration_seconds_bucket |
| kube-controller-manager | rest_client_requests_total |
| kube-controller-manager | workqueue_adds_total |
| kube-controller-manager | workqueue_depth |
| kube-controller-manager | workqueue_queue_duration_seconds_bucket |
| kube-controller-manager | process_cpu_seconds_total |
| kube-controller-manager | process_resident_memory_bytes |
| kube-controller-manager | storage_operation_duration_seconds_bucket |
| kube-controller-manager | storage_operation_duration_seconds_count |
| kube-controller-manager | storage_operation_errors_total |
| kube-controller-manager | up |
| kube-controller-manager | scrape_duration_seconds |
| kube-controller-manager | scrape_timeout_seconds |
| kube-controller-manager | scrape_series_added |
| kube-controller-manager | scrape_samples_scraped |
| kube-controller-manager | scrape_samples_post_metric_relabeling |
| kube-apiserver | apiserver_current_inflight_requests |
| kube-apiserver | apiserver_current_inqueue_requests |
| kube-apiserver | apiserver_init_events_total |
| kube-apiserver | apiserver_longrunning_gauge |
| kube-apiserver | apiserver_registered_watchers |
| kube-apiserver | apiserver_request_duration_seconds_bucket |
| kube-apiserver | apiserver_request_duration_seconds_sum |
| kube-apiserver | apiserver_request_duration_seconds_count |
| kube-apiserver | apiserver_request_filter_duration_seconds_bucket |

| | |
|---|---|
| kube-apiserver | apiserver_request_filter_duration_seconds_sum |
| kube-apiserver | apiserver_request_filter_duration_seconds_count |
| kube-apiserver | apiserver_request_total |
| kube-apiserver | apiserver_requested_deprecated_apis |
| kube-apiserver | apiserver_response_sizes_bucket |
| kube-apiserver | apiserver_response_sizes_sum |
| kube-apiserver | apiserver_response_sizes_count |
| kube-apiserver | apiserver_selfrequest_total |
| kube-apiserver | apiserver_tls_handshake_errors_total |
| kube-apiserver | apiserver_watch_events_sizes |
| kube-apiserver | apiserver_watch_events_sizes_bucket |
| kube-apiserver | apiserver_watch_events_sizes_sum |
| kube-apiserver | apiserver_watch_events_sizes_count |
| kube-apiserver | apiserver_watch_events_total |
| kube-scheduler | process_cpu_seconds_total |
| kube-scheduler | process_resident_memory_bytes |
| kube-scheduler | rest_client_request_duration_seconds_bucket |
| kube-scheduler | rest_client_requests_total |
| kube-scheduler | workqueue_adds_total |
| kube-scheduler | workqueue_depth |
| kube-scheduler | workqueue_queue_duration_seconds_bucket |
| kube-scheduler | up |
| kube-scheduler | scrape_duration_seconds |
| kube-scheduler | scrape_timeout_seconds |
| kube-scheduler | scrape_series_added |
| kube-scheduler | scrape_samples_scraped |
| kube-scheduler | scrape_samples_post_metric_relabeling |
| kube-proxy | rest_client_requests_total |
| kube-proxy | rest_client_request_duration_seconds_bucket |
| kube-proxy | process_resident_memory_bytes |
| kube-proxy | process_cpu_seconds_total |
| kube-proxy | up |
| kube-proxy | scrape_duration_seconds |
| kube-proxy | scrape_timeout_seconds |
| kube-proxy | scrape_series_added |

| | |
|---|---|
| kube-proxy | scrape_samples_scraped |
| kube-proxy | scrape_samples_post_metric_relabeling |

# 容器常用指标推荐

最近更新时间：2024-10-22 14:10:22

**基于大多数用户使用情况，专家建议配置如下常用的容器指标：**

> **注意：**
> 以下指标都是付费指标，指标的计费方式请参见 相关计费说明 。

| 所属配置文件 | 指标名 | 指标含义 |
| --- | --- | --- |
| kubelet | kubelet_running_container_count | kubelet_running_container_count Number of containers currently running. |
| kubelet | kubelet_running_pod_count | kubelet_running_pod_count Number of pods currently running. |
| kube-state-metrics | kube_pod_container_info | Information about a container in a pod. |
| kube-state-metrics | kube_deployment_status_replicas | The number of replicas per deployment. |
| kube-state-metrics | kube_deployment_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_pod_start_time | Start time in unix timestamp for a pod. |
| kube-state-metrics | kube_pod_status_ready | Describes whether the pod is ready to serve requests. |
| kube-state-metrics | kube_node_info | Information about a cluster node. |
| kube-state-metrics | kube_node_status_condition | The condition of a cluster node. |
| kube-state-metrics | kube_deployment_status_replicas_updated | The number of updated replicas per deployment. |
| kube-state-metrics | kube_deployment_status_replicas_available | The number of available replicas per deployment. |
| kube-state-metrics | kube_node_status_capacity_pods | The total pod resources of the node. |
| kube-state-metrics | kube_pod_container_status_ready | Describes whether the containers readiness check succeeded. |
| kube-state-metrics | kube_deployment_spec_replicas | Number of desired pods for a deployment. |
| kube-state-metrics | kube_pod_status_scheduled_time | Unix timestamp when pod moved into scheduled status. |
| kube-state-metrics | kube_node_status_allocatable_pods | The pod resources of a node that are available for scheduling. |
| kube-state-metrics | kube_pod_container_resource_limits | The number of requested limit resource by a container. |
| kube-state-metrics | node_filefd_maximum | File descriptor statistics: maximum. |
| kube-state-metrics | kube_pod_container_resource_requests | The number of requested request resource by a container. |

| | | |
|---|---|---|
| kube-state-metrics | kube_namespace_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_deployment_status_replicas_unavailable | The number of unavailable replicas per deployment. |
| kube-state-metrics | kube_pod_created | Unix creation timestamp. |
| kube-state-metrics | kube_pod_container_status_waiting_reason | Describes the reason the container is currently in waiting state. |
| kube-state-metrics | kube_daemonset_status_desired_number_scheduled | The number of nodes that should be running the daemon pod. |
| kube-state-metrics | kube_pod_restart_policy | Describes the restart policy in use by this pod. |
| kube-state-metrics | kube_deployment_metadata_generation | Sequence number representing a specific generation of the desired state. |
| kube-state-metrics | kube_statefulset_status_update_revision | Indicates the version of the StatefulSet used to generate Pods in the sequence [replicas-updatedReplicas]. |
| kube-state-metrics | kube_node_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_statefulset_replicas | Number of desired pods for a StatefulSet. |
| kube-state-metrics | kube_statefulset_status_observed_generation | The generation observed by the StatefulSet controller. |
| kube-state-metrics | kube_pod_container_status_last_terminated_reason | Describes the last reason the container was in terminated state. |
| kube-state-metrics | kube_replicaset_spec_replicas | Number of desired pods for a ReplicaSet. |
| kube-state-metrics | kube_statefulset_created | Unix creation timestamp. |
| kube-state-metrics | kube_statefulset_status_replicas_current | The number of current replicas per StatefulSet. |
| kube-state-metrics | kube_statefulset_status_current_revision | Indicates the version of the StatefulSet used to generate Pods in the sequence [0]. |
| kube-state-metrics | kube_statefulset_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_deployment_created | Unix creation timestamp. |
| kube-state-metrics | kube_namespace_created | Unix creation timestamp. |
| kube-state-metrics | kube_daemonset_status_number_ready | The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and ready. |
| kube-state-metrics | kube_deployment_status_observed_generation | The generation observed by the deployment controller. |
| kube-state-metrics | kube_endpoint_info | Information about endpoint. |

| | | |
|---|---|---|
| kube-state-metrics | kube_statefulset_status_replicas_updated | The number of updated replicas per StatefulSet. |
| kube-state-metrics | kube_statefulset_metadata_generation | Sequence number representing a specific generation of the desired state for the StatefulSet. |
| kube-state-metrics | kube_secret_created | Unix creation timestamp. |
| kube-state-metrics | kube_endpoint_address_not_ready | Number of addresses not ready in endpoint. |
| kube-state-metrics | kube_secret_type | Type about secret. |
| kube-state-metrics | kube_deployment_spec_paused | Whether the deployment is paused and will not be processed by the deployment controller. |
| kube-state-metrics | kube_pod_container_status_terminated_reason | Describes the reason the container is currently in terminated state. |
| kube-state-metrics | kube_statefulset_status_replicas_ready | The number of ready replicas per StatefulSet. |
| kube-state-metrics | kube_endpoint_address_available | Number of addresses available in endpoint. |
| kube-state-metrics | kube_secret_info | Information about secret. |
| kube-state-metrics | kube_service_info | Information about service. |
| kube-state-metrics | kube_node_status_allocatable | The allocatable for different resources of a node that are available for scheduling. |
| kube-state-metrics | kube_endpoint_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_deployment_status_condition | The current status conditions of a deployment. |
| kube-state-metrics | kube_endpoint_created | Unix creation timestamp. |
| kube-state-metrics | kube_replicaset_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_replicaset_metadata_generation | Sequence number representing a specific generation of the desired state. |
| kube-state-metrics | kube_namespace_status_phase | kubernetes namespace status phase. |
| kube-state-metrics | kube_service_created | Unix creation timestamp. |
| kube-state-metrics | kube_configmap_created | Unix creation timestamp. |
| kube-state-metrics | kube_secret_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_deployment_spec_strategy_rollingupdate_max_surge | Maximum number of replicas that can be scheduled above the desired number of replicas during a rolling update of a deployment. |

| | | |
|---|---|---|
| kube-state-metrics | kube_configmap_metadata_resource_version | Resource version representing a specific version of the configmap. |
| kube-state-metrics | kube_pod_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_replicaset_status_replicas | The number of replicas per ReplicaSet. |
| kube-state-metrics | kube_node_created | Unix creation timestamp. |
| kube-state-metrics | kube_service_spec_type | Type about service. |
| kube-state-metrics | kube_secret_metadata_resource_version | Resource version representing a specific version of secret. |
| kube-state-metrics | kube_configmap_info | Information about configmap. |
| kube-state-metrics | kube_replicaset_status_observed_generation | The generation observed by the ReplicaSet controller. |
| kube-state-metrics | kube_service_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_replicaset_created | Unix creation timestamp. |
| kube-state-metrics | kube_deployment_spec_strategy_rollingupdate_max_unavailable | Maximum number of unavailable replicas during a rolling update of a deployment. |
| kube-state-metrics | kube_replicaset_status_ready_replicas | The number of ready replicas per ReplicaSet. |
| kube-state-metrics | kube_replicaset_status_fully_labeled_replicas | The number of fully labeled replicas per ReplicaSet. |
| kube-state-metrics | kube_pod_status_scheduled | Describes the status of the scheduling process for the pod. |
| kube-state-metrics | kube_storageclass_created | Unix creation timestamp. |
| kube-state-metrics | kube_daemonset_status_number_misscheduled | The number of nodes running a daemon pod but are not supposed to. |
| kube-state-metrics | kube_storageclass_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_node_status_capacity | The capacity for different resources of a node. |
| kube-state-metrics | kube_daemonset_status_current_number_scheduled | The number of nodes running at least one daemon pod and are supposed to. |
| kube-state-metrics | kube_storageclass_info | Information about storageclass. |
| kube-state-metrics | kube_node_spec_unschedulable | Whether a node can schedule new pods. |
| kube-state-metrics | kube_daemonset_status_number_available | The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and available |

| | | |
|---|---|---|
| kube-state-metrics | kube_daemonset_labels | Kubernetes labels converted to Prometheus labels. |
| kube-state-metrics | kube_daemonset_created | Unix creation timestamp. |
| kube-state-metrics | kube_daemonset_status_number_unavailable | The number of nodes that should be running the daemon pod and have none of the daemon pod running and available. |
| kube-state-metrics | kube_daemonset_metadata_generation | Sequence number representing a specific generation of the desired state. |
| kube-state-metrics | kube_mutatingwebhookconfiguration_info | Information about the MutatingWebhookConfiguration. |
| kube-state-metrics | kube_mutatingwebhookconfiguration_created | Unix creation timestamp. |
| kube-state-metrics | kube_mutatingwebhookconfiguration_metadata_resource_version | Resource version representing a specific version of the MutatingWebhookConfiguration. |
| kube-state-metrics | kube_daemonset_updated_number_scheduled | The total number of nodes that are running updated daemon pod. |
| node-exporter | node_filesystem_files_free | Filesystem total free file nodes. |
| node-exporter | node_filesystem_files | Filesystem total file nodes. |
| node-exporter | node_sockstat_UDP_mem_bytes | Number of UDP sockets in state mem_bytes. |
| node-exporter | node_nf_conntrack_entries_limit | Maximum size of connection tracking table. |
| node-exporter | node_memory_Shmem_bytes | Memory information field Shmem_bytes. |
| node-exporter | node_netstat_Tcp_RetransSegs | Statistic TcpRetransSegs. |
| node-exporter | node_sockstat_TCP_mem_bytes | Number of TCP sockets in state mem_bytes. |
| node-exporter | node_network_info | Non-numeric data from /sys/class/net/<iface> |
| node-exporter | node_filesystem_readonly | Filesystem read-only status. |
| node-exporter | node_exporter_build_info | A metric with a constant '1' value labeled by version. |
| node-exporter | node_network_iface_link_mode | iface_link_mode value of /sys/class/net/<iface>. |
| node-exporter | node_network_receive_packets_total | Network device statistic receive_packets. |
| node-exporter | node_network_transmit_packets_total | Network device statistic transmit_packets. |
| node-exporter | node_memory_Mlocked_bytes | Memory information field Mlocked_bytes. |
| node-exporter | node_network_iface_id | iface_id value of /sys/class/net/<iface>. |
| node-exporter | node_memory_WritebackTmp_bytes | Memory information field WritebackTmp_bytes. |
| node-exporter | kube_service_status_load_balancer_ingress | Service load balancer ingress status. |
| node-exporter | node_vmstat_pgpgout | /proc/vmstat information field pgpgout. |
| node-exporter | node_nf_conntrack_entries | Number of currently allocated flow entries for connection tracking. |
| node-exporter | node_memory_Inactive_file_bytes | Memory information field Inactive_file_bytes. |

| | | |
|---|---|---|
| node-exporter | node_memory_SwapFree_bytes | Memory information field SwapFree_bytes. |
| node-exporter | node_sockstat_TCP_mem | Number of TCP sockets in state mem. |
| node-exporter | node_memory_Slab_bytes | Memory information field Slab_bytes. |
| node-exporter | node_network_transmit_errs_total | Network device statistic transmit_errs. |
| node-exporter | node_memory_Active_bytes | Memory information field Active_bytes. |
| node-exporter | node_procs_blocked | Number of processes blocked waiting for I/O to complete. |
| node-exporter | node_sockstat_UDP_mem | Number of UDP sockets in state mem. |
| node-exporter | node_timex_maxerror_seconds | Maximum error in seconds. |
| node-exporter | node_memory_Inactive_bytes | Memory information field Inactive_bytes. |
| node-exporter | node_network_receive_errs_total | Network device statistic receive_errs. |
| node-exporter | node_memory_Unevictable_bytes | Memory information field Unevictable_bytes. |
| node-exporter | node_memory_KernelStack_bytes | Memory information field KernelStack_bytes. |
| node-exporter | node_procs_running | Number of processes in runnable state. |
| node-exporter | node_memory_SwapTotal_bytes | Memory information field SwapTotal_bytes. |
| node-exporter | node_netstat_IpExt_OutOctets | Statistic IpExtOutOctets. |
| node-exporter | node_memory_Active_file_bytes | Memory information field Active_file_bytes. |
| node-exporter | node_memory_SwapCached_bytes | Memory information field SwapCached_bytes. |
| node-exporter | node_netstat_Icmp_InMsgs | Statistic IcmpInMsgs. |
| node-exporter | node_forks_total | Total number of forks. |
| node-exporter | node_sockstat_RAW_inuse | Number of RAW sockets in state inuse. |
| node-exporter | node_time_seconds | System time in seconds since epoch (1970). |
| node-exporter | node_vmstat_pgpgin | /proc/vmstat information field pgpgin. |
| node-exporter | node_memory_Mapped_bytes | Memory information field Mapped_bytes. |
| node-exporter | node_memory_SUnreclaim_bytes | Memory information field SUnreclaim_bytes. |
| node-exporter | node_memory_HardwareCorrupted_bytes | Memory information field HardwareCorrupted_bytes. |
| node-exporter | node_memory_PageTables_bytes | Memory information field PageTables_bytes. |
| node-exporter | node_netstat_Udp6_InDatagrams | Statistic Udp6InDatagrams. |
| node-exporter | node_netstat_Icmp_OutMsgs | Statistic IcmpOutMsgs. |
| node-exporter | node_netstat_Udp6_NoPorts | Statistic Udp6NoPorts. |
| node-exporter | node_memory_AnonPages_bytes | Memory information field AnonPages_bytes. |
| node-exporter | node_memory_Committed_AS_bytes | Memory information field Committed_AS_bytes. |
| node-exporter | node_netstat_TcpExt_ListenOverflows | Statistic TcpExtListenOverflows. |

| | | |
|---|---|---|
| node-exporter | node_netstat_UdpLite_InErrors | Statistic UdpLiteInErrors. |
| node-exporter | node_entropy_available_bits | Bits of available entropy. |
| node-exporter | node_memory_Inactive_anon_bytes | Memory information field Inactive_anon_bytes. |
| node-exporter | node_vmstat_pswpin | /proc/vmstat information field pswpin. |
| node-exporter | node_memory_AnonHugePages_bytes | Memory information field AnonHugePages_bytes. |
| node-exporter | node_memory_SReclaimable_bytes | Memory information field SReclaimable_bytes. |
| node-exporter | node_netstat_IpExt_InOctets | Statistic IpExtInOctets. |
| node-exporter | node_netstat_Udp_NoPorts | Statistic UdpNoPorts. |
| node-exporter | node_timex_sync_status | Is clock synchronized to a reliable server (1 = yes). |
| node-exporter | node_memory_CommitLimit_bytes | Memory information field CommitLimit_bytes. |
| node-exporter | node_memory_VmallocChunk_bytes | Memory information field VmallocChunk_bytes. |
| node-exporter | node_netstat_Udp_InDatagrams | Statistic UdpInDatagrams. |
| node-exporter | node_netstat_Icmp6_InErrors | Statistic Icmp6InErrors. |
| node-exporter | node_netstat_Icmp6_OutMsgs | Statistic Icmp6OutMsgs. |
| node-exporter | node_netstat_UdpLite6_InErrors | Statistic UdpLite6InErrors. |
| node-exporter | node_netstat_TcpExt_SyncookiesSent | Statistic TcpExtSyncookiesSent. |
| node-exporter | node_netstat_Tcp_InErrs | Statistic TcpInErrs. |
| node-exporter | node_intr_total | Total number of interrupts serviced. |
| node-exporter | node_timex_offset_seconds | Time offset in between local system and reference clock. |
| node-exporter | node_memory_Bounce_bytes | Memory information field Bounce_bytes. |
| node-exporter | node_memory_Writeback_bytes | Memory information field Writeback_bytes. |
| node-exporter | node_netstat_Udp_OutDatagrams | Statistic UdpOutDatagrams. |
| node-exporter | node_netstat_Icmp6_InMsgs | Statistic Icmp6InMsgs. |
| node-exporter | node_netstat_Ip6_OutOctets | Statistic Ip6OutOctets. |
| node-exporter | node_netstat_Ip_Forwarding | Statistic IpForwarding. |
| node-exporter | node_sockstat_TCP_orphan | Number of TCP sockets in state orphan. |
| node-exporter | node_netstat_Ip6_InOctets | Statistic Ip6InOctets. |
| node-exporter | node_netstat_TcpExt_SyncookiesFailed | Statistic TcpExtSyncookiesFailed. |
| node-exporter | node_netstat_Udp_InErrors | Statistic UdpInErrors. |
| node-exporter | node_vmstat_pgmajfault | /proc/vmstat information field pgmajfault. |
| node-exporter | node_network_transmit_drop_tot | Network device statistic transmit_drop. |

| | | |
|---|---|---|
| | al | |
| node-exporter | node_vmstat_pswpout | /proc/vmstat information field pswpout. |
| node-exporter | node_network_up | Value is 1 if operstate is 'up'. |
| node-exporter | node_memory_NFS_Unstable_bytes | Memory information field NFS_Unstable_bytes. |
| node-exporter | node_memory_VmallocTotal_bytes | Memory information field VmallocTotal_bytes. |
| node-exporter | node_sockstat_FRAG_inuse | Number of FRAG sockets in state inuse. |
| node-exporter | node_memory_Dirty_bytes | Memory information field Dirty_bytes. |
| node-exporter | node_netstat_Udp6_InErrors | Statistic Udp6InErrors. |
| node-exporter | node_netstat_TcpExt_SyncookiesRecv | Statistic TcpExtSyncookiesRecv. |
| node-exporter | node_netstat_Udp6_OutDatagrams | Statistic Udp6OutDatagrams. |
| node-exporter | node_memory_HugePages_Rsvd | Memory information field HugePages_Rsvd. |
| node-exporter | node_arp_entries | ARP entries by device. |
| node-exporter | node_network_carrier | carrier value of /sys/class/net/<iface>. |
| node-exporter | node_timex_pps_stability_exceeded_total | Pulse per second count of stability limit exceeded events. |
| node-exporter | node_network_receive_compressed_total | Network device statistic receive_compressed. |
| node-exporter | node_network_transmit_carrier_total | Network device statistic transmit_carrier. |
| node-exporter | node_memory_DirectMap2M_bytes | Memory information field DirectMap2M_bytes. |
| node-exporter | node_memory_Hugepagesize_bytes | Memory information field Hugepagesize_bytes. |
| node-exporter | node_network_address_assign_type | address_assign_type value of /sys/class/net/<iface>. |
| node-exporter | node_network_receive_multicast_total | Network device statistic receive_multicast. |
| node-exporter | node_network_transmit_compressed_total | Network device statistic transmit_compressed. |
| node-exporter | node_memory_DirectMap4k_bytes | Memory information field DirectMap4k_bytes. |
| node-exporter | node_network_transmit_queue_length | transmit_queue_length value of /sys/class/net/<iface>. |
| node-exporter | node_memory_HugePages_Free | Memory information field HugePages_Free. |
| node-exporter | node_network_receive_frame_total | Network device statistic receive_frame. |
| node-exporter | node_memory_HugePages_Total | Memory information field HugePages_Total. |
| node-exporter | node_network_flags | flags value of /sys/class/net/<iface>. |

| | | |
|---|---|---|
| node-exporter | node_network_receive_fifo_total | Network device statistic receive_fifo. |
| node-exporter | node_scrape_collector_duration_seconds | node_exporter: Duration of a collector scrape. |
| node-exporter | node_network_speed_bytes | speed_bytes value of /sys/class/net/<iface>. |
| node-exporter | node_sockstat_UDPLITE_inuse | Number of UDPLITE sockets in state inuse. |
| node-exporter | node_cpu_guest_seconds_total | Seconds the cpus spent in guests (VMs) for each mode. |
| node-exporter | node_filesystem_device_error | Whether an error occurred while getting statistics for the given device. |
| node-exporter | node_scrape_collector_success | node_exporter: Whether a collector succeeded. |
| node-exporter | node_network_transmit_fifo_total | Network device statistic transmit_fifo. |
| node-exporter | node_vmstat_pgfault | /proc/vmstat information field pgfault. |
| node-exporter | node_network_device_id | device_id value of /sys/class/net/<iface>. |
| node-exporter | node_network_protocol_type | protocol_type value of /sys/class/net/<iface>. |
| node-exporter | node_network_receive_drop_total | Network device statistic receive_drop. |
| node-exporter | node_timex_estimated_error_seconds | Estimated error in seconds. |
| node-exporter | node_disk_writes_merged_total | The number of writes merged. |
| node-exporter | node_network_transmit_colls_total | Network device statistic transmit_colls. |
| node-exporter | node_timex_tick_seconds | Seconds between clock ticks. |
| node-exporter | node_textfile_scrape_error | 1 if there was an error opening or reading a file |
| node-exporter | node_network_iface_link | iface_link value of /sys/class/net/<iface>. |
| node-exporter | node_disk_reads_merged_total | The total number of reads merged. |
| node-exporter | node_timex_status | Value of the status array bits. |
| node-exporter | node_netstat_Icmp_InErrors | Statistic IcmpInErrors. |
| node-exporter | node_memory_Active_anon_bytes | Memory information field Active_anon_bytes. |
| node-exporter | node_timex_pps_frequency_hertz | Pulse per second frequency. |
| node-exporter | node_network_mtu_bytes | mtu_bytes value of /sys/class/net/<iface>. |
| node-exporter | node_timex_tai_offset_seconds | International Atomic Time (TAI) offset. |
| node-exporter | node_timex_pps_jitter_total | Pulse per second count of jitter limit exceeded events. |
| node-exporter | node_timex_pps_jitter_seconds | Pulse per second jitter. |
| node-exporter | node_network_net_dev_group | net_dev_group value of /sys/class/net/<iface>. |
| node-exporter | node_network_dormant | dormant value of /sys/class/net/<iface>. |
| node-exporter | node_timex_pps_calibration_total | Pulse per second count of calibration intervals. |
| node-exporter | node_timex_pps_shift_seconds | Pulse per second interval duration. |
| node-exporter | node_timex_pps_error_total | Pulse per second count of calibration errors. |

| | | |
|---|---|---|
| node-exporter | node_memory_VmallocUsed_bytes | Memory information field VmallocUsed_bytes. |
| node-exporter | node_timex_frequency_adjustment_ratio | Local clock frequency adjustment. |
| node-exporter | node_sockstat_FRAG_memory | Number of FRAG sockets in state memory. |
| node-exporter | node_memory_HugePages_Surp | Memory information field HugePages_Surp. |
| node-exporter | node_timex_loop_time_constant | Phase-locked loop time constant. |
| node-exporter | node_timex_pps_stability_hertz | Pulse per second stability. |

# 容器监控图表指标

最近更新时间：2025-05-30 11:48:02

## 集群监控概览

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
| --- | --- | --- | --- |
| CPU Requests Commitment | sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"}) / sum(kube_node_status_allocatable_cpu_cores{cluster="$cluster"}) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | kube_node_status_allocatable_cpu_cores | kube-state-metrics |
| CPU Limits Commitment | sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster"}) / sum(kube_node_status_allocatable_cpu_cores{cluster="$cluster"}) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | kube_node_status_allocatable_cpu_cores | kube-state-metrics |
| Memory Utilisation | 1 - sum(:node_memory_MemAvailable_bytes:sum{cluster="$cluster"}) / sum(node_memory_MemTotal_bytes{cluster="$cluster"}) | node_memory_MemAvailable_bytes | node-exporter |
| | | node_memory_MemTotal_bytes | node-exporter |
| Memory Requests Commitment | sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) / sum(kube_node_status_allocatable_memory_bytes{cluster="$cluster"}) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | kube_node_status_allocatable_memory_bytes | kube-state-metrics |
| Memory Limits Commitment | sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster"}) / sum(kube_node_status_allocatable_memory_bytes{cluster="$cluster"}) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | | kube_node_status_allocatable_memory_bytes | kube-state-metrics |
| Node Count | count(kube_node_info{cluster="$cluster"}) | kube_node_info | kube-state-metrics |
| Pod Count | count(kube_pod_info{cluster="$cluster"}) | kube_pod_info | kube-state-metrics |
| Node Request CPU Average Percent | avg(sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"})by (node)/sum(kube_node_status_capacity_cpu_cores{cluster="$cluster"})by(node)) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | kube_node_status_capacity_cpu_cores | kube-state-metrics |
| Node Request Memory Average Percent | avg(sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) by (node)/sum(kube_node_status_capacity_memory_bytes{cluster="$cluster"})by(node)) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | kube_node_status_capacity_memory_bytes | kube-state-metrics |
| API Server Success Request Percent | sum(irate(apiserver_request_total{cluster="$cluster",code=~"20.*",verb=~"GET\|LIST"}[5m]))/sum(irate(apiserver_request_total{cluster="$cluster",verb=~"GET\|LIST"}[5m])) | apiserver_request_total | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Namespace Overview | count(kube_pod_info{cluster="$cluster"}) by (namespace) | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | count(kube_service_info{cluster="$cluster"}) by(namespace) | kube_service_info | kube-state-metrics |
| | count(kube_pod_container_info{cluster="$cluster"}) by(namespace) | kube_pod_container_info | kube-state-metrics |
| | count(kube_configmap_info{cluster="$cluster"}) by(namespace) | kube_configmap_info | kube-state-metrics |
| | count(kube_secret_info{cluster="$cluster"}) by(namespace) | kube_secret_info | kube-state-metrics |
| | count(kube_deployment_created{cluster="$cluster"}) by (namespace) | kube_deployment_created | kube-state-metrics |
| | count(kube_statefulset_created{cluster="$cluster"}) by (namespace) | kube_statefulset_created | kube-state-metrics |
| | count(kube_job_created{cluster="$cluster"}) by (namespace) | kube_job_created | kube-state-metrics |
| | count(kube_cronjob_created{cluster="$cluster"}) by (namespace) | kube_cronjob_created | kube-state-metrics |
| | count(kube_pod_status_ready{cluster="$cluster",condition="false"}==1) by(namespace) - (count(kube_pod_status_phase{cluster="$cluster",phase="Succeeded"}==1) by(namespace) or vector(0)) or count(kube_pod_status_ready{cluster="$cluster",condition="false"}==1) by(namespace) | kube_pod_status_ready | kube-state-metrics |
| | | kube_pod_status_phase | kube-state-metrics |
| | | kube_pod_status_ready | kube-state-metrics |
| | count(kube_deployment_status_replicas_ready{cluster="$cluster"} <kube_deployment_spec_replicas{cluster="$cluster"}) by (namespace) | kube_deployment_status_replicas_ready | kube-state-metrics |
| | | kube_deployment_spec_replicas | kube-state-metrics |
| | count(kube_statefulset_status_replicas_ready{cluster="$cluster"} <kube_statefulset_replicas{cluster="$cluster"}) by (namespace) | kube_statefulset_status_replicas_ready | kube-state-metrics |
| | | kube_statefulset_replicas | kube-state-metrics |
| | count(kube_daemonset_status_number_unavailable{cluster="$cluster"}>0)by(namespace) | kube_daemonset_status_number_unavailable | kube-state-metrics |
| | count(kube_job_status_failed{cluster="$cluster"} == 1) by (namespace) | kube_job_status_failed | kube-state-metrics |
| | count(kube_daemonset_created{cluster="$cluster"}) by (namespace) | kube_daemonset_created | kube-state-metrics |
| | count(kube_persistentvolumeclaim_info{cluster="$cluster"}) by (namespace) | kube_persistentvolumeclaim_info | kube-state-metrics |
| CPU Usage | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", container!="POD", container!=""}) by (namespace) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| CPU Quota | sum(kube_pod_owner{cluster="$cluster"}) by (namespace) | kube_pod_owner | kube-state-metrics |

| | | | |
|---|---|---|---|
| | count(avg(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster"}) by (workload, namespace)) by (namespace) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", container!="POD", container!=""}) by (namespace) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"}) by (namespace) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", container!="POD", container!=""}) by (namespace) / sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"}) by (namespace) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster"}) by (namespace) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", container!="POD", container!=""}) by (namespace) / sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster"}) by (namespace) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| Memory Usage (working_set) | sum(container_memory_working_set_bytes{cluster="$cluster", container!="", container!="POD"}) by (namespace) | container_memory_working_set_bytes | cadvisor |
| Memory Requests | sum(kube_pod_owner{cluster="$cluster"}) by (namespace) | kube_pod_owner | kube-state-metrics |
| | count(avg(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster"}) by (workload, namespace)) by (namespace) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(container_memory_rss{cluster="$cluster", container!="", container!="POD"}) by (namespace) | container_memory_rss | cadvisor |
| | sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) by (namespace) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | sum(container_memory_rss{cluster="$cluster", container!="", container!="POD"}) by (namespace) / sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) by (namespace) | container_memory_rss | cadvisor |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster"}) by (namespace) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | sum(container_memory_rss{cluster="$cluster", container!="", container!="POD"}) by | container_memory_rss | cadvisor |

| | | | |
|---|---|---|---|
| | (namespace) / sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster"}) by (namespace) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Node Memory Usage (Top 10) | sum(label_replace(topk(10, 1-(node_memory_MemAvailable_bytes{cluster="$cluster"} / node_memory_MemTotal_bytes{cluster="$cluster"})), "node_ip", "$1", "instance", "(.*)"))by(node_ip) | node_memory_MemAvailable_bytes | node-exporter |
| | | node_memory_MemTotal_bytes | node-exporter |
| Node CPU Usage (Top 10) | topk(10, sum(label_replace(1 - sum(rate(node_cpu_seconds_total{cluster="$cluster",mode="idle"}[1m])) by (instance) / sum(rate(node_cpu_seconds_total{cluster="$cluster"}[1m])) by (instance),"host_ip","$1","instance","(.*)"))by(host_ip)) | node_cpu_seconds_total | node-exporter |
| | | node_cpu_seconds_total | node-exporter |
| Node Disk Usage (Top 10) | topk(10, sum(label_replace(1-node_filesystem_free_bytes{cluster="$cluster",mountpoint="/"}/node_filesystem_size_bytes{cluster="$cluster",mountpoint="/",fstype!="rootfs"},"host_ip","$1","instance","(.*)"))by(host_ip)) | node_filesystem_free_bytes | node-exporter |
| Node Network In (Top 10) | topk(10, sum(label_replace(max(irate(node_network_receive_bytes_total{cluster="$cluster"}[1m])) by (instance),"host_ip","$1","instance","(.*)"))by(host_ip)) | node_network_receive_bytes_total | node-exporter |
| Node Network Out (Top 10) | topk(10, sum(label_replace(max(irate(node_network_transmit_bytes_total{cluster="$cluster"}[1m])) by (instance),"host_ip","$1","instance","(.*)"))by(host_ip)) | node_network_transmit_bytes_total | node-exporter |
| Node Sockets Count(Top 10) | topk(10, sum(label_replace(max(node_sockstat_TCP_alloc{cluster="$cluster"}) by (instance),"host_ip","$1","instance","(.*)"))by(host_ip)) | node_sockstat_TCP_alloc | node-exporter |
| Container Memory Usage(Top10) | topk(10, sum (container_memory_working_set_bytes{cluster="$cluster",container!="",container!="POD"}) by (container)) | container_memory_working_set_bytes | cadvisor |
| Container Memory Usage/Limit(Top10) | topk(10, avg(container_memory_working_set_bytes{cluster="$cluster",container!=""}/(container_spec_memory_limit_bytes{cluster="$cluster"}!=0)) by (container, pod, namespace)) | container_memory_working_set_bytes | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| Container CPU Usage(Top10) | topk(10, sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",container!="",container!="POD"}[2m])) by (container)) | container_cpu_usage_seconds_total | cadvisor |

| | | | |
|---|---|---|---|
| Container Network | topk(10, sum(irate(container_network_receive_bytes_total{cluster="$cluster",image!="",container!="",container!="POD"}[2m])) by (pod)) | container_network_receive_bytes_total | cadvisor |
| | -topk(10, sum(irate(container_network_transmit_bytes_total{cluster="$cluster",image!="",container!="",container!="POD"}[2m])) by (pod)) | container_network_transmit_bytes_total | cadvisor |
| Container Memory Usage/Limit (Top 10) | topk(10, avg(container_memory_working_set_bytes{cluster="$cluster",container!=""}/(container_spec_memory_limit_bytes{cluster="$cluster"}!=0)) by (container, pod, namespace)) | container_memory_working_set_bytes | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| Container CPU Usage (Top 10) | topk(10, sum(irate(container_cpu_usage_seconds_total{cluster="$cluster",container!="",container!="POD"}[1m])) by (container,pod,namespace)or on() vector(0)) | container_cpu_usage_seconds_total | cadvisor |
| Container Socket Count(Top 10) | topk(10, sum(container_sockets{cluster="$cluster", container!=""}) by (container,pod,namespace)or on() vector(0)) | container_sockets | cadvisor |

## 集群 Namespace 大盘

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| CPU Usage | sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$namespace",container!="",container!="POD"}[2m])) | container_cpu_usage_seconds_total | cadvisor |
| CPU Usage/Request(%) | sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$namespace",container!="",container!="POD"}[2m]))/sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace=~"$namespace", unit="core", resource="cpu"}) | container_cpu_usage_seconds_total | cadvisor |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU Usage/Limit(%) | sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$namespace",container!="",container!="POD"}[2m]))/sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namespace=~"$namespace", unit="core", resource="cpu"}) or on() vector(0) | container_cpu_usage_seconds_total | cadvisor |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| CPU Request | sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace=~"$namespace", unit="core", resource="cpu"}) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU Limit | sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namespace=~"$namespace", unit="core", resource="cpu"}) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| Cluster Available | sum(sum(kube_node_status_capacity{resource="cpu",cluster="$cluster",namespace=~"$namespace"}) by (node) + | kube_node_status_capacity | kube-state-metrics |

| | | | |
|---|---|---|---|
| | sum(kube_node_spec_unschedulable{cluster="$cluster",namespace=~"$namespace"}==0) by(node)) | kube_node_spec_unschedulable | kube-state-metrics |
| StatefulSet Created | count(kube_statefulset_created{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_statefulset_created | kube-state-metrics |
| Pod Created | count(kube_pod_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_pod_info | kube-state-metrics |
| Containers | count(kube_pod_container_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_pod_container_info | kube-state-metrics |
| DaemonSet Created | count(kube_daemonset_created{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_daemonset_created | kube-state-metrics |
| Job Created | count(kube_job_info{cluster="$cluster",namespace="$namespace"})or on() vector(0) | kube_job_info | kube-state-metrics |
| Job Active | count(kube_job_status_active{cluster="$cluster",namespace="$namespace"}==1)or on() vector(0) | kube_job_status_active | kube-state-metrics |
| Cron Job Created | count(kube_cronjob_created{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_cronjob_created | kube-state-metrics |
| Cron Job Active | count(kube_cronjob_status_active{cluster="$cluster",namespace="$namespace"}==1) or on() vector(0) | kube_cronjob_status_active | kube-state-metrics |
| Unbound PVC | count(kube_persistentvolumeclaim_status_phase{phase!="Bound", cluster="$cluster",namespace="$namespace"}==1) or on() vector(0) | kube_persistentvolumeclaim_status_phase | kube-state-metrics |
| PersistentVolumeClaim Created | count(kube_persistentvolumeclaim_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_persistentvolumeclaim_info | kube-state-metrics |
| Service Created | count(kube_service_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_service_info | kube-state-metrics |
| LoadBalancer Created | count(kube_service_spec_type{type="LoadBalancer", cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_service_spec_type | kube-state-metrics |
| Ingress Created | count(kube_ingress_info{cluster="$cluster",namespace="$namespace"})or on() vector(0) | kube_ingress_info | kube-state-metrics |
| ConfigMap Created | count(kube_configmap_info{cluster="$cluster",namespace="$namespace"}) | kube_configmap_info | kube-state-metrics |
| Secret Created | count(kube_secret_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_secret_info | kube-state-metrics |
| PVC Storage Requests Total | sum(kube_persistentvolumeclaim_resource_requests_storage_bytes{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_persistentvolumeclaim_resource_requests_storage_bytes | kube-state-metrics |
| Pod NotReady | count(kube_pod_status_ready{condition="false", cluster="$cluster",namespace="$namespace" | kube_pod_status_ready | kube-state-metrics |

| | | | |
|---|---|---|---|
| | }==1) by(namespace) - (count(kube_pod_status_phase{phase="Succeeded", cluster="$cluster",namespace="$namespace"}==1) by(namespace) or vector(0)) or count(kube_pod_status_ready{condition="false", cluster="$cluster",namespace="$namespace"}==1) by(namespace) | | |
| | | kube_pod_status_phase | kube-state-metrics |
| | | kube_pod_status_ready | kube-state-metrics |
| Pod UnSchedulable | count(kube_pod_status_unschedulable{cluster="$cluster",namespace="$namespace"}) or on() vector(0) | kube_pod_status_unschedulable | kube-state-metrics |
| Deployment NotReady | count(sum(kube_deployment_status_replicas_ready{cluster="$cluster",namespace="$namespace"}) by (deployment) <sum(kube_deployment_spec_replicas{cluster="$cluster",namespace="$namespace"}) by (deployment)) or on() vector(0) | kube_deployment_status_replicas_ready | kube-state-metrics |
| | | kube_deployment_spec_replicas | kube-state-metrics |
| Daemonset NotReady | count(kube_daemonset_status_number_unavailable{cluster="$cluster",namespace="$namespace"}>0) or on() vector(0) | kube_daemonset_status_number_unavailable | kube-state-metrics |
| Job Failed | count(kube_job_status_failed{cluster="$cluster",namespace="$namespace"} == 1) | kube_job_status_failed | kube-state-metrics |
| CPU Usage | sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$namespace",container!="",container!="POD"}[2m])) or on() vector(0) | container_cpu_usage_seconds_total | cadvisor |
| CPU Quota | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"}) by (pod) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}) by (pod) / sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}) by (pod) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}) by (pod) / sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| Memory Usage | sum(container_memory_working_set_bytes{cluster="$cluster",namespace=~"$namespace" | container_memory_working_set_bytes | cadvisor |

| | | | |
|---|---|---|---|
| | ,container!="",container!="POD"}) or on() vector(0) | | |
| Memory Usage/Request(%) | sum(container_memory_working_set_bytes{cluster="$cluster",namespace=~"$namespace",container!="",container!="POD"})/sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace=~"$namespace", unit="byte", resource="memory"}) or on() vector(0) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Memory Usage/Limit(%) | sum(container_memory_working_set_bytes{cluster="$cluster",namespace=~"$namespace",container!="",container!="POD"})/sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespace=~"$namespace", unit="byte", resource="memory"}) or on() vector(0) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Memory Request | sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace=~"$namespace", unit="byte", resource="memory"}) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Memory Limit | sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespace=~"$namespace", unit="byte", resource="memory"}) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Cluster Available | sum(sum(kube_node_status_capacity{resource="memory"}) by (node) + sum(kube_node_spec_unschedulable==0) by(node)) or on() vector(0) | kube_node_status_capacity | kube-state-metrics |
| | | kube_node_spec_unschedulable | kube-state-metrics |
| Memory Usage (w/o cache) | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}) by (pod) | container_memory_working_set_bytes | cadvisor |
| | scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="requests.memory"}) | kube_resourcequota | kube-state-metrics |
| | scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="limits.memory"}) | kube_resourcequota | kube-state-metrics |
| Memory Quota | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace",container!="", container!="POD"}) by (pod) | container_memory_working_set_bytes | cadvisor |
| | sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace",container!="", container!="POD"}) by (pod) / sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$namespace"}) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |

| | | | |
|---|---|---|---|
| | sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"}) by (pod) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace",container!="", container!="POD"}) by (pod) / sum(kube_pod_container_resource_limits_memory_bytes{namespace="$namespace"}) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | sum(container_memory_rss{cluster="$cluster", namespace="$namespace",container!="", container!="POD"}) by (pod) | container_memory_rss | cadvisor |
| | sum(container_memory_cache{cluster="$cluster", namespace="$namespace",container!="", container!="POD"}) by (pod) | container_memory_cache | cadvisor |
| | sum(container_memory_swap{cluster="$cluster", namespace="$namespace",container!="", container!="POD"}) by (pod) | container_memory_swap | cadvisor |
| Containers | group by (image, container, container_id, pod) (kube_pod_container_info{cluster="$cluster", namespace="$namespace"}) | kube_pod_container_info | kube-state-metrics |
| | group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster", namespace="$namespace"}) * on(container,pod) group_left() max by (container,pod) (kube_pod_container_status_running{cluster="$cluster",namespace="$namespace"}) | kube_pod_container_info | kube-state-metrics |
| | | kube_pod_container_status_running | kube-state-metrics |
| | group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster", namespace="$namespace"}) * on(container,pod) group_left() max by (container,pod) (kube_pod_container_status_restarts_total{cluster="$cluster",namespace="$namespace"}) | kube_pod_container_info | kube-state-metrics |
| | | kube_pod_container_status_restarts_total | kube-state-metrics |
| | group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster", namespace="$namespace"}) * on(container,pod) group_left() max(irate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[1m])) by (pod,container) | kube_pod_container_info | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster", namespace="$namespace"}) * on(container,pod) group_left() (max(irate(container_cpu_usage_seconds_total{container!="",container!="POD",cluster="$cluster",namespace="$namespace"}[1m])) by (container,pod) / (max(container_spec_cpu_quota{container!="",container!="POD",cluster="$cluster",namesp | kube_pod_container_info | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | | container_spec_cpu_quota | cadvisor |

| | | |
|---|---|---|
| ace=~"$namespace"}/100000 > 0) by (container,pod))<br>group by (container, container_id, pod)<br>(kube_pod_container_info{cluster="$cluster",<br>namespace="$namespace"}) *<br>on(container,pod) group_left() sum by<br>(container,pod)<br>(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$namespace"}) | kube_pod_container_info | kube-state-metrics |
| | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| group by (container, container_id, pod)<br>(kube_pod_container_info{cluster="$cluster",<br>namespace="$namespace"}) *<br>on(container,pod) group_left()<br>(max(irate(container_cpu_usage_seconds_total{container!="",container!="POD",cluster="$cluster",namespace="$namespace"}[1m])) by<br>(container,pod) / (max by (container,pod)<br>(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$namespace"})) | kube_pod_container_info | kube-state-metrics |
| | container_cpu_usage_seconds_total | cadvisor |
| | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| group by (container, container_id, pod)<br>(kube_pod_container_info{cluster="$cluster",<br>namespace="$namespace"}) *<br>on(container,pod) group_left() sum by<br>(container,pod)<br>(kube_pod_container_resource_limits{resource="cpu",cluster="$cluster",namespace="$namespace"}) | kube_pod_container_info | kube-state-metrics |
| | kube_pod_container_resource_limits | kube-state-metrics |
| group by (container, container_id, pod)<br>(kube_pod_container_info{cluster="$cluster",<br>namespace="$namespace"}) *<br>on(container,pod) group_left()<br>max(container_memory_working_set_bytes{container<br>!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,container) | kube_pod_container_info | kube-state-metrics |
| | container_memory_working_set_bytes | cadvisor |
| group by (container, container_id, pod)<br>(kube_pod_container_info{cluster="$cluster",<br>namespace="$namespace"}) *<br>on(container,pod) group_left()<br>(max(container_memory_working_set_bytes{container!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by<br>(pod,container) /<br>max(container_spec_memory_limit_bytes{container<br>!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod, container) < 1) | kube_pod_container_info | kube-state-metrics |
| | container_memory_working_set_bytes | cadvisor |
| | container_spec_memory_limit_bytes | cadvisor |
| group by (container, container_id, pod)<br>(kube_pod_container_info{cluster="$cluster",<br>namespace="$namespace"}) *<br>on(container,pod) group_left() sum by<br>(container,pod)<br>(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$namespace"}) | kube_pod_container_info | kube-state-metrics |
| | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |

| | | | |
|---|---|---|---|
| | group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster", namespace="$namespace"}) * on(container,pod) group_left() (max(container_memory_working_set_bytes{ container !="",container!="POD",cluster="$cluster",name space=~"$namespace"}) by (pod,container) / max by (container,pod) (kube_pod_container_resource_requests_me mory_bytes{cluster="$cluster",namespace="$ namespace"})) | kube_pod_container_info | kube-state-metrics |
| | | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |

## API Server（独立集群）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Availability > 99.000% | 1 - ( ( sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{cluster="$cluster"}[5m])) - sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_bucket{le="1", cluster="$cluster"}[5m])) ) + sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",code=~"5..", cluster="$cluster"}[5m]) or vector(0)) ) / sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",cluster="$cluster"}[5m])) | apiserver_request_duration_seconds_count | kube-apiserver |
| | | apiserver_request_duration_seconds_bucket | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| ErrorBudget > 99.000% | 100 * (1 - ( ( sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{cluster="$cluster"}[5m])) - sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_bucket{le="1", cluster="$cluster"}[5m])) ) + sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",code=~"5..", cluster="$cluster"}[5m]) or vector(0)) ) / sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",cluster="$cluster"}[5m])) -0.990000) | apiserver_request_duration_seconds_count | kube-apiserver |
| | | apiserver_request_duration_seconds_bucket | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Read Availability | 1 - ( ( sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{verb=~"LIST\| | apiserver_request_duration_seconds_count | kube-apiserver |

| | | | |
|---|---|---|---|
| | GET", cluster="$cluster"}[5m]))<br>–<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST\|GET",le="1", cluster="$cluster"}[5m]))<br>)<br>+<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET",code=~"5..", cluster="$cluster"}[5m]) or vector(0))<br>)<br>/<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET", cluster="$cluster"}[5m])) | | |
| | | apiserver_request_duration_seconds_bucket | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Read SLI – Requests | sum by (code) (rate(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET",cluster="$cluster"}[5m])) | apiserver_request_total | kube-apiserver |
| Read SLI – Errors | sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET",code=~"5..",cluster="$cluster"}[5m]))/<br>sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET",cluster="$cluster"}[5m])) | apiserver_request_total | kube-apiserver |
| Read SLI – Duration | histogram_quantile(0.99, sum by (le, resource,cluster,cluster_type) (rate(apiserver_request_duration_seconds_bucket{job="kube-apiserver",verb=~"LIST\|GET",cluster="$cluster"}[5m]))) > 0 | apiserver_request_duration_seconds_bucket | kube-apiserver |
| Write Availability | 1 – (<br>(<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_duration_seconds_count{verb=~"POST\|PUT\|PATCH\|DELETE", cluster="$cluster"}[5m]))<br>–<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_duration_seconds_bucket{verb=~"POST\|PUT\|PATCH\|DELETE",le="1", cluster="$cluster"}[5m]))<br>)<br>+<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",code=~"5..", cluster="$cluster"}[5m]) or vector(0))<br>)<br>/<br>sum by (cluster,cluster_type)<br>(increase(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE", cluster="$cluster"}[5m])) | apiserver_request_duration_seconds_count | kube-apiserver |
| | | apiserver_request_duration_seconds_bucket | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Write SLI – Requests | sum by (code) (rate(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",cluster="$cluster"}[5m])) | apiserver_request_total | kube-apiserver |
| Write SLI – Errors | sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",code=~"5..",cluster="$cluster"}[5m]))/<br>sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",cluster="$cluster"}[5m])) | apiserver_request_total | kube-apiserver |
| Write SLI – Duration | histogram_quantile(0.99, sum by (le, resource,cluster,cluster_type) (rate(apiserver_request_duration_seconds_bucket{job="kube- | apiserver_request_duration_sec | kube-apiserver |

| | | | |
|---|---|---|---|
| | apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",cluster="$cluster"}[5m])) > 0 | onds_bucket | |
| Work Queue Add Rate | sum(rate(workqueue_adds_total{job="kube-apiserver", instance=~"$instance", cluster=~"$cluster"}[5m])) by (instance, name) | workqueue_adds_total | kube-apiserver |
| Work Queue Depth | sum(rate(workqueue_depth{job="kube-apiserver", instance=~"$instance", cluster=~"$cluster"}[5m])) by (instance, name) | workqueue_depth | kube-apiserver |
| Work Queue Latency | histogram_quantile(0.99, sum(rate(workqueue_queue_duration_seconds_bucket{job="kube-apiserver", instance=~"$instance", cluster=~"$cluster"}[5m])) by (instance, name, le)) | workqueue_queue_duration_seconds_bucket | kube-apiserver |
| Memory | process_resident_memory_bytes{job="kube-apiserver",instance=~"$instance", cluster=~"$cluster"} | process_resident_memory_bytes | kube-apiserver |
| CPU usage | rate(process_cpu_seconds_total{job="kube-apiserver",instance=~"$instance", cluster=~"$cluster"}[5m]) | process_cpu_seconds_total | kube-apiserver |

## Controller Manager（独立集群）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Up | sum(up{cluster=~"$cluster",job="kube-controller-manager"}) | up | kube-controller-manager |
| Work Queue Add Rate | sum(rate(workqueue_adds_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name) | workqueue_adds_total | kube-controller-manager |
| Work Queue Depth | sum(rate(workqueue_depth{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name) | workqueue_depth | kube-controller-manager |
| Work Queue Latency | histogram_quantile(0.99, sum(rate(workqueue_queue_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name, le)) | workqueue_queue_duration_seconds_bucket | kube-controller-manager |
| Kube API Request Rate | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"5.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| Post Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance", verb="POST"}[5m])) by (verb, url, le)) | rest_client_request_duration_seconds_bucket | kube-controller-manager |
| Get Request | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~ | rest_client_request_duration_seconds_b | kube-controller- |

| | | | |
|---|---|---|---|
| Latency 99th Quantile | "$cluster",job="kube-controller-manager", instance=~"$instance", verb="GET"}[5m])) by (verb, url, le)) | ucket | manager |
| Memory | process_resident_memory_bytes{cluster=~"$cluster",job="kube-controller-manager",instance=~"$instance"} | process_resident_memory_bytes | kube-controller-manager |
| CPU usage | rate(process_cpu_seconds_total{cluster=~"$cluster",job="kube-controller-manager",instance=~"$instance"}[5m]) | process_cpu_seconds_total | kube-controller-manager |

## Scheduler（独立集群）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Up | sum(up{cluster=~"$cluster", job="kube-scheduler"}) | up | kube-scheduler |
| Kube API Request Rate | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"5.."}[5m])) | rest_client_requests_total | kube-scheduler |
| Post Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="POST"}[5m])) by (verb, url, le)) | rest_client_request_duration_seconds_bucket | kube-scheduler |
| Get Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="GET"}[5m])) by (verb, url, le)) | rest_client_request_duration_seconds_bucket | kube-scheduler |
| Memory | process_resident_memory_bytes{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance"} | process_resident_memory_bytes | kube-scheduler |
| CPU usage | rate(process_cpu_seconds_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance"}[5m]) | process_cpu_seconds_total | kube-scheduler |

## Kubelet

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Up | sum(up{cluster="$cluster", job="kubelet"}) | up | kubelet |
| Running Pods | sum(kubelet_running_pods{cluster="$cluster", job="kubelet", instance=~"$instance"}) | kubelet_running_pods | kubelet |
| Running Container | sum(kubelet_running_containers{cluster="$cluster", job="kubelet", instance=~"$instance"}) | kubelet_running_containers | kubelet |
| Actual Volume Count | sum(volume_manager_total_volumes{cluster="$cluster", job="kubelet", instance=~"$instance", state="actual_state_of_world"}) | volume_manager_total_volumes | kubelet |
| Desired Volume | sum(volume_manager_total_volumes{cluster="$cluster", job="kubelet", | volume_manager_total_volumes | kubelet |

| | | | |
|---|---|---|---|
| Count | instance=~"$instance",state="desired_state_of_world"}) | | |
| Config Error Count | sum(rate(kubelet_node_config_error{cluster="$cluster", job="kubelet", instance=~"$instance"}[5m])) | kubelet_node_config_error | kubelet |
| Operation Rate | sum(rate(kubelet_runtime_operations_total{cluster="$cluster", job="kubelet",instance=~"$instance"}[5m])) by (operation_type, instance) | kubelet_runtime_operations_total | kubelet |
| Operation Error Rate | sum(rate(kubelet_runtime_operations_errors_total{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, operation_type) | kubelet_runtime_operations_errors_total | kubelet |
| Operation duration 99th quantile | histogram_quantile(0.99, sum(rate(kubelet_runtime_operations_duration_seconds_bucket{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, operation_type, le)) | kubelet_runtime_operations_duration_seconds_bucket | kubelet |
| Pod Start Rate | sum(rate(kubelet_pod_start_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance) | kubelet_pod_start_duration_seconds_count | kubelet |
| | sum(rate(kubelet_pod_worker_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance) | kubelet_pod_worker_duration_seconds_count | kubelet |
| Pod Start Duration | histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, le)) | kubelet_pod_start_duration_seconds_count | kubelet |
| | histogram_quantile(0.99, sum(rate(kubelet_pod_worker_duration_seconds_bucket{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, le)) | kubelet_pod_worker_duration_seconds_bucket | kubelet |
| Storage Operation Rate | sum(rate(storage_operation_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, operation_name, volume_plugin) | storage_operation_duration_seconds_count | kubelet |
| Storage Operation Error Rate | sum(rate(storage_operation_errors_total{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, operation_name, volume_plugin) | storage_operation_errors_total | kubelet |
| Storage Operation Duration 99th quantile | histogram_quantile(0.99, sum(rate(storage_operation_duration_seconds_bucket{cluster="$cluster", job="kubelet", instance=~"$instance"}[5m])) by (instance, operation_name, volume_plugin, le)) | storage_operation_duration_seconds_bucket | kubelet |
| Cgroup manager operation rate | sum(rate(kubelet_cgroup_manager_duration_seconds_count{cluster="$cluster", job="kubelet", instance=~"$instance"}[5m])) by (instance, operation_type) | kubelet_cgroup_manager_duration_seconds_count | kubelet |
| Cgroup manager 99th quantile | histogram_quantile(0.99, sum(rate(kubelet_cgroup_manager_duration_seconds_bucket{cluster="$cluster", job="kubelet", instance=~"$instance"}[5m])) by (instance, operation_type, le)) | kubelet_cgroup_manager_duration_seconds_bucket | kubelet |
| PLEG relist rate | sum(rate(kubelet_pleg_relist_duration_seconds_count{cluster="$cluster", job="kubelet", instance=~"$instance"}[5m])) by (instance) | kubelet_pleg_relist_duration_seconds_count | kubelet |
| PLEG relist interval | histogram_quantile(0.99, sum(rate(kubelet_pleg_relist_interval_seconds_bucket{cluster | kubelet_pleg_relist_interval_seconds_bucket | kubelet |

| | | | |
|---|---|---|---|
| | ="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, le)) | | |
| PLEG relist duration | histogram_quantile(0.99, sum(rate(kubelet_pleg_relist_duration_seconds_bucket{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])) by (instance, le)) | kubelet_pleg_relist_duration_seconds_bucket | kubelet |
| RPC Rate | sum(rate(rest_client_requests_total{cluster="$cluster",job="kubelet", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kubelet |
| | sum(rate(rest_client_requests_total{cluster="$cluster",job="kubelet", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kubelet |
| Request duration 99th quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster="$cluster",job="kubelet", instance=~"$instance"}[5m])) by (instance, verb, url, le)) | rest_client_request_duration_seconds_bucket | kubelet |
| Memory | process_resident_memory_bytes{cluster="$cluster",job="kubelet",instance=~"$instance"} | process_resident_memory_bytes | kubelet |
| CPU usage | rate(process_cpu_seconds_total{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m]) | process_cpu_seconds_total | kubelet |
| Goroutines | go_goroutines{cluster="$cluster",job="kubelet",instance=~"$instance"} | go_goroutines | kubelet |

## Proxy（非默认安装组件）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Up | sum(up{job="kube-proxy"}) | up | kube-proxy |
| Rules Sync Rate | sum(rate(kubeproxy_sync_proxy_rules_duration_seconds_count{job="kube-proxy", instance=~"$instance"}[5m])) | kubeproxy_sync_proxy_rules_duration_seconds_count | kube-proxy |
| Rule Sync Latency 99th Quantile | histogram_quantile(0.99,rate(kubeproxy_sync_proxy_rules_duration_seconds_bucket{job="kube-proxy", instance=~"$instance"}[5m])) | kubeproxy_sync_proxy_rules_duration_seconds_bucket | kube-proxy |
| Network Programming Rate | sum(rate(kubeproxy_network_programming_duration_seconds_count{job="kube-proxy", instance=~"$instance"}[5m])) | kubeproxy_network_programming_duration_seconds_count | kube-proxy |
| Network Programming Latency 99th Quantile | histogram_quantile(0.99, sum(rate(kubeproxy_network_programming_duration_seconds_bucket{job="kube-proxy", instance=~"$instance"}[5m])) by (instance, le)) | kubeproxy_network_programming_duration_seconds_bucket | kube-proxy |
| Kube API Request Rate | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-proxy |
| | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kube-proxy |
| | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-proxy |
| | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"5.."}[5m])) | rest_client_requests_total | kube-proxy |
| Post Request Latency 99th | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{job= | rest_client_request_duration_seconds_bucket | kube-proxy |

| | | | |
|---|---|---|---|
| Quantile | "kube-proxy",instance=~"$instance",verb="POST"}[5m])) by (verb, url, le)) | | |
| Kube API Request Rate | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-proxy |
| | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kube-proxy |
| | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-proxy |
| | sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"5.."}[5m])) | rest_client_requests_total | kube-proxy |
| Post Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{job="kube-proxy",instance=~"$instance",verb="POST"}[5m])) by (verb, url, le)) | rest_client_request_duration_seconds_bucket | kube-proxy |
| Get Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{job="kube-proxy", instance=~"$instance", verb="GET"}[5m])) by (verb, url, le)) | rest_client_request_duration_seconds_bucket | kube-proxy |
| Memory | process_resident_memory_bytes{job="kube-proxy",instance=~"$instance"} | process_resident_memory_bytes | kube-proxy |
| CPU usage | rate(process_cpu_seconds_total{job="kube-proxy",instance=~"$instance"}[5m]) | process_cpu_seconds_total | kube-proxy |

## 集群节点监控详情

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| 服务器资源总览表 | node_uname_info{job=~"$job", cluster=~"$cluster"} - 0 | node_uname_info | node-exporter |
| | node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"} - 0 | node_memory_MemTotal_bytes | node-exporter |
| | count(node_cpu_seconds_total{job=~"$job",mode='system',cluster=~"$cluster"}) by (instance) | node_cpu_seconds_total | node-exporter |
| | sum(time() - node_boot_time_seconds{job=~"$job",cluster=~"$cluster"}) by(instance) | node_boot_time_seconds | node-exporter |
| | max((node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?\|xfs"}- node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?\|xfs"}) *100/(node_filesystem_avail_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?\|xfs"}+ (node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?\|xfs"}- node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?\|xfs"}))) by(instance) | node_filesystem_size_bytes | node-exporter |
| | | node_filesystem_avail_bytes | node-exporter |
| | | node_filesystem_free_bytes | node-exporter |
| | (1 - avg(irate(node_cpu_seconds_total{job=~"$job",mode="idle",cluster=~"$cluster"}[5m])) by (instance)) * 100 | node_cpu_seconds_total | node-exporter |
| | (1 - (node_memory_MemAvailable_bytes{job=~"$job",cluster=~"$cluster"} / | node_memory_MemAvailable_bytes | node-exporter |

| | | | |
|---|---|---|---|
| | (node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"}))* 100 | node_memory_MemTotal_bytes | node-exporter |
| | node_load5{job=~"$job",cluster=~"$cluster"} | node_load5 | node-exporter |
| | max(irate(node_disk_written_bytes_total{job=~"$job",cluster=~"$cluster"}[5m])) by (instance) | node_disk_written_bytes_total | node-exporter |
| | max(irate(node_network_receive_bytes_total{job=~"$job",cluster=~"$cluster"}[5m])*8) by (instance) | node_network_receive_bytes_total | node-exporter |
| | max(irate(node_network_transmit_bytes_total{job=~"$job",cluster=~"$cluster"}[5m])*8) by (instance) | node_network_transmit_bytes_total | node-exporter |
| | node_load5{job=~"$job",cluster=~"$cluster"} | node_load5 | node-exporter |
| 整体总负载与整体平均 CPU 使用率 | count(node_cpu_seconds_total{job=~"$job",cluster=~"$cluster", mode='system'}) | node_cpu_seconds_total | node-exporter |
| | sum(node_load5{job=~"$job",cluster=~"$cluster"}) | node_load5 | node-exporter |
| | avg(1 - avg(irate(node_cpu_seconds_total{job=~"$job",mode="idle",cluster=~"$cluster"}[5m])) by (instance)) * 100 | node_cpu_seconds_total | node-exporter |
| 整体总内存与整体平均内存使用率 | sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"}) | node_memory_MemTotal_bytes | node-exporter |
| | sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"} - node_memory_MemAvailable_bytes{job=~"$job",cluster=~"$cluster"}) | node_memory_MemTotal_bytes | node-exporter |
| | | node_memory_MemAvailable_bytes | node-exporter |
| | (sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"} - node_memory_MemAvailable_bytes{job=~"$job",cluster=~"$cluster"}) / sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"}))*100 | node_memory_MemTotal_bytes | node-exporter |
| | | node_memory_MemAvailable_bytes | node-exporter |
| 整体总磁盘与整体平均磁盘使用率 | sum(avg(node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance)) | node_filesystem_size_bytes | node-exporter |
| | sum(avg(node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance)) - sum(avg(node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance)) | node_filesystem_size_bytes | node-exporter |
| | | node_filesystem_free_bytes | node-exporter |
| | (sum(avg(node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance)) - sum(avg(node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance))) *100/(sum(avg(node_filesystem_avail_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance))+ (sum(avg(node_filesystem_size_bytes{job=~"$job",fstype=~"xfs\|ext.*"})by(device,instance)) - sum(avg(node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs\|ext.*"})by(device,instance)))) | node_filesystem_size_bytes | node-exporter |
| | | node_filesystem_free_bytes | node-exporter |
| | | node_filesystem_avail_bytes | node-exporter |
| 运行时间 | avg(time() - node_boot_time_seconds{instance=~"$node",cluster=~"$cl | node_boot_time_seconds | node-exporter |

| | | | |
|---|---|---|---|
| | uster"})<br>75 | | |
| CPU 核数 | count(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node", mode='system'}) | node_cpu_seconds_total | node-exporter |
| 总内存 | sum(node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"}) | node_memory_MemTotal_bytes | node-exporter |
| 总 CPU 使用率 | 100 - (avg(irate(node_cpu_seconds_total{instance=~"$node",mode="idle",cluster=~"$cluster"}[5m])) * 100) | node_cpu_seconds_total | node-exporter |
| 内存使用率 | (1 - (node_memory_MemAvailable_bytes{instance=~"$node",cluster=~"$cluster"} / (node_memory_MemTotal_bytes{instance=~"$node",cluster=~"$cluster"})))* 100 | node_memory_MemAvailable_bytes | node-exporter |
| 最大分区使用率 | (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint="$maxmount"}-node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint="$maxmount"})*100 /(node_filesystem_avail_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint="$maxmount"}+ (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint="$maxmount"}-node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint="$maxmount"})) | node_filesystem_size_bytes | node-exporter |
| | | node_filesystem_free_bytes | node-exporter |
| | | node_filesystem_avail_bytes | node-exporter |
| CPU iowait | avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="iowait"}[5m])) * 100 | node_cpu_seconds_total | node-exporter |
| 各分区可用空间 | node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}-0 | node_filesystem_size_bytes | node-exporter |
| | node_filesystem_avail_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}-0 | node_filesystem_avail_bytes | node-exporter |
| | (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}-node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}) *100/(node_filesystem_avail_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}+ (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}-node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"})) | node_filesystem_size_bytes | node-exporter |
| | | node_filesystem_free_bytes | node-exporter |
| | | node_filesystem_avail_bytes | node-exporter |
| CPU 使用率 | avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="system"}[5m])) by (instance) *100 | node_cpu_seconds_total | node-exporter |
| | avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="user"}[5m])) by (instance) *100 | node_cpu_seconds_total | node-exporter |
| | avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="iowait"}[5m])) by (instance) *100 | node_cpu_seconds_total | node-exporter |
| | (1 - avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="idle"}[5m])) by (instance))*100 | node_cpu_seconds_total | node-exporter |

| | | | |
|---|---|---|---|
| 内存信息 | node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"} | node_memory_MemTotal_bytes | node-exporter |
| | node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"} - node_memory_MemAvailable_bytes{cluster=~"$cluster",instance=~"$node"} | node_memory_MemTotal_bytes | node-exporter |
| | | node_memory_MemAvailable_bytes | node-exporter |
| | node_memory_MemAvailable_bytes{cluster=~"$cluster",instance=~"$node"} | node_memory_MemAvailable_bytes | node-exporter |
| | (1 - (node_memory_MemAvailable_bytes{cluster=~"$cluster",instance=~"$node"} / (node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"}))) * 100 | node_memory_MemAvailable_bytes | node-exporter |
| | | node_memory_MemTotal_bytes | node-exporter |
| 每秒网络带宽使用 | irate(node_network_receive_bytes_total{cluster=~"$cluster",instance=~'$node',device=~"$device"}[5m])*8 | node_network_receive_bytes_total | node-exporter |
| | irate(node_network_transmit_bytes_total{cluster=~"$cluster",instance=~'$node',device=~"$device"}[5m])*8 | node_network_transmit_bytes_total | node-exporter |
| 系统平均负载 | node_load1{cluster=~"$cluster",instance=~"$node"} | node_load1 | node-exporter |
| | node_load5{cluster=~"$cluster",instance=~"$node"} | node_load5 | node-exporter |
| | node_load15{cluster=~"$cluster",instance=~"$node"} | node_load15 | node-exporter |
| | sum(count(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node", mode='system'}) by (cpu,instance)) by(instance) | node_cpu_seconds_total | node-exporter |
| 每秒磁盘读写容量 | irate(node_disk_read_bytes_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_read_bytes_total | node-exporter |
| | irate(node_disk_written_bytes_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_written_bytes_total | node-exporter |
| 磁盘使用率 | (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}- node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}) *100/(node_filesystem_avail_bytes {cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}+ (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"}- node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*\|xfs",mountpoint !~".*pod.*"})) | node_filesystem_size_bytes | node-exporter |
| | | node_filesystem_free_bytes | node-exporter |
| | | node_filesystem_avail_bytes | node-exporter |
| | node_filesystem_files_free{cluster=~"$cluster",instance=~'$node',fstype=~"ext.?\|xfs"} / node_filesystem_files{cluster=~"$cluster",instance=~'$node',fstype=~"ext.?\|xfs"} | node_filesystem_files_free | node-exporter |
| 磁盘读写速率（IOPS） | irate(node_disk_reads_completed_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_reads_completed_total | node-exporter |
| | irate(node_disk_writes_completed_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_writes_completed_total | node-exporter |

| | | | |
|---|---|---|---|
| | node_disk_io_now{cluster=~"$cluster",instance=~"$node"} | node_disk_io_now | node-exporter |
| 每1秒内 I/O 操作耗时占比 | irate(node_disk_io_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_io_time_seconds_total | node-exporter |
| 每次 IO 读写的耗时 | irate(node_disk_read_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) / irate(node_disk_reads_completed_total{instance=~"$node"}[5m]) | node_disk_read_time_seconds_total | node-exporter |
| | | node_disk_reads_completed_total | node-exporter |
| | irate(node_disk_write_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) / irate(node_disk_writes_completed_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_write_time_seconds_total | node-exporter |
| | | node_disk_writes_completed_total | node-exporter |
| | irate(node_disk_io_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_io_time_seconds_total | node-exporter |
| | irate(node_disk_io_time_weighted_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_disk_io_time_weighted_seconds_total | node-exporter |
| 网络 Socket 连接信息 | node_netstat_Tcp_CurrEstab{cluster=~"$cluster",instance=~'$node'} | node_netstat_Tcp_CurrEstab | node-exporter |
| | node_sockstat_TCP_tw{cluster=~"$cluster",instance=~'$node'} | node_sockstat_TCP_tw | node-exporter |
| | node_sockstat_sockets_used{cluster=~"$cluster",instance=~'$node'} | node_sockstat_sockets_used | node-exporter |
| | node_sockstat_UDP_inuse{cluster=~"$cluster",instance=~'$node'} | node_sockstat_UDP_inuse | node-exporter |
| | node_sockstat_TCP_alloc{cluster=~"$cluster",instance=~'$node'} | node_sockstat_TCP_alloc | node-exporter |
| | irate(node_netstat_Tcp_PassiveOpens{cluster=~"$cluster",instance=~'$node'}[5m]) | node_netstat_Tcp_PassiveOpens | node-exporter |
| | irate(node_netstat_Tcp_ActiveOpens{cluster=~"$cluster",instance=~'$node'}[5m]) | node_netstat_Tcp_ActiveOpens | node-exporter |
| | irate(node_netstat_Tcp_InSegs{cluster=~"$cluster",instance=~'$node'}[5m]) | node_netstat_Tcp_InSegs | node-exporter |
| | irate(node_netstat_Tcp_OutSegs{cluster=~"$cluster",instance=~'$node'}[5m]) | node_netstat_Tcp_OutSegs | node-exporter |
| | irate(node_netstat_Tcp_RetransSegs{cluster=~"$cluster",instance=~'$node'}[5m]) | node_netstat_Tcp_RetransSegs | node-exporter |
| 打开的文件描述符(左 )/每秒上下文切换次数(右) | node_filefd_allocated{cluster=~"$cluster",instance=~"$node"} | node_filefd_allocated | node-exporter |
| | irate(node_context_switches_total{cluster=~"$cluster",instance=~"$node"}[5m]) | node_context_switches_total | node-exporter |
| | (node_filefd_allocated{cluster=~"$cluster",instance=~"$node"}/node_filefd_maximum{cluster=~"$cluster",instance=~"$node"}) *100 | node_filefd_allocated | node-exporter |
| | | node_filefd_maximum | node-exporter |

## 节点 Pod 监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
| --- | --- | --- | --- |
| Pods | count(kube_pod_info{node=~"$node"}) | kube_pod_info | kube-state-metrics |
| Pod Request Memory | sum(kube_pod_container_resource_requests_memory_bytes{node=~"$node"})by(node) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Pod Request CPU Cores | sum(kube_pod_container_resource_requests_cpu_cores{node=~"$node"})by(node) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU Usage | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", node=~"$node", container!="POD", container!=""}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| CPU Quota | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", node=~"$node", container!="POD", container!=""}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", node=~"$node", container!="POD", container!=""}) by (pod) / sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", node=~"$node", container!="POD", container!=""}) by (pod) / sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| Memory Usage | sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node", container!="", container!="POD"}) by (pod) | node_namespace_pod_container:container_memory_working_set_bytes | 预聚合指标 |
| Memory Quota | sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node",container!="", container!="POD"}) by (pod) | node_namespace_pod_container:container_memory_working_set_bytes | 预聚合指标 |
| | sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", node=~"$node"}) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node",container!="", container!="POD"}) by (pod) / sum(kube_pod_container_resource_requests_memory_bytes{node=~"$node"}) by (pod) | node_namespace_pod_container:container_memory_working_set_bytes | 预聚合指标 |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |

| | | | |
|---|---|---|---|
| | sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", node=~"$node"}) by (pod) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node",container!="", container!="POD"}) by (pod) / sum(kube_pod_container_resource_limits_memory_bytes{node=~"$node"}) by (pod) | node_namespace_pod_container:container_memory_working_set_bytes | 预聚合指标 |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Pod List | group (kube_pod_info{host_ip="$node"})by(created_by_kind, created_by_name,host_network,pod_ip,pod,priority_class,namespace) | kube_pod_info | kube-state-metrics |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(kube_pod_status_phase{}==1) by (pod, phase) | kube_pod_info | kube-state-metrics |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() sum(container_memory_working_set_bytes) by (pod) | kube_pod_info | kube-state-metrics |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() sum(rate(container_cpu_usage_seconds_total{image!=""}[5m])) by (pod) | kube_pod_info | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(time()-kube_pod_start_time) by (pod) | kube_pod_info | kube-state-metrics |
| | | kube_pod_start_time | kube-state-metrics |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) max(kube_pod_status_ready{condition="true"}) by (pod) or on() vector(0) | kube_pod_info | kube-state-metrics |
| | | kube_pod_status_ready | kube-state-metrics |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_network_receive_bytes_total{image!=""}[5m])) by (pod) or on() vector(0) | kube_pod_info | kube-state-metrics |
| | | container_network_receive_bytes_total | cadvisor |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_network_transmit_bytes_total{image!=""}[5m])) by (pod) or on() vector(0) | kube_pod_info | kube-state-metrics |
| | | container_network_transmit_bytes_total | cadvisor |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_fs_reads_bytes_total{container!="POD", container!=""}[5m])) by (pod) or on() vector(0) | kube_pod_info | kube-state-metrics |
| | | container_fs_reads_bytes_total | cadvisor |
| | min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_fs_writes_bytes_total{co | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | ntainer!="POD", container!=""}[5m])) by (pod) or on() vector(0) | container_fs_writes_bytes_total | cadvisor |

## 工作负载监控概览

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| CPU Usage | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="requests.cpu"}) | kube_resourcequota | kube-state-metrics |
| | scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="limits.cpu"}) | kube_resourcequota | kube-state-metrics |
| CPU Quota | count(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relab | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | el{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type)<br>/sum(<br>kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | | |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type)<br>/sum(<br>kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Usage | sum(<br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="requests.memory"}) | kube_resourcequota | kube-state-metrics |
| | scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="limits.memory"}) | kube_resourcequota | kube-state-metrics |

| | | | |
|---|---|---|---|
| Memory Quota | count(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type)<br>/sum(<br><br>kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | workload_type="$type"}<br>sum(<br>) by (workload, workload_type)<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type)<br>/sum(<br><br>kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}<br>) by (workload, workload_type) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标<br>Deployment |

## Deployment

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Age | time() - max(kube_deployment_created{cluster="$cluster",namespace="$namespace",deployment="$workload"}) | kube_deployment_created | kube-state-metrics |
| Replicas(Pods)-Request | max(kube_deployment_spec_replicas{deployment="$workload",cluster="$cluster",namespace="$namespace"}) | kube_deployment_spec_replicas | kube-state-metrics |
| Replicas(Pods)-Ready | max(kube_deployment_status_replicas_ready{deployment="$workload",cluster="$cluster",namespace="$namespace"}) | kube_deployment_status_replicas_ready | kube-state-metrics |
| Replica Trend | max(kube_deployment_spec_replicas{deployment="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_deployment_spec_replicas | kube-state-metrics |
| | max(kube_deployment_status_replicas{deployment="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_deployment_status_replicas | kube-state-metrics |
| | min(kube_deployment_status_replicas_ready{deployment="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_deployment_status_replicas_ready | kube-state-metrics |
| | min(kube_deployment_status_replicas_available{deployment="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_deployment_status_replicas_available | kube-state-metrics |
| | min(kube_deployment_status_replicas_updated{deployment="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_deployment_status_replicas_updated | kube-state-metrics |
| | min(kube_deployment_status_replicas_unavailable{deployment="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_deployment_status_replicas_unavailable | kube-state-metrics |
| CPU Usage | sum( | node_namespace_pod_container:cont | 预聚合指标 |

| | | | |
|---|---|---|---|
| | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | ainer_cpu_usage_seconds_total:sum_rate | |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| CPU Quota | sum( node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum( kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum( node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) /sum( kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum( kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | workload_type="deployment"}<br>) by (pod)<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"}<br>) by (pod)<br>/sum(<br><br>kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"}<br>) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| CPU Limit-Total | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(kube_pod_container_resource_limits_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| CPU Request-Total | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(kube_pod_container_resource_requests_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU Info | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node), | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | "replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod, container) | kube_replicaset_owner | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | max(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}) by (pod, container))by(container) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | max(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}) by (pod, container))by(container) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| CPU Usage/Limit (%) | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"} | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |

| | | | |
|---|---|---|---|
| | [5m])) by (pod, container) / max by(container, pod) (kube_pod_container_resource_limits_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| CPU Usage/Request(%) | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod, container) / max by(container, pod) (kube_pod_container_resource_requests_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU User Time(%) | avg(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>(max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod,container) / max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])+rate(container_cpu_system_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod,container))) by (pod,container) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | | container_cpu_usage_seconds_total | cadvisor |
| | | container_cpu_usage_seconds_total | cadvisor |
| Memory Usage | sum(<br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"}<br>) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Quota | sum(<br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod) | container_memory_working_set_bytes | cadvisor |

| | | |
|---|---|---|
| group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum( kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum( container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) /sum( kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | container_memory_working_set_bytes | cadvisor |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum( kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum( container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) /sum( kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="deployment"} ) by (pod) | container_memory_working_set_bytes | cadvisor |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | $cluster", namespace="$namespace", workload="$workload", workload_type="deployment"}<br>) by (pod) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Limit-Total | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(container_spec_memory_limit_bytes{cluster="$cluster",namespace="$namespace",container!=""}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_spec_memory_limit_bytes | cadvisor |
| Memory Request-Total | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(kube_pod_container_resource_requests_memory_bytes{resource="memory",<br>cluster="$cluster",namespace="$namespace"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Memory Info | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right() max by(container, pod)<br>(container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", container!="", image!="",<br>container!="POD"}) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_memory_working_set_bytes | cadvisor |
| | max(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node), | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | "replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right() max by(container, pod)<br>(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$namespace"}))by(container) | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | max(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right() max by(container, pod)<br>(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespace="$namespace"}))by(container) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Memory Usage/Limit (%) | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right() max by(container, pod)<br>(container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", container!="", image!="", container!="POD"})/max by(container, pod)<br>(kube_pod_container_resource_limits_memory_bytes{resource="memory", cluster="$cluster",namespace="$namespace"}) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Memory Usage/Request(%) | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right() max by(container, pod)<br>(container_memory_working_set_bytes{cluster="$cluster",na | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_memory_working_set_bytes | cadvisor |

| | | | |
|---|---|---|---|
| | mespace="$namespace", container!="", image!="", container!="POD"})/max by(container, pod) (kube_pod_container_resource_requests_memory_bytes{resource="memory", cluster="$cluster",namespace="$namespace"}) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Sockets | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(container_sockets{cluster="$cluster",namespace="$namespace",container!=""}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_sockets | cadvisor |
| Network In | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(rate(container_network_receive_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_network_receive_bytes_total | cadvisor |
| Network Out | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>sum(rate(container_network_transmit_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_network_transmit_bytes_total | cadvisor |
| Network Errors | sum(label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset", | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |

| | | | |
|---|---|---|---|
| | "$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>(sum(container_network_receive_errors_total{cluster="$cluster",namespace="$namespace"}) by (pod) +<br>sum(container_network_transmit_errors_total{cluster="$cluster",namespace="$namespace"}) by (pod))) | container_network_receive_errors_total | cadvisor |
| | | container_network_transmit_errors_total | cadvisor |
| Network IO | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_network_receive_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_network_receive_bytes_total | cadvisor |
| | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_network_transmit_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_network_transmit_bytes_total | cadvisor |
| File System Read | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_fs_reads_bytes_total{cluster="$cluster",namespace="$namespace", container!="POD", container!=""}[5m])) by (pod,container) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_fs_reads_bytes_total | cadvisor |

| | | | |
|---|---|---|---|
| File System Write | label_replace(<br><br>max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="ReplicaSet", pod_ip!=""}) by (created_by_name, uid, pod, pod_ip, node),<br>"replicaset",<br>"$1",<br>"created_by_name",<br>"(.+)"<br>) * on(replicaset) group_left()<br>max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="Deployment",owner_name="$workload"}) by (replicaset)<br>* on(pod) group_right()<br>max(rate(container_fs_writes_bytes_total{cluster="$cluster",namespace="$namespace", container!="POD", container!=""}[5m])) by (pod,container) | kube_pod_info | kube-state-metrics |
| | | kube_replicaset_owner | kube-state-metrics |
| | | container_fs_writes_bytes_total | cadvisor |

## StatefulSet

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Generation | max(kube_statefulset_metadata_generation{cluster="$cluster",namespace="$namespace", statefulset="$workload"}) | kube_statefulset_metadata_generation | kube-state-metrics |
| Replicas(Pods)-Request | max(kube_statefulset_replicas{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) | kube_statefulset_replicas | kube-state-metrics |
| Replicas(Pods)-Ready | max(kube_statefulset_status_replicas_ready{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) | kube_statefulset_status_replicas_ready | kube-state-metrics |
| Age | time() - max(kube_statefulset_created{cluster="$cluster",namespace="$namespace",statefulset="$workload"}) | kube_statefulset_created | kube-state-metrics |
| Replica Trend | max(kube_statefulset_replicas{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_statefulset_replicas | kube-state-metrics |
| | max(kube_statefulset_status_replicas{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_statefulset_status_replicas | kube-state-metrics |
| | min(kube_statefulset_status_replicas_ready{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_statefulset_status_replicas_ready | kube-state-metrics |
| | min(kube_statefulset_status_replicas_available{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_statefulset_status_replicas_available | kube-state-metrics |
| | min(kube_statefulset_status_replicas_updated{statefulset="$workload",cluster="$cluster",namespace="$namespace"}) without (instance, pod) | kube_statefulset_status_replicas_updated | kube-state-metrics |
| CPU Usage | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |

| | | | |
|---|---|---|---|
| | group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"} ) by (pod) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| CPU Quota | sum( node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"} ) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum( kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"} ) by (pod) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum( node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"} ) by (pod) /sum( kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"} ) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum( kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"} ) by (pod) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}<br>) by (pod)<br>/sum(<br><br>kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}<br>) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| CPU Limit-Total | sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet", pod_ip!="", created_by_name="$workload"}) by (pod)<br>* on(pod) group_right()<br>sum(kube_pod_container_resource_limits_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| CPU Request-Total | sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet", pod_ip!="", created_by_name="$workload"}) by (pod)<br>* on(pod) group_right()<br>sum(kube_pod_container_resource_requests_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU Info | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod)<br>* on(pod) group_right()<br>max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod, container,image) | kube_pod_info | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod)<br>* on(pod) group_right()<br>max(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namespace="$namespace"}) by (pod, container,image) | kube_pod_info | kube-state-metrics |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod)<br>* on(pod) group_right() | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | max(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$namespace"}) by (pod, container,image) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU Usage/Limit (%) | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod, container) / max by(container, pod) (kube_pod_container_resource_limits_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) | kube_pod_info | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| CPU Usage/Request(%) | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod, container) / max by(container, pod) (kube_pod_container_resource_requests_cpu_cores{resource="cpu", cluster="$cluster",namespace="$namespace"}) | kube_pod_info | kube-state-metrics |
| | | container_cpu_usage_seconds_total | cadvisor |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| CPU User Time(%) | avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet", pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() (max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod, container,image) / max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])+rate(container_cpu_system_seconds_total{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}[5m])) by (pod,container,image))) by (pod,container,image) | kube_pod_info | kube-state-metrics |
| | | container_cpu_user_seconds_total | cadvisor |
| | | container_cpu_user_seconds_total | cadvisor |
| | | container_cpu_system_seconds_total | cadvisor |
| Memory Usage | sum( container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Quota | sum( container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) | container_memory_working_set_bytes | cadvisor |

| | | |
|---|---|---|
| namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum(<br>kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum(<br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)<br>/sum(<br>kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) | container_memory_working_set_bytes | cadvisor |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum(<br>kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| sum(<br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)<br>/sum( | container_memory_working_set_bytes | cadvisor |
| | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Limit-Total | sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet", pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(container_spec_memory_limit_bytes{cluster="$cluster",namespace="$namespace",container!="",container!="POD"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | container_spec_memory_limit_bytes | cadvisor |
| Memory Request-Total | sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet", pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(kube_pod_container_resource_requests_memory_bytes{resource="memory", cluster="$cluster",namespace="$namespace"}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Memory Info | avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet", pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod, image) (container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", container!="", image!="", container!="POD"}))by (container, pod, image) | kube_pod_info | kube-state-metrics |
| | | container_memory_working_set_bytes | cadvisor |
| | max(avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod, image) (kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$namespace"}))by (container, pod))by(container) | kube_pod_info | kube-state-metrics |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | max(avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod) (kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespace="$namespace"}))by (container, pod))by(container) | kube_pod_info | kube-state-metrics |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Memory Usage/Limit(%) | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", container!="", image!="", container!="POD"})/max by(container, pod) (kube_pod_container_resource_limits_memory_bytes{resource="memory", cluster="$cluster",namespace="$namespace"}) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Memory Usage/Request(%) | group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", container!="", image!="", container!="POD"})/max by(container, pod) (kube_pod_container_resource_requests_memory_bytes{resource="memory", cluster="$cluster",namespace="$namespace"}) | kube_pod_info | kube-state-metrics |
| | | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Sockets | sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(container_sockets{cluster="$cluster",namespace="$namespace",container!=""}) by (pod)) | kube_pod_info | kube-state-metrics |
| | | container_sockets | cadvisor |
| Network In | sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(rate(container_network_receive_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod)) | kube_pod_info | kube-state-metrics |
| | | container_network_receive_bytes_total | cadvisor |
| Network Out | sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(rate(container_network_transmit_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod)) | kube_pod_info | kube-state-metrics |
| | | container_network_transmit_bytes_total | cadvisor |
| Network Errors | sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() (sum(container_network_receive_errors_total{cluster="$cluster",namespace="$namespace"}) by (pod) + sum(container_network_transmit_errors_total{cluster="$cluster",namespace="$namespace"}) by (pod))) | kube_pod_info | kube-state-metrics |
| | | container_network_receive_errors_total | cadvisor |
| | | container_network_transmit_errors_total | cadvisor |
| Network IO | sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_network_receive_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod) | kube_pod_info | kube-state-metrics |
| | | container_network_receive_bytes_total | cadvisor |
| | – sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="StatefulSet",pod_ip!="", created_by_name="$workload"}) by (pod) | kube_pod_info | kube-state-metrics |

| | | | |
|---|---|---|---|
| | * on(pod) group_right() max(rate(container_network_transmit_bytes_total{cluster="$cluster",namespace="$namespace"}[5m])) by (pod) | container_network_transmit_bytes_total | cadvisor |

## DaemonSet

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| CPU Usage | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}<br>) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| CPU Quota | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}<br>) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}<br>) by (pod) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}<br>) by (pod)<br>/sum(<br><br>kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |

| | | | |
|---|---|---|---|
| | namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster",<br>namespace="$namespace", container!="POD",<br>container!=""}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod)<br>/sum(<br><br>kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Usage | sum(<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="",<br>container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Memory Quota | sum(<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="",<br>container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | sum(<br>kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod)<br>/sum(<br><br>kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | container_memory_working_set_bytes | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br>kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sum(<br><br>container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod)<br>/sum(<br><br>kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"}<br>* on(namespace,pod)<br>group_left(workload, workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{<br>cluster="$cluster", namespace="$namespace",<br>workload="$workload", workload_type="daemonset"}<br>) by (pod) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

## 集群 Pod 监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
| --- | --- | --- | --- |
| Age | time() - max(kube_pod_created{pod=~"$pod",cluster="$cluster", namespace="$namespace"}) | kube_pod_created | kube-state-metrics |
| Restart Count-Last 1 Hour | ceil(sum (increase(kube_pod_container_status_restarts_total{pod=~"$pod",cluster="$cluster",namespace="$namespace"}[1h]))) | kube_pod_container_status_restarts_total | kube-state-metrics |
| Requests-CPU | sum(kube_pod_container_resource_requests_cpu_cores{pod=~"$pod"}) or vector(0) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| Requests-Memory | sum(kube_pod_container_resource_requests_memory_bytes{pod=~"$pod"}) or vector(0) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| Limits-CPU | sum(kube_pod_container_resource_limits_cpu_cores{pod=~"$pod"}) or vector(0) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| Limits-Memory | sum(kube_pod_container_resource_limits_memory_bytes{pod=~"$pod"}) or vector(0) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Containers | group by (image, container,pod) (kube_pod_container_info{cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_info | kube-state-metrics |
| | sum by (container,pod) (kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum by (container,pod) (kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | max by (container,pod) (kube_pod_container_status_running{cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_status_running | kube-state-metrics |
| | sum by (container,pod) (kube_pod_container_resource_limits{resource="cpu",cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_resource_limits | kube-state-metrics |
| | sum by (container,pod) (kube_pod_container_resource_limits{resource="memory",cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_resource_limits | kube-state-metrics |
| | max by (container,pod) (kube_pod_container_status_restarts_total{cluster="$cluster",namespace="$namespace", pod=~"$pod"}) | kube_pod_container_status_restarts_total | kube-state-metrics |
| CPU Usage (%) | max(irate(container_cpu_usage_seconds_total{pod=~"$pod",container!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}[1m])) by (container,namespace,pod) / max(container_spec_cpu_quota{pod=~"$pod",container | container_cpu_usage_seconds_total | cadvisor |

| | | | |
|---|---|---|---|
| | !="",container!="POD",cluster="$cluster",namespace=~"$namespace"}/100000) by (container,namespace,pod) or on() vector(0) | container_spec_cpu_quota | cadvisor |
| CPU Usage By Cores | max(irate(container_cpu_usage_seconds_total{pod=~"$pod",container!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}[1m])) by (pod,container,namespace)or on() vector(0) | container_cpu_usage_seconds_total | cadvisor |
| CPU Load (10s) | max(container_cpu_load_average_10s{namespace=~"$namespace", pod=~"$pod", container!="", container!="POD"} / 1000)by(pod,container) | container_cpu_load_average_10s | cadvisor |
| CPU Throttled Percent | max (rate (container_cpu_cfs_throttled_seconds_total{image!="", container!="", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (container,pod) / max (rate (container_cpu_cfs_periods_total{image!="", container!="", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (container,pod) or on() vector(0) | container_cpu_cfs_throttled_seconds_total | cadvisor |
| | | container_cpu_cfs_periods_total | cadvisor |
| CPU Quota | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", pod="$pod", container!="POD", container!=""}) by (container) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace", pod="$pod"}) by (container) | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", pod="$pod", container!="POD", container!=""}) by (container) / sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace", pod="$pod"}) by (container) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_requests_cpu_cores | kube-state-metrics |
| | sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace", pod="$pod"}) by (container) | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| | sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", pod="$pod", container!="POD", container!=""}) by (container) / sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace", pod="$pod"}) by (container) | node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate | 预聚合指标 |
| | | kube_pod_container_resource_limits_cpu_cores | kube-state-metrics |
| Memory Usage (WSS) | max(container_memory_working_set_bytes{pod=~"$pod",container!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,namespace,container) | container_memory_working_set_bytes | cadvisor |
| Memory Usage | max(container_memory_usage_bytes{pod=~"$pod",container!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,namespace,container) | container_memory_usage_bytes | cadvisor |

| | | | |
|---|---|---|---|
| Memory Usage (RSS) | max(container_memory_rss{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace,container) or on() vector(0) | container_memory_rss | cadvisor |
| Memory Cache | max(container_memory_cache{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace,container) | container_memory_cache | cadvisor |
| Usage WSS/Limit (%) | (max(container_memory_working_set_bytes{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace,container)/ max(container_spec_memory_limit_bytes{pod=~"$pod", container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace, container) * 100) <= 100 or on() vector(0) | container_memory_working_set_bytes | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| Usage/Limit (%) | (max(container_memory_usage_bytes{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace,container)/ max(container_spec_memory_limit_bytes{pod=~"$pod", container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace, container) * 100) <= 100 or on() vector(0) | container_memory_usage_bytes | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| Usage RSS/Limit (%) | (max(container_memory_rss{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace,container)/ sum(container_spec_memory_limit_bytes{pod=~"$pod", container !="",container!="POD",cluster="$cluster",namespace=~" $namespace"}) by (pod,namespace, container) * 100) <= 100 or on() vector(0) | container_memory_rss | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| Memory Failcnt | max (increase(container_memory_failcnt{cluster="$cluster", namespace=~"$namespace", pod=~"$pod", container!=""}[1m])) by (pod,container) | container_memory_failcnt | cadvisor |
| Memory Quota | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", pod="$pod", container!="POD", container!=""}) by (container) | container_memory_working_set_bytes | cadvisor |
| | sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace", pod="$pod"}) by (container) | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", pod="$pod", container!="", container!="POD"}) by (container) / sum(kube_pod_container_resource_requests_memory_bytes{namespace="$namespace", pod="$pod"}) by (container) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_container_resource_requests_memory_bytes | kube-state-metrics |
| | sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace", pod="$pod", container!=""}) by (container) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| | sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", pod="$pod", container!="", container!="POD"}) by (container) / | container_memory_working_set_bytes | cadvisor |

| | | | |
|---|---|---|---|
| | sum(kube_pod_container_resource_limits_memory_bytes{namespace="$namespace", pod="$pod"}) by (container) | kube_pod_container_resource_limits_memory_bytes | kube-state-metrics |
| Network Input | max (rate (container_network_receive_bytes_total{image!="",cluster="$cluster",namespace=~"$namespace", pod_name=~"$pod"}[1m])) by(pod) | container_network_receive_bytes_total | cadvisor |
| Network Output | max (rate (container_network_transmit_bytes_total{image!="",cluster="$cluster",namespace=~"$namespace", pod_name=~"$pod"}[1m]))by(pod) | container_network_transmit_bytes_total | cadvisor |
| Network Input Error (%) | max (increase (container_network_receive_packets_dropped_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) / max (increase (container_network_receive_packets_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) | container_network_receive_packets_dropped_total | cadvisor |
| | | container_network_receive_packets_total | cadvisor |
| | max (increase (container_network_receive_errors_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) / max (increase (container_network_receive_packets_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) | container_network_receive_errors_total | cadvisor |
| | | container_network_receive_packets_total | cadvisor |
| Network Output Error (%) | max (increase (container_network_transmit_packets_dropped_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) / max (increase (container_network_transmit_packets_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) | container_network_transmit_packets_dropped_total | cadvisor |
| | | container_network_transmit_packets_total | cadvisor |
| | max (increase (container_network_transmit_errors_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) / max (increase (container_network_receive_packets_total{id!="/", cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (pod,interface) | container_network_transmit_errors_total | cadvisor |
| | | container_network_receive_packets_total | cadvisor |
| File System Read | max (rate(container_fs_reads_bytes_total{cluster="$cluster", namespace=~"$namespace", pod=~"$pod", container!=""}[1m]))by (container,pod) | container_fs_reads_bytes_total | cadvisor |
| File System Write | max (rate(container_fs_writes_bytes_total{cluster="$cluster",namespace=~"$namespace", pod=~"$pod", container!=""}[1m])) by (container,pod) | container_fs_writes_bytes_total | cadvisor |
| Network Socket | max(container_sockets{cluster="$cluster",namespace=~"$namespace", pod=~"$pod", container!=""}) by (container,pod) | container_sockets | cadvisor |
| Process Number | count(container_processes{cluster="$cluster",namespace=~"$namespace", pod=~"$pod", container!=""}) by (container,pod) | container_processes | cadvisor |

## 集群网络监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
| --- | --- | --- | --- |
| Current Rate of Bytes Received | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_bytes_total | cadvisor |
| Current Rate of Bytes Transmitted | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_bytes_total | cadvisor |
| Current Status | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_bytes_total | cadvisor |
| | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_bytes_total | cadvisor |
| | sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_bytes_total | cadvisor |
| | sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_bytes_total | cadvisor |
| | sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_packets_total | cadvisor |
| | sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_packets_total | cadvisor |
| | sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_packets_total | cadvisor |
| | sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_packets_dropped_total | cadvisor |
| Average Rate of Bytes Received | sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_bytes_total | cadvisor |
| Average Rate of Bytes Transmitted | sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_bytes_total | cadvisor |
| Receive Bandwidth | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_bytes_total | cadvisor |
| Transmit Bandwidth | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_bytes_total | cadvisor |
| Rate of Received Packets | sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_packets_total | cadvisor |
| Rate of Transmitted | sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~".+"}[5m])) by | container_network_transmit_packets_total | cadvisor |

| | | | |
|---|---|---|---|
| Packets | (namespace)) | | |
| Rate of Received Packets Dropped | sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_receive_packets_dropped_total | cadvisor |
| Rate of Transmitted Packets Dropped | sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~".+"}[5m])) by (namespace)) | container_network_transmit_packets_dropped_total | cadvisor |
| Rate of TCP Retransmits out of all sent segments | sort_desc(sum(rate(node_netstat_Tcp_RetransSegs{cluster=~"$cluster"}[5m]) / rate(node_netstat_Tcp_OutSegs{cluster=~"$cluster"}[$interval:$resolution])) by (instance)) | node_netstat_Tcp_RetransSegs | node-exporter |
| Rate of TCP SYN Retransmits out of all retransmits | sort_desc(sum(rate(node_netstat_TcpExt_TCPSynRetrans{cluster=~"$cluster"}[$interval:$resolution]) / rate(node_netstat_Tcp_RetransSegs{cluster=~"$cluster"}[$interval:$resolution])) by (instance)) | node_netstat_TcpExt_TCPSynRetrans | node-exporter |
| | | node_netstat_Tcp_RetransSegs | node-exporter |

## 命名空间 Pods 网络监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Current Rate of Bytes Received | sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) | container_network_receive_bytes_total | cadvisor |
| Current Rate of Bytes Transmitted | sum(irate(container_network_transmit_bytes_total{namespace=~"$namespace"}[5m])) | container_network_transmit_bytes_total | cadvisor |
| Current Status | sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_receive_bytes_total | cadvisor |
| | sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_transmit_bytes_total | cadvisor |
| | sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_receive_packets_total | cadvisor |
| | sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_transmit_packets_total | cadvisor |
| | sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_receive_packets_dropped_total | cadvisor |
| | sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_transmit_packets_dropped_total | cadvisor |
| Receive Bandwidth | sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_receive_bytes_total | cadvisor |
| Transmit Bandwidth | sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"} | container_network_transmit_bytes_total | cadvisor |

| | | | |
|---|---|---|---|
| | [5m])) by (pod) | | |
| Rate of Received Packets | sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_receive_packets_total | cadvisor |
| Rate of Transmitted Packets | sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_transmit_packets_total | cadvisor |
| Rate of Received Packets Dropped | sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])) by (pod) | container_network_receive_packets_dropped_total | cadvisor |
| Rate of Transmitted Packets Dropped | sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster", namespace=~"$namespace"}[5m])) by (pod) | container_network_transmit_packets_dropped_total | cadvisor |

## 命名空间工作负载网络监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Current Rate of Bytes Received | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Current Rate of Bytes Transmitted | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Current Status | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| | sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_packets_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_packets_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_packets_dropped_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| | sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_packets_dropped_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Average Rate of Bytes Received | sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod) | container_network_receive_bytes_total | cadvisor |

| | | | |
|---|---|---|---|
| | group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Average Rate of Bytes Transmitted | sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Receive Bandwidth | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Transmit Bandwidth | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Rate of Received Packets | sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_packets_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Rate of Transmitted Packets | sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_packets_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Rate of Received Packets Dropped | sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel {cluster=~"$cluster",namespace=~"$namespace", workload=~".+", workload_type="$type"}) by (workload)) | container_network_receive_packets_dropped_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| Rate of Transmitted Packets Dropped | sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~".+", workload_type="$type"}) by (workload)) | container_network_transmit_packets_dropped_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

## Pod 网络监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Current Rate of Bytes Received | sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace",<br>pod=~"$pod"}[5m])) | container_network_receive_bytes_total | cadvisor |
| Current Rate of Bytes Transmitted | sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace",<br>pod=~"$pod"}[5m])) | container_network_transmit_bytes_total | cadvisor |
| Receive Bandwidth | sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace",<br>pod=~"$pod"}[5m])) by (pod) | container_network_receive_bytes_total | cadvisor |
| Transmit Bandwidth | sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace",<br>pod=~"$pod"}[5m])) by (pod) | container_network_transmit_bytes_total | cadvisor |
| Rate of Received Packets | sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespace",<br>pod=~"$pod"}[5m])) by (pod) | container_network_receive_packets_total | cadvisor |
| Rate of Transmitted Packets | sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespace",<br>pod=~"$pod"}[5m])) by (pod) | container_network_transmit_packets_total | cadvisor |
| Rate of Received Packets Dropped | sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace", pod=~"$pod"}[5m])) by (pod) | container_network_receive_packets_dropped_total | cadvisor |
| Rate of Transmitted Packets Dropped | sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace", pod=~"$pod"}[5m])) by (pod) | container_network_transmit_packets_dropped_total | cadvisor |

## 工作负载网络监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Current Rate of Bytes Received | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_transmit_bytes_total | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | cadvisor |
| Current Rate of Bytes Transmitted | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod) | container_network_transmit_bytes_total | cadvisor |

| | | | |
|---|---|---|---|
| | group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Average Rate of Bytes Received | sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_receive_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Average Rate of Bytes Transmitted | sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Receive Bandwidth | sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_receive_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Transmit Bandwidth | sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_transmit_bytes_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Rate of Received Packets | sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_receive_packets_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |
| Rate of Transmitted Packets | sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespace"}[5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace", workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_transmit_packets_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

| | | | |
|---|---|---|---|
| Rate of Received Packets Dropped | sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_receive_packets_dropped_total | 预聚合指标 |
| | | namespace_workload_pod:kube_pod_owner:relabel | cadvisor |
| Rate of Transmitted Packets Dropped | sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$namespace"}[5m])<br>* on (namespace,pod)<br>group_left(workload,workload_type)<br>namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespace",<br>workload=~"$workload", workload_type="$type"}) by (pod)) | container_network_transmit_packets_dropped_total | cadvisor |
| | | namespace_workload_pod:kube_pod_owner:relabel | 预聚合指标 |

## PVC 存储监控

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Volume Space Usage | (<br>sum without(instance, node)<br>(kubelet_volume_stats_capacity_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"})<br>–<br>sum without(instance, node)<br>(kubelet_volume_stats_available_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"})<br>) | kubelet_volume_stats_capacity_bytes | kubelet |
| | | kubelet_volume_stats_available_bytes | kubelet |
| | sum without(instance, node)<br>(kubelet_volume_stats_available_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}) | kubelet_volume_stats_available_bytes | kubelet |
| Volume Space Usage | (<br>kubelet_volume_stats_capacity_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}<br>–<br>kubelet_volume_stats_available_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}<br>)<br>/<br>kubelet_volume_stats_capacity_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}<br>* 100 | kubelet_volume_stats_capacity_bytes | kubelet |
| | | kubelet_volume_stats_available_bytes | kubelet |
| | | kubelet_volume_stats_capacity_bytes | kubelet |
| Volume inodes Usage | sum without(instance, node)<br>(kubelet_volume_stats_inodes_used{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}) | kubelet_volume_stats_inodes_used | kubelet |
| | (<br>sum without(instance, node)<br>(kubelet_volume_stats_inodes{cluster="$cluster", job="kubelet", | kubelet_volume_stats_inodes | kubelet |

| | | | |
|---|---|---|---|
| | namespace="$namespace", persistentvolumeclaim="$volume"})<br>–<br>sum without(instance, node)<br>(kubelet_volume_stats_inodes_used{cluster="$cluster",<br>job="kubelet", namespace="$namespace",<br>persistentvolumeclaim="$volume"})<br>) | | |
| | | kubelet_volume_stats_inodes_used | kubelet |
| Volume inodes Usage | kubelet_volume_stats_inodes_used{cluster="$cluster",<br>job="kubelet", namespace="$namespace",<br>persistentvolumeclaim="$volume"}<br>/<br>kubelet_volume_stats_inodes{cluster="$cluster", job="kubelet",<br>namespace="$namespace", persistentvolumeclaim="$volume"}<br>* 100 | kubelet_volume_stats_inodes_used | kubelet |
| | | kubelet_volume_stats_inodes | kubelet |

## Controller Manager（托管集群）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Memory Usage | pod_mem_usage{cluster="$cluster", job="kube-controller-manager",instance=~"$instance"} | pod_mem_usage | gpu 监控 |
| CPU Usage | pod_core_usage{cluster="$cluster", job="kube-controller-manager",instance=~"$instance"} | pod_core_usage | gpu 监控 |
| In Traffic | sum(rate(container_network_receive_bytes_total{cluster="$cluster", job="kube-controller-manager",instance=~"$instance"}[5m])) by(instance) | container_network_receive_bytes_total | cadvisor |
| Out Traffic | sum(rate(container_network_transmit_bytes_total{cluster="$cluster", job="kube-controller-manager",instance=~"$instance"}[5m])) by(instance) | container_network_transmit_bytes_total | cadvisor |
| Work Queue Depth | sum(rate(workqueue_depth{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name) | workqueue_depth | kubelet |
| Work Queue Add Rate | sum(rate(workqueue_adds_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name) | workqueue_adds_total | kube-controller-manager |
| Work Queue Latency | histogram_quantile(0.99, sum(rate(workqueue_queue_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name, le)) | workqueue_queue_duration_seconds_bucket | kube-controller-manager |
| Kube API Request Rate | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-controller-manager |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"5.."}[5m])) | rest_client_requests_total | kube-controller |

| | | | |
|---|---|---|---|
| | | | – manager |
| Post Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance", verb="POST"}[5m])) by (verb, le)) | rest_client_request_duration_seconds_bucket | kube-controller-manager |
| Get Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance", verb="GET"}[5m])) by (verb, le)) | rest_client_request_duration_seconds_bucket | kube-controller-manager |

## Scheduler（托管集群）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Memory Usage | pod_mem_usage{cluster="$cluster", job="kube-scheduler", instance=~"$instance"} | pod_mem_usage | gpu 监控 |
| CPU Usage | pod_core_usage{cluster="$cluster", job="kube-scheduler", instance=~"$instance"} | pod_core_usage | gpu 监控 |
| In Traffic | sum(rate(container_network_receive_bytes_total{job="kube-scheduler", cluster="$cluster",instance=~"$instance"}[5m])) by (instance) | container_network_receive_bytes_total | cadvisor |
| Out Traffic | sum(rate(container_network_transmit_bytes_total{job="kube-scheduler", cluster="$cluster",instance=~"$instance"}[5m])) by (instance) | container_network_transmit_bytes_total | cadvisor |
| Pending Pods | sum(scheduler_pending_pods{job="kube-scheduler", cluster="$cluster", instance=~"$instance"}) by (queue) | scheduler_pending_pods | kube-scheduler |
| Number of attempts to successfully schedule(P90) | histogram_quantile(0.9, sum(rate(scheduler_pod_scheduling_attempts_bucket{job="kube-scheduler", cluster="$cluster", instance=~"$instance"}[5m])) by (le,instance)) | scheduler_pod_scheduling_attempts_bucket | kube-scheduler |
| Get Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="GET"}[5m])) by (verb, le)) | rest_client_request_duration_seconds_bucket | kube-scheduler |
| Post Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="POST"}[5m])) by (verb, le)) | rest_client_request_duration_seconds_bucket | kube-scheduler |
| Kube API Request Rate | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"3.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"5.."}[5m])) | rest_client_requests_total | kube-scheduler |

## API Server（托管集群）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
| --- | --- | --- | --- |
| Memory Usage | pod_mem_usage{cluster="$cluster", job="kube-scheduler", instance=~"$instance"} | pod_mem_usage | gpu 监控 |
| CPU Usage | pod_core_usage{cluster="$cluster", job="kube-scheduler", instance=~"$instance"} | pod_core_usage | gpu 监控 |
| In Traffic | sum(rate(container_network_receive_bytes_total{job="kube-scheduler", cluster="$cluster",instance=~"$instance"}[5m])) by (instance) | container_network_receive_bytes_total | cadvisor |
| Out Traffic | sum(rate(container_network_transmit_bytes_total{job="kube-scheduler", cluster="$cluster",instance=~"$instance"}[5m])) by (instance) | container_network_transmit_bytes_total | cadvisor |
| Pending Pods | sum(scheduler_pending_pods{job="kube-scheduler", cluster="$cluster", instance=~"$instance"}) by (queue) | scheduler_pending_pods | kube-scheduler |
| | | scheduler_pod_scheduling_attempts_bucket | kube-scheduler |
| Get Request Latency 99th Quantile | histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="GET"}[5m])) by (verb, le)) | rest_client_request_duration_seconds_bucket | kube-scheduler |
| | | rest_client_request_duration_seconds_bucket | kube-scheduler |
| Kube API Request Rate | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"2.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | | rest_client_requests_total | kube-scheduler |
| | sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"4.."}[5m])) | rest_client_requests_total | kube-scheduler |
| | | rest_client_requests_total | kube-scheduler |
| Write Request/s | sum(rate(apiserver_request_count{cluster="$cluster", verb=~"POST\|PUT\|PATCH\|DELETE",instance=~"$instance"}[5m])) by (instance) or sum(rate(apiserver_request_total{cluster="$cluster", verb=~"POST\|PUT\|PATCH\|DELETE",instance=~"$instance"}[5m])) by (instance) | apiserver_request_count | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Read Request/s | sum(rate(apiserver_request_count{cluster="$cluster", verb=~"LIST\|GET",instance=~"$instance"}[5m])) by (instance) or sum(rate(apiserver_request_total{cluster="$cluster", verb=~"LIST\|GET",instance=~"$instance"}[5m])) by (instance) | apiserver_request_count | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Latency(Average) | (sum(rate(apiserver_request_duration_seconds_sum{cluster="$cluster", verb=~"GET\|LIST\|POST\|PUT\|DELETE\|PATCH", instance=~"$instance"}[5m])) by(instance)) /(sum(rate(apiserver_request_duration_seconds_count{cluster="$cluster", | apiserver_request_duration_seconds_sum | kube-apiserver |
| | | apiserver_request_duration_seconds_co | kube-apiserver |

| | | | |
|---|---|---|---|
| | verb=~"GET\|LIST\|POST\|PUT\|DELETE\|PATCH", instance=~"$instance"}[5m])) by(instance)) | unt | |
| Latency(P99) | histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb="LIST", cluster="$cluster",instance=~"$instance"}[5m])) by (le,instance)) | apiserver_request_duration_seconds_bucket | kube-apiserver |
| Current Inflight Request | sum(apiserver_current_inflight_requests{cluster="$cluster",instance=~"$instance"}) by (instance) | apiserver_current_inflight_requests | kube-apiserver |
| Self Request/s | sum by (instance) (rate(apiserver_selfrequest_total{cluster="$cluster",instance=~"$instance"}[5m])) | apiserver_selfrequest_total | kube-apiserver |
| Response Body Size(P99) | histogram_quantile(0.99, sum by (instance,le) (rate(apiserver_response_sizes_bucket{cluster="$cluster",instance=~"$instance"}[5m]))) | apiserver_response_sizes_bucket | kube-apiserver |
| Watch Events/s | sum by (instance) (rate(apiserver_watch_events_total{cluster="$cluster",instance=~"$instance"}[5m])) | apiserver_watch_events_total | kube-apiserver |
| Too Many Objects Events/s | sum(rate(list_too_many_objects_events_total{cluster="$cluster",instance=~"$instance"}[5m])) by (instance) | list_too_many_objects_events_total | kube-apiserver |
| Too Old Objects Events/s | sum(rate(watch_too_old_objects_events_total{cluster="$cluster",instance=~"$instance"}[5m])) by (instance) | watch_too_old_objects_events_total | kube-apiserver |
| Read Request/s | sum(rate(apiserver_request_count{cluster="$cluster",verb=~"GET\|LIST",instance=~"$instance",resource!=""}[5m])) by (resource) > 0 or sum(rate(apiserver_request_total{cluster="$cluster",verb=~"GET\|LIST",instance=~"$instance",resource!=""}[5m])) by (resource) > 0 | apiserver_request_count | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Write Request/s | sum(rate(apiserver_request_count{cluster="$cluster",verb=~"POST\|PUT\|PATCH\|DELETE",instance=~"$instance"}[5m])) by (resource) > 0 or sum(rate(apiserver_request_total{cluster="$cluster",verb=~"POST\|PUT\|PATCH\|DELETE",instance=~"$instance"}[5m])) by (resource) > 0 | apiserver_request_count | kube-apiserver |
| | | apiserver_request_total | kube-apiserver |
| Latency(Average) | (sum(rate(apiserver_request_duration_seconds_sum{cluster="$cluster",verb=~"GET\|LIST\|POST\|PUT\|DELETE\|PATCH",instance=~"$instance"}[5m])) by(resource)) /(sum(rate(apiserver_request_duration_seconds_count{cluster="$cluster",verb=~"GET\|LIST\|POST\|PUT\|DELETE\|PATCH",instance=~"$instance"}[5m])) by(resource)) >0 | apiserver_request_duration_seconds_sum | kube-apiserver |
| | | apiserver_request_duration_seconds_count | kube-apiserver |
| Latency(P99) | histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET\|LIST\|POST\|PUT\|DELETE\|PATCH",cluster="$cluster",instance=~"$instance"}[5m])) by (le,resource))>0 | apiserver_request_duration_seconds_bucket | kube-apiserver |

## GPU Cluster（GPU 集成）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|
| Total GPU Nodes | count(count by(Hostname) (DCGM_FI_DEV_COUNT{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"})) | DCGM_FI_DEV_COUNT | TKE GPU Exporter 集成 |

| | | | |
|---|---|---|---|
| Allocated GPUs | count(DCGM_FI_DEV_COUNT{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}) | DCGM_FI_DEV_COUNT | TKE GPU Exporter 集成 |
| | sum(max_over_time(gpu_core_allocated{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}[1m]))/100 | gpu_core_allocated | TKE GPU Exporter 集成 |
| Allocated GPU Memory Ratio | sum(gpu_mem_allocated{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}) / sum(gpu_mem_allocatable{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}) *100 | gpu_mem_allocated | TKE GPU Exporter 集成 |
| | | gpu_mem_allocatable | TKE GPU Exporter 集成 |
| Used GPU Memory Ratio | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) / sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) * 100 | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |
| Average GPU Utilization | avg(max_over_time(DCGM_FI_DEV_GPU_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) | DCGM_FI_DEV_GPU_UTIL | TKE GPU Exporter 集成 |
| GPU Memory Copy Utilization | avg(max_over_time(DCGM_FI_DEV_MEM_COPY_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) | DCGM_FI_DEV_MEM_COPY_UTIL | TKE GPU Exporter 集成 |
| The Last one XID Error | sum(DCGM_FI_DEV_XID_ERRORS{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}) | DCGM_FI_DEV_XID_ERRORS | TKE GPU Exporter 集成 |
| GPU Node Details | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,gpu) | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_GPU_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,gpu) | DCGM_FI_DEV_GPU_UTIL | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_POWER_USAGE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,gpu) | DCGM_FI_DEV_POWER_USAGE | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_GPU_TEMP{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,gpu) | DCGM_FI_DEV_GPU_TEMP | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_MEM_COPY_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,gpu) | DCGM_FI_DEV_MEM_COPY_UTIL | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,gpu) | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |

## GPU Node（GPU 集成）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
|---|---|---|---|

| | | | |
|---|---|---|---|
| GPU Utilization | avg(max_over_time(DCGM_FI_DEV_GPU_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) | DCGM_FI_DEV_GPU_UTIL | TKE GPU Exporter 集成 |
| Allocated GPU Memory | sum(max_over_time(gpu_mem_allocated{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}[1m]))/ sum(gpu_mem_allocatable{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}) *100 | gpu_mem_allocated | TKE GPU Exporter 集成 |
| | | gpu_mem_allocatable | TKE GPU Exporter 集成 |
| Used GPU Memory | avg(gpu_mem_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}) | gpu_mem_utilization_percentage | TKE GPU Exporter 集成 |
| Allocated GPUs | sum(max_over_time(gpu_core_allocated{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}[1m]))/100 | gpu_core_allocated | TKE GPU Exporter 集成 |
| | count(DCGM_FI_DEV_COUNT{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}) | DCGM_FI_DEV_COUNT | TKE GPU Exporter 集成 |
| Allocated Computing Power(Valid in GPU Sharing) | sum(gpu_power_usage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}) by (node) | gpu_power_usage | TKE GPU Exporter 集成 |
| The Last one XID Error | DCGM_FI_DEV_XID_ERRORS{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_XID_ERRORS | TKE GPU Exporter 集成 |
| GPU Utilization | DCGM_FI_DEV_GPU_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_GPU_UTIL | TKE GPU Exporter 集成 |
| GPU Memory Copy Utilization | DCGM_FI_DEV_MEM_COPY_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_MEM_COPY_UTIL | TKE GPU Exporter 集成 |
| Encoder Engine Utilization | DCGM_FI_DEV_ENC_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_ENC_UTIL | TKE GPU Exporter 集成 |
| Decoder Engine Utilization | DCGM_FI_DEV_DEC_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_DEC_UTIL | TKE GPU Exporter 集成 |
| GPU Memory Details | sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by (Hostname,UUID,gpu,modelName) | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m]))by(Hostname,UUID,gpu,modelName) | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m]))by (Hostname,UUID,gpu,modelName) / sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m])) by(Hostname,UUID,gpu,modelName) * 100 | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |

| | | | |
|---|---|---|---|
| BAR1 Used | DCGM_FI_DEV_BAR1_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_BAR1_USED | TKE GPU Exporter 集成 |
| BAR1 Total | DCGM_FI_DEV_BAR1_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_BAR1_TOTAL | TKE GPU Exporter 集成 |
| GPU Memory Used | DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| Graphics Engine Active(%) | DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 100 | DCGM_FI_PROF_GR_ENGINE_ACTIVE | TKE GPU Exporter 集成 |
| DRAM Active(%) | DCGM_FI_PROF_DRAM_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 100 | DCGM_FI_PROF_DRAM_ACTIVE | TKE GPU Exporter 集成 |
| SM Active(%) | DCGM_FI_PROF_SM_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}* 100 | DCGM_FI_PROF_SM_ACTIVE | TKE GPU Exporter 集成 |
| SM Occupancy(%) | DCGM_FI_PROF_SM_OCCUPANCY{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_PROF_SM_OCCUPANCY | TKE GPU Exporter 集成 |
| Tensor Core Engine Active(%) | DCGM_FI_PROF_PIPE_TENSOR_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 100 | DCGM_FI_PROF_PIPE_TENSOR_ACTIVE | TKE GPU Exporter 集成 |
| FP32 Engine Active(%) | DCGM_FI_PROF_PIPE_FP32_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 100 | DCGM_FI_PROF_PIPE_FP32_ACTIVE | TKE GPU Exporter 集成 |
| FP16 Engine Active(%) | DCGM_FI_PROF_PIPE_FP16_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 100 | DCGM_FI_PROF_PIPE_FP16_ACTIVE | TKE GPU Exporter 集成 |
| FP64 Engine Active(%) | DCGM_FI_PROF_PIPE_FP64_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 100 | DCGM_FI_PROF_PIPE_FP64_ACTIVE | TKE GPU Exporter 集成 |
| PCIE TX Bytes(Device to Host) | rate(DCGM_FI_PROF_PCIE_TX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m]) | DCGM_FI_PROF_PCIE_TX_BYTES | TKE GPU Exporter 集成 |
| PCIE RX Bytes(Host to Device) | rate(DCGM_FI_PROF_PCIE_RX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m]) | DCGM_FI_PROF_PCIE_RX_BYTES | TKE GPU Exporter 集成 |
| NVLINK TX Bytes | rate(DCGM_FI_PROF_NVLINK_TX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m]) | DCGM_FI_PROF_NVLINK_TX_BYTES | TKE GPU Exporter 集成 |
| NVLINK RX Bytes | rate(DCGM_FI_PROF_NVLINK_RX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"}[1m]) | DCGM_FI_PROF_NVLINK_RX_BYTES | TKE GPU Exporter 集成 |
| Power Usage(W) | DCGM_FI_DEV_POWER_USAGE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_POWER_USAGE | TKE GPU Exporter 集成 |
| Toal Energy Consumption(in J) | DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUN | DCGM_FI_DEV_TOTAL_ENERGY_CONS | TKE GPU Exporter 集 |

| | | | |
|---|---|---|---|
| | ode"} / 1000 | UMPTION | 成 |
| Memory Temperature(℃) | sum(gpu_temprature{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode"}) by (node,card) | gpu_temprature | TKE GPU Exporter 集成 |
| GPU Temperature(℃) | DCGM_FI_DEV_GPU_TEMP{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_GPU_TEMP | TKE GPU Exporter 集成 |
| SM Clock | DCGM_FI_DEV_SM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 1000* 1000 | DCGM_FI_DEV_SM_CLOCK | TKE GPU Exporter 集成 |
| Memory Clock | DCGM_FI_DEV_MEM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 1000 * 1000 | DCGM_FI_DEV_MEM_CLOCK | TKE GPU Exporter 集成 |
| APP SM Clock | DCGM_FI_DEV_APP_SM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 1000 * 1000 | DCGM_FI_DEV_APP_SM_CLOCK | TKE GPU Exporter 集成 |
| APP Memory Clock | DCGM_FI_DEV_APP_MEM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 1000 * 1000 | DCGM_FI_DEV_APP_MEM_CLOCK | TKE GPU Exporter 集成 |
| Video Clock | DCGM_FI_DEV_VIDEO_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} * 1000 * 1000 | DCGM_FI_DEV_VIDEO_CLOCK | TKE GPU Exporter 集成 |
| Clock Throttle Reasons | DCGM_FI_DEV_CLOCK_THROTTLE_REASONS{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_CLOCK_THROTTLE_REASONS | TKE GPU Exporter 集成 |
| Retired Pages(Single-bit Errors) | DCGM_FI_DEV_RETIRED_SBE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_RETIRED_SBE | TKE GPU Exporter 集成 |
| Retired Pages(Double-bit Errors) | DCGM_FI_DEV_RETIRED_DBE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_RETIRED_DBE | TKE GPU Exporter 集成 |
| Power Violation | DCGM_FI_DEV_POWER_VIOLATION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_POWER_VIOLATION | TKE GPU Exporter 集成 |
| Thermal Violation | DCGM_FI_DEV_THERMAL_VIOLATION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_THERMAL_VIOLATION | TKE GPU Exporter 集成 |
| Sync Boost Violation | DCGM_FI_DEV_SYNC_BOOST_VIOLATION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_SYNC_BOOST_VIOLATION | TKE GPU Exporter 集成 |
| Board Relability Violation | DCGM_FI_DEV_RELIABILITY_VIOLATION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_RELIABILITY_VIOLATION | TKE GPU Exporter 集成 |
| Board Limit Violation | DCGM_FI_DEV_BOARD_LIMIT_VIOLATION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_BOARD_LIMIT_VIOLATION | TKE GPU Exporter 集成 |
| Low Util Violation | DCGM_FI_DEV_LOW_UTIL_VIOLATION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode"} | DCGM_FI_DEV_LOW_UTIL_VIOLATION | TKE GPU Exporter 集成 |

## GPU Pod（GPU 集成）

| 图表名称 | 查询语句 | 使用的指标 | 配置文件 |
| --- | --- | --- | --- |
| Memory Usage | avg(sum(container_memory_rss{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod, namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_memory_rss | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | avg(sum(container_memory_working_set_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod, namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_memory_working_set_bytes | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | sum(container_memory_rss{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod,namespace) | container_memory_rss | cadvisor |
| | sum(container_memory_working_set_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod,namespace) | container_memory_working_set_bytes | cadvisor |
| Memory Percent | avg((sum(container_memory_rss{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod,namespace)/ sum(container_spec_memory_limit_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",namespace=~"$PodNamespace"}) by (pod,namespace) * 100) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{pod=~"$PodName",namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_memory_rss | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | avg((sum(container_memory_working_set_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod,namespace)/ sum(container_spec_memory_limit_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",namespace=~"$PodNamespace"}) by (pod,namespace) * 100) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{pod=~"$PodName",namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_memory_working_set_bytes | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | (sum(container_memory_rss{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}) by (pod,namespace)/ | container_memory_rss | cadvisor |

| | | | |
|---|---|---|---|
| | sum(container_spec_memory_limit_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",namespace=~"$PodNamespace"}) by (pod,namespace) * 100) | container_spec_memory_limit_bytes | cadvisor |
| | (sum(container_memory_working_set_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}) by (pod,namespace)/ sum(container_spec_memory_limit_bytes{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",namespace=~"$PodNamespace"}) by (pod,namespace) * 100) | container_memory_working_set_bytes | cadvisor |
| | | container_spec_memory_limit_bytes | cadvisor |
| CPU Usage By Cores | avg(sum(irate(container_cpu_usage_seconds_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}[1m])) by (pod,namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName",namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_cpu_usage_seconds_total | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | sum(irate(container_cpu_usage_seconds_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}[1m])) by (pod,namespace) | container_cpu_usage_seconds_total | cadvisor |
| CPU Usage Percent | avg(sum(irate(container_cpu_usage_seconds_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}[1m])) by (pod,namespace) / sum(container_spec_cpu_quota{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}/100000) by (pod,namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName",namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_cpu_usage_seconds_total | cadvisor |
| | | container_spec_cpu_quota | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | sum(irate(container_cpu_usage_seconds_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}[1m])) by (pod,namespace) / sum(container_spec_cpu_quota{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}/100000) by (pod,namespace) | container_cpu_usage_seconds_total | cadvisor |
| | | container_spec_cpu_quota | cadvisor |
| Network Bandwidth Usage | avg(sum(rate(container_network_receive_bytes_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container!="",container!="POD",instance=~"$GPUNode",namespace=~"$PodNamespace"}[1m])) by (pod, namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",namespace=~"$Pod | container_network_receive_bytes_total | cadvisor |
| | | kube_pod_owner | kube-state-metrics |

| | | | |
|---|---|---|---|
| | Namespace", pod=~"$PodName"})) by (namespace,owner_name) | | |
| | avg(sum(rate(container_network_transmit_bytes_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}[1m])) by (pod, namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",namespace=~"$PodNamespace", pod=~"$PodName"})) by (namespace,owner_name) | container_network_transmit_bytes_total | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | sum (rate (container_network_receive_bytes_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}[1m]))by (pod,namespace) | container_network_receive_bytes_total | cadvisor |
| | sum (rate (container_network_transmit_bytes_total{cluster=~"$cluster",nodepool=~"$NodePool",pod=~"$PodName",container !="",container!="POD",instance=~"$GPUNode", namespace=~"$PodNamespace"}[1m]))by (pod,namespace) | container_network_transmit_bytes_total | cadvisor |
| Network Socket | avg(sum(container_sockets{cluster=~"$cluster",nodepool=~"$NodePool",namespace=~"$PodNamespace", pod=~"$PodName", instance=~"$GPUNode", container!=""}) by (pod,namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_sockets | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | sum(container_sockets{cluster=~"$cluster",nodepool=~"$NodePool",namespace=~"$PodNamespace", pod=~"$PodName", instance=~"$GPUNode", container!=""}) by (pod,namespace) | container_sockets | cadvisor |
| File System | avg(sum (rate(container_fs_reads_bytes_total{cluster=~"$cluster",nodepool=~"$NodePool",namespace=~"$PodNamespace", pod=~"$PodName", instance=~"$GPUNode", container!=""}[1m]))by (pod,namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_fs_reads_bytes_total | cadvisor |
| | | kube_pod_owner | kube-state-metrics |
| | avg(sum (rate(container_fs_writes_bytes_total{cluster=~"$cluster", nodepool=~"$NodePool",namespace=~"$PodNamespace", pod=~"$PodName", instance=~"$GPUNode", container!=""}[1m])) by (pod,namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | container_fs_writes_bytes_total | cadvisor |

| | | | |
|---|---|---|---|
| | sum (rate(container_fs_reads_bytes_total{cluster=~"$cluster",nodepool=~"$NodePool",namespace=~"$PodNamespace", pod=~"$PodName", instance=~"$GPUNode", container!=""}[1m]))by (pod,namespace) | container_fs_reads_bytes_total | cadvisor |
| | sum (rate(container_fs_writes_bytes_total{cluster=~"$cluster", nodepool=~"$NodePool",namespace=~"$PodNamespace", pod=~"$PodName", instance=~"$GPUNode", container!=""}[1m])) by (pod,namespace) | container_fs_writes_bytes_total | cadvisor |
| Pods Average SM Utilization | avg(avg(pod_core_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",namespace=~"$PodNamespace", pod=~"$PodName"}) by (namespace, pod)* on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | pod_core_utilization_percentage | TKE GPU Exporter 集成 |
| | | kube_pod_owner | kube-state-metrics |
| | avg(pod_core_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",namespace=~"$PodNamespace", pod=~"$PodName"}) by (namespace, pod) | pod_core_utilization_percentage | TKE GPU Exporter 集成 |
| Pods GPU Memory Used Percentage | avg(sum(pod_mem_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",pod=~"$PodName", namespace=~"$PodNamespace"}) by (pod, namespace,job) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | pod_mem_utilization_percentage | TKE GPU Exporter 集成 |
| | | kube_pod_owner | kube-state-metrics |
| | sum(pod_mem_utilization_percentage{cluster=~"$cluster", nodepool=~"$NodePool",node=~"$GPUNode",pod=~"$PodName", namespace=~"$PodNamespace"}) by (pod, namespace,job) | pod_mem_utilization_percentage | TKE GPU Exporter 集成 |
| Pods Used GPU Memory | avg(sum(pod_mem_usage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",pod=~"$PodName",namespace=~"$PodNamespace"}) by (pod, namespace) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | pod_mem_usage | TKE GPU Exporter 集成 |
| | | kube_pod_owner | kube-state-metrics |
| | sum(pod_mem_usage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",pod=~"$PodName",namespace=~"$PodNamespace"}) by (pod, namespace) | pod_mem_usage | TKE GPU Exporter 集成 |
| Pods GPU Encode Utilization | avg(avg(pod_enc_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",namespace=~"$PodNamespace",pod=~"$PodName"}) by (namespace, pod) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | pod_enc_utilization_percentage | TKE GPU Exporter 集成 |
| | | kube_pod_owner | kube-state-metrics |

| | | | |
|---|---|---|---|
| | avg(pod_enc_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",namespace=~"$PodNamespace",pod=~"$PodName"}) by (namespace, pod) | pod_enc_utilization_percentage | TKE GPU Exporter 集成 |
| Pods GPU Decode Utilization | avg(avg(pod_dec_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",namespace=~"$PodNamespace",pod=~"$PodName"}) by (namespace, pod) * on(pod, namespace) group_left(owner_name) avg by (owner_name, pod, namespace) (kube_pod_owner{cluster=~"$cluster",pod=~"$PodName", namespace=~"$PodNamespace"})) by (namespace,owner_name) | pod_dec_utilization_percentage | TKE GPU Exporter 集成 |
| | | kube_pod_owner | kube-state-metrics |
| | avg(pod_dec_utilization_percentage{cluster=~"$cluster",nodepool=~"$NodePool",node=~"$GPUNode",namespace=~"$PodNamespace",pod=~"$PodName"}) by (namespace, pod) | pod_dec_utilization_percentage | TKE GPU Exporter 集成 |
| GPU Utilization | DCGM_FI_DEV_GPU_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_GPU_UTIL | TKE GPU Exporter 集成 |
| GPU Memory Copy Utilization | DCGM_FI_DEV_MEM_COPY_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_MEM_COPY_UTIL | TKE GPU Exporter 集成 |
| Encoder Engine Utilization | DCGM_FI_DEV_ENC_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_ENC_UTIL | TKE GPU Exporter 集成 |
| Decoder Engine Utilization | DCGM_FI_DEV_DEC_UTIL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_DEC_UTIL | TKE GPU Exporter 集成 |
| GPU Memory Details | sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m])) by (UUID,Hostname,modelName) | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]))by (UUID,Hostname,modelName) | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]))by (UUID,Hostname,modelName) / sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m])) by (UUID,Hostname,modelName) * 100 | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |
| GPU Memory Used | DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| GPU Memory Used Percentage | sum(max_over_time(DCGM_FI_DEV_FB_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]))by (Hostname,UUID) / sum(max_over_time(DCGM_FI_DEV_FB_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m])) by (Hostname,UUID) * 100 | DCGM_FI_DEV_FB_USED | TKE GPU Exporter 集成 |
| | | DCGM_FI_DEV_FB_TOTAL | TKE GPU Exporter 集成 |

| | | | |
|---|---|---|---|
| BAR1 Used | DCGM_FI_DEV_BAR1_USED{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_BAR1_USED | TKE GPU Exporter 集成 |
| BAR1 Total | DCGM_FI_DEV_BAR1_TOTAL{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_BAR1_TOTAL | TKE GPU Exporter 集成 |
| Graphics Engine Active(%) | DCGM_FI_PROF_GR_ENGINE_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 100 | DCGM_FI_PROF_GR_ENGINE_ACTIVE | TKE GPU Exporter 集成 |
| DRAM Active(%) | DCGM_FI_PROF_DRAM_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 100 | DCGM_FI_PROF_DRAM_ACTIVE | TKE GPU Exporter 集成 |
| SM Active(%) | DCGM_FI_PROF_SM_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}* 100 | DCGM_FI_PROF_SM_ACTIVE | TKE GPU Exporter 集成 |
| SM Occupancy(%) | DCGM_FI_PROF_SM_OCCUPANCY{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_PROF_SM_OCCUPANCY | TKE GPU Exporter 集成 |
| Tensor Core Engine Active(%) | DCGM_FI_PROF_PIPE_TENSOR_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 100 | DCGM_FI_PROF_PIPE_TENSOR_ACTIVE | TKE GPU Exporter 集成 |
| FP32 Engine Active(%) | DCGM_FI_PROF_PIPE_FP32_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 100 | DCGM_FI_PROF_PIPE_FP32_ACTIVE | TKE GPU Exporter 集成 |
| FP16 Engine Active(%) | DCGM_FI_PROF_PIPE_FP16_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 100 | DCGM_FI_PROF_PIPE_FP16_ACTIVE | TKE GPU Exporter 集成 |
| FP64 Engine Active(%) | DCGM_FI_PROF_PIPE_FP64_ACTIVE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 100 | DCGM_FI_PROF_PIPE_FP64_ACTIVE | TKE GPU Exporter 集成 |
| PCIE TX Bytes(Device to Host) | rate(DCGM_FI_PROF_PCIE_TX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]) | DCGM_FI_PROF_PCIE_TX_BYTES | TKE GPU Exporter 集成 |
| PCIE RX Bytes(Host to Device) | rate(DCGM_FI_PROF_PCIE_RX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]) | DCGM_FI_PROF_PCIE_RX_BYTES | TKE GPU Exporter 集成 |
| NVLINK TX Bytes | rate(DCGM_FI_PROF_NVLINK_TX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]) | DCGM_FI_PROF_NVLINK_TX_BYTES | TKE GPU Exporter 集成 |
| NVLINK RX Bytes | rate(DCGM_FI_PROF_NVLINK_RX_BYTES{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"}[1m]) | DCGM_FI_PROF_NVLINK_RX_BYTES | TKE GPU Exporter 集成 |
| Power Usage(W) | DCGM_FI_DEV_POWER_USAGE{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_POWER_USAGE | TKE GPU Exporter 集成 |
| Toal Energy Consumption(in J) | DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} / 1000 | DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION | TKE GPU Exporter 集成 |
| GPU Temperature(℃) | DCGM_FI_DEV_GPU_TEMP{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_GPU_TEMP | TKE GPU Exporter 集 |

| | | | |
|---|---|---|---|
| | | | 成 |
| SM Clock | DCGM_FI_DEV_SM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 1000* 1000 | DCGM_FI_DEV_SM_CLOCK | TKE GPU Exporter 集成 |
| Memory Clock | DCGM_FI_DEV_MEM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 1000 * 1000 | DCGM_FI_DEV_MEM_CLOCK | TKE GPU Exporter 集成 |
| APP SM Clock | DCGM_FI_DEV_APP_SM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 1000 * 1000 | DCGM_FI_DEV_APP_SM_CLOCK | TKE GPU Exporter 集成 |
| APP Memory Clock | DCGM_FI_DEV_APP_MEM_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 1000 * 1000 | DCGM_FI_DEV_APP_MEM_CLOCK | TKE GPU Exporter 集成 |
| Video Clock | DCGM_FI_DEV_VIDEO_CLOCK{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} * 1000 * 1000 | DCGM_FI_DEV_VIDEO_CLOCK | TKE GPU Exporter 集成 |
| Clock Throttle Reasons | DCGM_FI_DEV_CLOCK_THROTTLE_REASONS{cluster=~"$cluster",nodepool=~"$NodePool",Hostname=~"$GPUNode",UUID=~"$GPU"} | DCGM_FI_DEV_CLOCK_THROTTLE_REASONS | TKE GPU Exporter 集成 |

# 数据采集配置

最近更新时间：2024-07-05 11:48:11

## 操作场景

本文档介绍如何为已完成关联的集群配置监控采集项。

## 前提条件

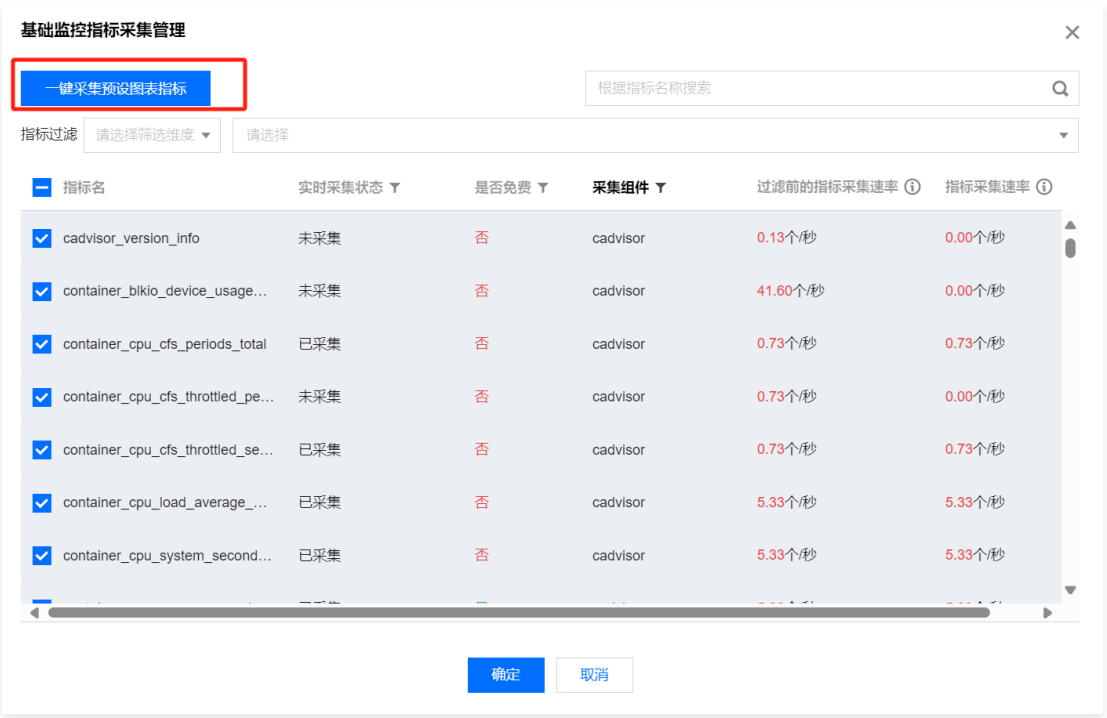
在配置监控数据采集项前，您需要完成以下操作：

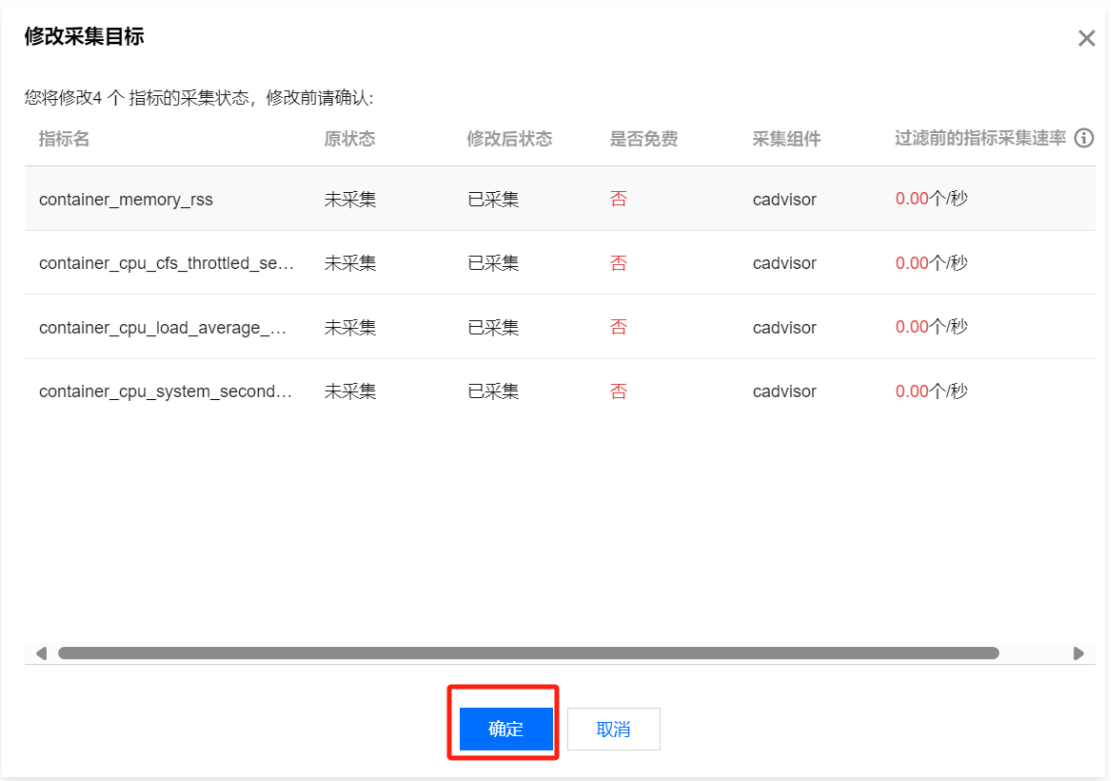
- 已成功创建 Prometheus 监控实例。
- 已将需要监控的集群关联到相应实例中。

## 操作步骤

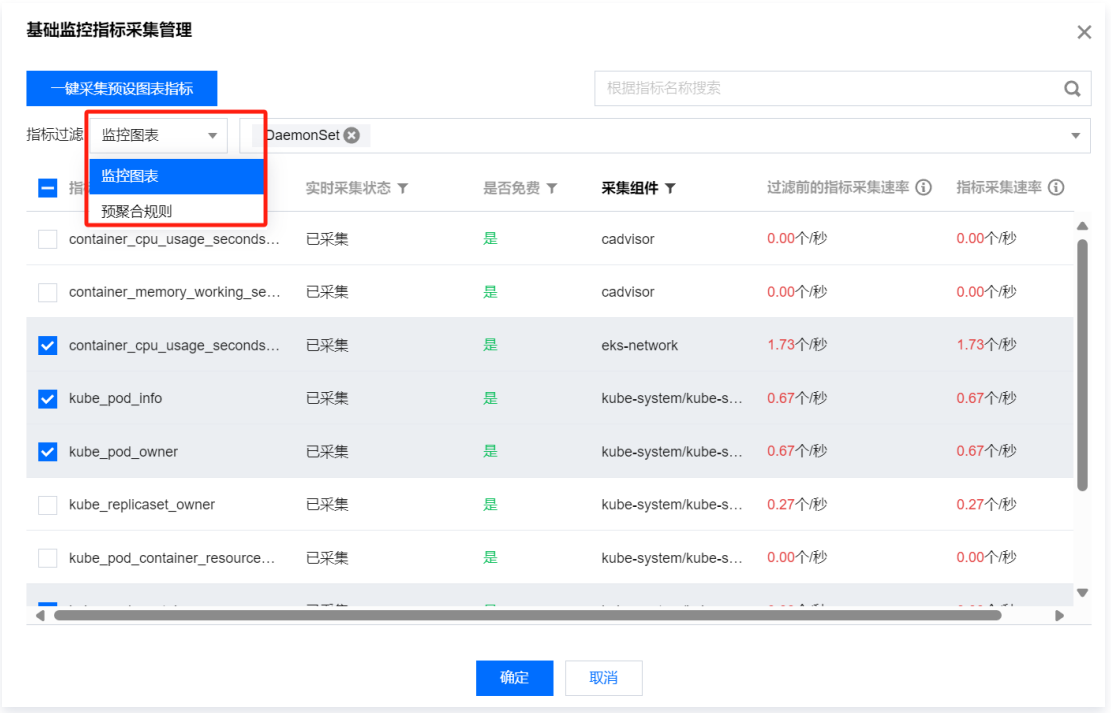
### 配置数据采集

1. 登录 腾讯云可观测平台，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在**数据采集 > 集成容器服务**页面，单击实例右侧的**数据采集配置**，进入采集配置列表页。
4. 点击页面上方的**采集指标管理**，会弹出基础监控指标采集管理页面。
   - 若需要采集所有容器图表指标，可以在弹出的窗口中点击**一键采集预设图表指标**，在弹出的确认窗口点击**确定**，将会采集所有预设图表指标。






- 若只需要采集部分指标，可以通过**监控图表**和**预聚合规则**进行指标筛选，勾选自己需要的指标后再点击**确定**即可。如下图所示：



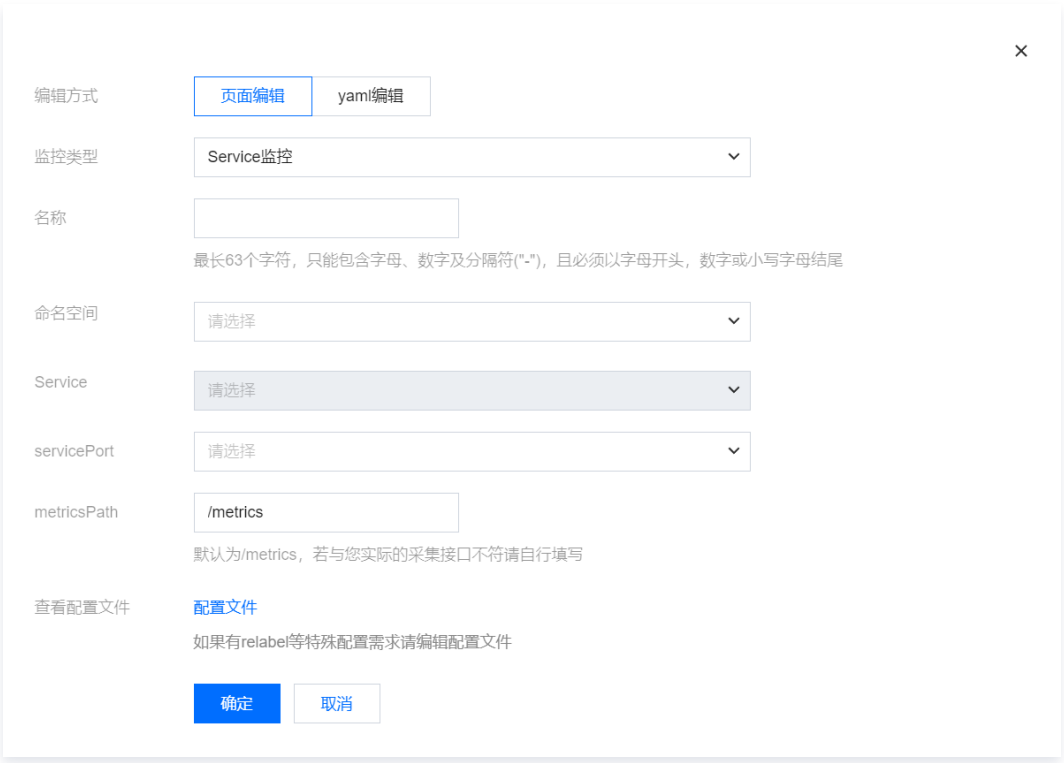

5. 在“数据采集配置”页中，单击**新建自定义监控**，新增数据采集配置。Prometheus 监控服务预置了部分采集配置文件，用于采集常规的监控数据。您可以通过以下两种方式配置新的数据采集规则，以监控您的业务数据。

通过控制台新增配置

### 监控 Service

1. 单击**页面编辑**。

2. 在“新建采集配置”弹窗中，填写配置信息。如下图所示：

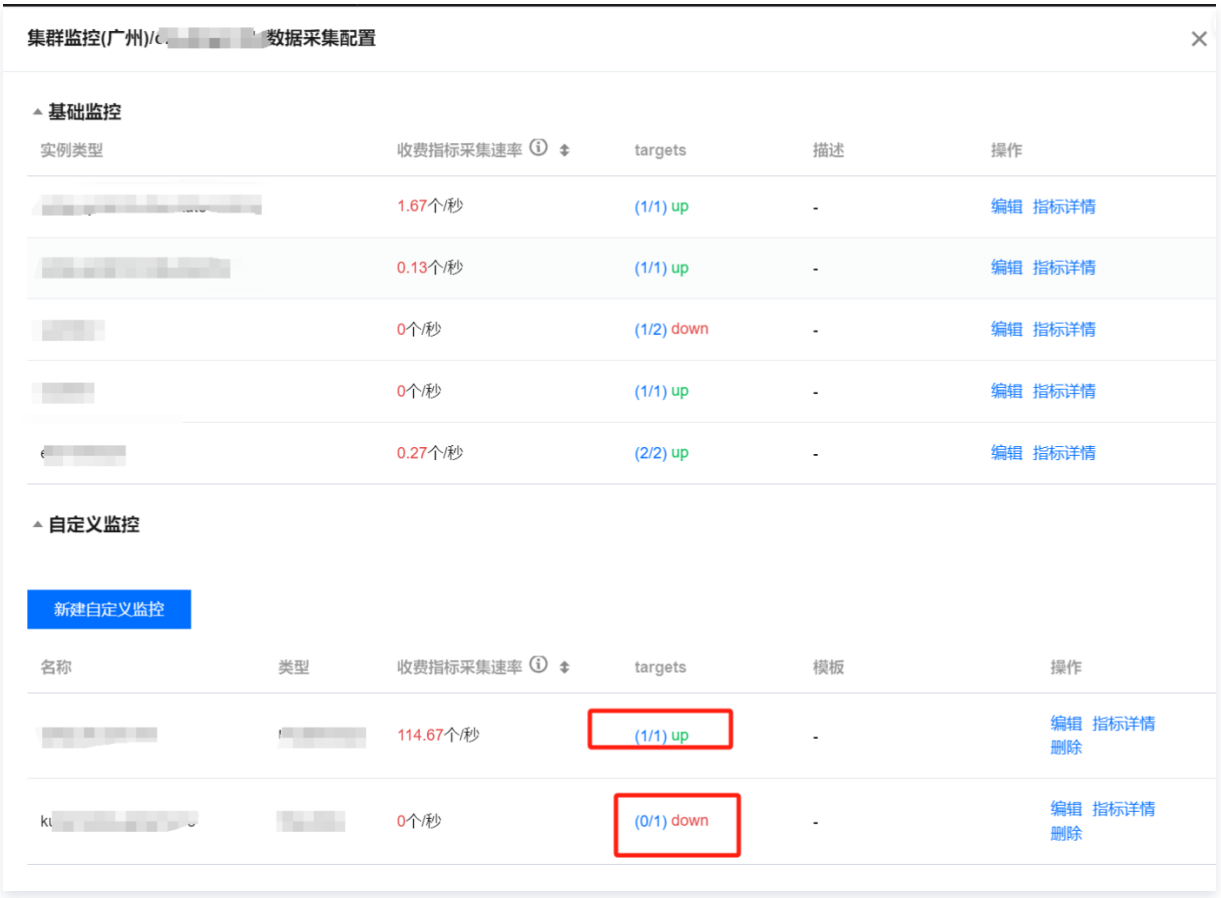



**通过 yaml 文件新增配置**

1. 单击 **yaml 编辑**。
2. 在弹窗中，选择监控类型，并填写相应配置。
   您可以按照社区的使用方式通过提交相应的 yaml 来完成数据采集的配置。
   - **工作负载监控**：对应配置为 PodMonitors。
   - **service 监控**：对应配置为 ServiceMonitors。
   - **RawJobs 监控**：对应配置为 RawJobs。

6. 单击**确定**完成配置。

7. 在该实例的“数据采集配置”页中，查看采集目标状态。如下图所示：

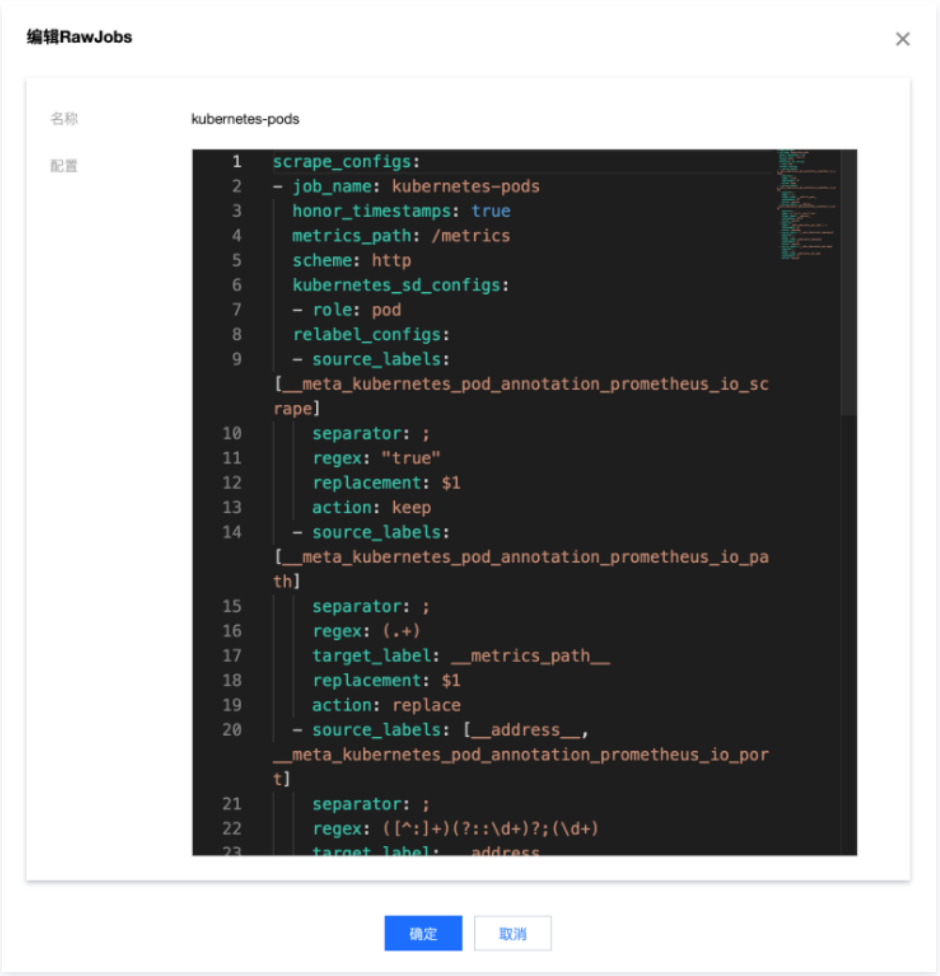



**targets（1/1）**表示（实际抓取的 targets 数为1 / 探测的采集目标数为1）。当实际抓取数和探测数的数值相等时，显示为 **up**，即表示当前抓取正常。当实际抓取数小于探测数时，显示为 **down**，即表示有部分 endpoints 抓取失败。
单击上图中的字段值（**0/1**）即可查看采集目标的详细信息。如下图所示 down 的失败状态：


## 查看已有配置

1. 登录 腾讯云可观测平台，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在**数据采集 > 集成容器服务**页面，单击实例右侧的**数据采集配置**，进入采集配置列表页。选择**基础监控**或者**自定义监控**，单击右侧的**编辑**。

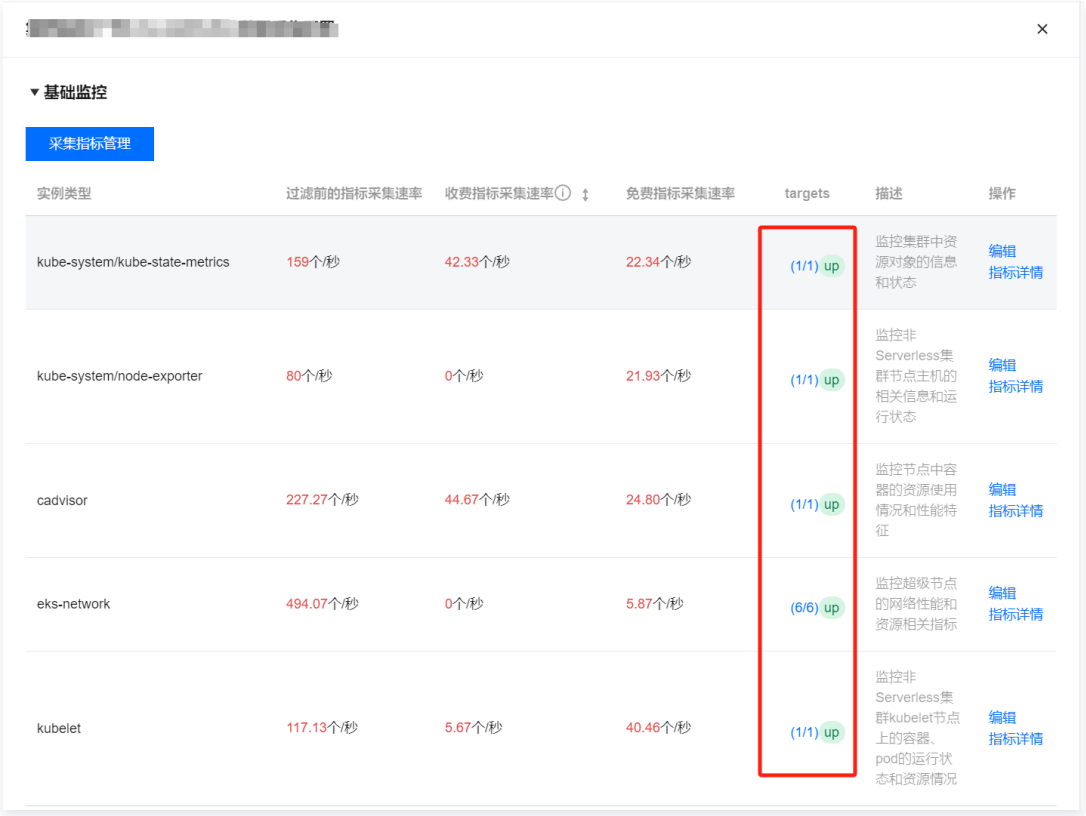
4. 在弹出的窗口查看 yaml 文件中当前配置的所有监控对象。如下图所示：

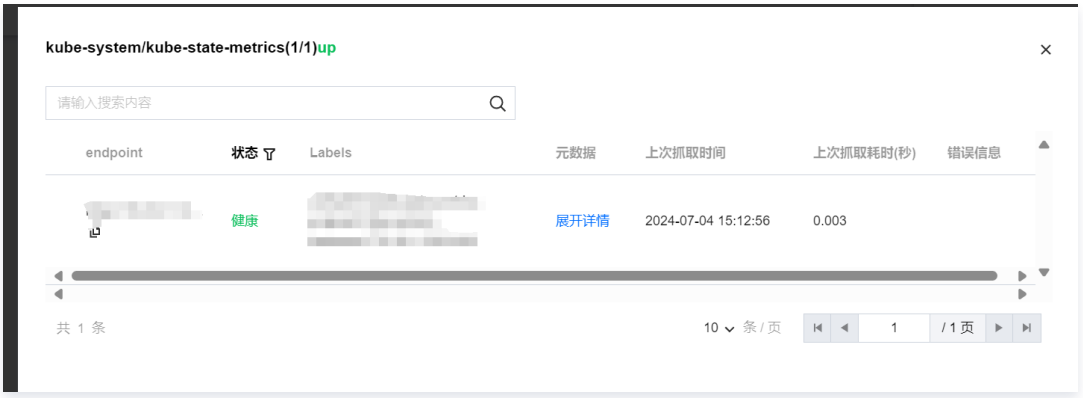



## 查看采集目标

1. 登录 腾讯云可观测平台，选择左侧导航栏中的 **Prometheus 监控服务**。
2. 在监控实例列表页，选择需要查看 Targets 的实例名称，进入该实例详情页。
3. 在**数据采集 > 集成容器服务**页面，单击实例右侧的**数据采集配置**。

4. 在弹出的窗口中点击 targets 下方的显示状态，即可跳转到当前数据的详情页。




## 相关操作

### 挂载文件到采集器

在配置采集项的时候，如果您需要为配置提供一些文件，例如证书，可以通过以下方式向采集器挂载文件，文件的更新会实时同步到采集器内。

- **prometheus.tke.tencent.cloud.com/scrape-mount = "true"**
  prom-xxx 命名空间下的 configmap 添加如上 label，其中所有的 key 会被挂载到采集器的路径 `/etc/prometheus/configmaps/[configmap-name]/` 。
- **prometheus.tke.tencent.cloud.com/scrape-mount = "true"**
  prom-xxx 命名空间下的 secret 添加如上 label，其中所有的 key 会被挂载到采集器的路径 `/etc/prometheus/secrets/[secret-name]/` 。

# 精简监控指标

最近更新时间：2024-10-31 18:11:52

本文档介绍如何精简 Prometheus 监控服务的**采集指标**，避免不必要的费用支出。

## 前提条件

在配置监控数据采集项前，您需要完成以下操作：

- 已成功 创建 Prometheus 监控实例 。
- 已将需要 监控的集群关联到相应实例 中。

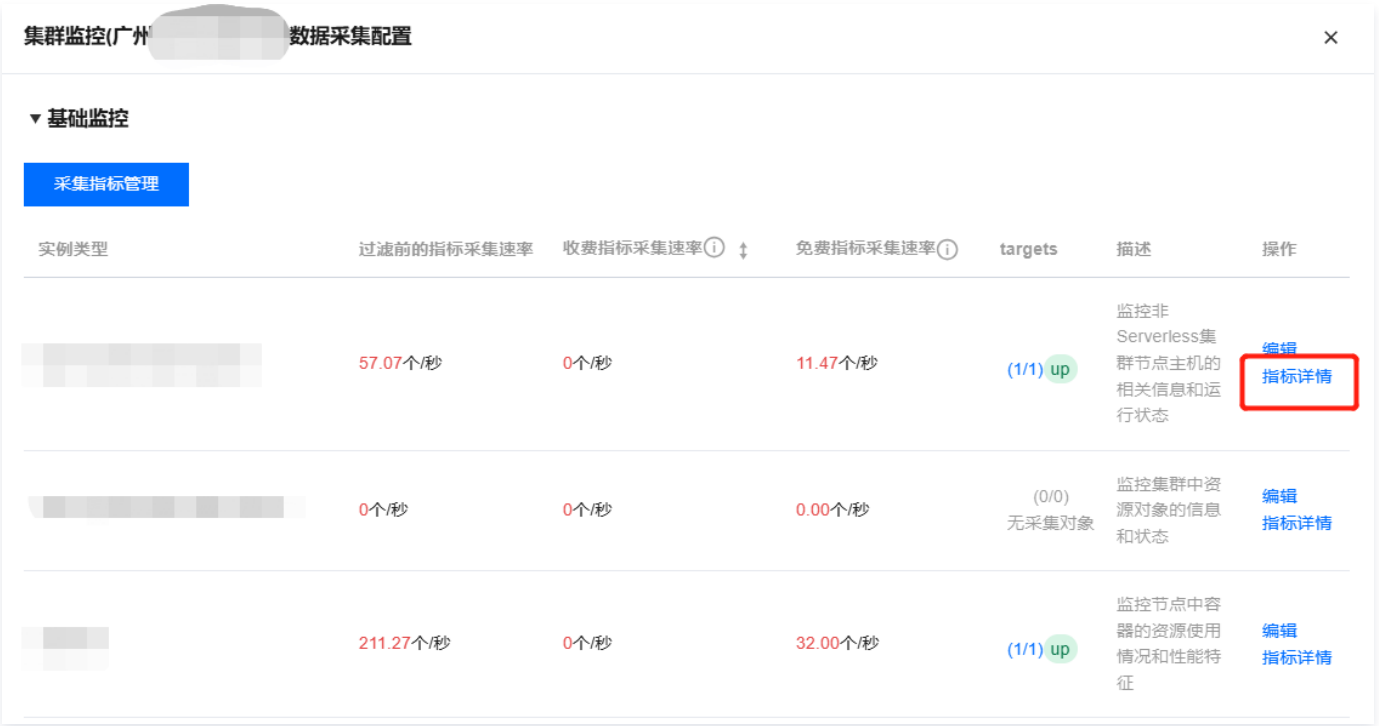
## 精简指标

### 通过控制台精简指标

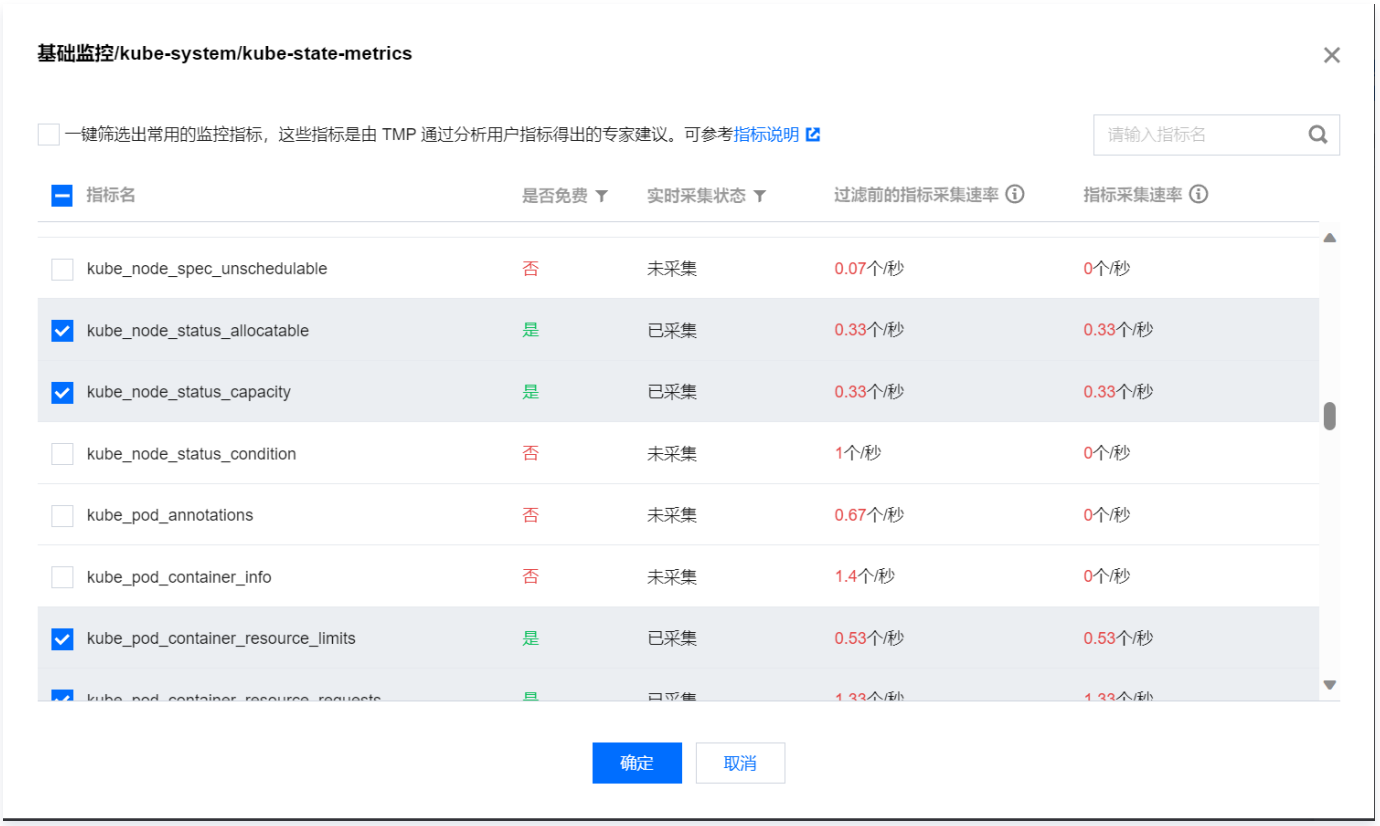
Prometheus 监控服务提供了一百多个免费的基础监控指标，完整的指标列表可查看 按量付费免费指标 。

1. 登录 腾讯云可观测平台控制台，选择左侧导航栏中的 Prometheus 监控 。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：


4. 进入采集配置列表页后，基础指标支持通过产品化的页面增加/减少采集对象，单击右侧的**指标详情**，如下图所示：




5. 在以下页面您可以查看到每个指标是否免费，指标勾选表示会采集这些指标，您可按需勾选使用。仅基础监控提供免费的监控指标，完整的免费指标详情请参见 按量付费免费指标 。付费指标计算详情见 Prometheus 监控服务按量计费 。




## 通过 YAML 精简指标

Prometheus 目前收费模式为按监控数据的点数收费，为了最大程度减少不必要的浪费，建议您针对采集配置进行优化，只采集需要的指标，过滤掉非必要指标，从而减少整体上报量。详细的计费方式和相关云资源的使用请查看 文档 。

以下步骤将分别介绍如何在自定义指标的 ServiceMonitor、PodMonitor，以及原生 Job 中加入过滤配置，精简自定义指标。

1. 登录 腾讯云可观测平台控制台，选择左侧导航栏中的 Prometheus 监控 。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**。
4. 单击实例右侧的**编辑**查看指标详情。

### ServiceMonitor 和 PodMonitor

ServiceMonitor 和 PodMonitor 的过滤配置字段相同，本文以 ServiceMonitor 为例。

ServiceMonitor 示例：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 1.9.7
  name: kube-state-metrics
  namespace: kube-system
spec:
  endpoints:
  - bearerTokenSecret:
      key: ""
    interval: 15s # 该参数为采集频率，您可以调大以降低数据存储费用，例如不重要的指标可以改为 300s，可以降低20倍的监控数据采集量
    port: http-metrics
```

```
    scrapeTimeout: 15s # 该参数为采集超时时间，Prometheus 的配置要求采集超时时间不能超过采集间隔，即：
scrapeTimeout <= interval
  jobLabel: app.kubernetes.io/name
  namespaceSelector: {}
  selector:
    matchLabels:
      app.kubernetes.io/name: kube-state-metrics
```

若要采集 `kube_node_info` 和 `kube_node_role` 的指标，则需要在 ServiceMonitor 的 endpoints 列表中，加入 `metricRelabelings` 字段配置。注意：是 **`metricRelabelings`** 而不是 `relabelings`。
添加 `metricRelabelings` 示例：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 1.9.7
  name: kube-state-metrics
  namespace: kube-system
spec:
  endpoints:
  - bearerTokenSecret:
      key: ""
    interval: 15s # 该参数为采集频率，您可以调大以降低数据存储费用，例如不重要的指标可以改为 300s，可以降低20倍的监控数据采集量
    port: http-metrics
    scrapeTimeout: 15s
    # 加了如下四行：
    metricRelabelings: # 针对每个采集到的点都会做如下处理
    - sourceLabels: ["__name__"] # 要检测的label名称，__name__ 表示指标名称，也可以是任意这个点所带的label
      regex: kube_node_info|kube_node_role # 上述label是否满足这个正则，在这里，我们希望__name__满足kube_node_info或kube_node_role
      action:  keep # 如果点满足上述条件，则保留，否则就自动抛弃
  jobLabel: app.kubernetes.io/name
  namespaceSelector: {}
  selector:
```

### 原生 Job

如果使用的是 Prometheus 原生的 Job，则可以参考以下方式进行指标过滤。
Job 示例：

```
scrape_configs:
  - job_name: job1
    scrape_interval: 15s # 该参数为采集频率，您可以调大以降低数据存储费用，例如不重要的指标可以改为 300s，可以降低20倍的监控数据采集量
    static_configs:
      - targets:
          - '1.1.1.1'
```

若只需采集 `kube_node_info` 和 `kube_node_role` 的指标，则需要加入 `metric_relabel_configs` 配置。注意：是 **`metric_relabel_configs`** 而不是 `relabel_configs`。
添加 `metric_relabel_configs` 示例：

```
scrape_configs:
```

```
  - job_name: job1
    scrape_interval: 15s # 该参数为采集频率，您可以调大以降低数据存储费用，例如不重要的指标可以改为 300s，可以降低20倍的监控数据采集量
    static_configs:
    - targets:
      - '1.1.1.1'
    # 加了如下四行：
    metric_relabel_configs: # 针对每个采集到的点都会做如下处理
    - source_labels: ["__name__"] # 要检测的label名称，__name__ 表示指标名称，也可以是任意这个点所带的label
      regex: kube_node_info|kube_node_role # 上述label是否满足这个正则，在这里，我们希望__name__满足kube_node_info或kube_node_role
      action: keep # 如果点满足上述条件，则保留，否则就自动抛弃
```

5. 单击**确定**。

## 屏蔽部分采集对象

### 屏蔽整个命名空间的监控

Prometheus 关联集群后，默认会纳管集群中所有 ServiceMonitor 和 PodMonitor，若您想屏蔽某个命名空间下的监控，可以为指定命名空间添加 label：`tps-skip-monitor: "true"`，关于 label 的操作请 参考 。

### 屏蔽部分采集对象

Prometheus 通过在用户的集群里面创建 ServiceMonitor 和 PodMonitor 类型的 CRD 资源进行监控数据的采集，若您想屏蔽指定 ServiceMonitor 和 PodMonitor 的采集，可以为这些 CRD 资源添加 label：`tps-skip-monitor: "true"`，关于 label 的操作请 参考 。

# 调整采集间隔

最近更新时间：2024-11-06 18:22:52

本文档介绍如何调整 Prometheus 监控服务的**采集间隔**，避免不必要的费用支出。

## 前提条件

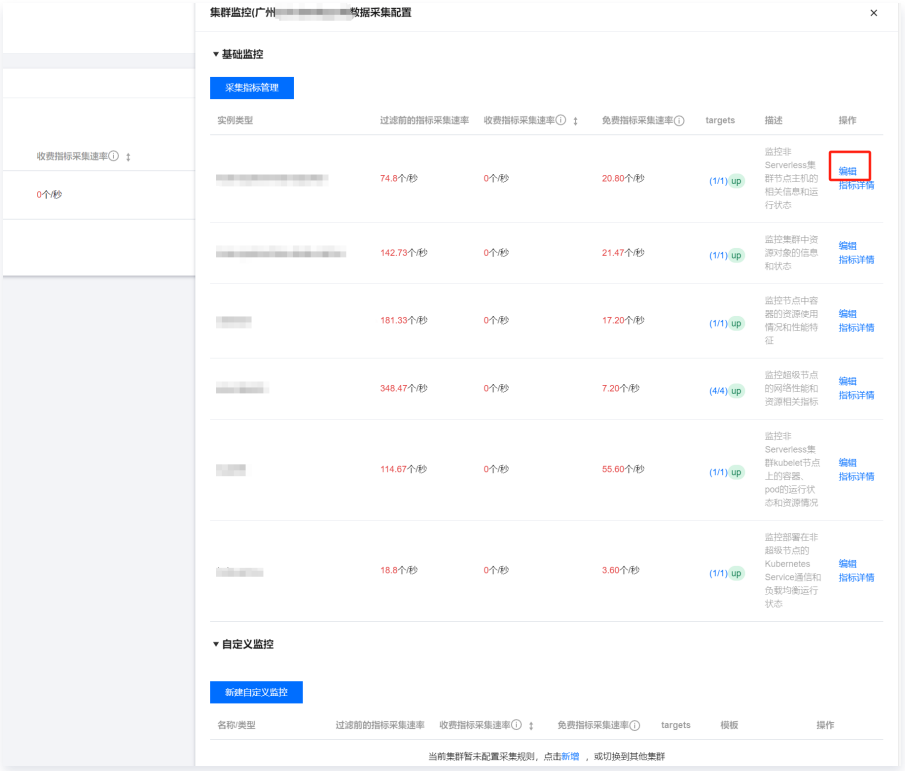
在配置监控数据采集项前，您需要完成以下操作：

- 已成功 创建 Prometheus 监控实例 。
- 已将需要 监控的集群关联到相应实例 中。

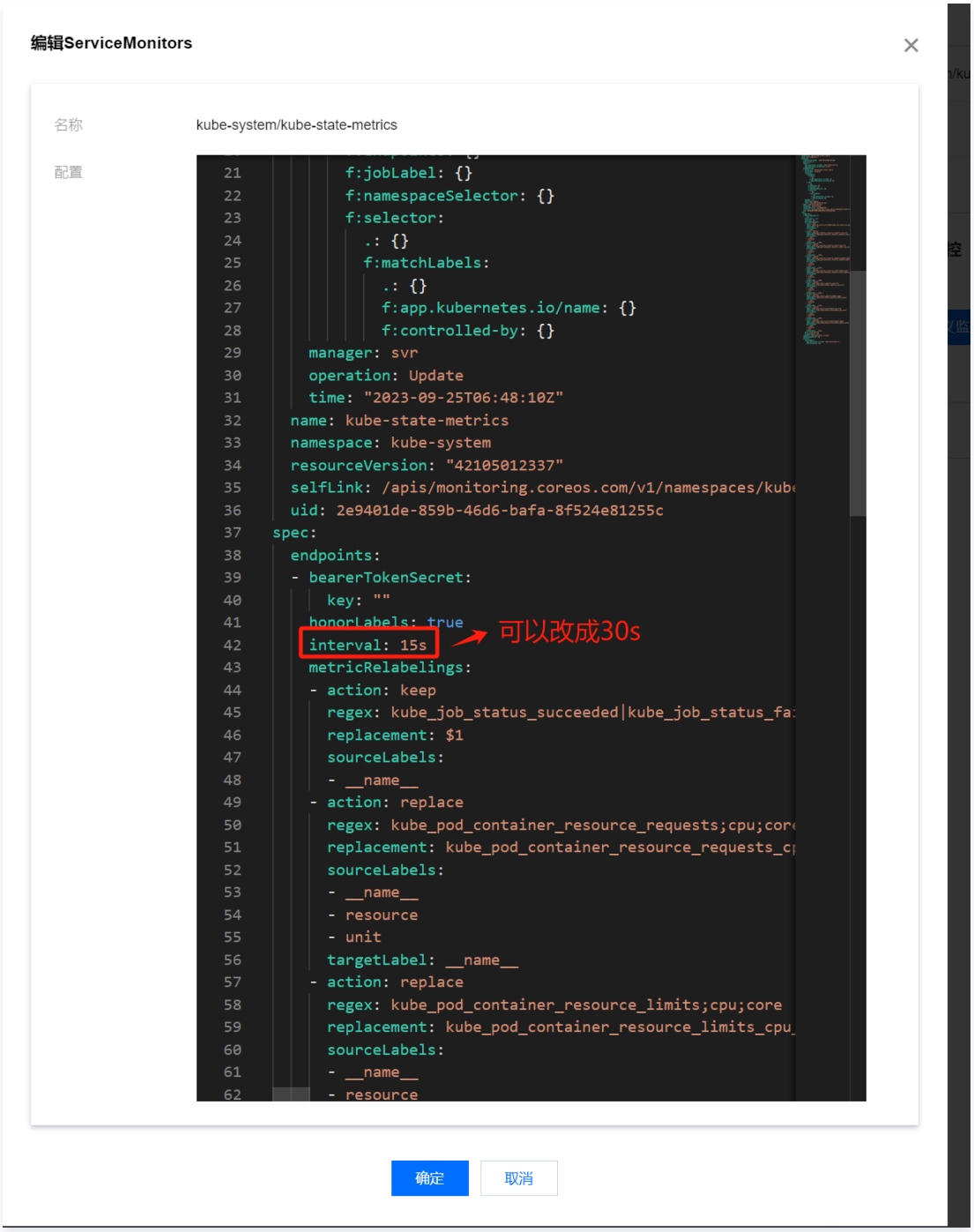
## 通过集成容器服务的控制台调整采集间隔

1. 登录 腾讯云可观测平台控制台 ，选择左侧导航栏中的 Prometheus 监控 。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**， 如下图所示：

4. 进入采集配置列表页后，单击右侧的**编辑**，如下图所示：



5. 找到 **interval** 字段，该参数为采集频率，您可以调大以降低数据存储费用，如下图所示：



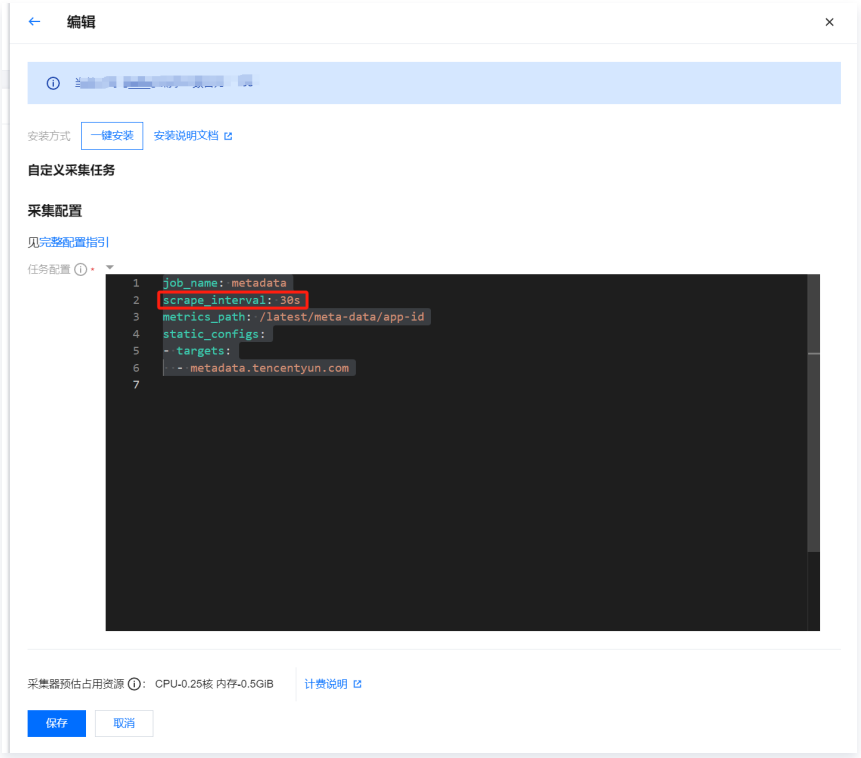

6. 配置完成后单击**确定**。

### 通过集成中心的控制台调整采集间隔

> **说明：**
> “抓取任务”集成已经一键安装成功，若未安装需要先单击右侧的**一键安装**后再开始下列操作。

1. 登录 腾讯云可观测平台控制台，选择左侧导航栏中的 Prometheus 监控。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在**数据采集**页面，单击**集成中心**，选择**抓取任务**，在弹出的窗口中选择**已集成**， 如下图所示：


4. 在集成列表页面，单击任意一个名称，进入采集任务配置页，找到 **scrape_interval** 字段，该参数为采集频率，您可以调大以降低数据存储费用，如下图所示：




5. 配置完成后单击**保存**。

# 相关资源使用及计费说明

最近更新时间：2024-10-21 22:06:01

当您使用 Prometheus 监控服务（TMP）服务时，可能会使用到 **EKS 集群**、**Grafana 服务**和**负载均衡 CLB** 资源，本文将为您介绍这些资源使用场景以及相关费用。

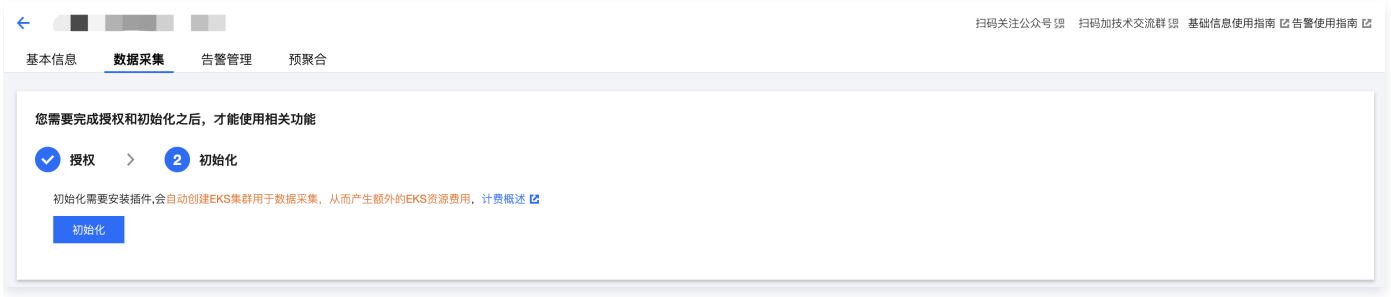
## EKS 集群

### 使用场景

配置数据采集，需要进行初始化，创建采集端所需的 EKS 集群，包括以下使用场景：

- 关联容器集群来监控容器服务。
- 在集成中心安装集成常用组件。

初始化入口如下图所示：




在 集群列表页 可查看已创建的 EKS 集群，如下图所示：


**特殊说明**：如果账号类型是非带宽上移账号，需要先 升级 为带宽上移账号后，才能进行初始化并创建 EKS 集群。


### 注意事项

该 EKS 集群的名称为 Prometheus 监控服务**实例的 ID**，集群描述里面说明为 **Prometheus 监控专用，请勿修改或删除**。


### 计费说明

**计费方式为按量计费，计费详情请参见 EKS 产品定价 。**
**EKS 集群会按照监控量进行自动扩缩容，监控规模和 EKS 集群费用的关系可参见下表：**

| 用户上报的瞬时 Series 量级 | 预估需要的 EKS 资源 | 对应的刊例价费用/日 |
| --- | --- | --- |
| <50w | 1.25核 1.6GiB | 1.3元 |
| 100w | 0.5核1.5GiB*2 | 5.5元 |
| 500万 | 1核3GiB*3 | 11元 |
| 2000万 | 1核6GiB*5 | 30元 |
| 3000万 | 1核6GiB*8 | 48元 |

**EKS 集群成本示例**
**一个新初始化的 Prometheus 实例所用 EKS 集群消耗了：CPU：1.25 核、内存：1.5GiB。预计一天刊例价费用为：0.12 x 24 + 0.05 x 24 = 4.08 元。**

## Grafana 服务

### 使用场景

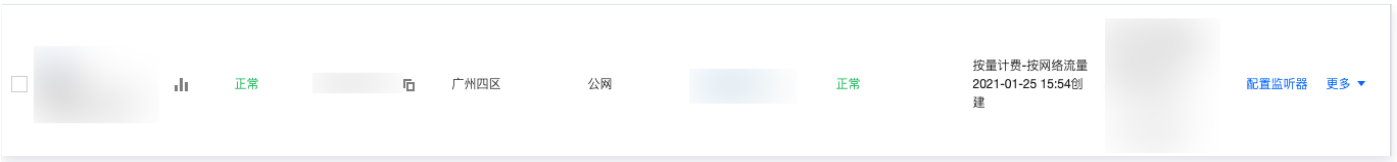
**当您创建 Prometheus 实例时，需要绑定一个地域相同的 Grafana 实例，用于可视化展示 Prometheus 采集的监控数据。详细说明请参见 Grafana 服务计费说明 。**

## 负载均衡 CLB

### 使用场景

**若用户使用 Prometheus 监控服务时，关联了边缘集群或跨 VPC 关联了未打通网络的集群，需要创建公网的 CLB 进行网络联通，此时会创建一个公网 CLB。**

我们将会对这些 CLB 资源收取费用，可在 负载均衡控制台 查看创建的公网 LB 信息，如下图所示：




该资源按实际使用量计费，计费详情请参见负载均衡 > 标准账户类型计费说明 文档。

## 资源销毁

在 Prometheus 监控服务控制台 销毁监控实例，对应的所有资源将会一并销毁。腾讯云不主动回收用户的监控实例，若您不再使用 Prometheus 监控服务，请务必及时删除监控实例，以免发生资源的额外扣费。如需销毁 Prometheus 实例可参见 销毁实例 文档。

# 实践教程
# 自建 Prometheus 迁入

最近更新时间：2024-07-11 15:30:11

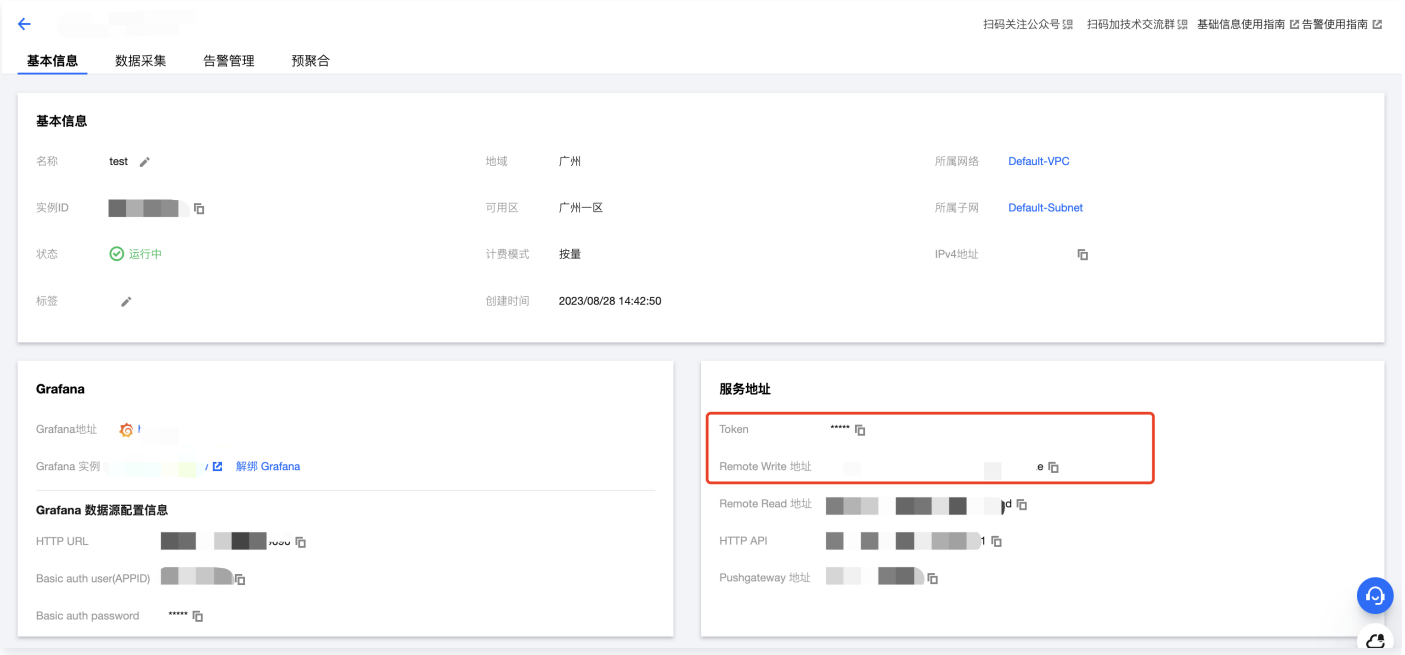
## 操作场景

已经有自建 Prometheus 需快速迁移到 Prometheus 监控服务。

## 操作步骤

Prometheus 本身支持 Remote Write 到一个外部存储，因此沿用这个思想，在自建 Prometheus 的配置文件中加一个 Remote Write 配置指向到 Prometheus 监控服务即可。具体操作步骤如下：

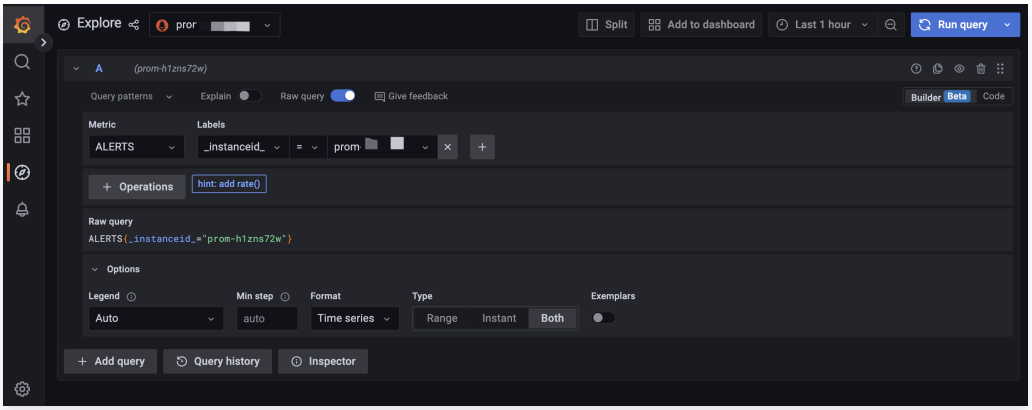
1. 选择对应的 Prometheus 实例，进入实例详情页，单击顶部菜单栏的**基本信息**，在**服务地址**卡片中获取 Prometheus 监控服务的 Remote Write 地址及 Token，如下：




2. 修改 `prometheus.yml`，修改完成 `重启 Prometheus`，具体配置如下，如需了解更多 Remote Write 配置参数，请参见 remote_write 。

```
remote_write:

  - name: cm_prometheus # Remote write 的名称
    url: http://ip:port/api/v1/prom/write  # 从 Prometheus 基本信息中获取 Remote Write 地址，建议加上双引号避免特殊字符解析错误
    remote_timeout: 30s # 根据实际情况设置
    bearer_token: k32*****trR # 从 Prometheus 基本信息中获取 Token 信息
```

3. 打开 Prometheus 监控服务自带的 Grafana，通过 `Explore` 来验证数据是否写入成功，如下图，也可以 自定义 Grafana 监控大盘 。




4. 也可以通过 Prometheus API 进行自建可视化，详情请参见 监控数据查询 。

# 云服务器场景下自定义接入

最近更新时间：2024-10-17 22:07:32

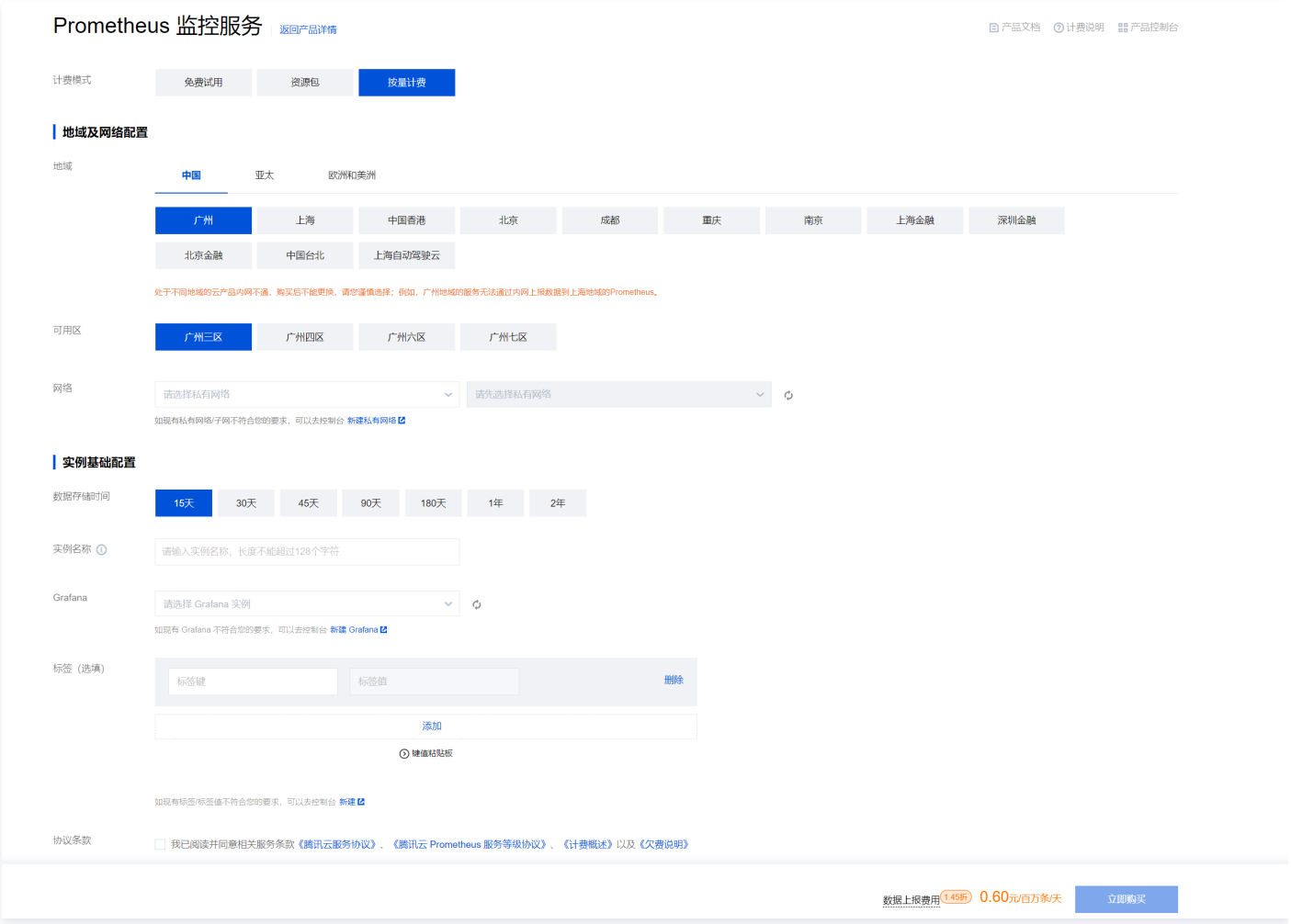
本文将详细为您介绍腾讯云云服务器（CVM）如何从零开始接入 Prometheus 监控服务。

## 购买 Prometheus

> **说明**
> 购买的 Prometheus 实例需跟监控的云服务器同一个 vpc 下，才能实现网络互通。

1. 登录 Prometheus 监控服务控制台，单击**新建**购买 Prometheus 实例 。


2. 在购买页中，选择合适的实例规格、网络。选择相同 vpc 网段，保证 Prometheus 能与需要采集的云服务器网段相同，才能采集到数据。实例规格，可根据自己的业务上报量进行选择。




3. 选择完后，单击**立即购买**并支付即可。

> **说明**
> 如需了解 Prometheus 更多定价规则，请参见 产品定价 。

## 接入云服务器基础指标

1. 下载安装 node_exporter：
   在需要上报数据的主机上，下载并安装 node_exporter，您可以点击进入 Prometheus 开源官网下载地址 node_exporter，也可以直接执行下列命令：

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter -O node_exporter
```

目录为当前文件夹：

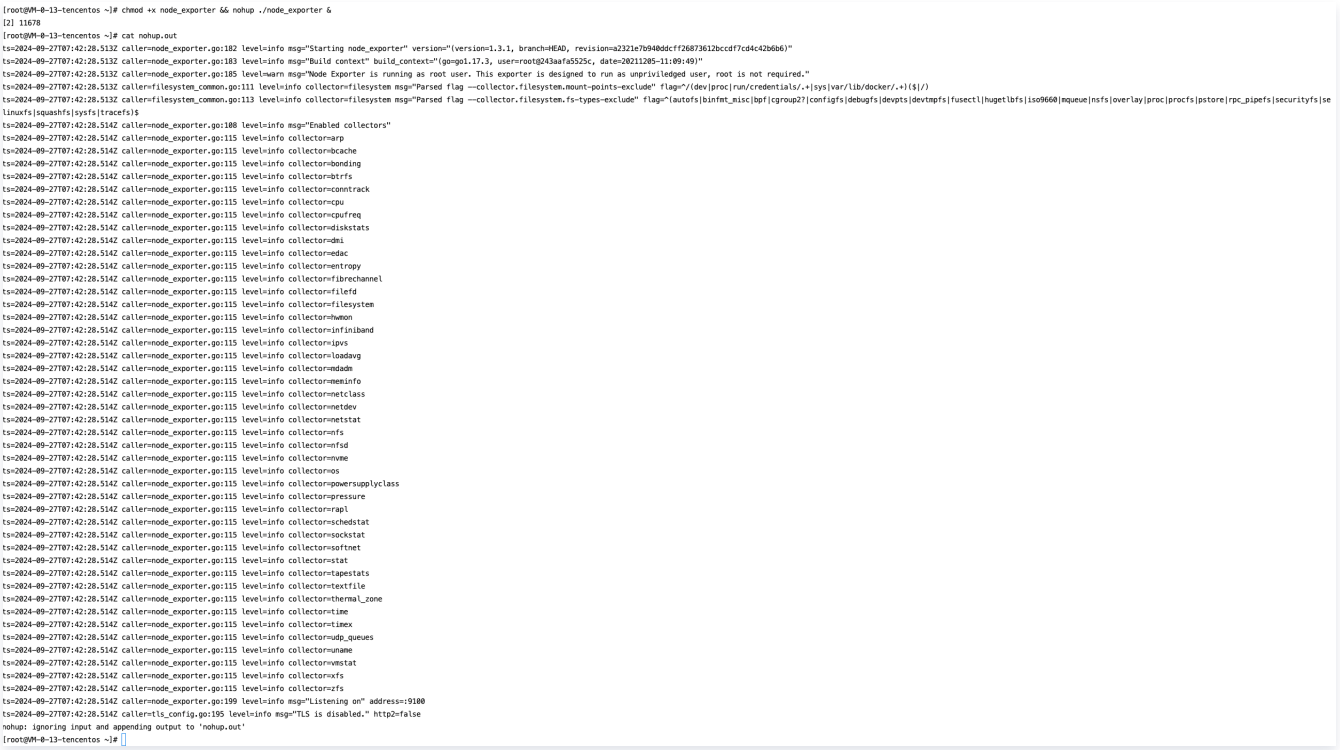
```
[root@VM-0-46-tencentos ~]# ls
node_exporter
[root@VM-0-46-tencentos ~]#
```

2. 运行 node_exporter 采集基础监控数据：

- 赋予权限，执行 node_exporter 并查看日志。

```
chmod +x node_exporter && nohup ./node_exporter &
cat nohup.out
```

如下图所示即为执行成功：



```
[root@VM-0-13-tencentos ~]# chmod +x node_exporter && nohup ./node_exporter &
[2] 11678
[root@VM-0-13-tencentos ~]# cat nohup.out
ts=2024-09-27T07:42:28.513Z caller=node_exporter.go:182 level=info msg="Starting node_exporter" version="(version=1.3.1, branch=HEAD, revision=a2321e7b940ddcff26873612bccdf7cd4c42b6b6)"
ts=2024-09-27T07:42:28.513Z caller=node_exporter.go:183 level=info msg="Build context" build_context="(go=go1.17.3, user=root@243aafa5525c, date=20211205-11:09:49)"
ts=2024-09-27T07:42:28.513Z caller=node_exporter.go:185 level=warn msg="Node Exporter is running as root user. This exporter is designed to run as unprivileged user, root is not required."
ts=2024-09-27T07:42:28.513Z caller=filesystem_common.go:111 level=info collector=filesystem msg="Parsed flag --collector.filesystem.mount-points-exclude" flag="^/(dev|proc|run/credentials/.+|sys|var/lib/docker/.+)($|/)"
ts=2024-09-27T07:42:28.513Z caller=filesystem_common.go:113 level=info collector=filesystem msg="Parsed flag --collector.filesystem.fs-types-exclude" flag="^(autofs|binfmt_misc|bpf|cgroup2?|configfs|debugfs|devpts|devtmpfs|fusectl|hugetlbfs|iso9660|mqueue|nsfs|overlay|proc|procfs|pstore|rpc_pipefs|securityfs|selinuxfs|squashfs|sysfs|tracefs)$"
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:108 level=info msg="Enabled collectors"
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=arp
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=bcache
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=bonding
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=btrfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=conntrack
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=cpu
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=cpufreq
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=diskstats
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=dmi
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=edac
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=entropy
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=fibrechannel
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=filefd
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=filesystem
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=hwmon
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=infiniband
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=ipvs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=loadavg
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=mdadm
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=meminfo
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=netclass
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=netdev
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=netstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=nfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=nfsd
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=nvme
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=os
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=powersupplyclass
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=pressure
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=rapl
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=schedstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=sockstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=softnet
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=stat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=tapestats
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=textfile
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=thermal_zone
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=time
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=timex
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=udp_queues
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=uname
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=vmstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=xfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=zfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:199 level=info msg="Listening on" address=:9100
ts=2024-09-27T07:42:28.514Z caller=tls_config.go:195 level=info msg="TLS is disabled." http2=false
nohup: ignoring input and appending output to 'nohup.out'
[root@VM-0-13-tencentos ~]#
```

- 可通过下列命令，查看暴露在9100端口的监控数据：

```
curl 127.0.0.1:9100/metrics
```

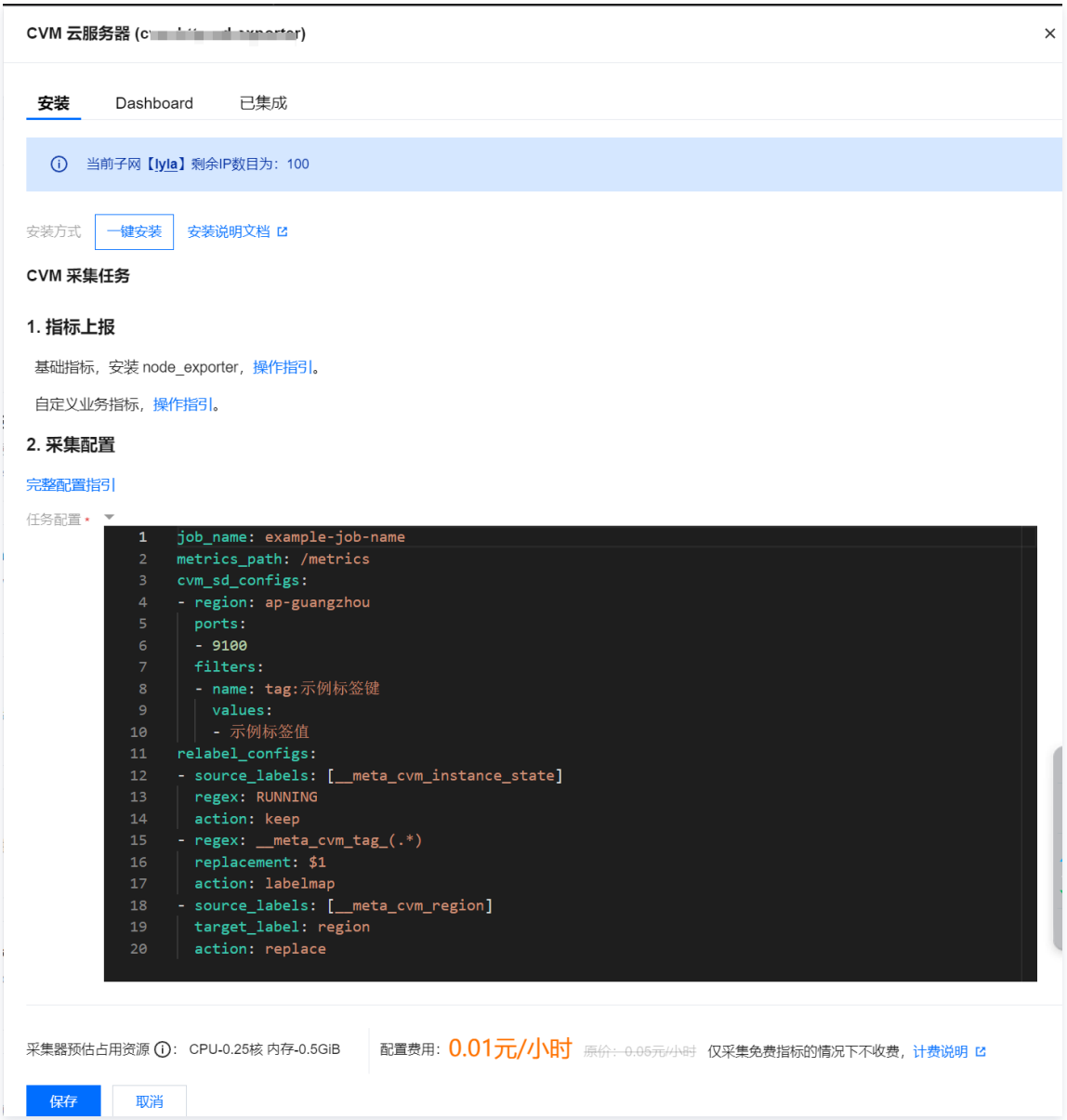
如下图为执行命令后看到的暴露出来的指标监控数据：

```
[root@VM-0-13-tencentos ~]# curl 127.0.0.1:9100/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.38724e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.38724e+06
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.445375e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 753
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 4.19036e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
```

3. 新增抓取任务：

i. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。

ii. 进入**数据采集 > 集成中心**,在集成中心页面找到**CVM 云服务器**卡片并点击弹出集成页面，根据页面提示进行配置即可。




iii. 抓取任务参考配置 如下：

```
job_name: example-job-name
metrics_path: /metrics
cvm_sd_configs:
- region: ap-guangzhou
  ports:
  - 9100
  filters:
  - name: tag:示例标签键
    values:
    - 示例标签值
relabel_configs:
- source_labels: [__meta_cvm_instance_state]
  regex: RUNNING
  action: keep
- regex: __meta_cvm_tag_(.*)
  replacement: $1
  action: labelmap
- source_labels: [__meta_cvm_region]
  target_label: region
  action: replace
```

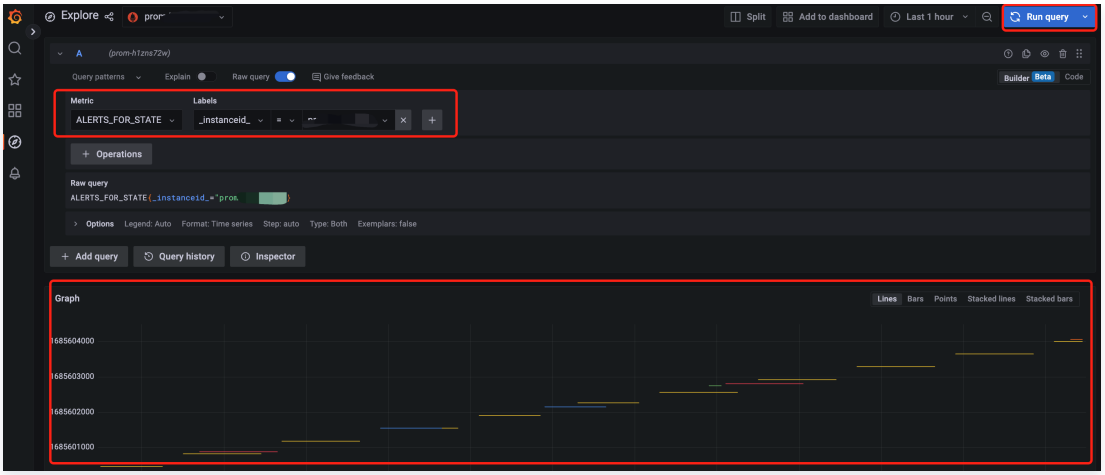
如需使用公网 IP 作为采集目标，需要在抓取任务中增加以下 relabel 规则：

```
- action: replace
  regex: ([^:]+):(.*);(.+)
  replacement: $3:$2
  source_labels:
  - __address__
  - __meta_cvm_public_ip
  target_label: __address__
```

4. 查看数据是否上报成功：

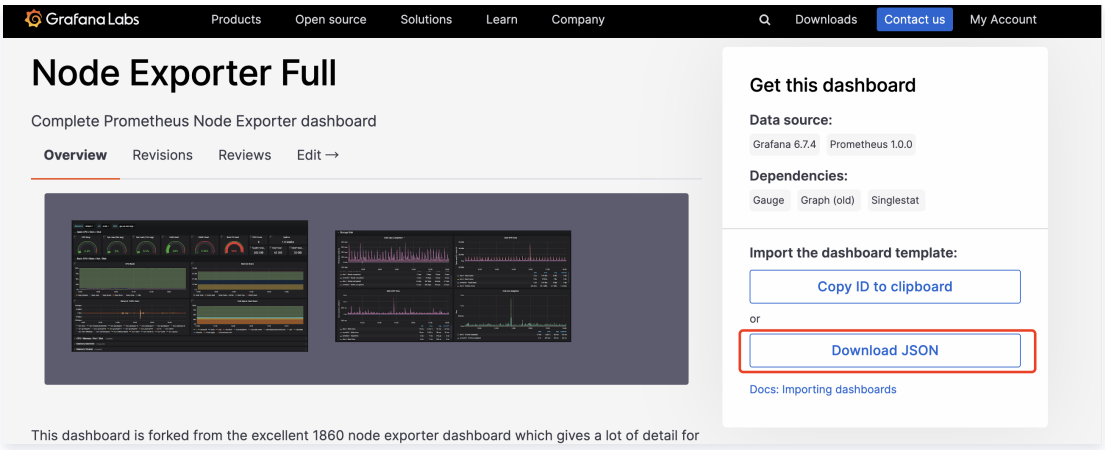
i. 登录 Prometheus 监控服务控制台，单击 Grafana 访问地址列的登录 Grafana。


ii. 如下图所示，到 Explore 搜索下 `{job="cvm_node_exporter"}` 查看是否有数据，若有数据，则表示上报成功。

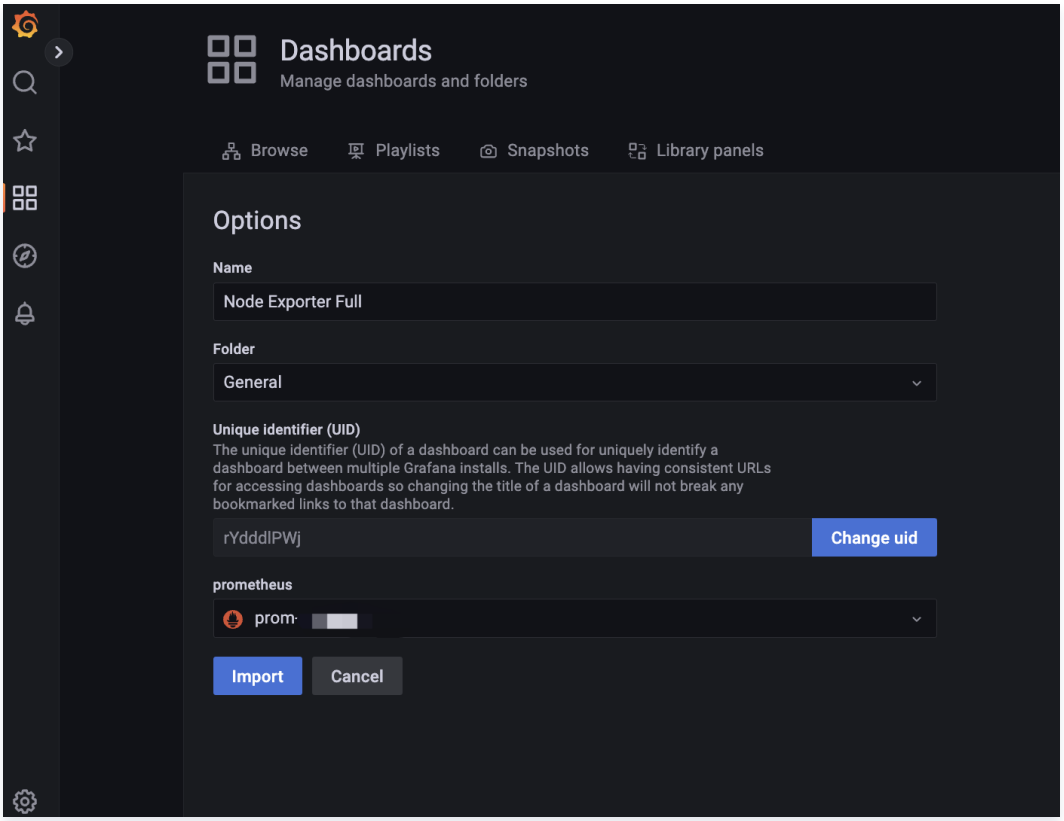



5. 配置 Dashboard 界面：Dashboard 界面每个产品都会有一些现成的 json 文件，可以直接导入。

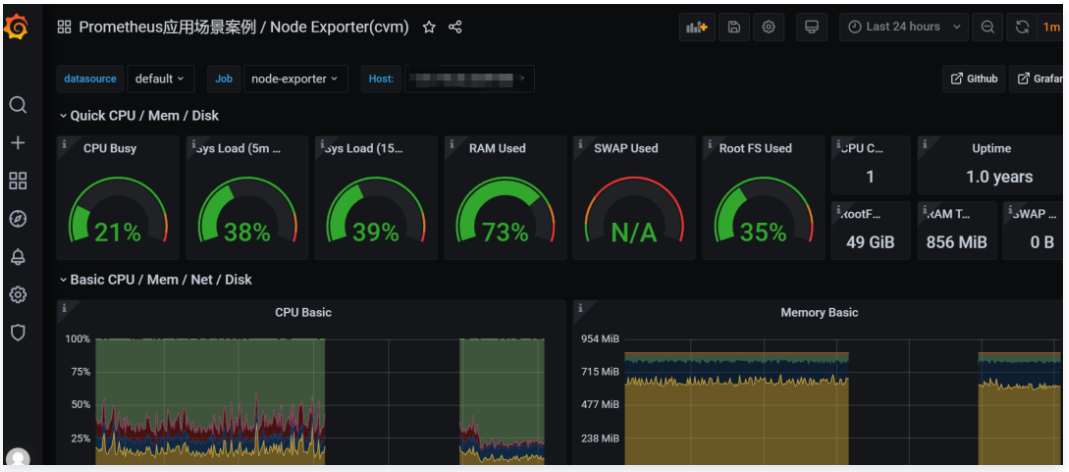
   i. **下载 Dashboard 文件**：登录 Dashboard-Node Exporter Full 页面，选择最新的 Dashboard 并下载。




   ii. **导入 Dashboard 的 json 文件**：登录 Prometheus 监控服务控制台，进入**基本信息 > Grafana 地址**，单击进入 Grafana，在 **Grafana 控制台 > Dashboards > Import > 在 Upload JSON file** 中上传 Dashboard 文件。




配置完后的效果如下：




## 接入云服务器业务层指标

Prometheus 根据监控的不同场景提供了 Counter、Gauge、Histogram、Summary 四种指标类型。Prometheus 社区提供了多种开发语言的 SDK，每种语言的使用方法基本上类似，主要是开发语言语法上的区别，下面主要以 Go 作为例子，使用 Counter 指标类型如何上报自定义监控指标数据。其余指标类型请参见 自定义监控 。

### Counter

计数类型，数据是单调递增的指标，服务重启之后会重置。可以用 Counter 来监控请求数、异常数、用户登录数和订单数等。

1. 如何通过 Counter 来监控订单数：

```
package order

import (
 "github.com/prometheus/client_golang/prometheus"
 "github.com/prometheus/client_golang/prometheus/promauto"
)
```

```
// 定义需要监控 Counter 类型对象
var (
 opsProcessed = promauto.NewCounterVec(prometheus.CounterOpts{
     Name: "order_service_processed_orders_total",
     Help: "The total number of processed orders",
 }, []string{"status"}) // 处理状态
)

// 订单处理
func makeOrder() {
 opsProcessed.WithLabelValues("success").Inc() // 成功状态
 // opsProcessed.WithLabelValues("fail").Inc() // 失败状态

 // 下单的业务逻辑
}
```

例如，通过 `rate()` 函数获取订单的增长率：

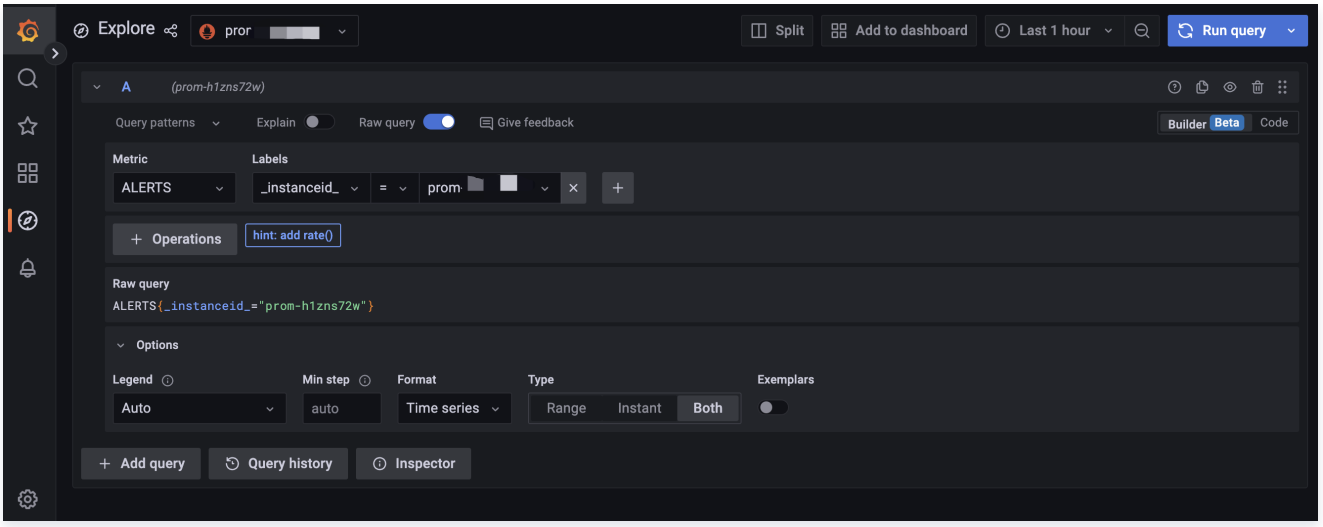
```
rate(order_service_processed_orders_total[5m])
```

2. 暴露 Prometheus 指标：
   通过 `promhttp.Handler()` 把监控埋点数据暴露到 HTTP 服务上。

```
package main

import (
 "net/http"

 "github.com/prometheus/client_golang/prometheus/promhttp"
)

func main() {
     // 业务代码

     // 把 Prometheus 指标暴露在 HTTP 服务上
     http.Handle("/metrics", promhttp.Handler())

     // 业务代码
}
```

3. 采集数据：
   完成相关业务自定义监控埋点之后，应用发布，即可通过 Prometheus 来抓取监控指标数据。采集完成后，等待数分钟，您可在 Prometheus 监控服务集

成的 Grafana 中查看业务指标监控数据。



# 容器场景监控

最近更新时间：2024-11-06 18:22:52

## 实践背景

众所周知，Prometheus 是容器场景的最佳监控工具，但自建 Prometheus 对于运维人力有限的中小型企业而言，成本太高；对于业务发展快速的大企业又容易出现性能瓶颈。因而使用云上托管 Prometheus 已成为越来越多上云企业的第一选择。那我们该如何使用托管 Prometheus 监控 腾讯云容器服务（TKE）。

## 实践步骤

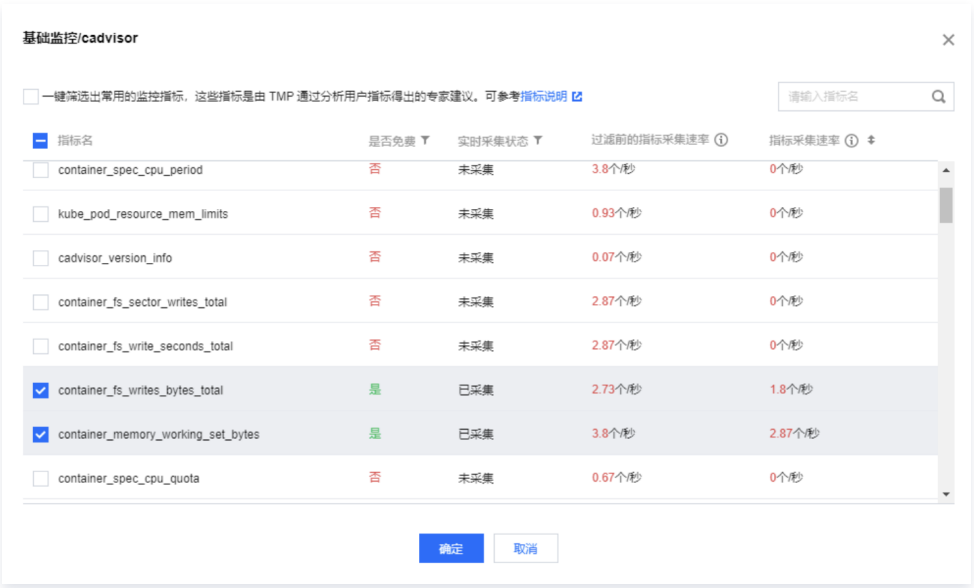
### 步骤1：购买实例

1. 登录 Prometheus 监控服务控制台。
2. 单击**新建**，根据您的需求选择购买地域、存储时长并选择需要关联的 Grafana 实例，若无 Grafana 实例，可参见 相关操作指引 创建（需创建实例并完成购买）。
3. 完成后，单击**立即购买**即可。详细计费规则请查看 计费规则。

### 步骤2：集成容器服务

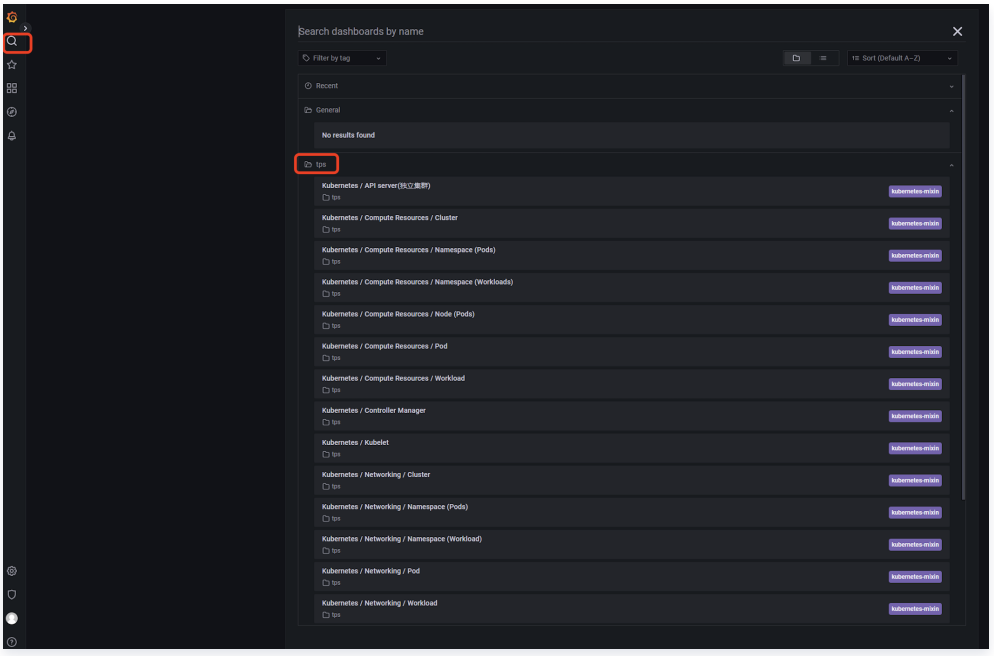
1. 新建完实例后，在实例列表中单击对应的实例 ID/名称，进入实例详情页面。
2. 在顶部导航栏中单击**数据采集 > 集成容器服务 > 关联集群**。
3. 在弹窗中选择需要关联的集群，共支持接入4种集群类型（标准集群、弹性集群、注册集群、边缘集群），同时支持集群跨 VPC，不同 VPC 网络不互通的情况需要勾选创建公网 CLB。


4. **关联集群后，可以点击已关联集群列表页中操作列数据采集配置，进入数据采集配置页面，再点击基础监控操作列的指标详情，查看默认采集的免费基础指标，同时可以调整新增或减少采集指标。**

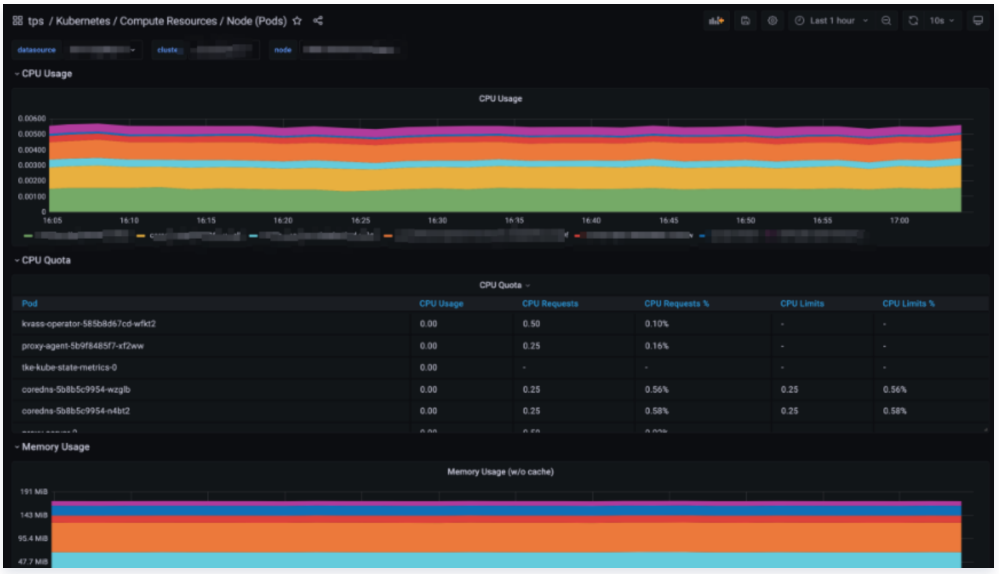



## 步骤3：Grafana 查看监控数据

1. 在实例列表中单击实例名称右侧的 Grafana 图标，进入 Grafana 服务平台。
2. 在 Grafana 服务平台 > Dashboard 搜索列表，默认预设了容器相关的监控面板，单击某个面板名称。




进入面板页面，可以查看预设好的监控数据图表。




### 步骤4：配置告警策略

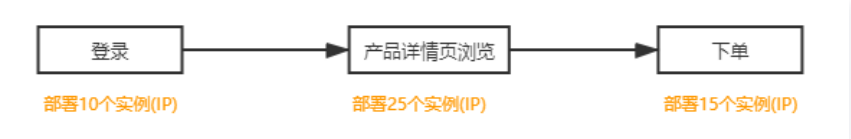
在实例列表中，选择对应的 Prometheus 实例，单击顶部菜单栏的**告警管理**，进入告警策略页面，单击**新建告警策略**，详情请参见 新建告警策略 。

# 基于 Prometheus 多维能力的告警优化

最近更新时间：2024-09-09 22:00:41

## 实践背景

通常来说，监控系统的四个黄金指标（Four Golden Signals，参考 Google 运维解密）是错误类指标、延迟类指标、流量指标和饱和度指标，可以在服务级别衡量终端用户体验、服务质量、业务影响等层面的问题。以一个典型的电商服务关键路径（登录 > 产品浏览详情页 > 下单）举例。




针对图中三个服务，需要设定成功率等监控指标。传统的以实例(例如 IP )为监控对象的场景下，通常会对每个服务的实例都配置告警策略，用于满足最细粒度的告警对象质量监测，但也会带来一系列痛点：

- 需要针对每个实例都配置一条告警策略，告警策略太多，维护成本高。
- 缺乏整个业务层面的监控(监控对象是3个业务，而不是每个实例)。
- 业务异常时，多个实例同时触发告警，容易引起告警风暴。
- 对于其他维度的业务质量监控能力不足，例如业务分区部署，需要看某个区的质量情况等。

针对上述痛点，Prometheus 监控服务可以提供“优雅”的解决方案。通过将监控对象从实例扩展为不同标签（label），利用Prometheus 监控服务的多维能力，可以聚合出服务层面的宏观监控指标。详见下文。

## 监控场景

**服务指标的宏观维度监控&告警场景**

以上述电商服务关键路径举例。服务上报指标为成功率，上报标签为：服务名、IP 和区域。


相比传统只上报实例（IP）的单一标签，这里还扩展了服务名等其他标签。

在设置告警策略时，通过不同标签聚合，来解决上述提到的传统实例监控中碰到的痛点。具体步骤如下：

1. 按服务维度的成功率看整体情况。
2. 按服务&地区维度聚合，查看某个更细维度的质量 情况。
3. 利用 PromQL 可以大幅降低告警策略数量(对比传统实例监控)。

## 实践步骤

### 步骤1：指标定义&服务部署

**1. 指标定义**

定义服务需要上报的指标和标签。举例，这里上报指标为：成功率(svcSuccessRate)。标签为：服务名(svc)，IP(ip)，区域(region)。

```
svcSuccessRate = prometheus.NewGaugeVec(
    prometheus.GaugeOpts{
        Name: "svcSuccessRate",
        Help:    "service success rate",
    },
    []string{"svc", "region", "ip"})
```

**2. 服务部署**

服务可以通过云服务器(CVM)的形式部署在云上(本文采用 CVM 的部署形式)，也可以通过容器(TKE)的形式部署。
部署到 CVM 上，并检查是否正常暴露指标(示例中通过8581端口暴露指标)。

```
svcSuccessRate{ip="[redacted]",region="ap-shanghai",svc="svc.proDtl"} 0.9535
svcSuccessRate{ip="[redacted]",region="ap-singapore",svc="svc.deal"} 0.987
svcSuccessRate{ip="[redacted]",region="ap-singapore",svc="svc.login"} 0.94
svcSuccessRate{ip="[redacted]",region="ap-singapore",svc="svc.proDtl"} 0.9242
[root@VM-16-14-tlinux /usr/local/testEsen]# curl http://localhost:8581/metrics
```

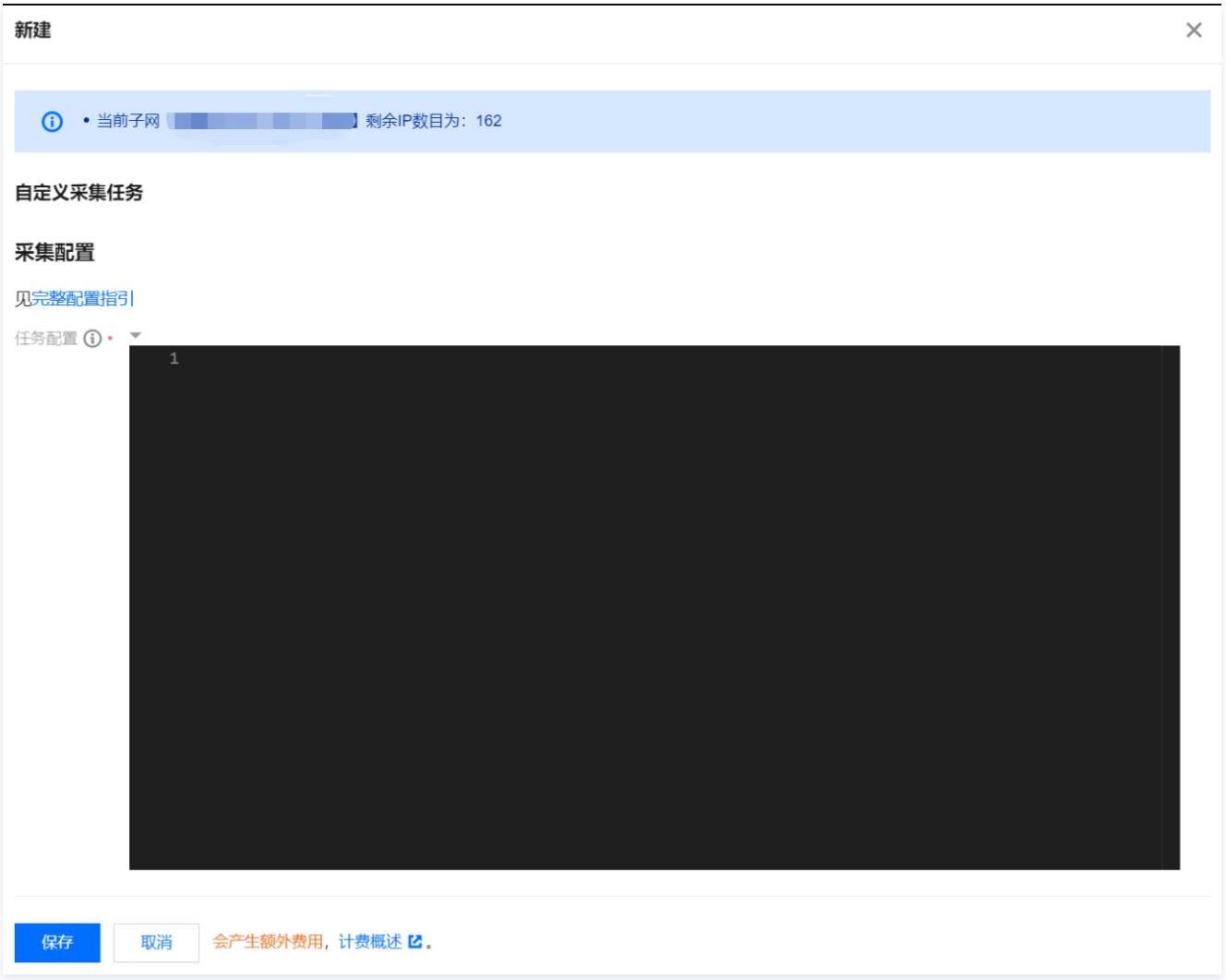
**说明：**

- Prometheus go client 提供了完善的指标接入库，详情请查看 官方文档 。
- 更多语言 SDK 可参考 自定义监控 。

## 步骤2：配置抓取任务

1. 登录 Prometheus 监控服务控制台，点击**新建**，创建 Prometheus 实例。
2. 点击新建的 Prometheus 实例，进入实例详情页，点击**数据采集 > 集成中心**，搜索“抓取任务”，再点击**一键安装**，如下图：


3. 安装成功后再进行配置，详细配置信息请参考 cvm_sd_config 配置 。




4. 验证数据是否抓取成功。通过 Grafana 查询对应的指标和标签。


## 步骤3：配置告警策略

1. 配置服务宏观层面的告警：当服务的成功率低于两个9(0.99)时告警。


效果如下图所示，当上报的三个服务(登录，查看产品详情页，下单)成功率低于0.99时，就会触发告警。


2. 配置服务更细维度的告警：当服务某个区域的成功率低于0.99时，触发告警。


效果如下图所示，当上报的三个服务(登录，查看产品详情页，下单)在某个地区的成功率低于0.99时，就会触发告警。

# Prometheus 实例访问公网

最近更新时间：2024-11-11 10:15:12

## 操作背景

在一些场景下，您可能需要您的 Prometheus 实例具备访问公网的能力，例如：

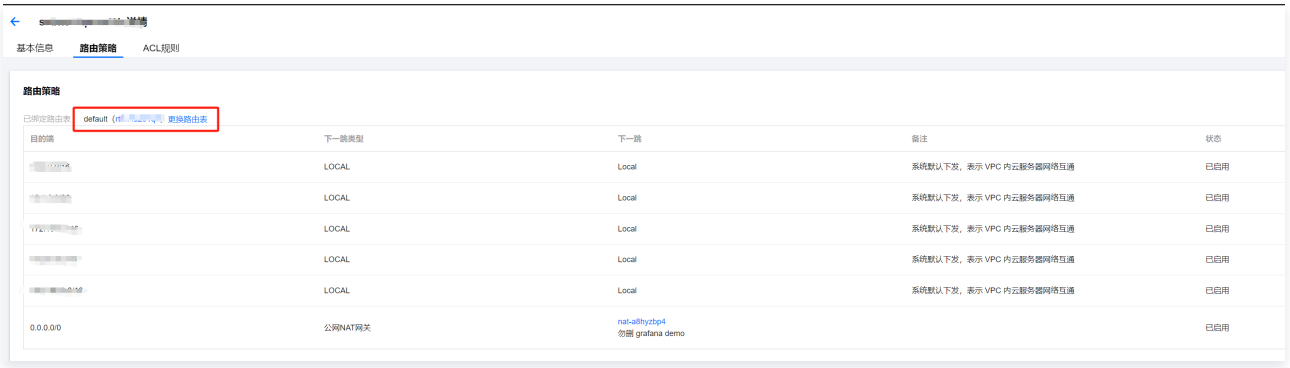
- Prometheus 集成对云上 COS 产品（对象存储）的监控时，由于 COS 产品不支持内网访问，所以需要 Prometheus 实例具备访问公网的能力。
- 使用 Prometheus 告警策略里的 webhook，而该 webhook 为公网地址时。

在这类情形下，您需要为 Prometheus 开通公网访问（底层是为 Prometheus 实例组件所在的 TKE 托管集群开通公网访问）。

## 操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 单击对应的实例，进入实例管理页，单击基本信息页的**所属子网**。


3. 在子网顶部菜单，切换为**路由策略**栏。单击路由表链接（列表上的 rtb-xxx）进入路由表界面。




4. 在路由表界面单击**新增路由策略**。
   - **目的端**：填写 0.0.0.0/0。
   - **下一跳类型**：选择 NAT 网关。
   - **下一跳**：选择对应的网关，若无网关，请参见 NAT 网关 指引进行创建 。


5. 填写完成后，单击**创建**。创建成功后表示 TKE Serverless 开通外网访问。

# 配置 Prometheus 公网地址

最近更新时间：2024-12-24 12:16:11

本文将为您介绍如何为 Prometheus 监控配置公网地址。

## 操作步骤

### 步骤1：购买 Prometheus 实例

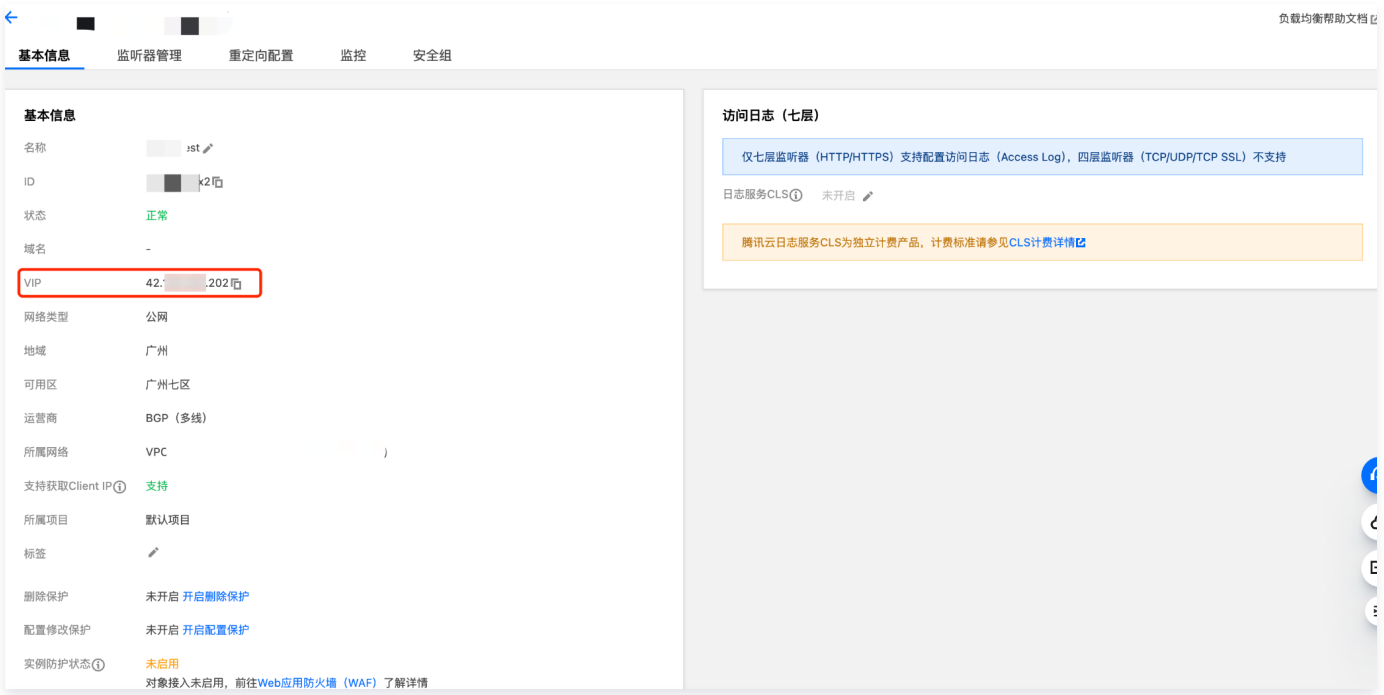
1. 登录 Prometheus 监控服务控制台 。
2. 单击左上角的**新建**，进入 Prometheus 购买页，可根据自己的实际情况购买对应的实例，详情请参见 创建实例 。
3. 成功购买后，单击创建的实例 ID/名称，进入实例详情页的**基本信息**，获取 Prometheus **所属网络**以及**IPV4 地址**。


### 步骤2：新建公网 CLB 实例

1. 进入 负载均衡控制台，新建 CLB 实例。
2. 根据提示选择并填写信息，具体操作说明可参见 创建负载均衡实例 。网络类型选择公网，来开通 CLB 公网访问。所属网络选择 步骤1 获取的 Prometheus 所属网络。


> **说明：**
> 需创建与 Prometheus 同 VPC 下的 CLB 实例，才可绑定 Prometheus 内网 IP。若已有公网的 CLB 实例且与 Prometheus同 VPC，可复用已有公网 CLB。

3. 创建完后进入实例基本信息页，可查看 CLB 对应的公网 IP。

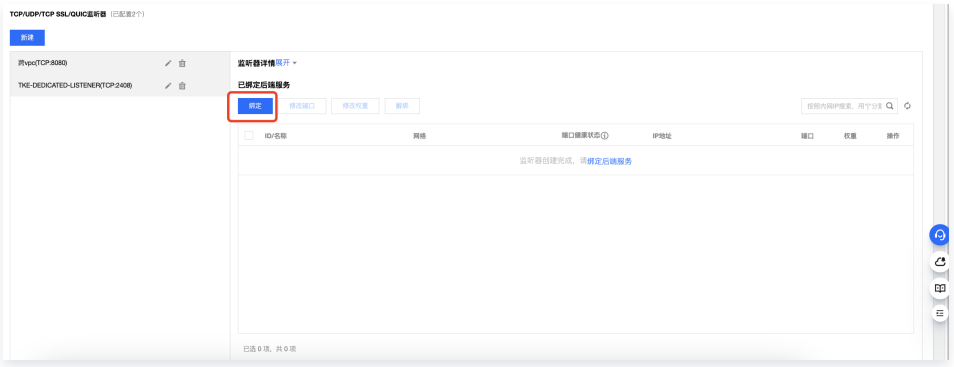



## 步骤3：绑定后端服务

1. 进入**监听器管理**页面。
2. 单击 TCP/UDP/TCP SSL/QUIC 监听器下的**新建**。


可参见 **负载均衡监听器** 操作指引新建监听器。


3. 新建完监听器后，单击监听器名称。在子窗口中单击**绑定**，绑定后端服务。

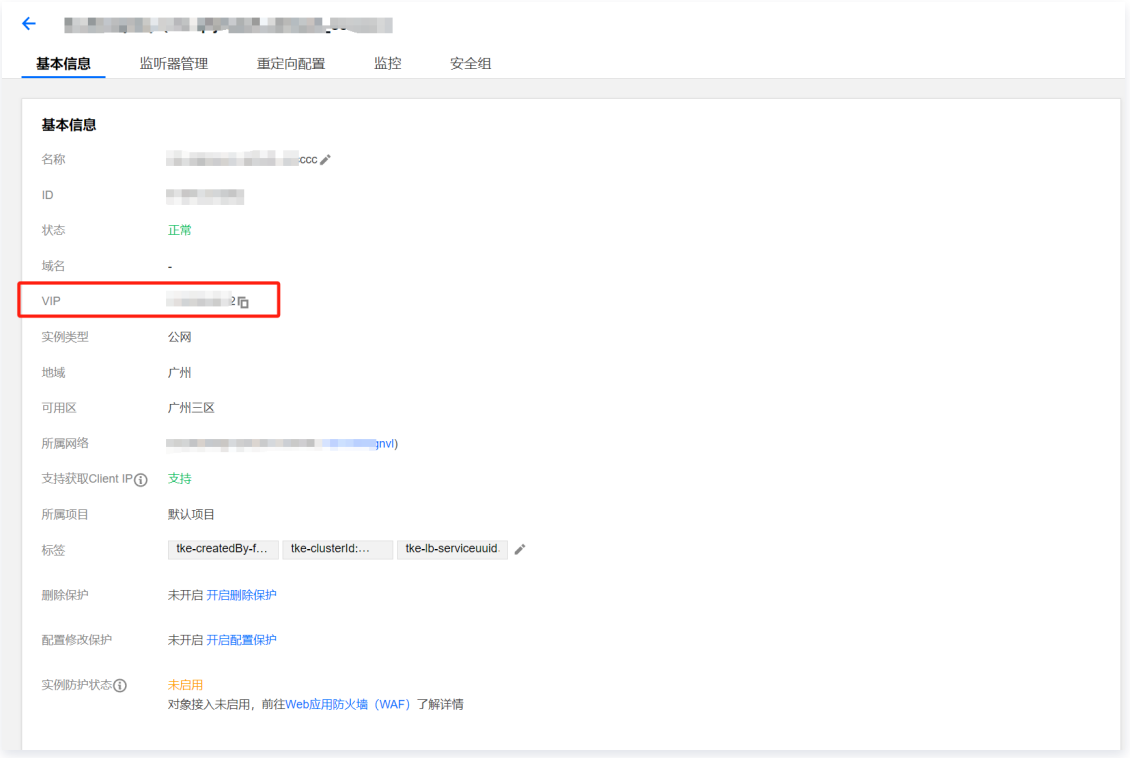



- **目标类型：选择 IP 类型。**
- **IP 地址：需填写步骤1获取的 Prometheus IPV4 地址。**
- **端口：填写 Prometheus 默认开放端口9090。**


4. 单击**监听器名称**，查看是否监听正常。

### 步骤4：测试是否配置成功

1. 查看公网 CLB 的地址，这里假设查看的地址为：192.168.1.1 。




2. 查看监听配置端口。如下列端口为：8080。


根据上述两个信息，确定 Prometheus 转发的公网地址为如下新 IP:PORT 地址为： `192.168.1.1:8080` 。

3. 到浏览器或者机器上查看是否可以通过这个 IP 获取 UP 数据。
   HTTP API 地址：

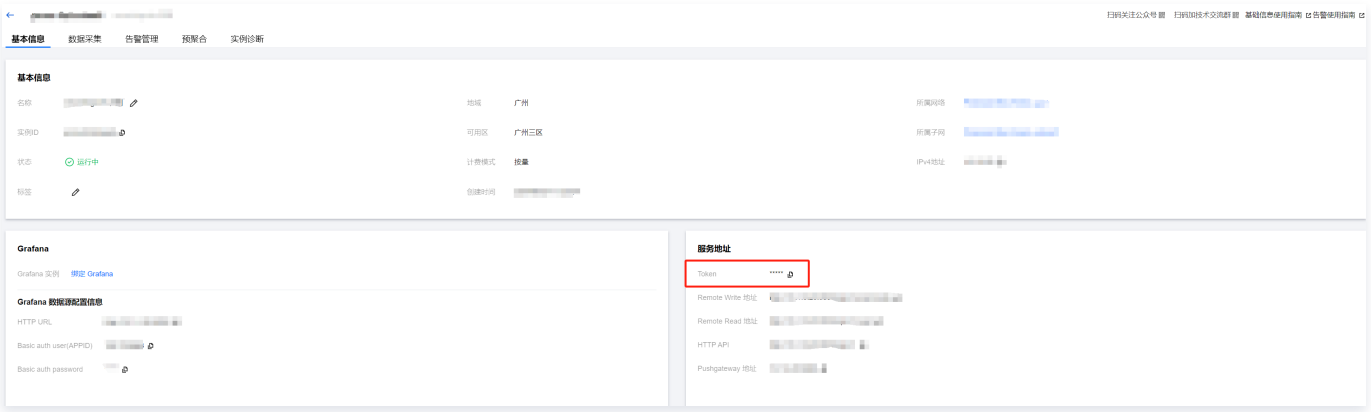
```
http://IP:PORT/api/v1/query?query=up
```

   用对应 CLB 的**公网 IP 和端口** 替换 IP:PORT 后如下：

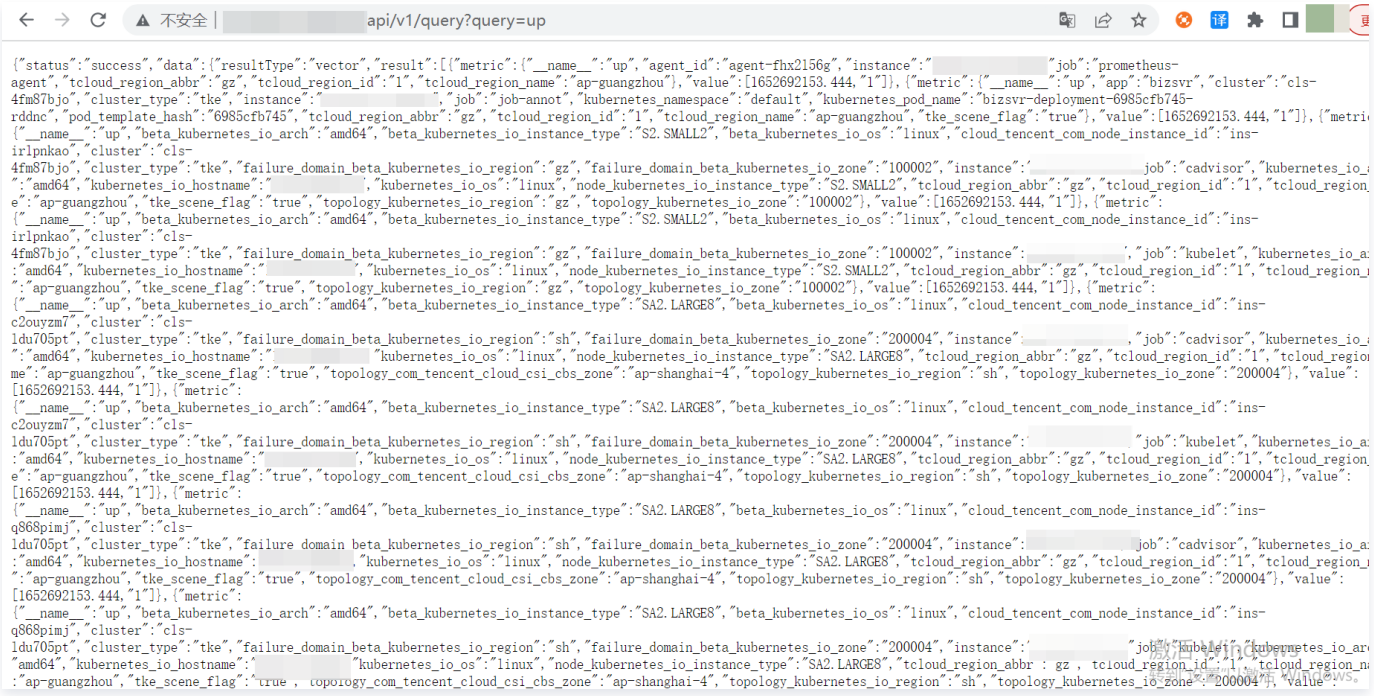
```
http://81.71.21.123:8080/api/v1/query?query=up
```

4. 进入链接地址 `http://81.71.21.123:8080/api/v1/query?query=up` 。

- 用户名：填写您的主账号 ID (APPID)。
- 密码：在实例的基本信息页面获取 Token。




如下图所示，Prometheus 配置公网地址成功。



# Prometheus 监控服务如何接入本地 Grafana

最近更新时间：2024-08-16 11:52:01

若您需要在本地的 Grafana 系统中查看 Prometheus 监控服务相关数据，可以利用 Prometheus 监控服务提供的 HTTP API 来实现。本文介绍如何将 Prometheus 监控服务数据接入本地 Grafana 的实现方法。

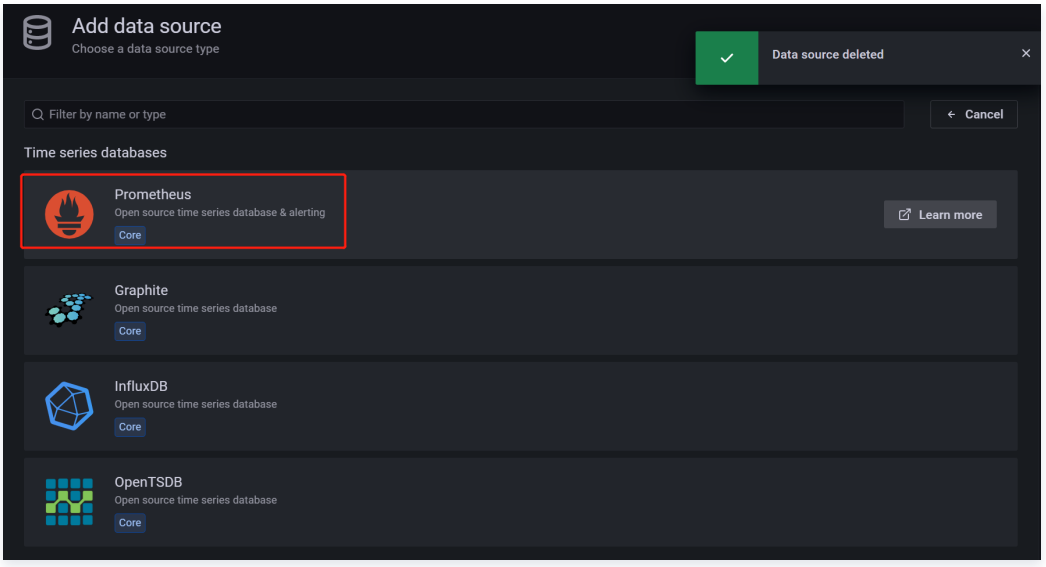
## 操作步骤

### 步骤1：获取 Prometheus 监控服务提供的 HTTP API

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**。
3. 单击对应的按量付费实例，进入 Prometheus 基本信息页。
4. 在服务地址模块获取 HTTP API 地址。若您需要提高 Grafana 数据读取的安全性，可获取 Prometheus 实例的鉴权 Token，按照**步骤2**的指引填入即可。

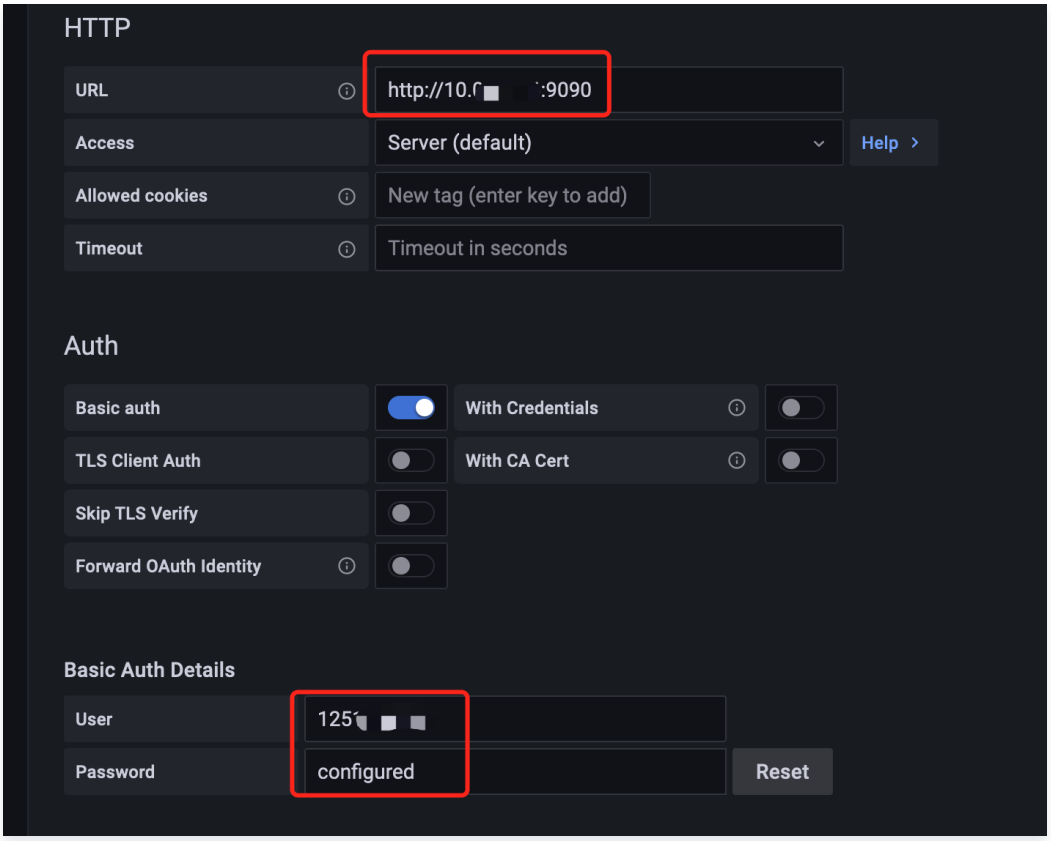

### 步骤2：在本地 Grafana 添加数据源

1. 以**管理员账号**登录本地 Grafana 系统。
2. 在左侧导航栏中选择 **Configuration > Data Sources**（非管理员无法查看该菜单）。
3. 在 Data Sources页中单击 **Add data source**。
4. 在 Add data source 页面中选择 **Prometheus**。




5. 在 Settings 页签的 Name 字段输入自定义的名称，在 URL 字段中粘贴步骤1：「Grafana 数据源配置信息」中的 **HTTP API**。
6. 在 Auth 区域开启 Basic Auth 开关，设置 User 为「Grafana 数据源配置信息」中的 **Basic auth user**，设置 Password 为「Grafana 数据源配置信息」中的 **Basic auth password**。

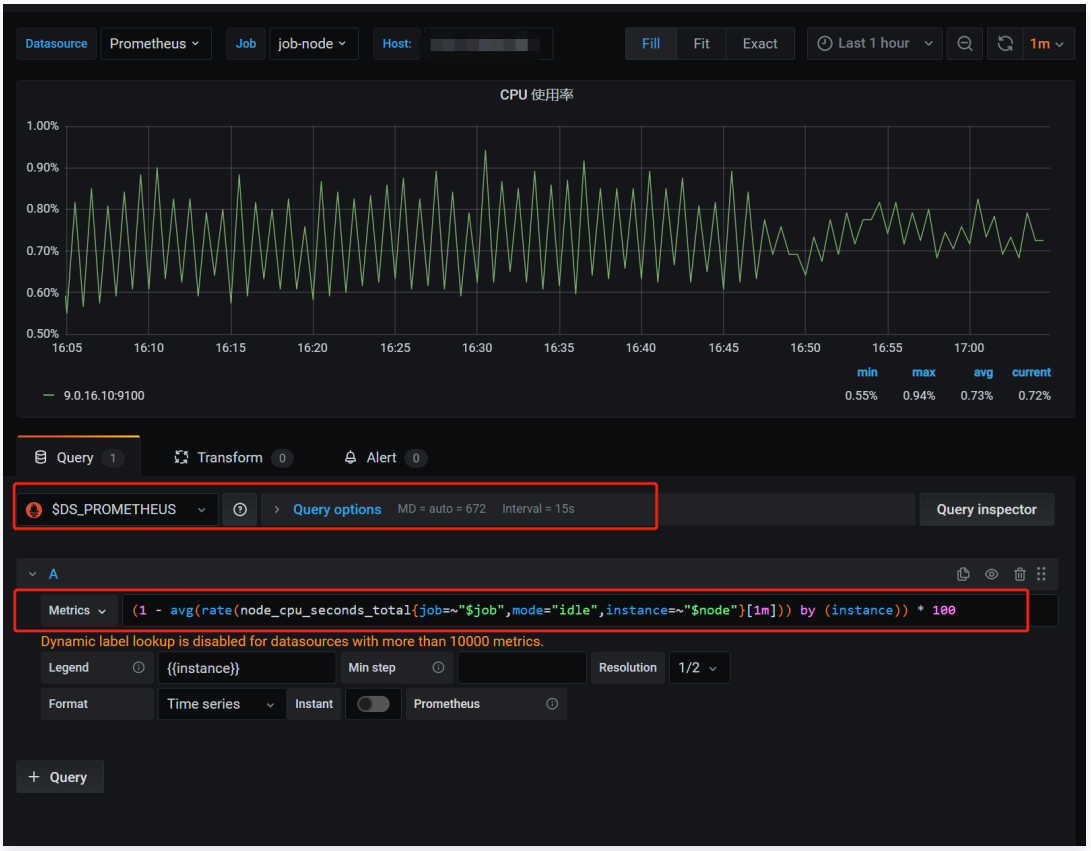
7. 完成后单击 **save & test** 即可。




## 步骤3：验证是否添加成功

请按照以下步骤验证 Prometheus 监控服务是否成功接入本地 Grafana：

1. 登录您本地 Grafana 系统。
2. 在左侧导航栏中选择 > **New dashboard** 。
3. 在 New dashboard 页面中单击 **Add a new panel** 。
4. 在 Edit Panel 页面的 Query 页签的下拉框中，选择步骤2中所添加的数据源（如下图红框1处）。
5. 在 A 区域的 Metrics 字段输入指标名称并按回车。
6. 若能显示出相应指标的图表数据，则说明操作成功。否则请检查填写的接口地址或 Token 是否正确，以及步骤2中添加的数据源是否有 Prometheus 监控服务数据。


Datasource
Prometheus
Job
job-node
Host:
Fill
Fit
Exact
Last 1 hour
1m
CPU 使用率
1.00%
0.90%
0.80%
0.70%
0.60%
0.50%
16:05
16:10
16:15
16:20
16:25
16:30
16:35
16:40
16:45
16:50
16:55
17:00
min
max
avg
current
9.0.16.10:9100
0.55%
0.94%
0.73%
0.72%
Query 1
Transform 0
Alert 0
$DS_PROMETHEUS
Query options
MD = auto = 672
Interval = 15s
Query inspector
A
Metrics
(1 - avg(rate(node_cpu_seconds_total{job=~"$job",mode="idle",instance=~"$node"}[1m])) by (instance)) * 100
Dynamic label lookup is disabled for datasources with more than 10000 metrics.
Legend
{{instance}}
Min step
Resolution
1/2
Format
Time series
Instant
Prometheus
+ Query

# 使用实例诊断分析 Prometheus 问题

最近更新时间：2024-10-25 18:14:02

为了提升用户在使用 Prometheus 监控采集功能时的体验，Prometheus 监控团队特推出了新的采集架构，且为用户提供了名为**实例诊断**的控制台页面。此功能有助于用户对当前 Prometheus 实例采集和存储功能的运行情况有更加细致的了解，并在出现问题时可以更加快速地定位和排查。

> **说明：**
> 若用户名下有存量的 Prometheus 实例，则需要先将其升级到新的采集架构再来使用实例诊断的功能。详情可参考 实例诊断 。

## 问题一：采集组件升级

### 现象

实例诊断页面中，提示采集组件有新版本可升级。


### 解释和处理

Prometheus 采集端主要包含四个组件，用来实现采集目标的发现和调度、采集操作的进行、采集集群和用户集群之间的通信等功能：

- **tmp-operator**：负责采集目标的发现和调度，以及采集分片 tmp-agent 的安装和配置生成，并以集群化的形式管理所有采集分片。
- **tmp-agent**：负责实施实际的采集操作，并将处理后的指标推送到指定的远端地址。
- **proxy-server**：为采集组件访问用户集群提供代理功能，是通过多个端口实现多路代理的七层代理器，支持多个 agent 连接同一个 server。
- **proxy-agent**：与 proxy-server 共同提供完整的代理功能，该组件安装在用户集群中。

可以在**实例诊断**页面查看各个组件的当前版本、功能描述以及不同版本的更新信息。在版本的更新中，会对组件可能存在的问题进行处理修复。因此，用户使用的过程中可以尽量保持组件升级到最新的版本。当实例诊断中有新版本提示时，可根据自身需要对采集组件进行升级。其中 tmp-operator、tmp-agent 和 proxy-agent 的升级不会影响数据采集，即不会产生数据断点；而 proxy-server 的升级则可能会出现数据断点，具体的恢复时间和底层集群 Pod 的调度时间有关，需要谨慎升级。


## 问题二：网络异常

### 现象

- 在查询 Prometheus 指标时陆续出现断点；
- 在控制台上显示采集目标一直因为 timeout 而处于 down 的状态；
- 在实例诊断页面则可能会显示 proxy-server 和 proxy-agent 的连接异常。

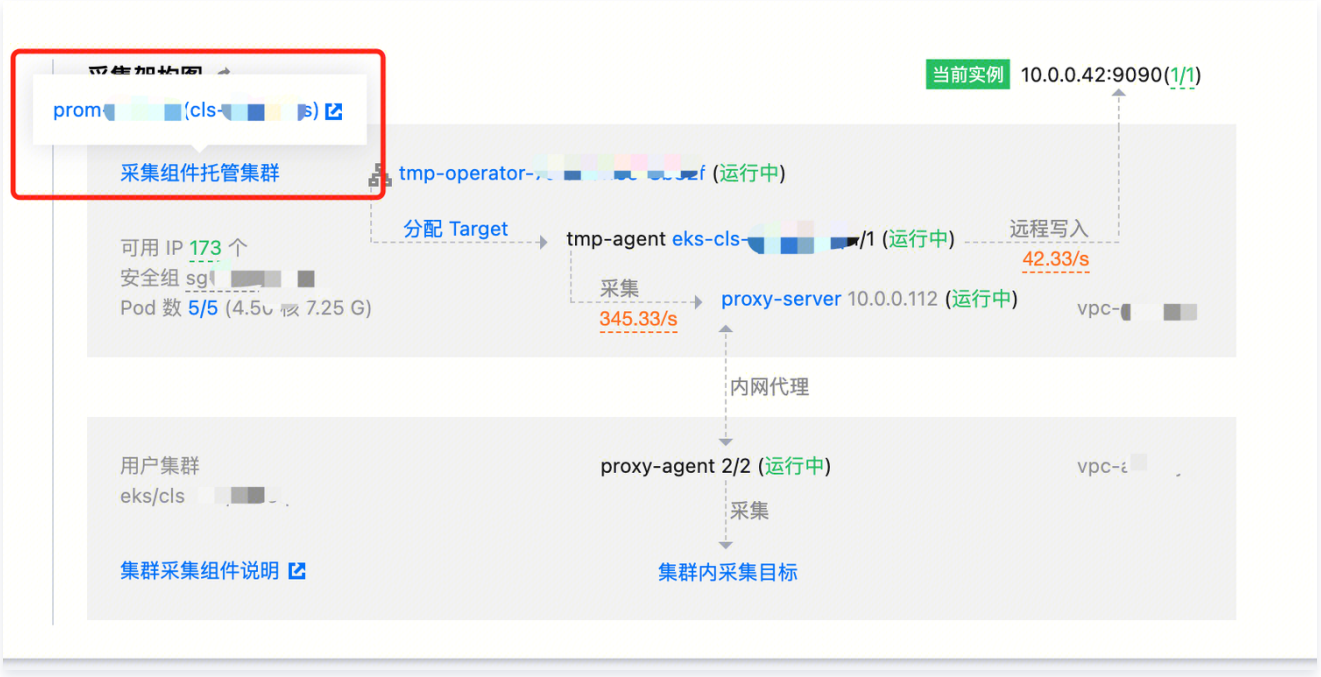

### 解释和处理

Prometheus 采集涉及到集群内、跨集群、跨 VPC 的通信，不同对象之间的网络异常可能会导致通信异常，使得表现出来采集功能也存在异常。


这种情况，通常是带宽问题或安全组问题，特别是带宽问题。当链路上某一节点的带宽占满之后，请求或响应在该节点被阻塞，最终导致请求或响应超时。

1. 在跨 VPC 采集的情况下，如果使用的是公网 CLB，那么可以优先排查公网 CLB 的链路是否通畅、带宽是否占满；如果使用的是云联网连接两个集群，那么可以优先排查云联网的链路是否通畅、带宽是否占满。


2. 在排除跨 VPC 通信的问题后，可以对采集组件 proxy-server 的带宽是否占满进行排查（进入采集集群，并在 Pod 中查看监控即可）。




3. 如果已排除带宽的问题，那么可能是安全组的问题。安全组未放通一般在集群关联时就会暴露出来，但不排除后续被修改的可能，被安全组阻挡的请求就无法到达目标 Pod，需要按照实例诊断中的安全组要求进行配置。


## 问题三： 数据多写堆积

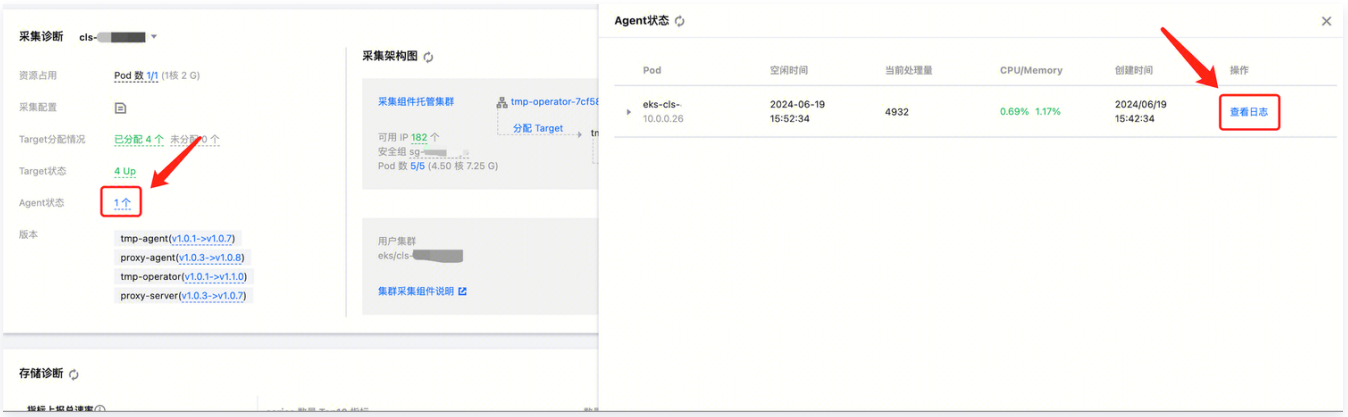
### 现象

- 用户短信、站内信等收到数据多写配置异常的告警；
- 在实例诊断页面显示数据多写异常堆积。


## 解释和处理

数据多写功能支持将集成容器服务和集成中心的数据写到自建 Prometheus 或其他 Prometheus 监控实例中。如果多写异常，说明采集分片在向目标地址推送指标的时候出现了问题；采集分片如果推送失败，会将指标保存在本地进行重试，如果数据堆积量过大，还可能导致 Pod 异常影响指标采集。
当出现这样的问题，可参考如下步骤排查：

1. 在实例诊断中确认具体的目标地址，短信告警中没有具体的地址；
2. 查看对应的 Prometheus 是否可以正常写入、是否有异常的日志，以排除目标 Prometheus 的问题；
3. 可以查看采集分片的日志，是否存在报错，以及分析具体报错的信息。





## 问题四：可用 IP 数量不足

### 现象

实例诊断页面提示 IP 数量不足（红色并闪烁）。

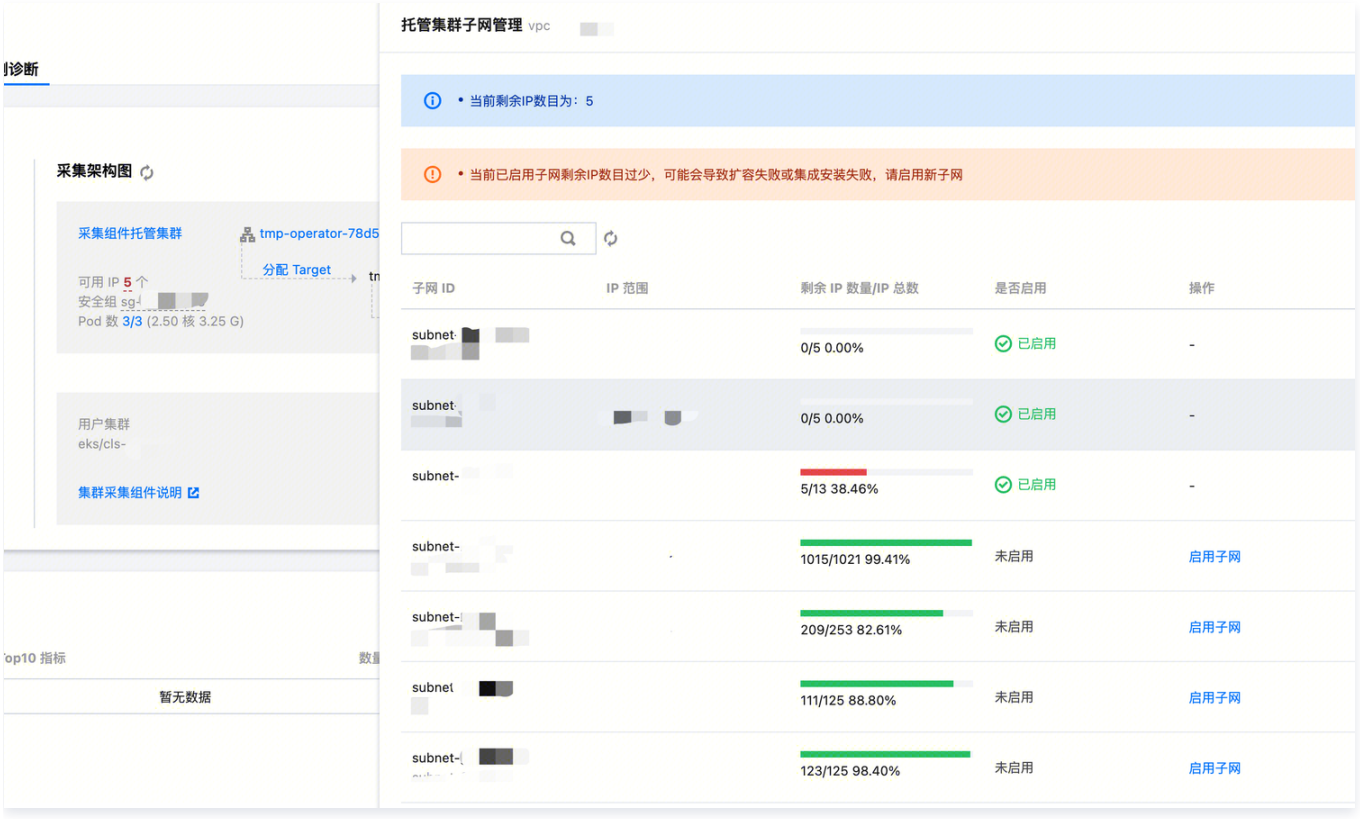

## 解释和处理

采集组件托管集群所关联的子网可用 IP 数量不足，可能会导致如下问题：

- 当采集规模不断增大，达到单个采集分片无法承受时，调度器会对采集群片进行扩容，扩容出来的新分片 Pod 会占用子网内部的 IP；如果可用 IP 数量不足，会出现无法扩容的情况，已有的采集分片持续超负荷运转，会影响采集的稳定性。
- 当进行采集组件升级时，部分组件会先产生新的正常运行的副本，再替换旧的副本；如果可用 IP 数量不足，会出现新的副本无法调度成功，导致组件升级异常。

因此，建议始终保持一定数量的可用 IP。当出现上述实例诊断页面显示可用 IP 数量较少时，可以在页面中直接添加子网，具体操作如下：

1. 点击子网数字，进入**托管集群子网管理**页面；
2. 显示当前 VPC 内已添加到托管集群的子网和未添加的子网，可根据规划点击启用子网启用新的子网；
3. 若没有可用子网，可以先 添加子网 后再在当前页面中启用。



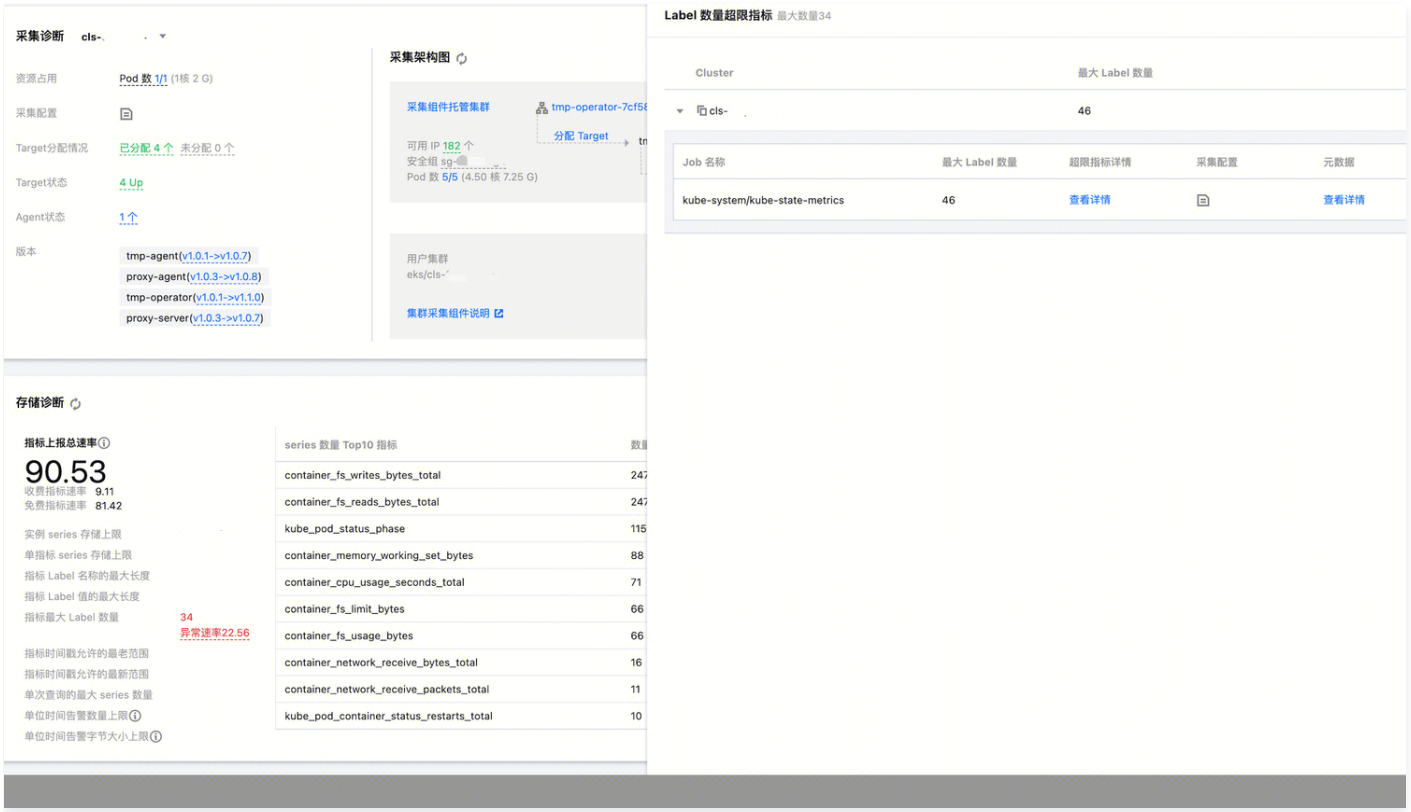
## 问题五：指标标签数量过大导致写入异常

### 现象

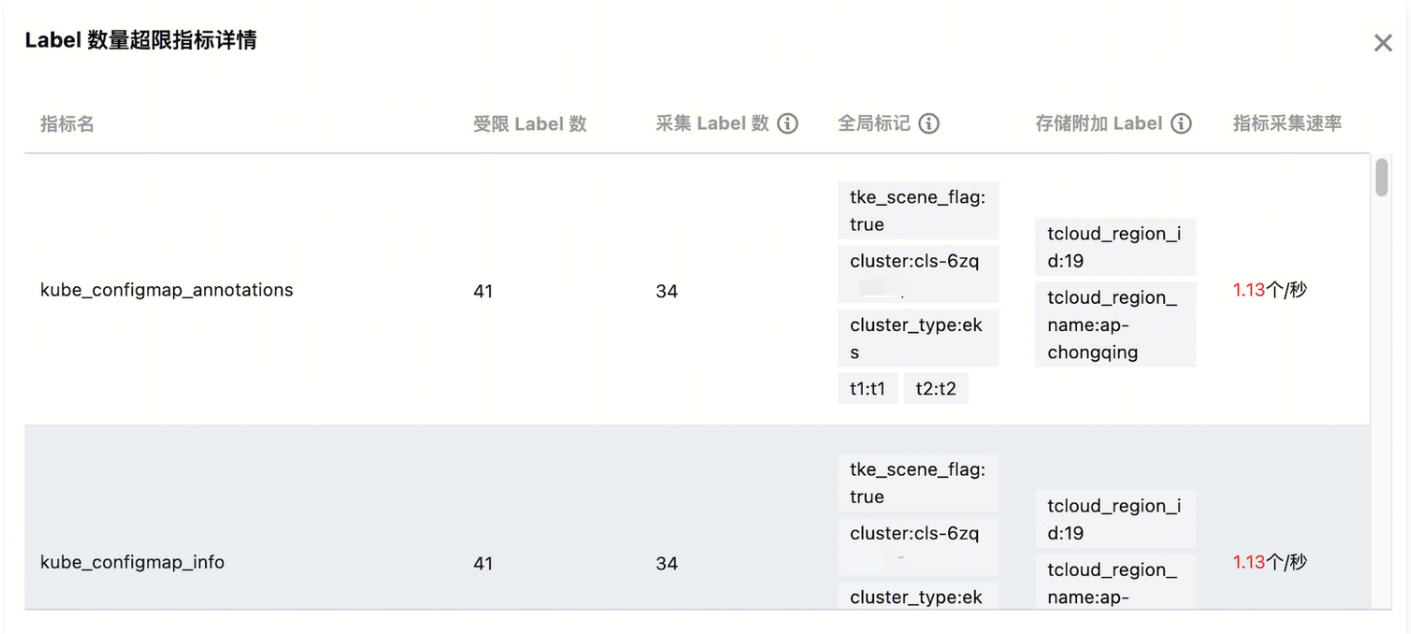
- 存储诊断中，**指标最大 Label 数量**显示**异常速率**；
- 同时，Prometheus 实例**基本信息**页面的实例监控中，“因不合法被丢弃的 Samples”图标中显示有数据被丢弃；
- 在查询时有部分预期的指标查不出来。


### 解释和处理

**指标最大 Label 数量**代表写入指标标签的最大数量 限制，如果使用过程中，在指标中加入大量无用标签，就可能会超过限制导致需要的指标写入失败，出现上述的异常提示。
在存储诊断中点击**异常速率**弹出窗口时，可以看到对应目标集群内尝试写入存储的**最大 Label 数量**，以及对应不同采集任务 Job 的详细信息。

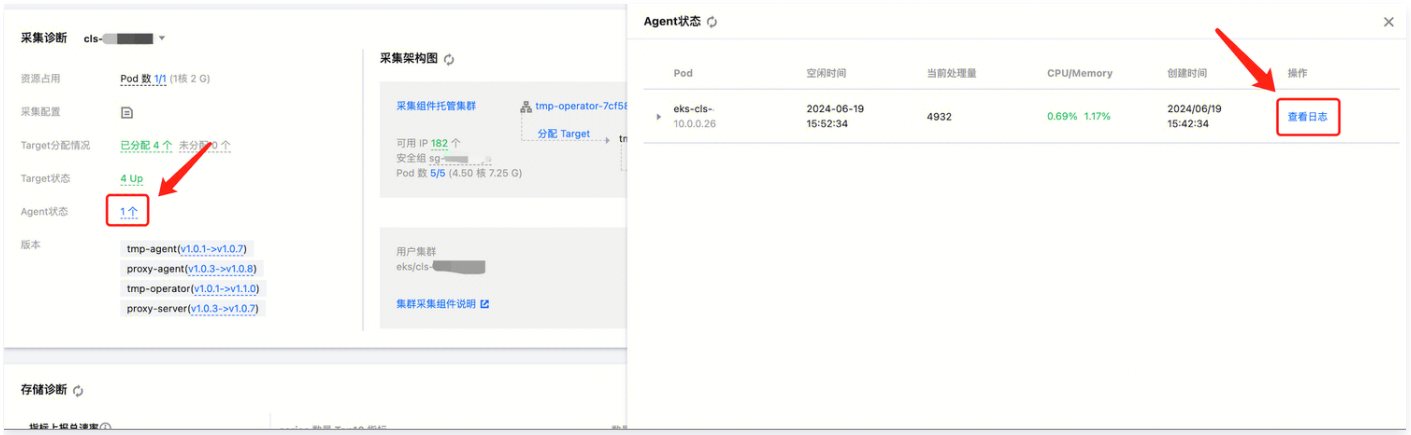


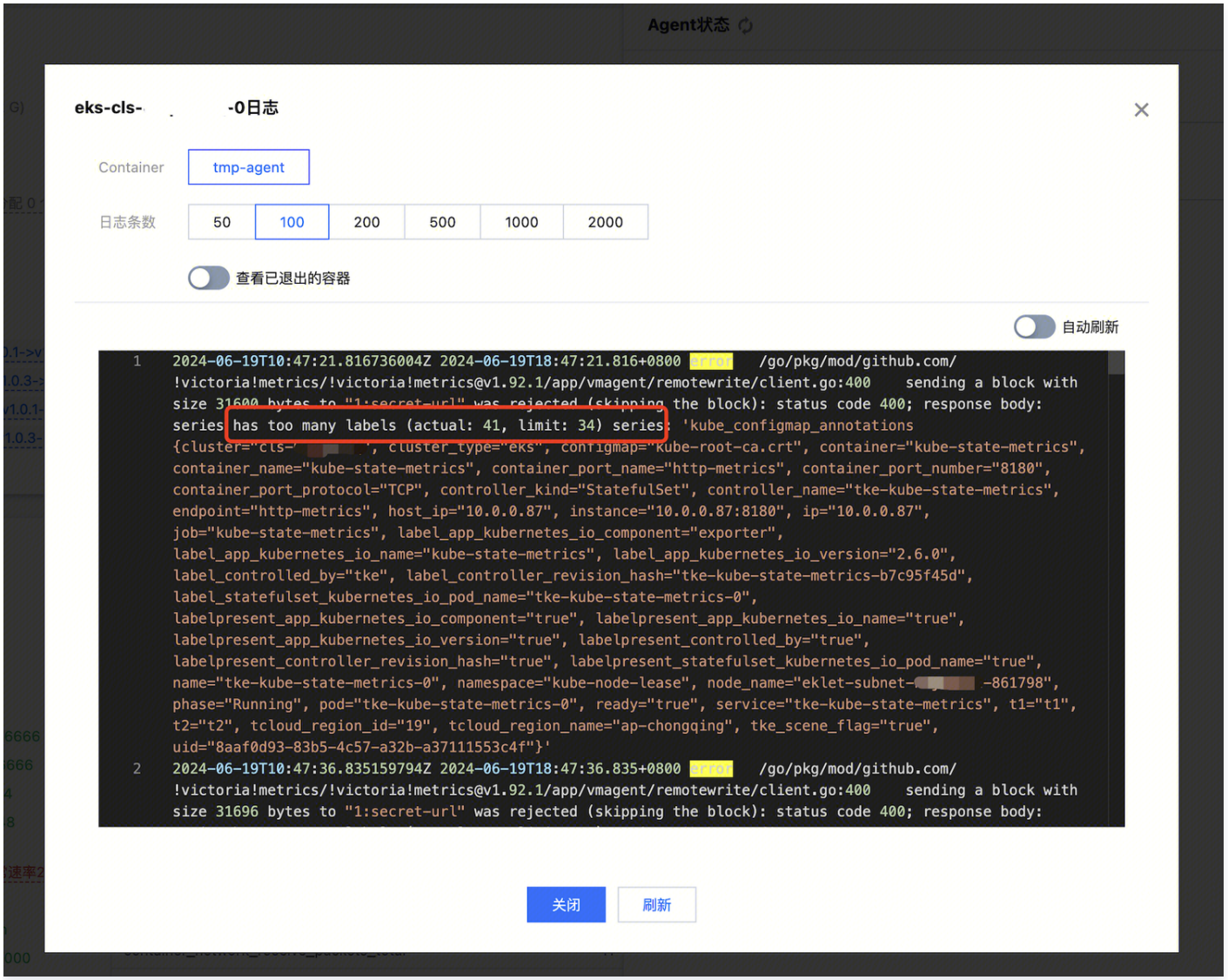

在 **Label 数量超限指标详情**中，可以看到对应指标的总 Label 数量，以及采集 Label 数、全局标记和存储附加 Label 的信息。




- **采集 Label 数**：根据采集配置，对采集目标服务发现的标签进行 relabel 之后的标签，加上指标本身的标签之后的标签数量，不包括全局标记以及存储附加标签。
- **全局标记**：用户额外配置的，在当前集群所有采集任务上都会添加的标签；默认情况下会在关联集群的时候，添加包含集群信息的“cluster_type”和“cluster”全局标记；
- **存储附加 Label**：由存储侧默认写入的标签，无法修改，用于表示存储本身的信息。

因此，要解决指标标签超限的问题，可以从**采集 Label 数**和**全局标记**两个方面处理。处理之前，可以通过查看日志了解被丢弃的指标在写入之前包含的标签信息，在采集分片的日志中进行查看，并确认需要丢弃哪些标签。




- 对于采集 Label 数，可以在采集配置中的 metric_relabel_configs 或 metricRelabelings 中增加 labeldrop 配置来丢弃掉。

```
metricRelabelings:
- action: labeldrop
regex: label_(.+) # 正则匹配需要去掉的标签
```

- 对于全局标记，可以在控制台的数据采集中进行编辑，有不需要的全局标记可以在这里去掉，不过需要注意的是，全局标记的变更会影响到整个集群的采集。


## 问题六：指标数量达到存储上限导致写入异常

### 现象

- 存储诊断中，“实例 series 存储上限”或“单指标 series 存储上限”显示“异常速率”。
- 同时，Prometheus 实例**基本信息**页面的实例监控中，“因限流丢弃的 samples”图表中显示有 Samples 被丢弃；在查询时有部分预期的指标查不出来。

### 解释和处理

在存储诊断中，series 存储上限分别代表了整个 Prometheus 实例和单个指标的 series 存储 上限，当存储中的 series 达到上限之后，会出现新的 series 无法写入的问题，详情如下。（图中的存储上限仅供演示）


#### 单指标存储上限

通常，单个指标 series 数量庞大，是因为某些标签值变化过于频繁，例如将用户的 id、时间戳等作为标签值，或许这在日志收集中很常见，但对于 Prometheus 来说，这种使用方式造成的高基数问题，会严重影响 Prometheus 系统的性能。
解决该问题，即解决标签值频繁变化的情况。在存储诊断中，可以看到“series 数量 Top10 指标”，当出现异常的时候，这里的数值和“单指标 series 存储上限”是对应的。


只要找到了对应指标，看是具体哪个标签有问题，在业务暴露指标中去掉这个标签，或是在采集配置的 metric_relabel_configs（原生 Job 配置）或 metricRelabelings（servicemonitor、podmonitor 等 crd，以及集成中心的云监控集成）将该标签 labeldrop 即可。

```
metricRelabelings:
- action: labeldrop
regex: label_(.+) # 正则匹配需要去掉的标签
metric_relabel_configs:
- action: labeldrop
regex: label_(.+) # 正则匹配需要去掉的标签
```

### 实例存储上限

在该处显示异常，说明预期写入的 series 数量超过了单实例可以存储的数量；而绝大部分情况下，该限制已经满足普通业务量的需要了。您可从如下方面考虑：

1. 确认是否因为某些标签值变化过于频繁导致的整体存储 series 数量过多，毕竟 series 数量和费用关联紧密，如果是因为不需要的标签值变化导致 series 数量较大，最好将该标签去除；
2. 根据业务需要，去掉不需要写入的 series，精简指标体系；
3. 如果业务真的需要上报如此大量的 series，请 联系我们 。

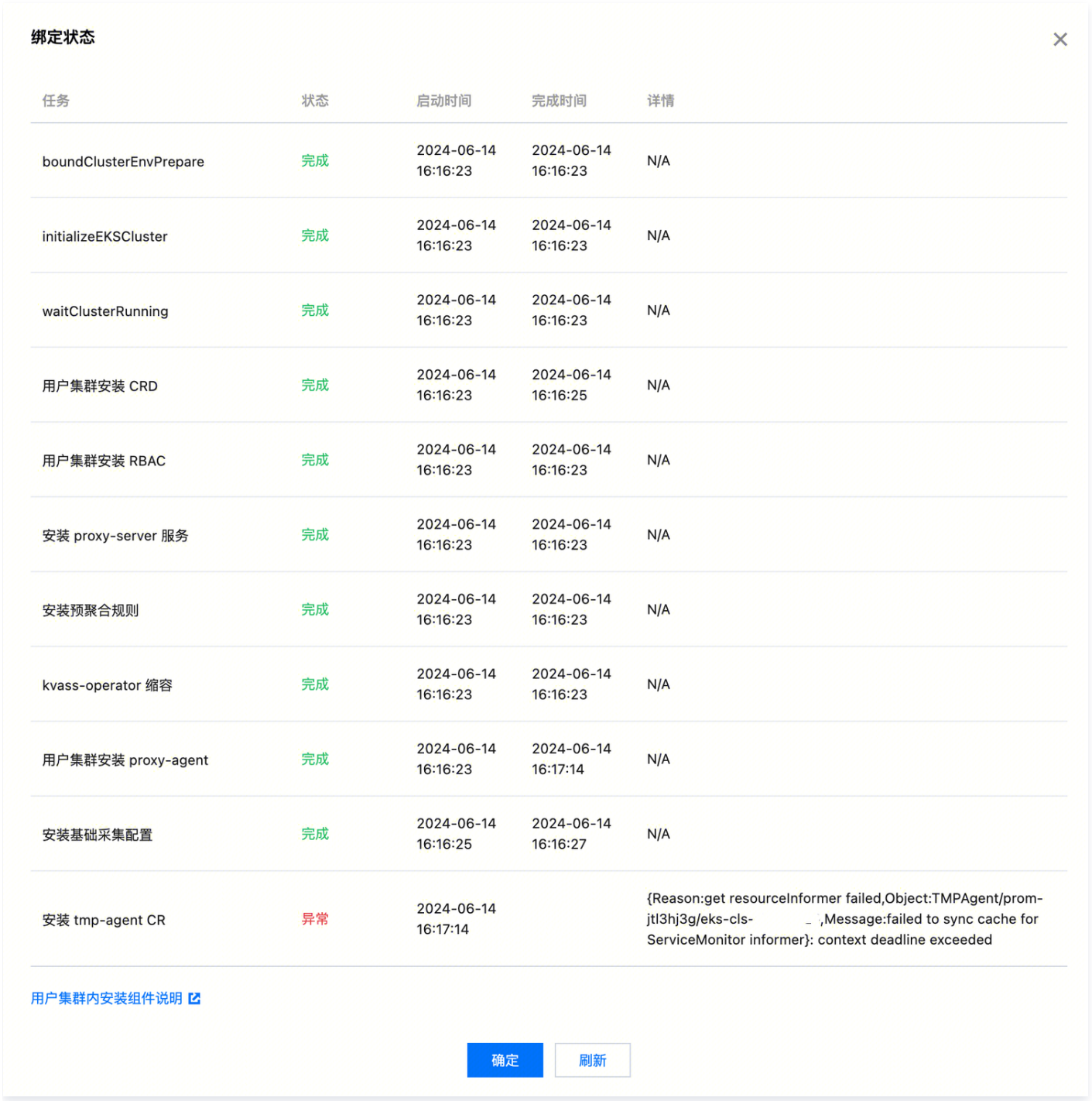
## 问题七：关联失败

### 现象

在 Prometheus 控制台 > 点击实例 ID/名称 > 数据采集 > 集成容器服务中关联集群时，偶尔会出现“状态异常”关联失败的情况；在实例诊断中也会显示“采集组件关联异常”，这个异常出现的时候采集功能是没有正常发挥作用的。


**解释和处理**

关联成功是正常使用采集容器指标的必要条件，出现关联失败问题的暴露时间点位于实际采集和存储发生之前。您可在**集成容器服务**或**实例诊断**中，查看整个关联过程的详情，包含具体出现异常的环节。



绑定状态

| 任务 | 状态 | 启动时间 | 完成时间 | 详情 |
| --- | --- | --- | --- | --- |
| boundClusterEnvPrepare | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| initializeEKSCluster | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| waitClusterRunning | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| 用户集群安装 CRD | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:25 | N/A |
| 用户集群安装 RBAC | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| 安装 proxy-server 服务 | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| 安装预聚合规则 | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| kvass-operator 缩容 | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:16:23 | N/A |
| 用户集群安装 proxy-agent | 完成 | 2024-06-14 16:16:23 | 2024-06-14 16:17:14 | N/A |
| 安装基础采集配置 | 完成 | 2024-06-14 16:16:25 | 2024-06-14 16:16:27 | N/A |
| 安装 tmp-agent CR | 异常 | 2024-06-14 16:17:14 | | {Reason:get resourceInformer failed,Object:TMPAgent/prom-jtl3hj3g/eks-cls- _ ,Message:failed to sync cache for ServiceMonitor informer}: context deadline exceeded |

用户集群内安装组件说明

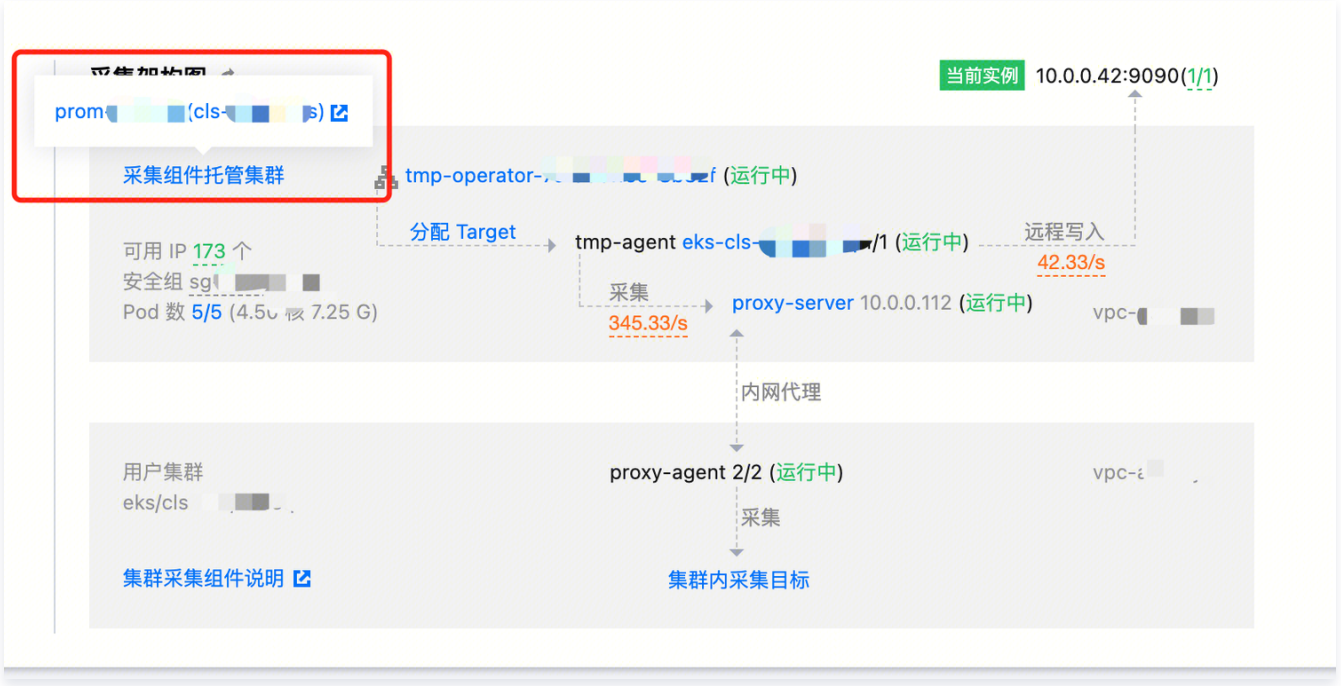
确定 刷新

常见的异常及原因通常包括：

- **集群内剩余可用 IP 不足**
  - 这在异常消息中通常会以以下形式出现：
  - “network-unavailable” 的 taint 提示，是 IP 不足时 EKS 集群自动给 node 添加的标记；
  - “insufficient IP available” 提示可用 IP 不足；
  - 其他与 IP 不足相关的提示。

**解决方法：**

- 如果出现异常的是 initializeEKSCluster 或安装 proxy-server 阶段，需要给采集集群（即同地域下名称为 Prom 实例 ID 的 Serverless 集群）添加有足够 IP 的子网对应的超级节点。




- 如果出现异常的是安装 proxy-agent 阶段，则需要给被关联的用户集群添加可用的 IP。

- **集群内资源不足：**
  - 提示“nodes are available”，没有足够的 CPU 或内存；
  - 提示“max node group size reached”，节点池无法再扩充节点。

  **解决方法：**这通常发生在目标集群内，需要您增加集群内资源后重试关联。
- **网络不通：**

  一般异常消息会包含“failed to sync cache”，由于网络问题导致采集集群无法 watch 目标集群。

  **解决方法：**可进入目标集群以当前 Prom 实例 ID 为名称的命名空间下，查看 proxy-agent 的日志，通常会有无法和对端连接的日志，如“connect to server failed”。

  如果是同 VPC，可能是 proxy-agent 的安全组限制，具体放通要求可以参考如下：
  - 入站放通 TCP：9093,9090,10901,10902,9990,3000,8080,8008,8100-8200。
  - 出站放通 TCP：ALL


如果是跨 VPC，则需要查看公网 CLB 是否正确配置。例如：如果集群是 EKS 集群是否配置 NAT 网关，或是对应的云联网是否正常等。


采集架构图
当前实例 10.0.0.42:9090(1/1)
采集组件托管集群
tmp-operator-7cf588ffc6- (运行中)
分配 Target
tmp-agent tke-cls- 1/1 (运行中)
远程写入
62.93/s
可用 IP 172 个
安全组 sg-
Pod 数 6/6 (5.50 核 9.25 G)
采集
701.07/s
proxy-server 10.0.0.112 (运行中)
vpc-
公网代理 lb-
用户集群
tke/cls-
proxy-agent (运行中)
vpc-
采集
集群采集组件说明
集群内采集目标

# Terraform
# Terraform 概述

最近更新时间：2024-07-03 12:14:41

Terraform 是一种基础设施即代码（Infrastructure as Code）工具，使用 Terraform 可以编写可重复的、可维护的基础架构代码，并通过代码进行管理和更新，简化了云基础架构的部署和管理。Terraform 提供了丰富的资源信息和功能模块，以及灵活的定制化配置选项，通过 Terraform，您可以轻松创建并管理云服务器、负载均衡等各种资源。

## Terraform 功能与优势

Terraform 是一个功能强大的开源工具，用于管理云基础架构的自动化部署和编排。

- Terraform 能够让您在腾讯云上使用简单的模板语句命令操作、构建、定义、部署和预览云基础架构。
- Terraform 非常易于使用。它使用简单的 HCL 或 JSON 配置语言来定义基础设施，几乎不需要编写代码。
- Terraform 的变量、模块和数据源特性，使得配置文件易于使用、重用和维护。
- Terraform 可以在不同的阶段输出计划报告，方便用户了解实际执行的操作。
- Terraform 可以通过使用状态文件记录基础设施的实际配置和状态信息，以及支持回滚操作。

> **说明：**
> 有关 Terraform 应用场景的具体介绍，请参见 应用场景 。

## Terraform 资源

在 Terraform 中，资源主要分为资源（Resource）和数据源（Data Source）两类：

- 资源（Resource）

  资源用于创建和管理新的云基础设施组件。通过资源定义，Terraform 可以创建、修改和删除云服务提供商支持的各种资源。例如，创建一个新的 cvm 实例。

```
#  cvm chc_assist_vpc 的资源。
# 参考文档：
https://registry.terraform.io/providers/tencentcloudstack/tencentcloud/latest/docs/resources/cvm_chc_assist_vpc
resource "tencentcloud_cvm_chc_assist_vpc" "chc_assist_vpc" {
  chc_id = "chc-xxxxx"
}
```

- 数据源（Data Source）

  数据源用于查询和获取已存在的资源信息，并将其作为输入或属性在配置中使用。数据源可以检索已经存在的云基础设施中的配置和属性信息，以便在 Terraform 配置文件中进行引用和使用。例如，查询一个已经创建 cvm chc_denied_actions 的示例如下：

```
# 使用该数据源查询 cvm chc_denied_actions 的详细信息。
# 参考文档：https://registry.terraform.io/providers/tencentcloudstack/tencentcloud/latest/docs/data-sources/cvm_chc_denied_actions
data "tencentcloud_cvm_chc_denied_actions" "chc_denied_actions" {
  chc_ids = ["chc-xxxxx"]
}
```

关于可观测监控 Prometheus 版的 Resource 的相关信息，请参见通过 Terraform使用 。

## Terraform 工具

- **Terraform CLI（命令行工具）：**

  Terraform CLI 是用于创建、管理和操作基础设施的命令行工具。它提供了一组命令和选项，让用户能够执行各种操作，如初始化配置、创建计划、应用变更等。Terraform CLI 还包括一些子命令，用于与 Terraform 相关的其他功能，如验证配置、格式化代码等。通过使用 Terraform CLI，用户可以与 Terraform 配置文件交互，并对基础设施进行管理。
- **Terraform Provider（提供者）：**

每个云厂商都提供了自己的 Terraform Provider。提供者是一个插件，用于将云供应商的功能集成到 Terraform 中。每个 Terraform Provider 负责与相应的云服务提供商的 API 进行交互，并定义可用的资源和数据源，以及执行创建、修改和删除这些资源的操作，通过配置正确的提供者，在 Terraform 配置中可以直接使用云供应商的资源和功能。

> **说明：**
> Terraform CLI 是用于管理 Terraform 的命令行工具，而 Terraform Provider 则是用于将不同云供应商的功能集成到 Terraform 中的插件。这两个部分共同协作，使得用户能够使用 Terraform 来创建和管理基础设施。关于 Terraform 的更多信息，请参见 Terraform 使用文档。

### 使用优势

- 多云方案：Terraform 适用于多云方案，能够将类似的基础架构部署到 腾讯云 和 本地数据中心。由于 Terraform 具有可移植性，所以开发人员可以使用相同的工具和配置文件同时管理不同云提供商的资源。这种灵活性允许用户在任何时候切换到其他云平台，而不需要更改其配置文件。
- 自动化管理：Terraform 可以创建配置文件的模板，以可重复、可预测的方式定义、预配和配置资源，减少因人为因素导致的部署和管理错误。通过使用 Terraform，用户能够多次部署同一模板，创建相同的开发、测试和生产环境。此外，Terraform 还能自动化处理基础设施的创建和管理过程，提高了部署速度和效率。
- 基础设施即代码：Terraform 的基础设施即代码（IaC）方法允许用代码来管理资源，提供了版本控制、协作和重用代码的便利性。使用 Terraform，您可以将基础设施的状态保存下来，以便跟踪对系统进行的更改，并与其他人共享这些配置。这种可代码化的管理方式不仅使部署简单化，还能确保一致性、安全性和可维护性。
- 可视化界面：Terraform 图形界面能够查看和管理基础架构资源。这使得用户可以更加直观地查看和理解他们的基础设施，以及它们之间的相关性。通过这个可视化界面，用户可以更快、更容易地定位并修复潜在的配置问题。
- 社区支持：Terraform 是由 HashiCorp 公司开发的，并有一个庞大的社区支持。这个社区覆盖了各种行业和经验水平的专业人士，他们能够分享最佳实践和解决方案，提供论坛和邮件列表等支持。

## 通过 Terraform 使用腾讯云可观测平台 Prometheus 托管服务

Prometheus 支持通过 Terraform 管理以下 Resource：

| 名称 | 描述 |
| --- | --- |
| tencentcloud_monitor_tmp_instance | Prometheus 实例 |
| tencentcloud_monitor_tmp_alert_rule | Prometheus 告警规则 |
| tencentcloud_monitor_tmp_recording_rule | Prometheus 预聚合规则 |
| tencentcloud_monitor_tmp_cvm_agent | Prometheus Cvm Agent |
| tencentcloud_monitor_tmp_scrape_job | Prometheus cvm_agent 抓取任务 |
| tencentcloud_monitor_tmp_exporter_integration | Prometheus 集成中心资源 |
| tencentcloud_monitor_tmp_manage_grafana_attachment | Prometheus 绑定 grafana |
| tencentcloud_monitor_tmp_tke_cluster_agent | Prometheus 关联容器 |

| | |
|---|---|
| tencentcloud_monitor_tmp_tke_config | Prometheus 集群采集配置 |

# 使用 Terraform 管理 Prometheus 实例

最近更新时间：2025-03-19 11:01:02

## 前提条件

### 安装 Terraform

- 腾讯云 Cloud Shell 是一款帮助您运维的免费产品，预装了 Terraform 相关组件，并配置好腾讯云临时凭证（credentials）。
- 如果您不使用 Cloud Shell，关于安装 Terraform 的具体操作，请参见在 本地安装和配置 Terraform。

> **说明：**
> - Terraform 安装版本不得低于 v1.6.3，您可通过 terraform --version 命令查看安装的 Terraform 版本。
> - 如果您是通过云端管理，支持通过 云端安装，云端安装获取访问密钥与本地相同。

### 配置腾讯云账号信息

在首次使用 Terraform 之前，请前往 云 API 密钥 申请安全凭证 SecretId 和 SecretKey。如果已有可使用的安全凭证，则跳过该步骤。

1. 登录 访问管理控制台，在左侧导航栏，进入**访问密钥** > **API 密钥管理**。
2. 在 API 密钥管理页面，单击**新建密钥**，即可以创建一对 SecretId / SecretKey。

**配置腾讯云账号信息，有以下两种方式：**

- 静态凭证鉴权。

  在用户目录下创建 `provider.tf` 文件，输入如下内容。其中 `my-secret-id` 和 `my-secret-key` 需替换为密钥 SecretId 和 SecretKey。

```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

- 环境变量鉴权。

  配置电脑环境变量或云端环境变量，请执行以下命令。其中 `YOUR_SECRET_ID` 和 `YOUR_SECRET_KEY` 需替换为密钥 SecretId 和 SecretKey。

```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

## 创建 prometheus 实例

1. 创建 Terraform 的配置文件。在项目目录中创建一个名为 main.tf 的文件，并使用合适的编辑器打开，在 main.tf 文件或者 provider.tf 文件中添加腾讯云提供者配置。

```
  provider "tencentcloud" {
    region     = "your_region"
    secret_id  = "your_secret_id"
    secret_key = "your_secret_key"
  }
```

2. 定义腾讯云资源：在 main.tf 文件中使用 Terraform 腾讯云提供的资源来定义您想要创建和管理的资源。

```
# 指定 provider 配置信息

terraform {
 required_providers {
   tencentcloud = {
     source = "tencentcloudstack/tencentcloud"
   }
```

```
  }
}

 # 创建 prometheus 实例

resource "tencentcloud_monitor_tmp_instance" "foo" {
  instance_name       = "tf-tmp-instance-sjtest"
  vpc_id              = "vpc-0n42dxzs"
  subnet_id           = "subnet-es8rv1kx"
  data_retention_time = 30
  zone                = "ap-guangzhou-3"
  tags = {
    "createdBy" = "terraform"
  }
}

# 创建 grafana 实例（选填）

resource "tencentcloud_monitor_grafana_instance" "foo" {
  instance_name         = "tf-grfana-cstest"
  vpc_id                = "vpc-0n42dxzs"
  subnet_ids            = ["subnet-es8rv1kx"]
  grafana_init_password = "1234567890"
  enable_internet       = false
  is_destroy            = true

  tags = {
    "createdBy" = "test"
  }
}

# grafana 与 Prometheus 实例做绑定（选填）
resource "tencentcloud_monitor_tmp_manage_grafana_attachment" "foo" {
  grafana_id  = tencentcloud_monitor_grafana_instance.foo.id
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
}
```

3. 初始化 Terraform 运行环境，执行命令如下。

```
terraform init
```

预期输出信息：

```
  Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lock file
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

4. 生成资源规划，执行命令如下。

```
terraform plan
```

预期输出信息：

```
  tencentcloud_vpc.vpc: Refreshing state... [id=vpc-3csjp7k8]
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-4wcvt7p1]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_instance.foo will be created
  + resource "tencentcloud_monitor_tmp_instance" "foo" {
      + api_root_path       = (known after apply)
      + data_retention_time = 30
      + id                  = (known after apply)
      + instance_name       = "tf-tmp-instance-sjtest"
      + ipv4_address        = (known after apply)
      + proxy_address       = (known after apply)
      + remote_write        = (known after apply)
      + subnet_id           = "subnet-es8rv1kx"
      + tags                = {
          + "createdBy" = "terraform"
        }
      + vpc_id              = "vpc-0n42dxzs"
      + zone                = "ap-guangzhou-3"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

5. 创建实例，执行命令如下。

```
terraform apply
```

预期输出信息：

```
tencentcloud_vpc.vpc: Refreshing state... [id=vpc-3csjp7k8]
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-4wcvt7p1]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_instance.foo will be created
  + resource "tencentcloud_monitor_tmp_instance" "foo" {
      + api_root_path       = (known after apply)
      + data_retention_time = 30
      + id                  = (known after apply)
      + instance_name       = "tf-tmp-instance-sjtest"
      + ipv4_address        = (known after apply)
```

```
      + proxy_address        = (known after apply)
      + remote_write         = (known after apply)
      + subnet_id            = "subnet-es8rv1kx"
      + tags                 = {
          + "createdBy" = "terraform"
        }
      + vpc_id               = "vpc-0n42dxzs"
      + zone                 = "ap-guangzhou-3"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

**以下信息填入 “yes” 继续操作：**

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

**若出现以下信息，表示您已创建成功：**

```
tencentcloud_monitor_tmp_instance.foo: Creating...
tencentcloud_monitor_tmp_instance.foo: Still creating... [10s elapsed]
tencentcloud_monitor_tmp_instance.foo: Creation complete after 12s [id=prom-8dyb6iny]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

## 查看 Prometheus 实例状态

登录 腾讯云可观测平台，在左侧导航栏选择 **prometheus 监控**，可在 prometheus 实例列表中看到存在的实例。

## 删除 Prometheus 实例

删除 Prometheus 实例，执行命令如下。

```
terraform destroy
```

出现以下信息填入提示您填入 “yes” 确认。

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  - destroy

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_instance.foo will be destroyed
  - resource "tencentcloud_monitor_tmp_instance" "foo" {
      - api_root_path       = "http://10.0.0.34:9090/api/v1" -> null
      - data_retention_time = 30 -> null
      - id                  = "prom-8dyb6iny" -> null
      - instance_name       = "tf-tmp-instance-sjtest" -> null
      - ipv4_address        = "10.0.0.34" -> null
      - proxy_address       = "10.0.0.34:9090" -> null
      - remote_write        = "http://10.0.0.34:9090/api/v1/prom/write" -> null
      - subnet_id           = "subnet-es8rv1kx" -> null
```

```
      - tags                = {
          - "createdBy" = "terraform"
        } -> null
      - vpc_id              = "vpc-0n42dxzs" -> null
      - zone                = "ap-guangzhou-3" -> null
    }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

**说明：**
当出现 `Destroy complete! Resources: 1 destroyed.` 表示您已删除该实例。

# 使用 Terraform 管理 Prometheus 实例的集成中心

最近更新时间：2024-05-29 17:15:22

## 前提条件

### 安装 Terraform

- 腾讯云 Cloud Shell 是一款帮助您运维的免费产品，预装了 Terraform 相关组件，并配置好腾讯云临时凭证（credentials）。
- 如果您不使用 Cloud Shell，关于安装 Terraform 的具体操作，请参见在 本地安装和配置 Terraform。

> **说明：**
> Terraform 安装版本不得低于 v1.6.3，您可通过 terraform --version 命令查看安装的 Terraform 版本。
> 若您是通过云端管理，可前往 云端安装，云端安装获取访问密钥与本地相同。

### 配置腾讯云账号信息

在首次使用 Terraform 之前，请前往 云 API 密钥 申请安全凭证 SecretId 和 SecretKey。若已有可使用的安全凭证，则跳过该步骤。

1. 登录 访问管理控制台，在左侧导航栏，选择**访问密钥 > API 密钥管理**。
2. 在 API 密钥管理页面，单击**新建密钥**，即可以创建一对 SecretId / SecretKey。

### 配置腾讯云账号信息，有以下两种方式：

- 静态凭证鉴权

在用户目录下创建 `provider.tf` 文件，输入如下内容。其中 `my-secret-id` 和 `my-secret-key` 需替换为密钥 SecretId 和 SecretKey。

```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

- 环境变量鉴权

配置电脑环境变量或云端环境变量，请执行以下命令。其中 `YOUR_SECRET_ID` 和 `YOUR_SECRET_KEY` 需替换为密钥 SecretId 和 SecretKey。

```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

## 增加 Prometheus 实例的集成中心组件集成

1. 创建一个新的 Terraform 配置文件：创建一个新的目录，并在该目录下创建一个 main.tf 的文件 ，配置如下信息：

```
# 指定 provider 配置信息

terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}

# prometheus管理云监控集成

## black-box 集成
resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterIntegration" {
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  kind        = "blackbox-exporter"
```

```
  content     = "{\"name\":\"balck-box-tf-test\",\"kind\":\"blackbox-exporter\",\"spec\":
{\"instanceSpec\":{\"module\":\"http_get\",\"urls\":[\"http://baidu.com\"]}}}"
  kube_type   = 1
  cluster_id  = ""
}

## 云监控插件集成
resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterMointor" {
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  kind        = "qcloud-exporter"
  content     = "{\"name\":\"tf-test-cjtest\",\"kind\":\"qcloud-exporter\",\"spec\":{\"instanceSpec\":
{\"region\":\"ap-
guangzhou\",\"delaySeconds\":0,\"reload_interval_minutes\":10,\"useRole\":true,\"labels\":
{}},\"exporterSpec\":
{\"cvm\":true,\"cbs\":false,\"lb_public\":true,\"lb_private\":false,\"tgw_set\":false,\"cmongo\":false,\"c
db\":false,\"redis\":false,\"redis_mem\":false,\"mariadb\":false,\"postgres\":false,\"tdmysql\":false,\"cy
nosdb_mysql\":false,\"sqlserver\":false,\"nat_gateway\":false,\"ckafka\":false,\"rocketmq\":false,\"tdmq\"
:false,\"lb\":false,\"vpngw\":false,\"vpnx\":false,\"cdn\":false,\"ov_cdn\":false,\"cos\":false,\"dc\":fal
se,\"dcx\":false,\"dcg\":false,\"lighthouse\":false,\"nacos\":false,\"zookeeper\":false,\"ces\":false,\"dt
s\":false,\"vbc\":false,\"gaap\":false,\"waf\":false,\"cfs\":false,\"bwp\":false,\"scf_v2\":false,\"vod\":
false,\"cls\":false,\"apigateway\":false,\"self\":false},\"scrapeSpec\":
{\"relabelConfigs\":\"metricRelabelings:\\n- action: labeldrop\\n  regex: tmp_test_label\\n\"}}}"
  kube_type   = 1
  cluster_id  = ""
}
```

**说明：**

创建 Prometheus 实例的集成中心组件时配置的字段如下：

- cluster_id：（必填，字符串）集群 ID。
- content：（必填，字符串）集成配置（content 内部参数可自行根据需要更换）。
- instance_id：（必填，字符串）实例 ID。
- kind：（必填，字符串）类型。
- kube_type：（必填，整数）集成配置。

您若需要获取更多详细参数，请参考 腾讯接入云集成接入 github，也可以通过 Terraform 腾讯云提供商 了解更多。

2. 初始化 Terraform 运行环境，执行命令如下：

```
terraform init
```

预期输出信息：

```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lock file
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

**3. 生成资源规划，执行命令如下：**

```
terraform plan
```

**预期输出信息：**

```
Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration will be created
  + resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterIntegration" {
      + content     = jsonencode(
            XXX...
        )
      + id          = (known after apply)
      + instance_id = (known after apply)
      + kind        = "blackbox-exporter"
      + kube_type   = 1
    }

  # tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor will be created
  + resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterMointor" {
      + content     = jsonencode(
            XXX...
        )
      + id          = (known after apply)
      + instance_id = (known after apply)
      + kind        = "qcloud-exporter"
      + kube_type   = 1
    }

  # tencentcloud_monitor_tmp_instance.foo will be created
  + resource "tencentcloud_monitor_tmp_instance" "foo" {
      + api_root_path       = (known after apply)
      + data_retention_time = 30
      + id                  = (known after apply)
      + instance_name       = "tf-tmp-instance-sjtest"
      + ipv4_address        = (known after apply)
      + proxy_address       = (known after apply)
      + remote_write        = (known after apply)
      + subnet_id           = "subnet-es8rv1kx"
      + tags                = {
          + "createdBy" = "terraform"
        }
      + vpc_id              = "vpc-0n42dxzs"
      + zone                = "ap-mumbai-1"
    }

Plan: 3 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------------------------------------
--------------

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these
actions if
```

```
you run "terraform apply" now.
```

4. 创建集成中心组件集成，执行命令如下：

```
   terraform apply
```

预期输出信息：

```
Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:
XXX...


Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Creating...
tencentcloud_monitor_tmp_instance.foo: Still creating... [10s elapsed]
。。。

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

> **注意：**
> 当出现 Apply complete! Resources: 3 added, 0 changed, 0 destroyed. 表示您已创建成功。

## 查看 Prometheus 实例状态

登录 腾讯云可观测平台，在左侧导航栏，选择 **prometheus 监控**，可在 prometheus 实例列表中看到存在的实例。

## 删除 Prometheus 实例集成中心组件集成

销毁资源，执行命令如下：

```
terraform destroy
```

预期输出信息：

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  - destroy

Terraform will perform the following actions:
XXX...

Plan: 0 to add, 0 to change, 1 to destroy.
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

**注意：**
当出现 `Destroy complete! Resources: 1 destroyed.` 表示您已删除该实例。

# 使用 Terraform 采集容器监控数据

最近更新时间：2024-05-29 17:15:22

## 前提条件

### 安装 Terraform

- 腾讯云 Cloud Shell 是一款帮助您运维的免费产品，预装了 Terraform 相关组件，并配置好腾讯云临时凭证（credentials）。
- 如果您不使用 Cloud Shell，关于安装 Terraform 的具体操作，请参见在 本地安装和配置 Terraform。

> **说明：**
> Terraform 安装版本不得低于 v1.6.3，可通过 terraform --version 命令查看安装后 Terraform 版本。
> 若您是云端管理，可前往 云端安装，云端安装获取访问密钥与本地相同。

### 配置腾讯云账号信息

在首次使用 Terraform 之前，请前往 云 API 密钥页面 申请安全凭证 SecretId 和 SecretKey。若已有可使用的安全凭证，则跳过该步骤。

1. 登录 访问管理控制台，在左侧导航栏，选择**访问密钥 > API 密钥管理**。
2. 在 API 密钥管理页面，单击**新建密钥**，即可以创建一对 SecretId / SecretKey。

### 配置腾讯云账号信息，有以下两种方式：

- 静态凭证鉴权

在用户目录下创建 `provider.tf` 文件，输入如下内容。其中 `my-secret-id` 和 `my-secret-key` 需替换为密钥 SecretId 和 SecretKey。

```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

- 环境变量鉴权

配置电脑环境变量或云端环境变量，请执行以下命令。其中 `YOUR_SECRET_ID` 和 `YOUR_SECRET_KEY` 需替换为密钥 SecretId 和 SecretKey。

```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

## 增加 Prometheus 实例的采集容器监控数据

1. 创建一个新的 Terraform 配置文件：创建一个新目录，并在该目录下创建一个名为 main.tf 的文件，配置如下信息：

```
# 指定 provider 配置信息

terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}
# prometheus管理云监控集成

#prometheus 管理容器集群(采集容器)

resource "tencentcloud_monitor_tmp_tke_cluster_agent" "foo" {
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  agents {
    region          = "ap-mumbai"
```

```
    cluster_type    = "eks"
    cluster_id      = "cls-1uary7z2" #需要关联的集群id
    enable_external = false
  }
}
```

**说明：**

创建 Prometheus 实例的集成中心组件需配置对应参数，需配置的字段如下：

- instance_id:（必填、字符串、ForceNew）实例 ID。
- agents：（必填，列表）代理列表。
  - cluster_id：（必填，字符串）标识集群的 id。
  - region：（必填，字符串）区域限制。
  - enable_external：（必填，Bool）是否开启公网 CLB。
  - cluster_type:（必填，字符串）集群类型。

您若需要获取更多详细参数请参考 腾讯接入云集成接入 github，也可以通过 Terraform 腾讯云提供商 了解更多。

2. 初始化 Terraform 运行环境，执行命令如下：

```
terraform init
```

预期输出信息：

```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lock file
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

3. 生成资源规划，执行命令如下：

```
  terraform plan
```

预期输出信息：

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing state... [id=balck-box-
tf-test#prom-jh0zntj2#1##blackbox-exporter]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state... [id=tf-test-
cjtest#prom-jh0zntj2#1##qcloud-exporter]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create
```

```
Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_tke_cluster_agent.foo will be created
  + resource "tencentcloud_monitor_tmp_tke_cluster_agent" "foo" {
      + id          = (known after apply)
      + instance_id = "prom-jh0zntj2"

      + agents {
          + cluster_id      = "cls-1uary7z2"
          + cluster_name    = (known after apply)
          + cluster_type    = "eks"
          + enable_external = false
          + region          = "ap-mumbai"
          + status          = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------------------------------------------
---------------

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these
actions if
you run "terraform apply" now.
```

**4. 创建集成中心组件集成，执行命令如下：**

```
   terraform apply
```

**预期输出信息：**

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing state... [id=balck-box-
tf-test#prom-jh0zntj2#1##blackbox-exporter]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state... [id=tf-test-
cjtest#prom-jh0zntj2#1##qcloud-exporter]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_tke_cluster_agent.foo will be created
  + resource "tencentcloud_monitor_tmp_tke_cluster_agent" "foo" {
      + id          = (known after apply)
      + instance_id = "prom-jh0zntj2"

      + agents {
          + cluster_id      = "cls-1uary7z2"
          + cluster_name    = (known after apply)
          + cluster_type    = "eks"
          + enable_external = false
          + region          = "ap-mumbai"
          + status          = (known after apply)
        }
    }
```

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tencentcloud_monitor_tmp_tke_cluster_agent.foo: Creating...
实例内容。。。

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

## 查看 Prometheus 实例状态

登录 腾讯云可观测平台，在左侧导航栏，选择 **prometheus 监控**，可在 prometheus 实例列表中看到存在的实例。

## 删除 Prometheus 实例集成中心组件集成

销毁资源，执行命令如下：

```
terraform destroy
```

预期输出信息：

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  - destroy

Terraform will perform the following actions:
实例内容。。。

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

> ⚠ **注意：**
> 当出现 `Destroy complete! Resources: （存在的实例个数） destroyed.` 表示您已删除该实例。

# 使用 Terraform 配置告警策略

最近更新时间：2024-12-05 10:59:42

## 前提条件

### 安装 Terraform

- 腾讯云 Cloud Shell 是一款帮助您运维的免费产品，预装了 Terraform 相关组件，并配置好腾讯云临时凭证（credentials）。
- 如果您不使用 Cloud Shell，关于安装 Terraform 的具体操作，请参见在 本地安装和配置 Terraform 。

> **说明：**
> Terraform 安装版本不得低于 v1.6.3，您可通过 terraform --version 命令查看安装的 Terraform 版本。
> 若您是通过云端管理，可前往 云端安装，云端安装获取访问密钥与本地相同。

### 配置腾讯云账号信息

在首次使用 Terraform 之前，请前往 云 API 密钥 申请安全凭证 SecretId 和 SecretKey。如果已有可使用的安全凭证，则跳过该步骤。

1. 登录 访问管理控制台，在左侧导航栏，选择**访问密钥 > API 密钥管理**。
2. 在 API 密钥管理页面，单击**新建密钥**，即可以创建一对 SecretId / SecretKey。

### 配置腾讯云账号信息，有以下两种方式：

- 静态凭证鉴权

在用户目录下创建 `provider.tf` 文件，输入如下内容。其中 `my-secret-id` 和 `my-secret-key` 需替换为密钥 SecretId 和 SecretKey。

```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

- 环境变量鉴权

配置电脑环境变量或云端环境变量，请执行以下命令。其中 `YOUR_SECRET_ID` 和 `YOUR_SECRET_KEY` 需替换为密钥 SecretId 和 SecretKey。

```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

## 增加 Prometheus 实例的 Terraform 配置告警策略

1. 创建一个新的 Terraform 配置文件。创建一个新的目录，并在该目录下创建一个 main.tf 文件，配置如下信息：

```
# 指定 provider 配置信息

terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}

# prometheus 配置告警 （云监控侧配置）

resource "tencentcloud_monitor_tmp_alert_rule" "foo" {
  duration    = "2m"
  expr        = "avg by (instance) (mysql_global_status_threads_connected) / avg by (instance)
(mysql_global_variables_max_connections)  > 0.8"
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
```

```
  receivers    = ["notice-zmjsavnp"] # 此处可通过云监控的tf创建通知模板
  rule_name    = "MySQL 连接数过多--tf-云监控test"
  rule_state   = 2
  type         = "MySQL/MySQL 连接数过多"

  annotations {
    key   = "description"
    value = "MySQL 连接数过多，实例：{{$labels.instance}}，当前值：{{ $value | humanizePercentage }}。"
  }
  annotations {
    key   = "summary"
    value = "MySQL 连接数过多(>80%)"
  }

  labels {
    key   = "severity"
    value = "warning"
  }
}
```

**说明：**

配置的字段如下：

- duration：规则持续时间。
- expr：报警表达式。
- instance_id：实例 ID。
- receivers：报警接收人列表。
- rule_name：报警规则名称。
- rule_state：报警规则状态。
- type：报警规则类型。

您若需要获取更多详细参数请参考 腾讯云集成接入 github，也可以通过 Terraform 腾讯云提供商 了解更多。

2. 初始化 Terraform 运行环境，执行命令如下：

```
terraform init
```

预期输出信息：

```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lock file
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

3. 生成资源规划，执行命令如下：

```
  terraform plan
```

**预期输出信息：**

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing state... [id=balck-box-
tf-test#prom-jh0zntj2#1##blackbox-exporter]
tencentcloud_monitor_tmp_tke_cluster_agent.foo: Refreshing state... [id=prom-jh0zntj2#cls-1uary7z2#eks]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state... [id=tf-test-
cjtest#prom-jh0zntj2#1##qcloud-exporter]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_alert_rule.foo will be created
  + resource "tencentcloud_monitor_tmp_alert_rule" "foo" {
      + duration     = "2m"
      + expr         = "avg by (instance) (mysql_global_status_threads_connected) / avg by (instance)
(mysql_global_variables_max_connections)  > 0.8"
      + id           = (known after apply)
      + instance_id  = "prom-jh0zntj2"
      + receivers    = [
          + "notice-zmjsavnp",
        ]
      + rule_name    = "MySQL 连接数过多--tf-云监控test"
      + rule_state   = 2
      + type         = "MySQL/MySQL 连接数过多"

      + annotations {
          + key   = "description"
          + value = "MySQL 连接数过多，实例：{{$labels.instance}}，当前值：{{ $value | humanizePercentage
}}。"
        }
      + annotations {
          + key   = "summary"
          + value = "MySQL 连接数过多(>80%)"
        }

      + labels {
          + key   = "severity"
          + value = "warning"
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------------
------------

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these
actions if
you run "terraform apply" now.
```

**4. 创建集成中心组件集成，执行命令如下：**

```
terraform apply
```

预期输出信息：

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing state... [id=balck-box-
tf-test#prom-jh0zntj2#1##blackbox-exporter]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state... [id=tf-test-
cjtest#prom-jh0zntj2#1##qcloud-exporter]
tencentcloud_monitor_tmp_tke_cluster_agent.foo: Refreshing state... [id=prom-jh0zntj2#cls-1uary7z2#eks]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create

Terraform will perform the following actions:
 实例内容。。。

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tencentcloud_monitor_tmp_alert_rule.foo: Creating...
tencentcloud_monitor_tmp_alert_rule.foo: Creation complete after 2s [id=prom-jh0zntj2#arule-n76kqshg]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

## 查看 Prometheus 实例状态

登录 腾讯云可观测平台，在左侧导航栏，选择 prometheus 监控，可在 prometheus 实例列表中看到存在的实例。

## 删除 Prometheus 实例集成中心组件集成

销毁资源，执行命令如下：

```
terraform destroy
```

预期输出信息：

```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  - destroy

Terraform will perform the following actions:
实例内容。。。

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.
```

```
  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

**注意：**

当出现 `Destroy complete! Resources: （存在的实例个数） destroyed.` 表示您已删除该实例。

# 常见问题
# 基础问题

最近更新时间：2024-10-21 22:06:02

## Prometheus 原生默认的 UI 等功能怎么使用?

TMP 不同于开源的单机版 Prometheus，腾讯云 Prometheus 监控服务是采集和存储分离的结构，不提供原生默认的 UI 功能，查询功能请使用 Grafana 服务作为替代。

## Prometheus 原生的某些 API 功能是否支持?

TMP 不同于开源的单机版 Prometheus，腾讯云 Prometheus 监控服务是采集和存储分离的结构，不支持所有原生的 API 功能。请参考目前提供的 API 列表 。

## 告警恢复时通知模板里面的 $value 不正确，如何处理?

告警恢复的 $value 是最后一次满足告警表达式条件的值无法获取不满足条件的值。从设计上来讲，告警表达式是作为一个整体的，告警表达式和普通场景下的 PromQL 查询没有任何区别，查询结果中的 series 如果满足持续时长就会被触发，当下次查询的结果不包含某个 series 时，该 series 对应的告警会变成恢复状态，Prometheus 无法自行对告警表达式进行拆解和解释，因为有些表达式本身是不包含阈值等比较关系的，例如：`a and b` 、`123456789` 。

## 如何查看告警历史?

Prometheus 本身没有告警历史的概念，由于特殊原因无法完整的支持告警历史，但是相关告警及状态可以通过 ALERTS 或者 ALERTS_FOR_STATE 指标来进行查看。
另外，Prometheus 所有的告警目前都会投递到腾讯云可观测平台告警服务，然后再由腾讯云可观测平台的各种通知渠道发送到用户端，腾讯云可观测平台提供的告警历史功能可以一定程度上弥补这个缺陷，由于设计等概念上的不同不能作为完整替代但可用于排障等需求。您可以通过 告警历史功能 查看。

## 告警重复时间间隔和配置的时长不一致?

Prometheus 重复告警间隔 (repeat_interval) 并不是针对单个告警设置的，而是针对整个分组的，当分组中存在新告警或者告警恢复时整个分组都会通知出去，然后再按重复告警间隔进行重复通知，当前 Prometheus 监控服务 是按单个告警策略进行分组的，例如：单个告警策略可能是针对实例 A/B/C 是否重启的情况，当 A/B 重启后告警触发然后按照重复间隔进行通知，一段时间后 C 也重启了，这个重复告警的间隔就会被打断并重置，此时 A/B/C 作为一个“分组”都会一起通知到用户，目前基于腾讯云可观测平台投递的告警由于实现上的限制无法整个分组聚合作为单一的消息通知到用户，有需要的可以使用自定义 Alertmanager 或者 Grafana 服务。

## 原始指标存在的情况下，rate/irate 函数为什么没有产生任何数据?

rate/irate 函数需要至少两个数据点才能进行计算，所以要保证 rate/irate 计算的时间范围覆盖到至少两个数据点，考虑到网络等异常可能出现的数据点丢失的情况，这个时间范围官方推荐为**四倍的采集间隔**。

## rate/irate 函数为什么计算出一个极大的异常值?

rate/irate 函数只能用于 Counter 类型的指标，Counter 类型的指标定义为严格递增的数字，Prometheus 查询时会处理 Counter 重置为 0 的问题， 如服务器重启等，正常情况下不对计算结果产生影响，除非出现数据点乱序的问题，例如 9999 和 10000 两个秒级的数据点乱序，导致异常值为 (10000+9999)−10000=9999（正常情况应该为 1），出现这种情况的典型场景如下：

- 存在多个采集组件采集同一个指标，并重复上报到同一个 Prometheus 实例，这时可能会产生乱序的问题，导致经过 Prometheus 的重置处理逻辑后产生一个极大的异常值。解决方案：如果采集组件采集同一个指标是不符合预期的建议排查并解决重复采集的问题；如果是为了保证采集组件的高可用性而设计的采集方案，那么不同的采集组件对所采集的所有指标需要额外加上 replica 或者其它标签进行区分。
- 通过 pushgateway 等方式直接上报到 Prometheus 监控服务 没有带上时间戳信息，Prometheus 监控服务 端会将当前时间作为指标的时间戳并存储，这样如果网络出现抖动等情况，数据点到达时间会变得无序或者不同进程/线程处理不同的数据点时赋予的时间戳信息导致乱序，从而导致计算出错。解决方案是在上报时带上时间戳信息，Prometheus 监控服务 内置的 pushgateway 上报方式本身只作为 remotewrite 的补充并没有实现完整的功能，非必要情况建议使用 Agent(remotewrite) 进行采集上报。

## 查询返回的数据点间隔为什么和抓取间隔不一样?

Prometheus 查询返回的数据点间隔由查询参数 interval/step 来决定，每个数据点都是严格按照 interval/step 来补齐对齐的，和抓取间隔没有一一对应的关系，在数据丢失过多或者抓取间隔过大的情况下不会进行数据补齐，存储端也不会存储任何抓取配置相关的信息，需要用户对自己的抓取配置自行处理。

## 查询为什么多返回了最近五分钟的数据?

Prometheus 默认会对某些查询的进行数据补齐，即使最近五分钟只有一个数据点可能也会返回五分钟多个数据点（根据查询的 step/interval 参数的不同而不同），这个是开源 Prometheus 的默认行为，暂时无法调整。一般情况下，并不影响正常使用。

## PromQL 时间相关的函数如何处理本地时区?

Prometheus 内部所有的时间都是 UTC 时间，没有特殊情况，设计上也并没有考虑时区的概念可能会导致 `day_of_*` 系列函数使用起来不那么方便，短期内官方也不会支持，国内时区的临时解决方案可以这样：`day_of_week(vector(time() + 8*3600))` 。

## Prometheus 相关的使用限制可以调整吗?

Prometheus 相关的使用限制部分是可以调整的，大部分情况下超出这些使用限制可能会带来使用体验以及性能的降级，所以向上调整限制后不能保证调整之前的查询和写入性能基准，服务等级协议可能不再适用，对于潜在的相关风险需要用户有一定心理预期。

## 从图表上看告警满足持续时间而实际未触发?

Prometheus 告警检测的基本原理是每隔一分钟使用 Instant Query 查询数据，如果（持续）满足告警条件则触发。某些情况下 Prometheus 的 Range Query 会补齐填充数据，图表上连续的时间线，在告警组件的执行逻辑下可能就是不连续的，另外由于告警定时检查的特性，告警被检测到的时间可能会有些许延后，可以通过查询 ALERTS 或者 ALERTS_FOR_STATE 指标来查看告警的状态。

# 集成容器服务相关

最近更新时间：2024-10-21 22:06:02

## 无法从内网访问容器服务 TKE 集群?

在安装 Agent 时，需要通过内网访问容器服务，如果对应的容器服务集群未打开**内网访问**将会导致安装失败，可以通过如下步骤指引进行解决：

1. 登录 容器服务控制台，选择对应地域下的容器集群。
2. 在**基本信息** > **集群APIServer信息**下开启**内网访问**。

## kube-proxy 采集目标状态全部为 DOWN，该如何解决?

TKE 中 kube-proxy 未指定启动参数 `--metrics-bind-address`，而 metrics 服务默认监听地址为127.0.0.1，因此 Agent 无法根据 POD IP 拉取到 metrics，可通过如下步骤指引进行解决：

1. 登录 容器服务控制台，选择对应地域下的容器集群。
2. 在**基本信息** > **集群APIServer信息** > **通过Kubectl连接Kubernetes集群操作说明**根据指引设置 kubectl。
3. 执行命令 `kubectl edit ds kube-proxy -n kube-system`，在 `spec.template.spec.containers.args` 中添加启动参数 `--metrics-bind-address=0.0.0.0`。

## 独立 TKE 集群 Master 节点上组件采集目标状态全部为 DOWN，该如何解决?

独立 TKE 集群 Master 节点的默认安全组入站规则不允许访问部分组件的 metrics 端口，可通过如下步骤指引进行解决：

1. 登录 安全组控制台，选择对应区域。
2. 在安全组搜索框中输入 `tke-master-security-for-<tke cluster id>`。例如集群 ID。

# 产品咨询

最近更新时间：2024-10-21 22:06:02

## Prometheus 监控服务是否支持自定义上报数据?

支持，Prometheus 监控服务支持多种语言自定义上报指标监控数据，并展示在集成的 Grafana 大盘中。

## Prometheus 监控服务的监控数据是怎么采集的?

通过 Prometheus agent 拉取，也支持通过 pushgateway 方式写入。完全兼容开源 Prometheus 采集方式。

## 每个实例都是一个单独的 exporter 么?

目前社区中 MySQL，Kafka 这种是一个实例一个 exporter，Redis 是支持一个 exporter 多实例的。

## Prometheus 监控服务和自建有什么区别吗?

TMP 完全兼容开源生态，并与腾讯云可观测平台-云产品监控数据打通，帮助用户快速搭建监控体系（自定义监控，组件监控，基础监控等），支持 Grafana 并预设了常用的监控 Dashboard，支持丰富的 Exporter 并预设了常见的告警模板；很好解决了开源社区 Prometheus 高可用搭建困难， Prometheus 性能可扩展性差，运维消耗人力等痛点。

## Prometheus支持哪些云产品?

云服务器、云数据库 MongoDB、云数据库 MySQL、云数据库 PostgreSQL、云数据库 Redis、ElasticSearch、容器服务等。详情请参见 集成中心 。

## 可以在 Prometheus 监控服务的 Grafana 界面集成其他数据源吗?

Prometheus 监控服务不支持其他数据源，建议使用 腾讯云 Grafana 服务 集成其它数据源。

# 使用&技术问题

最近更新时间：2024-10-21 22:06:02

## 自建 Prometheus如何迁移到 Prometheus 监控服务?

在自建 Prometheus 的配置文件中加一个 Remote Write 配置指向到 Prometheus 监控服务即可进行迁移，详情请参见 自建 Prometheus 迁入 。

## 请问 Grafana 能支持批量导出导入 Dashboard 吗?

目前只能通过 API 导出，详情请参见 HTTP API Reference 。

## 一个 exporter 数据太多了怎么办?

prometheus 服务能支持这种场景，但是不建议在 exporter 暴露特别多的 metrics，exporter 本身和 prometheus agent 会消耗比较多的内存。建议暴露一些关键指标，不要在指标的 label 中使用用户 id、url 等参数信息。

## 如何使用 Prometheus 监控服务监控两个不同 VPC 下的集群?

1. 通过 云联网 打通两个集群之间的 VPC 网络，即可在 Prometheus 集成两个不同 VPC 下的集群。
2. 在其中一个集群 安装 Agent，并通过 Nginx 代理，remote write 目标地址到 Prometheus 实例。

## 原生的容器服务如何接入 Prometheus?

暂不支持，可通过自建数据，在自建 Prometheus 的配置文件中加一个 Remote Write 配置指向到 Prometheus 监控服务即可，详情请参见 自建 Prometheus 迁入 。

## Prometheus 接口创建告警规则，告警周期的参数要怎么设置?

在新建 Labels 参数中加 `key=_interval_  value=1m/5m/10m/15m/30m/60m` 即可。

## 重建实例会丢失数据吗?

数据存储在腾讯云的时序数据库（TencentDB for CTSDB）上，重建实例数据不会丢失。但由于重建期间无法正常上报数据，数据会出现断点情况。

## Prometheus实例重建后，产生数据量飙升，这是正常的吗?

短时间内的波动是正常的，重建后会有重试操作，所以数据量短时间看起来是有较大的波动。

## Prometheus 集成中心-Mysql 集成，可以批量添加 MySQL 实例吗?

暂不支持，目前只能一个一个添加。

## Prometheus 创建的弹性集群（EKS）的安全组在哪里可以查看?

私有网络(VPC) > 安全 > 安全组 > 以当前 Prometheus 实例 ID 为关键词搜索即可查看。

## pod monitor 是否可以配置多个抓取?

可以，只需保证 port name 相同、label 相同即可。

## （Prometheus 监控服务 pushgateway 用法）如果多个实例的客户端 push 同个 job 但不同 label 指标时，会有指标覆盖的情况，如何处理?

通过 groupingkey 可以解决这个问题，代码示例如下：

```
if err := push.New("http://$IP:$PORT", "db_backup").
        BasicAuth("$APPID", "$TOKEN").
        Collector(completionTime).
        Grouping("instance", "$INSTANCE").
```