容器服务

云原生服务指南

产品文档

腾讯云

文档目录

云原生服务指南
云原生 AI 指南
运维控制台指南
管理 AI 环境

最近更新时间：2022-10-12 11:56:32

本文档向您介绍什么是 AI 环境，及如何管理 AI 环境，包括创建 AI 环境、查看 AI 环境以及删除 AI 环境。

AI 环境概述

AI 环境是云原生 AI 服务的重要抽象概念。一个 AI 环境运行在特定（TKE/EKS）容器集群上，运维人员可按需管理 AI 环境的生命周期。例如，针对不同的上层 AI 业务，AI 运维方可以基于不同组件搭建符合不同业务需求的 AI 环境。

操作步骤

创建 AI 环境

1. 登录 容器服务控制台 ，选择左侧导航栏中的**云原生 AI**。
2. 在 “AI 环境列表页”，单击**新建**，进入 “新建 AI 环境” 页面，参考以下提示进行设置。

- **环境名称**：自定义，可根据业务需求等信息进行命名，方便后续资源管理。
- **地域**：希望部署 AI 环境集群所在的地域。
- **集群类型**：希望部署 AI 环境集群的类型。
- **集群**：希望部署 AI 环境的特定集群。
- **部署组件**：希望在 AI 环境部署的组件，您也可以在环境新建完成后通过 组件管理 来安装或删除组件。

5. 单击**创建**即可创建 AI 环境。

查看 AI 环境

成功创建 AI 环境后，您可以在 AI 环境列表页查看 AI 环境。

删除 AI 环境

1. 登录 容器服务控制台 ，选择左侧导航栏中的**云原生 AI**。

2. 在 “AI 环境” 列表页面，选择目标环境行的最右侧的**删除**按钮。
3. 在 “删除 AI 环境” 弹窗中，阅读删除说明，单击**确定**删除 AI 环境。

AI 组件管理

最近更新时间：2022-10-12 11:37:14

简介

创建好 AI 环境以后，您可以自行拼装 AI 组件，搭建 AI 平台。本文档将介绍如何对 AI 组件进行增删改查：

> 注意：
> AI 组件底层基于 Helm Chart 实现。创建 AI 环境后，不建议您前往应用市场相关页面管理 AI 组件，请**直接在 AI 环境**管理 AI 组件，以免造成数据不一致的情况。

AI 组件列表

组件名称	业务场景	组件介绍
TF Operator	模型训练	安装后，用户可以运行 TF 单机 / 分布式训练任务。
MPI Operator	弹性训练	用户可以运行弹性训练任务，充分利用算力资源。
Fluid	缓存加速	Fluid 通过使用分布式缓存引擎（GooseFS/Alluxio）为云上应用提供数据预热与加速，同时可以保障缓存数据的可观测性，可迁移性和自动化的水平扩展。
Elastic Jupyter Operator	算法调试	为用户按需提供弹性的 Jupyter Notebook 服务，按需分配计算资源。

AI 组件生命周期管理

创建 AI 组件

1. 登录 容器服务控制台 ，选择左侧导航栏中的**云原生 AI**。
2. 在 “AI 环境” 列表页面，选择目标 AI 环境 ID，进入该 AI 环境 “基本信息” 页面。
3. 选择左侧菜单栏中的**组件管理**，进入 “组件列表” 页面。

4. 单击**新建**，进入 “新建 AI 组件” 页面，参考以下提示进行设置。
   主要参数信息如下：
   - 组件名：自定义组件名称。
   - 命名空间：安装组件的命名空间
   - Chart：对应组件的安装包，一次只能安装一个组件。
   - 参数：和组件配置有关的参数，创建组件以后仍然可以参考 “更新 AI 组件” 指引来更新相关参数。
5. 单击**完成**即可创建 AI 组件。

查看 AI 组件

成功创建 AI 环境后，可以在 AI 环境内查看已经安装的 AI 组件列表。

删除 AI 组件

1. 选择某个 AI 环境 id，进入 AI 环境的 “基本信息” 页面。
2. 选择左侧菜单栏中的**组件管理**，进入 “组件列表” 页面。
3. 选择需更新的组件所在行右侧的**删除**。
4. 在弹出的 “删除组件” 弹窗中，阅读删除说明并单击**确定**完成删除。

更新 AI 组件

1. 选择某个 AI 环境 id，进入 AI 环境的 “基本信息” 页面。
2. 选择左侧菜单栏中的**组件管理**，进入 “组件列表” 页面。
3. 选择需更新的组件所在行右侧的**更新配置**。
4. 在弹出的 “更新组件” 页面中，按需进行组件参数配置，并单击**完成**。

AI 组件列表

Fluid

最近更新时间：2022-06-21 11:13:21

简介

Fluid 是一个开源的 Kubernetes 原生分布式数据集编排和加速引擎，目前它由云原生计算基金会 CNCF 作为 Sandbox 项目托管。主要服务于云原生场景下的数据密集型应用，例如大数据应用、AI 应用等。通过定义数据集资源的抽象，实现如下功能：

- **数据集抽象原生支持**：将数据密集型应用所需基础支撑能力功能化，实现数据高效访问并降低多维管理成本。
- **云上数据预热与加速**：Fluid 通过使用分布式缓存引擎（GooseFS/Alluxio）为云上应用提供数据预热与加速，同时可以保障缓存数据的可观测性、可迁移性和自动化的水平扩展。
- **数据应用协同编排**：在云上调度应用和数据时，同时考虑两者特性与位置，实现协同编排，提升性能。
- **多命名空间管理支持**：用户可以创建和管理不同 namespace 的数据集。
- **异构数据源管理**：一次性统一访问不同来源的底层数据（对象存储 COS，HDFS 和 Ceph 等存储），适用于混合云场景。

重要概念

Dataset：数据集是逻辑上相关的一组数据的集合，会被运算引擎使用，例如大数据的 Spark，AI 场景的 TensorFlow。而这些数据智能的应用会创造工业界的核心价值。Dataset 的管理实际上也有多个维度，例如安全性，版本管理和数据加速。我们希望从数据加速出发，对于数据集的管理提供支持。

Runtime：实现数据集安全性，版本管理和数据加速等能力的执行引擎，定义了一系列生命周期的接口。可以通过实现这些接口，支持数据集的管理和加速。

GooseFSRuntime：来源于腾讯云 COS 团队 GooseFS，基于 Java 实现的支撑 Dataset 数据管理和缓存的执行引擎实现，支持对象存储 COS。GooseFS 为腾讯云产品，有专门的产品级支持，但是代码不开源。Fluid 通过管理和调度 GooseFS Runtime 实现数据集的可见性，弹性伸缩， 数据迁移。

AlluxioRuntime：来源于 Alluixo 社区，是支撑 Dataset 数据管理和缓存的执行引擎实现，支持 PVC，Ceph，CPFS 计算，有效支持混合云场景。但是 Alluxio 为开源社区方案，对于数据缓存的稳定性和性能优化，腾讯云会和社区一起推动，但是时效性和响应会有延时。Fluid 通过管理和调度 Alluxio Runtime 实现数据集的可见性，弹性伸缩， 数据迁移。

-	Alluxio	GooseFS

-	Alluxio	GooseFS
底层存储类型	PVC，Ceph，HDFS，CPFS，NFS...	OSS，EMR，PVC，Ceph，HDFS，CPFS，NFS...
支持方式	开源社区	腾讯云产品

组件安装

前置依赖

- Kubernetes 集群（version >= 1.14）
- 集群支持 CSI 功能

参数配置

在通过 Helm 部署的过程中，所有的配置项都集中于 `values.yaml` 。
以下是部分较为可能需要自定义的字段：

参数	描述	默认值
`workdir`	缓存引擎备份元数据地址	`/tmp`
`dataset.controller.image.repository`	Dataset Controller 镜像所在仓库	`ccr.ccs.tencentyun.com/f controller`
`dataset.controller.image.tag`	Dataset Controller 镜像的版本	`"v0.6.0-0bfc552"`
`csi.registrar.image.repository`	CSI registrar 镜像所在仓库	`"ccr.ccs.tencentyun.com/ driver-registrar"`
`csi.registrar.image.tag`	CSI registrar 镜像的版本	`"v1.2.0"`

参数	描述	默认值
`csi.plugins.image.repository`	CSI plugins 镜像所在仓库	`"ccr.ccs.tencentyun.com/`
`csi.plugins.image.tag`	CSI plugins 镜像的版本	`"v0.6.0-def5316"`
`csi.kubelet.rootDir`	kubelet root 文件夹	`"/var/lib/kubelet"`
`runtime.mountRoot`	缓存引擎 fuse mount 点的 Root 地址	`"/var/lib/kubelet"`
`runtime.goosefs.enable`	开启 GooseFS 缓存引擎支持	`"true"`
`runtime.goosefs.init.image.repository`	GooseFS 缓存引擎初始化镜像所在仓库	`"ccr.ccs.tencentyun.com/`
`runtime.goosefs.init.image.tag`	GooseFS 缓存引擎初始化镜像的版本	`"v0.6.0-0cd802e"`
`runtime.goosefs.controller.image.repository`	GooseFS 缓存引擎控制器镜像所在仓库	`"ccr.ccs.tencentyun.com/controller"`

参数	描述	默认值
`runtime.goosefs.controller.image.tag`	GooseFS 缓存引擎控制器镜像的版本	`"v0.6.0-bbf4ea0"`
`runtime.goosefs.runtime.image.repository`	GooseFS 缓存引擎镜像所在仓库	`"ccr.ccs.tencentyun.com/`
`runtime.goosefs.runtime.image.tag`	GooseFS 缓存引擎镜像的版本	`"v1.1.10"`
`runtime.goosefs.fuse.image.repository`	GooseFS 缓存引擎 Fuse 组件镜像所在仓库	`"ccr.ccs.tencentyun.com/`
`runtime.goosefs.fuse.image.tag`	GooseFS 缓存引擎 Fuse 组件镜像的版本	`"v1.1.10"`
`runtime.alluxio.runtimeWorkers`	Alluxio 缓存引擎控制器最大并发 worker 数量	`"3"`
`runtime.alluxio.portRange`	Alluxio 缓存引擎组件端口占用分配段	`"20000-26000"`
`runtime.alluxio.enable`	开启 Alluxio 缓存引擎支持	`"true"`

参数	描述	默认值
runtime.alluxio.init.image.repository	Alluxio 缓存引擎初始化镜像所在仓库	"ccr.ccs.tencentyun.com/
runtime.alluxio.init.image.tag	Alluxio 缓存引擎初始化镜像的版本	"v0.6.0-def5316"
runtime.alluxio.controller.image.repository	Alluxio 缓存引擎控制器镜像所在仓库	"ccr.ccs.tencentyun.com/controller"
runtime.alluxio.controller.image.tag	Alluxio 缓存引擎控制器镜像的版本	"v0.6.0-0cd802e"
runtime.alluxio.runtime.image.repository	Alluxio 缓存引擎镜像所在仓库	"ccr.ccs.tencentyun.com/
runtime.alluxio.runtime.image.tag	Alluxio 缓存引擎镜像的版本	"release-2.5.0-2-SNAPSHO
runtime.alluxio.fuse.image.repository	Alluxio 缓存引擎 Fuse 组件镜像所在仓库	"ccr.ccs.tencentyun.com/fuse"
runtime.alluxio.fuse.image.tag	Alluxio 缓存引擎 Fuse 组件镜像的版本	"release-2.5.0-2-SNAPSHO

最佳实践

请参见 Fluid 使用文档。

TF Operator

最近更新时间：2022-10-12 11:37:14

简介

TF-Operator 是 Kubeflow 社区开发，用以支持在 Kubernetes 上部署执行 TensorFlow 分布式训练任务的组件。

在部署完成之后，用户可以创建、查看、删除 TFJob。

前置依赖

Kubernetes 集群（version >= 1.16）

部署

在通过 Helm 部署的过程中，所有的配置项都集中于 `values.yaml` 。

以下是部分较为可能需要自定义的字段：

参数	描述	默认值
`image.repository`	TF-Operator 镜像所在仓库	`ccr.ccs.tencentyun.com/kubeflow-oteam/tf-operator`
`image.tag`	TF-Operator 镜像的版本	`"latest"`
`namespace.create`	是否为 TF-Operator 创建独立的命名空间	`true`
`namespace.name`	部署 TF-Operator 的命名空间	`"tf-operator"`

最佳实践

请参见 运行 TF 训练任务。

MPI Operator

最近更新时间：2022-10-12 11:37:14

简介

MPI-Operator 是 Kubeflow 社区开发，用于支持以 Horovod 为代表的数据并行分布式训练在 Kubernetes 集群上部署运行的组件。

在部署完成之后，用户可以创建、查看、删除 MPIJob。

前置依赖

Kubernetes 集群（version >= 1.16）

部署

在通过 Helm 部署的过程中，所有的配置项都集中于 `values.yaml` 。

以下是部分较为可能需要自定义的字段：

参数	描述	默认值
`image.repository`	MPI-Operator 镜像所在仓库	`ccr.ccs.tencentyun.com/kubeflow-oteam/mpi-operator`
`image.tag`	MPI-Operator 镜像的版本	`"latest"`
`namespace.create`	是否为 MPI-Operator 创建独立的命名空间	`true`
`namespace.name`	部署 MPI-Operator 的命名空间	`"mpi-operator"`

Elastic Jupyter Operator

最近更新时间：2022-10-12 16:05:09

简介

elastic-jupyter-operator 是 Kubernetes 原生的弹性 Jupyter 服务。为用户按需提供弹性的 Jupyter Notebook 服务。elastic-jupyter-operator 提供以下特性：

- GPU 空闲时自动释放资源到 Kubernetes 集群。
- 资源延迟申请，在使用时按需申请对应 CPU/内存/GPU 资源。
- 多 Jupyter 共享资源池，提高资源利用率。

部署

在通过 Helm 部署过程中，所有的配置项都集中于 `values.yaml` 。

以下是部分较为可能需要自定义的字段：

参数	描述	默认值
`image.repository`	镜像所在仓库	`ccr.ccs.tencentyun.com/kubeflow-oteam/elastic-jupyter-operator`
`image.tag`	镜像的版本	`"v0.1.1"`
`namespace.name`	命名空间	`"enterprise-gateway"`

使用

> 说明：
> 更多详细说明，请参见 使用文档。

1. 执行以下命令，创建一个 Jupyter Gateway CR：

```
kubectl apply -f ./config/samples/kubeflow.tkestack.io_v1alpha1_jupytergateway.yaml
```

YAML 文件内容如下：

```
apiVersion: kubeflow.tkestack.io/v1alpha1
kind: JupyterGateway
metadata:
name: jupytergateway-sample
spec:
cullIdleTimeout: 3600
```

其中 `cullIdleTimeout` 是一个配置项，在 Kernel 空闲指定 `cullIdleTimeout` 秒内，会由 Gateway 回收对应 Kernel 以释放资源。

2. 执行以下命令，创建一个 Jupyter Notebook CR 实例，并且指定对应的 Gateway CR：

```
kubectl apply -f ./config/samples/kubeflow.tkestack.io_v1alpha1_jupyternotebook.y
aml
```

YAML 文件内容如下：

```
apiVersion: kubeflow.tkestack.io/v1alpha1
kind: JupyterNotebook
metadata:
name: jupyternotebook-sample
spec:
gateway:
name: jupytergateway-sample
namespace: default
```

3. 集群上所有资源如下所示：

```
NAME READY STATUS RESTARTS AGE
pod/jupytergateway-sample-6d5d97949c-p8bj6 1/1 Running 2 11d
pod/jupyternotebook-sample-5bf7d9d9fb-nq9b8 1/1 Running 2 11d
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/jupytergateway-sample ClusterIP 10.96.138.111 <none> 8888/TCP 11d
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 31d
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/jupytergateway-sample 1/1 1 1 11d
deployment.apps/jupyternotebook-sample 1/1 1 1 11d
NAME DESIRED CURRENT READY AGE
replicaset.apps/jupytergateway-sample-6d5d97949c 1 1 1 11d
replicaset.apps/jupyternotebook-sample-5bf7d9d9fb 1 1 1 11d
```

4. 通过 NodePort、 `kubectl port-forward` 、ingress 等方式将 Notebook CR 对外暴露提供服务，这里以 `kubectl port-forward` 为例，执行命令如下：

```
kubectl port-forward jupyternotebook-sample-5bf7d9d9fb-nq9b8 8888
```

API 文档

请参见 API 文档。

模型训练
运行 TF 训练任务

最近更新时间：2023-05-19 17:13:05

本文为您介绍如何运行 TF 训练任务。

前提条件

- AI 环境中已安装 TF Operator。
- AI 环境中有 GPU 资源。

操作步骤

以下操作指南参考 `TF-Operator` 官方提供的 PS/Worker 模式的 分布式训练案例。

准备训练代码

本示例中使用 Kubeflow 官方提供的示例代码 dist_mnist.py。

制作训练镜像

镜像的制作过程较简单，只需基于一个 TensorFlow 1.5.0 的官方镜像，并将代码复制到镜像内，并配置好 `entrypoint` 即可。

> 说明：
> 如果不配置 entrypoint，也可以在提交 TFJob 时配置容器的启动命令。

任务提交

1. 准备一个 TFJob 的 YAML 文件，定义2个 PS 和4个 Worker。

> 注意
> 用户需要用上传后的训练镜像地址替换 `<训练镜像>` 所在占位。

```
apiVersion: "kubeflow.org/v1"
kind: "TFJob"
metadata:
name: "dist-mnist-for-e2e-test"
spec:
tfReplicaSpecs:
PS:
replicas: 2
restartPolicy: Never
template:
spec:
containers:
- name: tensorflow
image: <训练镜像>
Worker:
replicas: 4
restartPolicy: Never
template:
spec:
containers:
- name: tensorflow
image: <训练镜像>
```

2. 执行以下命令，通过 `kubectl` 提交该 TFJob：

```
kubectl create -f ./tf_job_mnist.yaml
```

3. 执行以下命令，查看任务状态：

```
kubectl get tfjob dist-mnist-for-e2e-test -o yaml
kubectl get pods -l pytorch_job_name=pytorch-tcp-dist-mnist
```

运行 PyTorch 训练任务

最近更新时间：2023-05-19 17:08:09

本文为您介绍如何运行 PyTorch 训练任务。

前提条件

- AI 环境中已安装 PyTorch Operator。
- AI 环境中有 GPU 资源。

操作步骤

以下操作指南参考 `PyTorch-Operator` 官方提供的分布式训练 案例。

准备训练代码

本示例使用 Kubeflow 官方提供的示例代码 mnist.py。

制作训练镜像

训练镜像的制作过程较简单，只需基于一个 PyTorch 1.0 的官方镜像，将代码复制到镜像内，并配置好 `entrypoint` 即可。（如果不配置 entrypoint，用户也可以在提交 PyTorchJob 时配置启动命令。）

> 注意：
> 训练代码基于 PyTorch 1.0 构建，由于 PyTorch 在版本间可能存在 API 不兼容的问题，上述训练代码在其他版本的 PyTorch 环境下可能需要自行调整代码。

任务提交

1. 准备一个 PyTorchJob 的 YAML 文件，定义1个 Master Worker 和1个 Worker。

> 注意
> - 用户需要用上传后的训练镜像地址替换 `<训练镜像>` 所在占位。
> - 由于在资源配置中设置了 GPU 资源，在 `args` 为训练配置的 `backend` 为 `"nccl"` ；在未使用（Nvidia）GPU 的任务中，请使用其他（例如 gloo）backend。

```
apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
name: "pytorch-dist-mnist-nccl"
spec:
pytorchReplicaSpecs:
Master:
replicas: 1
restartPolicy: OnFailure
template:
metadata:
annotations:
sidecar.istio.io/inject: "false"
spec:
containers:
- name: pytorch
image: <训练镜像>
args: ["--backend", "nccl"]
resources:
limits:
nvidia.com/gpu: 1
Worker:
replicas: 1
restartPolicy: OnFailure
template:
metadata:
annotations:
sidecar.istio.io/inject: "false"
spec:
containers:
- name: pytorch
image: <训练镜像>
args: ["--backend", "nccl"]
resources:
limits:
nvidia.com/gpu: 1
```

2. 执行以下命令，通过 `kubectl` 提交该 PyTorch Job：

```
kubectl create -f ./pytorch_job_mnist_nccl.yaml
```

3. 执行以下命令，查看该 PyTorch Job：

```
kubectl get -o yaml pytorchjobs pytorch-dist-mnist-nccl
```

4. 执行以下命令，查看 PyTorch 任务创建的相关 Pod：

```
kubectl get pods -l pytorch_job_name=pytorch-dist-mnist-nccl
```

Prometheus 监控服务
Prometheus 监控概述

最近更新时间：2023-02-02 17:05:22

产品简介

Prometheus 监控服务（Managed Service for Prometheus，TMP）是针对云原生服务场景进行优化的监控和报警解决方案，全面支持开源 Prometheus 的监控能力，为用户提供轻量、稳定、高可用的云原生 Prometheus 监控服务。借助 TMP，您无需自行搭建 Prometheus 监控系统，也无需关心数据存储、数据展示、系统运维等问题，只需简单配置即可享受支持多集群的高性能Prometheus 监控服务。

Prometheus 简介

Prometheus 是一套开源的系统监控报警框架，其彻底颠覆了传统监控系统的测试和告警模型，是一种基于中央化的规则计算、统一分析和告警的新模型。作为云原生计算基金会 Cloud Native Computing Foundation 中受欢迎度仅次于 Kubernetes 的项目，Prometheus 依靠其强劲的单机性能、灵活的 PromSQL、活跃的社区生态，逐渐成为云原生时代最核心的监控组件。

Prometheus 优势

- 支持强大的多维数据模型。
- 内置灵活的查询语言 PromQL。
- 支持全面监控。
- 拥有良好的开放性。
- 支持通过动态服务或静态配置发现采集目标。

开源 Prometheus 不足

- 原生 Prometheus 为单点架构，不提供集群化功能，单机性能瓶颈使其无法作为大规模集群下的监控方案。
- 无法便捷地实现动态的扩缩容和负载均衡。
- 部署使用技术门槛高。

腾讯云 Prometheus 监控与开源 Prometheus 对比

对比项	腾讯云 Prometheus 监控	开源 Prometheus
场景	针对容器云原生场景优化，支持利用 集成中心 实现非容器场景的监控	面向多种场景
量级	超轻量级	内存占用高

对比项	腾讯云 Prometheus 监控	开源 Prometheus
稳定性	高于原生	无法保证
可用性	高	低
数据存储能力	无限制	受限于本地磁盘
超大集群监控	支持	不支持
数据可视化	基于 Grafana 提供优秀的可视化能力，且支持一个 Grafana 同时查看多个监控实例的数据	原生的 Prometheus UI 可视化能力有限
开源生态	完全兼容	原生支持
使用门槛	低	高
成本	低	高
跨集群采集	支持	不支持
跨地域跨 VPC 采集	支持采集其它地域和 VPC 的集群，可 Prometheus 监控服务关联集群	不支持
告警策略配置	丰富的 告警和通知模板	完全需要用户手工填写

产品优势

完全兼容 Prometheus 配置和核心 API，保留 Prometheus 原生特性和优势

支持自定义多维数据模型。

内置灵活的查询语言 PromQL。

支持通过动态服务或静态配置发现采集目标。

兼容核心 PrometheusAPI。

支持超大规模集群的监控

在针对单机 Prometheus 的性能压测中，当 Series 数量超过300万（每个 Label 以及值的长度固定为10个字符）时，Prometheus 的内存增长非常明显，需要20GB及以上的 Memory，因此需要使用较大内存的机器来运行。

腾讯云通过自研的分片技术，支持超大规模集群的监控。

支持在同一实例里进行跨 VPC 的多集群监控

支持同一监控实例内关联多个集群。支持监控其它 VPC 的集群。

支持模板化管理配置

针对多实例多集群的监控，Prometheus 监控服务支持配置监控模板，用户可以使用模板一键完成对多集群的统一监控。

超轻量、无侵入式的监控

腾讯云Prometheus 监控相较于开源 Prometheus 更加轻量化，开源 Prometheus 需要占用用户16GB - 128GB内存，但Prometheus 监控部署在用户集群内的只有非常轻量的 Agent，监控100个节点的集群约只占用20M内存，且无论集群多大，也不会超过1G的内存占用。

当用户关联集群后，Prometheus 监控会自动在用户的集群内部署 Agent，用户无需安装任何组件即可开始监控业务，超轻量级的 Agent 不会对用户集群内的业务和组件产生任何影响。

支持实时动态扩缩，满足弹性需求

腾讯云Prometheus 监控采用腾讯云自研的分片和调度技术，可以针对采集任务进行实时的动态扩缩，满足用户的弹性需求，同时支持负载均衡。

高可用性

采用技术手段，保障数据不断点，不缺失，为用户提供高可用的监控服务。

接入成本低

控制台支持产品化的配置文件编写，用户无需精通 Prometheus 即可轻松使用。针对有 Prometheus 实际使用经验的用户，腾讯云也提供原生 YAML 文件提交配置的方式，方便用户自定义高级功能完成个性化监控。

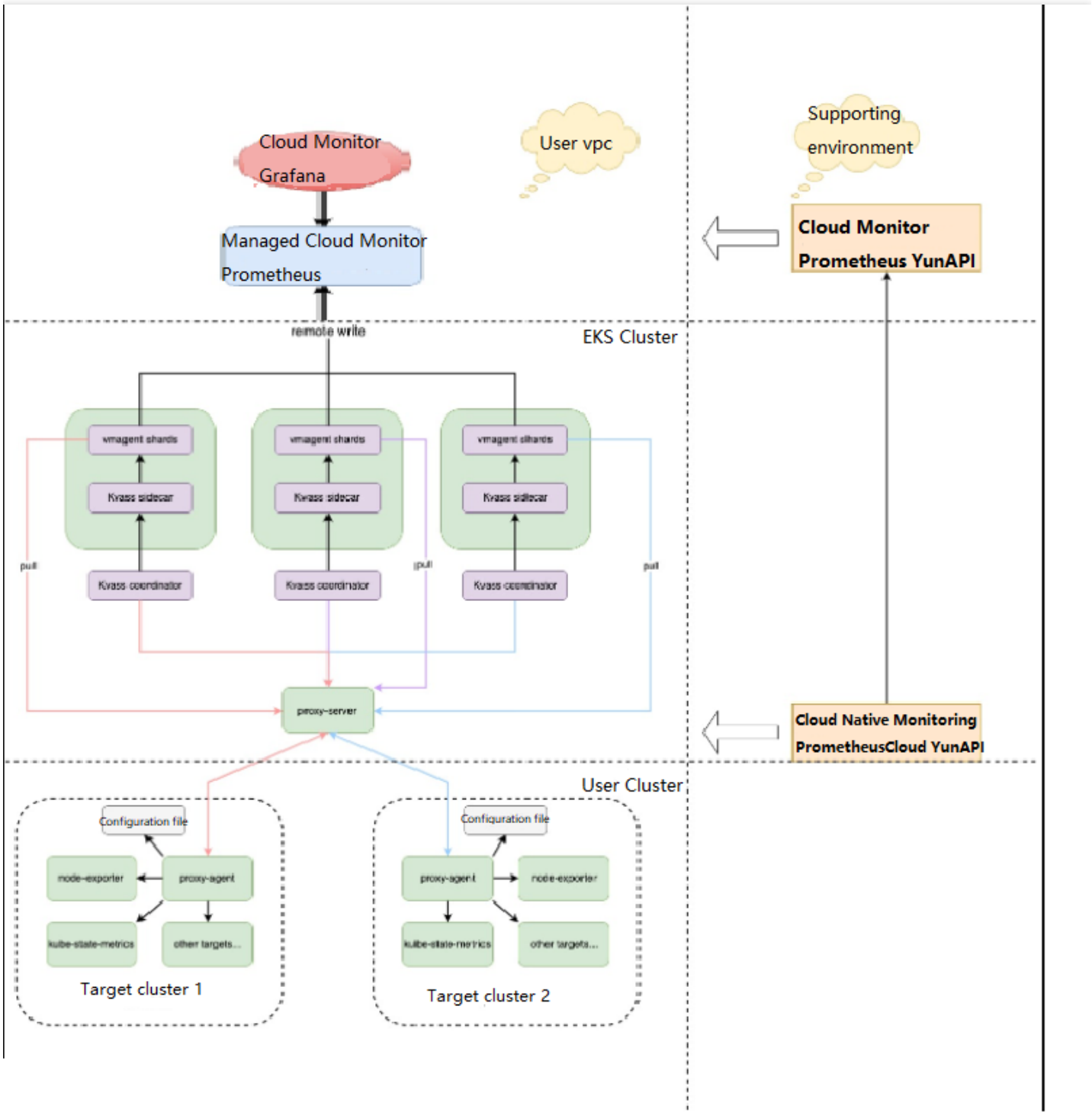
更多优势请参考。

产品架构

腾讯云 Prometheus 监控服务作为超轻量、高可用、无侵入式的监控系统。

- 在用户集群内仅包含一个轻量级的 Agent
- 采集器是在用户账号下创建的 TKE Serverless 集群，对用户原生集群没有任何影响
- 监控数据存储和展示都是独立的模块
- 云监控的 Grafana 支持对接多个监控实例，实现统一的查看

产品架构如下图所示：



Prometheus 监控支持跨地域跨 VPC 的多集群监控、支持 VPC 网络中非集群内业务的监控、支持超大集群的监控并实时进行监控组件的扩缩容，保障高可用的监控服务。

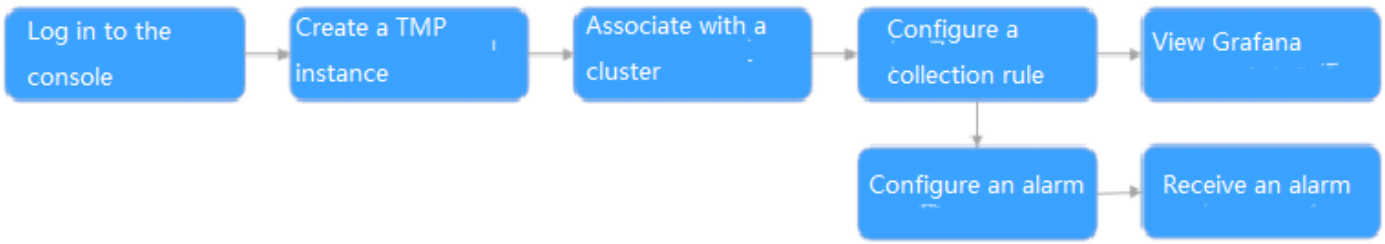
在用户关联集群后，Prometheus 监控服务将默认添加社区主流的采集配置，用户在不做任何个性化配置的基础上能够开箱即用。

此外，TMP 预设常用的 Grafana Dashboard 及告警规则模板。

使用流程

用户需要登录腾讯云账号，进入 Prometheus 监控控制台：

1. 创建监控实例。
2. 关联集群。在新创建的监控实例下完成集群关联操作，此时系统会自动在用户集群内完成 Agent 的部署，在用户的新建 TKE Serverless 集群里面内完成监控组件的部署，用户无需进行任何插件的安装。
3. 配置采集规则。成功关联集群后用户可以按照实际需求灵活配置数据采集规则，并按需要配置告警规则，配置完成后即可打开 Grafana 查看监控数据。



关键概念解释

- **监控实例**：一个监控实例对应一整套监控服务，拥有独立的可视化页面，一个监控实例下可以关联同一 VPC 下的多个集群并完成对多个集群的统一监控。
- **集群**：通常指用户在腾讯云上的 TKE 或 TKE Serverless 集群。
- **关联集群**：指将监控实例与用户的集群进行关联的操作。
- **采集规则**：指用户自定义的监控数据采集的规则。
- **Job**：在 Prometheus 中，一个 Job 即为一个采集任务，定义了一个 Job 工作负载下所有监控目标的公共配置，多个 Job 共同组成采集任务的配置文件。
- **Target**：指通过静态配置或者服务发现得到的需要进行数据采集的采集对象。例如，当监控 Pod 时，其 Target 即为 Pod 中的每个 Container。
- **Metric**：用于记录监控指标数据，所有 Metrics 皆为时序数据并以指标名字作区分，即每个指标收集到的样本数据包含至少三个维度（指标名、时刻和指标值）的信息。

- **Series**：一个 Metric+Label 的集合，在监控面板中表现为一条直线。

应用场景

腾讯云 Prometheus 监控服务主要针对容器云原生业务场景进行监控，除了实现容器和 Kubernetes 的主流监控方案之外，还灵活支持用户按照自己的业务进行自定义监控，通过逐步完善不同场景的预设面板，不断总结行业最佳实践，来帮助用户完成监控数据的多维分析以及数据的个性化展示， Prometheus 监控服务致力于成为容器化场景下的最佳监控解决方案。

产品定价

Prometheus 监控服务定价详情见 按量计费。
此外，目前使用 Prometheus 监控服务时将会在用户的账户下创建 TKE Serverless 集群，以及内外网 负载均衡 CLB 资源，按用户实际使用的云资源收费。具体创建的资源和参考价格可参考 云原生监控资源使用情况。

创建监控实例

最近更新时间：2022-12-27 15:28:16

操作场景

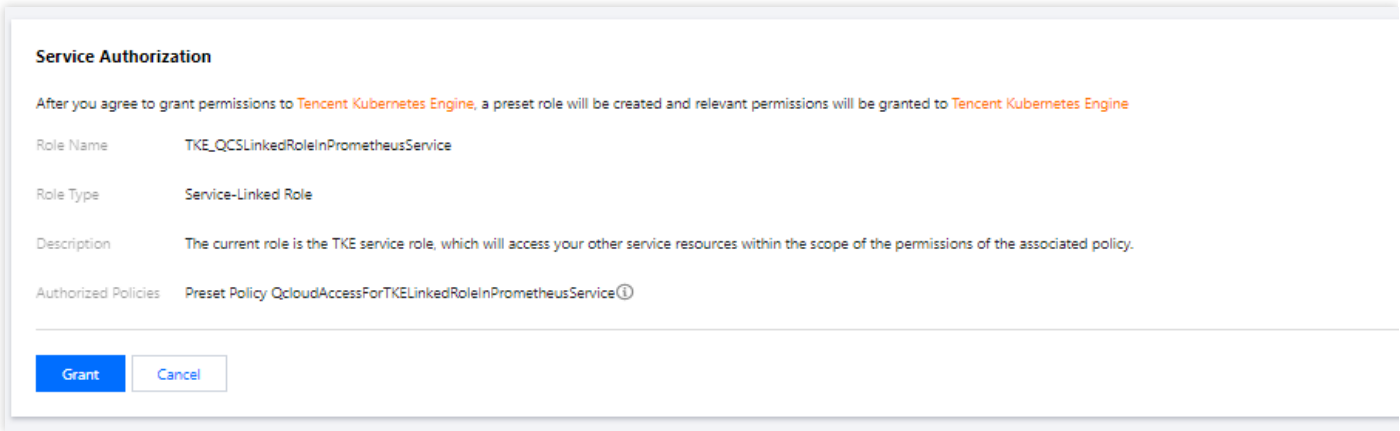
您在创建完 Prometheus 监控服务实例后可以将当前地域中的集群与此实例相关联。关联同一 Prometheus 实例中的集群可以实现监控指标的联查和统一告警。目前 Prometheus 监控服务功能服务支持的集群类型包括托管集群、独立集群、Serverless 集群以及边缘集群。本文介绍如何在腾讯云容器服务控制台 中创建和管理监控实例，您可根据以下指引进行监控实例的创建。

操作步骤

服务授权

初次使用 Prometheus 监控服务功能服务需要授权名为 TKE_QCSLinkedRoleInPrometheusService 的服务相关角色，该角色用于授权 Prometheus 监控服务功能服务对相关云产品的访问权限。

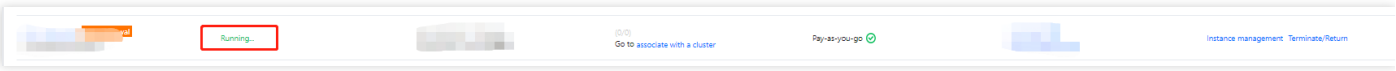
1. 登录 容器服务控制台 ，选择左侧导航栏中的 **Prometheus 监控**，弹出**服务授权**窗口。
2. 单击**前往访问管理**，进入角色管理页面。
3. 单击**同意授权**，完成身份验证后即可成功授权。如下图所示：



创建监控实例

1. 登录 容器服务控制台 ，单击左侧导航栏中的 **Prometheus 监控**。
2. 进入 Prometheus 监控服务实例列表页面，单击实例列表上方的**新建**。
3. 新建会跳转到 **Prometheus 监控服务** 页面。
4. 可根据自己的实际情况购买对应的实例，新建的购买参数详情请参见 创建实例。

5. 单击**完成**，即可完成创建。此时单击“前往关联容器服务”，查看容器侧的 Prometheus 实例列表。
6. 您可在该列表页面查看实例创建进度。当实例状态为“运行中”时，表示当前实例已成功创建并处于可用状态。如下图所示：



> 说明：
> 若实例创建花费时间过长，或显示状态为异常，可 提交工单 联系我们。

关联集群

最近更新时间：2023-05-19 15:21:25

操作场景

本文档介绍如何在 Prometheus 监控服务中关联集群与监控实例，关联成功后即可编辑数据采集规则等配置。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 容器服务控制台，并创建集群。
- 已创建 监控实例。

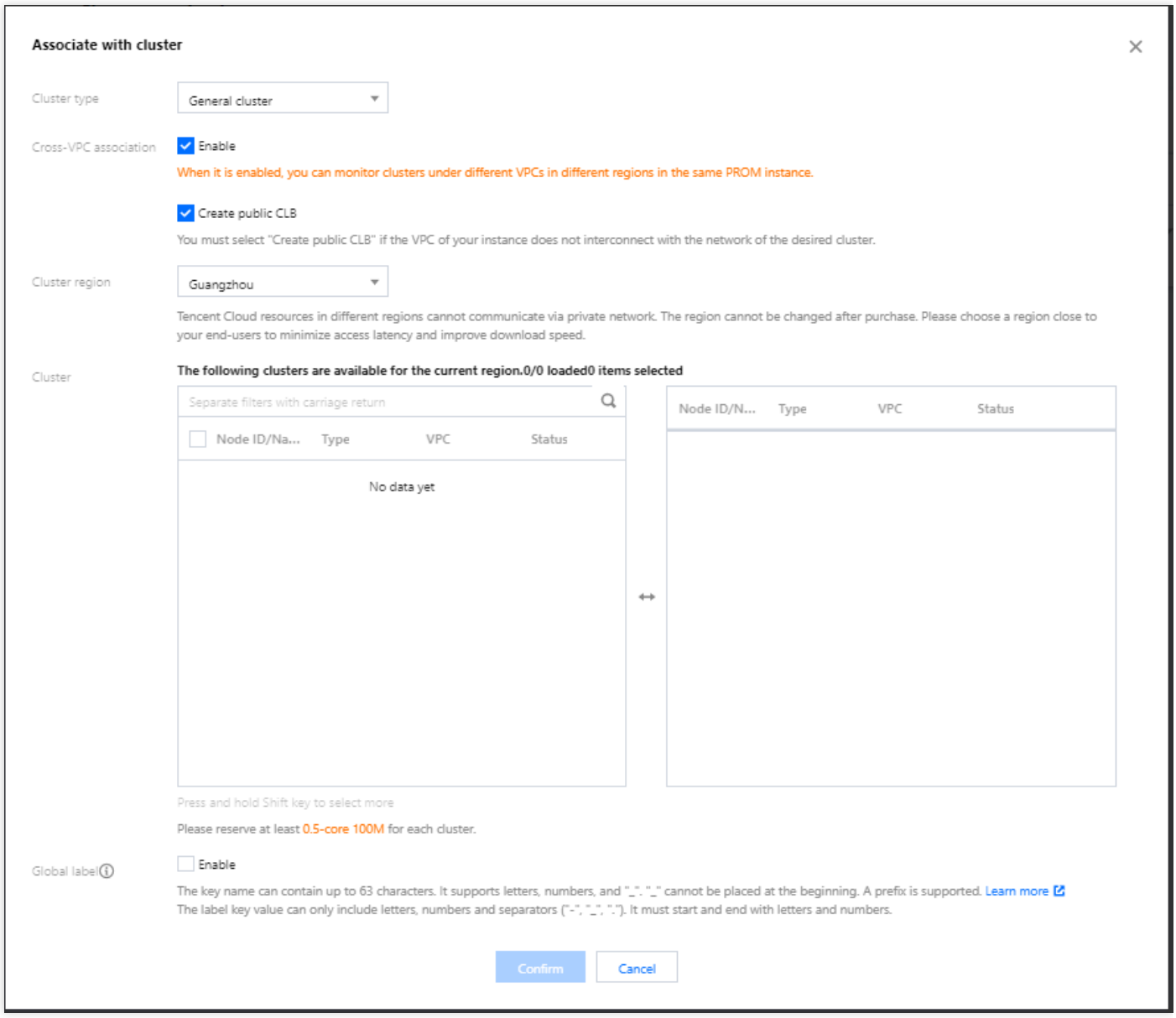
操作步骤

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。

1. 登录 容器服务控制台，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“集群监控”页面，单击**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、Serverless 集群、边缘集群、注册集群。
- **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。
  - 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。

3. **地域**：选择集群所在地域。
4. **集群**：选择需要关联的集群，支持多选。
5. **全局标记**：用于给每个监控指标打上相同的键值对。
6. 单击**确定**即可将所选集群和当前监控实例关联。

解除关联

1. 登录 容器服务控制台 ，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“关联集群”页面，单击实例右侧的**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。

数据采集配置

最近更新时间：2023-05-19 15:16:47

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

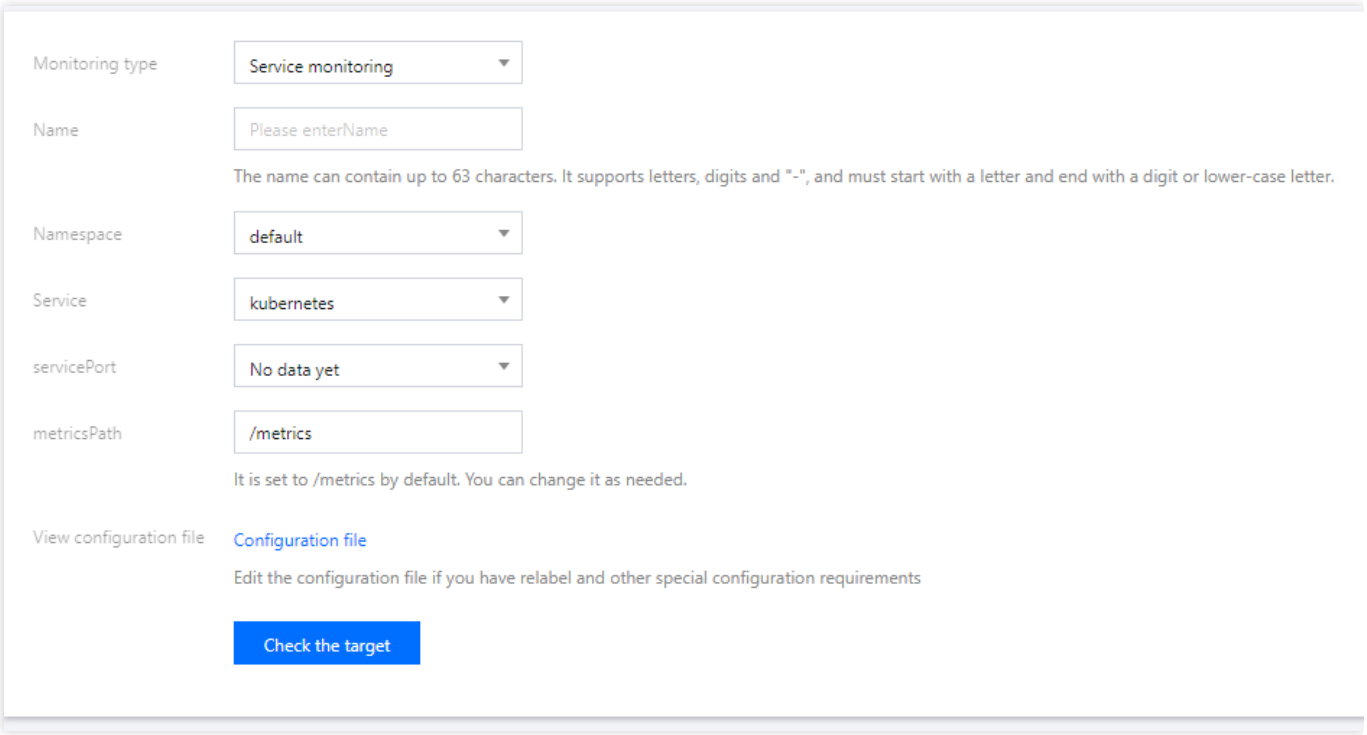
配置数据采集

1. 登录 容器服务控制台，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“集群监控”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. 在“数据采集配置”页中，单击“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，用来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来监控您的业务数据。
   - 通过控制台新增配置
   - 通过 yaml 文件新增配置

   监控 Service

   i. 单击**新增**。

ii. 在“新建采集配置”弹窗中，填写配置信息。如下图所示：

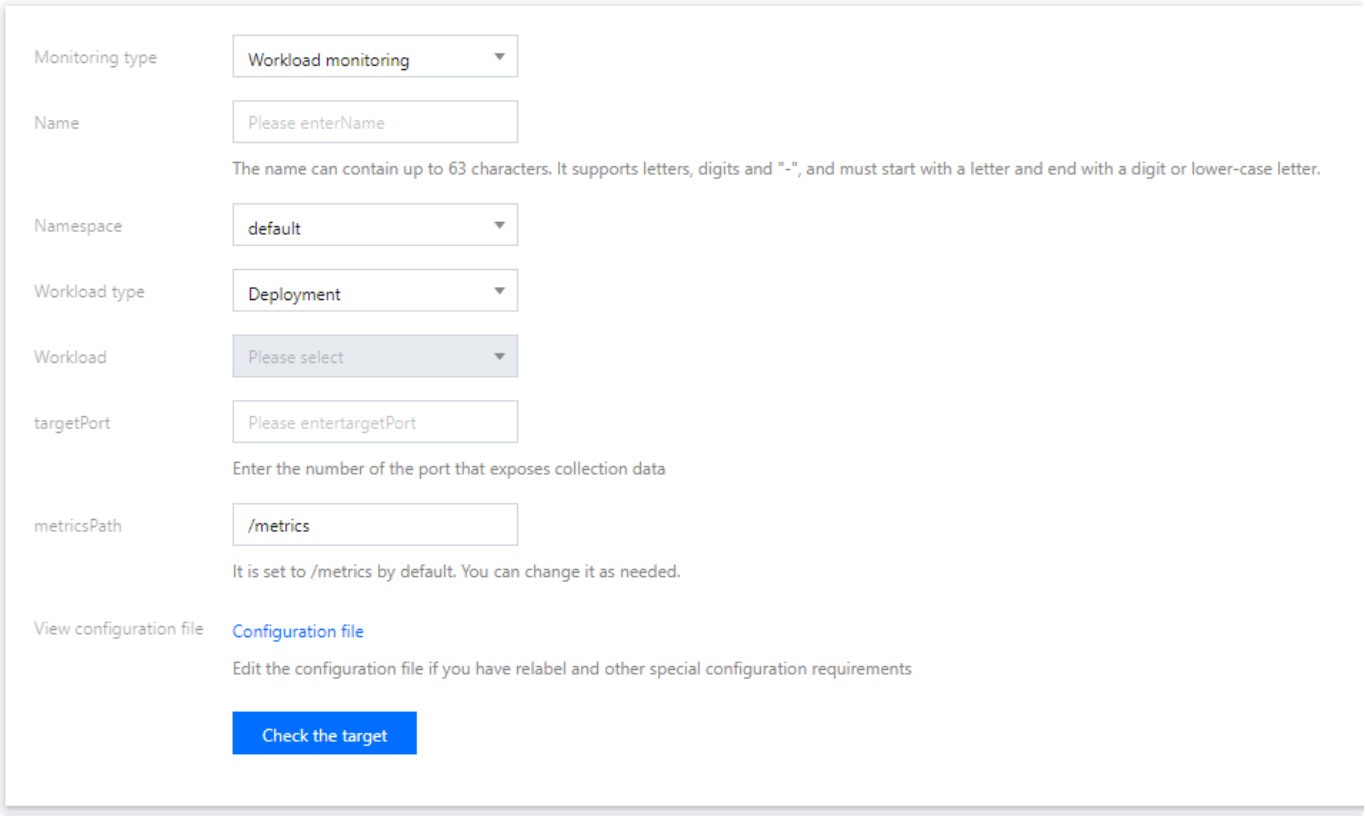


- **监控类型**：选择 “Service监控” 。
- **名称**：填写规则名称。
- **命名空间**：选择 Service 所在的命名空间。
- **Service**：选择需要监控的 Service 名称。
- **ServicePort**：选择相应的 Port 值。
- **MetricsPath**：默认为 `/metrics` ，您可根据需求执行填写采集接口。
- **查看配置文件**：单击“配置文档”可查看当前配置文件。如果您有 relabel 等相关特殊配置的需求，可以在配置文件内进行编辑。
- **探测采集目标**：单击**探测采集目标**，即可显示当前采集配置下能够采集到的所有 target 列表，您可通过此功能确认采集配置是否符合您的预期。

监控工作负载

i. 单击**新增**。

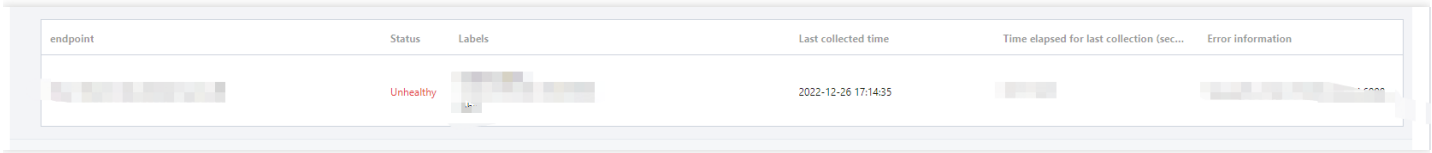
ii. 在“新建采集配置”弹窗中，填写配置信息。如下图所示：



- **监控类型**：选择 “工作负载监控” 。
- **名称**：填写规则名称。
- **命名空间**：选择工作负载所在的命名空间。
- **工作负载类型**：选择需要监控的工作负载类型。
- **工作负载**：选择需要监控的工作负载。
- **targetPort**：填写暴露采集指标的目标端口，通过端口找到采集目标。若端口填写错误将无法获取到正确的采集目标。
- **MetricsPath**：默认为 `/metrics` ，您可根据需求执行填写采集接口。
- **查看配置文件**：单击“配置文档”可查看当前配置文件。如果您有 relabel 等相关特殊配置的需求，可以在配置文件内进行编辑。
- **探测采集目标**：单击**探测采集目标**，即可显示当前采集配置下能够采集到的所有 target 列表，您可通过此功能确认采集配置是否符合您的预期。

5. 单击**确认**完成配置。
6. 在该实例的“数据采集配置”页中，查看采集目标状态。

targets（3/3）表示（实际抓取的 targets 数为3 / 探测的采集目标数为3）。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。单击字段值（3/3）即可查看采集目标的详细信息。如下图所示 down 的失败状态：



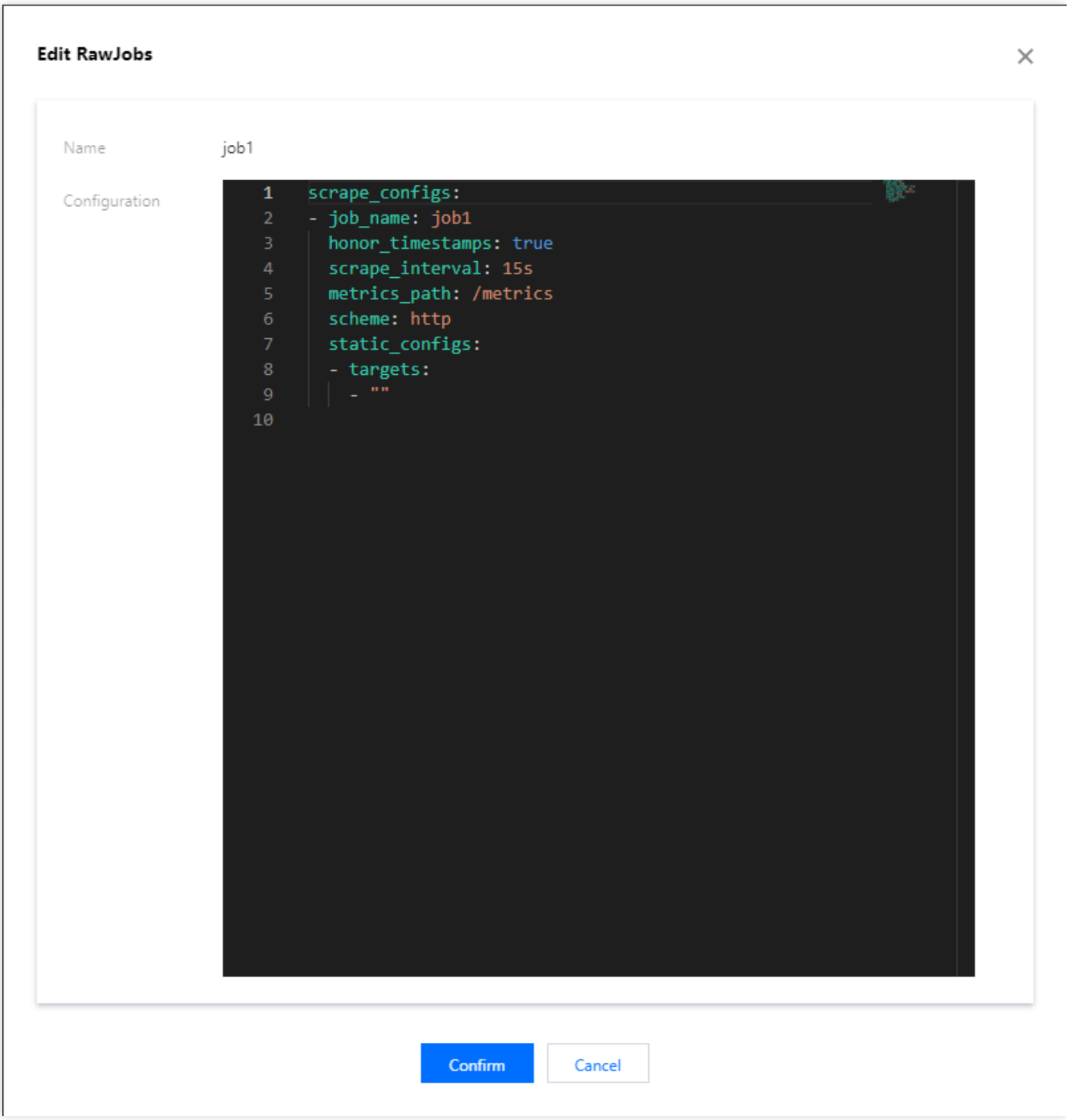
您还可以在该实例的“集群监控”页中，单击集群名称右侧的**更多 > 查看采集目标**，查看该集群下所有的采集目标情况。

查看已有配置

> 注意：
> 查看已有配置的 YAML 文件仅支持“自定义监控”，不支持“基础监控”。基础监控的数据采集配置全已产品化，您只需要通过**点击/勾选**来增加/减少监控指标。

1. 登录 容器服务控制台，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“集群监控”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“自定义监控”，单击右侧的**编辑**。

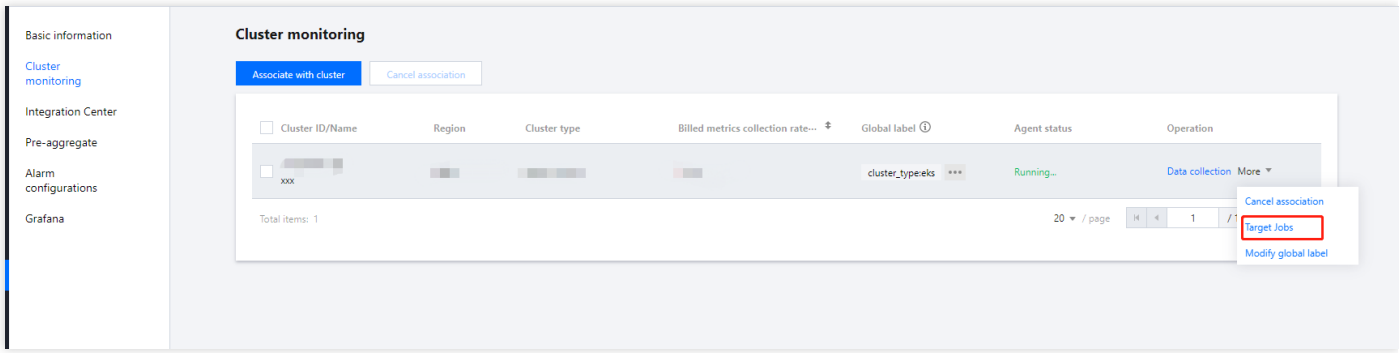
4. 在弹出的“编辑RawJobs”窗口，查看 yaml 文件中当前配置的所有监控对象。如下图所示：



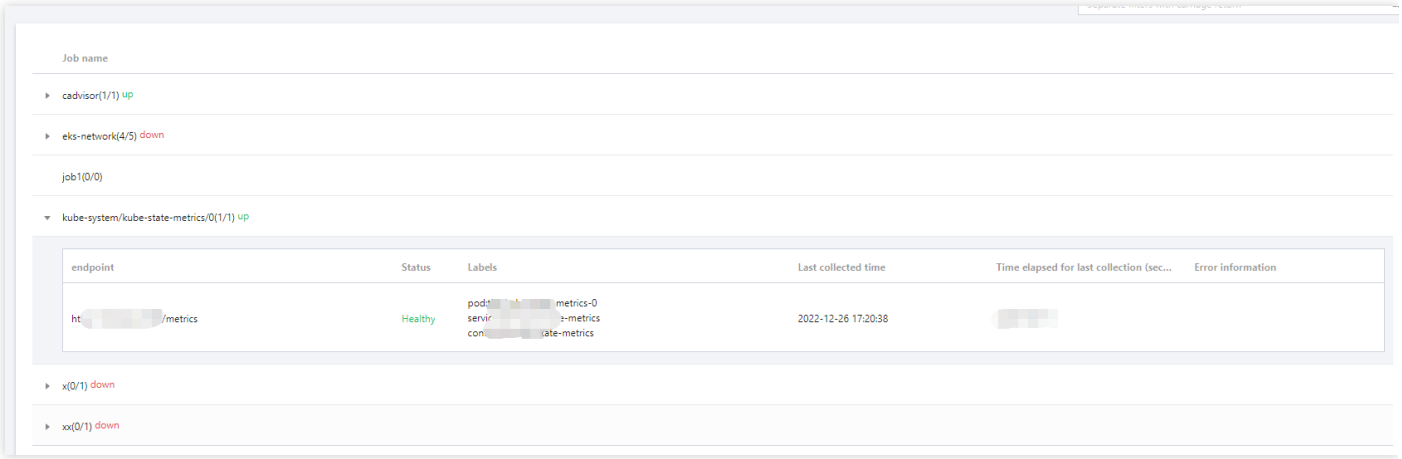
查看采集目标

1. 登录 容器服务控制台，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“集群管理”页面，单击实例右侧的**查看采集目标**。



4. 在 Targets 列表页即可查看当前数据拉取状态。如下图所示：



说明：

- 状态为“不健康”的 endpoints 默认显示在列表上方，方便及时查看。
- 实例中“采集目标”页面支持检索，可以按资源属性进行过滤。

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：2023-05-06 19:41:07

注意

TMP 已于2022年10月27日调整免费指标的免费存储时长为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。具体收费规则可参考 计费说明。

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。

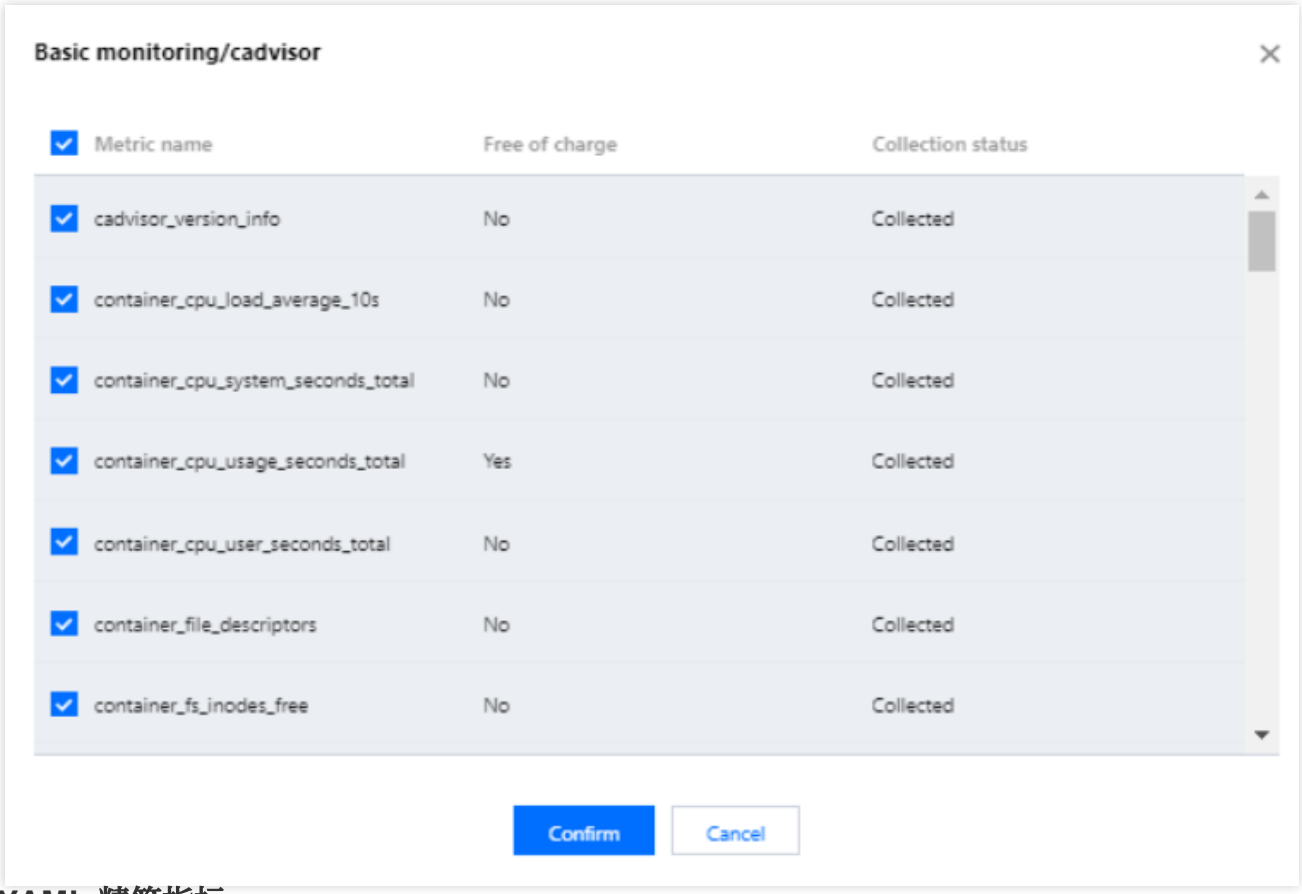
已将需要 监控的集群关联到相应实例 中。

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 容器服务控制台，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“集群监控”页面，单击集群右侧的**数据采集配置**，进入采集配置列表页。
4. 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”。
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，建议您取消勾选付费指标，以免造成额外的成本。仅基础监控提供免费的监控指标，完整的免费指标详情见 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

TMP 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。

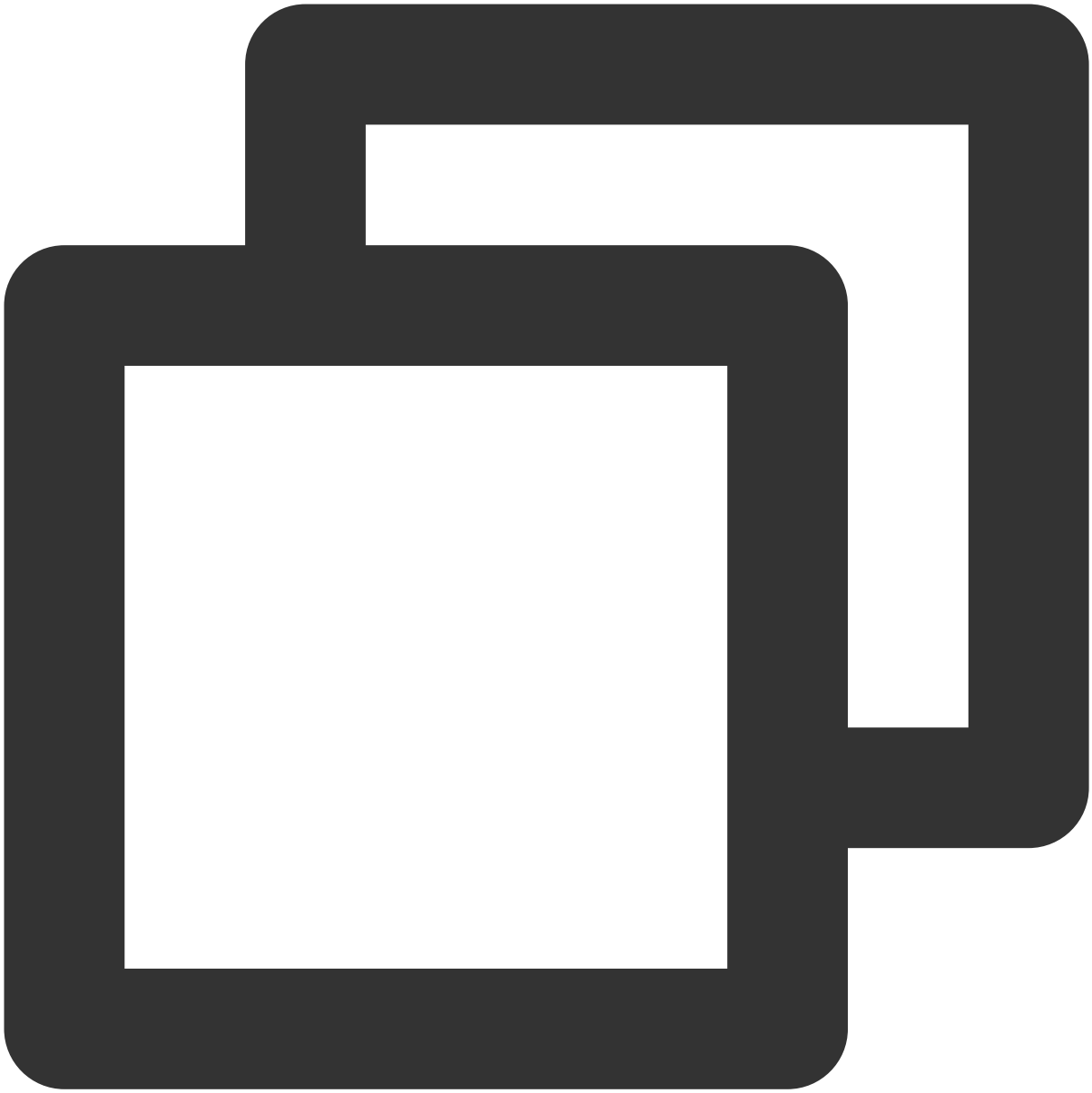
1. 登录 容器服务控制台，选择左侧导航栏中的 Prometheus 监控。
2. 在监控实例列表页，选择需要配置数据采集规则的实例名称，进入该实例详情页。
3. 在“集群监控”页面，单击集群右侧的**数据采集配置**，进入采集配置列表页。
4. 单击实例右侧的**编辑**查看指标详情。

ServiceMonitor 和 PodMonitor

原生 Job

ServiceMonitor 和 PodMonitor 的过滤配置字段相同，本文以 ServiceMonitor 为例。

ServiceMonitor 示例：

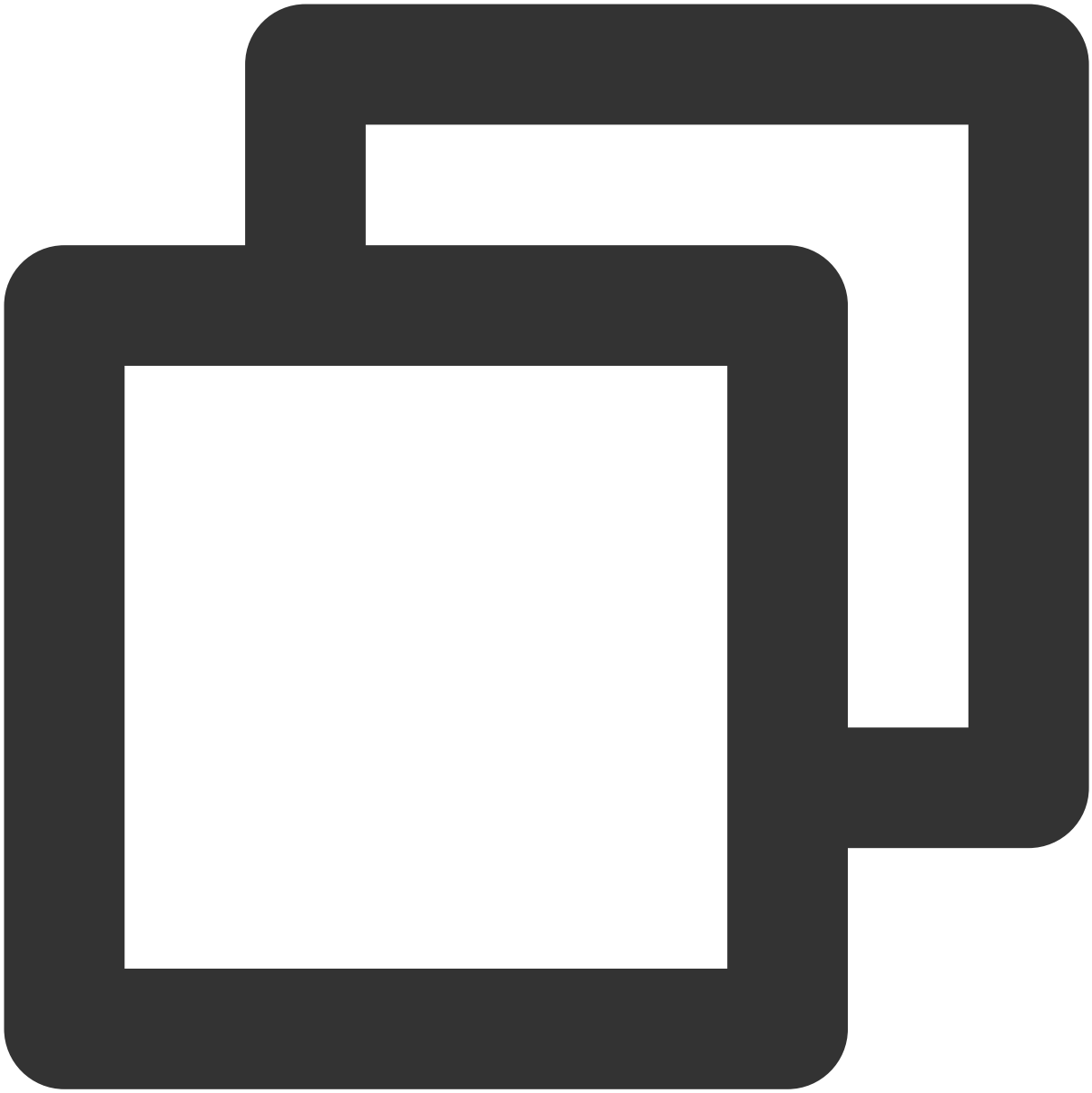


```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 1.9.7
  name: kube-state-metrics
  namespace: kube-system
spec:
  endpoints:
  - bearerTokenSecret:
```

```
      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 30
    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
  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：

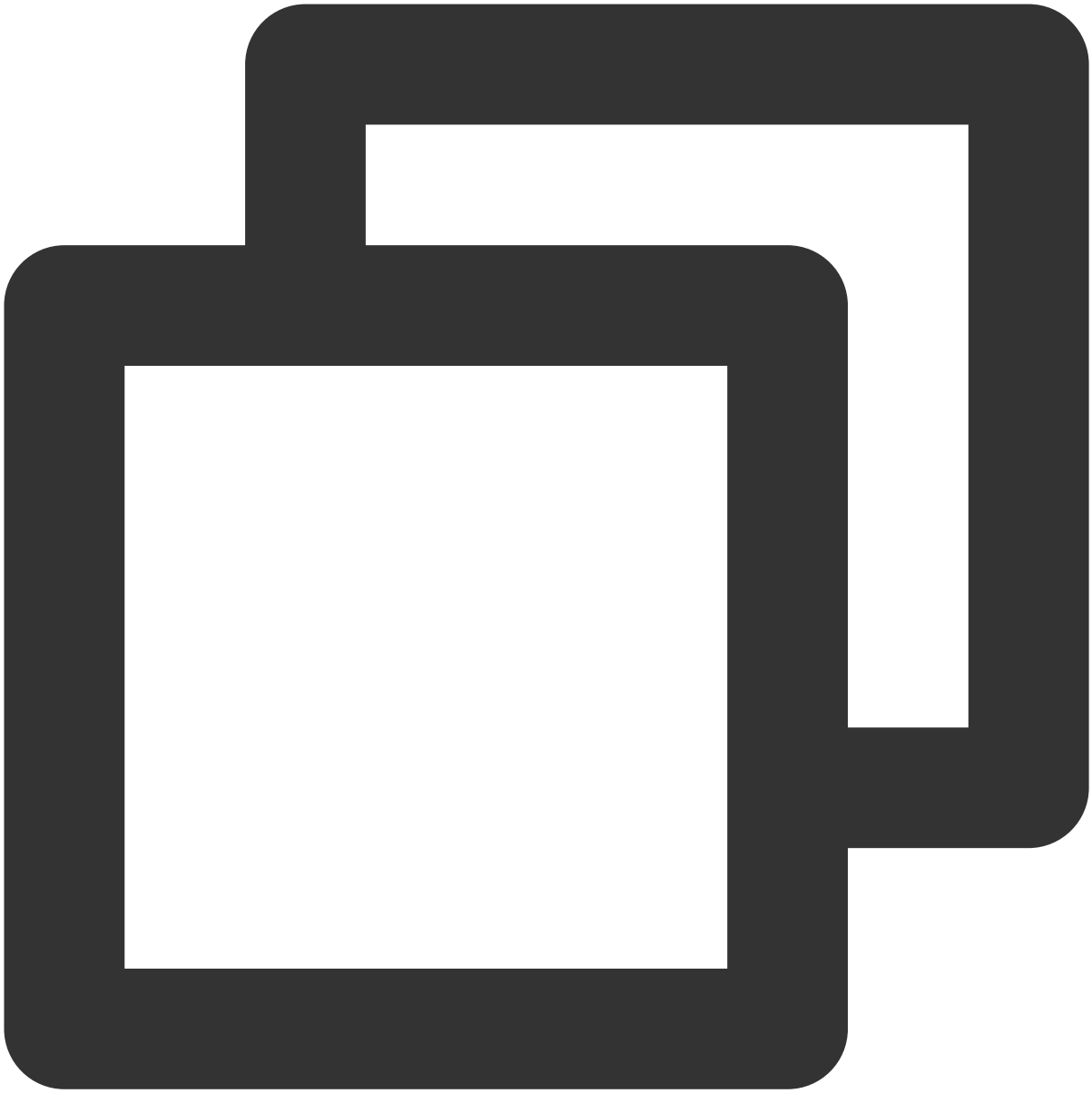


```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 1.9.7
  name: kube-state-metrics
  namespace: kube-system
spec:
  endpoints:
  - bearerTokenSecret:
```

```
      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 30
    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
      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__
      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:
```

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。

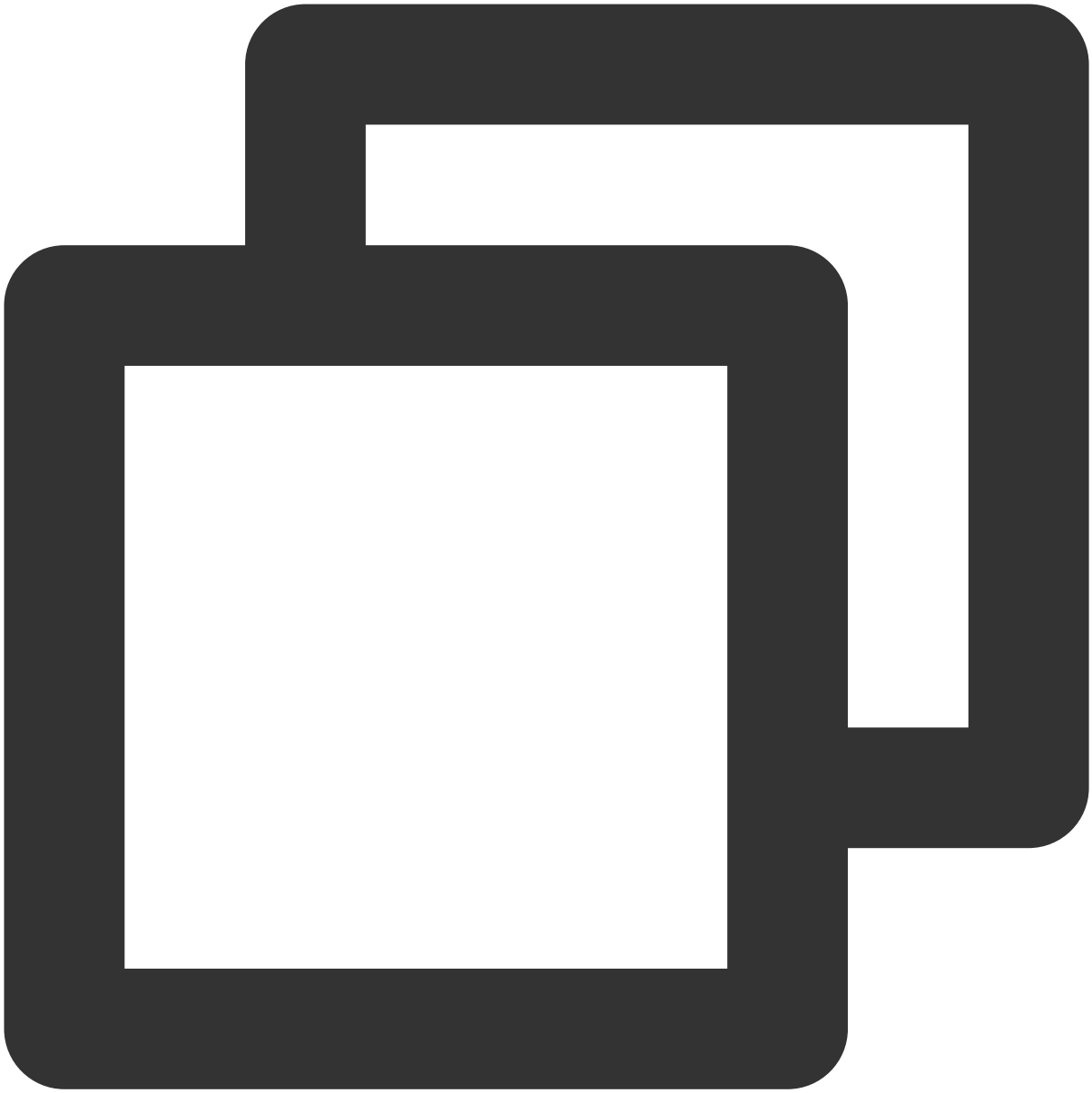
Job 示例：



```
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
    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
    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
      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_
      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

TMP 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

TMP 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 labe：`tps-skip-monitor: "true"`，关于 label 的操作请 参考。

集成中心

最近更新时间：2023-02-02 17:05:22

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，应用层监控、Kubernetes 容器监控三大监控场景，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
大数据	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
开发	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	CVM node_expoter
	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
中间件	Kafka	包括 Broker/Topic/Consumer Group 等监控	支持	Kafka Exporter 接入
	Consul	Consul 监控	支持	Consul Exporter 接入
	Etcd	Etcd 监控	不支持	-
	Istio	Istio 监控	不支持	-
基础设施	Kubernetes	包括 API Server/DNS/Workload/Network 等监控	支持	安装 Agent 接入 Kubernete
数据库	云数据库	包括文档数/读写性能/网络流量等	支持	MongoDB

	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
	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
	云数据库 Memcached	Memcached 监控	支持	Memcached Exporter 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
云监控	云监控	云产品监控	支持	-
自定义	抓取任务	使用原生 static_config 配置抓取任务	支持	抓取配置说明
	CVM 抓取任务	使用扩展的 cvm_sd_config 配置 CVM 抓取任务	支持	抓取配置说明

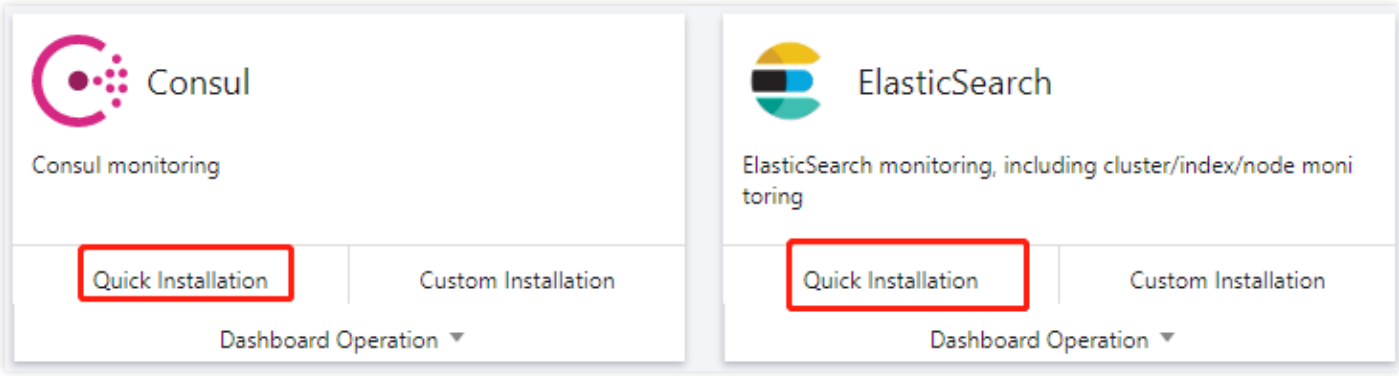
操作步骤

一键安装

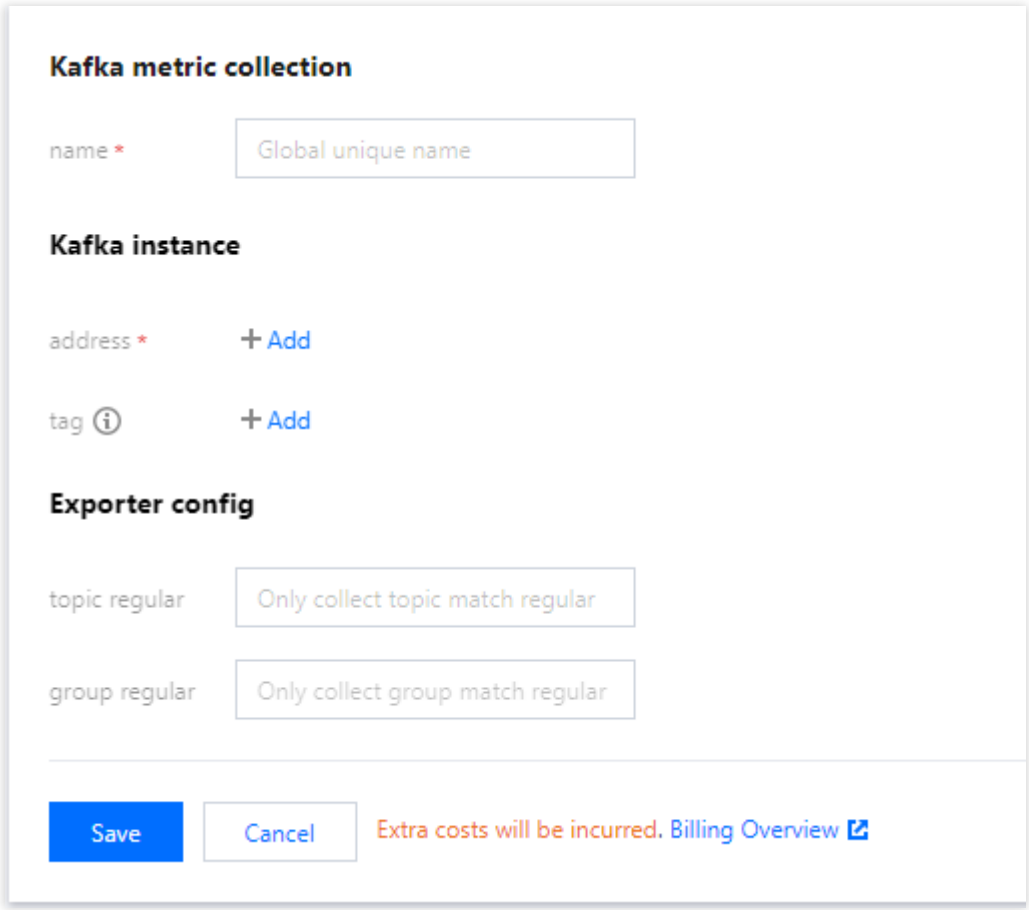
部分服务支持一键安装 Agent，详情请参见 支持服务列表。

1. 登录 云监控 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 **集成中心**。

4. 在集成中心选择支持一键安装的服务，单击模块左下角的 **安装**。

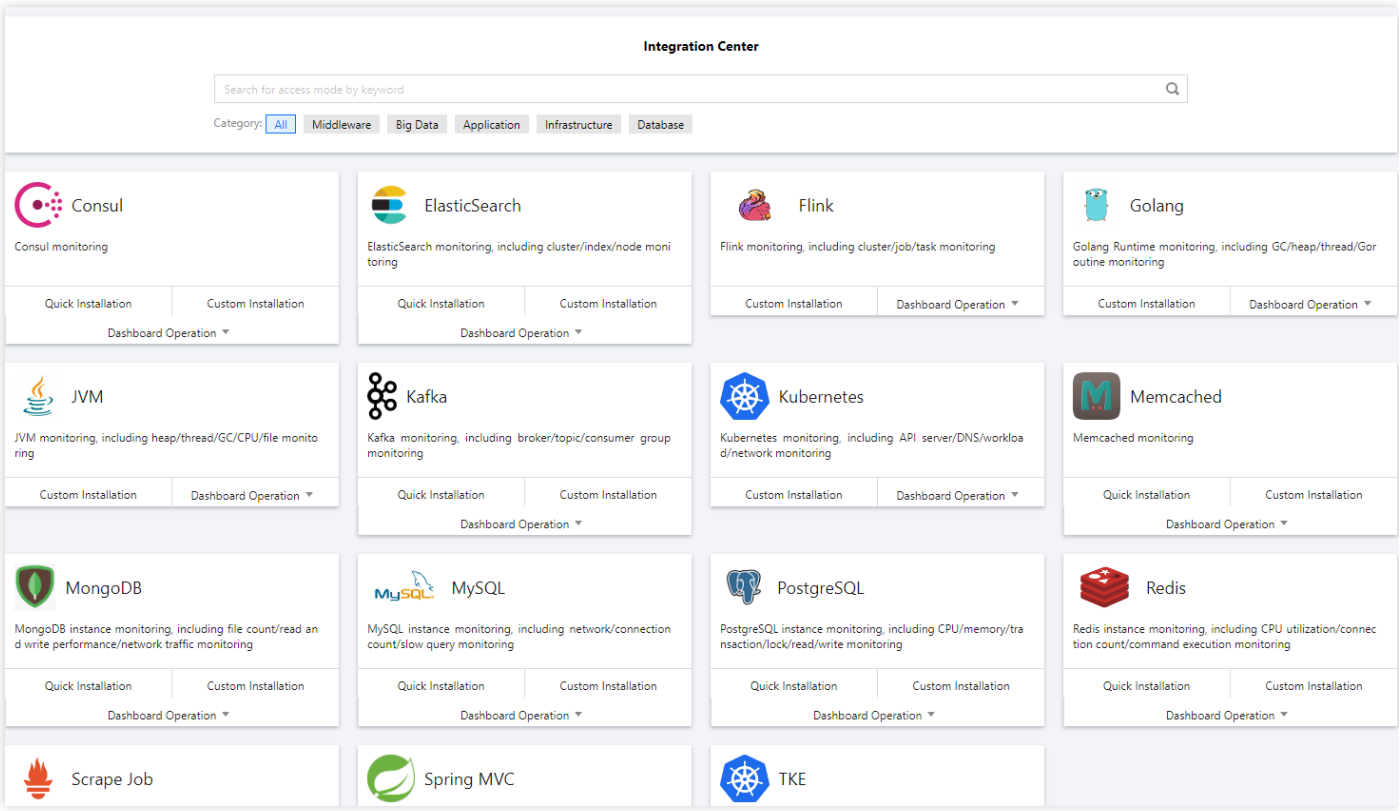


5. 在集成列表页，填写指标采集名称和地址等信息，并单击保存即可。如下图以 Kafka 为例：



自定义安装

1. 登录 云监控 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 **集成中心**。
4. 在集成中心选择对应的服务。您可以单击 **接入指南** 查看接入指引，接入成功后即可实时监控对应的服务。您还可以单击 **Dashboard 安装/升级** 安装或升级该服务的 Grafana Dashboard。


Integration Center
Search for access mode by keyword
Category: All Middleware Big Data Application Infrastructure Database
Consul
Consul monitoring
Quick Installation
Custom Installation
Dashboard Operation
ElasticSearch
ElasticSearch monitoring, including cluster/index/node monitoring
Quick Installation
Custom Installation
Dashboard Operation
Flink
Flink monitoring, including cluster/job/task monitoring
Custom Installation
Dashboard Operation
Golang
Golang Runtime monitoring, including GC/heap/thread/Goroutine monitoring
Custom Installation
Dashboard Operation
JVM
JVM monitoring, including heap/thread/GC/CPU/file monitoring
Custom Installation
Dashboard Operation
Kafka
Kafka monitoring, including broker/topic/consumer group monitoring
Quick Installation
Custom Installation
Dashboard Operation
Kubernetes
Kubernetes monitoring, including API server/DNS/workload/network monitoring
Custom Installation
Dashboard Operation
Memcached
Memcached monitoring
Quick Installation
Custom Installation
Dashboard Operation
MongoDB
MongoDB instance monitoring, including file count/read and write performance/network traffic monitoring
Quick Installation
Custom Installation
Dashboard Operation
MySQL
MySQL instance monitoring, including network/connection count/slow query monitoring
Quick Installation
Custom Installation
Dashboard Operation
PostgreSQL
PostgreSQL instance monitoring, including CPU/memory/transaction/lock/read/write monitoring
Quick Installation
Custom Installation
Dashboard Operation
Redis
Redis instance monitoring, including CPU utilization/connection count/command execution monitoring
Quick Installation
Custom Installation
Dashboard Operation
Scrape Job
Spring MVC
TKE

创建聚合规则

最近更新时间：2022-08-26 17:44:49

操作场景

本文档介绍应对复杂查询场景时如何配置聚合规则，提高查询的效率。

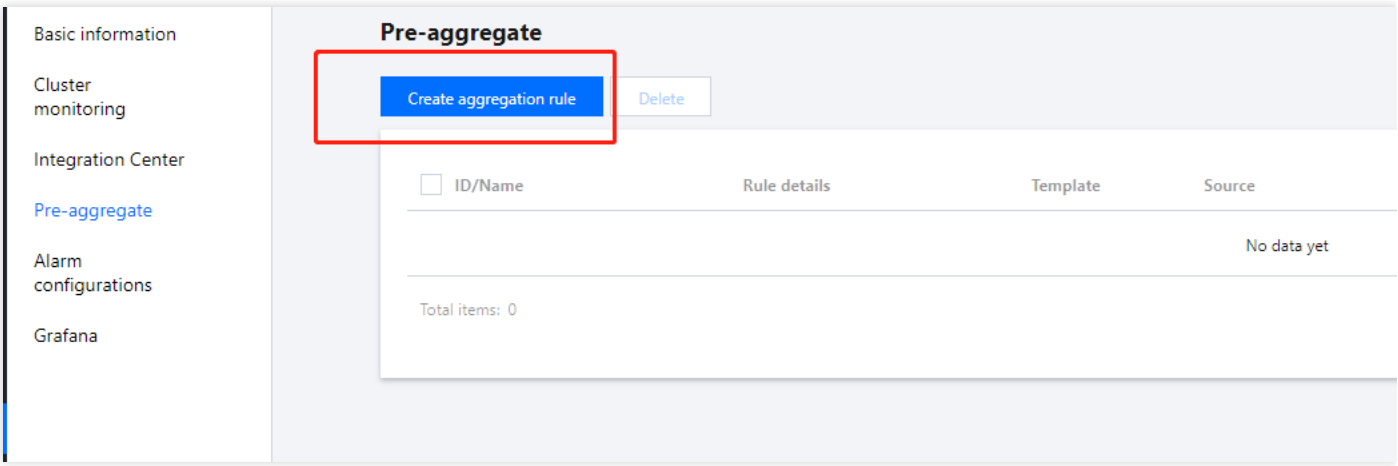
前提条件

在配置聚合规则前，您需要完成以下操作：

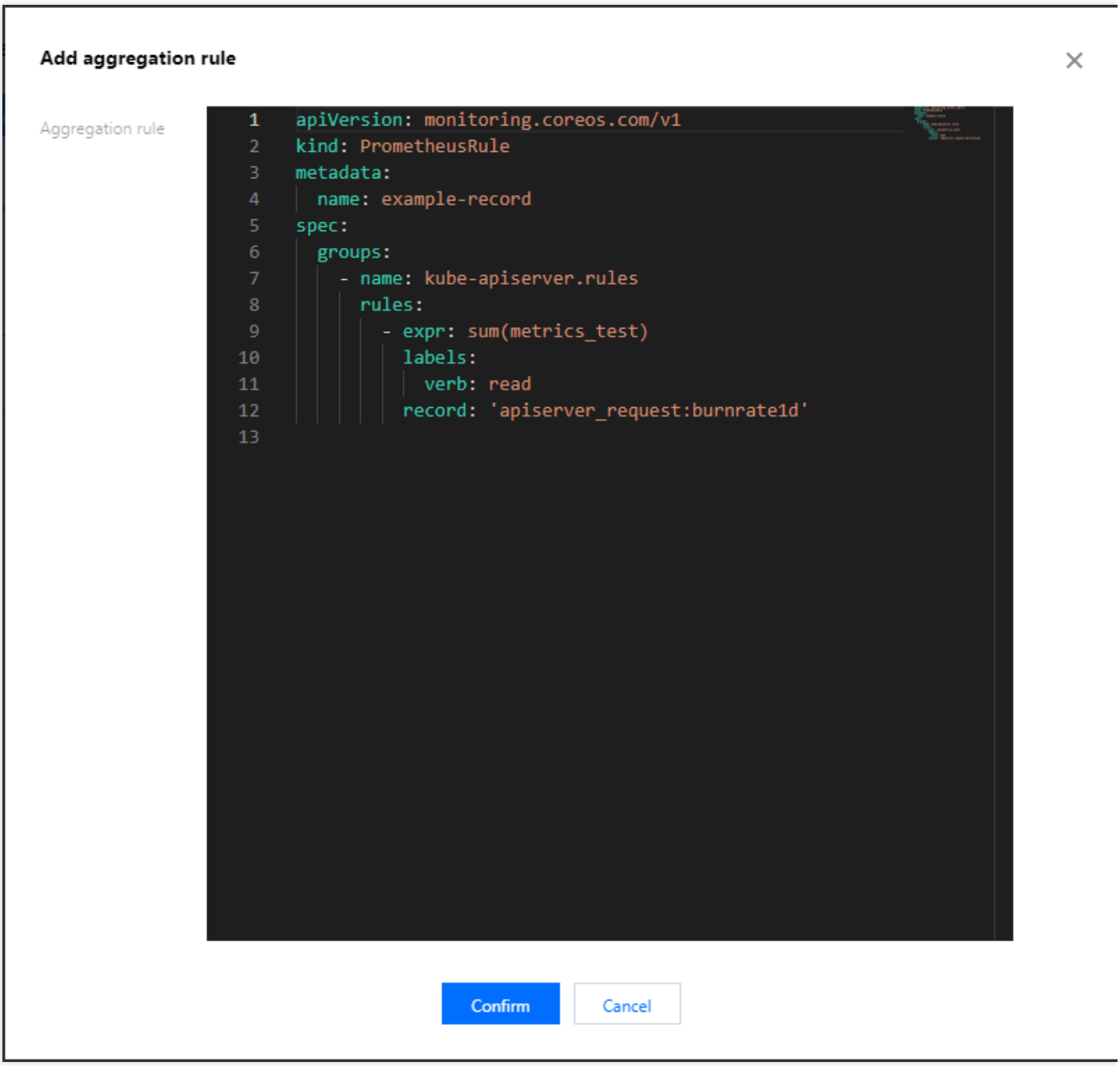
- 已登录 容器服务控制台 ，并创建独立集群。
- 已创建监控实例。

操作步骤

1. 登录 容器服务控制台 ，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要创建聚合规则的实例名称，进入该实例详情页。
3. 在“预聚合”页面，单击**新建聚合规则**。如下图所示：



4. 在弹出的“新增聚合规则”窗口，编辑聚合规则。如下图所示：



5. 单击**确定**，即可完成创建聚合规则。

告警配置

最近更新时间：2022-07-13 15:36:51

操作场景

本文档介绍如何在云原生监控服务中配置告警规则。

前提条件

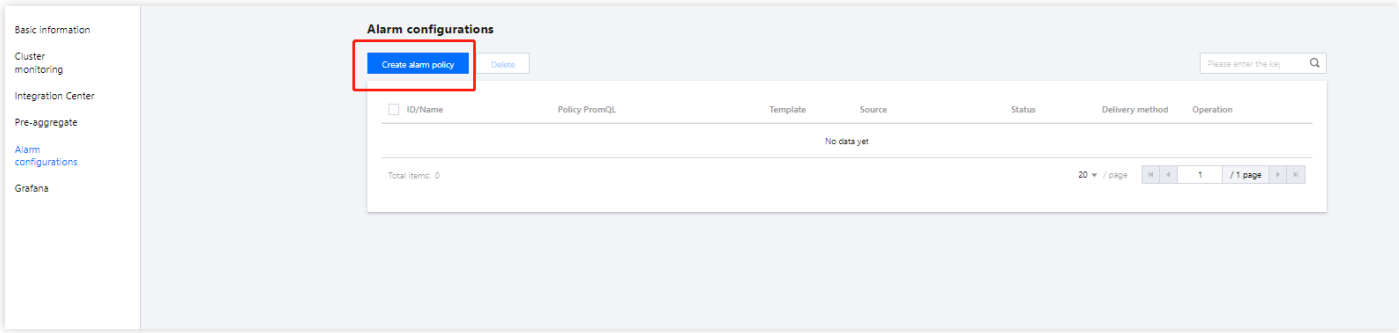
在配置告警前，您需要完成以下准备工作：

- 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。
- 已将需要采集的信息添加到集群数据采集配置。

操作步骤

配置告警规则

1. 登录 容器服务控制台 ，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要配置告警规则的实例名称，进入该实例详情页。
3. 在“告警配置”页面，单击**新建告警策略**。如下图所示：



4. 在“新建告警策略”页面，添加策略详细信息。

- **名称**：告警策略名称
- **策略模板**：选择策略模板。Prometheus 监控服务预制了很多监控模板，客户可按需选择。
- **规则**：
  - **规则名称**：告警规则的名称，不超过63个字符。

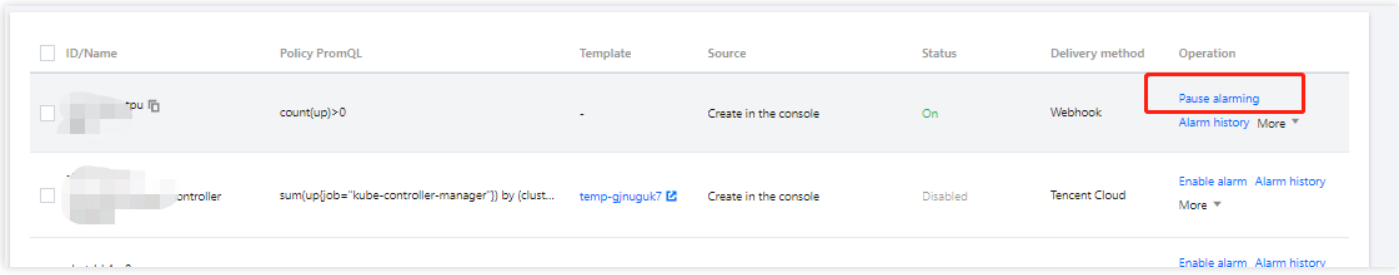
  - **规则描述**：告警规则的描述。
  - **PromQL**：告警规则语句。您可使用默认模板或自定义，表示基于 PromQL 的表达式告警触发条件，用于计算是否有时间序列满足该条件。
  - **Label**：对应每条规则添加 Prometheus 标签。
  - **Annotation**：表示允许用户定义告警附加消息。
  - **告警内容**：告警触发后通过邮件或短信等渠道发送告警通知的具体内容。
  - **持续时间**：满足上述语句所描述的条件的时间，达到该持续时间则会触发告警。
- **收敛时间**：在该周期内，若多次满足告警条件，仅会发送一次通知。
- **告警渠道**：告警后发送告警内容的渠道。
- **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**：模板默认命名为填写的告警策略名称，保存后可在主页的模板设置里对模板名称和模板内容进行编辑。

5. 单击**完成**，即可完成新建告警策略。

> 注意：
> 新建告警策略后，默认告警策略生效。

暂停告警

1. 登录 容器服务控制台 ，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要暂停告警的实例名称，进入该实例详情页。
3. 在“告警配置”页面，单击实例右侧的**暂停告警**。如下图所示：



4. 在弹出的“关闭告警设置”窗口单击**确定**，即可暂停告警策略。

告警历史

最近更新时间：2022-08-26 17:44:49

操作场景

本文档介绍如何在云原生监控功能服务中查看告警历史。

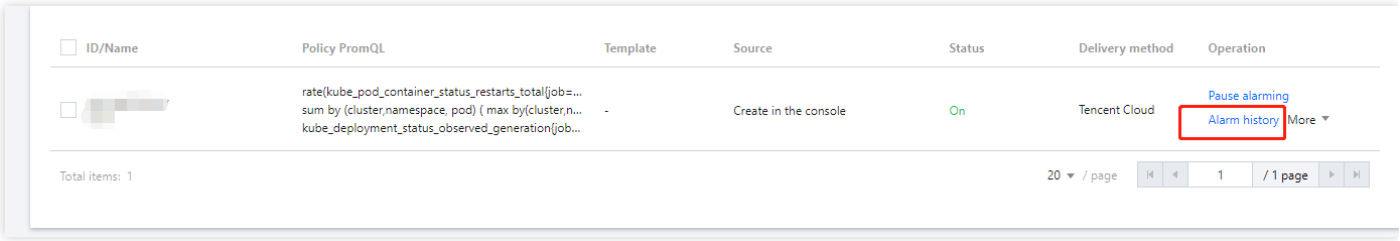
前提条件

在查看告警历史前，需要完成以下前置操作：

- 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。
- 已将需要采集的信息添加到集群数据采集配置。
- 已配置告警规则。

操作步骤

1. 登录 容器服务控制台 ，选择左侧导航栏中的 **Prometheus 监控**。
2. 在监控实例列表页，选择需要查看告警历史的实例名称，进入该实例详情页。
3. 在“告警配置”页面，选择“告警历史”。如下图所示：



计费方式和资源使用

最近更新时间：2022-06-10 19:32:52

目前使用 Prometheus 监控服务（TMP）时将会在用户的账户下创建 EKS 集群、内外网 负载均衡 CLB 资源，以及 Prometheus 服务本身的费用，按用户实际使用的云资源收费。本文向您介绍使用 Prometheus 监控服务时资源的使用情况。

资源列表

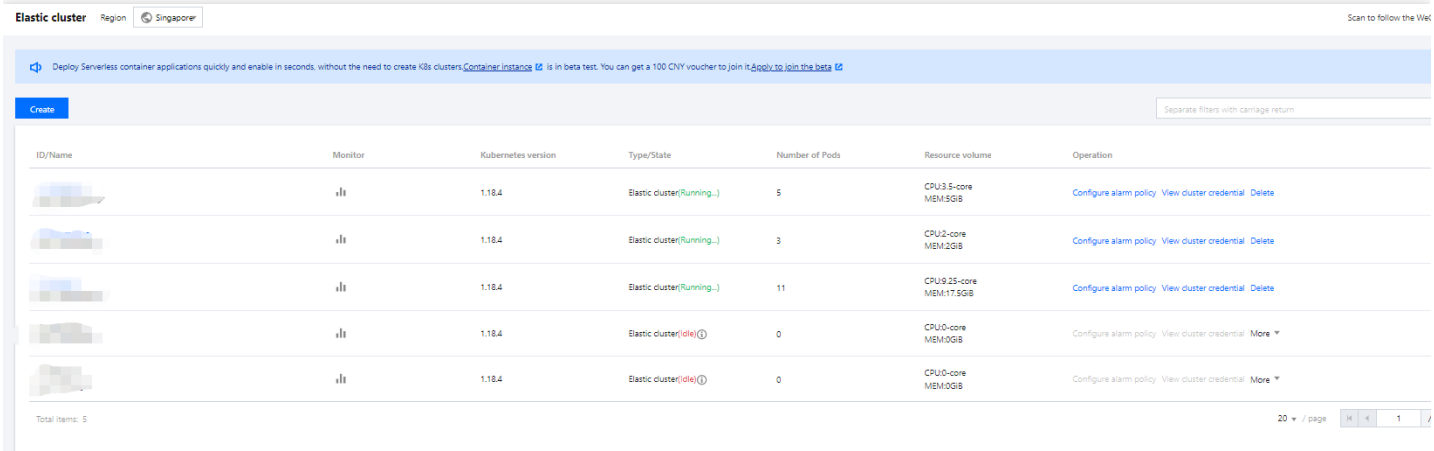
TMP 实例

TMP 上线了“收费指标采集速率”的能力，您可以用该数值估算监控实例/集群/采集对象/指标等多个维度的预估费用：

1. 登录 容器服务控制台 ，选择左侧导航栏中的 **Prometheus 监控**。
2. 在 Prometheus 监控列表中，查看“收费指标采集速率”。该指标表示 TMP 实例的收费指标采集速率，根据用户的指标上报量和采集频率预估算出。该数值乘以 86400 则为一天的监控数据点数，根据 按量计费 可以计算预估的监控数据刊例价。
   您也可以在“关联集群”、“数据采集配置”、“指标详情”等多个页面查看到不同维度下的收费指标采集速率。

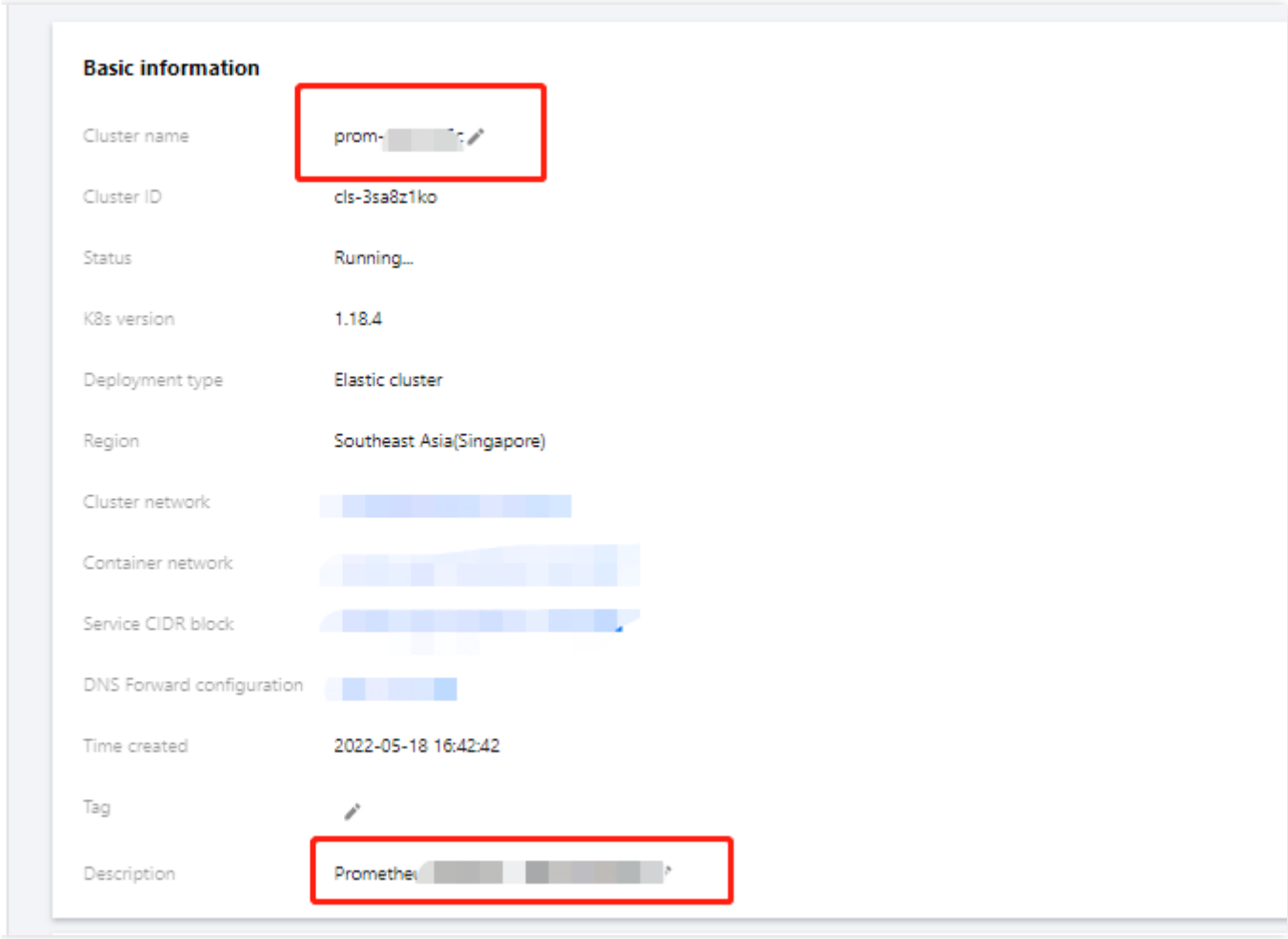
EKS 集群

每创建一个 Prometheus 监控实例后，会在用户的账户下创建一个按量付费 EKS 集群，用于数据采集。在 弹性集群列表页 查看资源信息，如下图所示：



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

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考：

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
<50万	1.25核 1.6GiB	0.35美元
100万	0.5核1.5GiB*2	1.46美元
500万	1核3GiB*3	2.93美元
2000万	1核6GiB*5	7.98美元
3000万	1核6GiB*8	12.77美元

EKS 集群成本示例如下：

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：

0.0319 x 24 + 0.0132 x 24 = 1.0824美 元

负载均衡 CLB

使用 Prometheus 服务监控关联集群监控容器服务，常规情况下会在用户账户下创建一个内网 CLB 用于打通采集器与用户集群的网络。

若用户关联了边缘集群，或跨集群关联了未打通网络的集群，支持创建公网的 CLB 进行网络联通，此时会创建一个公网 CLB。

若要使用通过外网访问 Grafana 服务，则需要创建一个相应的公网 CLB。

这些 CLB 资源会收取费用，创建的公网 LB 可在 负载均衡控制台 查看资源信息。

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

目前不支持用户直接在对应控制台删除资源，例如需要在 Prometheus 监控销毁监控实例，对应的所有资源会一并销毁。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。

按量付费免费指标

最近更新时间：2022-05-16 15:39:27

下列指标在按量计费模式下不计费。

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

所属配置文件	指标名
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
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

所属配置文件	指标名
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
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

所属配置文件	指标名
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
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

所属配置文件	指标名
kube-state-metrics	kube_statefulset_status_replicas
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-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

所属配置文件	指标名
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

销毁监控实例

最近更新时间：2023-05-19 15:09:15

操作场景

当您不需要再使用 Prometheus 监控服务监控集群时，可以通过 Prometheus 监控控制台删除所有监控实例，系统会自动卸载监控组件并销毁相关资源。

操作步骤

1. 登录 **容器服务控制台**，选择左侧导航中的 **Prometheus 监控**。
2. 在**监控实例列表**页中找到需要删除的实例，单击实例名称右侧的**销毁/退还**。
3. 在“销毁/退还”弹窗中确认监控实例信息后，单击**确定**。

> 说明：
>
> - 实例删除后，Prometheus 监控控制台不再展示该实例信息。
> - 实例删除后，实例内已有的监控功能组件等资源及配置均将被删除，实例关联的集群将自动解除关联不再被监控，实例关联的 TKE Serverless 集群将随实例一并删除。
> - 删除操作不可逆，以上实例数据将无法恢复，请谨慎操作。