云监控

Prometheus 服务

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2021 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

Prometheus 服务

服务简介

最近更新时间：2021-06-04 17:14:44

云监控 Prometheus 服务在继承开源 Prometheus 监控能力的同时，还提供高可用的 Prometheus 托管服务及与开源可视化的 Grafana，为您减少用户的开发及运维成本。

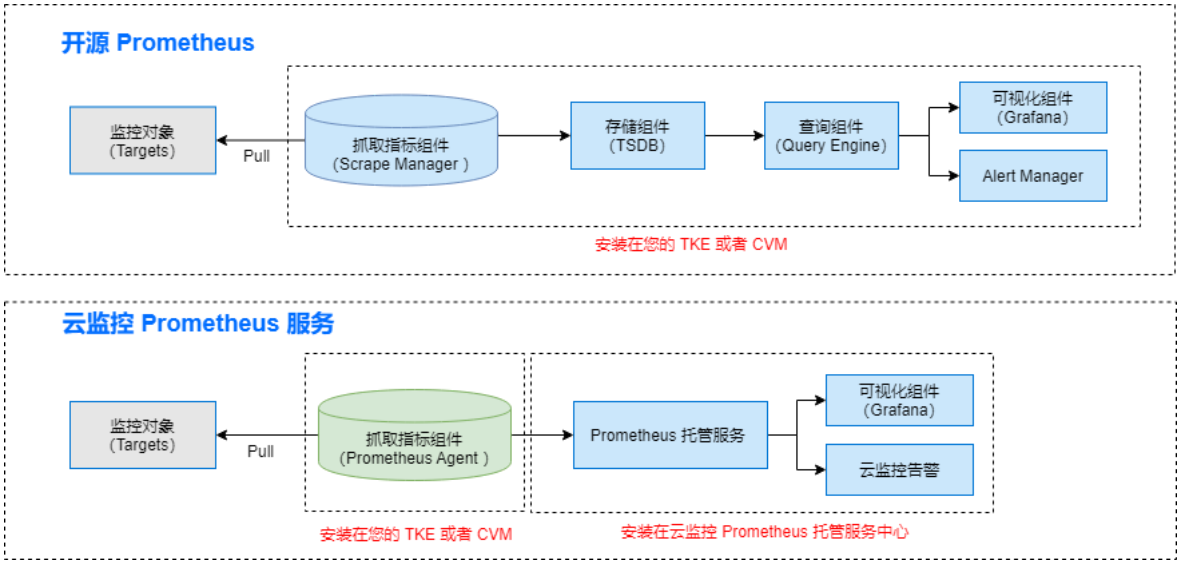
Prometheus 简介

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 可方便的支持多种可视化图表及仪表盘。

云监控 Prometheus 服务优势



与开源的 Prometheus 对比，云监控 Prometheus 服务有哪些优势？

- 更轻量、更稳定、可用性更高。
- 完全兼容开源 Prometheus 生态。
- 无需您手动搭建，节省开发成本。
- 与腾讯云容器服务高度集成，节省与 Kubernetes 集成的开发成本。
- 与云监控告警体系打通，节省开发告警通知成本。
- 同时也集成了常用的 Grafana Dashboard 及告警规则模板。

更轻量、更稳定、可用性更高

- 与开源 Prometheus 监控相比，云监控 Prometheus 托管服务的整体结构更加轻量化，您在云监控创建 Prometheus 实例后，即可拥有 Prometheus 服务。
- 在系统稳定性方面，开源 Prometheus 一般会占用几十 GB 的内存，云监控 Prometheus 只会占用 MB 级别的用户资源，为您提供了更低资源占用的 Prometheus 服务。
- Prometheus 托管服务结合腾讯云云存储服务及自身的副本能力，为您提供了可用性更强的 Prometheus 监控服务，减少系统中断运行次数。

版权所有：腾讯云计算（北京）有限责任公司

节省开发运维成本

- 云监控提供了原生的 Prometheus 一站式服务，在您购买 Prometheus 实例之后，可以快速与腾讯云容器服务 TKE 集成，Prometheus 为运行在 Kubernetes 之上的服务提供监控服务，免去用户搭建运维及开发成本。
- 配合 Prometheus，云监控提供了开通即可使用的 Grafana 服务，同时也集成了丰富的 Kubernetes 基础监控的 Dashboard，以及常用服务监控的 Dashboard，用户开通后即可快速使用，免去自己维护 Dashboard 的成本。
- 基于云监控告警通道的能力，打通 Prometheus Alertmanager，同时提供丰富的报警规则模板，免去用户学习告警配置的成本。

快速开始

版权所有：腾讯云计算（北京）有限责任公司

最近更新时间：2021-04-02 09:14:15

功能介绍

云监控 Prometheus 服务在继承开源 Prometheus 监控能力的同时，还提供高可用的 Prometheus 托管服务及与开源可视化的 Grafana。为您减少用户的开发及运维成本。

对于已创建腾讯云 容器服务 TKE 的用户，您可以在云监控创建 Prometheus 实例并安装 Prometheus 监控插件对其进行监控，同时云监控 Prometheus 服务集成 Grafana 及预定义 Dashboard 来查看不同维度的性能指标数据。

前提条件

创建腾讯云容器服务 托管版集群。

操作步骤

创建 Prometheus 实例

1. 登录 云监控 Prometheus 控制台。
2. 单击【新建】，进入新建购买页，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。

集成容器服务 TKE

云监控 Prometheus 托管服务已经深度集成了腾讯云容器服务 TKE，用户只需要一键安装就可以对 Kubernetes 集群及运行在上面的服务进行监控。

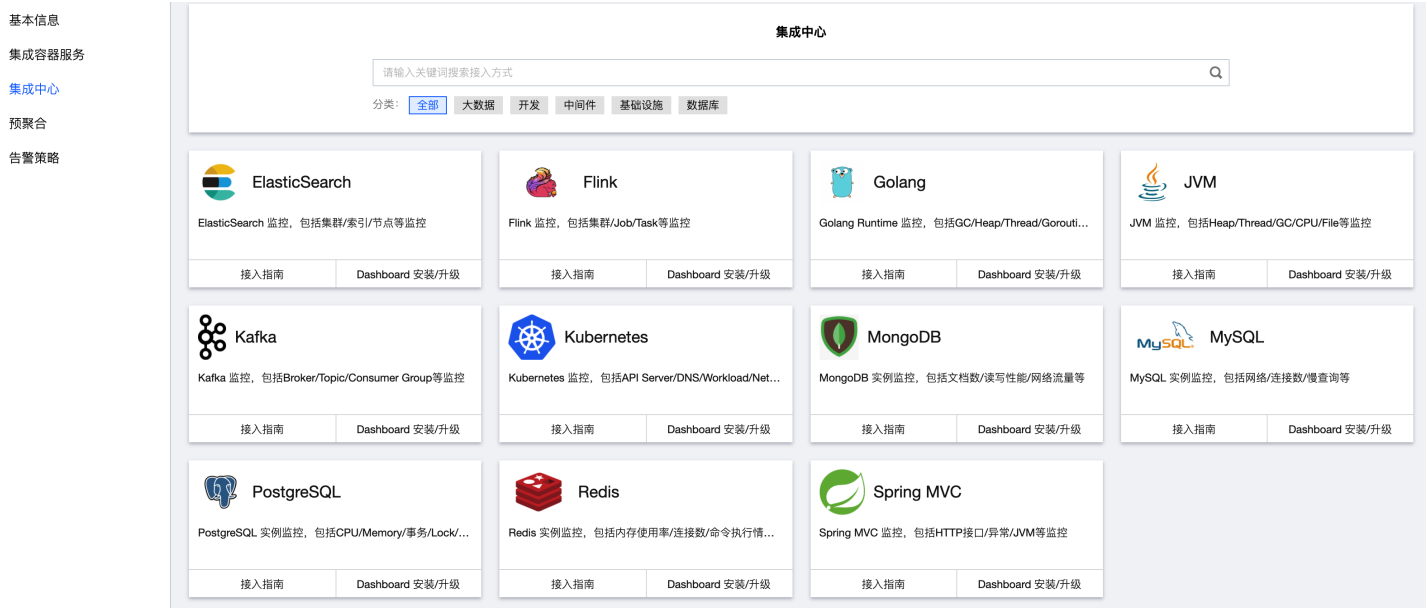
1. 在实例列表中，选择需要集成的 Prometheus 实例，单击【实例 ID】或者右侧的【管理】，进入 Prometheus 实例管理页。
2. 单击【集成容器服务】> 选择对应的容器集群 > 单击【安装】来进行自动化集成，在安装弹框中可以选择需要集成的基础监控组件。

3. 整体集成操作为异步操作，大概需要2 – 3分钟，监控状态显示“已安装”即安装成功。

> 说明：
> 集成过程中需要用户授权之后来操作腾讯云容器服务 TKE，具体的授权操作请参见 服务授权相关角色权限说明。

集成中心

为了方便用户接入，云监控 Prometheus 托管服务对常用的 开发语言 / 中间件 / 大数据 进行了集成，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。

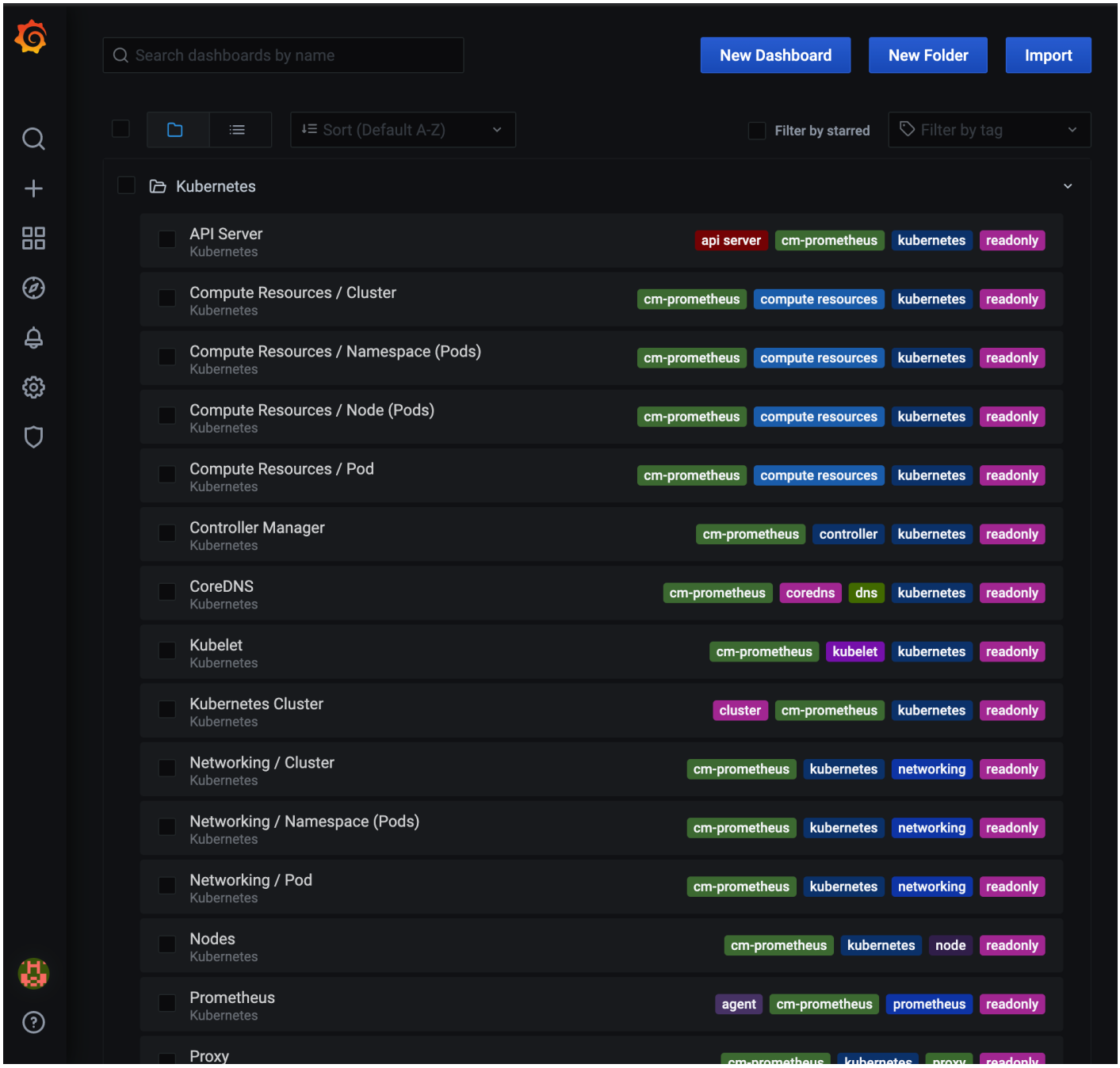


Grafana 查看监控数据

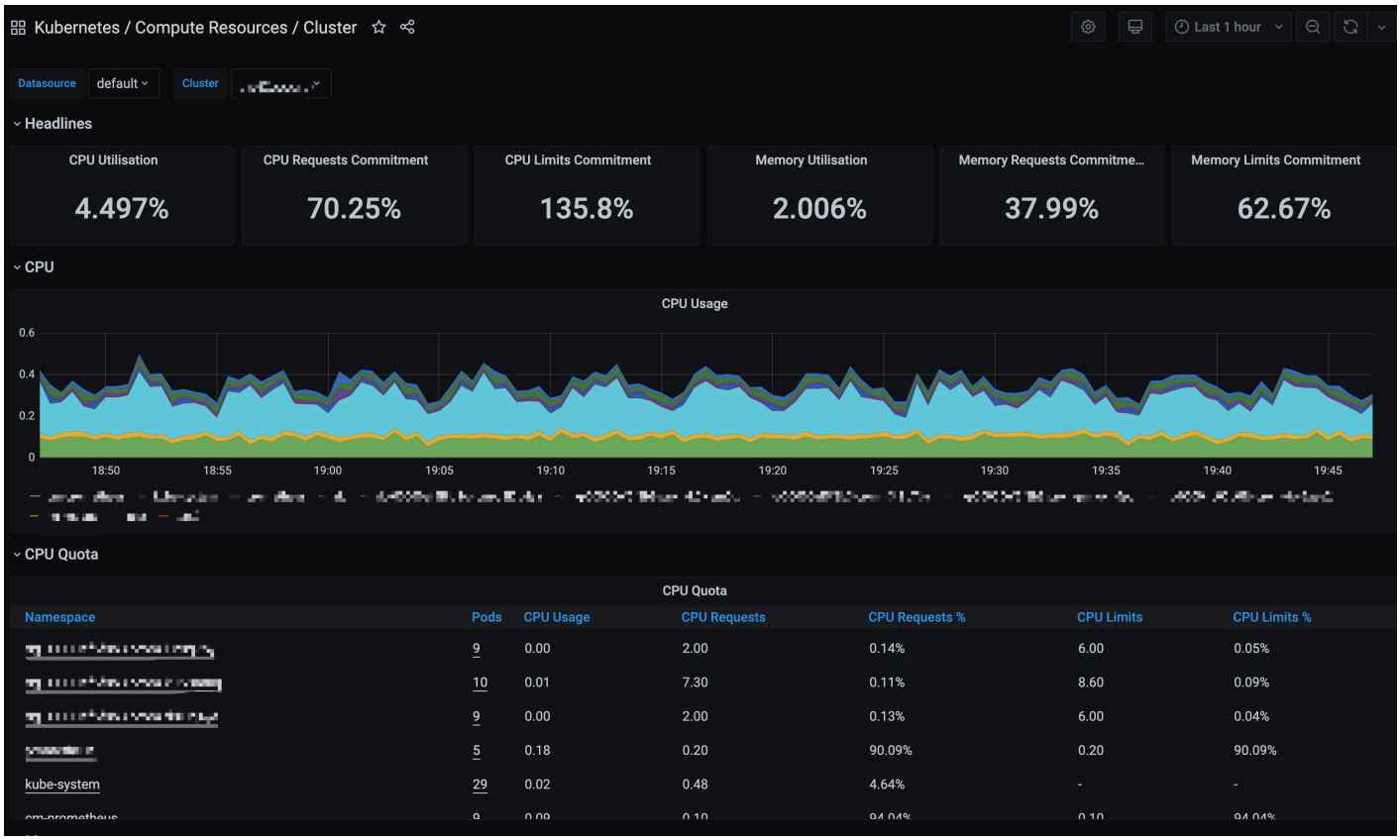
配合 Prometheus，云监控提供了开箱即用的 Grafana 服务供用户使用，同时也集成了丰富的 Kubernetes 基础监控的 Dashboard，以及常用服务监控的 Dashboard，用户可以开箱即用。

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，输入账号密码，即可进行 Grafana 可视化大屏操作区。

2. 进入 Grafana，单击【 】图表，展开监控面板。



3. 单击对应的监控图表名称即可查看监控数据。



> **说明：**
> 如需了解 Grafana 更多操作说明，请参见 Grafana 官网使用手册。

操作指引
实例
创建实例

最近更新时间：2021-02-09 16:56:46

本文以自定义配置方式为例，指导您如何创建腾讯云云监控 Prometheus 托管实例。

准备工作

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 云监控 Prometheus 控制台。
2. 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	目前仅支持包年包月计费模式。
可用区域	必选	根据您云产品所在区域选择。每个地区价格可能会不一样，以购买页实时的价格为主。处于不同地域的云产品内网不通，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。
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的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，可以参考文档 新建私有网络 和 新建子网 进行创建。
产品规格	必选	不同的产品规格价格有差异，请参考 产品定价。
数据存储时间	必选	不同的存储时长价格有差异，请参考 产品定价。
实例名称	必选	用户自定义 prometheus 实例名称。
Grafana/Grafana 密码	必填	默认开启 Grafana，实例创建成功后系统会生成一个公网可访问的域名。该 Grafana 的默认账号为：admin，登录密码由您自定义。
购买时长	必选	提供多种时长购买

3. 配置完后，单击【立即购买】即可。

产品规格 *

	型号	峰值TPS（条/秒）	Series上限（条数）
●	基础1	15千	45万
○	基础2	30千	90万
○	集群1	25千	75万
○	集群2	30千	90万
○	集群3	45千	135万

不同规格的差异、适用场景，可参考 购买指南

数据存储时间 * 15天 | 30天 | 45天

实例名称 * 默认项目

Grafana ☑ 开启 （当前版本仅支持默认开启Grafana，实例创建成功之后开通可访问的公网域名地址，用户名为admin）

密码必须以数字或字母开头，至少包含一个@$!%*#?&特殊字符，长度在[6,24]之间

Grafana密码 * ••••••••

Grafana确认密码 * ••••••••

标签（选填） monitor | aaa ×

+添加

如现有标签/标签值不符合您的要求，可以去控制台 新建

购买时长 * 1月 | 2月 | 3月 | 4月 | 5月 | 6月 | 12月 | 更多

自动续费 ☑ 账户余额足够时，设备到期后按月自动续费

费用 配置费用 元 原价1 元 | 流量费用 元

立即购买

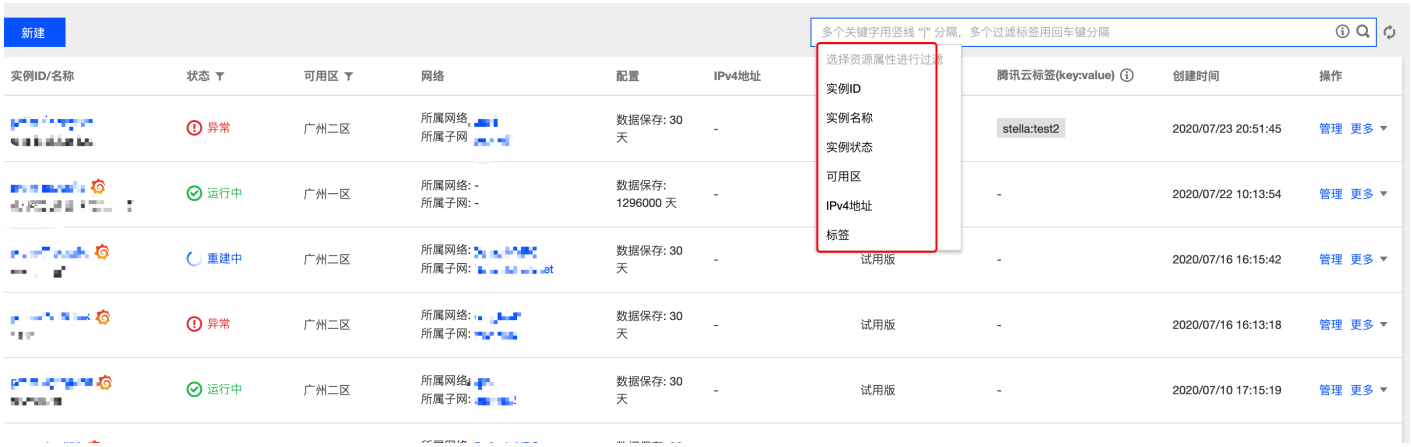
搜索实例

最近更新时间：2020-12-04 15:57:38

默认情况下，云监控 Prometheus 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 云监控 Prometheus 控制台。
2. 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。



3. 支持用户根据不同的维度来过滤实例列表，目前支持如果几种维度：
   - 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。
   - 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：支持多个IPv4过滤，每个IPv4仅支持完全匹配过滤。
   - 标签：支持多个标签组合来过滤实例，同时也支持直接点击实例列表中对应的标签值来进行过滤。

版权所有：腾讯云计算（北京）有限责任公司

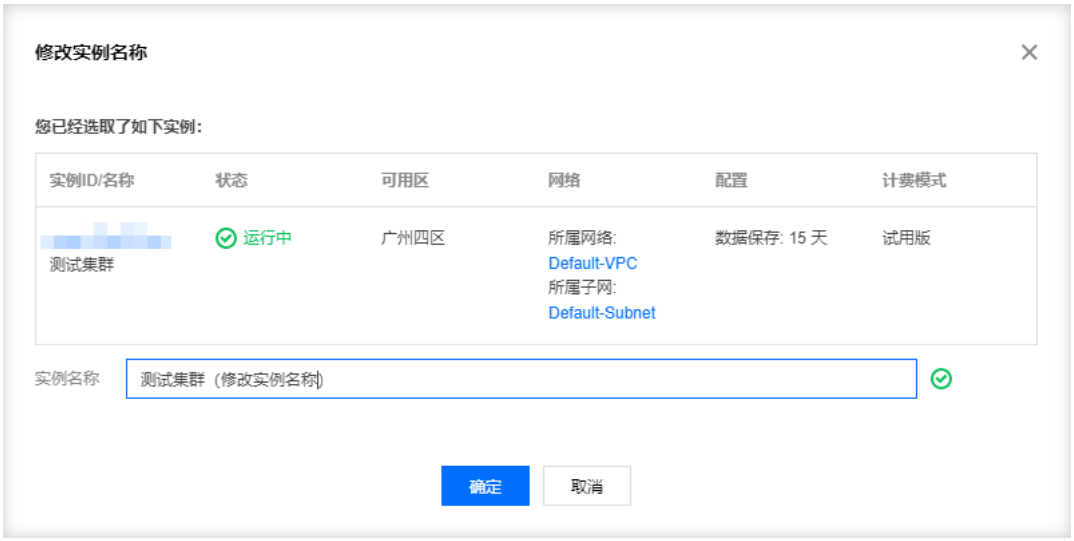
修改实例名称

最近更新时间：2020-12-04 15:57:42

为了方便用户在云监控 Prometheus 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 云监控 Prometheus 控制台。
2. 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】。
3. 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：2021-02-04 18:09:22

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中为准。

操作步骤

1. 登录 云监控 Prometheus 控制台。
2. 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】。
3. 由于销毁操作属于高危操作，在弹出的“销毁”窗口中，完成销毁操作步骤，单击【确定】即可。

重建实例

最近更新时间：2020-12-04 15:57:50

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 云监控 Prometheus 控制台。
2. 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】。
3. 在弹出的“重建”窗口中，单击【确定】即可。

说明：
如果当前实例状态为运行中，对其进行重建操作，会导致在重建过程中服务中断的情况，请您确认操作的风险。

查看实例基本信息

最近更新时间：2020-12-04 15:57:54

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 云监控 Prometheus 控制台。
2. 在实例列表中，选择需要查看的 Prometheus 实例，单击【实例ID】或者右侧的【管理】。

3. 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。
- 修改 Grafana Admin 密码。
- 升级 Grafana 预设 Dashboard。

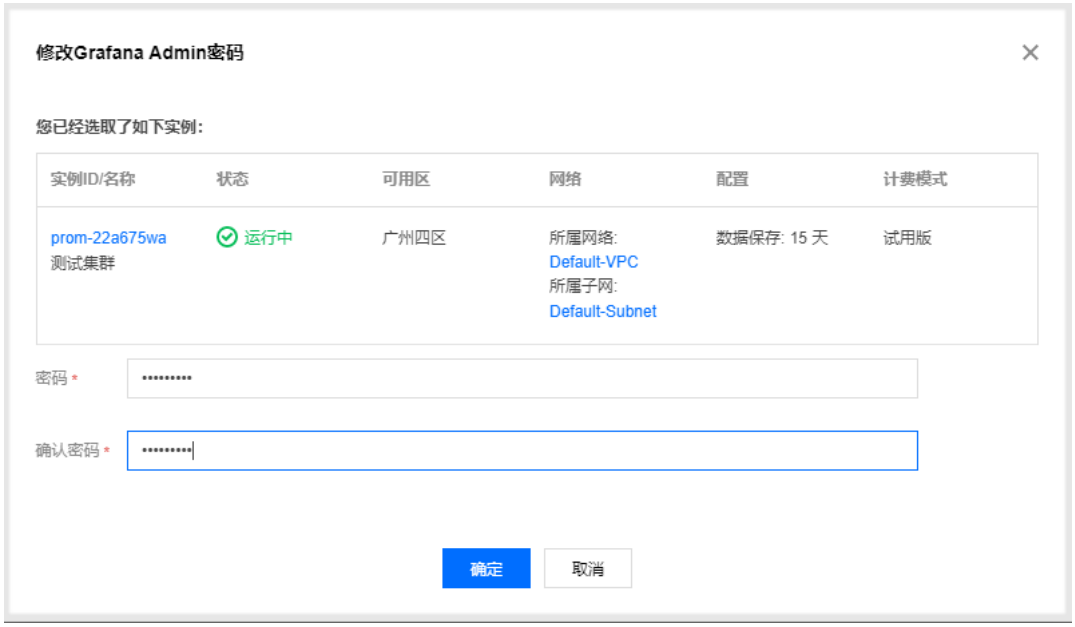
重置 Grafana 密码

最近更新时间：2020-12-04 15:57:59

如果您遗忘了 Grafana Admin 密码，您可以在控制台上重新设置实例对应 Grafana Admin 的登录密码。

操作步骤

1. 登录 云监控 Prometheus 控制台。
2. 在实例列表中，选择需要被修改的 Prometheus 实例，单击右侧的【更多】>【Grafana】>【修改密码】。
3. 在弹出的“修改密码”窗口中，输入新的 Grafana Admin 密码，单击【确定】即可。



升级 Grafana Dashboard

最近更新时间：2020-12-04 15:58:04

为了方便用户在云监控 Prometheus 监控服务，会不断优化或新增预设的 Grafana Dashboard，用户可以更新预设 Grafana Dashboard。

操作步骤

1. 登录 云监控 Prometheus 控制台。
2. 在实例列表中，选择需要被升级的 Prometheus 实例，单击右侧的【更多】>【Grafana】>【升级Dashboard】。
3. 在弹出的“升级 Dashboard”窗口中，单击【确定】即可。

说明：
此过程只会对 Prometheus 托管服务提供的预设 Grafana Dashboard 进行升级，不会影响用户自己添加的 Dashboard，整个升级过程大约需要1 – 3分钟，期间可能会存在对应 Dashboard 无法访问的情况。

集成容器服务

概述

最近更新时间：2020-10-14 11:28:05

腾讯云容器服务（Tencent Kubernetes Engine，TKE）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。作为容器领域事实的标准，Kubernetes 可以极大的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。

主要内容

- Prometheus 在 Kubernetes 下的服务发现机制。
- 监控 Kubernetes 集群状态。
- 监控集群基础设施。
- 监控集群应用容器资源使用情况。
- 监控用户部署的应用程序。

基本概念

述语	说明
Prometheus Operator	CoreOS 为简化 Kubernetes 操作，引入了 Operator 的概念，针对 Prometheus 推出了 Prometheus Operator，提供 Prometheus 相关自定义 Kubernetes 资源，方便操作 Prometheus
ServiceMonitor	声明式的管理 Service 相关监控配置
PodMonitor	声明式的管理 Pod 相关监控配置
服务发现	Prometheus 提供了多种抓取任务的服务发现机制，例如基于文件、Consul 等

操作步骤

1. 登录 云监控 Prometheus 控制台。
2. 在实例列表中，选择需要操作的 Prometheus 实例，单击【实例 ID】或者右侧的【管理】进入实例管理页面。
3. 在管理页面的左侧菜单【集成容器服务】进入容器集成页面，目前主要提供如下服务。
   - 安装 Agent。
   - 集成 Kubernetes 相关的 基础监控。
   - 抓取任务状态 及 Agent 状态。

版权所有：腾讯云计算（北京）有限责任公司

Agent 管理

最近更新时间：2021-04-02 09:16:26

本身 Prometheus 是 Cloud Native Computing Foundation（CNCF）下毕业的第2个项目，所以与 Kubernetes 有着深度的集成，但是还是有一定的学习成本，云监控 Prometheus 托管服务为了降低用户的学习及使用成本，与腾讯云上的容器服务 TKE 做了深度的集成，可一键安装 Agent 的同时提供原生的 Prometheus 服务。

准备工作

- 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。
- 服务角色授权：集成过程中需要用户授权之后来操作腾讯云容器服务（TKE），具体的授权操作请参见 服务授权相关角色权限说明。
- 已经进入到集成容器服务管理页面。

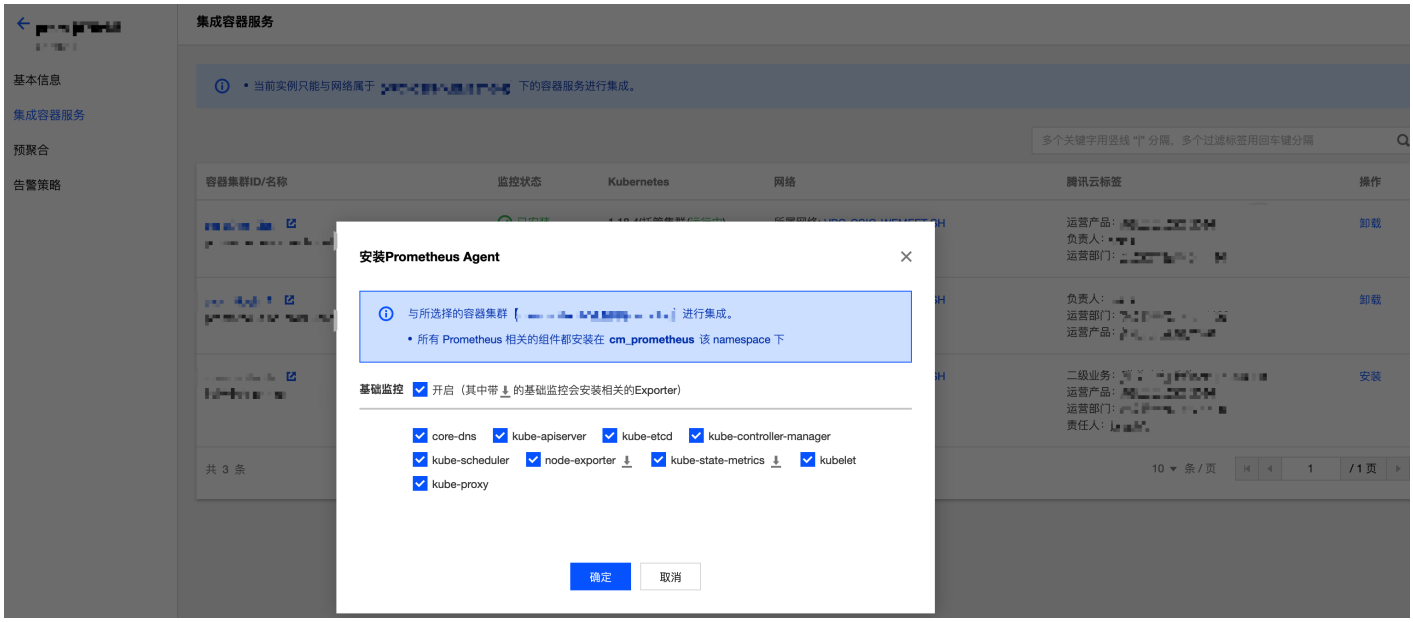
操作步骤

安装 Agent

1. 用户成功授权服务角色之后，可以列出当前地域与 Prometheus 服务相同私有网络 VPC 下的容器服务 TKE 集群列表信息。

> 说明：
> 由于网络的原因，不在同一私有网络 VPC 下的容器服务 TKE 集群不会显示在列表中。

2. 在集群列表中选择对应的容器集群 > 单击【安装】来进行自动化集成，在安装弹框中可以选择需要集成的 基础监控 组件，整个集成安装操作为异步操作，大概需要2 – 3分钟左右，监控状态显示“已安装”即装成功。



卸载监控组件

如需停止对容器服务的 Kubernetes 集群监控，请按照以下步骤卸载 Prometheus Agent 及其监控组件。

集群列表中选择对应的容器集群 > 单击【卸载】来进行自动化卸载操作，在弹框中确认卸载即可，整个卸载操作为异步操作，大概需要2 – 3分钟左右。

服务发现

最近更新时间：2020-09-28 19:49:28

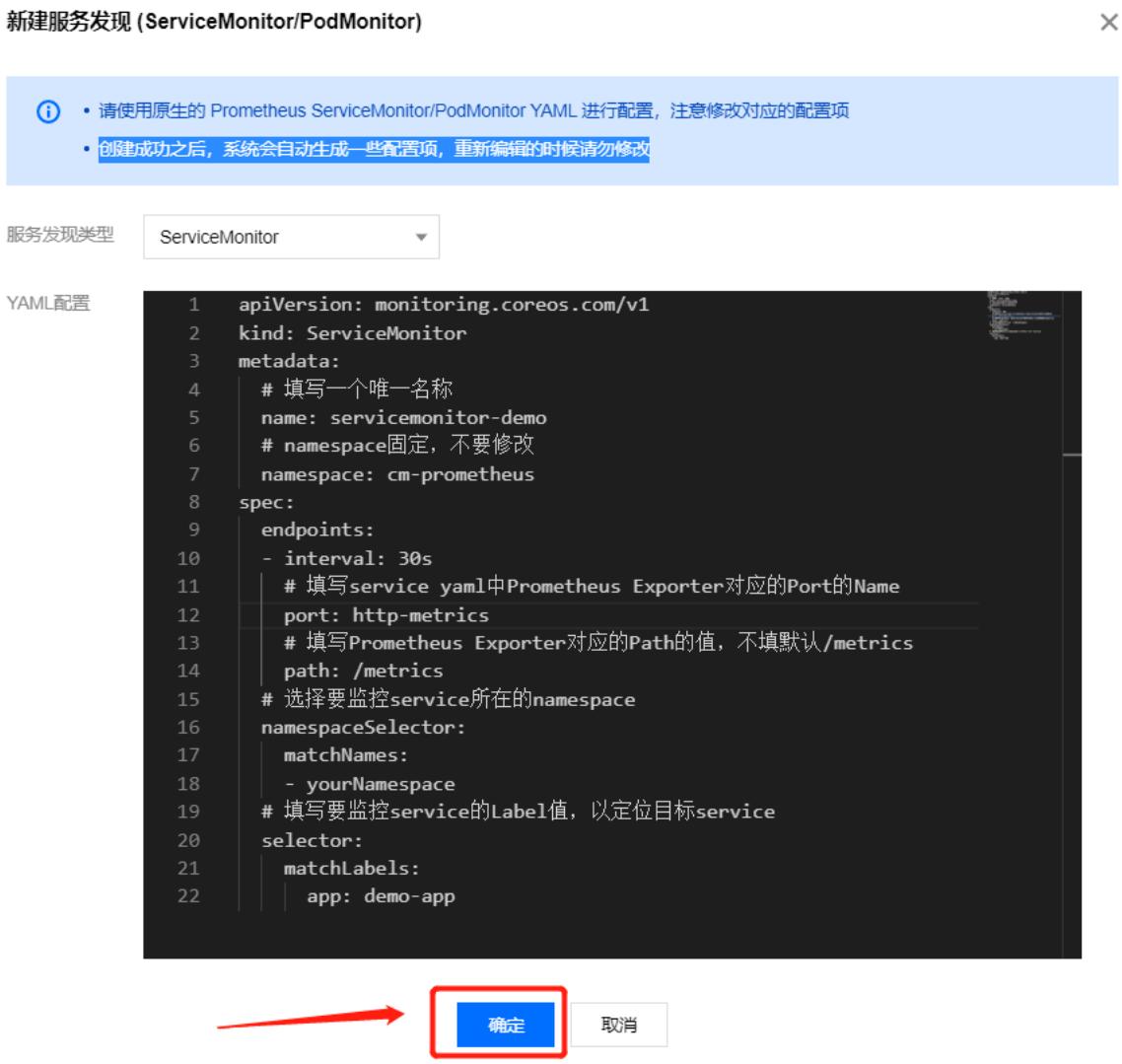
您可以通过管理【ServiceMonitor】或者【PodMonitor】来监控在容器服务 TKE 中服务。

准备工作

- 已经对某个容器服务 TKE 安装了 Prometheus Agent，具体可以参考 Agent 管理。
- 通过容器服务列表点击【集群 ID】进入到容器服务集成管理页面。

操作步骤

1. 单击【集成容器服务】上面的【服务发现】，进入服务发现的管理页面。
2. 单击【新建】弹出新建页面，在新建服务发现管理页中，选择服务发现类型。根据提示修改 YAML 配置。



3. 配置完后，单击【确认】即可。

> 说明：
> 创建成功之后，系统会自动生成一些配置项，重新编辑的时候请勿修改。

基础监控

最近更新时间：2020-09-28 19:49:08

用户在安装 Prometheus Agent 的时候可以选择所需要基础监控，但是有些情况下用户需要事后开启或者关闭某些监控监控。

准备工作

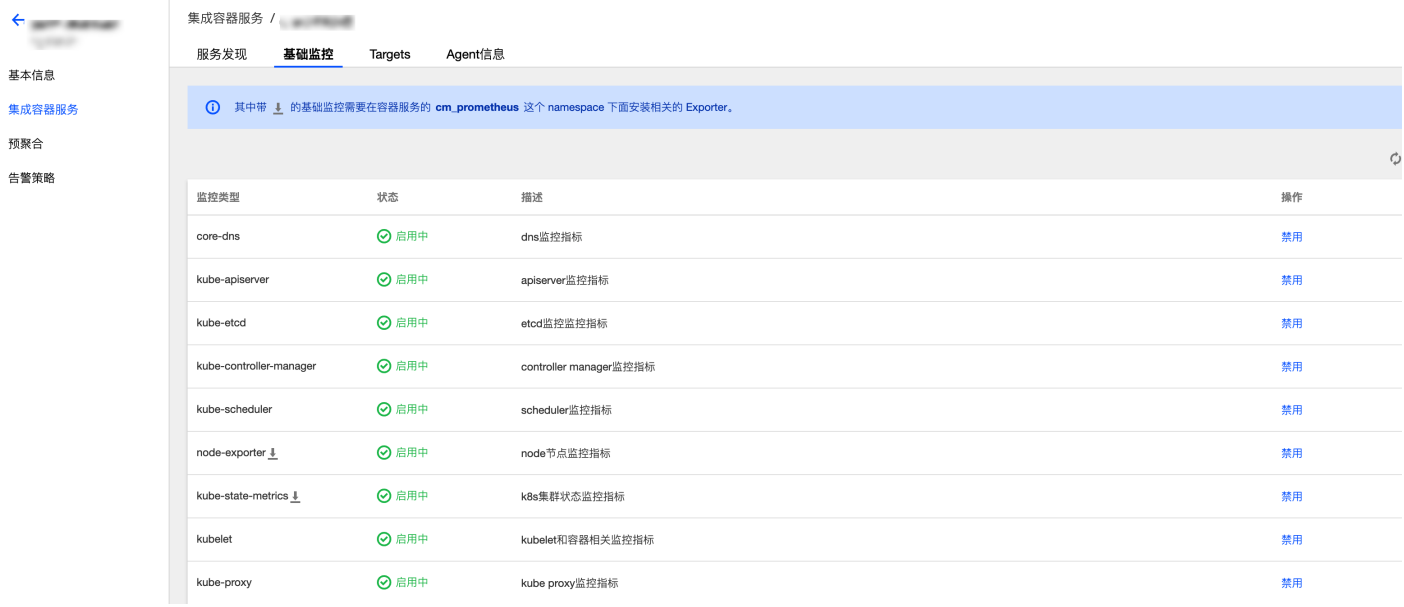
- 已经对某个容器服务 TKE 安装了 Prometheus Agent，详情请参见 Agent 管理。
- 通过容器服务列表单击【集群 ID】进入到容器服务集成管理页面。

监控组件说明

组件名称	作用
core-dns	采集 DNS 的监控指标数据
kube-apiserver	采集 APISever 的监控指标数据
kube-etcd	采集 Etcd 的监控指标数据
kube-controller-manager	采集 Controller Manager 的监控指标数据
kube-scheduler	采集 Scheduler 的监控指标数据
node-exporter	采集 Node节点的监控指标数据
kube-state-metrics	采集 k8s 集群状态的监控指标数据
kubelet	采集 Kubelet 和容器相关监控指标的监控数据
kube-proxy	采集 Kube Proxy 的监控指标数据

操作步骤

1. 在基础监控列表中，可以对【未开启】状态的基础监控进行开启操作。
2. 在基础监控列表中，可以对【已开启】状态的基础监控进行关闭操作。



> 说明：
> 相应的操作为异步操作，大约需要2 – 3分钟左右。

Targets

最近更新时间：2021-02-01 09:26:25

为了方便用户查看 Prometheus 抓取运行的状态，实时了解监控数据是否正常被 Prometheus Agent 抓取到。

准备工作

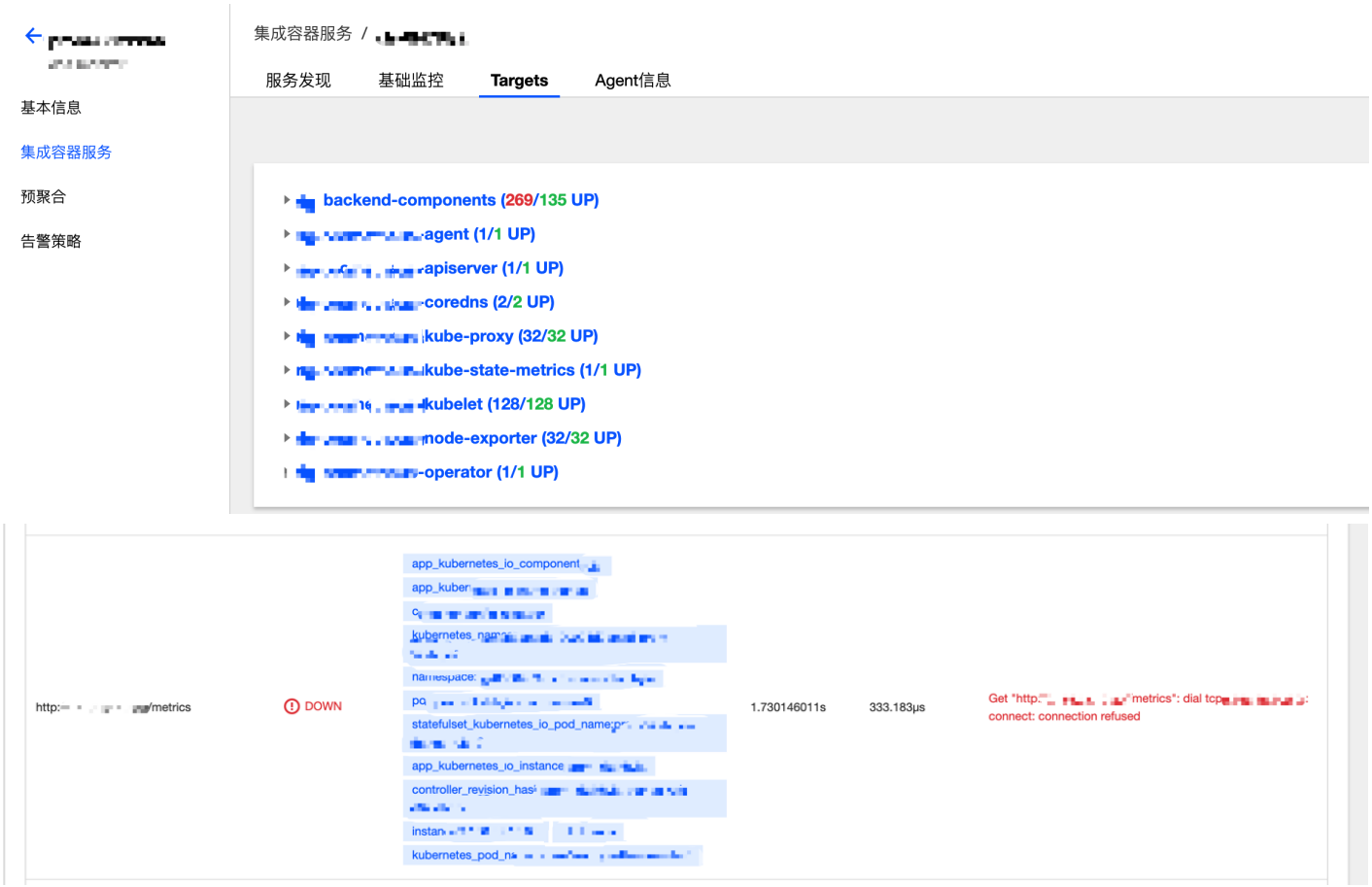
- 已经为容器服务 TKE 集群安装 Prometheus Agent，具体可以参考 Agent 管理。
- 通过 容器服务 集群管理页面中单击【集群 ID】进入到容器服务集成管理页面。

说明

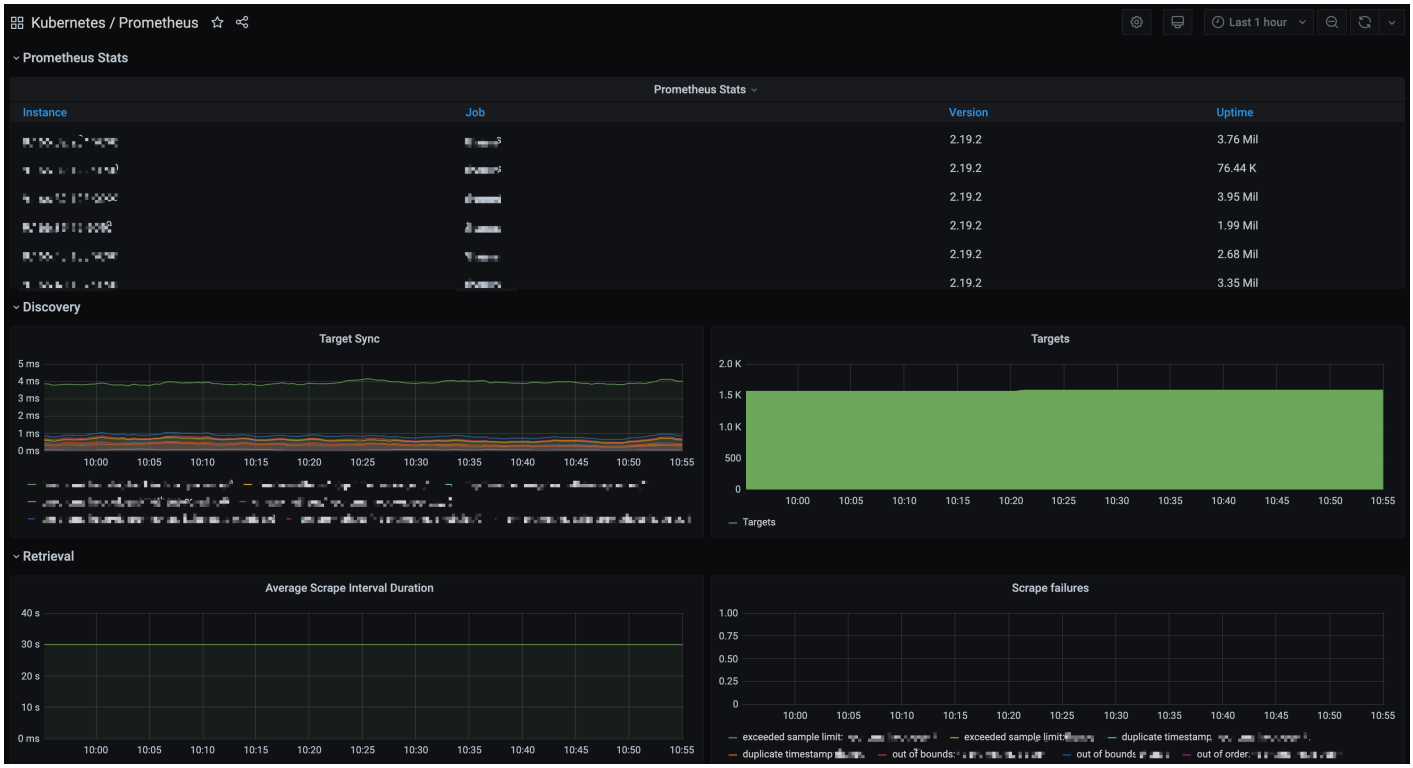
- 在任务状态中没有找到对应的抓取任务，说明 Prometheus Agent 没有正确获取到对应的抓取任务的配置，请查看对应的配置是否正确。
- 【红色】数值表示该任务下有多少个抓取任务失败，展开可以查看具体的失败原因。

操作步骤

1. 单击【集成容器服务】上面的【Targets】，可以查看当前 Prometheus Agent 所有抓取任务的任务以及失败的原因，如下图：



2. 同时也可以通过云监控 Prometheus 托管服务提供的 Grafana 服务查看更详细的抓取任务的监控状态，打开对应的 Grafana 查看预设的【Prometheus】Dashboard 来查看，如下图：


Kubernetes / Prometheus
Last 1 hour
Prometheus Stats
Prometheus Stats
Instance
Job
Version
Uptime
2.19.2
3.76 Mil
2.19.2
76.44 K
2.19.2
3.95 Mil
2.19.2
1.99 Mil
2.19.2
2.68 Mil
2.19.2
3.35 Mil
Discovery
Target Sync
5 ms
4 ms
3 ms
2 ms
1 ms
0 ms
10:00
10:05
10:10
10:15
10:20
10:25
10:30
10:35
10:40
10:45
10:50
10:55
Targets
2.0 K
1.5 K
1.0 K
500
0
Targets
Retrieval
Average Scrape Interval Duration
40 s
30 s
20 s
10 s
0 ms
Scrape failures
1.00
0.75
0.50
0.25
0
exceeded sample limit:
exceeded sample limit:
duplicate timestamp:
duplicate timestamp
out of bounds:
out of bounds
out of order:

Agent 信息

最近更新时间：2020-09-28 19:49:16

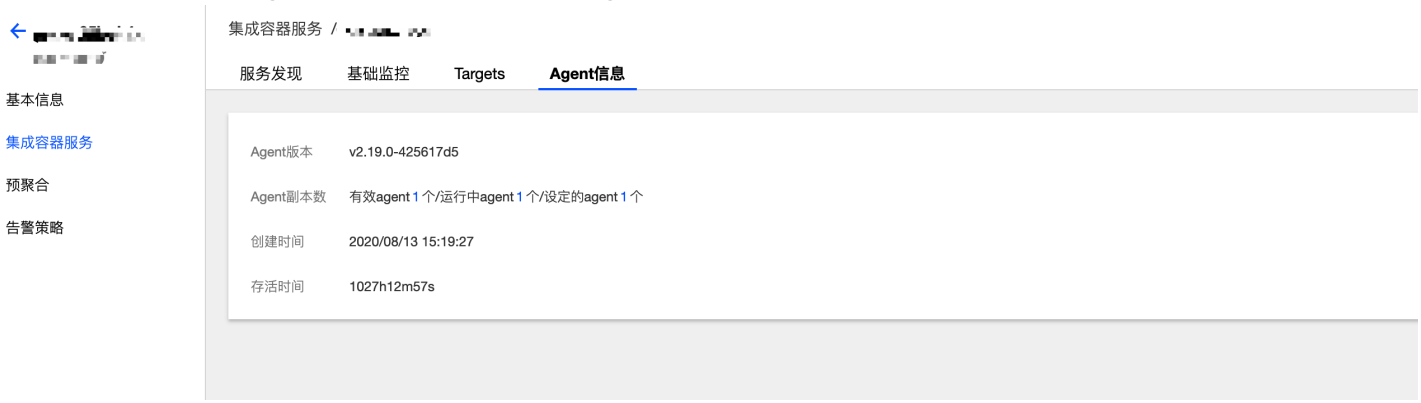
为了方便用户查看 Prometheus Agent 运行的状态，实时了解 Agent 是否工作正常。

前提条件

- 已经对某个容器服务 TKE 安装了 Prometheus Agent，详情请参见 Agent 管理。
- 通过容器服务列表点击【集群 ID】进入到容器服务集成管理页面。

操作步骤

单击【集成容器服务】上面的【Agent 信息】可以查看 Prometheus Agent 在容器服务内的运行情况，如下图所示。



告警

概述

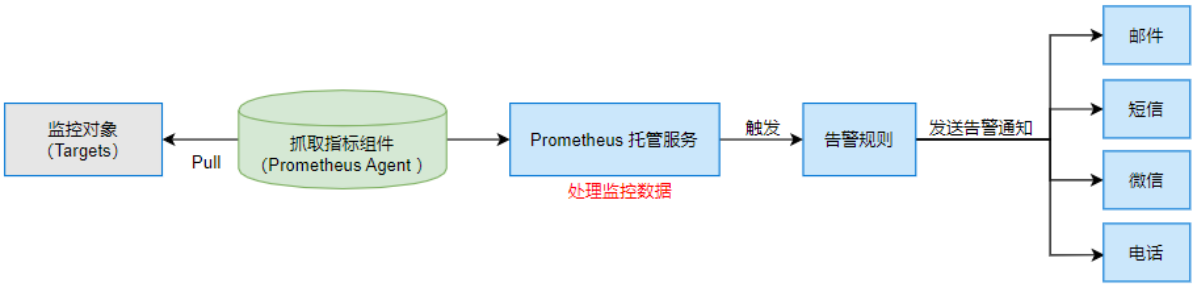
最近更新时间：2020-10-19 11:13:32

云监控 Prometheus 托管服务支持您基于 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。

Prometheus 托管服务告警结合了云监控告警能力和 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。
- 使用云监控告警通知模板，支持邮件、短信、电话多种接收方式



组成部分

- 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。

版权所有：腾讯云计算（北京）有限责任公司

告警规则说明

最近更新时间：2020-09-30 16:21:52

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."
```

版权所有：腾讯云计算（北京）有限责任公司

```
# 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：2021-04-02 09:17:52

本文指导您在云监控 Prometheus 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。

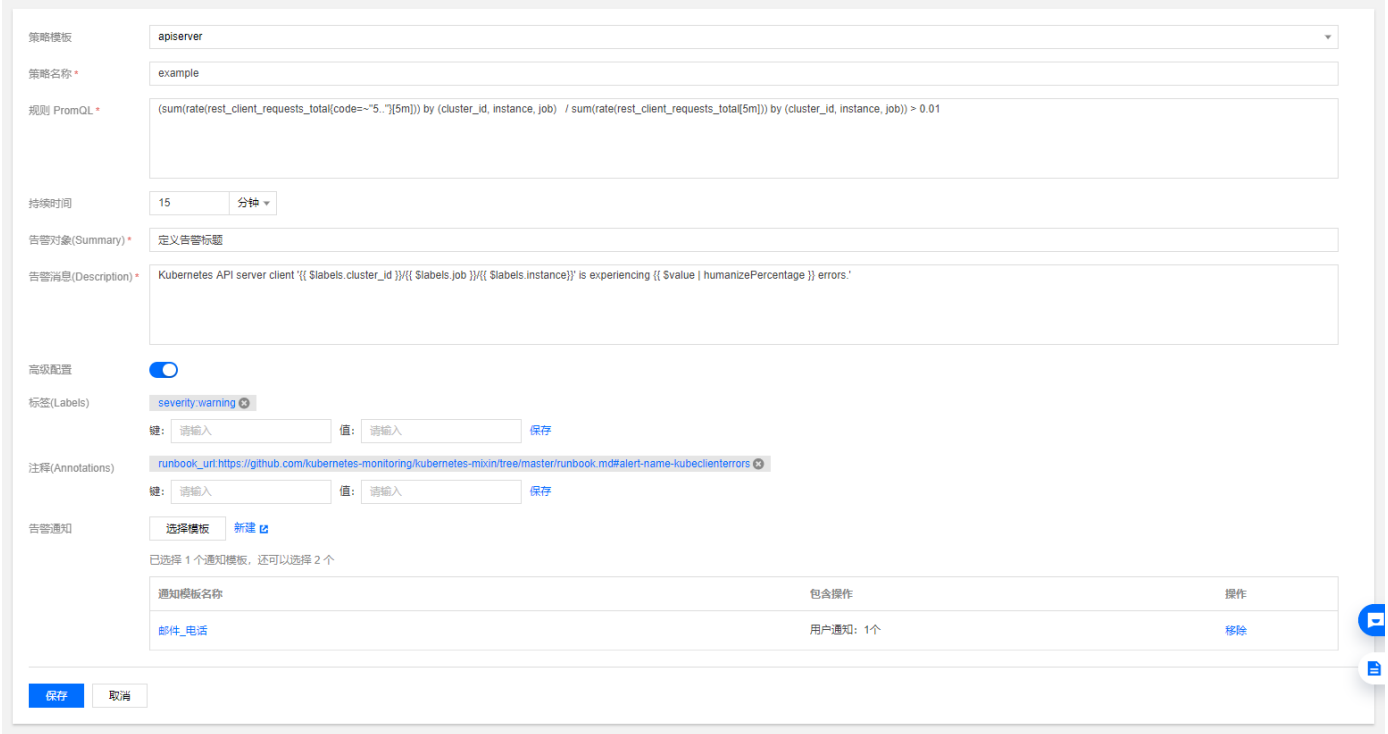
前提条件

- 已创建 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。
- 已创建 Prometheus 实例。
- 已安装 Prometheus Agent 及其监控组件。

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 云监控 Prometheus 控制台。
2. 在实例列表中，选择对应的 Prometheus 实例，单击左侧菜单栏的【告警策略】。
3. 在告警策略管理页中单击【新建】，在弹框中配置告警策略信息。
   - **策略模板类型**：选择策略模板类型，详细说明请参见 告警策略类型说明。
   - **策略名称**：可使用默认策略名称也可自定义。
   - **规则 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。
   - **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。
   - **告警对象**：允许用户自定义告警标题。
   - **告警消息**：可使用默认模板也可自定义，表示允许用户自定义告警内容。
   - **高级配置**：点击开启配置，包含标签和注释配置。
     - **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。
     - **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。
   - **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。



关闭告警策略

最近更新时间：2020-09-30 16:03:23

本文指导您在云监控 Prometheus 控制台中，如何关闭目标实例下的告警策略。

操作步骤

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，单击左边菜单栏的【告警策略】。
3. 找到需要关闭的告警策略，在操作列表中单击【关闭】。
4. 在弹框中单击【确定】即可。

策略类型说明

最近更新时间：2020-09-30 16:03:15

云监控为 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 (instance, job, cluster_id) / sum(rate(rest_client_requests_total[5m])) by (instance, job, cluster_id))> 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_id, 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_id, 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_id) (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_id) > 0)	5m	APIServer 从采集目标中消失
Scheduler 故障	absent(sum(up{job="kube-scheduler"}) by (cluster_id) > 0)	15m	Scheduler 从采集目标中消失
Controller Manager 故障	absent(sum(up{job="kube-controller-manager"}) by (cluster_id) > 0)	15m	Controller Manager 从采集目标中消失

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_id, node) ((kube_pod_status_phase{job=~".*kube-state-metrics",phase="Running"} == 1) * on(instance,pod,namespace,cluster_id) group_left(node) topk by(instance,pod,namespace,cluster_id) (1, kube_pod_info{job=~".*kube-state-metrics"}))/max by(cluster_id, node) (kube_node_status_capacity_pods{job=~".*kube-state-metrics"} != 1) > 0.95	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_id, 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

版权所有：腾讯云计算（北京）有限责任公司

策略名称	策略表达式	持续时间	策略描述
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_id, instance, le) * on(instance, cluster_id) 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_id, instance, le)) * on(cluster_id, 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_id) > 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_id) (max by (cluster_id, namespace, pod, container) (kube_pod_container_resource_requests_cpu_cores{job=~".*kube-state-metrics"}) * on(cluster_id, namespace, pod) group_left() max by (cluster_id, namespace, pod) (kube_pod_status_phase{phase=~"Pending|Running"} == 1))/sum by (cluster_id) (kube_node_status_allocatable_cpu_cores)>(count by (cluster_id) (kube_node_status_allocatable_cpu_cores)-1) / count by (cluster_id) (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_id) (max by (cluster_id, namespace, pod, container) (kube_pod_container_resource_requests_memory_bytes{job=~".*kube-state-metrics"}) * on(cluster_id, namespace, pod) group_left() max by (cluster_id, namespace, pod) (kube_pod_status_phase{phase=~"Pending|Running"} == 1))/sum by (cluster_id) (kube_node_status_allocatable_memory_bytes) > (count by (cluster_id) (kube_node_status_allocatable_memory_bytes)-1) / count by (cluster_id) (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_id) (kube_resourcequota{job=~".*kube-state-metrics", type="hard", resource="cpu"})/sum by (cluster_id) (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_id) (kube_resourcequota{job=~".*kube-state-metrics", type="hard", resource="memory"}) / sum by (cluster_id) (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_id, namespace, resource) kube_resourcequota{job=~".*kube-state-metrics", type="used"} / sum by (cluster_id, 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_id, container, pod, namespace) /sum(increase(container_cpu_cfs_periods_total{}[5m])) by (cluster_id, 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"}[1m])) by (cluster_id, namespace, pod, container) / sum(kube_pod_container_resource_limits_cpu_cores) by (cluster_id, namespace, pod, container) > 0.75	15m	Pod 的 CPU 使用率超过75%

策略名称	策略表达式	持续时间	策略描述
Pod 的内存使用率高	sum(rate(container_memory_working_set_bytes{job="kubelet", metrics_path="/metrics/cadvisor", image!="", container!="POD"}[1m])) by (cluster_id, namespace, pod, container) /sum(kube_pod_container_resource_limits_memory_bytes) by (cluster_id, namespace, pod, container) > 0.75	15m	Pod 的内存使用率超高跟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_id) (max by(namespace, pod, cluster_id) (kube_pod_status_phase{job=~".*kube-state-metrics", phase=~"Pending|Unknown"}) * on(namespace, pod, cluster_id) group_left(owner_kind) topk by(namespace, pod) (1, max by(namespace, pod, owner_kind, cluster_id) (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_id) (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_observed_generation{job=~".*kube-state-metrics"} != kube_statefulset_metadata_generation{job=~".*kube-state-metrics"}	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_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	实际副本数和设置副本数不一致
Statefulset 更新未生效	(maxwithout(revision)(kube_statefulset_status_current_revision{job=~".*kube-state-metrics"}unlesskube_statefulset_status_update_revision{job=~".*kube-state-metrics"})*(kube_statefulset_replicas{job=~".*kube-state-metrics"}!=kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"})) and (changes(kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"}[5m])==0)	15m	Statefulset 部分 pod 没有更新
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

策略名称	策略表达式	持续时间	策略描述
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"}) and changes(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<15 and predict_linear(node_filesystem_avail_bytes{job="node-exporter",fstype!=""}[6h],4*60*60)<0 and 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<5 and 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<20 and predict_linear(node_filesystem_files_free{job="node-exporter",fstype!=""}[6h],4*60*60)<0 and 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<3 and 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])	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

策略名称	策略表达式	持续时间	策略描述
机器时钟漂移	(node_timex_offset_seconds>0.05 and deriv(node_timex_offset_seconds[5m])>=0) or (node_timex_offset_seconds<-0.05 and 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

预聚合

概述

最近更新时间：2020-10-09 17:34:20

什么是预聚合规则

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> ]
```

以下为一个简单预聚合规则例子：

```
groups:
- name: example
rules:
- record: job:http_inprogress_requests:sum
expr: sum by (job) (http_inprogress_requests)
```

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` 。

1. level：表示聚合级别，以及规则的输出标签。
2. metric：是指标名称。
3. 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：2020-09-28 19:49:39

操作场景

用户可以通过云监控 Prometheus 托管服务控制台来管理 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。

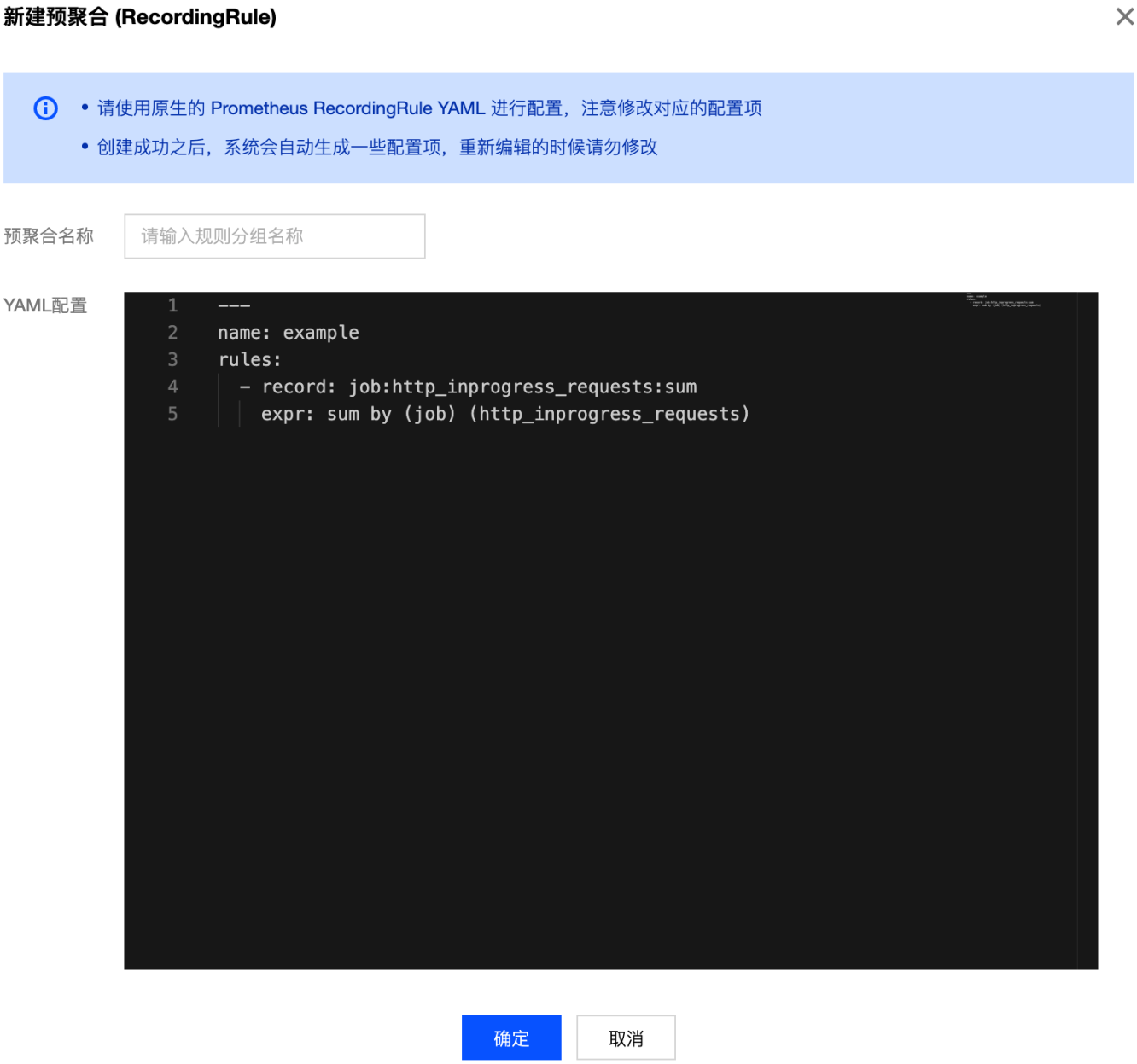
准备工作

1. 登录 云监控 Prometheus 控制台。
2. 创建 Prometheus 托管实例，详情请参见 创建实例。
3. 通过实例列表进入到 Prometheus 实例的管理页面。
4. 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. 通过实例管理页面的左侧菜单【预聚合】>【新建】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。



2. 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。

标签

标签示例

最近更新时间：2020-10-09 17:33:35

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 托管资源进行分类管理，通过标签非常方便地筛选过滤出对应的资源。标签键值对对腾讯云没有任何语义意义，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 托管服务实例，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 托管服务资源，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 托管服务实例
  按照筛选规则筛选出运维负责人为“王五”的 Prometheus 资源即可，具体筛选办法请参考 使用标签。
- 筛选出游戏部门中李四负责的 Prometheus 托管服务实例
  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，具体筛选办法请参考 使用标签。

版权所有：腾讯云计算（北京）有限责任公司

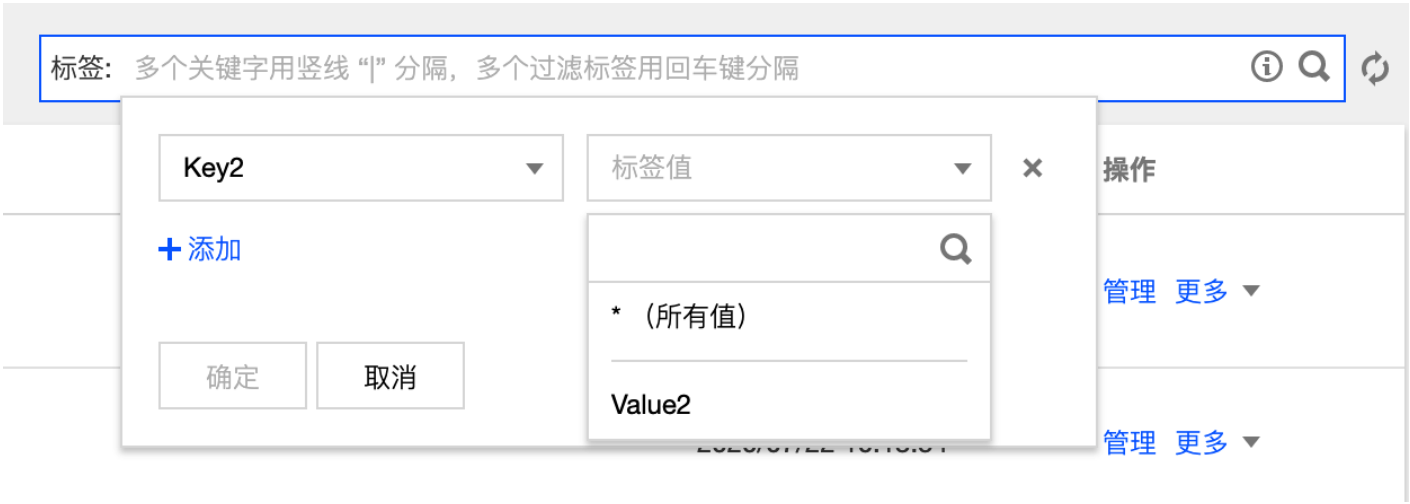
使用标签

最近更新时间：2020-12-04 15:58:21

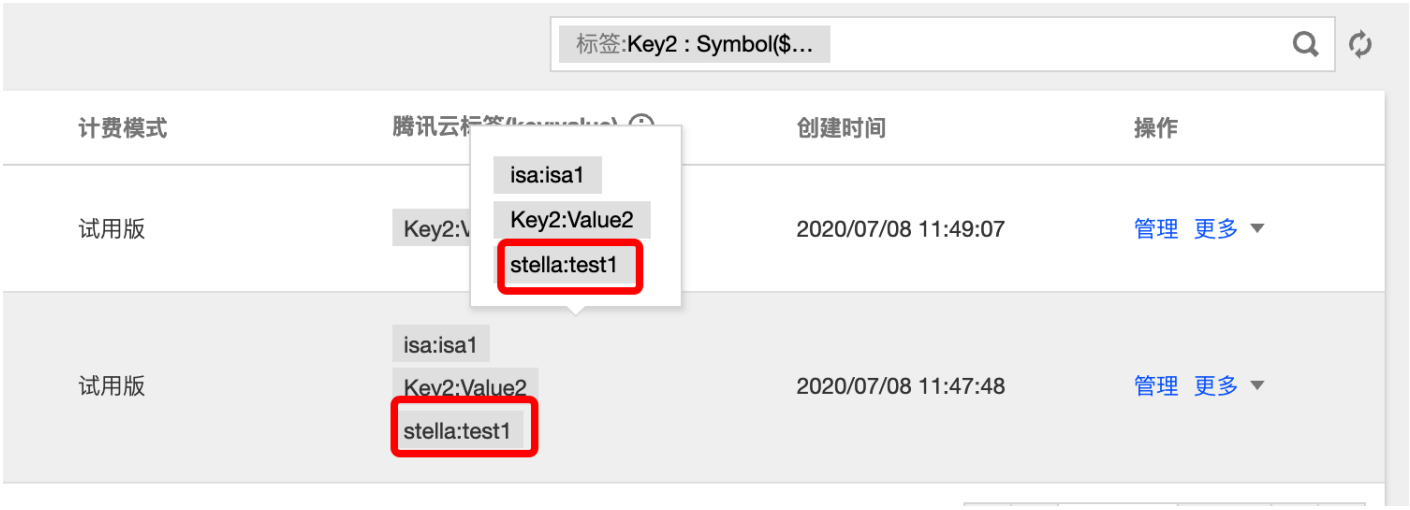
本文指导您在云监控 Prometheus 控制台中，根据标签对实例进行资源筛选，过滤出对应的资源。

操作步骤

1. 登录 云监控 Prometheus 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:】后面的标签内容进行编辑。
6. 同时也支持直接点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：2020-12-04 15:58:25

本文指导您在云监控 Prometheus 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 云监控 Prometheus 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“您已经选择1个云资源”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：2020-09-28 19:50:01

标签是一个键-值对（Key-Value），您可以在云监控 Prometheus 控制台，通过对 Prometheus 托管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255个字符。

标签值限制

- 只能为字母、数字、空格或汉字，支持+、-、=、.、_、:、/、@。
- 标签值最大长度为127个字符。

版权所有：腾讯云计算（北京）有限责任公司

访问控制

概述

最近更新时间：2020-09-28 19:50:15

如果您在腾讯云中使用到了云监控 Prometheus 托管服务，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。
- 您无法限制其它人的访问权限，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 托管服务部分 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。

版权所有：腾讯云计算（北京）有限责任公司

策略设置

最近更新时间：2020-10-09 17:33:50

概述

访问策略可用于授予访问 Prometheus 托管服务实例的权限。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 托管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考 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

云监控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 托管服务资源等。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 托管服务的六段式信息：

```
"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：2020-09-28 19:50:23

Prometheus 自定义策略

如果预设策略无法满足需求，可单击【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考 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。Prometheus 托管服务部分支持资源级权限，即表示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 托管服务	qcs::monitor:$region:$account:prom-instance/* qcs::monitor:$region:$account:prom-instance/$instanceId

下表将介绍当前支持资源级权限的 Prometheus 托管服务 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

API 操作	API 描述
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

不支持资源级授权的 API 列表

针对不支持资源级权限的 Prometheus 托管服务 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

服务授权相关角色权限说明

最近更新时间：2021-02-09 09:35:39

在使用腾讯云云监控 Prometheus 托管服务过程中，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 CM_QCSRole 服务角色。本文接下来将分角色展示各个授权策略的详情、授权场景及授权步骤。

CM_QCSRole 角色默认关联的预设策略包含如下：

QcloudAccessForCMRoleInPromHostingService：Prometheus 托管服务所需要的容器服务 TKE 权限。

场景

当您成功创建 Prometheus 服务实例之后，需要监控腾讯云容器服务（TKE）上运行的服务，为了可以更方便的集成容器服务，需要访问腾讯云容器服务（TKE）相关的API服务，需要您的授权委托，才能正常访问腾讯云容器服务（TKE）来安装基本的监控组件及获取对应监控组件的运行状态。

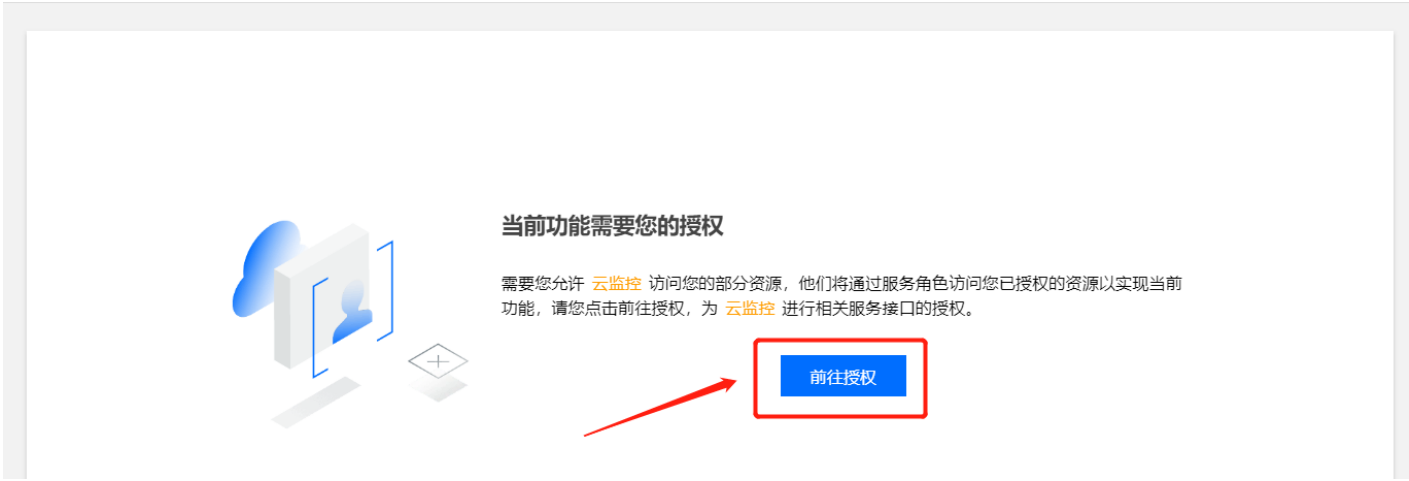
此角色无需主动寻找配置，在未授权的情况下，在您成功创建 Prometheus 服务实例后，进入到对应实例管理下的 “集成容器服务” 时会自动弹出授权界面。

授权步骤

主账号授权步骤

1. 当您成功 创建 Prometheus 实例 后，在访问“集成容器服务”时将出现授权提示框，如下所示。

集成容器服务



2. 在子窗口中单击【前往授权】。

3. 在访问管理—角色管理页单击【同意授权】，提示授权成功即可。

> 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。

子账号授权步骤

主账号完成上述授权操作，成功创建了 CM_QCSRole 角色后，子账号无权限访问 CM_QCSRole 角色，需子账号对主账号授予 PassRole 权限，子账号才能在 Prometheus 托管服务中正常访问容器服务 TKE，否则访问容器服务列表时会提示失败。

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 或 QcloudCamFullAcces
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	详情请参考 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创建如下自定义策略，策略语法如下：

```
{
"version": "2.0",
"statement": [
{
"effect": "allow",
"action": "cam:PassRole",
"resource": "qcs::cam::uin/${OwnerUin}:roleName/CM_QCSRole"
}
]
}
```

2. 新建完后，参考 访问管理-授权管理 在自定义策略下关联子账号即可。
授予子账号 cam:PassRole 权限之后，再次访问对应 Prometheus 实例管理下的“集成容器服务”页面，将出现授权提示框。

API 使用指南
API 概览

最近更新时间：2021-02-04 17:40:35

HTTP API

Prometheus 所有稳定的 HTTP API 都在 `/api/v1` 路径下。当我们有数据查询需求时，可以通过查询 API 请求监控数据，提交数据可以使用 remote write 协议或者 Pushgateway 的方式。

支持的 API

API	说明	需要认证	方法
/api/v1/query	查询接口	是	GET/POST
/api/v1/query_range	范围查询	是	GET/POST
/api/v1/series	series 查询	是	GET/POST
/api/v1/labels	labels 查询	是	GET/POST
/api/v1/label/<label_name>/values	label value 查询	是	GET
/api/v1/prom/write	remote write 数据提交	是	remote write
Pushgateway	pushgateway 数据提交	是	SDK

认证方法

默认开启认证，因此所有的接口都需要认证，且所有的认证方式都支持 Bearer Token和 Basic Auth。

Bearer Token

Bearer Token 随着实例产生而生成，可以通过控制台进行查询。了解 Bearer Token 更多信息，请参见 Bearer Authentication。

Basic Auth

Basic Auth 兼容原生 Prometheus Query 的认证方式，用户名为用户的 APPID，密码为 bearer token（实例产生时生成），可以通过控制台进行查询。了解 Basic Auth 更多信息，请参见 Basic Authentication。

数据返回格式

所有 API 的响应数据格式都为 JSON。每一次成功的请求会返回 `2xx` 状态码。

无效的请求会返回一个包含错误对象的 JSON 格式数据，同时也将包含一个如下表格的状态码：

状态码	含义
401	认证失败
400	当参数缺失或错误时返回无效的请求状态码
422	当一个无效的表达式无法被指定时 (RFC4918)
503	当查询不可用或者被取消时返回服务不可用状态码

无效请求响应返回模板如下：

```
{
"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>"]
}
```

监控数据查询

最近更新时间：2020-12-04 15:57:05

Prometheus 提供查询 API 接口，通过该接口可以查询监控数据。

查询 API 接口

```
GET /api/v1/query
POST /api/v1/query
```

查询参数说明

- 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。

简单查询示例

如下示例为 API 数据查询，查询服务地址和认证信息可以在相应实例控制台查看：

```
curl -u "appid:token" 'http://IP:PORT/api/v1/query?query=up'
```

如果返回状态码为 401，请检查认证信息是否正确。

```
HTTP/1.1 401 Unauthorized
Content-Length: 0
```

范围查询

```
GET /api/v1/query_range
POST /api/v1/query_range
```

通过 /api/v1/query_range 接口，可以查询指定时间范围的数据。示例如下：

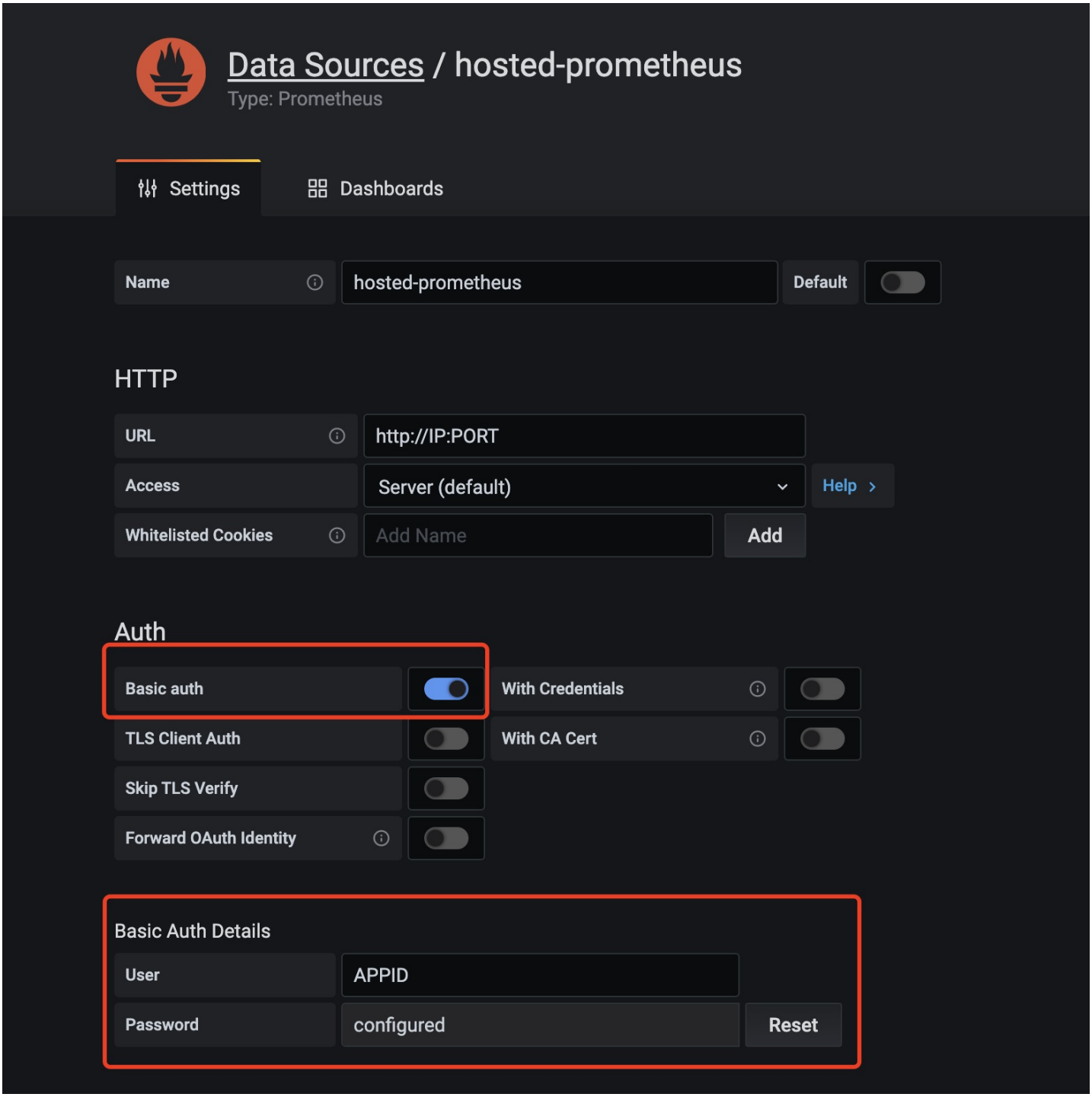
```
$ curl -u "appid:token" 'http://IP:PORT/api/v1/query_range?query=up&start=2015-07-01T20:10:30.781Z&end=2015-07-01T20:11:00.781Z&
step=15s'
{
"status" : "success",
"data" : {
"resultType" : "matrix",
"result" : [
{
"metric" : {
"__name__" : "up",
"job" : "prometheus",
"instance" : "localhost:9090"
},
"values" : [
[ 1435781430.781, "1" ],
[ 1435781445.781, "1" ],
[ 1435781460.781, "1" ]
]
},
{
"metric" : {
"__name__" : "up",
"job" : "node",
```

```
"instance" : "localhost:9091"
},
"values" : [
[ 1435781430.781, "0" ],
[ 1435781445.781, "0" ],
[ 1435781460.781, "1" ]
]
}
]
}
}
```

自建 Grafana 添加数据源

通过部署的 Grafana 可以添加托管的 Prometheus 为数据源，方便自己的 Grafana 中查看数据，前提需要保证 Grafana 和 Prometheus 在同一 VPC 内，保证网络可以互相访问。

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

数据写入

最近更新时间：2020-12-04 15:56:46

操作场景

类似 flink job 上报数据的场景，我们需要通过 API 直接将数据写入 Prometheus，因为 job 的生命周期可能会很短，来不及等待 Prometheus 来拉取数据。写入数据可以直接使用 Remote Write 协议或者 Pushgateway 的方式。

Remote Write

```
POST /api/v1/prom/write
```

Remote Write 是 Prometheus 标准的协议，更多介绍可以参考 Prometheus-remote write 。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 场景主张以拉为主，但是有些场景我们仍然需要使用推的方式，可以参考 Prometheus -Pushgateway 说明。

腾讯云 Prometheus 托管服务原生集成了 Pushgateway 模块，可以直接推送数据到服务中。下面是一个使用 Go 语言推送数据的简单的例子，记得将 `$IP`、`$PORT`、`$APPID`、`$TOKEN` 这些变量改为自己实例的认证链接信息，这些链接信息可以从控制台进行查询。

```
package main
import (
"fmt"
"time"
"github.com/prometheus/client_golang/prometheus"
"github.com/prometheus/client_golang/prometheus/push"
)
var completionTime = prometheus.NewGauge(prometheus.GaugeOpts{
Name: "db_backup_last_completion_timestamp_seconds",
Help: "The timestamp of the last successful completion of a DB backup.",
})
func do() {
completionTime.SetToCurrentTime()
}
func ExamplePusher_Push() {
if err := push.New("http://$IP:$PORT", "db_backup").
BasicAuth("$APPID", "$TOKEN").
Collector(completionTime).
Grouping("db", "customers").
Push(); err != nil {
fmt.Println("Could not push completion time to Pushgateway:", err)
}
}
func main() {
do()
ticker := time.NewTicker(2 * time.Second)
done := make(chan bool)
for {
select {
case <-done:
return
case <-ticker.C:
ExamplePusher_Push()
}
```

```
}
}
```

> 注意：
> push.New 生成的对象可以通过 Client 方法自定义 HTTP Client，我们推荐设置一个合适的超时时间，同时 push 数据我们建议使用异步的方式调用，以避免阻塞业务主流程。

各个语言推送数据的 SDK 请参见 Prometheus 客户库。

集成中心

最近更新时间：2021-02-20 10:56:57

概述

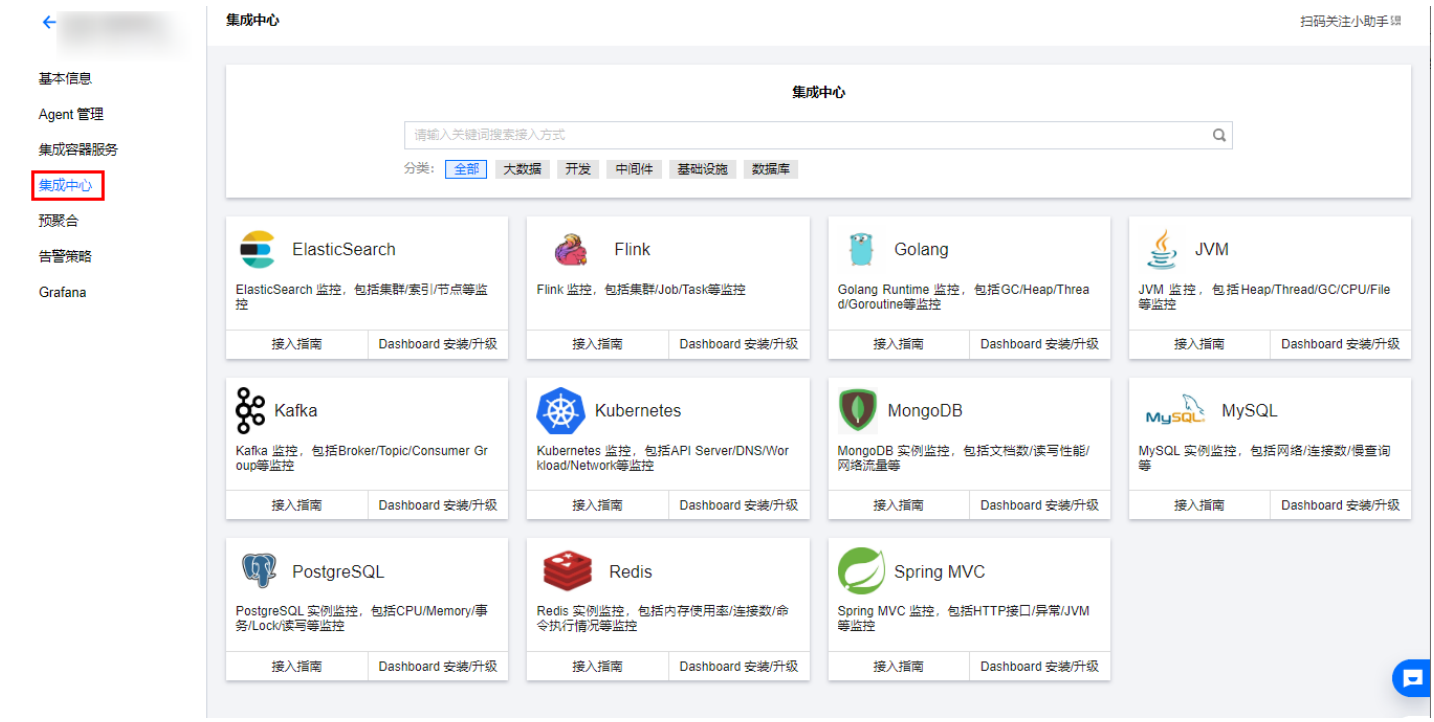
云监控 Prometheus 托管服务对常用的开发语言/中间件/大数据/基础设施数据库进行了集成，用户只需根据指引即可对相应的组件进行监控，同时提供了开箱即用的 Grafana 监控大盘。集成中心涵盖了基础服务监控，应用层监控、Kubernetes 容器监控三大监控场景，方便您快速接入并使用。

支持服务

服务类型	服务名称	监控项
大数据	ElasticSearch	包括 集群/索引/节点 等监控
	Flink	包括 集群/Job/Task 等监控
开发语言	Golang	包括 GC/Heap/Thread/Goroutine 等监控
	JVM	包括 Heap/Thread/GC/CPU/File 等监控
	Spring MVC	包括 HTTP接口/异常/JVM 等监控
中间件	Kafka	包括 Broker/Topic/Consumer Group 等监控
基础设施	Kubernetes	包括 API Server/DNS/Workload/Network 等监控
数据库	云数据库 MongoDB	包括 文档数/读写性能/网络流量 等
	云数据库 MySQL	包括 网络/连接数/慢查询 等
	云数据库 PostgreSQL	包括 CPU/Memory/事务/Lock/读写 等监控
	云数据库 Redis	包括 内存使用率/连接数/命令执行情况 等监控

操作步骤

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】按钮安装或升级该服务的 Grafana Dashboard。


集成中心
扫码关注小助手
基本信息
Agent 管理
集成容器服务
集成中心
预聚合
告警策略
Grafana
集成中心
请输入关键词搜索接入方式
分类：
全部
大数据
开发
中间件
基础设施
数据库
ElasticSearch
ElasticSearch 监控，包括集群/索引/节点等监控
接入指南
Dashboard 安装/升级
Flink
Flink 监控，包括集群/Job/Task等监控
接入指南
Dashboard 安装/升级
Golang
Golang Runtime 监控，包括GC/Heap/Thread/Goroutine等监控
接入指南
Dashboard 安装/升级
JVM
JVM 监控，包括Heap/Thread/GC/CPU/File等监控
接入指南
Dashboard 安装/升级
Kafka
Kafka 监控，包括Broker/Topic/Consumer Group等监控
接入指南
Dashboard 安装/升级
Kubernetes
Kubernetes 监控，包括API Server/DNS/Workload/Network等监控
接入指南
Dashboard 安装/升级
MongoDB
MongoDB 实例监控，包括文档数/读写性能/网络流量等
接入指南
Dashboard 安装/升级
MySQL
MySQL 实例监控，包括网络/连接数/慢查询等
接入指南
Dashboard 安装/升级
PostgreSQL
PostgreSQL 实例监控，包括CPU/Memory/事务/Lock/读写等监控
接入指南
Dashboard 安装/升级
Redis
Redis 实例监控，包括内存使用率/连接数/命令执行情况等监控
接入指南
Dashboard 安装/升级
Spring MVC
Spring MVC 监控，包括HTTP接口/异常/JVM等监控
接入指南
Dashboard 安装/升级

Grafana

概述

最近更新时间：2021-02-20 10:57:09

云监控 Prometheus 托管服务支持您一键安装 Grafana 主流插件和手动添加客户端所属的 IP 地址到 Grafana 白名单，以允许该客户端访问对应的 Grafana。

Grafana 插件

开源Grafana的插件分为三种：Application，Panel 和 Datasource。详情可参考 Grafana 插件介绍。
目前 Prometheus 托管服务支持两种插件一键安装：

- grafana-clock-panel：“时钟”图表类型插件。
- grafana-piechart-panel：“饼图”图表类型插件。

如需了解 Grafana 插件安装步骤请参考 Grafana 插件指引。

> 说明：
> 如果发现所需的插件没有在预设列表里面，可以添加云监控助手（ hitherecm ）进行反馈，我们竭诚为您服务！

白名单

为了增强您 Grafana 监控数据的安全性，您可以为 Grafana 设置白名单 IP，用于限制非法 IP 访问对应的 Grafana。详情请参考 白名单指引。

Grafana 插件

最近更新时间：2021-02-20 10:57:15

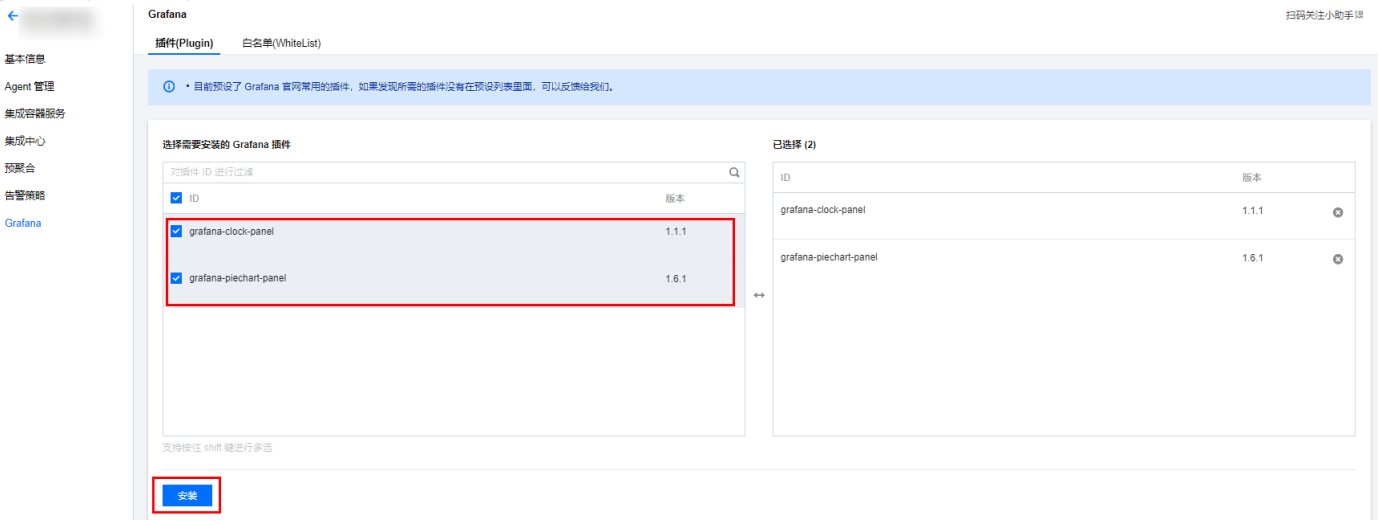
本文将为您介绍如何安装 Prometheus 托管服务的 Grafana 插件。

前提条件

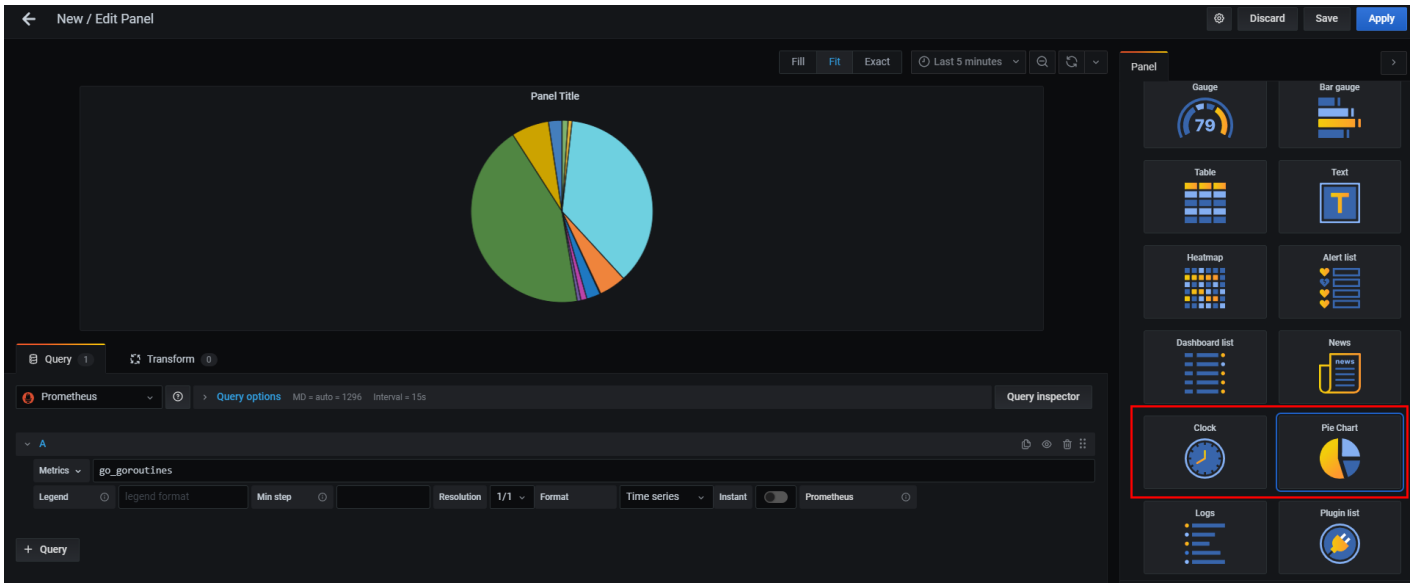
已创建 Prometheus 实例。

操作步骤

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. 在左侧菜单栏中选择【Grafana】。
4. 在 Grafana 中勾选对应的插件，勾选完后，单击【安装】，待系统提示“提交安装 Grafana 插件任务成功”即安装成功。
   - grafana-clock-panel：“时钟”图表类型插件。
   - grafana-piechart-panel：“饼图”图表类型插件。



5. 插件安装成功后，您可以在 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，输入账号密码，即可在创建图表中应用此插件。



白名单

最近更新时间：2021-02-20 10:57:20

本文将为您介绍如何设置 Grafana 白名单 IP。

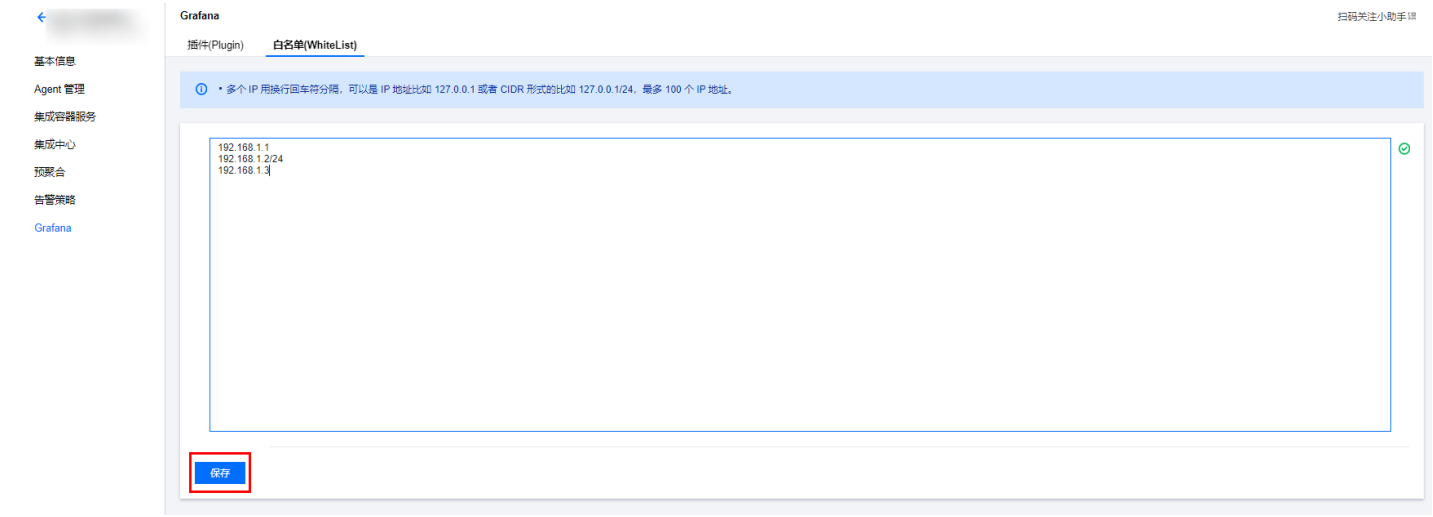
前提条件

已创建 Prometheus 实例。

操作步骤

您可以手动添加客户端所属的IP地址到 Grafana 白名单，以允许该客户端访问对应的 Grafana。操作步骤如下。

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. 在左侧菜单栏中选择【Grafana】。
4. 进入 Grafana 管理页后，单击【白名单(WhiteList)】> 单击文本框底部的【编辑】按钮。
5. 在文本框输入白名单地址，可输入 IP 地址，如：127.0.0.1 或者 CIDR 形式的地址，如： 127.0.0.1/24，多个 IP 地址请换行输入，如下图。



6. 输入完成后，单击【保存】即可。

Agent 管理

概述

最近更新时间：2021-02-20 10:57:30

Agent 管理用于实现监控对象部署在 CVM 或者 自建 IDC 时，自定义上报监控数据到 Prometheus 托管服务。您成功上报数据后，可以通过集成的开源可视化的 Grafana 查看监控大屏，还可以对该监控对象设置告警规则，实时监控其状态，在状态异常时第一时间给您发送告警通知。

操作介绍

1. 新建 Agent
2. 安装 Agent（上报监控数据到 Prometheus 托管服务）
3. 新建抓取任务（定义 Agent 抓取任务规则）

版权所有：腾讯云计算（北京）有限责任公司

新建 Agent

最近更新时间：2021-02-20 10:57:35

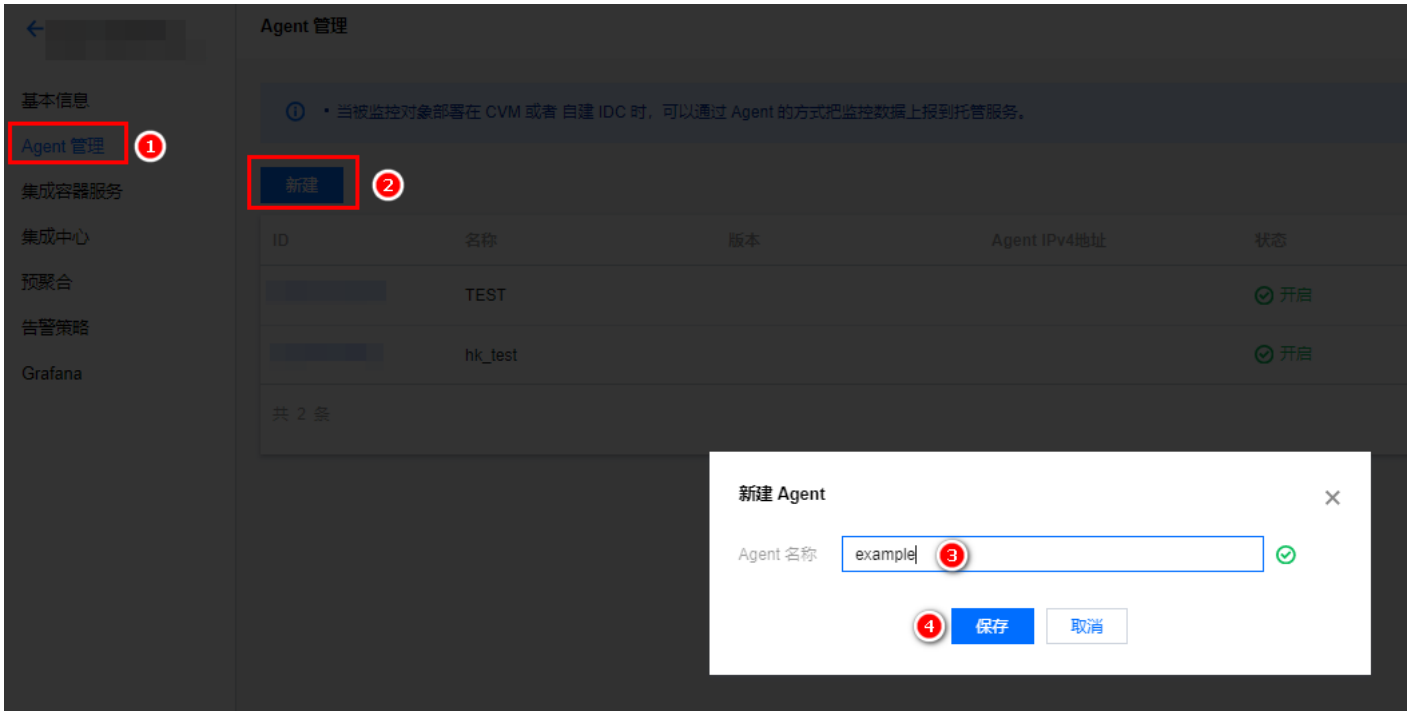
本文将为您介绍如何新建 Agent。

前提条件

已创建 Prometheus 实例。

操作步骤

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，单击左侧菜单栏的【Agent 管理】。
3. 在 Agent 管理页单击【新建】。
4. 在弹框中填写 Agent 名称，填写完后单击【保存】即可。



安装 Agent

最近更新时间：2021-02-26 17:55:12

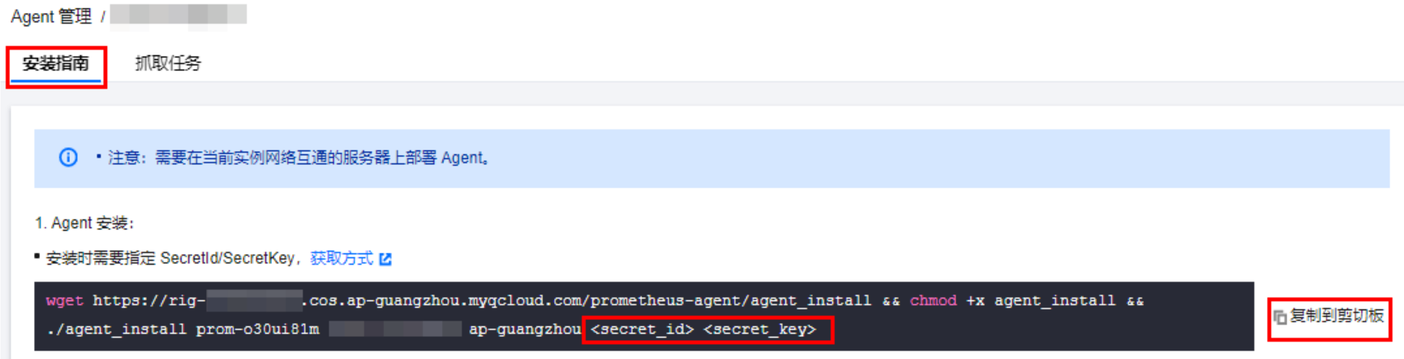
本文将为您介绍如何安装 Agent。

前提条件

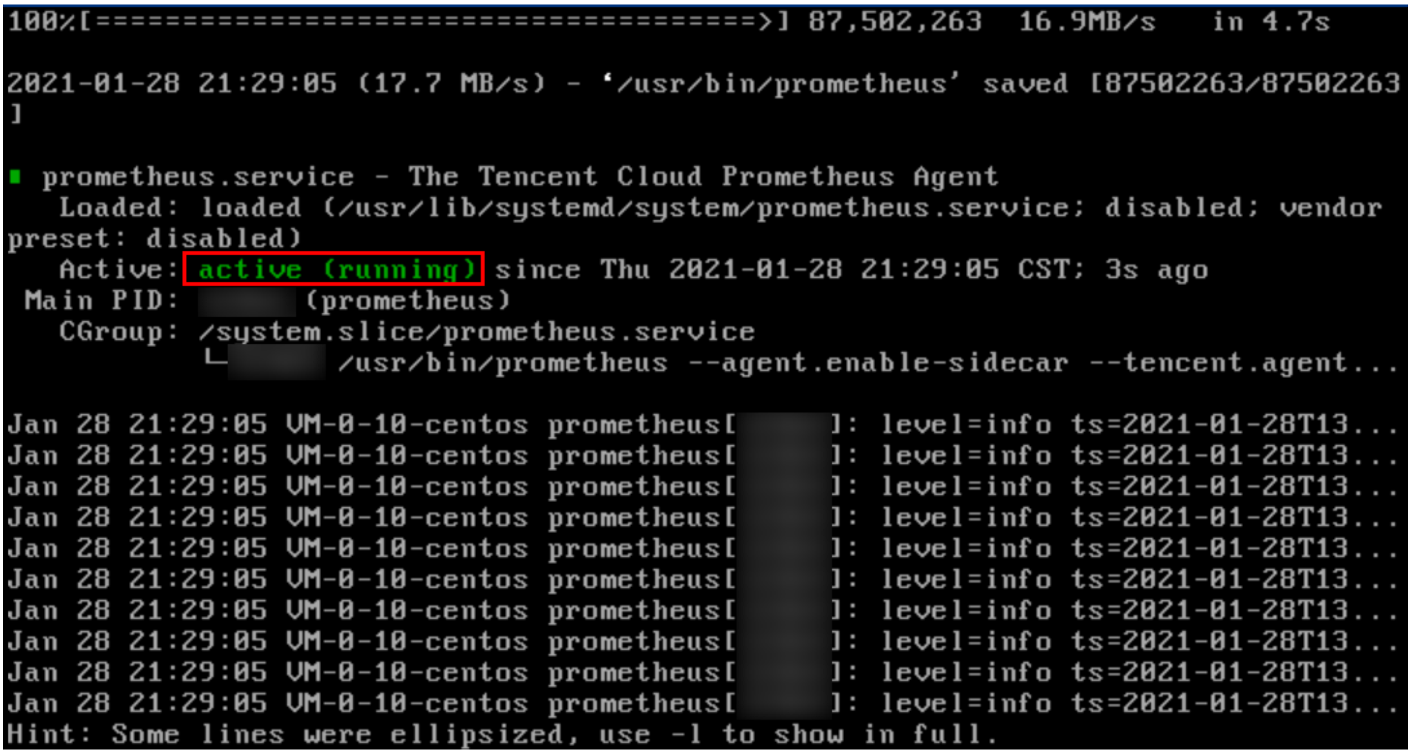
- 已创建 Prometheus 实例。
- 已 创建 Agent 。

操作步骤

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，单击左侧菜单栏的【Agent 管理】。
3. 单击 Agent 对应的 ID, 进入 Agent 安装指南页，复制 Agent 安装命令于您的云服务器或自建 IDC ，修改命令中的<secret_id>和<secret_key>后即可执行。



执行成功示例如下：



4. 返回 Agent 管理页面，如果 Agent 正常运行，可以看到 Agent 上报的版本，IP 地址和心跳时间。

其它命令

重启 Agent

执行如下命令：

```
systemctl restart prometheus
```

停止 Agent

执行如下命令：

```
systemctl stop prometheus
```

检查 Agent 状态

执行如下命令：

```
systemctl status prometheus
```

查看 Agent 日志

执行如下命令：

```
journalctl -f --unit=prometheus
```

卸载 Agent

执行如下命令：

```
systemctl stop prometheus && rm -rf /usr/lib/systemd/system/prometheus.service
```

版权所有：腾讯云计算（北京）有限责任公司

新建抓取任务

最近更新时间：2021-03-01 15:45:42

本文将为您介绍如何新建抓取任务。

前提条件

- 已创建 Prometheus 实例。
- 已 创建 Agent 并 安装 Agent。

操作步骤

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，单击左侧菜单栏的【Agent 管理】。
3. 单击 Agent 对应的 ID，单击【抓取任务】。
4. 进入抓取任务页，单击【新建】。
5. 在新建抓取任务页，根据提示填写 Agent 抓取任务规则，任务配置可以参考 抓取配置说明。

6. 配置完成后，单击【保存】即可。

接入指南

抓取配置说明

最近更新时间：2021-03-01 15:45:52

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。
- 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> ... ]
# CVM 服务发现配置，详见下面的说明。
cvm_sd_configs:
[ - <cvm_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> ... ]
# 数据抓取完成写入之前，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。
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> ... ]
```

cvm_sd_config 配置

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 服务发现配置说明：

```
# 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。
[ 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>, [...] ]
```

例子

静态配置

```
- job_name: 'prometheus'
scrape_interval: 5s
static_configs:
- targets: ['localhost:9090']
- job_name: 'node'
scrape_interval: 8s
static_configs:
- targets: ['127.0.0.1:9100', '127.0.0.12:9100']
- job_name: 'mysqld'
static_configs:
- targets: ['127.0.0.1:9104']
- job_name: 'memcached'
static_configs:
- targets: ['127.0.0.1:9150']
```

CVM 服务发现配置

```
- 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固定，不需要修改
# 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://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 ]
```

案例

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
name: redis-exporter # 填写一个唯一名称
namespace: cm-prometheus # 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
```

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固定，不需要修改
# 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 ]
```

案例

```
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: 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。
relabelings:
[ - <relabel_config> ...]
# 数据抓取完成写入之前，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 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 设值并添加相应的 label
# keep: 如果 regex 没有匹配到，丢弃
# drop: 如果 regex 匹配到，丢弃
# hashmod: 通过 moduels 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 替换对就的 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 ]
```

EMR 接入
Flink 接入

最近更新时间：2021-04-02 09:25:15

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。云监控的 Prometheus 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

1. 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。
2. 在 Prometheus 实例对应地域及私有网络 VPC 下，创建腾讯云容器服务 托管版集群。

操作步骤

产品接入

获取 PushGateway 访问配置

1. 前往【弹性 MapReduce】>【选择对应的“实例”】>【基本信息】>【实例信息】页面，获取 Pushgateway 地址和 Token。

2. 在 账号信息 页面获取 APPID。

修改 Flink 配置

1. 进入【弹性 MapReduce】>【选择对应的“实例”】>【集群服务】页面。
2. 找到【Flink】配置项，在右侧选择【操作】>【配置管理】，进入配置管理页面。
3. 在页面右侧单击【新增配置项】，依次添加以下配置。

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	–
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.host	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.port	-1	整数	push gateway 服务端口	控制台上 prometheus 实例的服务端口
metrics.reporter.promgateway.needBasicAuth	false	布尔	push gateway 服务是否需要认证	设置为 true，prometheus 托管服务的 push gateway 需要认证
metrics.reporter.promgateway.user	无	字符串	认证的用户名	用户的 APPID

配置名	默认	类型	描述	建议
metrics.reporter.promgateway.password	无	字符串	认证的密码	控制台上 prometheus 实例的访问 Token
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: climatePredict
metrics.reporter.promgateway.randomJobNameSuffix:true
metrics.reporter.promgateway.interval: 60 SECONDS
metrics.reporter.promgateway.groupingKey:instance_id=emr-xxxx
metrics.reporter.promgateway.host: 172.xx.xx.xx
metrics.reporter.promgateway.port: 9090
metrics.reporter.promgateway.needBasicAuth: true
metrics.reporter.promgateway.user: appid
metrics.reporter.promgateway.password: token
```

安装 Flink PushGateway 插件

官方包中的 push gateway 插件目前还不支持配置认证信息，但是托管服务需要认证才允许写入，建议使用我们提供的 jar 包。我们也向 flink 官方提交了支持认证的 PR。

1. 为防止类冲突，如果已经使用 Flink 官方插件，需要先执行以下命令删除官方插件。

```
cd /usr/local/service/flink/lib
rm flink-metrics-prometheus*jar
```

2. 在【 弹性 MapReduce 控制台 】>【 选择对应的“实例” 】>【 集群资源 】>【 资源管理 】>【 Master 】页面，查看 Master 节点。

3. 单击实例 ID 跳转至 CVM 控制台，登录 CVM 执行以下命令安装插件。

```
cd /usr/local/service/flink/lib
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/flink/flink-metrics-prometheus_2.11-auth.jar -O flink-metrics-promet
heus_2.11-auth.jar
```

验证

1. 在 Master 节点上执行 `flink run` 命令提交新任务，查看任务日志。

```
grep metrics /usr/local/service/flink/log/flink-hadoop-client-*.log
```

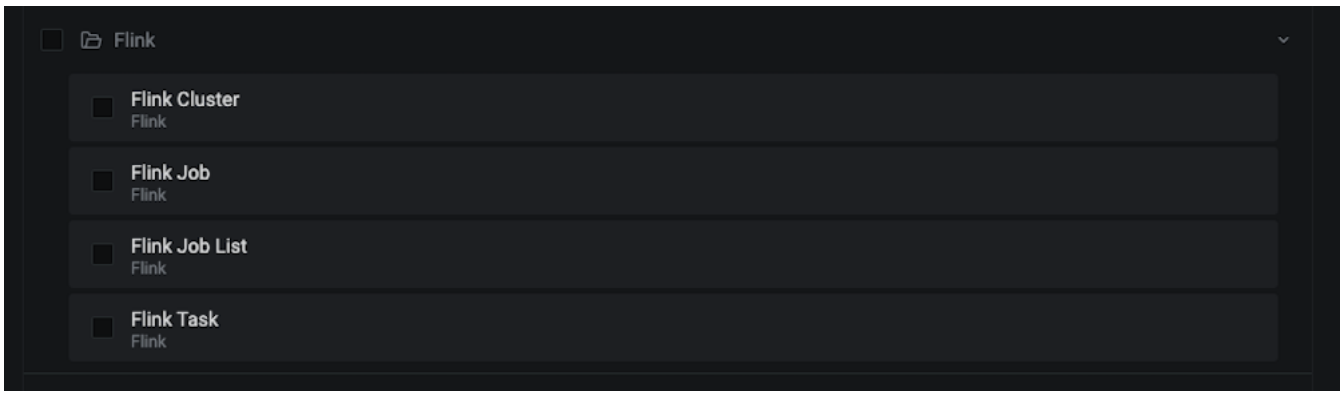
2. 日志中包含下图内容，表示配置加载成功：

```
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.class, org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporter
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.groupingKey, instance_id=emr-
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.host, 1
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.interval, 60 SECONDS
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.jobName,
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.needBasicAuth, true
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.password, ******
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.port, 9090
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.randomJobNameSuffix, true
2020-12-11 16:09:04,116 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configuration property: me
trics.reporter.promgateway.user,
```

> ⚠ 注意：
> 集群中已经提交的任务，由于使用的是旧配置文件，因此不会上报 metrics。

查看监控

1. 在对应 Prometheus 实例 >【集成中心】中找到 `Flink` 监控，安装对应的 Grafana Dashboard 即可开启 Flink 监控大盘。
2. 进入 Grafana，单击【 】展开 Flink 监控面板。

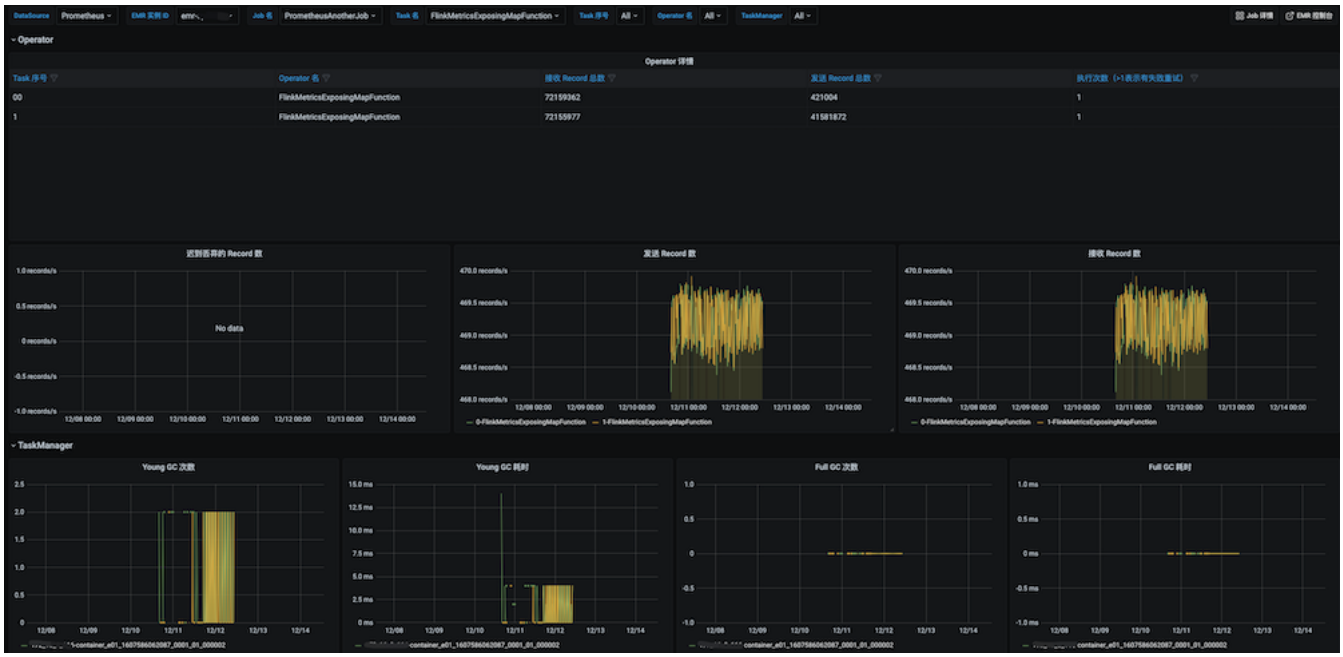


3. 单击【Flink Job List】查看监控。

4. 单击表格中的【Job 名】或【Job ID 列值】，查看 Job 监控详情。

5. 单击右上角的【Flink 集群】，查看 Flink 集群监控。

6. 单击表格中的【Task 名列值】，查看 Task 监控详情。



告警接入

1. 登录 云监控 Prometheus 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。

Java 应用接入
Spring Boot 接入

最近更新时间：2021-04-02 09:18:56

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。腾讯云 Prometheus 托管服务基于 Spring Actuator 机制采集 JVM 等数据，结合配套提供的 Grafana Dashboard 可以方便的监控 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，介绍如何通过托管 Prometheus 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。
- 使用私有镜像仓库管理应用镜像。
- 应用基于 Spring Boot 框架进行开发。

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：**
>
> 1. 访问 prometheus metrics 的地址和2.0不一样，1.5默认的是/prometues，即http://localhost:8080/prometheus。
> 2. 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。
> 3. 若项目中用 bootstrap.yml 来配置参数，在 bootstrap.yml 中需改 management 不启作用，需要在 application.yml 中修改，原因： spring boot 启动加载顺序有关。
> 4. metric common tag 不能通过 yml 来添加，只有通过代码加一个 bean 的方式添加，详细信息可参见 spring boot 1.5 接入。

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。

```
management:
endpoints:
web:
exposure:
include: prometheus # 打开 Prometheus 的 Web 访问 Path
metrics:
# 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
# 在 Prometheus 中添加特别的 Labels
tags:
# 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
application: spring-boot-mvc-demo
```

`

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 镜像仓库基本教程 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:8-jdk
WORKDIR /spring-boot-demo
ADD target/spring-boot-demo-*.jar /spring-boot-demo/spring-boot-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：

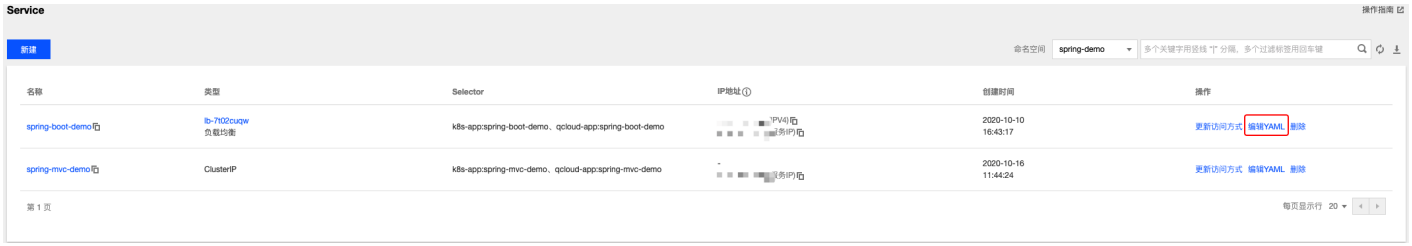
```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/prom_spring_demo/spring-boot-demo:latest
docker push ccr.ccs.tencentyun.com/prom_spring_demo/spring-boot-demo:latest
```

步骤3：应用部署

1. 登录 容器服务控制台，选择需要部署的容器集群。
2. 单击【工作负载】>【Deployment】，进入 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。


工作负载名
spring-mvc-demo
最长40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
标签
k8s-app
=
spring-mvc-demo
新增变量
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
命名空间
spring-demo
类型
Deployment（可扩展的部署Pod）
DaemonSet（在每个主机上运行Pod）
StatefulSet（有状态集的运行Pod）
CronJob（按照Cron的计划定时运行）
Job（单次任务）
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
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
ccr.ccs.tencentyun.com/prom_sp
选择镜像
镜像版本（Tag）
不填默认为 latest
选择镜像版本
镜像拉取策略
Always
IfNotPresent
Never
若不设置镜像拉取策略，当镜像版本为空或:latest时，使用Always策略，否则使用IfNotPresent策略
CPU/内存限制
CPU限制
内存限制

3. 为对应的 Service 添加 K8S Labels，如果使用命令方式新建，可以将 Labels 直接加上。这里介绍在容器控制台调整配置，选择需要调整的容器集群。
单击【服务与路由】>【Service】，进入 Service 管理页面，选择对应的命名空间来调整 Service Yaml 配置，如下图：



配置示例如下：

```
apiVersion: v1
kind: Service
metadata:
labels: # 可以根据实际情况添加对应的 labels
k8sapp: spring-mvc-demo
name: spring-mvc-demo
namespace: spring-demo
spec:
ports:
- name: 8080-8080-tcp # ServiceMonitor 抓取任务中 port 对应的值
port: 8080
protocol: TCP
targetPort: 8080
selector:
k8s-app: spring-mvc-demo
qcloud-app: spring-mvc-demo
sessionAffinity: None
type: ClusterIP
status:
loadBalancer: {}
```

步骤4：添加采取任务

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击集成容器服务列表中的【集群 ID】，进入到容器服务集成管理页面。

3. 通过服务发现添加 Service Monitor，目前支持基于 Labels 发现对应的目标实例地址，所以可以对一些服务添加特定的 K8S Labels，配置之后在 Labels 下的服务都将被 Prometheus 服务自动识别出来，不需要再为每个服务一一添加采取任务。以该例子介绍，配置信息如下：

> 说明：
> 这里需要注意的是 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: spring-mvc-demo # 填写一个唯一名称
namespace: cm-prometheus # namespace固定，不要修改
spec:
endpoints:
- interval: 30s
port: 8080-8080-tcp # 填写service yaml中Prometheus Exporter对应的Port的Name
path: /actuator/prometheus # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
namespaceSelector: # 选择要监控service所在的namespace
matchNames:
- spring-demo
selector: # 填写要监控service的Label值，以定位目标service
matchLabels:
k8sapp: spring-mvc-demo
```

步骤5：查看监控

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)
P99 P95 P50
Throughput(每秒请求数)
now now - 1d now - 7d
Error(失败数)
now
成功率
now
应用层
2XX
2XX
3XX/4XX/5XX
5XX
日志级别
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


Uptime
1.1 week
Start time
2020-10-13 09:53:48
Heap used
11.58%
Non-Heap used
5.64%
GC
GC 次数
0.040 ops
0.030 ops
0.020 ops
0.010 ops
0 ops
18:50
19:00
19:10
19:20
19:30
19:40
end of major GC (Allocation Failure)
end of minor GC (Allocation Failure)
GC 耗时
4 ms
3 ms
2 ms
1 ms
0 ns
avg end of major GC (Allocation Failure)
avg end of minor GC (Allocation Failure)
max end of major GC (Allocation Failure)
max end of minor GC (Allocation Failure)
Allocated/Promoted
500 kB/s
400 kB/s
300 kB/s
200 kB/s
100 kB/s
0 B/s
allocated
promoted
JVM Memory
JVM Heap
286 MiB
191 MiB
95 MiB
0 B
used Max: 31.4 MiB Current: 28.7 MiB
committed Max: 41.4 MiB Current: 41.4 MiB
max Max: 247.5 MiB Current: 247.5 MiB
JVM Non-Heap
1.4 GiB
954 MiB
477 MiB
0 B
used Max: 71 MiB Current: 71 MiB
committed Max: 75 MiB Current: 75 MiB
max Max: 1.234 GiB Current: 1.234 GiB
JVM Total
1.9 GiB
1.4 GiB
954 MiB
477 MiB
0 B
used Max: 103 MiB Current: 100 MiB
committed Max: 117 MiB Current: 117 MiB
max Max: 1.476 GiB Current: 1.476 GiB
JVM Misc
CPU Usage
100.0%
80.0%
60.0%
40.0%
20.0%
0%
Load
1.5
1.0
0.5
Threads
25
20
15
10
5
Thread States
15
10
5
0

JVM 接入

最近更新时间：2021-04-02 09:29:47

操作场景

在使用 Java 作为开发语言的时候，需要监控 JVM 的性能。腾讯云 Prometheus 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。

本文以如何在容器服务上部署普通 Java 应用为例，介绍如何通过托管 Prometheus 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。
- 使用私有镜像仓库管理应用镜像。

操作步骤

> 说明：
> Java 作为主流的开发语言其生态较为完善，其中 micrometer 作为指标打点 SDK 已经被广泛运行，本文以 micrometer 为例介绍如何监控 JVM。

修改应用的依赖及配置

步骤1：修改 pom 依赖

在 `pom.xml` 文件中添加相关的 Maven 依赖项，试情况调整相应的版本，示例如下：

```
<dependency>
<groupId>io.prometheus</groupId>
<artifactId>simpleclient</artifactId>
<version>0.9.0</version>
</dependency>
<dependency>
<groupId>io.micrometer</groupId>
<artifactId>micrometer-registry-prometheus</artifactId>
<version>1.1.7</version>
</dependency>
```

步骤2：修改代码

在项目启动时，添加相应的监控配置，同时 micrometer 也提供了部分常用的监控数据采集，具体在 `io.micrometer.core.instrument.binder` 包下，可以按实际情况添加。示例如下：

```
public class Application {
// 作为全局变量，可以在自定义监控中使用
public static final PrometheusMeterRegistry registry = new PrometheusMeterRegistry(PrometheusConfig.DEFAULT);
static {
// 添加 Prometheus 全局 Label，建议加一上对应的应用名
registry.config().commonTags("application", "java-demo");
}
public static void main(String[] args) throws Exception {
// 添加 JVM 监控
new ClassLoaderMetrics().bindTo(registry);
new JvmMemoryMetrics().bindTo(registry);
new JvmGcMetrics().bindTo(registry);
```

```
new ProcessorMetrics().bindTo(registry);
new JvmThreadMetrics().bindTo(registry);
new UptimeMetrics().bindTo(registry);
new FileDescriptorMetrics().bindTo(registry);
System.gc(); // Test GC
try {
// 暴露 Prometheus HTTP 服务，如果已经有，可以使用已有的 HTTP Server
HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
server.createContext("/metrics", httpExchange -> {
String response = registry.scrape();
httpExchange.sendResponseHeaders(200, response.getBytes().length);
try (OutputStream os = httpExchange.getResponseBody()) {
os.write(response.getBytes());
}
});
new Thread(server::start).start();
} catch (IOException e) {
throw new RuntimeException(e);
}
}
}
```

> **说明：**
> 由于 JVM GC Pause 监控是通过 GarbageCollector Notification 机制实现，因此只有发生 GC 之后才有监控数据。上述示例为了测试更直观，主动调用了 `System.gc()` 。

步骤3：本地验证

本地启动之后，可以通过 `http://localhost:8080/metrics` 访问到 Prometheus 协议的指标数据。

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，可以参见容器镜像服务 快速入门 文档进行配置。若已配置请执行下一步。

步骤2：打包及上传镜像

1. 在项目根目录下添加 `Dockerfile` ，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. 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` / 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]
```

示例如下：

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

步骤3：应用部署

1. 登录 容器服务控制台，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 来进行部署服务，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: spring-demo
spec:
replicas: 1
selector:
matchLabels:
k8s-app: java-demo
template:
metadata:
labels:
k8s-app: java-demo
spec:
containers:
- image: ccr.ccs.tencentyun.com/prom_spring_demo/java-demo
imagePullPolicy: Always
name: java-demo
ports:
- containerPort: 8080
name: metric-port
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
dnsPolicy: ClusterFirst
imagePullSecrets:
- name: qcloudregistrykey
restartPolicy: Always
schedulerName: default-scheduler
terminationGracePeriodSeconds: 30
```

步骤4：添加采取任务

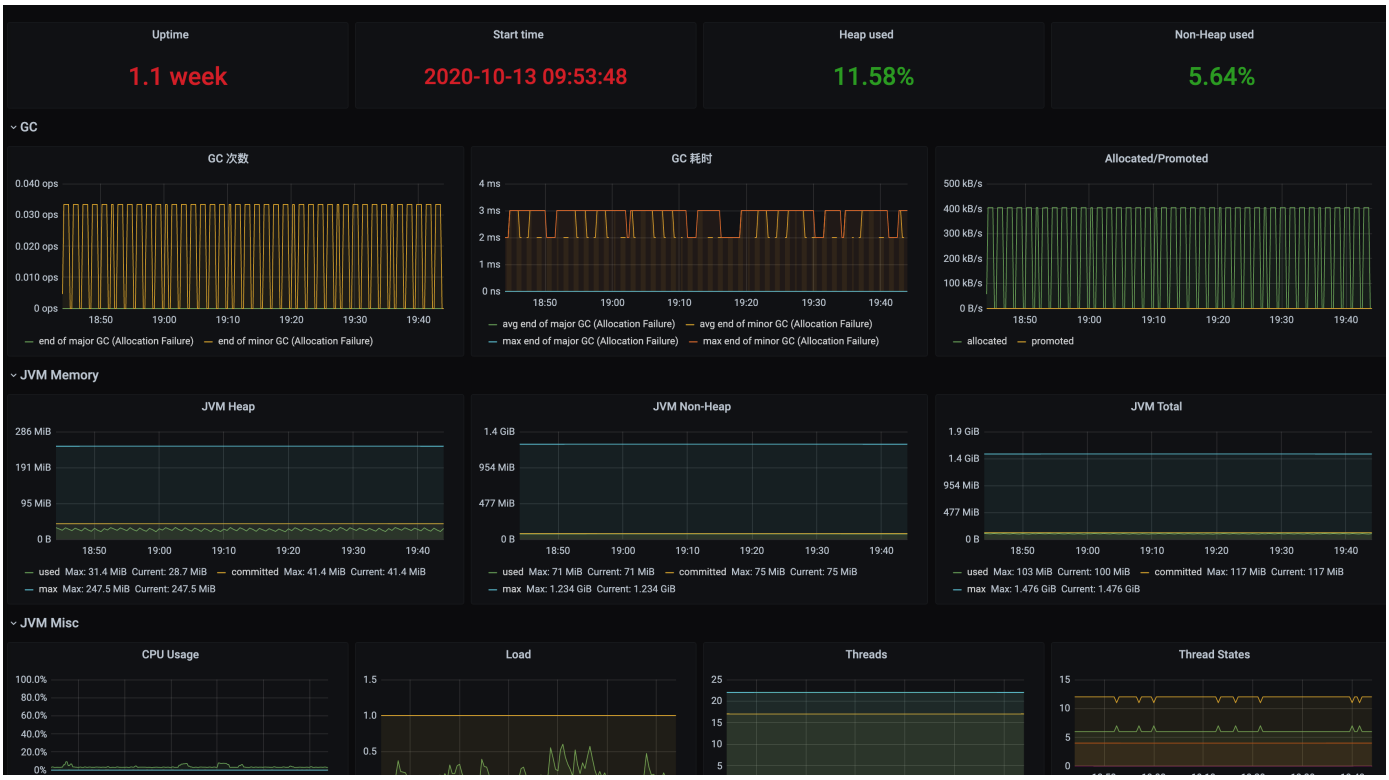
1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表点击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 Pod Monitor 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
name: java-demo
namespace: cm-prometheus
spec:
namespaceSelector:
matchNames:
- java-demo
```

```
podMetricsEndpoints:
- interval: 30s
path: /metrics
port: metric-port
selector:
matchLabels:
k8s-app: java-demo
```

步骤5：查看监控

1. 在对应 Prometheus 实例 >【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/Manage/Application 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。



Golang 应用接入
Golang 接入

最近更新时间：2021-02-26 17:54:45

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，并使用腾讯云监控 Prometheus 托管服务来采集指标展示数据等。

> 说明：
> Golang Client API 相关的文档详见 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
```

使用腾讯云 Prometheus 监控服务

上述我们提供了两个示例展示如何使用 Prometheus Golang 库来暴露应用的指标数据，但暴露的监控指标数据为文本类型，需要搭建维护额外的 Prometheus 服务来抓取指标，可能还需要额外的 Grafana 来对数据进行可视化展示。

通过使用 Prometheus 托管服务可以直接省去如上步骤，只需简单的单击操作即可使用。详情请参见 快速使用指南。

打包部署应用

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。

5. 可以通过 容器服务控制台 或者直接使用 kubectl 将这些资源定义提交给 Kubernetes，然后等待创建成功。

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 托管服务发现并采集监控指标：

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表单击【集群 ID】，进入到容器服务集成管理页面。
3. 通过服务发现添加 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一一添加采取任务，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。

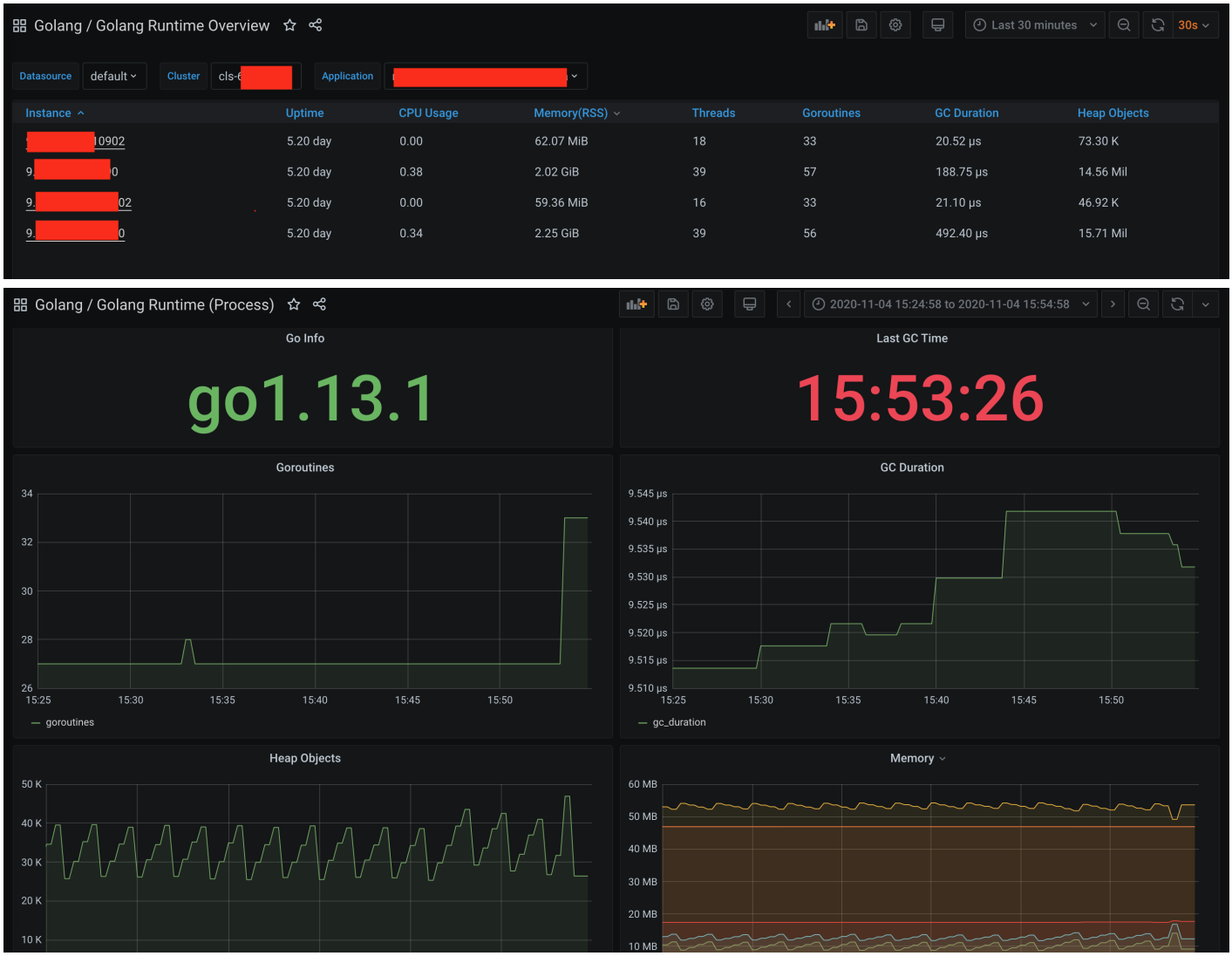
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
name: go-demo # 填写一个唯一名称
namespace: cm-prometheus # namespace固定，不要修改
spec:
endpoints:
- interval: 30s
# 填写service yaml中Prometheus Exporter对应的Port的Name
port: 2112
# 填写Prometheus Exporter对应的Path的值，不填默认/metrics
path: /metrics
relabelings: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

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. 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 托管服务，以及如何使用内置的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，Histgram 以及 Summary 类型的指标， 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：2021-04-02 09:26:19

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，云监控 Prometheus 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：
> 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 托管版集群，并为集群创建 命名空间。
- 在【云监控 Prometheus 控制台】>【选择“对应的 Prometheus 实例”】>【集成容器服务】中找到对应容器集群完成集成操作，详情请参见 Agent 管理。

操作步骤

Exporter 部署

1. 登录 容器服务 控制台。
2. 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。
3. 执行以下 使用 Secret 管理 ElasticSearch 连接串 > 部署 Kafka Exporter > 验证 步骤完成 Exporter 部署。

使用 Secret 管理 ElasticSearch 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 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
```

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

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
name: es-exporter
```

```
namespace: es-demo
spec:
replicas: 1
selector:
matchLabels:
k8s-app: es-exporter
template:
metadata:
labels:
k8s-app: es-exporter
spec:
containers:
- env:
- name: ES_URI
valueFrom:
secretKeyRef:
name: es-secret-test
key: esURI
- name: ES_ALL
value: "true"
image: bitnami/elasticsearch-exporter:latest
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。

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：

```
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
```

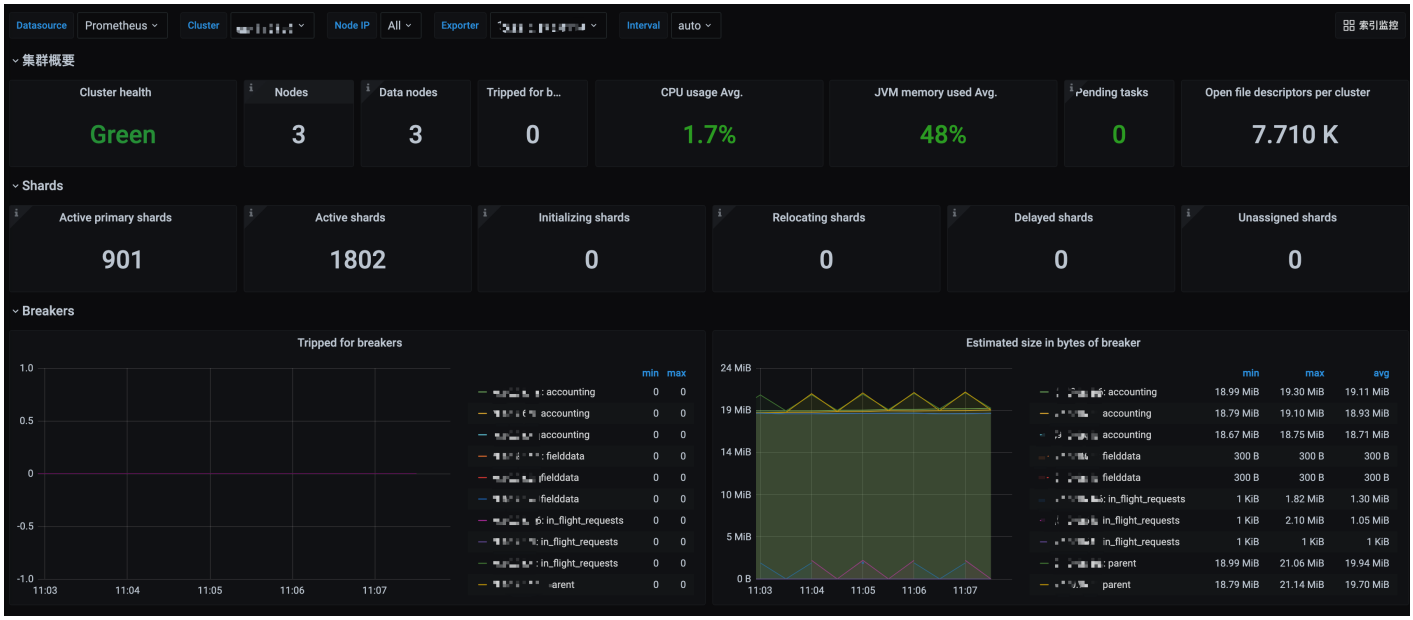
添加采取任务

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表点击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 `Pod Monitor` 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
namespace: cm-prometheus
spec:
namespaceSelector:
matchNames:
- es-demo
podMetricsEndpoints:
- interval: 30s
path: /metrics
port: metric-port
selector:
matchLabels:
k8s-app: es-exporter
```

查看监控

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



告警以及接入

1. 登录 云监控 Prometheus 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。

Kafka Exporter 接入

最近更新时间：2021-04-02 09:27:16

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，云监控 Prometheus 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：
> 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 托管版集群，并为集群创建 命名空间。
- 在【云监控 Prometheus 控制台】>【选择“对应的 Prometheus 实例”】>【集成容器服务】中找到对应容器集群完成集成操作，详情请参见 Agent 管理。

操作步骤

Exporter 部署

1. 登录 容器服务 控制台。
2. 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。
3. 在左侧菜单中选择【工作负载】>【Deployment】，进入 Deployment 页面。
4. 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
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
namespace: kafka-demo
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
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
spec:
containers:
- args:
- --kafka.server=x.x.x.x:9092 # 对应 Kafka 实例的地址信息
image: danielqsj/kafka-exporter:latest
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。

添加采取任务

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表点击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 Pod Monitor 来定义 Prometheus 抓取任务，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 # 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: '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
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

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

2. 单击【集成中心】，进入集成中心页面。找到 kafka 监控，安装对应的 Grafana Dashboard 即可开启 kafaka 监控大盘，查看实例相关的监控数据，如下图所示：

告警以及接入

1. 登录 云监控 Prometheus 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。

MongoDB Exporter 接入

最近更新时间：2021-04-02 09:28:09

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，云监控 Prometheus 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：
> 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 Proemtheus 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。
- 在【云监控 Prometheus 控制台】>【选择“对应的 Prometheus 实例”】>【集成容器服务】中找到对应容器集群完成集成操作，详情请参见 Agent 管理。

操作步骤

Exporter 部署

1. 登录 容器服务 控制台。
2. 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。
3. 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
```

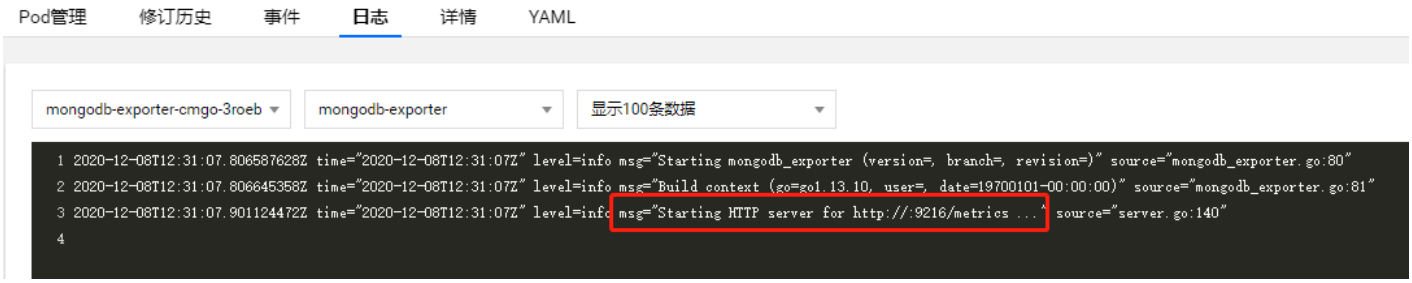
```
template:
metadata:
labels:
k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
spec:
containers:
- args:
- --collect.database # 启用采集 Database metrics
- --collect.collection # 启用采集 Collection metrics
- --collect.topmetrics # 启用采集 table top metrics
- --collect.indexusage # 启用采集 per index usage stats
- --collect.connpoolstats # 启动采集 MongoDB connpoolstats
env:
- name: MONGODB_URI
valueFrom:
secretKeyRef:
name: mongodb-secret-test
key: datasource
image: ssheehy/mongodb-exporter
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

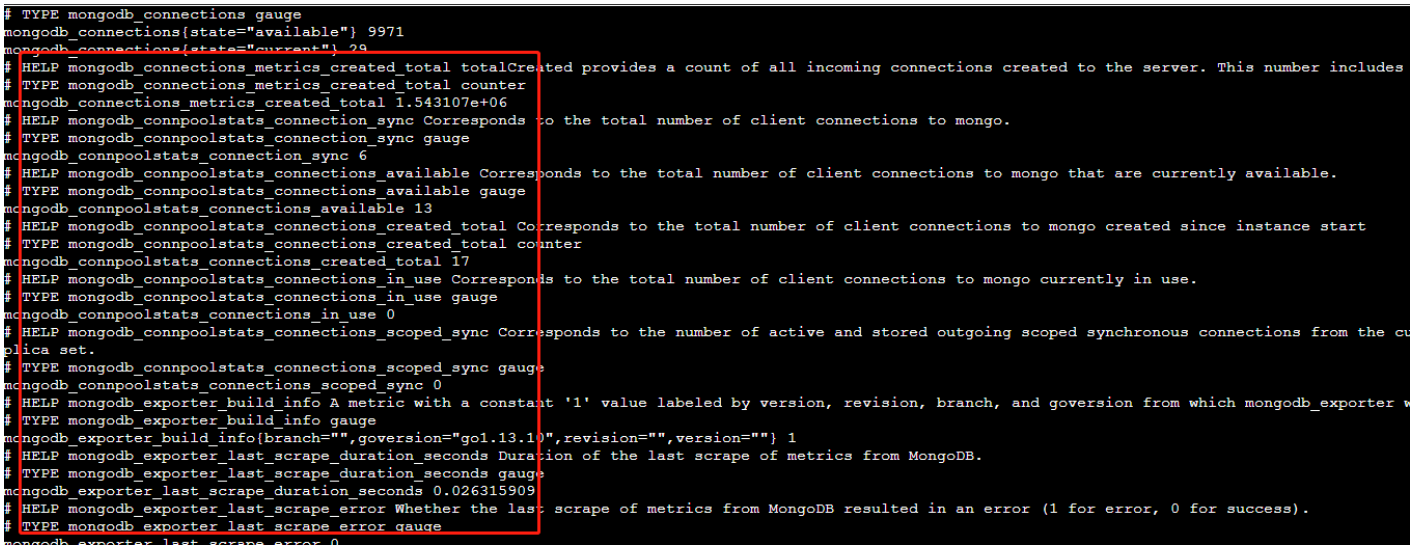
1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击【日志】页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：



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：



添加采集任务

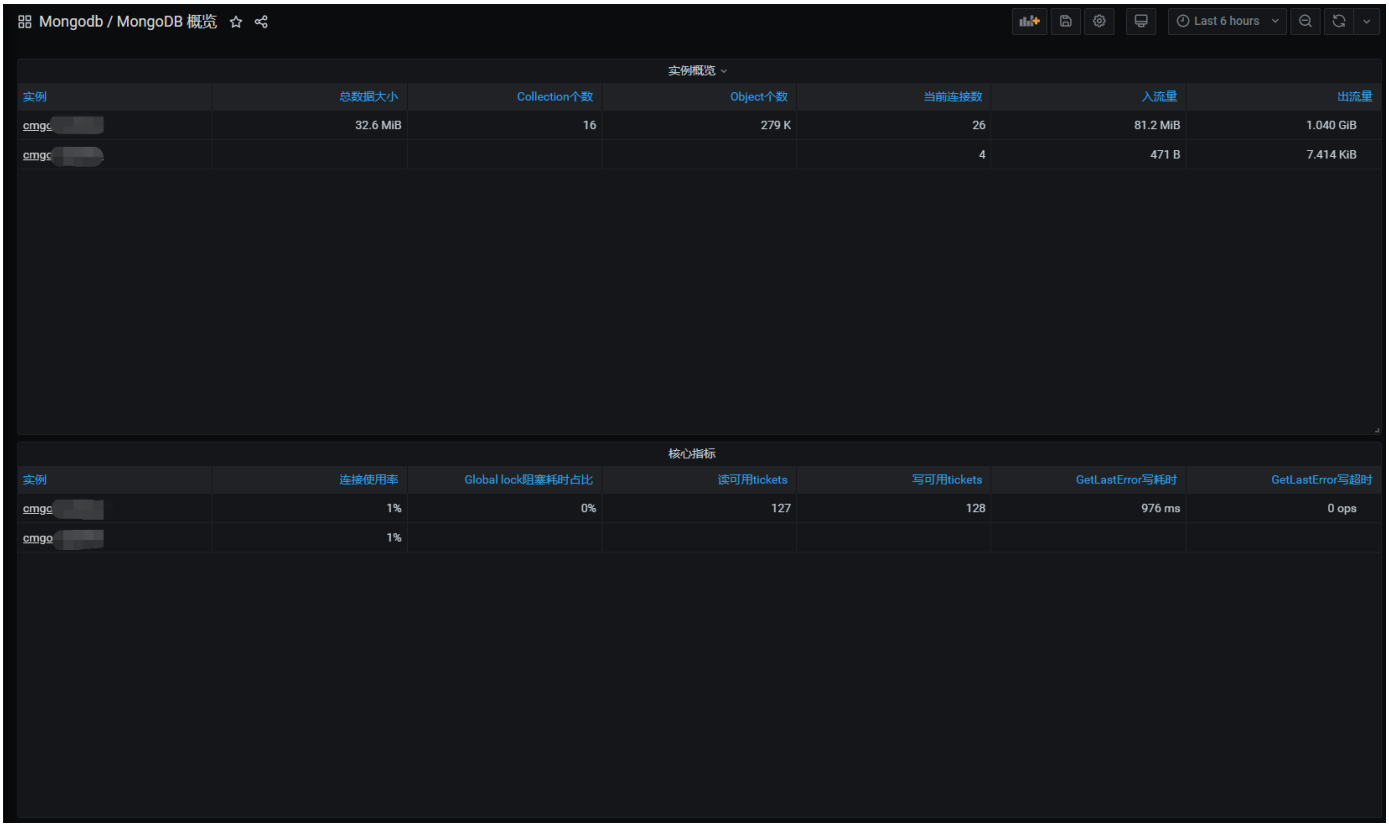
1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表点击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 Pod Monitor 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
name: mongodb-exporter # 填写一个唯一名称
namespace: cm-prometheus # 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

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击【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**：以实例的纬度查看实例状态，例如文档个数、连接使用率、读写耗时等，可点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：
> 每个图表可以点击左侧的【！】进行查看说明

告警以及接入

1. 登录 云监控 Prometheus 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：2021-04-02 09:29:00

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，云监控 Prometheus 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：
> 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 Proemtheus 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。
- 在【云监控 Prometheus 控制台】>【选择“对应的 Prometheus 实例”】>【集成容器服务】中找到对应容器集群完成集成操作，详情请参见 Agent 管理。

操作步骤

Exporter 部署

1. 登录 容器服务 控制台。
2. 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。
3. 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 `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（请直接复制下面的内容，根据**实际业务调**整相**应**的参数）：

```
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"
- "--log.level=debug"
env:
- name: DATA_SOURCE_USER
valueFrom:
secretKeyRef:
name: postgres-test
key: username
- name: DATA_SOURCE_PASS
valueFrom:
secretKeyRef:
name: postgres-test
key: password
- name: DATA_SOURCE_URI
value: "x.x.x.x:5432/postgres?sslmode=disable"
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：
- disable	不使用 SSL
- require	总是使用（跳过验证）
- 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）
- 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，参考 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`，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
```

添加采取任务

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

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表点击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 `Pod Monitor` 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
namespace: cm-prometheus
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
```

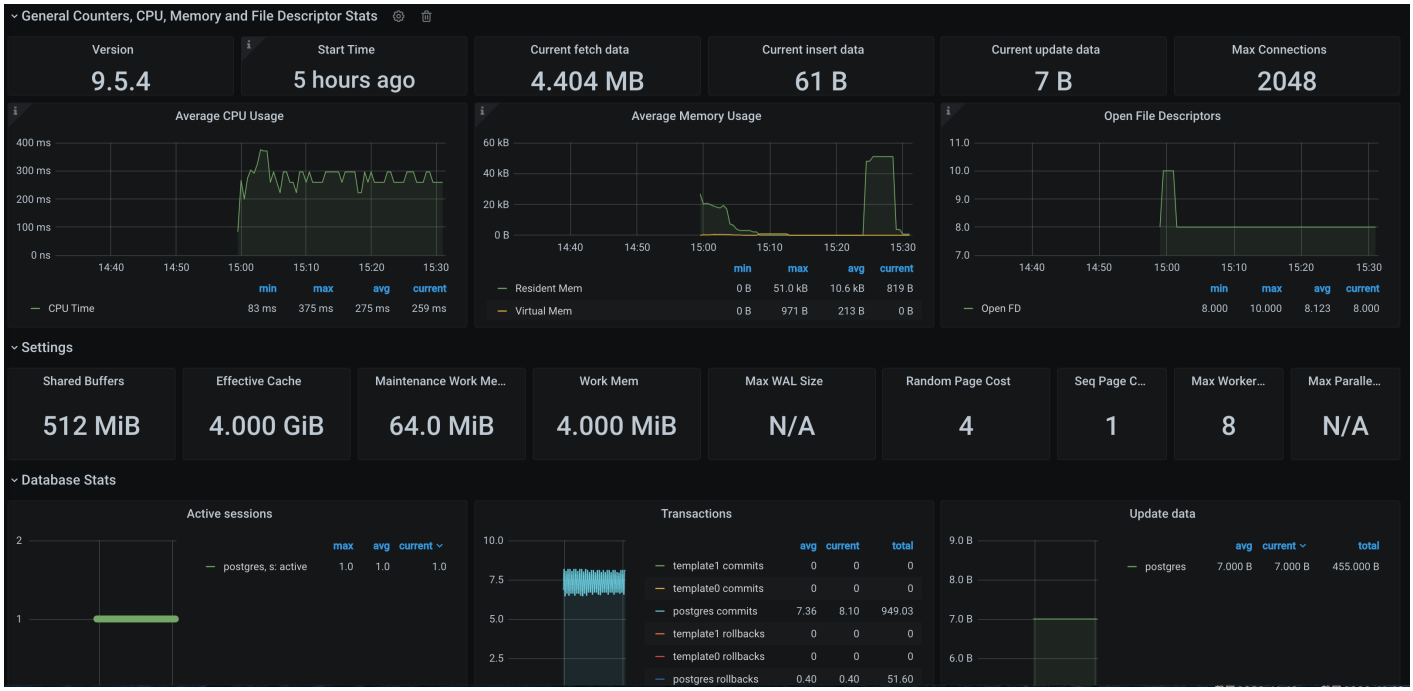
> 说明：
> 更多高阶用法请参见 ServiceMonitor 和 PodMonitor。

Grafana 大屏可视化

> 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。

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. 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。



告警以及接入

1. 登录 云监控 Prometheus 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。

> 说明：
> 后续云监控将提供更多 PostgreSQL 相关的告警模板。

Nginx Prometheus Exporter 接入

最近更新时间：2021-04-23 17:31:40

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。

操作步骤

使用 Docker 容器运行 Exporter

方式1：使用 nginx-prometheus-exporter 通过 Docker 容器快速部署 Exporter。执行 Docker 命令如下：

```
$ docker run -p 9113:9113 nginx/nginx-prometheus-exporter:0.8.0 -nginx.scrape-uri http://<nginx>:8080/stub_status
```

方式2：使用 nginx-prometheus-exporter 镜像将服务部署在腾讯云 容器服务 TKE 中，通过托管 Prometheus 的监控自发现 CRD PodMonitor 或者 ServiceMonitor 来采集监控数据。

使用二进制程序运行 Exporter

下载安装

1. 根据实际运行环境在社区中下载相应的 Nginx Prometheus Exporter 。
2. 安装 Nginx Prometheus Exporter。

开启 NGINX stub_status 功能

1. 开源 Nginx 提供一个简单页面用于展示状态数据，该页面由 tub_status 模块提供。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 tub_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 \
… \
--with-http_stub_status_module
make
sudo make install
```

2. 确认 stub_status 模块启用之后，修改 Nginx 的配置文件指定 status 页面的 URL。示例如下：

```
server {
location /nginx_status {
stub_status;
access_log off;
allow 127.0.0.1;
deny all;
}
}
```

3. 检查并重新加载 nginx 的配置使其生效。

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

4. 完成上述步之后，可以通过配置的 URL 查看 Nginx 的指标：

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

运行 NGINX Prometheus Exporter

执行以下命令启动 NGINX Prometheus Exporter：

```
$ nginx-prometheus-exporter -nginx.scrape-uri http://<nginx>:8080/nginx_status
```

上报指标

- `nginxexporter_build_info` -- exporter 编译信息。
- 所有的 stub_status 指标。
- `nginx_up` -- 展示上次抓取的状态：1表示抓取成功， 0表示抓取失败。

配置 Prometheus 的抓取 Job

1. Nginx Prometheus Exporter 正常运行后，修改 Nginx 的配置文件，将 Job 添加到 Prometheus 的抓取任务中。

```
...
- job_name: 'nginx_exporter'
static_configs:
- targets: ['your_exporter:port']
```

2. 通常情况下，Exporter 和 Nginx 并非共同运行，所以数据上报的 `instance` 并不能真实描述是哪个实例，为了方便数据的检索和观察，我们可以修改 `instance` 标签，使用真实的 IP 进行替换以便更加直观。示例如下：

```
...
- job_name: 'mysqld_exporter'
static_configs:
- targets: ['your_exporter:port']
relabel_configs:
- source_labels: [__address__]
regex: '.*'
target_label: instance
replacement: '10.0.0.1:80'
```

启用数据库监控大盘

腾讯云 Prometheus 托管服务在 Grafana 中提供预先配置的 Nginx Exporter Dashboard，您可以根据以下操作步骤查看 Nginx 监控数据。

1. 登录 云监控 Prometheus 控制台。

2. 单击对应实例 ID 右侧的【 】，即可查看数据。

Redis Exporter 接入

最近更新时间：2021-04-02 09:22:39

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。云监控 Prometheus 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云监控 Prometheus 监控 Redis。

> 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 Proemtheus 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【云监控 Prometheus 控制台】>【选择“对应的 Prometheus 实例”】>【集成容器服务】中找到对应容器集群完成集成操作，详情请参见 Agent 管理。

操作步骤

Exporter 部署

1. 登录 容器服务 控制台。
2. 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。
3. 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
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
namespace: redis-test
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
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
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/redis-operator/redis-exporter:1.12.0
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】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 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
```

添加采取任务

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表单击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 Pod Monitor 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 # 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

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击【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：

告警以及接入

1. 登录 云监控 Prometheus 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。

MySQL Exporter 接入

最近更新时间：2021-04-02 09:23:52

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，云监控 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：
> 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

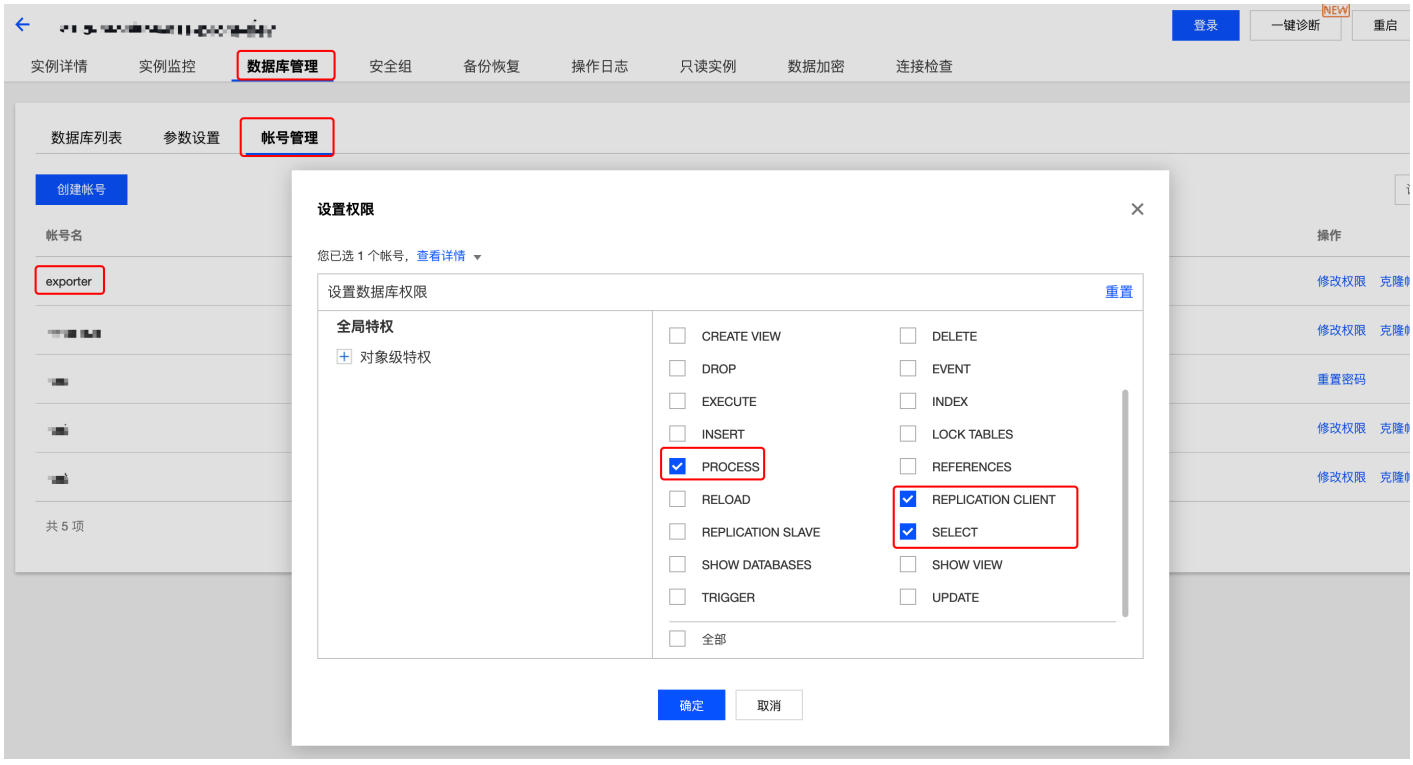
- 在 Proemtheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间。
- 在【云监控 Prometheus 控制台】>【选择“对应的 Prometheus 实例”】>【集成容器服务】中找到对应容器集群完成集成操作，详情请参见 Agent 管理。

操作步骤

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。帐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】>【帐号管理】，进入帐号管理页面。
4. 单击帐号右侧操作项下的【修改权限】，修改对应权限。示例如下图所示：



您可以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。

Exporter 部署

1. 登录 容器服务 控制台。
2. 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。
3. 执行以下 使用 Secret 管理 MySQL 连接串 > 部署 MySQL Exporter > 验证 步骤完成 Exporter 部署。

使用 Secret 管理 MySQL 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 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
```

部署 MySQL Exporter

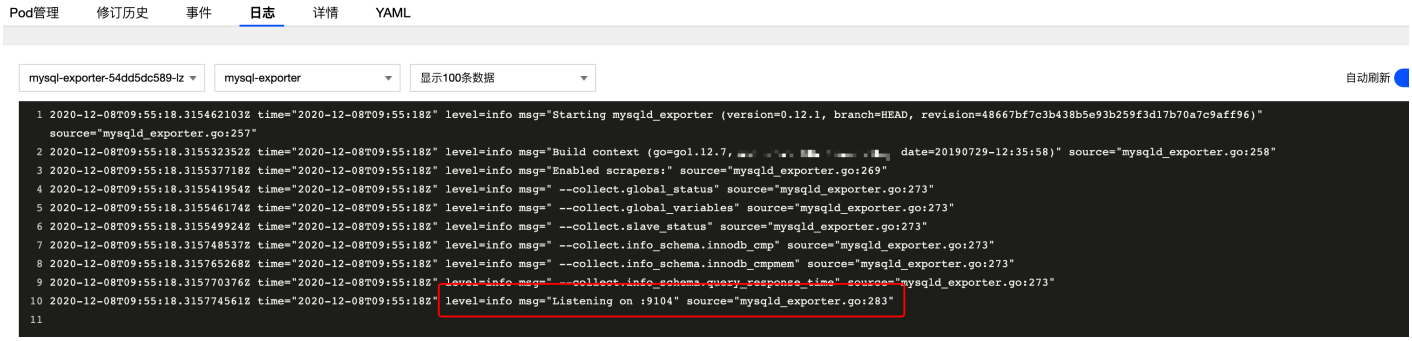
在 Deployment 管理页面，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
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
key: datasource
image: ccr.ccs.tencentyun.com/k8s-comm/mysqld-exporter:0.12.1
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
```

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 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：

```
curl localhost:9104/metrics
```

执行结果如下图所示：

```
mysql_info_schema_innodb_cmpmem_pages_used_total{buffer_pool="0",page_size="4096"} 0
mysql_info_schema_innodb_cmpmem_pages_used_total{buffer_pool="0",page_size="8192"} 0
# HELP mysql_info_schema_innodb_cmpmem_relocation_ops_total Number of times a block of the size PAGE_SIZE has been
# TYPE mysql_info_schema_innodb_cmpmem_relocation_ops_total counter
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="1024"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="16384"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="2048"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="4096"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="8192"} 0
# HELP mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total Total time in seconds spent in relocating bloc
# TYPE mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total counter
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="1024"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="16384"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="2048"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="4096"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="8192"} 0
# HELP mysql_up Whether the MySQL server is up.
# TYPE mysql_up gauge
mysql_up 1
# HELP mysql_version_info MySQL version and distribution.
# TYPE mysql_version_info gauge
```

添加采取任务

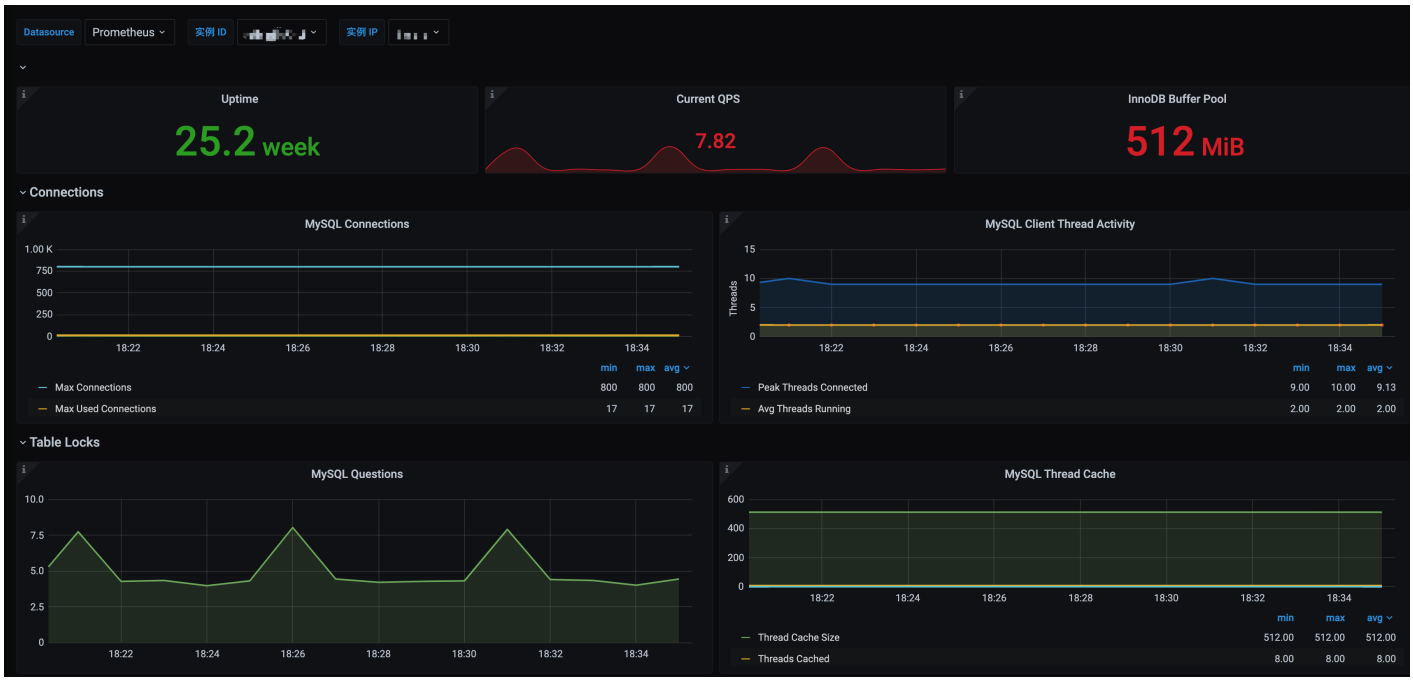
1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过集成容器服务列表点击【集群 ID】进入到容器服务集成管理页面。
3. 通过服务发现添加 Pod Monitor 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
```

```
metadata:
name: mysql-exporter # 填写一个唯一名称
namespace: cm-prometheus # 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
```

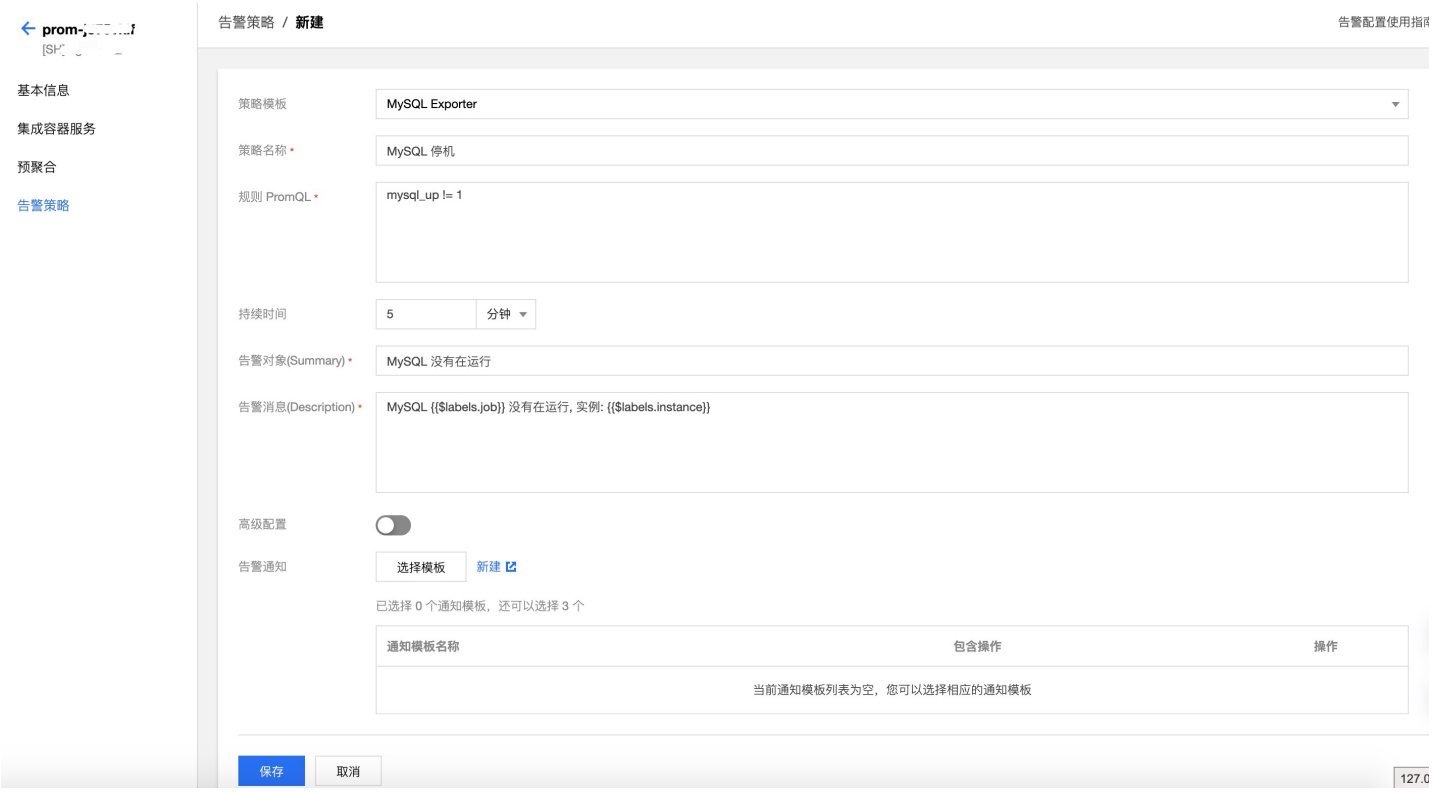
查看监控

1. 登录 云监控 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击【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：



告警以及接入

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。



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
collect.info_schema.innodb_cmpmem	5.5	从 information_schema.innodb_cmpmem 中采集 InnoDB buffer p 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
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。

名称	MySQL 版本	描述
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_dig 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_dig summed 中采集监控数据。
collect.perf_schema.eventswaits	5.5	从 performance_schema.events_waits_summary_global_by_eve 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
collect.perf_schema.file_instances	5.5	从 performance_schema.file_summary_by_instance 中采集监控
collect.perf_schema.indexiowaits	5.6	从 performance_schema.table_io_waits_summary_by_index_u 采集监控数据。
collect.perf_schema.tableiowaits	5.6	从 performance_schema.table_io_waits_summary_by_table 中 据。
collect.perf_schema.tablelocks	5.6	从 performance_schema.table_lock_waits_summary_by_table 控数据。
collect.perf_schema.replication_group_members	5.7	从 performance_schema.replication_group_members 中采集监控
collect.perf_schema.replication_group_member_stats	5.7	从 from performance_schema.replication_group_member_stat 控数据。
collect.perf_schema.replication_applier_status_by_worker	5.7	从 performance_schema.replication_applier_status_by_worke 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 --u 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。提示：不支持 Oracle MySQL。

名称	描述
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。

集成容器服务常见问题

最近更新时间：2020-12-11 17:13:45

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 是 cls-xxx，那么搜索内容为 `tke-master-security-for-cls-xxx`。
3. 单击搜索出的安全组 ID 进入编辑入站规则对话框。
4. 要编辑的规则的协议端口列应包含 TCP:60001,60002，逐条选择规则，添加端口10249,10252,10251,9100,9153。各个端口用途如下：
   - 10249 kube-proxy metrics 的端口
   - 10252 kube-controller-manager metrics 的端口
   - 10251 kube-scheduler metrics 的端口
   - 9100 node-exporter metrics 的端口
   - 9153 core-dns metrics 的端口

版权所有：腾讯云计算（北京）有限责任公司