ALIBABA CLOUD

阿里云

Prometheus监控服务

Prometheus监控公共云合集

文档版本：20220708

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.动态与公告 11
1.1. 功能发布记录 11
1.1.1. 2022年 11
1.1.2. 2021年 12
1.1.3. 2020年 15
1.2. 产品公告 15
1.2.1. 【产品公告】Prometheus监控商用通知 15
2.产品简介 16
2.1. 什么是Prometheus监控 16
2.2. 功能特性 18
2.3. 产品优势 21
2.4. 基本概念 23
2.5. 开服地域 24
2.6. 阿里云Prometheus监控开源一致性认证 26
3.产品计费 53
3.1. 按量付费 53
3.2. 查看账单 56
4.创建Prometheus实例 59
4.1. 查看Prometheus实例 59
4.2. Prometheus实例 for 容器服务 60
4.3. Prometheus实例 for 云服务 63
4.4. Prometheus实例 for VPC 65
4.5. Prometheus实例 for Kubernetes 67
4.6. Prometheus实例 for Remote Write 69
4.7. Prometheus实例 for GlobalView 76
5.集成中心 79

6.组件监控接入 85
6.1. 组件监控概述 85
6.2. 使用阿里云Prometheus监控MySQL 86
6.3. 使用阿里云Prometheus监控Redis 89
6.4. 使用阿里云Prometheus监控Elasticsearch 91
6.5. 使用阿里云Prometheus监控MongoDB 93
6.6. 使用阿里云Prometheus监控Kafka 95
6.7. 使用阿里云Prometheus监控RabbitMQ 97
6.8. 使用阿里云Prometheus监控RocketMQ 99
6.9. 使用阿里云Prometheus监控PostgreSQL 102
6.10. 使用阿里云Prometheus监控ZooKeeper 104
6.11. 使用阿里云Prometheus监控Nginx（旧版） 106
6.12. 使用阿里云Prometheus监控Nginx（新版） 110
6.13. 其他类型的组件接入 114
7.客户端接入 121
7.1. 通过阿里云Prometheus监控JVM 121
7.2. 通过阿里云Prometheus监控Go应用 125
8.云服务监控接入 134
8.1. 管理云服务监控接入 134
8.2. 接入并监控云服务（旧版） 135
9.管理大盘 136
9.1. 大盘列表 136
9.2. 重置大盘 138
10.管理Prometheus实例 140
10.1. 配置指标 140
10.2. 云服务指标 142
10.3. 云服务设置 142
10.4. 配置Prometheus监控采集规则 144

10.5. 服务发现 145
10.5.1. 管理Kubernetes集群服务发现 145
10.5.2. 管理VPC服务发现 147
10.6. 探针设置 149
10.7. 获取Remote Read、Remote Write和HTTP API地址 152
10.8. 升级组件版本 154
10.9. 编辑RecordingRule.yaml 155
10.10. 卸载Prometheus监控插件 160
11.全局配置 161
11.1. GlobalView聚合实例（旧版） 161
12.健康巡检 164
12.1. 创建自定义巡检 164
12.2. 创建ACK Service巡检 164
12.3. 创建云服务巡检 165
12.4. 管理健康巡检 166
13.报警（旧版） 168
13.1. 创建报警 168
13.2. 管理报警 169
13.3. 云服务报警配置 170
14.告警管理（新版） 172
14.1. Prometheus告警规则 172
14.2. Prometheus告警规则模板 178
14.3. 查看告警发送历史 182
14.4. 查看告警事件历史 188
14.5. 通知策略 190
14.6. 升级策略 194
14.7. 联系人管理 195
14.7.1. 联系人 195

14.7.2. 联系人组 197
14.7.3. 钉钉机器人 198
14.7.4. 企业微信机器人 199
14.7.5. 飞书机器人 200
14.7.6. 通过Webhook自定义告警通知人 202
14.7.7. 获取钉钉机器人Webhook地址 207
15.参考信息 210
15.1. 基础指标说明 210
15.2. 报警规则说明 240
15.3. Helm命令参数说明 253
15.4. 基础大盘说明 254
15.5. Helm版本说明 259
15.6. Remote Write和Remote Read地址使用说明 260
15.7. HTTP API地址使用说明 263
15.8. Agent水平伸缩（HPA）自动扩容能力说明 266
15.9. mysqld_exporter访问MySQL数据库所需的权限说明 267
16.API参考 268
16.1. Prometheus监控 268
16.1.1. AddGrafana 268
16.1.2. DeleteGrafanaResource 269
16.1.3. AddIntegration 270
16.1.4. DeleteIntegration 271
16.1.5. GetIntegrationState 273
16.1.6. GetPrometheusApiToken 274
16.1.7. ListDashboards 275
16.1.8. CreatePrometheusAlertRule 280
16.1.9. ListPrometheusAlertRules 285
16.1.10. DescribePrometheusAlertRule 289

16.1.11. UpdatePrometheusAlertRule 293
16.1.12. DeletePrometheusAlertRule 298
16.1.13. ListPrometheusAlertTemplates 299
16.1.14. ListActivatedAlerts 302
16.1.15. GetRecordingRule 307
16.1.16. AddRecordingRule 309
16.1.17. GetAuthToken 310
16.1.18. AddPrometheusGlobalViewByAliClusterIds 311
16.1.19. RemoveAliClusterIdsFromPrometheusGlobalView 313
16.1.20. AddAliClusterIdsToPrometheusGlobalView 315
16.1.21. AddPrometheusGlobalView 318
16.1.22. GetPrometheusGlobalView 321
16.1.23. ListPrometheusGlobalView 322
16.1.24. DeletePrometheusGlobalView 323
16.1.25. AppendInstancesToPrometheusGlobalView 325
16.1.26. RemoveSourcesFromPrometheusGlobalView 328
16.2. 报警 330
16.2.1. CreateDispatchRule 330
16.2.2. DescribeDispatchRule 335
16.2.3. UpdateDispatchRule 339
16.2.4. DeleteDispatchRule 344
17.最佳实践 346
17.1. 监控VPC网络下ECS实例中的Java应用 346
17.2. 监控VPC网络下ECS实例中的Node节点 353
17.3. 将阿里云Prometheus监控数据接入本地Grafana 360
17.4. 开始使用日志监控 362
17.5. 将Prometheus监控接入PagerDuty 366
17.6. 通过阿里云Prometheus自定义Grafana大盘 369

17.7. 通过ServiceMonitor创建服务发现 379
17.8. 公网Kubernetes集群接入Prometheus监控 383
17.9. 使用智能检测算子发现异常数据 386
17.10. 对于自建Kubernetes集群如何自定义Prometheus配置 387
17.11. 配置Prometheus for VPC服务发现 389
17.12. 使用阿里云Prometheus监控腾讯云产品 394
17.13. Spring Boot应用如何快速接入Prometheus监控 401
17.14. VPC网络下的SAE应用如何接入阿里云Prometheus监控 407
17.15. 如何实现集群内ServiceMonitor和PodMonitor的同步 411
18.常见问题 413
18.1. 常见问题概述 413
18.2. 计费相关 413
18.2.1. 接入ARMS Prometheus监控后，为什么会产生额外的费用？ 413
18.2.2. 如果不需要某些自定义指标，应该如何避免收费？ 413
18.3. 大盘相关 413
18.3.1. 为什么在创建Grafana大盘时，没有Kubelet和API Server的监... 414
18.3.2. 为什么Exporter对应的大盘看不到具体的指标？ 414
18.4. 报警相关 414
18.4.1. 如何禁止默认报警自动创建 414
18.4.2. 如何开启报警自动启用 415
18.5. 其他 415
18.5.1. 如何查看采集到的数据？ 415
18.5.2. 为什么获取不到CSI组件的Metric？ 416
18.5.3. Grafana、Istio和HPA等第三方系统如何集成Prometheus监控.. 416
18.5.4. 为什么ACK集群已删除，Prometheus Agent没有同步删除？ 417
18.5.5. 如何关闭对云数据库MongoDB版的监控？ 417
18.5.6. 如何关闭对实时计算Flink版的监控？ 419
18.5.7. 为什么在容器、节点、Pod中得到的内存值不一致？ 419

19.相关协议 420
19.1. Prometheus服务等级协议 420
19.2. Prometheus监控服务试用条款 420
19.3. Prometheus监控服务专家版协议 424

1.动态与公告

1.1. 功能发布记录

1.1.1. 2022年

本文为Prometheus监控2022年的版本发布记录，介绍历次发布的特性变更情况。

2022年05月

功能名称	功能概述	支持地域	版本号
Prometheus for ACK上线新版集成中心	集成中心作为Prometheus实例的入口，将容器服务、自定义服务发现、组件监控的关联数据和高频操作进行集中化展示，提供ACK集群内一站式监控配置与管理。更多信息，请参见集成中心。	请参见开服地域。	v2.8.4.1
同步ServiceMonitor和PodMonitor	阿里云Prometheus支持同步集群内的ServiceMonitor和PodMonitor。更多信息，请参见如何实现集群内ServiceMonitor和PodMonitor的同步。	请参见开服地域。	v2.8.4.1

2022年04月

功能名称	功能概述	支持地域	版本号
新增开服地域	新开服华北6（乌兰察布）地域。	请参见开服地域。	v2.8.3.4
更新全局聚合实例功能	提供多个阿里云Prometheus实例或自建Prometheus集群的虚拟聚合实例，针对这个虚拟聚合实例可以实现Prometheus指标的统一查询，统一Grafana数据源和统一告警。更多信息，请参见GlobalView聚合实例（旧版）。	请参见开服地域。	v2.8.3.4

2022年02月

功能名称	功能概述	支持地域	版本号
针对部分地域进行商业化计费	针对政务云、金融云、青岛、呼和浩特、河源、广州、成都、中国香港、美国、英国、德国等地域正式收费。更多信息，请参见按量付费。	请参见开服地域。	v2.8.3.2
新增报警模板功能，支持针对多个Kubernetes集群统一做报警配置	为了实现同步管理多个跨地域Prometheus实例的告警，阿里云Prometheus监控提供了告警规则模板功能，帮助您快速为多个Prometheus实例创建告警规则，统一管理，降低管理多个Prometheus实例告警规则的成本。更多信息，请参见Prometheus告警规则模板。	请参见开服地域。	v2.8.3.2

2022年01月

功能名称	功能概述	支持地域	版本号
RocketMQ组件监控新增大盘	阿里云Prometheus监控提供一键安装配置RocketMQ类型组件，并提供开箱即用的专属监控大盘。更多信息，请参见使用阿里云Prometheus监控RocketMQ。	请参见开服地域。	v2.8.3

1.1.2. 2021年

本文为Prometheus监控2021年的版本发布记录，介绍发布的特性变更情况。

2021年12月

功能名称	功能概述	发布时间	支持地域	版本号
多实例聚合查询GlobalView	阿里云Prometheus监控提供地域级别的GlobalView聚合实例的功能。GlobalView聚合实例功能可以为您提供在当前地域下所有Prometheus实例的一个虚拟聚合实例。针对这个虚拟聚合实例可以实现统一的指标查询和告警。更多信息，请参见GlobalView聚合实例（旧版）。	2021-12-03	请参见开服地域。	v2.8.2.2
提供指标存储市场自定义功能	指标存储时长最高支持180天。	2021-12-03	请参见开服地域。	v2.8.2.2

2021年11月

功能名称	功能概述	发布时间	支持地域	版本号
预聚合	预聚合（Recording Rule）可以对落地的指标数据做二次开发。可以配置预聚合规则将计算过程提前到写入端，减少查询端资源占用，尤其在大规模集群和复杂业务场景下可以有效的降低PromQL的复杂度，从而提高查询性能，解决用户配置以及查询慢的问题。	2021-11-19	请参见开服地域。	v2.8.2.1

2021年10月

功能名称	功能概述	发布时间	支持地域	版本号
新增Recording Rule	阿里云Prometheus监控的RecordingRule.yaml只需配置Rule Group。 暂不支持Recording Rule的Remote Write。更多信息，请参见Prometheus监控设置。	2021-10-14	请参见开服地域。	v2.8.1.5

2021年09月

功能名称	功能概述	发布时间	支持地域	版本号
鉴权Token	阿里云Prometheus接入自建Grafana，使用Prometheus Metrics实现HPA等场景需要远程读取ARMS Prometheus存储数据；自建Prometheus、纳管集群（注册集群）等场景需要远程写入ARMS Prometheus存储。在远程读写Prometheus数据时，使用鉴权Token可以提高数据读写的安全性。更多信息，请参见探针设置。	2021-09-29	请参见开服地域。	v2.8.1.4

2021年08月

功能名称	功能概述	发布时间	支持地域	版本号
产品优化	• 支持ACK纳管集群采集存储指标。 • 预置大盘Ingress更新。	2021-08-05	请参见开服地域。	v2.8.1.1
Bug修复	修复CMS采集的指标Region Label值错误问题。	2021-08-05	请参见开服地域。	v2.8.1.1

2021年07月

功能名称	功能概述	发布时间	支持地域	版本号
支持接入VPC网络下的ECS集群	Prometheus监控支持接入VPC网络的ECS集群，创建大盘后可以监控ECS集群的众多性能指标。更多内容，请参见Prometheus实例 for VPC。	2021-07-15	• 华东2（上海） • 华南1（深圳） • 华北2（北京） • 华北3（张家口）	v2.8.0.3

2021年05月

功能名称	功能概述	发布时间	支持地域	版本号
功能优化	标签值更新之后，Prometheus监控不再采集旧的指标。	2021-05-13	请参见开服地域。	v2.8.0

2021年04月

功能名称	功能概述	发布时间	支持地域	版本号
HTTP监测	Prometheus的健康巡检支持对ACK Service的HTTP类型的外部端点进行监测。更多内容，请参见创建ACK Service巡检。	2021-04-22	请参见开服地域。	v2.7.9.3
重置大盘	• 支持将所有基础大盘重置到最新版本。 • 支持升级单个基础大盘。 • 支持查看大盘的核心指标。 更多内容，请参见重置大盘。	2021-04-08	请参见开服地域。	v2.7.9.2

2021年02月

功能名称	功能概述	发布时间	支持地域	版本号
默认服务发现	支持开启和关闭Prometheus的默认服务发现。	2021-02-25	请参见开服地域。	V2.7.8.3

2021年01月

功能名称	功能概述	发布时间	支持地域	版本号
更新报警配置	Prometheus报警新增设置通知策略。更多内容，请参见创建报警。	2021-01-29	请参见开服地域。	v2.7.8.2

1.1.3. 2020年

本文为Prometheus监控2020年的版本发布记录，介绍发布的特性变更情况。

2020年12月

支持远程存储。

2020年7月

支持函数计算服务的默认监控接入。

2020年6月

支持自建Kubernetes集群接入。

2020年5月

支持基于blackbox的内网监控巡检。

1.2. 产品公告

1.2.1. 【产品公告】Prometheus监控商用通知

阿里云Prometheus监控在金融云、政务云、华北1（青岛）、华北5（呼和浩特）、华南2（河源）、华南3（广州）、西南1（成都）、中国香港（香港）、美国东部1（弗吉尼亚）、美国西部1（硅谷）、英国伦敦（伦敦）以及欧洲中部1（法兰克福）这些地域于2022年02月17日00:00起正式商用。

收费说明

商用后阿里云Prometheus监控定价详情，请参见按量付费。

阿里云Prometheus监控目前支持的地域信息，请参见开服地域。

2.产品简介

2.1. 什么是Prometheus监控

阿里云Prometheus监控全面对接开源Prometheus生态，支持类型丰富的组件监控，提供多种开箱即用的预置监控大盘，且提供全面托管的Prometheus服务。

> **说明** Prometheus是一套开源的监控报警系统。主要特点包括多维数据模型、灵活查询语句PromQL以及数据可视化展示等。更多信息，请参见Prometheus官方文档。

什么是Prometheus实例

Prometheus实例是阿里云Prometheus监控服务提供的管理Prometheus数据采集和数据存储分析的逻辑单元，每个Prometheus实例提供对应的Prometheus数据采集配置、时序数据库实例、Dashboard监控大盘和报警配置等。根据Prometheus监控的不同对象和使用场景，Prometheus实例可以分为以下几种类型。

Prometheus实例类型	Prometheus监控的对象	监控能力	应用场景
Prometheus实例 for 容器服务	阿里云容器服务ACK和ASK集群	• 提供与容器服务原生的集成能力。包括容器服务Helm Chart的安装和更新，以及默认对容器服务集群控制面板和工作负载的监控。 • 默认开启以下服务发现能力：Kubernetes SD、 Service Monitor、Pod Monitor和基于Prometheus.yaml的自定义服务发现。	适合需要对容器服务集群及其上面运行的应用进行一体化监控场景。
Prometheus实例 for VPC	阿里云ECS环境	• 通过在VPC内的Prometheus探针提供针对此VPC内ECS应用和组件（如数据库，中间库等）的一体化监控。 • 提供ECS默认服务发现和基于Prometheus.yaml的自定义服务发现。	适合需要在阿里云VPC内（通常为ECS集群）进行Prometheus监控的场景。
Prometheus实例 for Kubernetes	非阿里云容器服务的Kubernetes集群，通常为在阿里云上自建的Kubernetes集群或者运行在其他云环境的Kubernetes集群。	• 提供标准的Helm Chart的安装和更新方式。 • 默认开启以下服务发现能力：Kubernetes Pod SD、 Service Monitor、Pod Monitor和基于Prometheus.yaml的自定义服务发现。	适合需要对自建Kubernetes集群及其上面运行的应用的一体化监控场景。

Prometheus实例类型	Prometheus监控的对象	监控能力	应用场景
Prometheus实例 for Remote Write	自建的Prometheus	• 提供Prometheus时序数据库的远端存储。 • 提供Grafana监控大盘进行数据的展示。 说明 由于Prometheus Server为用户自己运维，所以相应的服务发现、数据采集等需用户自行配置。	适合已自建了Prometheus Server，但是需要通过Remote Write的方式来解决Prometheus存储的可用性和可扩展性的场景。
Prometheus实例 for 云服务	阿里云云服务	• 提供各阿里云云服务（ECS、RDS、SLB等）的监控数据。 • 提供对应的Grafana监控大盘和告警。	适合需要通过Prometheus监控来统一采集、存储和显示阿里云云服务的监控数据的场景。

为什么选择Prometheus监控

阿里云Prometheus为用户的应用平台提供多场景、多层次、多维度指标数据的监控能力，结合Grafana大盘和AlertManager告警功能。在完全兼容开源Prometheus生态，以开放的方式为用户提供服务的原则下，阿里云Prometheus监控帮助用户轻松构建全面、稳定、安全、高可用性和高扩展性的可观测平台。

多场景应用监控

阿里云Prometheus监控支持为Kubernetes容器应用、ECS集群(VPC)、远程存储等场景提供免费、开箱即用的Node，Kube控制面组件和容器的监控告警能力。同时支持一键安装部署接入自建的Kubernetes。

多层次组件监控

提供包括基础设施层、中间层、应用层等近30款云产品的组件监控告警能力，在中间层提供了Prometheus社区开源Exporter的安装部署。同时阿里云Prometheus监控服务支持快速接入应用层的监控告警。

多维度指标监控

提供20多种语言的Client Library，可以低成本、低消耗的生产和暴露OpenMetrics格式的指标供Prometheus监控采集，不仅可以采集常规维度的指标。同时还支持采集事件、标签维度的指标，以及通过其他方式转换的指标采集。

更多详细信息，请参见功能特性和产品优势。

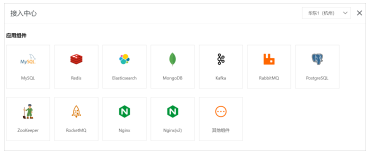
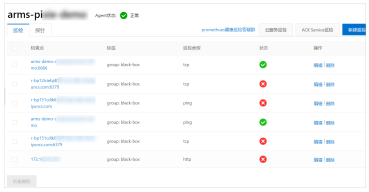
相关文档

-
- 按量付费

2.2. 功能特性

阿里云Prometheus监控具有监控数据采集、存储、计算、数据展示、报警等能力。监控指标覆盖广，涵盖容器、Kubernetes、云服务、中间件、数据库、应用以及业务等多种监控数据。本文介绍Prometheus监控支持的主要功能。

监控对象接入

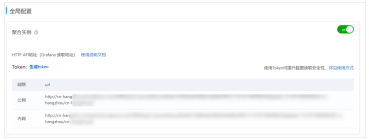
功能	功能说明	控制台示例图
创建Prometheus实例	支持创建5种类型的Prometheus实例。您可以根据需求选择创建任一类型的Prometheus实例。	
组件监控接入	支持一键接入多种组件应用。自动创建Exporter以及对应的Grafana面板，监测并展示其指标数据。	
健康巡检	支持云服务巡检、ACK Service巡检以及自定义健康巡检方式。 定期对监控的服务进行连接测试。帮助您掌握服务的健康状况，及时发现异常，从而采取针对性的有效措施。	

监控指标采集

功能	功能说明	控制台示例图
服务发现	默认服务发现：是Prometheus监控内置的服务发现功能，在接入Prometheus监控时自动开启。 当前默认服务发现指标采集对象为Kubernetes集群下所有Namespace包含的Pod。	
	ServiceMonitor：支持手动添加ServiceMonitor配置Prometheus监控的采集规则进行指标采集。	
	PodMonitor：支持手动添加PodMonitor配置Prometheus监控的采集规则进行指标采集。	
编辑Prometheus.yaml	支持通过编辑Prometheus.yaml的方式为应用配置Prometheus监控的采集规则。	

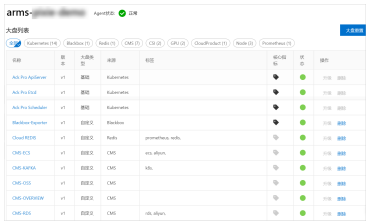
功能	功能说明	控制台示例图
查看指标	支持查看基础指标和自定义指标。对于不再需要监控的指标，支持配置废弃指标。	
Targets	支持通过Targets可以直观查看正在被抓取的目标，以及抓取状态是否正常。同时支持查看目标中暴露的metrics。	

监控数据处理

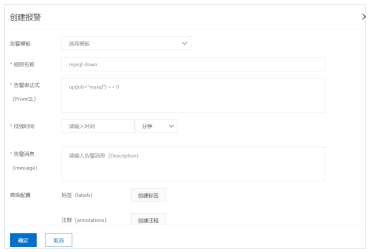
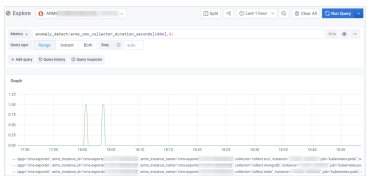
功能	功能说明	控制台示例图
获取Remote Write地址 获取Remote Read地址	Remote Write功能支持作为远程数据库存储Prometheus监控数据。您可以使用Remote Read地址和Remote Write地址，将自建Prometheus的监控数据存储到阿里云Prometheus实例中，实现远程存储。	
编辑RecordingRule.yaml	预聚合（Recording Rule）可以对落地的指标数据做二次开发。可以配置预聚合规则将计算过程提前到写入端，减少查询端资源占用，尤其在大规模集群和复杂业务场景下可以有效的降低PromQL的复杂度，从而提高查询性能，解决用户配置以及查询慢的问题。	
聚合实例	提供在当前地域下所有Prometheus实例的一个虚拟聚合实例。针对这个虚拟聚合实例可以实现统一的指标查询和告警。	

监控数据展示

功能	功能说明	控制台示例图

功能	功能说明	控制台示例图
查看Grafana大盘	预置丰富的Grafana大盘，同时支持自定义大盘来展示监控数据。	
获取HTTP API地址	提供了HTTP API地址，您可以通过该地址将阿里云Prometheus实例的监控数据接入自建的Grafana大盘展示数据，也可以获取阿里云Prometheus监控数据进行二次开发。	

报警

功能	功能说明	控制台示例图
创建报警	预置多种报警规则，支持针对特定监控对象自定义报警规则。 当规则被触发时，系统会以您指定的报警方式向报警联系人分组发送报警信息，以提醒报警联系人采取必要的问题解决措施。	
管理报警	支持对报警规则执行开启、关闭、编辑、删除等操作。	
智能检测算子	支持通过智能检测算子算法自动地发现KPI时间序列数据中的异常波动，实现时间序列的异常检测，为后续的告警、自动止损、根因分析等提供决策依据。	

Prometheus实例管理

功能	功能说明	控制台示例图

功能	功能说明	控制台示例图
调整存储时长	支持手动设置指标的存储天数。	
设置Agent副本数	支持Agent副本数水平伸缩（HPA）自动扩容的能力，均衡分解采集任务，实现动态扩缩，解决开源版本无法水平扩展与高可用问题。	
探针管理	支持查看Prometheus探针的基本信息和健康检查结果、设置Agent副本数、重启探针。	

> **说明**
>
> - 不同类型的Prometheus实例支持的功能可能会有所差异。
> - Prometheus监控仅支持Helm包安装，Helm包的安装命令会在文档中提供，命令包含Prometheus Operator版本信息。更多信息，请参见Helm命令参数说明。

2.3. 产品优势

阿里云Prometheus监控全面对接开源Prometheus生态，支持类型丰富的组件监控，覆盖绝大部分开源基础设施软件指标采集能力。提供多种开箱即用的预置监控大盘，并集成丰富的Kubernetes基础监控以及常用服务预设看板，且提供全面托管的Prometheus服务。阿里云Prometheus监控的优势包含以下几点。

阿里云Prometheus监控与开源Prometheus对比

优势对比项	阿里云Prometheus监控	开源Prometheus
购买资源和系统搭建	阿里云全托管	自行购买相关资源并部署系统
运维成本	免运维	日常自行运维
高可用性	支持采集存储组件多副本，可水平扩展	单进程，无法水平扩展
数据接入	一键接入常见云产品，覆盖数据库、中间件等主流应用组件，以及Java和Go等主流编程语言构建的应用，支持ECS集群中间件的免Agent安装监控	创建对应组件的Exporter，完成数据接入
数据存储	基于云上存储，存储容量无上限	受限于存储容量

优势对比项	阿里云Prometheus监控	开源Prometheus
数据可视化	内置Grafana，各类常见监控模板开箱即用	需要单独部署Grafana，并自行配置看板
告警管理	集成ARMS告警中心，提升告警效率与精度	自行接入Alertmanager插件
单副本采集性能（2C 4G）	600w 时间点/次	100w 时间点/次
数据查询性能（6亿时间点）	8~10s	180s
安全管理	阿里云安全能力加持，并支持鉴权功能	不支持
其他能力	支持预计算、降采样等能力	不支持

开箱即用

- 一键安装部署即可监控Kubernetes以及各类云产品。
- 一键接入各种应用组件及告警工具。
- 扩增主动健康巡检、Agent升级、白屏化配置、云产品集成等场景功能，强化运维监控能力。

低成本

- 多种指标免费用，覆盖Kubernetes标准组件。
- 提供全托管式服务，无需另购资源，可降低监控成本，且维护成本几乎为零。
- 与阿里云容器服务ACK集成并提供监控服务，容器监控体系创建时间从3天降低至10分钟。

开源兼容

- 兼容标准开源Prometheus.yaml采集规则配置文件、适合自定义Kubernetes内监控采集规则ServiceMonitor、默认采集规则Annotation。
- 支持自定义多维数据模型、HTTP API模块、PromQL查询。
- 静态文件配置和动态发现机制发现监控对象，实现轻松迁移及接入。

数据规模无上限

- 凭借云存储能力，数据存储无上限，不再受限于本地容量。云端分布式存储保障数据可靠性。
- 通过Global DataSource和Global View实现对多套Kubernetes集群的统一监控，用户可以实现跨Kubernetes集群的聚合查询。

高性能

- 相较开源版本结构更轻量，资源消耗更低。通过单进程一体化Agent监控Kubernetes集群，采集性

能提升20倍。

- Agent部署在用户侧，保留原生采集能力同时能够最大程度的减少资源的使用。
- 通过采集存储分离架构，全面提升整体性能。
- 采集组件优化，提升单副本采集能力，降低资源消耗。
- 通过多副本横向扩展均衡分解采集任务，实现动态扩缩，解决开源水平扩展问题。

高可用性

- 双副本：数据采集、处理和存储组件支持多副本横向扩展，保证核心数据链路高可用。
- 水平扩展：基于集群规模可直接进行弹性扩容。
- 数据重传：支持数据自动重传，彻底解决丢弃逻辑弊病，确保数据完整性与准确性。

2.4. 基本概念

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

概念	说明
Exporter	和监控对象伴生运行的应用。通常用于将监控对象存量的监控数据转换成Prometheus监控可以识别的OpenMetrics数据格式，暴露指标。目前有100+官方或者三方Exporter可供使用，请参见Exporter详情。
Job	一组Target的配置集合。定义了抓取间隔，访问限制等作用于一组Target的抓取行为。
Prometheus监控	阿里云Prometheus监控全面对接开源Prometheus生态，支持类型丰富的组件监控，提供多种开箱即用的预置监控大盘，且提供全面托管的Prometheus服务。通过阿里云Prometheus监控可以创建多种类型的Prometheus实例。
Prometheus实例	阿里云Prometheus监控提供的管理Prometheus监控数据采集和数据存储分析的逻辑单元。
Prometheus探针	部署在用户侧或者云产品侧Kubernetes集群。负责自动发现采集目标、采集指标和远程写到其他库。
PromQL	Prometheus监控的查询语言。支持瞬时查询和时间跨度查询，内置多种函数和操作符。可以对原始数据进行聚合、切片、预测和联合。
Sample	一条时间线在某个时间点对应的数值。在Prometheus监控中，每个Sample由一个float64数据类型的值和一个毫秒精度的时间戳构成。
Target	Prometheus探针要抓取的采集目标。采集目标暴露自身运行、业务指标，或者代理暴露监控对象的运行、业务指标。
告警规则	Prometheus监控Alerting Rule格式的告警配置。可以通过PromQL描述。
标签	描述指标的一组Key-Value值。
服务发现	Prometheus监控的功能特点之一，无需静态配置，可以自动发现采集目标。支持Kubernetes SD、Consul、Eureka等多种服务发现方式，支持通过Service Monitor、Pod Monitor的方式暴露采集目标。

概念	说明
预计算	Prometheus监控Recording Rule能力。可以通过PromQL将原始数据加工成新的指标，提升查询效率。
时间线	由指标名和标签组成。相同的指标名和标签在时间序列中构成唯一的一条时间线。
远程存储	阿里云自研的时序数据存储组件。支持Prometheus监控Remote Write协议，由云产品全面托管。
云产品监控	无缝集成了阿里云多种云产品的监控数据。用户如果有阿里云云产品的监控需求，可以通过接入云产品监控实施。
指标	采集目标暴露的、可以完整反映监控对象运行或者业务状态的一系列标签化数据。Prometheus监控采用OpenMetrics的标准数据格式描述指标。

2.5. 开服地域

本文介绍Prometheus监控支持的地域。

亚太-中国

云服务	地域名称	所在城市	Region ID	试用版	专家版（按量付费）
公共云	华东1	杭州	cn-hangzhou	✓	✓
	华东2	上海	cn-shanghai	✓	✓
	华北1	青岛	cn-qingdao	✓	✓
	华北2	北京	cn-beijing	✓	✓
	华北3	张家口	cn-zhangjiakou	✓	✓
	华北5	呼和浩特	cn-huhehaote	✓	✓
	华北6	乌兰察布	cn-wulanchabu	公测中	
	华南1	深圳	cn-shenzhen	✓	✓
	华南2	河源	cn-heyuan	✓	✓

云服务	地域名称	所在城市	Region ID	试用版	专家版（按量付费）
	华南3	广州	cn-guangzhou	✓	✓
	西南1	成都	cn-chengdu	✓	✓
	中国香港	香港	cn-hongkong	✓	✓
金融云	华东1	杭州	cn-hangzhou-finance	✓	✓
	华东2	上海	cn-shanghai-finance-1	✓	✓
	华南1	深圳	cn-shenzhen-finance-1	✓	✓
政务云	华北2	北京	cn-north-2-gov-1	✓	✓

亚太-其他

云服务	地域名称	所在城市	Region ID	试用版	专家版（按量付费）
公共云	亚太东南1	新加坡	ap-southeast-1	✓	✓
	亚太东南3	吉隆坡	ap-southeast-3	✓	✓
	亚太东南5	雅加达	ap-southeast-5	✓	✓
	亚太东南6	马尼拉	ap-southeast-6	公测中	
	亚太东北1	东京	ap-northeast-1	公测中	
	亚太南部1	孟买	ap-south-1	✓	✓

欧洲与美洲

云服务	地域名称	所在城市	Region ID	试用版	专家版（按量付费）
公共云	欧洲中部1	法兰克福	eu-central-1	✓	✓
	英国（伦敦）	伦敦	eu-west-1	✓	✓
	美国东部1	弗吉尼亚	us-east-1	✓	✓
	美国西部1	硅谷	us-west-1	✓	✓

2.6. 阿里云Prometheus监控开源一致性认证

在最新的Prometheus PromQL开源一致性认证中，阿里云Prometheus监控的通过率为97.99%（比较基准为开源Prometheus v2.32.1版本），这意味着阿里云Prometheus监控在PromQL的使用体验上，与开源版本几乎完全保持一致，最大程度方便用户迁移上云。

背景信息

Prometheus开源一致性认证，是由PromLabs发起，此项认证目前已被纳入开源Prometheus官方的Git项目库。

开源一致性认证会从多个维度比对各个服务提供商提供的服务，与开源版本保持一致的程度。其中包括PromQL、alert_generator、openmetrics、remote_write_receiver和remote_write_sender。其中，PromQL一致性认证项目，涉及采样数据的写入和读取全流程，以最终查询结果与开源实现完全一致作为通过标准，是流程最长的一项测试，也是最能体现最终用户感受的认证项目。

认证结果说明

阿里云Prometheus监控产品功能在不断丰富和演进，对于其一致性的评测也是有一定时效性的，对于此项认证，您需要了解以下事项：

- 作为云服务提供商，在跟进Prometheus开源版本的新功能时可能会有一些迟滞，所以在进行PromLabs的开源一致性认证时，会使用当前开源Prometheus最新版本的N-2版本作为认证测试的基准。比如截止2022年02月09日，开源Prometheus的最新版本为v2.33.1，上一个版本为v2.33.0，上上一个版本为v2.32.1，那么就使用v2.32.1版本作为PromLabs的开源一致性认证基准版本。
- 认证配置文件中允许服务提供商配置query_tweaks，即对查询做一些微调，比如增加一些条件或者丢弃一些Label等，虽然这样做是允许的，但如果不使用query_tweaks更好一些。阿里云Prometheus监控未使用任何query_tweaks。
- 认证结果的有效期时长为Prometheus的两个小版本（即minor版本）更新时长，或者是12周的时长，这两者之间选择时长较长的那一个为准。
- 阿里云Prometheus监控未通过的测试用例（case）主要与last_over_time和 clamp这两个近期新增的PromQL函数相关，目前阿里云Prometheus监控团队正在对此进行相关开发。

测试结果明细

阿里云Prometheus监控PromLabs开源一致性认证的测试用例总数为548，通过数为537，即通过率为97.99%，具体信息如下：

Query	Outcome
42	PASS
1.234	PASS
.123	PASS
1.23e-3	PASS
0x3d	PASS
Inf	PASS
+Inf	PASS
-Inf	PASS
NaN	PASS
demo_memory_usage_bytes	PASS
{__name__="demo_memory_usage_bytes"}	PASS
demo_memory_usage_bytes{type="free"}	PASS
demo_memory_usage_bytes{type!="free"}	PASS
demo_memory_usage_bytes{instance=~"demo.promlabs.com:.*"}	PASS
demo_memory_usage_bytes{instance=~"host"}	PASS
demo_memory_usage_bytes{instance!~".*:10000"}	PASS
demo_memory_usage_bytes{type="free", instance!="demo.promlabs.com:10000"}	PASS
{type="free", instance!="demo.promlabs.com:10000"}	PASS
{__name__=~".*"}	PASS
nonexistent_metric_name	PASS
demo_memory_usage_bytes offset 1m	PASS
demo_memory_usage_bytes offset 5m	PASS
demo_memory_usage_bytes offset 10m	PASS

Query	Outcome
demo_memory_usage_bytes offset -1m	PASS
demo_memory_usage_bytes offset -5m	PASS
demo_memory_usage_bytes offset -10m	PASS
demo_intermittent_metric	FAIL
sum(demo_memory_usage_bytes)	PASS
avg(demo_memory_usage_bytes)	PASS
max(demo_memory_usage_bytes)	PASS
min(demo_memory_usage_bytes)	PASS
count(demo_memory_usage_bytes)	PASS
stddev(demo_memory_usage_bytes)	PASS
stdvar(demo_memory_usage_bytes)	PASS
sum(nonexistent_metric_name)	PASS
avg(nonexistent_metric_name)	PASS
max(nonexistent_metric_name)	PASS
min(nonexistent_metric_name)	PASS
count(nonexistent_metric_name)	PASS
stddev(nonexistent_metric_name)	PASS
stdvar(nonexistent_metric_name)	PASS
sum by() (demo_memory_usage_bytes)	PASS
avg by() (demo_memory_usage_bytes)	PASS
max by() (demo_memory_usage_bytes)	PASS
min by() (demo_memory_usage_bytes)	PASS
count by() (demo_memory_usage_bytes)	PASS
stddev by() (demo_memory_usage_bytes)	PASS
stdvar by() (demo_memory_usage_bytes)	PASS
sum by(instance) (demo_memory_usage_bytes)	PASS

Query	Outcome
avg by(instance) (demo_memory_usage_bytes)	PASS
max by(instance) (demo_memory_usage_bytes)	PASS
min by(instance) (demo_memory_usage_bytes)	PASS
count by(instance) (demo_memory_usage_bytes)	PASS
stddev by(instance) (demo_memory_usage_bytes)	PASS
stdvar by(instance) (demo_memory_usage_bytes)	PASS
sum by(instance, type) (demo_memory_usage_bytes)	PASS
avg by(instance, type) (demo_memory_usage_bytes)	PASS
max by(instance, type) (demo_memory_usage_bytes)	PASS
min by(instance, type) (demo_memory_usage_bytes)	PASS
count by(instance, type) (demo_memory_usage_bytes)	PASS
stddev by(instance, type) (demo_memory_usage_bytes)	PASS
stdvar by(instance, type) (demo_memory_usage_bytes)	PASS
sum by(nonexistent) (demo_memory_usage_bytes)	PASS
avg by(nonexistent) (demo_memory_usage_bytes)	PASS
max by(nonexistent) (demo_memory_usage_bytes)	PASS
min by(nonexistent) (demo_memory_usage_bytes)	PASS
count by(nonexistent) (demo_memory_usage_bytes)	PASS
stddev by(nonexistent) (demo_memory_usage_bytes)	PASS
stdvar by(nonexistent) (demo_memory_usage_bytes)	PASS
sum without() (demo_memory_usage_bytes)	PASS

Query	Outcome
avg without() (demo_memory_usage_bytes)	PASS
max without() (demo_memory_usage_bytes)	PASS
min without() (demo_memory_usage_bytes)	PASS
count without() (demo_memory_usage_bytes)	PASS
stddev without() (demo_memory_usage_bytes)	PASS
stdvar without() (demo_memory_usage_bytes)	PASS
sum without(instance) (demo_memory_usage_bytes)	PASS
avg without(instance) (demo_memory_usage_bytes)	PASS
max without(instance) (demo_memory_usage_bytes)	PASS
min without(instance) (demo_memory_usage_bytes)	PASS
count without(instance) (demo_memory_usage_bytes)	PASS
stddev without(instance) (demo_memory_usage_bytes)	PASS
stdvar without(instance) (demo_memory_usage_bytes)	PASS
sum without(instance, type) (demo_memory_usage_bytes)	PASS
avg without(instance, type) (demo_memory_usage_bytes)	PASS
max without(instance, type) (demo_memory_usage_bytes)	PASS
min without(instance, type) (demo_memory_usage_bytes)	PASS
count without(instance, type) (demo_memory_usage_bytes)	PASS
stddev without(instance, type) (demo_memory_usage_bytes)	PASS
stdvar without(instance, type) (demo_memory_usage_bytes)	PASS

Query	Outcome
sum without(nonexistent) (demo_memory_usage_bytes)	PASS
avg without(nonexistent) (demo_memory_usage_bytes)	PASS
max without(nonexistent) (demo_memory_usage_bytes)	PASS
min without(nonexistent) (demo_memory_usage_bytes)	PASS
count without(nonexistent) (demo_memory_usage_bytes)	PASS
stddev without(nonexistent) (demo_memory_usage_bytes)	PASS
stdvar without(nonexistent) (demo_memory_usage_bytes)	PASS
topk (3, demo_memory_usage_bytes)	PASS
bottomk (3, demo_memory_usage_bytes)	PASS
topk by(instance) (2, demo_memory_usage_bytes)	PASS
bottomk by(instance) (2, demo_memory_usage_bytes)	PASS
topk without(instance) (2, demo_memory_usage_bytes)	PASS
bottomk without(instance) (2, demo_memory_usage_bytes)	PASS
topk without() (2, demo_memory_usage_bytes)	PASS
bottomk without() (2, demo_memory_usage_bytes)	PASS
quantile(-0.5, demo_memory_usage_bytes)	PASS
quantile(0.1, demo_memory_usage_bytes)	PASS
quantile(0.5, demo_memory_usage_bytes)	PASS
quantile(0.75, demo_memory_usage_bytes)	PASS
quantile(0.95, demo_memory_usage_bytes)	PASS
quantile(0.90, demo_memory_usage_bytes)	PASS

Query	Outcome
quantile(0.99, demo_memory_usage_bytes)	PASS
quantile(1, demo_memory_usage_bytes)	PASS
quantile(1.5, demo_memory_usage_bytes)	PASS
avg(max by(type) (demo_memory_usage_bytes))	PASS
1 * 2 + 4 / 6 - 10 % 2 ^ 2	PASS
demo_num_cpus + (1 == bool 2)	PASS
demo_num_cpus + (1 != bool 2)	PASS
demo_num_cpus + (1 < bool 2)	PASS
demo_num_cpus + (1 > bool 2)	PASS
demo_num_cpus + (1 <= bool 2)	PASS
demo_num_cpus + (1 >= bool 2)	PASS
demo_memory_usage_bytes + 1.2345	PASS
demo_memory_usage_bytes - 1.2345	PASS
demo_memory_usage_bytes * 1.2345	PASS
demo_memory_usage_bytes / 1.2345	PASS
demo_memory_usage_bytes % 1.2345	PASS
demo_memory_usage_bytes ^ 1.2345	PASS
demo_memory_usage_bytes == 1.2345	PASS
demo_memory_usage_bytes != 1.2345	PASS
demo_memory_usage_bytes < 1.2345	PASS
demo_memory_usage_bytes > 1.2345	PASS
demo_memory_usage_bytes <= 1.2345	PASS
demo_memory_usage_bytes >= 1.2345	PASS
demo_memory_usage_bytes == bool 1.2345	PASS
demo_memory_usage_bytes != bool 1.2345	PASS
demo_memory_usage_bytes < bool 1.2345	PASS

Query	Outcome
demo_memory_usage_bytes > bool 1.2345	PASS
demo_memory_usage_bytes <= bool 1.2345	PASS
demo_memory_usage_bytes >= bool 1.2345	PASS
1.2345 == bool demo_memory_usage_bytes	PASS
1.2345 != bool demo_memory_usage_bytes	PASS
1.2345 < bool demo_memory_usage_bytes	PASS
1.2345 > bool demo_memory_usage_bytes	PASS
1.2345 <= bool demo_memory_usage_bytes	PASS
1.2345 >= bool demo_memory_usage_bytes	PASS
0.12345 + demo_memory_usage_bytes	PASS
0.12345 - demo_memory_usage_bytes	PASS
0.12345 * demo_memory_usage_bytes	PASS
0.12345 / demo_memory_usage_bytes	PASS
0.12345 % demo_memory_usage_bytes	PASS
0.12345 ^ demo_memory_usage_bytes	PASS
0.12345 == demo_memory_usage_bytes	PASS
0.12345 != demo_memory_usage_bytes	PASS
0.12345 < demo_memory_usage_bytes	PASS
0.12345 > demo_memory_usage_bytes	PASS
0.12345 <= demo_memory_usage_bytes	PASS
0.12345 >= demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) + demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) - demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) * demo_memory_usage_bytes	PASS

Query	Outcome
(1 * 2 + 4 / 6 - (10%7)^2) / demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) % demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) ^ demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) == demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) != demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) < demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) > demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) <= demo_memory_usage_bytes	PASS
(1 * 2 + 4 / 6 - (10%7)^2) >= demo_memory_usage_bytes	PASS
demo_memory_usage_bytes + (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes - (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes * (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes / (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes % (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes ^ (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes == (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes != (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes < (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes > (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes <= (1 * 2 + 4 / 6 - 10)	PASS
demo_memory_usage_bytes >= (1 * 2 + 4 / 6 - 10)	PASS
timestamp(demo_memory_usage_bytes * 1)	PASS

Query	Outcome
timestamp(-demo_memory_usage_bytes)	PASS
demo_memory_usage_bytes + on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes - on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes * on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes / on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes % on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes ^ on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes == on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes != on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes < on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes > on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes <= on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes >= on(instance, job, type) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) + on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) - on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) * on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) / on(instance, type) group_left(job) demo_memory_usage_bytes	PASS

Query	Outcome
sum by(instance, type) (demo_memory_usage_bytes) % on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) ^ on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) == on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) != on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) < on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) > on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) <= on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
sum by(instance, type) (demo_memory_usage_bytes) >= on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes == bool on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes != bool on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes < bool on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes > bool on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes <= bool on(instance, job, type) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes >= bool on(instance, job, type) demo_memory_usage_bytes	PASS

Query	Outcome
demo_memory_usage_bytes / on(instance, job, type, __name__) demo_memory_usage_bytes	PASS
sum without(job) (demo_memory_usage_bytes) / on(instance, type) demo_memory_usage_bytes	PASS
sum without(job) (demo_memory_usage_bytes) / on(instance, type) group_left demo_memory_usage_bytes	PASS
sum without(job) (demo_memory_usage_bytes) / on(instance, type) group_left(job) demo_memory_usage_bytes	PASS
demo_memory_usage_bytes / on(instance, job) group_left demo_num_cpus	PASS
demo_memory_usage_bytes / on(instance, type, job, non_existent) demo_memory_usage_bytes	PASS
demo_num_cpus * Inf	PASS
demo_num_cpus * -Inf	PASS
demo_num_cpus * NaN	PASS
demo_memory_usage_bytes + -(1)	PASS
-demo_memory_usage_bytes	PASS
-1 ^ 2	PASS
1 + time()	PASS
1 - time()	PASS
1 * time()	PASS
1 / time()	PASS
1 % time()	PASS
1 ^ time()	PASS
time() + 1	PASS
time() - 1	PASS
time() * 1	PASS
time() / 1	PASS

Query	Outcome
time() % 1	PASS
time() ^ 1	PASS
time() == bool 1	PASS
time() != bool 1	PASS
time() < bool 1	PASS
time() > bool 1	PASS
time() <= bool 1	PASS
time() >= bool 1	PASS
1 == bool time()	PASS
1 != bool time()	PASS
1 < bool time()	PASS
1 > bool time()	PASS
1 <= bool time()	PASS
1 >= bool time()	PASS
time() + time()	PASS
time() - time()	PASS
time() * time()	PASS
time() / time()	PASS
time() % time()	PASS
time() ^ time()	PASS
time() == bool time()	PASS
time() != bool time()	PASS
time() < bool time()	PASS
time() > bool time()	PASS
time() <= bool time()	PASS
time() >= bool time()	PASS

Query	Outcome
time() + demo_memory_usage_bytes	PASS
time() - demo_memory_usage_bytes	PASS
time() * demo_memory_usage_bytes	PASS
time() / demo_memory_usage_bytes	PASS
time() % demo_memory_usage_bytes	PASS
time() ^ demo_memory_usage_bytes	PASS
time() == demo_memory_usage_bytes	PASS
time() != demo_memory_usage_bytes	PASS
time() < demo_memory_usage_bytes	PASS
time() > demo_memory_usage_bytes	PASS
time() <= demo_memory_usage_bytes	PASS
time() >= demo_memory_usage_bytes	PASS
demo_memory_usage_bytes + time()	PASS
demo_memory_usage_bytes - time()	PASS
demo_memory_usage_bytes * time()	PASS
demo_memory_usage_bytes / time()	PASS
demo_memory_usage_bytes % time()	PASS
demo_memory_usage_bytes ^ time()	PASS
demo_memory_usage_bytes == time()	PASS
demo_memory_usage_bytes != time()	PASS
demo_memory_usage_bytes < time()	PASS
demo_memory_usage_bytes > time()	PASS
demo_memory_usage_bytes <= time()	PASS
demo_memory_usage_bytes >= time()	PASS
sum_over_time(demo_memory_usage_bytes[1s])	PASS
sum_over_time(demo_memory_usage_bytes[15s])	PASS

Query	Outcome
sum_over_time(demo_memory_usage_bytes[1m])	PASS
sum_over_time(demo_memory_usage_bytes[5m])	PASS
sum_over_time(demo_memory_usage_bytes[15m])	PASS
sum_over_time(demo_memory_usage_bytes[1h])	PASS
avg_over_time(demo_memory_usage_bytes[1s])	PASS
avg_over_time(demo_memory_usage_bytes[15s])	PASS
avg_over_time(demo_memory_usage_bytes[1m])	PASS
avg_over_time(demo_memory_usage_bytes[5m])	PASS
avg_over_time(demo_memory_usage_bytes[15m])	PASS
avg_over_time(demo_memory_usage_bytes[1h])	PASS
max_over_time(demo_memory_usage_bytes[1s])	PASS
max_over_time(demo_memory_usage_bytes[15s])	PASS
max_over_time(demo_memory_usage_bytes[1m])	PASS
max_over_time(demo_memory_usage_bytes[5m])	PASS
max_over_time(demo_memory_usage_bytes[15m])	PASS
max_over_time(demo_memory_usage_bytes[1h])	PASS
min_over_time(demo_memory_usage_bytes[1s])	PASS
min_over_time(demo_memory_usage_bytes[15s])	PASS
min_over_time(demo_memory_usage_bytes[1m])	PASS
min_over_time(demo_memory_usage_bytes[5m])	PASS
min_over_time(demo_memory_usage_bytes[15m])	PASS
min_over_time(demo_memory_usage_bytes[1h])	PASS
count_over_time(demo_memory_usage_bytes[1s])	PASS
count_over_time(demo_memory_usage_bytes[15s])	PASS
count_over_time(demo_memory_usage_bytes[1m])	PASS
count_over_time(demo_memory_usage_bytes[5m])	PASS

Query	Outcome
count_over_time(demo_memory_usage_bytes[15m])	PASS
count_over_time(demo_memory_usage_bytes[1h])	PASS
stddev_over_time(demo_memory_usage_bytes[1s])	PASS
stddev_over_time(demo_memory_usage_bytes[15s])	PASS
stddev_over_time(demo_memory_usage_bytes[1m])	PASS
stddev_over_time(demo_memory_usage_bytes[5m])	PASS
stddev_over_time(demo_memory_usage_bytes[15m])	PASS
stddev_over_time(demo_memory_usage_bytes[1h])	PASS
stdvar_over_time(demo_memory_usage_bytes[1s])	PASS
stdvar_over_time(demo_memory_usage_bytes[15s])	PASS
stdvar_over_time(demo_memory_usage_bytes[1m])	PASS
stdvar_over_time(demo_memory_usage_bytes[5m])	PASS
stdvar_over_time(demo_memory_usage_bytes[15m])	PASS
stdvar_over_time(demo_memory_usage_bytes[1h])	PASS
absent_over_time(demo_memory_usage_bytes[1s])	PASS
absent_over_time(demo_memory_usage_bytes[15s])	PASS
absent_over_time(demo_memory_usage_bytes[1m])	PASS
absent_over_time(demo_memory_usage_bytes[5m])	PASS
absent_over_time(demo_memory_usage_bytes[15m])	PASS
absent_over_time(demo_memory_usage_bytes[1h])	PASS
last_over_time(demo_memory_usage_bytes[1s])	FAIL
last_over_time(demo_memory_usage_bytes[15s])	FAIL
last_over_time(demo_memory_usage_bytes[1m])	FAIL
last_over_time(demo_memory_usage_bytes[5m])	FAIL

Query	Outcome
last_over_time(demo_memory_usage_bytes[15m])	FAIL
last_over_time(demo_memory_usage_bytes[1h])	FAIL
quantile_over_time(-0.5, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(-0.5, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(-0.5, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(-0.5, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(-0.5, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(-0.5, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(0.1, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(0.1, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(0.1, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(0.1, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(0.1, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(0.1, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(0.5, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(0.5, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(0.5, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(0.5, demo_memory_usage_bytes[5m])	PASS

Query	Outcome
quantile_over_time(0.5, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(0.5, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(0.75, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(0.75, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(0.75, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(0.75, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(0.75, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(0.75, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(0.95, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(0.95, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(0.95, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(0.95, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(0.95, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(0.95, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(0.90, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(0.90, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(0.90, demo_memory_usage_bytes[1m])	PASS

Query	Outcome
quantile_over_time(0.90, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(0.90, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(0.90, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(0.99, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(0.99, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(0.99, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(0.99, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(0.99, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(0.99, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(1, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(1, demo_memory_usage_bytes[15s])	PASS
quantile_over_time(1, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(1, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(1, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(1, demo_memory_usage_bytes[1h])	PASS
quantile_over_time(1.5, demo_memory_usage_bytes[1s])	PASS
quantile_over_time(1.5, demo_memory_usage_bytes[15s])	PASS

Query	Outcome
quantile_over_time(1.5, demo_memory_usage_bytes[1m])	PASS
quantile_over_time(1.5, demo_memory_usage_bytes[5m])	PASS
quantile_over_time(1.5, demo_memory_usage_bytes[15m])	PASS
quantile_over_time(1.5, demo_memory_usage_bytes[1h])	PASS
timestamp(demo_num_cpus)	PASS
timestamp(timestamp(demo_num_cpus))	PASS
abs(demo_memory_usage_bytes)	PASS
ceil(demo_memory_usage_bytes)	PASS
floor(demo_memory_usage_bytes)	PASS
exp(demo_memory_usage_bytes)	PASS
sqrt(demo_memory_usage_bytes)	PASS
ln(demo_memory_usage_bytes)	PASS
log2(demo_memory_usage_bytes)	PASS
log10(demo_memory_usage_bytes)	PASS
round(demo_memory_usage_bytes)	PASS
abs(-demo_memory_usage_bytes)	PASS
ceil(-demo_memory_usage_bytes)	PASS
floor(-demo_memory_usage_bytes)	PASS
exp(-demo_memory_usage_bytes)	PASS
sqrt(-demo_memory_usage_bytes)	PASS
ln(-demo_memory_usage_bytes)	PASS
log2(-demo_memory_usage_bytes)	PASS
log10(-demo_memory_usage_bytes)	PASS
round(-demo_memory_usage_bytes)	PASS

Query	Outcome
delta(nonexistent_metric[5m])	PASS
rate(nonexistent_metric[5m])	PASS
increase(nonexistent_metric[5m])	PASS
delta(demo_cpu_usage_seconds_total[1s])	PASS
delta(demo_cpu_usage_seconds_total[15s])	PASS
delta(demo_cpu_usage_seconds_total[1m])	PASS
delta(demo_cpu_usage_seconds_total[5m])	PASS
delta(demo_cpu_usage_seconds_total[15m])	PASS
delta(demo_cpu_usage_seconds_total[1h])	PASS
rate(demo_cpu_usage_seconds_total[1s])	PASS
rate(demo_cpu_usage_seconds_total[15s])	PASS
rate(demo_cpu_usage_seconds_total[1m])	PASS
rate(demo_cpu_usage_seconds_total[5m])	PASS
rate(demo_cpu_usage_seconds_total[15m])	PASS
rate(demo_cpu_usage_seconds_total[1h])	PASS
increase(demo_cpu_usage_seconds_total[1s])	PASS
increase(demo_cpu_usage_seconds_total[15s])	PASS
increase(demo_cpu_usage_seconds_total[1m])	PASS
increase(demo_cpu_usage_seconds_total[5m])	PASS
increase(demo_cpu_usage_seconds_total[15m])	PASS
increase(demo_cpu_usage_seconds_total[1h])	PASS
deriv(demo_disk_usage_bytes[1s])	PASS
deriv(demo_disk_usage_bytes[15s])	PASS
deriv(demo_disk_usage_bytes[1m])	PASS
deriv(demo_disk_usage_bytes[5m])	PASS
deriv(demo_disk_usage_bytes[15m])	PASS

Query	Outcome
deriv(demo_disk_usage_bytes[1h])	PASS
predict_linear(demo_disk_usage_bytes[1s], 600)	PASS
predict_linear(demo_disk_usage_bytes[15s], 600)	PASS
predict_linear(demo_disk_usage_bytes[1m], 600)	PASS
predict_linear(demo_disk_usage_bytes[5m], 600)	PASS
predict_linear(demo_disk_usage_bytes[15m], 600)	PASS
predict_linear(demo_disk_usage_bytes[1h], 600)	PASS
time()	PASS
label_replace(demo_num_cpus, "job", "destination-value-$1", "instance", "demo.promlabs.com:(.*)")	PASS
label_replace(demo_num_cpus, "job", "destination-value-$1", "instance", "host:(.*)")	PASS
label_replace(demo_num_cpus, "job", "$1-$2", "instance", "local(.*):(.*)")	PASS
label_replace(demo_num_cpus, "job", "value-$1", "nonexistent-src", "source-value-(.*)")	PASS
label_replace(demo_num_cpus, "job", "value-$1", "nonexistent-src", "(.*)")	PASS
label_replace(demo_num_cpus, "job", "value-$1", "instance", "non-matching-regex")	PASS
label_replace(demo_num_cpus, "job", "", "dst", ".*")	PASS
label_replace(demo_num_cpus, "job", "value-$1", "src", "(.*")	PASS
label_replace(demo_num_cpus, "~invalid", "", "src", "(.*)")	PASS
label_replace(demo_num_cpus, "instance", "", "", "")	PASS
label_join(demo_num_cpus, "new_label", "-", "instance", "job")	PASS
label_join(demo_num_cpus, "job", "-", "instance", "job")	PASS
label_join(demo_num_cpus, "job", "-", "instance")	PASS

Query	Outcome
label_join(demo_num_cpus, "~invalid", "-", "instance")	PASS
day_of_month()	PASS
day_of_week()	PASS
days_in_month()	PASS
hour()	PASS
minute()	PASS
month()	PASS
year()	PASS
day_of_month(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
day_of_month(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
day_of_month(demo_batch_last_success_timestamp_seconds offset 10m)	PASS
day_of_week(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
day_of_week(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
day_of_week(demo_batch_last_success_timestamp_seconds offset 10m)	PASS
days_in_month(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
days_in_month(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
days_in_month(demo_batch_last_success_timestamp_seconds offset 10m)	PASS
hour(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
hour(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
hour(demo_batch_last_success_timestamp_seconds offset 10m)	PASS

Query	Outcome
minute(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
minute(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
minute(demo_batch_last_success_timestamp_seconds offset 10m)	PASS
month(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
month(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
month(demo_batch_last_success_timestamp_seconds offset 10m)	PASS
year(demo_batch_last_success_timestamp_seconds offset 1m)	PASS
year(demo_batch_last_success_timestamp_seconds offset 5m)	PASS
year(demo_batch_last_success_timestamp_seconds offset 10m)	PASS
idelta(demo_cpu_usage_seconds_total[1s])	PASS
idelta(demo_cpu_usage_seconds_total[15s])	PASS
idelta(demo_cpu_usage_seconds_total[1m])	PASS
idelta(demo_cpu_usage_seconds_total[5m])	PASS
idelta(demo_cpu_usage_seconds_total[15m])	PASS
idelta(demo_cpu_usage_seconds_total[1h])	PASS
irate(demo_cpu_usage_seconds_total[1s])	PASS
irate(demo_cpu_usage_seconds_total[15s])	PASS
irate(demo_cpu_usage_seconds_total[1m])	PASS
irate(demo_cpu_usage_seconds_total[5m])	PASS
irate(demo_cpu_usage_seconds_total[15m])	PASS
irate(demo_cpu_usage_seconds_total[1h])	PASS
clamp_min(demo_memory_usage_bytes, 2)	PASS

Query	Outcome
clamp_max(demo_memory_usage_bytes, 2)	PASS
clamp(demo_memory_usage_bytes, 0, 1)	FAIL
clamp(demo_memory_usage_bytes, 0, 1000000000000)	FAIL
clamp(demo_memory_usage_bytes, 1000000000000, 0)	FAIL
clamp(demo_memory_usage_bytes, 1000000000000, 1000000000000)	FAIL
resets(demo_cpu_usage_seconds_total[1s])	PASS
resets(demo_cpu_usage_seconds_total[15s])	PASS
resets(demo_cpu_usage_seconds_total[1m])	PASS
resets(demo_cpu_usage_seconds_total[5m])	PASS
resets(demo_cpu_usage_seconds_total[15m])	PASS
resets(demo_cpu_usage_seconds_total[1h])	PASS
changes(demo_batch_last_success_timestamp_seconds[1s])	PASS
changes(demo_batch_last_success_timestamp_seconds[15s])	PASS
changes(demo_batch_last_success_timestamp_seconds[1m])	PASS
changes(demo_batch_last_success_timestamp_seconds[5m])	PASS
changes(demo_batch_last_success_timestamp_seconds[15m])	PASS
changes(demo_batch_last_success_timestamp_seconds[1h])	PASS
vector(1.23)	PASS
vector(time())	PASS
histogram_quantile(-0.5, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS

Query	Outcome
histogram_quantile(0.1, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(0.5, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(0.75, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(0.95, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(0.90, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(0.99, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(1, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(1.5, rate(demo_api_request_duration_seconds_bucket[1m]))	PASS
histogram_quantile(0.9, nonexistent_metric)	PASS
histogram_quantile(0.9, demo_memory_usage_bytes)	PASS
histogram_quantile(0.9, {__name__=~"demo_api_request_duration_seconds_.+"})	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.1, 0.1)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.1, 0.5)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.1, 0.8)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.5, 0.1)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.5, 0.5)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.5, 0.8)	PASS

Query	Outcome
holt_winters(demo_disk_usage_bytes[10m], 0.8, 0.1)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.8, 0.5)	PASS
holt_winters(demo_disk_usage_bytes[10m], 0.8, 0.8)	PASS
count_values("value", demo_api_request_duration_seconds_bucket)	PASS
absent(demo_memory_usage_bytes)	PASS
absent(nonexistent_metric_name)	PASS
max_over_time((time() - max(demo_batch_last_success_timestamp_seconds) < 1000)[5m:10s] offset 5m)	PASS
avg_over_time(rate(demo_cpu_usage_seconds_total[1m])[2m:10s])	PASS

3.产品计费

3.1. 按量付费

阿里云Prometheus监控产品专家版支持按量后付费模式。本文介绍专家版公共云、金融云、政务云的按量计费详情。

背景信息

Prometheus监控专家版按量后付费按照指标的采样点数据量和指标存储时长收费。具体详情，请参见整体收费说明。

专家版收费说明

Prometheus监控试用版到期后，如需继续使用，您需要开通专家版。如何开通专家版，请参见开通方式。

指标类型	上报指标采样点数量费用	存储费用
基础指标	免费	收费（15天内免费）
自定义指标	收费，具体请参见专家版公共云收费标准、专家版金融云收费标准和专家版政务云收费标准。	收费（15天内免费）

> 注意
>
> - 每条上报指标最大不能超过2 KB。
> - 每条指标默认免费存储15天，如需存储更长时间，您可以在**存储时长**页面自行设置**存储天数**。具体操作，请参见存储时长设置。
> - 目前Prometheus监控支持公共云、金融云、政务云的云服务类型，具体信息请参见开服地域 。不同云服务类型的Prometheus监控专家版的收费标准不同，具体请参见专家版公共云收费标准、专家版金融云收费标准和专家版政务云收费标准。
> - 您可以在Prometheus控制台的Prometheus监控任务列表页面，单击右上角的**资源消耗统计**查看Prometheus监控昨日的资源消耗情况，具体操作，请参见查看资源消耗情况。
> - 您可以在用户中心查看Prometheus监控服务的总体消费情况，具体操作，请参见查看费用账单。

专家版公共云收费标准

中国内地（大陆）地域按量收费标准如下

- 上报指标采样点数量费用：上报自定义指标收费标准根据每天上报的指标数量范围，按照阶梯式递减方式进行累加计算，如下表所示。上报指标采样点数量费用标准

日上报指标数范围（百万条）	单价（元/百万条）	日付费范围（元）
0~50	0.8	0~40
50~150	0.65	40~105
150~300	0.55	105~187.5

日上报指标数范围（百万条）	单价（元/百万条）	日付费范围（元）
300~600	0.45	187.5~322.5
600~1200	0.35	322.5~532.5
1200以上	0.25	532.5以上

- 存储指标费用：15天内免费，默认设置**存储天数**为15天，若手动变更**存储天数**超出15天，根据每天的指标上报数量，以每天存储每百万条指标收费0.01元进行累计计算。

> **说明**
> 以华东1（杭州）地域为例，假设平均每天上报的指标采样点数量是300百万条，其中200百万条采样点归属于基础指标，100百万条采样点归属于自定义指标，这300百万指标需要分别存储15天和90天。
>
> - 指标存储15天，每天的费用计算逻辑为：总费用=上报指标数量收费+存储收费，即(50*0.8+(100-50)*0.65)+0=72.5元
> - 指标存储90天，每天的费用计算逻辑为：总费用=上报指标数量收费+存储收费，即(50*0.8+(100-50)*0.65)+[300*0.01*(90-15)]=297.5元。

中国香港和海外地域按量收费标准如下

- 上报指标采样点数量费用：如下表所示。上报指标采样点数量费用标准

日上报指标数范围（百万条）	单价（元/百万条）	日付费范围（元）
0~50	1.12	0~56
50~150	0.91	56~147
150~300	0.77	147~262.5
300~600	0.63	262.5~451.5
600~1200	0.49	451.5~745.5
1200以上	0.35	745.5以上

- 存储指标费用：15天内免费，默认设置**存储天数**为15天，若手动变更**存储天数**超出15天，根据每天的指标上报数量，以每天存储每百万条指标收费0.01元进行累计计算。

> **说明** 假设首日上报的指标采样点数量是150百万，每天的费用计算逻辑为：(50*1.12+(150-50)*0.91)=147元。

专家版金融云收费标准

Prometheus监控专家版金融云上报自定义指标收费标准根据每天上报的指标数量范围，按照阶梯式递减方式进行累加计算。金融云华东1（杭州）、华东2（上海）和华南1（深圳）地域的收费标准如下。

- 上报指标采样点数量费用：如下表所示。上报指标采样点数量费用标准

单日消费范围（百万条）	单价（元/百万条）	日付费范围（元）
0~50	1.6	0~80
50~150	1.3	80~210
150~300	1.1	210~375
300~600	0.9	375~645
600~1200	0.7	645~1065
1200以上	0.5	1065以上

- 存储指标费用：15天内免费，默认设置**存储天数**为15天，若手动变更**存储天数**超出15天，根据每天的指标上报数量，以每天存储每百万条指标收费0.01元进行累计计算。

> **说明** 假设首日上报的指标采样点数量是150百万，每天的费用计算逻辑为：(50*1.6+(150-50)*1.3)+0=210元。

专家版政务云收费标准

Prometheus监控专家版政务云上报自定义指标收费标准根据每天上报的指标数量范围，按照阶梯式递减方式进行累加计算。政务云华北2（北京）地域的收费标准如下。

- 上报指标采样点数量费用：如下表所示。上报指标采样点数量费用标准

单日消费范围（百万条）	单价（元/百万条）	日付费范围（元）
0~50	1.2	0~60
50~150	0.975	60~157.5
150~300	0.825	157.5~281.25
300~600	0.675	281.25~483.75
600~1200	0.525	483.75~798.75
1200以上	0.375	798.75以上

- 存储指标费用：15天内免费，默认设置**存储天数**为15天，若手动变更**存储天数**超出15天，根据每天的指标上报数量，以每天存储每百万条指标收费0.01元进行累计计算。

> **说明** 假设首日上报的指标采样点数量是150百万，每天的费用计算逻辑为：(50*1.2+(150-50)*0.975)+0=157.5元。

相关文档

- 基础指标说明
-
- 查看账单

-
- 开服地域
-

3.2. 查看账单

本文介绍如何查看Prometheus监控的费用账单、资源消耗情况以及废弃指标的计费情况。

查看费用账单

1. 登录用户中心。
2. 在左侧菜单栏选择**账单管理 > 账单详情**。
3. 在**账期流水**页签选择**账期**、**账号/Owner账号**，输入**订单/账单号**等过滤条件，完成后单击页面左侧的**搜索**。
4. 在账单列表中单击**产品**右侧的图标，选择**应用实时监控服务**，然后单击**产品明细**右侧的图标，选择**Prometheus**，查看Prometheus监控的消费账单。

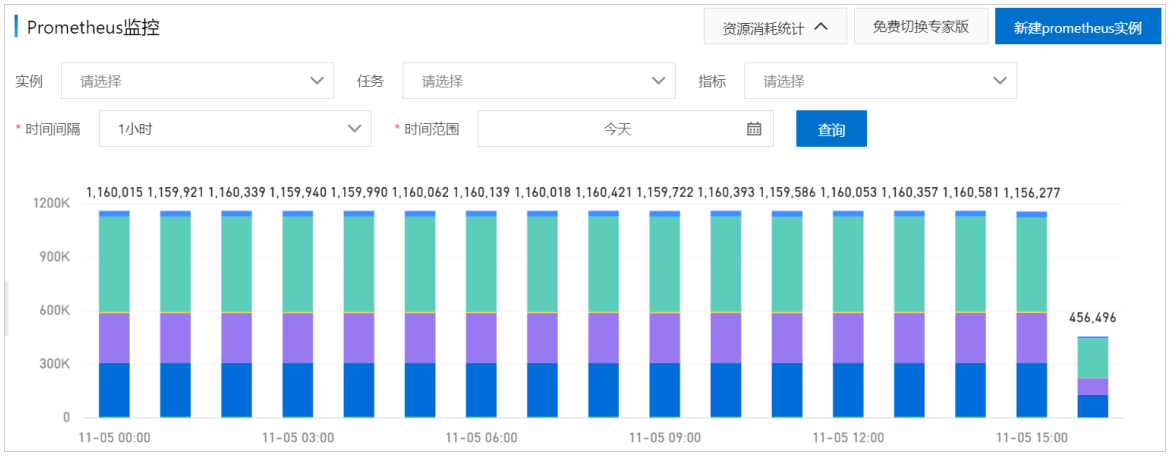
   单击**消费类型**、**账单类型**和**支付状态**右侧的图标，可进一步筛选您的消费账单。

查看资源消耗情况

Prometheus监控控制台为已开通专家版的用户提供查看资源消耗的功能，您可以通过设置过滤条件查看单个或全部Prometheus实例在特定时间内的自定义指标资源消耗情况。

查看全部Prometheus实例资源消耗情况

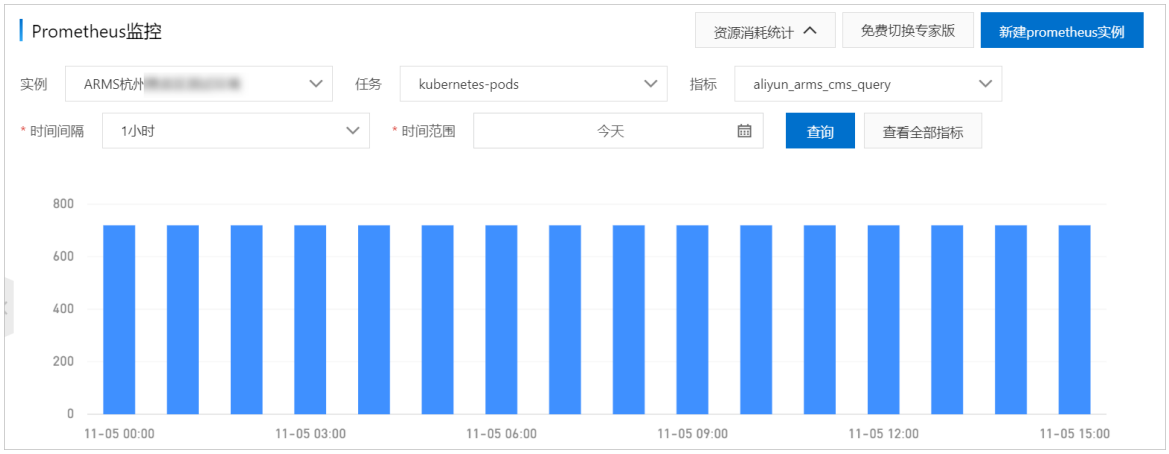
1. 登录Prometheus控制台。
2. 在Prometheus监控页面单击**资源消耗统计**。
3. 在**资源消耗统计**展开区域选择**时间间隔**和**时间范围**的过滤条件，完成后单击**查询**，即可看到您全部Prometheus实例的自定义指标资源消耗情况。



> **说明**
>
> - 图中纵坐标为上报的自定义指标数。
> - 将鼠标放在任一时段的统计图上，会显示当前时段消耗的自定义指标总数及单个Prometheus实例消耗的自定义指标数。

查看单个Prometheus实例资源消耗情况

1. 登录Prometheus控制台。
2. 在Prometheus监控页面单击**资源消耗统计**。
3. 在**资源消耗统计**展开区域选择**实例**、**任务**、**指标**、**时间间隔**和**时间范围**过滤条件，完成后单击**查询**，即可看到您选定的单个Prometheus实例自定义指标资源消耗情况。



> **说明**
> - 图中纵坐标为上报的自定义指标数。
> - 将鼠标放在任一时段的统计图上，会显示当前时段消耗的自定义指标总数及单个Prometheus实例消耗的自定义指标数。

4. 单击**查询**右侧的**查看全部指标**，可在**设置**页面的**指标**页签查看该Prometheus实例的全部指标情况，具体操作，请参见查看容器服务类型的Prometheus实例指标。

配置废弃指标

当您的Prometheus实例不再需要监控某些自定义指标的数据时，为了避免这些自定义指标继续产生费用，您可以废弃这些自定义指标。已经配置为废弃的自定义指标将不再继续产生费用。

> **说明** 云服务类型的Prometheus实例不支持配置废弃指标操作。

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后单击Prometheus实例名称。
3. 在左侧导航栏单击**服务发现**。
4. 在右侧页面，单击**指标**页签，然后配置废弃指标：
   - 配置单条废弃指标
     a. 在需要废弃的指标的右侧**操作**列，单击**废弃**。
     b. 在弹出的**配置废弃指标**对话框中，单击**确定**。
   - 配置多条废弃指标
     a. 单击页面右上角的**配置废弃指标**。
     b. 在弹出的**配置废弃指标**对话框中，输入要废弃的指标的名称，然后单击**确定**。
   - 废弃全部指标
     a. 单击页面右上角的**全部废弃**

a. 单击页面左下角的**全部废弃**。

b. 在弹出的确认对话框中，单击**确定**。

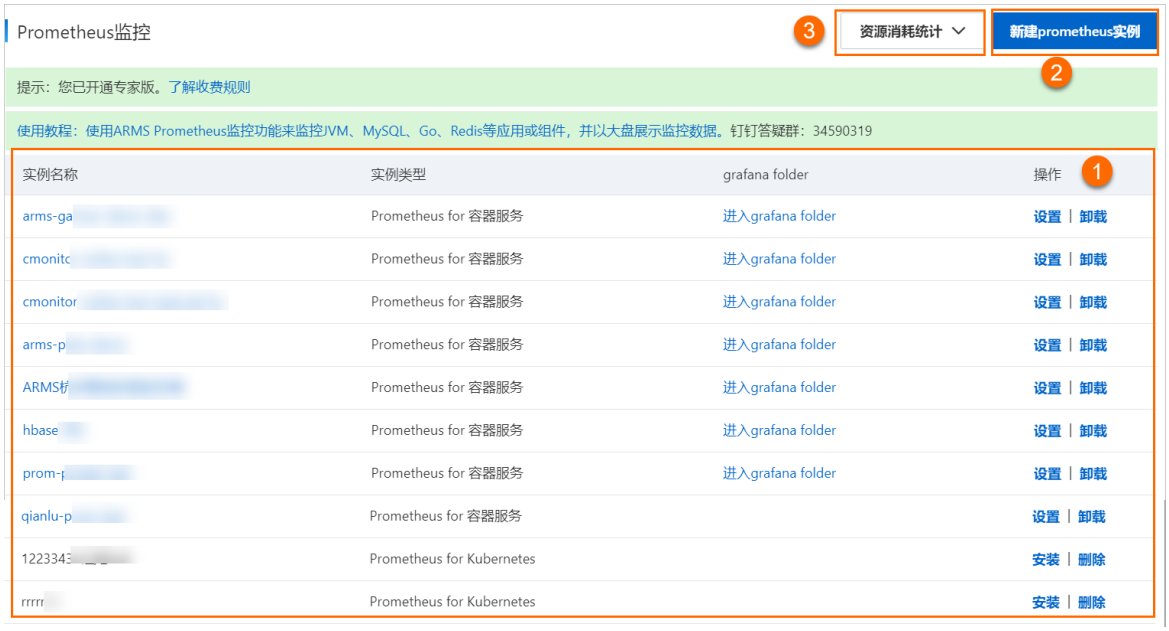
> **说明** 对于自建的Kubernetes集群，配置废弃指标的具体操作，请参见配置废弃指标。

4.创建Prometheus实例

4.1. 查看Prometheus实例

本文介绍如何查看已创建的Prometheus实例。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，查看当前地域已创建的所有Prometheus实例。



区域	参数/功能	说明
	实例名称	创建的Prometheus实例名称。

区域	参数/功能	说明
①	**实例类型**	Prometheus监控的实例类型，包括： ○ Prometheus for 容器服务：适合需要对阿里云容器服务集群及其上面运行的应用进行一体化监控告警场景。如何创建该实例的具体操作，请参见Prometheus实例 for 容器服务。 ○ Prometheus for VPC：适合需要在阿里云VPC内（通常为ECS集群）进行Prometheus监控的场景。如何创建该实例的具体操作，请参见Prometheus实例 for VPC。 ○ Prometheus for 云服务：适合需要监控多种阿里云云产品数据的场景。如何创建该实例的具体操作，请参见Prometheus实例 for 云服务。 ○ Prometheus for Kubernetes：适合需要对自建Kubernetes集群及其上面运行的应用的一体化监控场景。如何创建该实例的具体操作，请参见Prometheus实例 for Kubernetes。 ○ Prometheus for Remote Write：适合已自建了Prometheus Server，但是需要通过Remote Write的方式来解决Prometheus存储的可用性和可扩展性的场景。同时支持将分散的Prometheus实例采集的指标统一写到一个数据源场景。如何创建该实例的具体操作，请参见Prometheus实例 for Remote Write。
	grafana folder	系统预置的Grafana大盘。
	操作	○ **设置**：管理Prometheus实例的运维操作，包括指标管理、服务发现、探针设置等。 ○ **卸载**：卸载Prometheus实例插件。Prometheus实例卸载后，将不会显示在当前Prometheus监控实例列表页面。 ○ **安装**：安装Kubernetes类型的实例。 ○ **删除**：删除Kubernetes类型的实例。
②	**新建prometheus实例**	您可以创建不同类型的Prometheus实例。Prometheus实例的类型有5种，具体请参见什么是Prometheus实例。
③	**资源消耗统计**	您可以查看资源消耗统计情况。具体操作，请参见查看资源消耗情况。

4.2. Prometheus实例 for 容器服务

本文说明如何创建Prometheus实例 for 容器服务，即如何将容器服务Kubernetes版集群接入Prometheus监控，从而使用预定义的大盘监控主机和Kubernetes集群的众多性能指标。

前提条件

- 已创建容器服务Kubernetes版集群。具体操作，请参见：
  - 创建Kubernetes托管版集群
  - 创建Kubernetes专有版集群

- 创建Serverless Kubernetes集群

- 已开通ARMS。具体操作，请参见开通ARMS。

开启阿里云Prometheus监控

您可以通过以下两种方式开启阿里云Prometheus监控：

通过配置集群参数开启Prometheus监控

1. 登录容器服务管理控制台或者Prometheus控制台。具体操作如下：
   - 功能入口一：
     a. 登录容器服务管理控制台。
     b. 在控制台左侧导航栏中，单击**集群**。
   - 功能入口二：
     a. 登录Prometheus控制台。
     b. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建Prometheus实例**。
     c. 在**新建Prometheus实例**页面，单击**Prometheus实例 for 容器服务**区域。
     d. 在**接入容器服务Kubernetes集群**面板的目标集群右侧**操作**列，单击**安装**。然后在弹出的对话框中单击**确认**或者**安装ack-arms-prometheus**。
2. 在**集群列表**页面中，单击页面右上角的**创建集群**。
3. 选择目标集群模板，然后配置创建集群参数。在**组件配置**页面，选中**使用Prometheus监控服务**。

有关创建集群的具体步骤，请参见创建Kubernetes托管版集群。

> **说明** 在创建集群时，系统默认选中**使用Prometheus监控服务**。

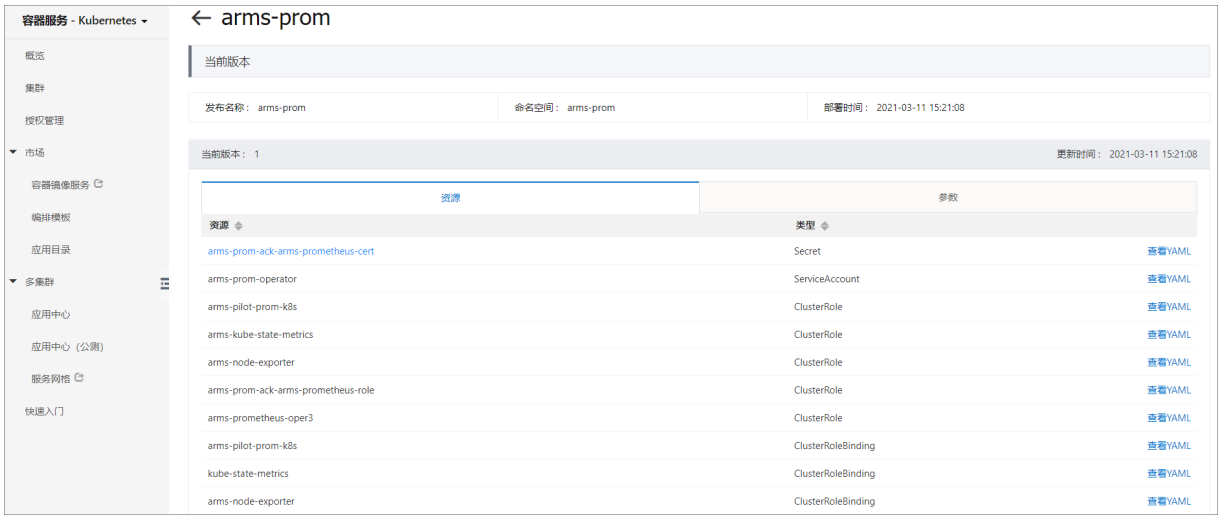
集群创建完成后，系统将自动配置阿里云Prometheus监控服务。

通过ACK控制台Prometheus监控开启Prometheus监控

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理左侧导航栏中，选择**运维管理 > Prometheus监控**。
5. 在**Prometheus监控**页面中间，单击**开始安装**。

执行结果

安装完成后会自动跳转到应用**arms-prom**页面，可查看应用信息。



为容器服务Kubernetes版某一集群开启阿里云Prometheus监控后，在**Prometheus监控**页面系统会自动创建**监控对象类型**为K8s集群的Prometheus实例。

通过Grafana大盘查看目标集群的监控指标

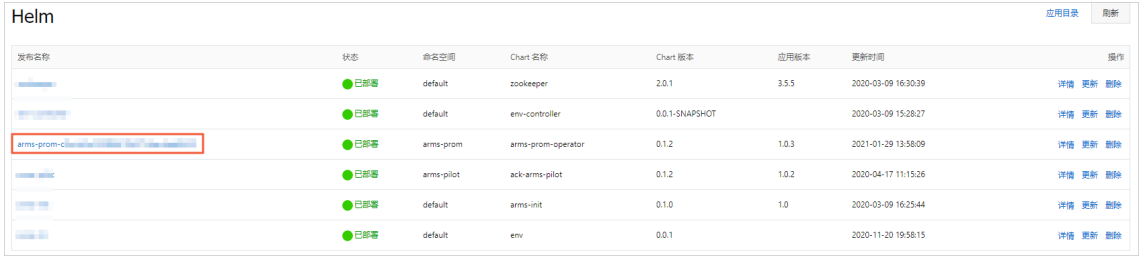
将容器服务Kubernetes版集群接入Prometheus监控后，可以通过预置的Grafana大盘查看目标集群的性能指标数据。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击目标Prometheus实例名称。
3. 在**大盘列表**页面，单击大盘**名称**，查看目标集群的性能指标数据。

停止监控Kubernetes集群

如需停止使用Prometheus监控对Kubernetes集群进行监控，请按照以下步骤卸载Prometheus监控插件。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，找到要卸载的K8s集群，然后在其右侧**操作**列，单击**卸载**。
3. 在弹出的**确认**对话框，单击**确认**。
4. 登录容器服务管理控制台。
5. 在左侧导航栏单击**集群**，然后单击需停止监控的集群名称。
6. 在左侧导航栏选择**应用 > Helm**，并根据情况采取以下任一操作：
   - 如果Helm页面没有`arms-prom-****`记录，则说明监控插件卸载成功，您无需采取任何操作。
   - 如果Helm页面有`arms-prom-****`记录，请在其右侧的**操作**列中单击**删除**。



4.3. Prometheus实例 for 云服务

本文说明如何创建Prometheus实例 for 云服务，即如何将阿里云云服务接入Prometheus监控，并使用内置的云服务大盘和报警规则实现对云服务的监控和报警。目前已经支持ECS、MongoDB、Redis、OSS、RDS、NAT、SLB、RocketMQ、Kafka、EIP、Elasticsearch、DRDS、PolarDB、Logstash、E-MapReduce这15款阿里云云服务的一键接入。

前提条件

- 已开通ARMS。具体操作，请参见开通ARMS。
- 已开通Prometheus监控。具体操作，请参见开通Prometheus监控专家版。

背景信息

云监控（CloudMonitor）是一项针对阿里云资源和互联网应用进行监控的服务。更多信息，请参见云监控。

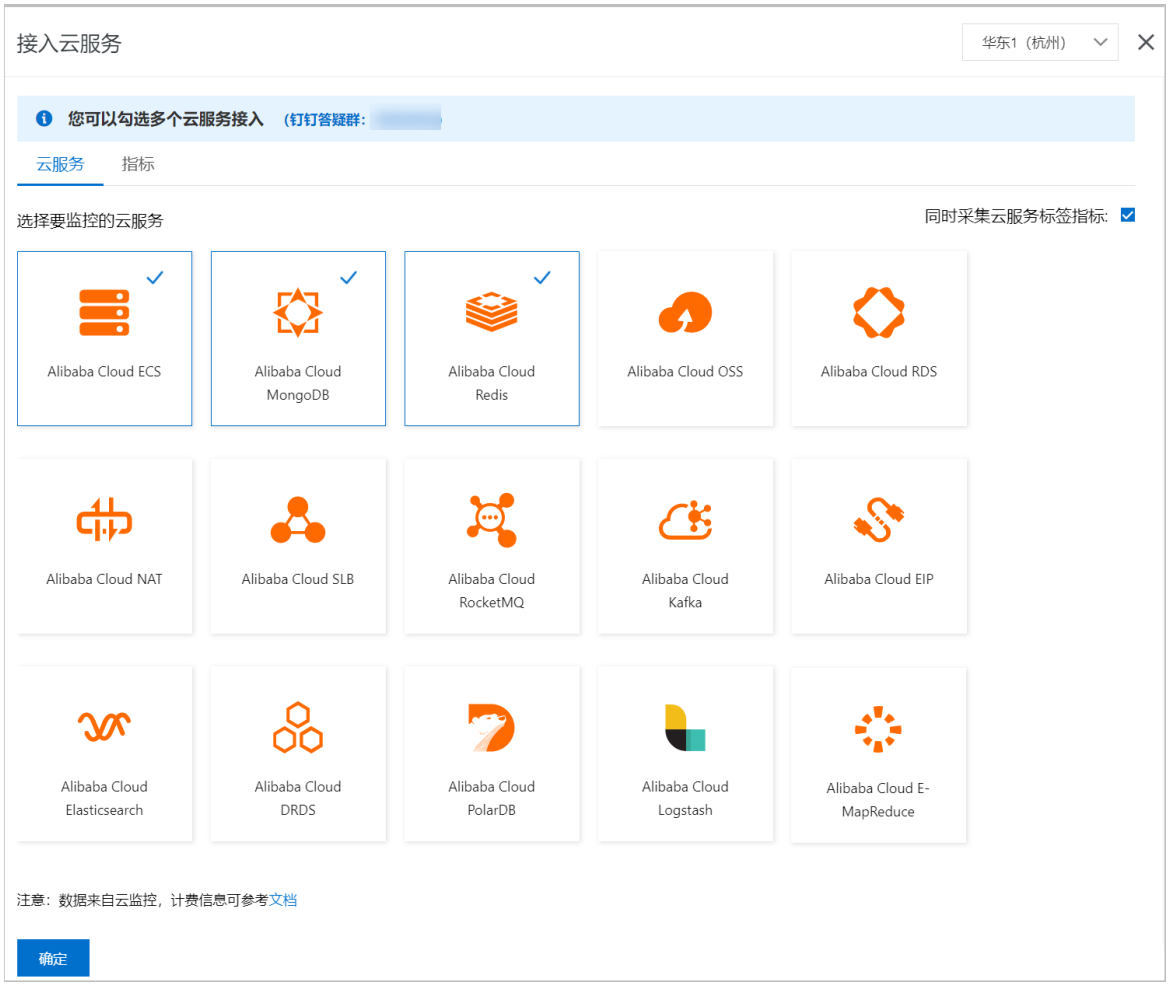
阿里云Prometheus监控集成云监控后，云监控以及采集的监控数据均不会额外收费。

功能入口

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建Prometheus实例**。
3. 在**新建Prometheus实例**页面，单击**Prometheus实例 for 云服务**区域。
   接入云服务面板显示了当前地域下支持接入的所有阿里云云服务。

创建云服务类型的Prometheus实例

1. 在**云服务**页签中选择需要监控的云服务，然后单击**确定**。

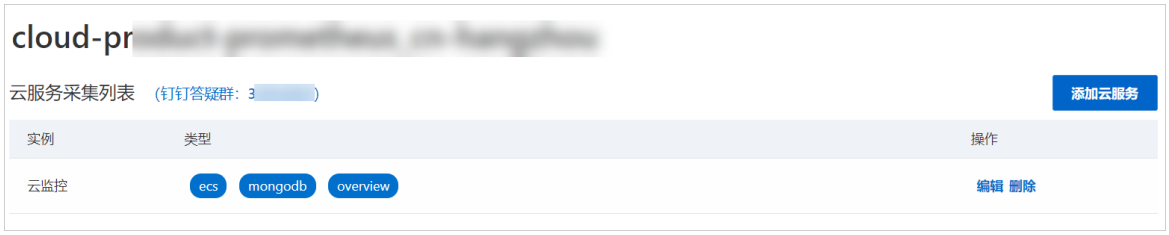


系统会自动在**Prometheus监控**页面创建实例类型为Prometheus for 云服务的Prometheus实例。

> **说明**
>
> - 创建云服务类型的Prometheus实例时，在**云服务**页签右上角会自动勾选**同时采集云服务标签指标**，勾选后采集的云服务监控指标会关联上配置的标签。若您不需要监控指标关联标签，可自行取消勾选。
> - 云服务类型的Prometheus实例只能创建一个，因此在**Prometheus监控**页面始终仅显示一条实例类型为云服务的Prometheus实例。

创建云服务类型的Prometheus实例后，控制台自动跳转至**云服务采集列表**页面，并显示接入的云服务信息。

如果需要调整云服务接入，您可以在**云服务采集列表**页面增加或删除云服务。具体操作，请参见管理云服务监控接入。



> **说明** 调整云服务接入会自动变更预置的大盘和报警配置功能。

通过Grafana大盘查看云服务的监控指标

将阿里云云服务接入Prometheus监控后，可以通过预置的Grafana大盘查看云服务的性能指标数据。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域。
3. 单击云服务类型的Prometheus实例名称，在**大盘列表**页面单击大盘**名称**，查看云服务的性能指标数据。

停止监控云服务

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，找到要卸载的云服务，然后在其右侧**操作**列，单击**卸载**。
3. 在弹出的**确认**对话框，单击**确认**。
   卸载完毕后，**Prometheus监控**页面不再显示该云服务类型的Prometheus实例。同时在**云服务采集列表**页面也不再显示云服务信息。

4.4. Prometheus实例 for VPC

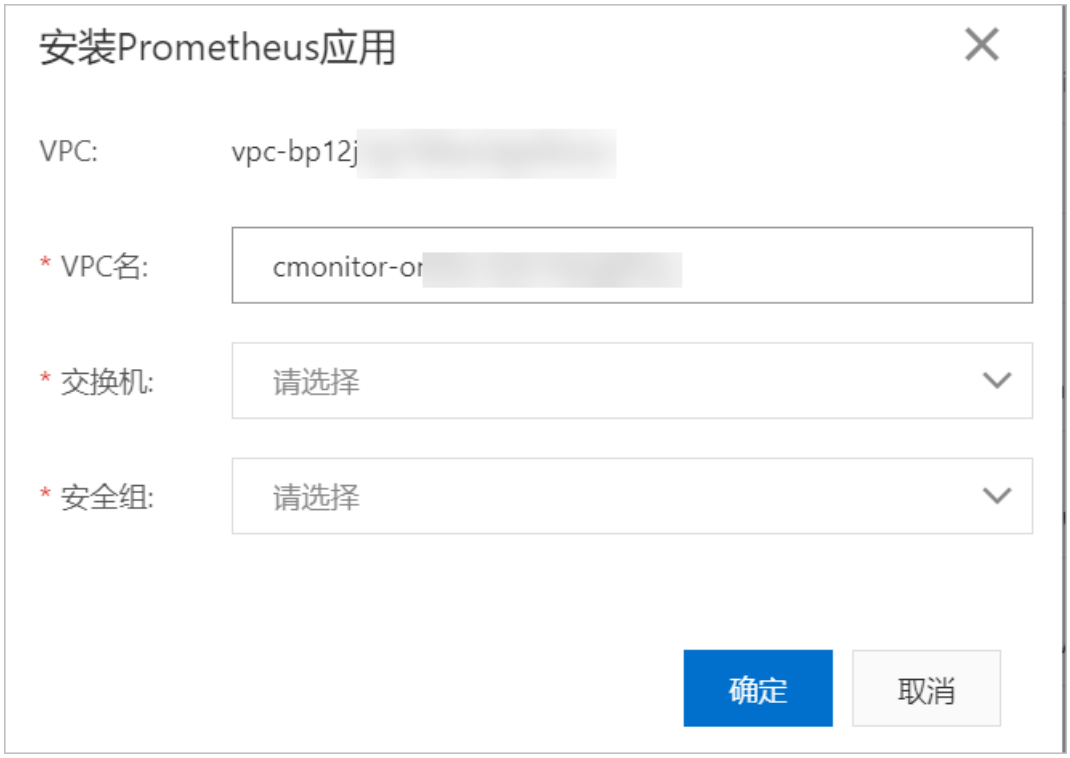
本文说明如何创建Prometheus实例 for VPC，即如何将VPC网络的ECS集群接入Prometheus监控，并创建大盘监控ECS集群的众多性能指标。

前提条件

- 已开通ARMS。具体操作，请参见开通ARMS。
- 已创建VPC网络和ECS实例。具体操作，请参见搭建IPv4专有网络。

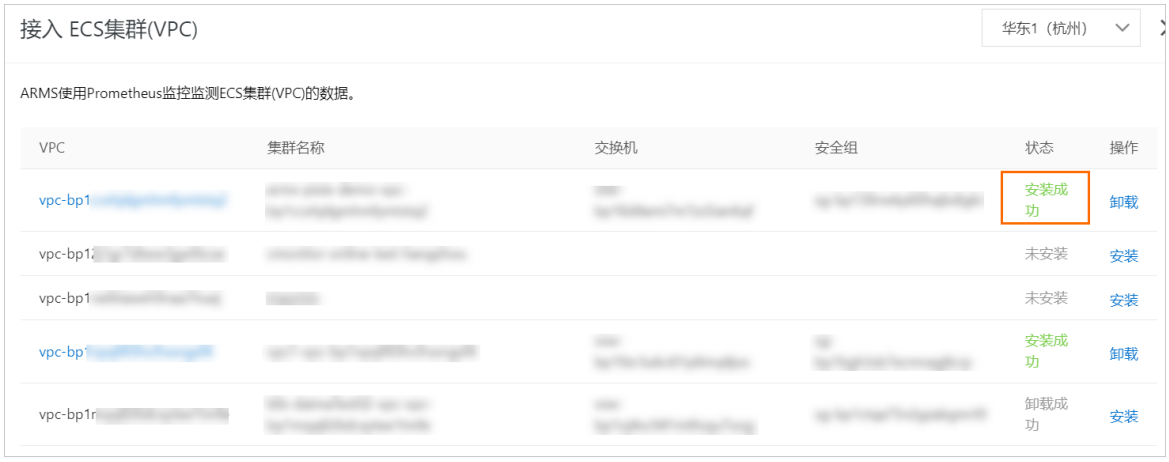
开启阿里云Prometheus监控

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建Prometheus实例**。
3. 在**新建Prometheus实例**页面，单击**Prometheus实例 for VPC**区域。
   在**接入 ECS集群(VPC)**面板显示当前地域下的所有VPC列表。
4. 在**接入 ECS集群(VPC)**面板的目标VPC右侧**操作**列，单击**安装**。
5. 在**安装Prometheus应用**对话框中，输入VPC名称，选择交换机和安全组，然后单击**确定**。



- VPC名：自定义VPC显示名称。
- 交换机：选择需要监控的ECS实例所在的交换机。
- 安全组：选择需要监控的ECS实例所在的安全组。

安装成功后，对应VPC右侧**状态**列显示安装成功。



创建大盘查看目标ECS集群的监控指标

将VPC网络的ECS集群接入Prometheus监控后，可以通过自定义Grafana大盘查看目标ECS集群的性能指标数据。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域。

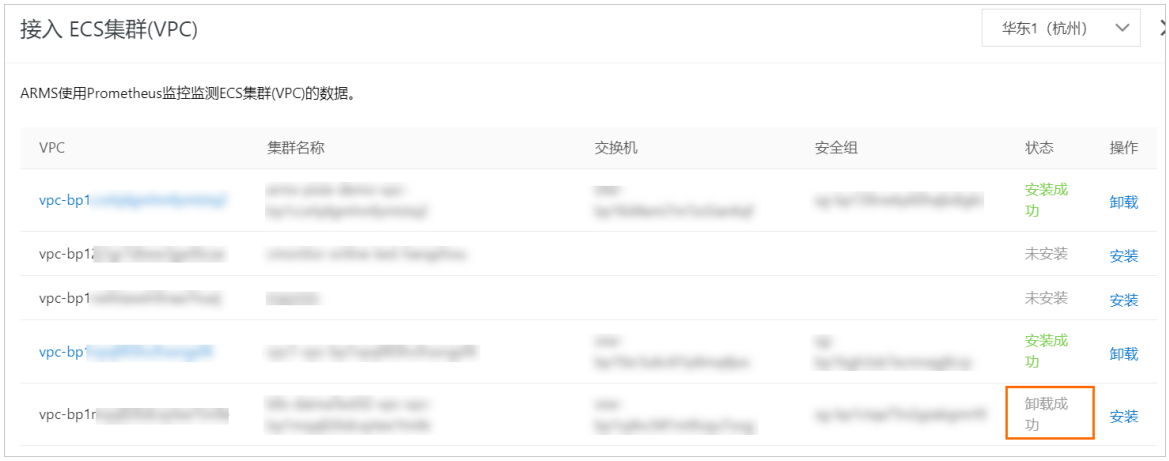
   Prometheus监控页面显示了所有接入Prometheus监控的集群。
3. 单击VPC类型的Prometheus实例名称。
4. 在**大盘列表**页面右上角单击**创建大盘**。

5. 根据需求在Grafana控制台创建大盘。具体操作，请参见Grafana文档。

停止监控ECS集群

如需停止使用Prometheus监控对ECS集群进行监控，请按照以下步骤卸载Prometheus监控插件。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建Prometheus实例**。
3. 在**新建Prometheus实例**页面，单击**Prometheus实例 for VPC**区域。
4. 在**接入 ECS集群(VPC)**面板的目标VPC右侧**操作**列，单击**卸载**。
5. 在弹出的**提示**对话框中，单击**确定**。
   卸载成功后，对应VPC右侧**状态**列显示卸载成功。



4.5. Prometheus实例 for Kubernetes

本文说明如何创建Prometheus实例 for Kubernetes，即如何将自建或非阿里云容器服务的Kubernetes集群接入Prometheus监控，从而使用预定义的大盘监控主机和Kubernetes集群的众多性能指标。

前提条件

已开通ARMS。具体操作，请参见开通ARMS。

> **说明**
> 如果您的Kubernetes集群已接入阿里云内网，请参考本文将Kubernetes集群接入Prometheus监控；如果您的Kubernetes集群为公网集群，集群接入Prometheus监控的操作，请参见公网Kubernetes集群接入Prometheus监控。

创建Prometheus实例 for Kubernetes

创建Prometheus实例 for Kubernetes，即将自建或非阿里云的Kubernetes集群接入Prometheus监控，需要安装Prometheus Agent。具体操作如下：

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建Prometheus实例**。
3. 在**新建Prometheus实例**页面，单击**Prometheus实例 for Kubernetes**区域。
4. 在**接入自建Kubernetes集群**面板右上角选择Kubernetes集群需要接入的地域，然后完成以下操作：
   i. 自定义Prometheus监控实例的名称，然后单击**新建**。

ii. 执行以下命令添加阿里云的Helm Repository。

> **注意** 不同地域添加阿里云的Helm Repository的命令不同，请根据实际地域替换命令中的 `{region_id}`，或直接在**接入自建Kubernetes集群**面板获取准确的添加命令。

```
helm repo add aliyun http://aliacs-k8s-{region_id}.oss-{region_id}.aliyuncs.com/app/charts-incubator/
```

iii. 执行**安装Prometheus探针**区域的命令为自建Kubernetes集群安装探针。

```
helm install arms-prom-operator aliyun/ack-arms-prometheus \
  --namespace arms-prom \
  --set controller.cluster_id=$CLUSTER_ID \     //请在安装Prometheus探针区域获取集群ID。
  --set controller.uid="***" \                  //请在安装Prometheus探针区域获取UID。
  --set controller.region_id=*** \              //请在安装Prometheus探针区域获取Region ID。
  --set controller.vpc_prefix=registry.         //从公网拉取镜像。如果您的镜像存储在阿里云内网，则可以不用配置此参数。
```

> **说明** 关于Helm命令的参数说明，请参见Helm命令参数说明。

自建或非阿里云的Kubernetes集群接入Prometheus监控成功后，**Prometheus监控**页面将会显示接入的自建Kubernetes集群。

查看Prometheus监控指标

将自建或非阿里云的Kubernetes集群接入Prometheus监控成功后，可以通过Grafana大盘查看Prometheus监控数据。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击目标自建Kubernetes集群名称。
3. 在**大盘列表**页面单击需要查看的大盘。

> **说明** 关于Grafana大盘的说明，请参见基础大盘说明。

停止监控自建Kubernetes集群

如需停止使用Prometheus监控对自建Kubernetes集群进行监控，请按照以下步骤卸载Prometheus监控插件。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，找到要卸载监控插件的Kubernetes集群，然后在其右侧**操作**列，单击**卸载**，并在弹出的**确认**对话框单击**确认**。
   卸载插件完毕后，**Prometheus监控**页面不再显示该自建Kubernetes集群。

相关文档

- 公网Kubernetes集群接入Prometheus监控
-

4.6. Prometheus实例 for Remote Write

阿里云Prometheus监控的Remote Write功能支持作为远程数据库存储Prometheus监控数据。本文将介绍如何创建Prometheus实例 for Remote Write，即如何使用阿里云Prometheus监控的Remote Write对接自建Prometheus，构建监控数据的高效存储方案。

（可选）

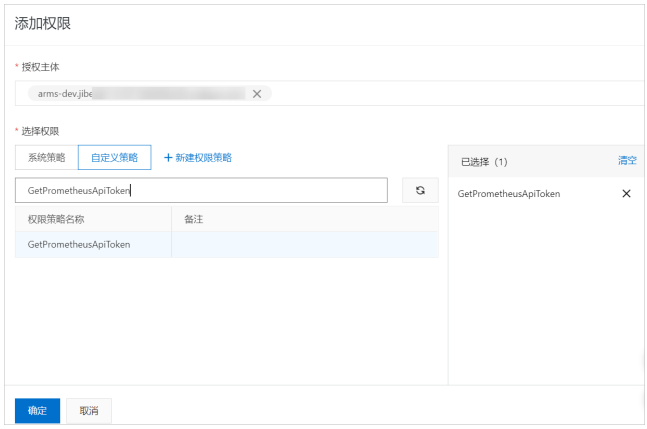
（可选）步骤一：为RAM用户授予GetPrometheusApiToken接口调用权限

自建Prometheus写入阿里云Prometheus监控时需要调用GetPrometheusApiToken接口。阿里云账号（主账号）默认支持调用GetPrometheusApiToken接口。如果您是RAM用户（子账号），则需要先使用阿里云账号为RAM用户授予GetPrometheusApiToken接口的调用权限。

1. 使用阿里云账号（主账号）登录RAM控制台。
2. 在左侧导航栏选择**权限管理 > 权限策略**。
3. 在**权限策略**页面单击**创建权限策略**。
4. 在脚本编辑页签输入以下内容，然后单击**下一步**。

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "arms:GetPrometheusApiToken"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

5. 在**基本信息**区域输入权限策略名称，然后单击**确定**。
6. 在左侧导航栏选择**身份管理 > 用户**。
7. 在**用户**页面，单击目标RAM用户右侧操作列的**添加权限**。
8. 在**添加权限**面板的**选择权限**区域，单击**自定义策略**，然后通过搜索查找步骤4创建的权限策略，单击权限策略名称将权限策略添加至右侧**已选择**区域，然后单击**确定**。



步骤二：创建Remote Write并获取读写URL

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，在Prometheus监控页面右上角单击**新建Prometheus实例**。
3. 在**新建Prometheus实例**页面单击**Prometheus实例 for Remote Write**。
4. 自定义Prometheus监控实例的名称，然后单击**新建**。

> **说明** 单击新建后，若控制台显示报错信息，是因为您设置的实例名称已存在，请重新设置实例名称即可。

5. 复制并保存生成的Remote Read地址和Remote Write地址。

步骤三：配置Prometheus

1. 安装Prometheus，安装方法可以参考官方文档。
2. 打开Prometheus.yaml配置文件，并在文件末尾增加以下内容，将 `remote_write` 和 `remote_read` 链接替换为步骤二：创建Remote Write并获取读写URL中获取的URL，然后保存文件。

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
remote_write:
  - url: "http://ts-xxxxxxxxxxxx.hitsdb.rds.aliyuncs.com:3242/api/prom_write"
    basic_auth:
      //Username和Password需要遵循AK/SK的限制，且AK/SK需要能够调用GetPrometheusApiToken。
      username: access-key-id
      password: access-key-secret
remote_read:
  - url: "http://ts-xxxxxxxxxxxx.hitsdb.rds.aliyuncs.com:3242/api/prom_read"
    read_recent: true
```

> **说明** 自建Prometheus写入阿里云Prometheus监控时需要调用GetPrometheusApiToken接口。阿里云账号（主账号）默认支持调用GetPrometheusApiToken接口。如果您是RAM用户（子账号），则需要先使用阿里云账号为RAM用户授予GetPrometheusApiToken接口的调用权限。具体操作，请参见步骤一。

使用远程存储对接OpenTelemetry

阿里云Prometheus监控的远程存储对接OpenTelemetry之后，您可以使用Prometheus监控的远程存储功能存储OpenTelemetry的数据。

1. 业务代码进行OpenTelemetry埋点。[Demo]

```
package stat
import (
    "context"
    "fmt"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/metric"
    "time"
)
var buyCounter metric.Int64Counter
func init() {
    fmt.Println(time.Now(), " - initMetrics start......")
    meter := otel.GetMeterProvider().Meter("github.com/liguozhong/prometheus-arms-aliyu
n-go-demo")
    buyCounter = metric.Must(meter).NewInt64Counter(
        "buy_total",
        metric.WithDescription("Measures  buy"),
    )
}
func DoBuy() (string, error) {
    buyCounter.Add(context.Background(), 1)
    return "buy success", nil
}
```

2. OpenTelemetry对Meter进行初始化并和连接Prometheus。[Demo]

```
package stat
import (
    "context"
    prometheusPushExporter "go.opentelemetry.io/contrib/exporters/metric/cortex"
    prometheusExporter "go.opentelemetry.io/otel/exporters/metric/prometheus"
    "errors"
    "fmt"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/exporters/otlp"
    "go.opentelemetry.io/otel/label"
    "go.opentelemetry.io/otel/sdk/metric/controller/pull"
    "go.opentelemetry.io/otel/sdk/metric/controller/push"
    "go.opentelemetry.io/otel/sdk/metric/processor/basic"
    "go.opentelemetry.io/otel/sdk/metric/selector/simple"
    "go.opentelemetry.io/otel/sdk/resource"
    "net/http"
    "time"
    _ "net/http/pprof"
)
func InitMeter(app string, push bool) error {
    fmt.Println(time.Now(), " - initMeter start......")
    if push {
        fmt.Println(time.Now(), " - initMeter opentelemetry push......")
        remoteUrl := "http://region.arms.aliyuncs.com/prometheus/../../../../api/v3/wri
te"
        ak := "ak"
        sk := "sk"
        return initPushMeter(app, remoteUrl, ak, sk)
    }
```

```
    fmt.Println(time.Now(), " - initMeter opentelemetry pull......")
    return initPullMeter(app)
}
func initPushMeter(regionId string, remoteWriteUrl string, ak string, sk string) error
{
    fmt.Println(time.Now(), " - initPushMeter start......")
    var validatedStandardConfig = prometheusPushExporter.Config{
        Endpoint:      remoteWriteUrl,
        Name:          "AliyunConfig",
        RemoteTimeout: 30 * time.Second,
        PushInterval:  10 * time.Second,
        Quantiles:     []float64{0.5, 0.9, 0.95, 0.99},
        BasicAuth: map[string]string{
            "username": ak,
            "password": sk,
        },
    }
    if validatedStandardConfig.Endpoint == "" {
        return errors.New(" validatedStandardConfig.Endpoint==empty.regionId:" + region
Id)
    }
    fmt.Println("Success: Created Config struct")
    r, err := resource.New(context.Background(),
        resource.WithAttributes(
            label.String("cluster", "test-otel"),
            label.String("app", "buy")))
    if err != nil {
        fmt.Println("resource Error:", err)
    }
    pusher, err := prometheusPushExporter.InstallNewPipeline(validatedStandardConfig,
        push.WithPeriod(30*time.Second), push.WithResource(r))
    if err != nil {
        fmt.Println("InstallNewPipeline Error:", err)
    }
    otel.SetMeterProvider(pusher.MeterProvider())
    return nil
}
func initPullMeter(app string) error {
    fmt.Println(time.Now(), " - initPullMeter start......")
    r, err := resource.New(context.Background(),
        resource.WithAttributes(
            label.String("cluster", "test-otel"),
            label.String("app", app)))
    if err != nil {
        fmt.Println("resource Error:", err)
    }
    exporter, err := prometheusExporter.NewExportPipeline(
        prometheusExporter.Config{
            DefaultHistogramBoundaries: []float64{-0.5, 1},
        },
        pull.WithCachePeriod(0),
        pull.WithResource(r),
    )
    if err != nil {
        return err
```

```
        return err
    }
    http.HandleFunc("/opentelemetry", exporter.ServeHTTP)
    otel.SetMeterProvider(exporter.MeterProvider())
    return nil
}
func initOtlpProvider(regionId string) (*push.Controller, error) {
    exporter, err := otlp.NewExporter(
        context.Background(),
        otlp.WithInsecure(),
        otlp.WithAddress(regionId+"-intranet.arms.aliyuncs.com:8000"),
    )
    if err != nil {
        return nil, err
    }
    pusher := push.New(
        basic.New(
            simple.NewWithExactDistribution(),
            exporter,
        ),
        exporter,
        push.WithPeriod(30*time.Second),
    )
    otel.SetMeterProvider(pusher.MeterProvider())
    pusher.Start()
    return pusher, err
}
```

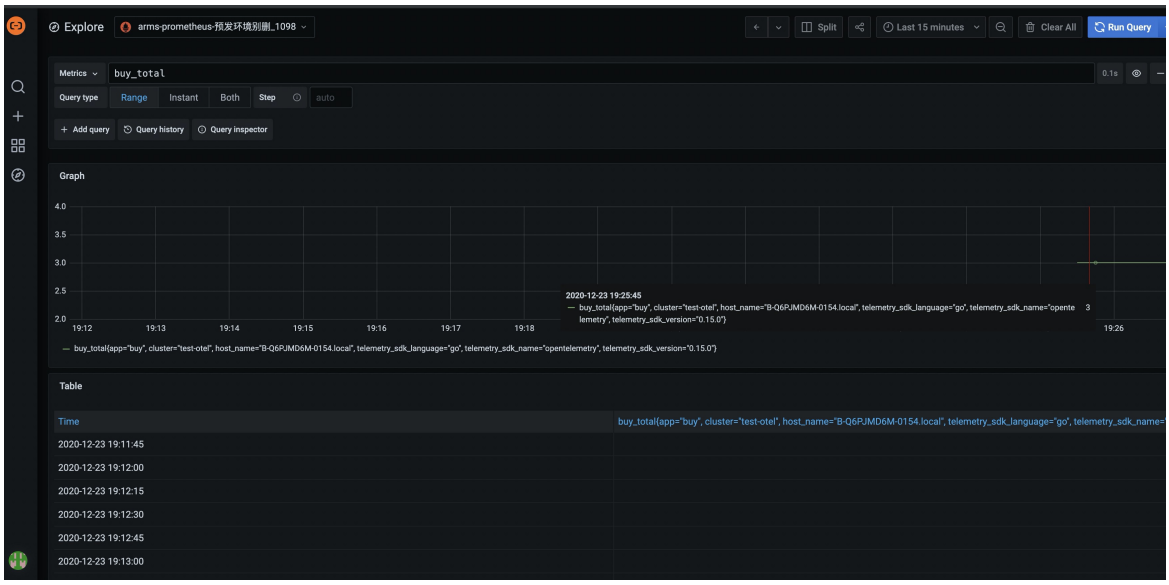
3. 业务入口初始化，并启动OpenTelemetry功能。[Demo]

```
package pkg
import (
    "fmt"
    stat "github.com/liguozhong/prometheus-arms-aliyun-go-demo/pkg/opentelemetry"
    "github.com/prometheus/client_golang/prometheus/promhttp"
    "io"
    "net/http"
    "strconv"
)
type Server struct {
    port int
}
func NewServer(port int) *Server {
    return &Server{
        port: port,
    }
}
func (s *Server) Run() error {
    port := ":" + strconv.Itoa(s.port)
    path := "/metrics"
    service := "/buy"
    http.Handle(path, promhttp.Handler()) //初始一个http handler
    http.HandleFunc(service, func(writer http.ResponseWriter, request *http.Request) {
        content, err := stat.DoBuy()
        if err != nil {
            io.WriteString(writer, err.Error())
            return
        }
        io.WriteString(writer, content)
    })
    stat.InitMeter("buy2", true)
    fmt.Println("http.url: http://localhost" + port + path)
    fmt.Println("service.url: http://localhost" + port + service)
    err := http.ListenAndServe(port, nil)
    if err != nil {
        return err
    }
    return nil
}
```

4. 查看Go module的依赖列表。[Demo]

```
module github.com/liguozhong/prometheus-arms-aliyun-go-demo
go 1.12
require (
    github.com/go-kit/kit v0.9.0
    github.com/prometheus/client_golang v1.7.1
    go.opentelemetry.io/contrib/exporters/metric/cortex v0.15.0
    go.opentelemetry.io/otel v0.15.0
    go.opentelemetry.io/otel/exporters/metric/prometheus v0.15.0
    go.opentelemetry.io/otel/exporters/otlp v0.15.0
    go.opentelemetry.io/otel/sdk v0.15.0
    golang.org/x/text v0.3.3 // indirect
)
```

5. 在Grafana大盘中查看数据。



停止远程存储Prometheus监控数据

如需停止远程数据库存储Prometheus监控数据，请按照以下步骤卸载Prometheus监控插件。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，找到要卸载监控插件的Remote Write实例，然后在其右侧**操作**列，单击**卸载**，并在弹出的**确认**对话框单击**确认**。
   卸载插件完毕后，**Prometheus监控**页面不再显示该类型的实例。

4.7. Prometheus实例 for GlobalView

本文说明如何创建Prometheus实例 for GlobalView（全局聚合实例），即如何将阿里云或自建Prometheus集群的全局聚合实例接入Prometheus监控，并使用内置的云服务大盘和报警规则实现对全局聚合实例的监控和报警。

前提条件

已开通ARMS。具体操作，请参见开通ARMS。

功能入口

1. 登录Prometheus控制台。

创建全局聚合实例

1. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建Prometheus实例**。
2. 在**新建Prometheus实例**页面，单击**全局聚合实例**区域，并在弹出的面板按照控制台界面提示配置基本信息，然后单击**创建聚合实例**完成创建。
   创建完成后，在**Prometheus监控**页面将会显示实例类型为**Prometheus for Globalview**的全局聚合实例。

> **说明**
> - Endpoint是请求的访问点、告警的配置地域。建议您选择实例数量最多的地域作为访问点，若您选择其他地域可能会影响访问速度和系统稳定性。
> - 您可以在**STEP3**区域选择不同地域（Region）下的实例名称以实现跨Region的实例聚合，不过在您选择实例之前需要在**STEP2**区域指定访问的Endpoint。
> - 您可以单击已创建的全局聚合实例右侧**操作**的**编辑**，编辑创建的全局聚合实例信息。若您重新修改了Endpoint信息，会导致在原Endpoint下配置的告警规则失效，因此不建议您随意变更Endpoint。

通过Grafana大盘查看全局聚合实例监控指标

通过阿里云Prometheus监控创建全局聚合实例后，您可以通过预置的Grafana大盘查看全局聚合实例的性能指标数据。

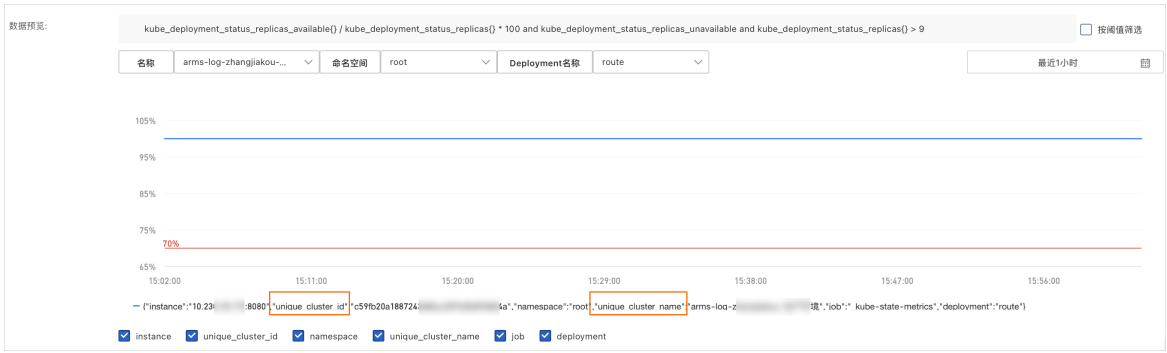
1. 在**Prometheus监控**页面单击已创建的全局聚合实例名称，进入**大盘列表**页面。
2. 在**大盘列表**页面，单击大盘名称，查看全局聚合实例的性能指标数据。

> **说明** 单击右上角的**大盘重置**，您可以对大盘进行重置。更多信息，请参见重置大盘。

创建全局聚合实例的告警

1. 在**Prometheus监控**页面单击已创建的全局聚合实例名称，进入**大盘列表**页面。
2. 在左侧导航栏单击**告警规则**，然后在页面右上角单击**创建Prometheus告警规则**，并按照控制台界面提示配置告警规则的基本信息。具体操作，请参见Prometheus告警规则。

> **说明** 其中，在**创建Prometheus告警规则**页面的**数据预览**区域，全局聚合实例提供了unique_cluster_id（实例的唯一标识）和unique_cluster_name（实例名称），以便您在追踪告警对象时，能快速定位到对应触发告警阈值的实例。



卸载删除全局聚合实例

若您暂不需要监控全局聚合实例，您可以卸载Prometheus监控插件。

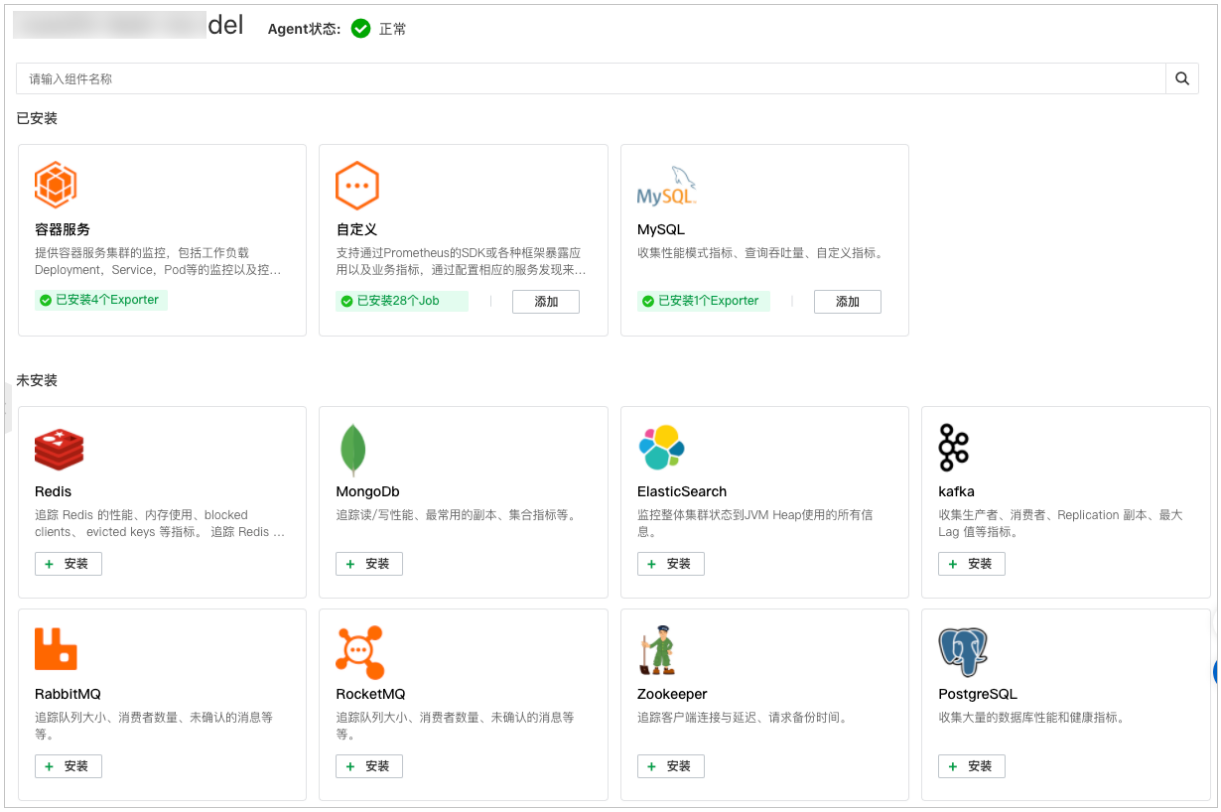
在**Prometheus监控**页面，单击目标全局聚合实例右侧**操作**的**卸载**，并在弹出的提示框中单击**确认**，即可完成卸载。

5.集成中心

集成中心作为Prometheus实例的入口，将容器服务、自定义服务发现、组件监控的关联数据和高频操作进行集中化展示。

前提条件

您的Prometheus Helm版本已升级至v1.1.5或以上。升级Helm版本的具体操作，请参见升级组件版本。



集成中心的优势

- 快速添加、查看不同类别的集成对象信息。
- 灵活查看Targets关联的指标、服务发现配置及Grafana大盘。
- 快速接入自定义服务发现，支持服务发现配置（YAML）格式校验。
- 组件监控提供Exporter一键升级、更新连接串信息、查看日志、详情等功能。
- 提供面向全量Job的服务发现配置和编辑能力。
- 提供更加灵活、便捷的交互体验。

支持的集成对象

Prometheus监控目前支持3种集成对象类型，包括容器服务集成、自定义集成、组件集成（共11种）。

集成类型	集成对象	说明
容器服务集成	容器服务	提供容器服务集群的监控。包括负载率Deployment、Service、Pod等的监控，以及控制面ETCD、APIServer等的监控。

集成类型	集成对象	说明
自定义集成	自定义	支持通过Prometheus监控的SDK或各种框架暴露应用以及指标业务，支持配置相应的服务发现来采集监控数据。
组件集成	MySQL	性能指标、性能查询、采集自定义指标。
	Redis	追踪Redis的性能、内存使用、Blocked Clients、Evicted Keys等指标。
	MongoDB	追踪读写性能、最常用的副本、集合指标等。
	Elasticsearch	监控整体集群状态到JVM Heap使用的所有信息。
	Kafka	收集生产者、消费者、Replication副本、最大Lag值等指标。
	RabbitMQ	追踪队列大小、消费者数量、未确认的消息等。
	RocketMQ	追踪队列大小、消费者数量、未确认的消息等。
	ZooKeeper	追踪客户端连接与延迟、请求备份时间。
	PostgreSQL	收集大量的数据库性能和健康指标。
	Nginx	监测连接和请求指标。
	Nginx（V2）	监测连接和请求指标。

功能入口

Prometheus监控集成中心页面展示当前支持的集成对象以及安装状态，您可以选择对应的集成对象类别卡片查看详情或添加集成。

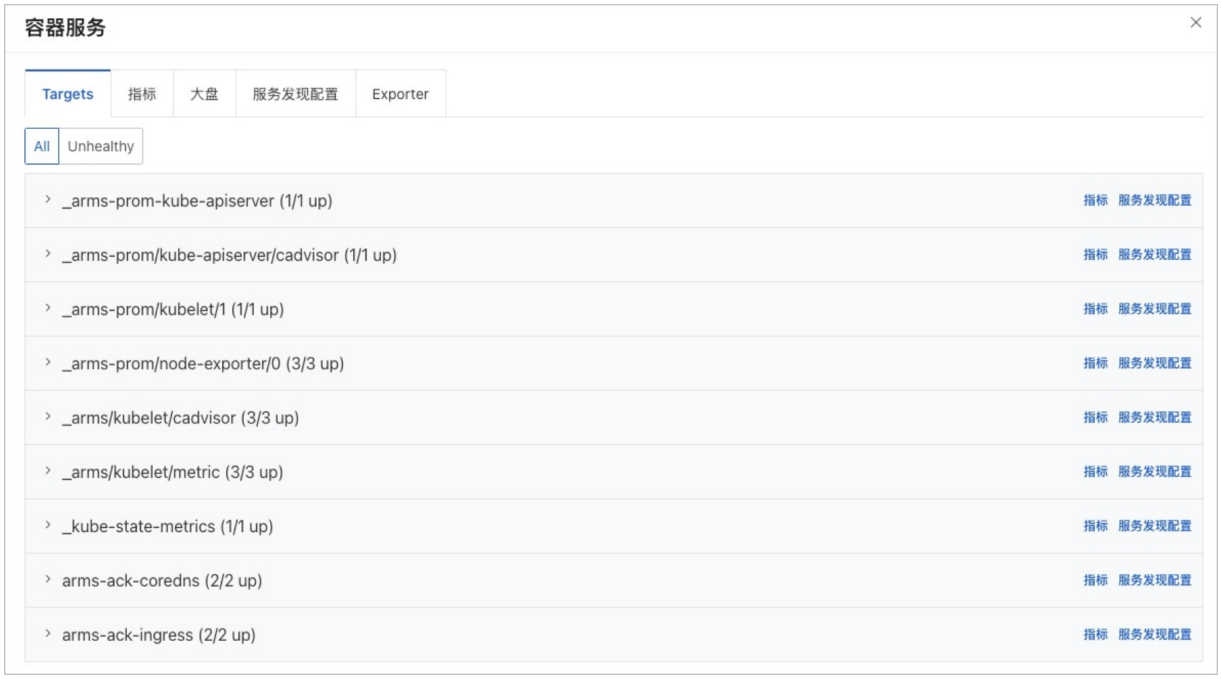
1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击目标实例名称，即可进入集成中心页面。

容器服务集成

容器服务集成在集群注册时Prometheus监控会默认安装。在**集成中心**页面单击**容器服务**卡片。**容器服务**展示集成对象的集成详情，包括Targets、指标、大盘、服务发现配置和Exporter信息。

Targets

您可以在**Targets**页签查看容器服务集成默认Job发现的Targets列表，以及指定Target的指标和服务发现配置。



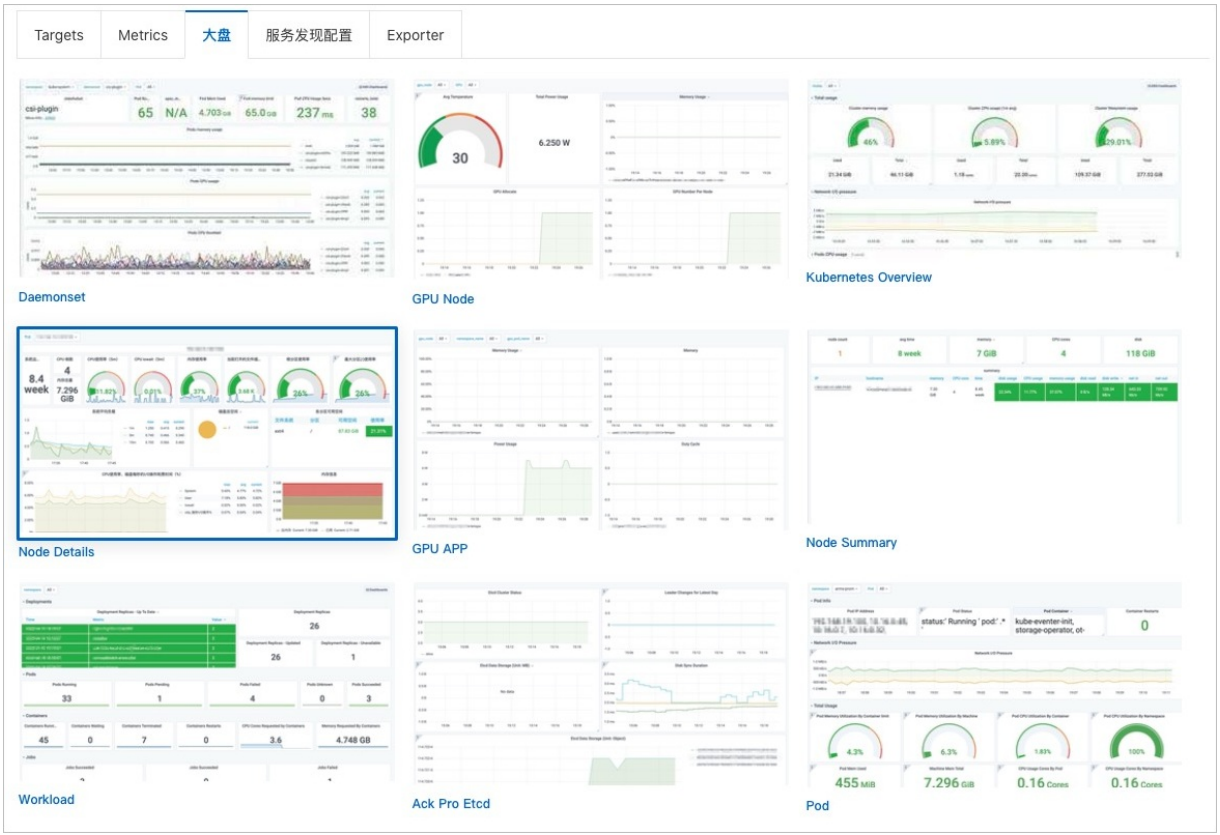
指标

您可以在**指标**页签查看具体的指标信息并对指标进行废弃配置。具体操作，请参见配置指标。

> **说明** Prometheus监控将容器服务ACK的Target与对应指标项进行了关联。

大盘

您可以在**大盘**页签查看Prometheus监控提供的预置大盘，同时可以通过单击任一大盘名称跳转至Grafana平台查看更多大盘数据。



服务发现配置

您可以在**服务发现配置**页签对默认基础监控Job进行操作，例如编辑抓取时间间隔、关闭采集Job等。更多信息，请参见管理Kubernetes集群服务发现。

Exporter

您可以在**Exporter**页签查看默认安装的`node-exporter`、`kube-state-metrics`以及对应的版本号信息。

自定义集成

您可以添加自定义集成服务发现，进行自定义采集接入，具体操作如下。

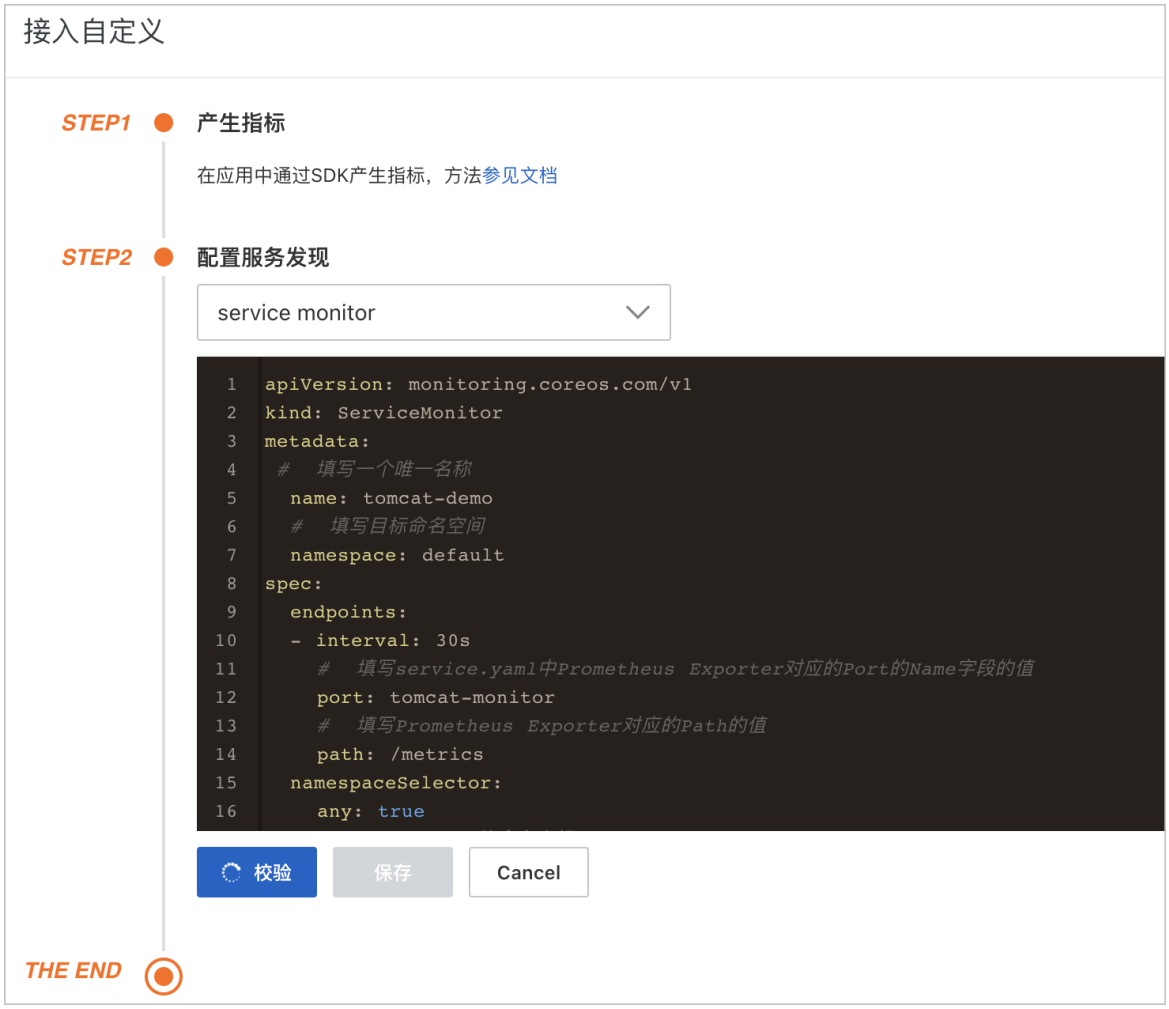
1. 在**集成中心**页面单击**自定义**卡片的**添加**。
2. 在弹出的**接入自定义**面板的**STEP2**区域配置服务发现。

   Prometheus监控目前支持4种服务发现方式：

   - pod annotation
   - service monitor
   - pod monitor
   - 自定义服务发现

   配置完服务发现后，您可以单击**校验**，对您的服务发现配置进行（YAML）格式校验，当页面提示**校验通过**后单击**保存**。

> **注意** 校验过程是十分必要的，因为服务发现配置错误可能会影响到您当前运行的Job。



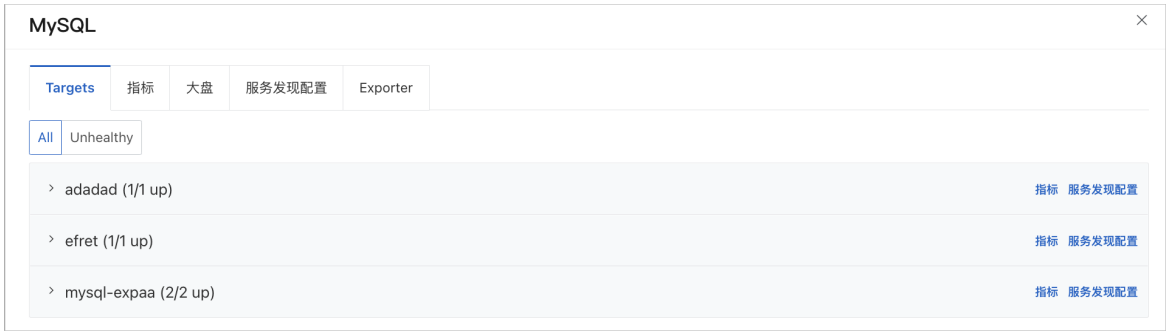
3. 查看配置的服务发现是否生效。

   单击**自定义**卡片，在Targets页签您可以查看配置的服务发现是否生效。若查看到未生效的Target，您可以单击**服务发现配置**页签编辑已配置的Job。

组件集成

您可以添加MySQL、Redis等组件，进行组件采集接入，这里以添加MySQL组件接入为例，为您介绍具体操作，其他组件接入的方式类似。

1. 在**集成中心**页面单击**MySQL**组件卡片的**添加**。
2. 在弹出的**接入MySQL**面板填写相关参数配置。参数配置详情请参见使用阿里云Prometheus监控MySQL。参数配置完成后，单击**确定**系统会自动跳转至Targets页签，您可以查看接入的Targets列表。


MySQL
Targets
指标
大盘
服务发现配置
Exporter
All
Unhealthy
adadad (1/1 up)
指标 服务发现配置
efret (1/1 up)
指标 服务发现配置
mysql-expaa (2/2 up)
指标 服务发现配置

相关文档

-
- 组件监控概述

6.组件监控接入

6.1. 组件监控概述

本教程介绍如何使用阿里云Prometheus通过组件监控接入您的MySQL、Redis、MongoDB等应用或组件，并以Grafana大盘展示监控数据。

通过阿里云Prometheus安装的组件监控为开源组件监控，您可以通过组件监控接入自建或阿里云的应用和组件。例如，您可以通过组件监控接入自建的MySQL，也可以接入阿里云提供的RDS for MySQL。可以通过组件监控接入MySQL、Redis、MongoDB等应用或组件的Prometheus实例有3种，分别是Prometheus实例for 容器服务、Prometheus实例 for Kubernetes和Prometheus实例 for ECS。

Redis

- 使用阿里云Prometheus监控Redis

MySQL

- 使用阿里云Prometheus监控MySQL

Elasticsearch

- 使用阿里云Prometheus监控Elasticsearch

MongoDB

- 使用阿里云Prometheus监控MongoDB

PostgreSQL

- 使用阿里云Prometheus监控PostgreSQL

Kafka
- 使用阿里云Prometheus监控Kafka

RabbitMQ
- 使用阿里云Prometheus监控RabbitMQ

RocketMQ
- 使用阿里云Prometheus监控RocketMQ

Nginx（旧版）
- 使用阿里云Prometheus监控Nginx（旧版）

Nginx（新版）
- 使用阿里云Prometheus监控Nginx（新版）

ZooKeeper
- 使用阿里云Prometheus监控ZooKeeper

其他应用组件
- 其他类型组件接入

6.2. 使用阿里云Prometheus监控MySQL

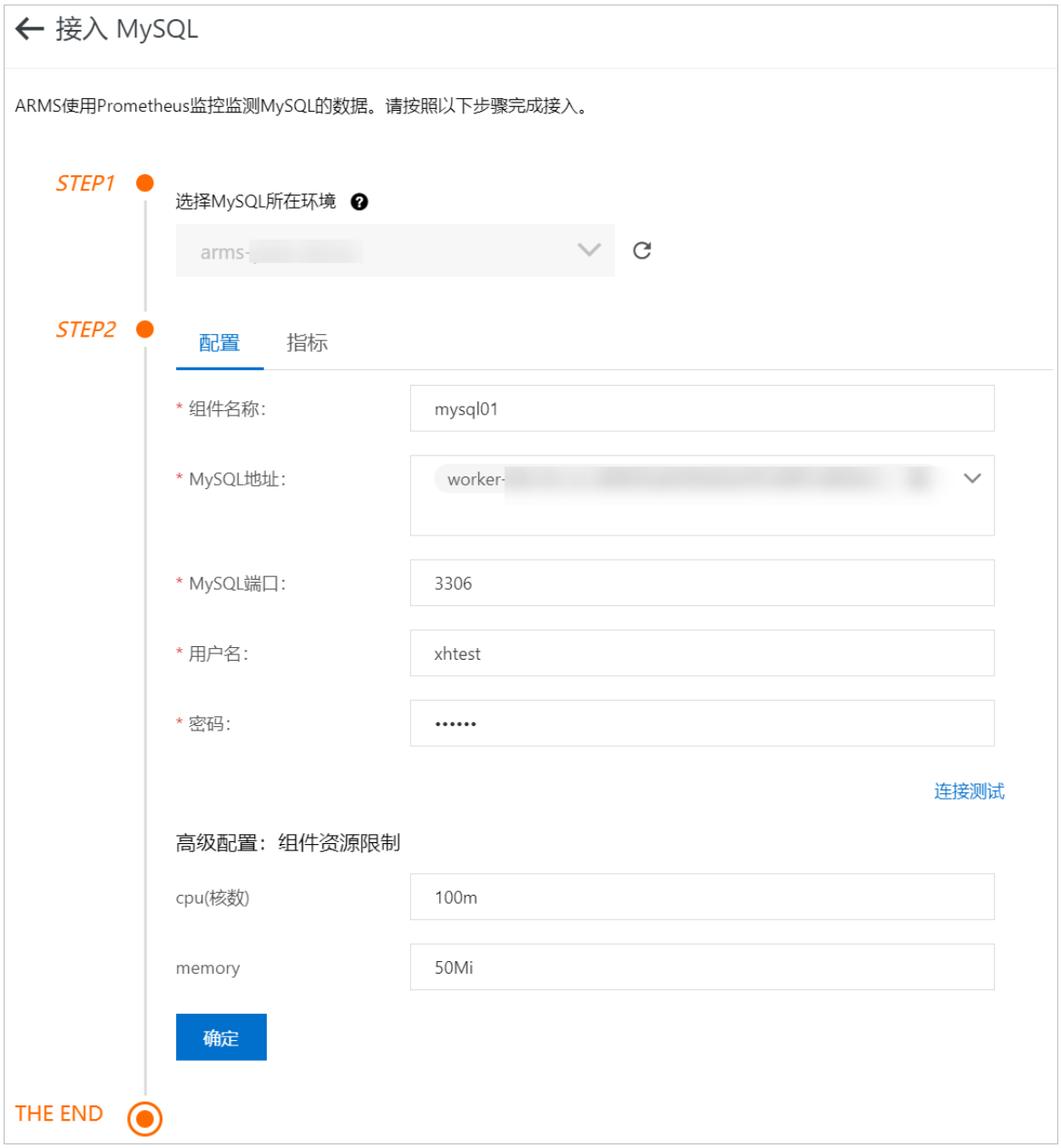
阿里云Prometheus监控提供一键安装配置MySQL类型的组件功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加MySQL类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击MySQL组件图标。
3. 在**接入MySQL**面板**STEP2**区域的**配置**页签输入各项参数。



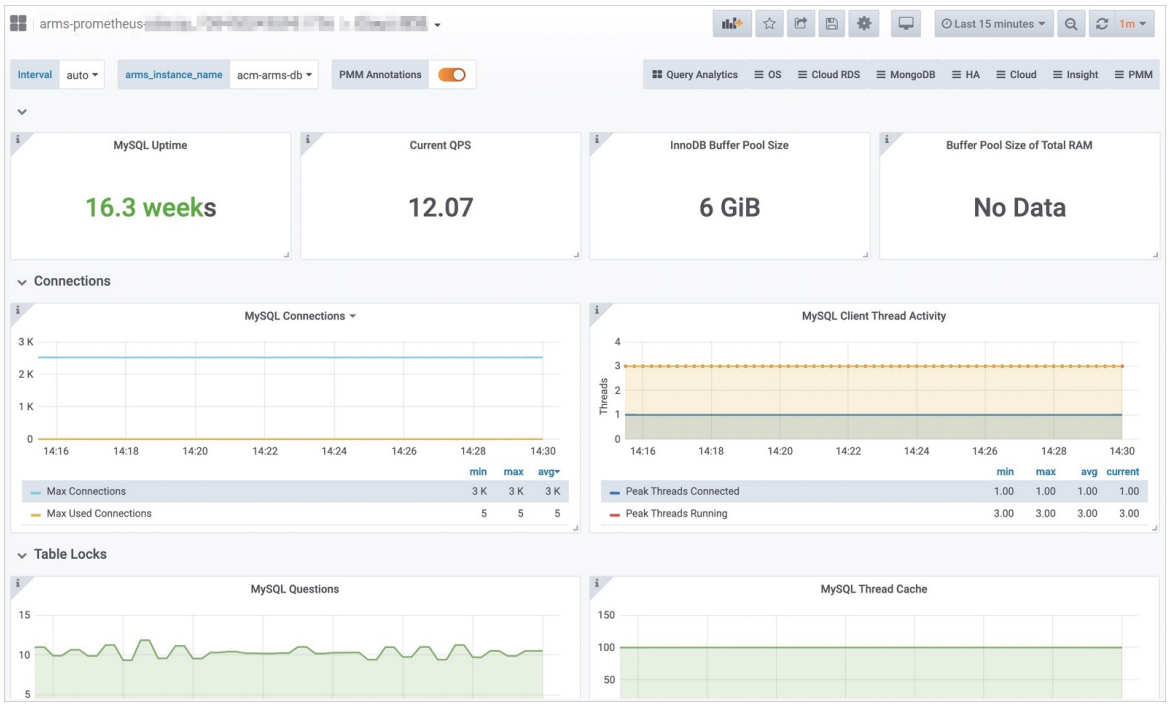
参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。

参数	描述
MySQL地址	MySQL的连接地址。 说明 支持部署在容器服务Kubernetes版、云服务器ECS、云数据库RDS的MySQL地址联想。
MySQL端口	MySQL的端口号，例如：3306。
用户名	MySQL的用户名称。
密码	MySQL的密码。

说明 在**接入MySQL**面板**STEP 2**区域的**指标**页签可查看监控指标。

4. （可选）在**接入MySQL**面板**STEP 2**区域的**高级配置**页签设置组件采集数据时可以使用的最大CPU核数和内存。
5. 配置完成后，单击**确定**。在**组件监控**页面，会显示已接入的组件实例。

6. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



说明 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.3. 使用阿里云Prometheus监控Redis

阿里云Prometheus监控提供一键安装配置Redis类型的组件，并提供开箱即用的专属监控大盘。

功能入口

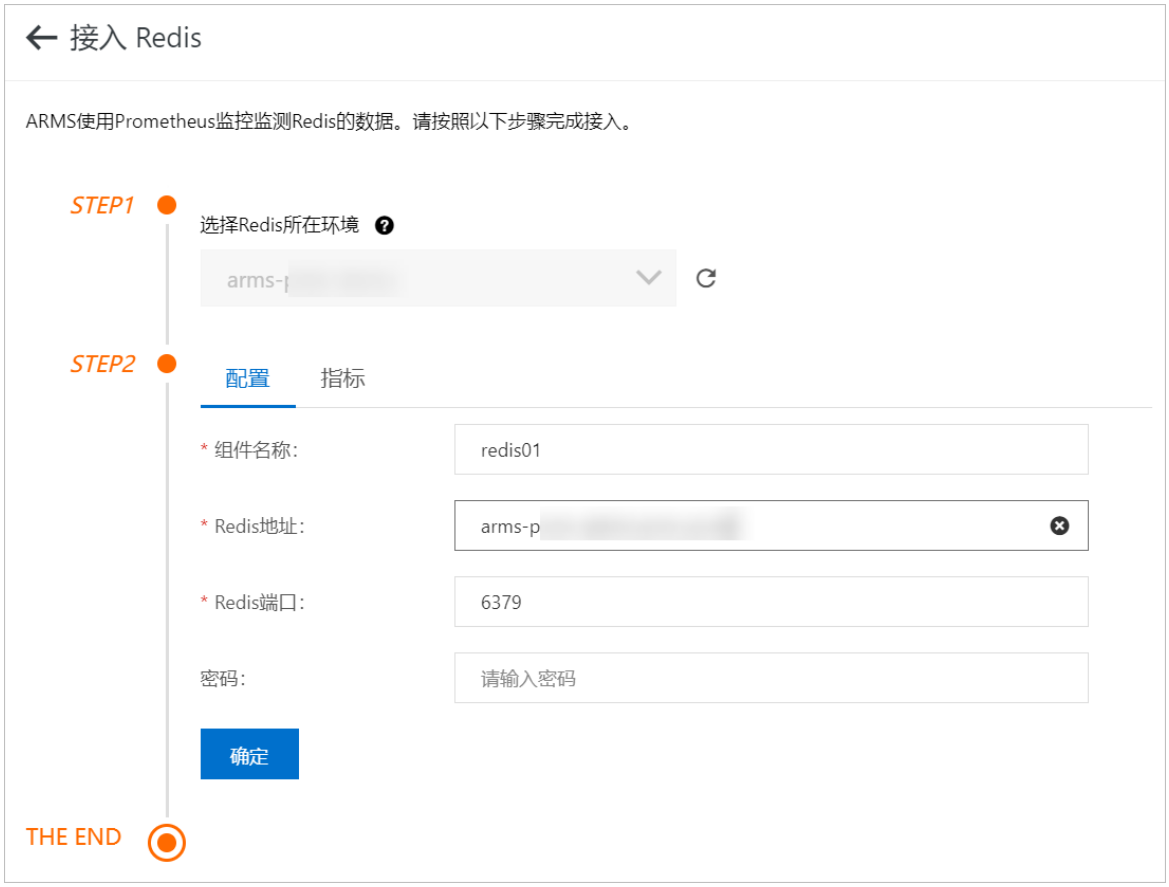
1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者VPC类型的Prometheus实例名称。

   > **说明** 云服务实例类型的Prometheus监控暂不支持组件监控接入。

3. 在左侧导航栏单击**组件监控**。

添加Redis类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击Redis组件图标。
3. 在**接入Redis**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**，完成Redis组件接入。

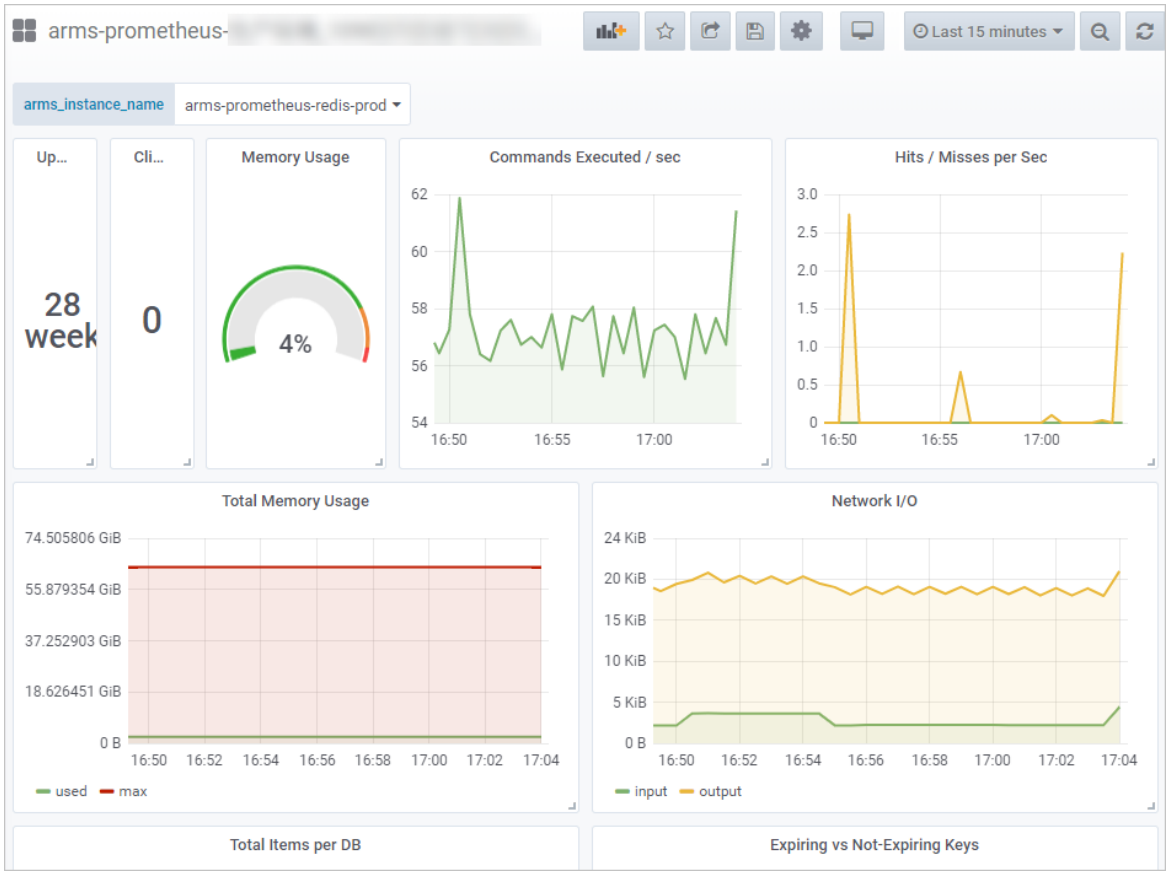


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
Redis地址	Redis的连接地址。
Redis端口	Redis的端口号，例如：6379。
密码	Redis的连接密码。

说明 在**接入Redis面板**STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> **说明** 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.4. 使用阿里云Prometheus监控Elasticsearch

阿里云Prometheus监控提供一键安装配置Elasticsearch类型的组件，并提供开箱即用的专属监控大盘。

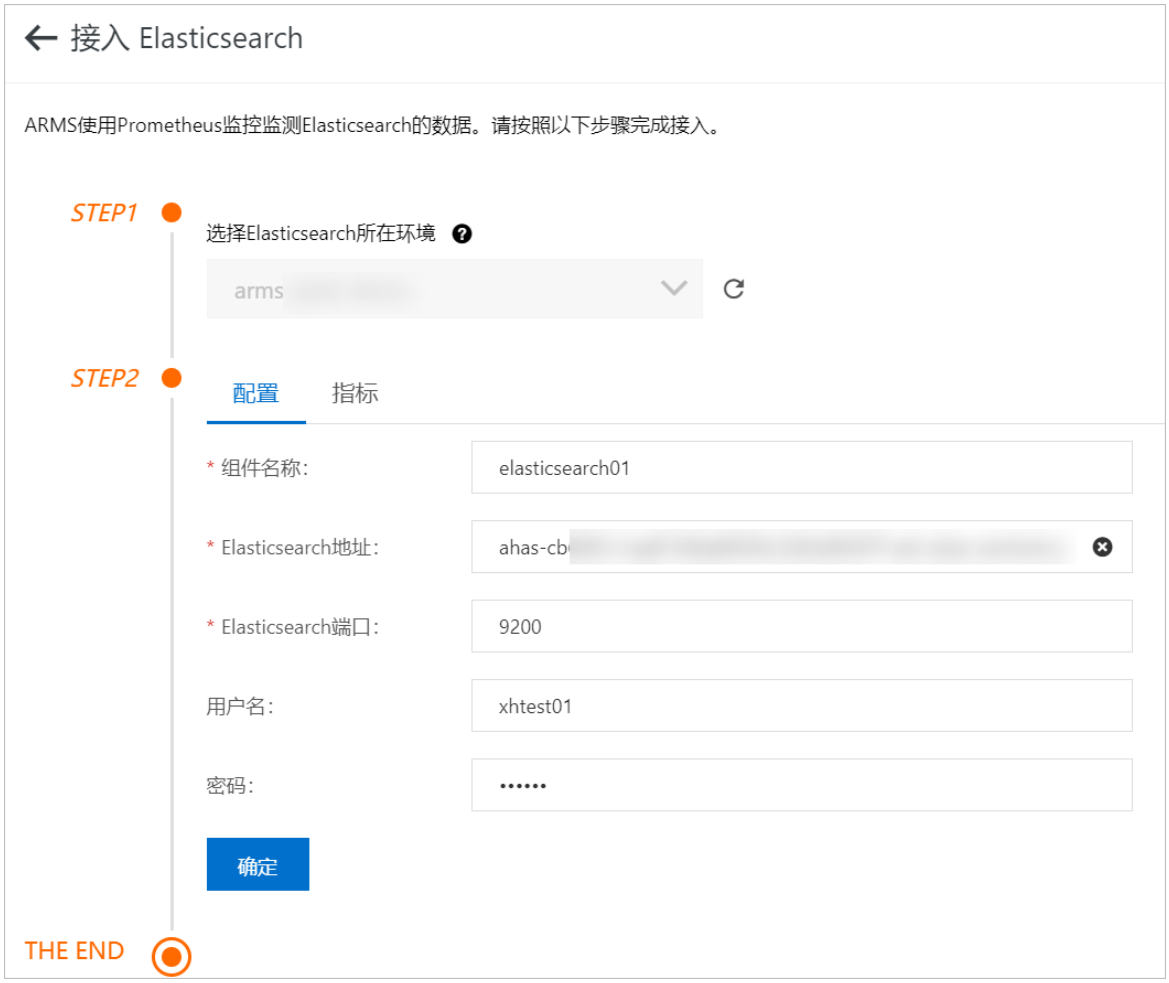
功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加Elasticsearch类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。

2. 在**接入中心**面板中单击Elasticsearch组件图标。
3. 在**接入Elasticsearch**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**。

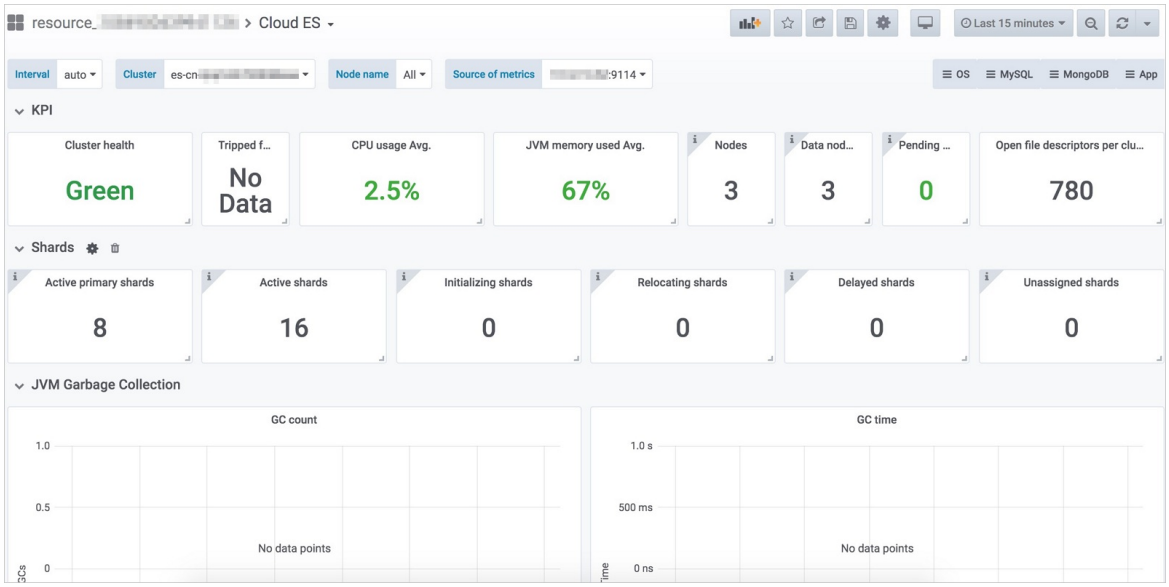


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
Elasticsearch地址	Elasticsearch的连接地址。
Elasticsearch端口	Elasticsearch的端口号，例如：9200。
用户名	Elasticsearch的用户名称。
密码	Elasticsearch的密码。

说明 在**接入Elasticsearch**面板**STEP2**区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> **说明** 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.5. 使用阿里云Prometheus监控MongoDB

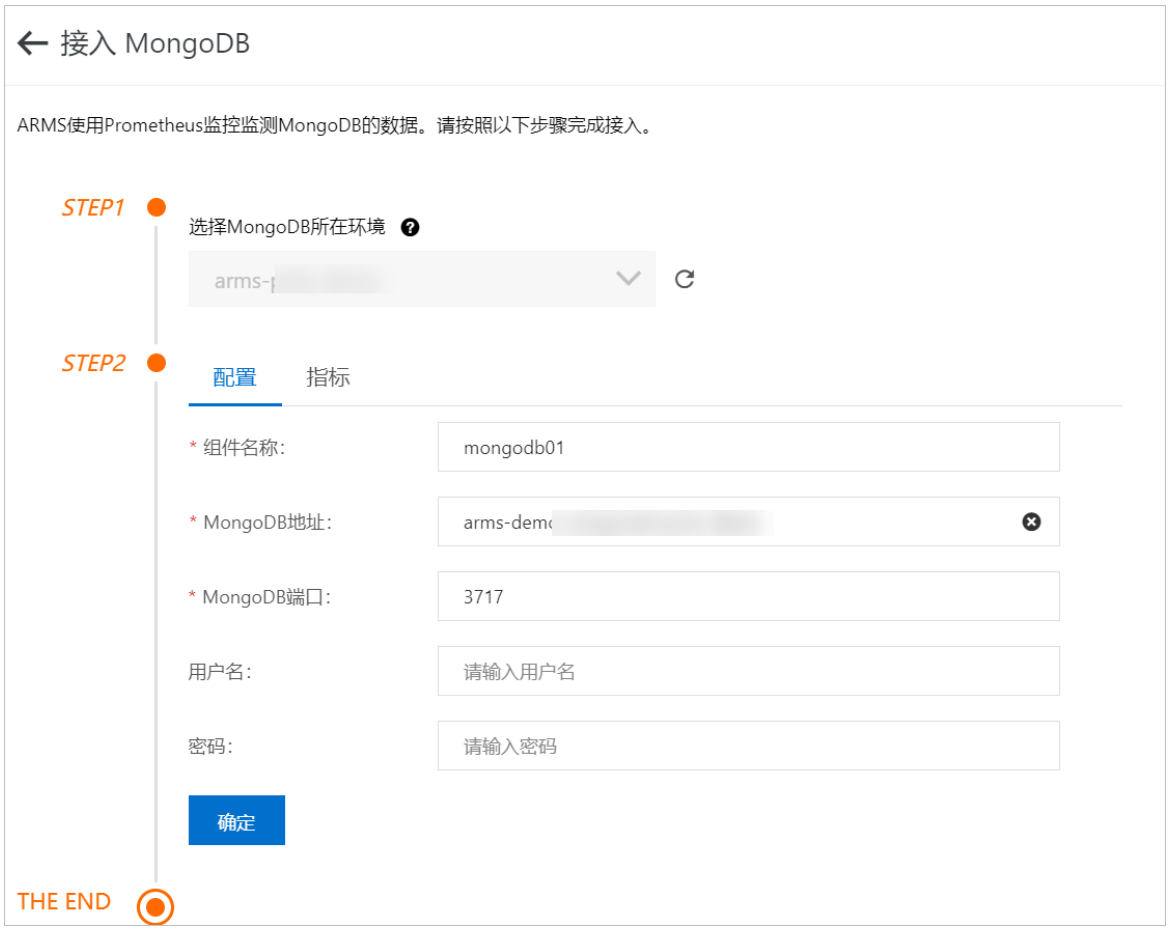
阿里云Prometheus监控提供一键安装配置MongoDB类型的组件，并提供开箱即用的专属监控大盘。

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加MongoDB类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击MongoDB组件图标。
3. 在**接入MongoDB**面板STEP2区域的**配置**页签输入各项参数，并单击**确定**。

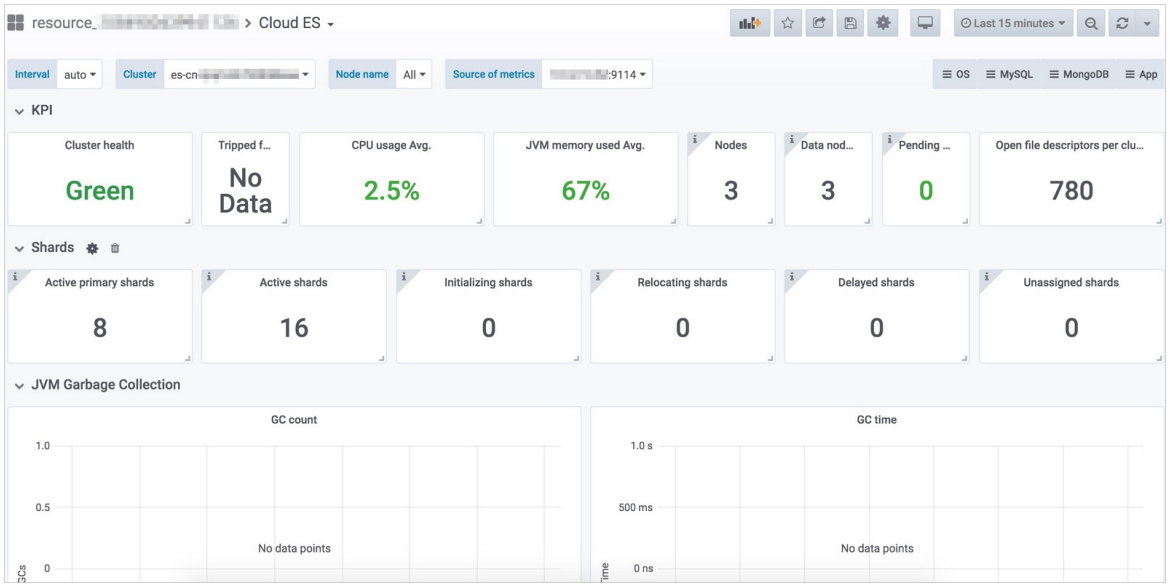


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
MongoDB地址	MongoDB的连接地址。
MongoDB端口	MongoDB的端口号，例如：3717。
用户名	MongoDB的用户名称。
密码	MongoDB的密码。

说明 在**接入MongoDB**面板STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> **说明** 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.6. 使用阿里云Prometheus监控Kafka

阿里云Prometheus监控提供一键安装配置Kafka类型的组件，并提供开箱即用的专属监控大盘。

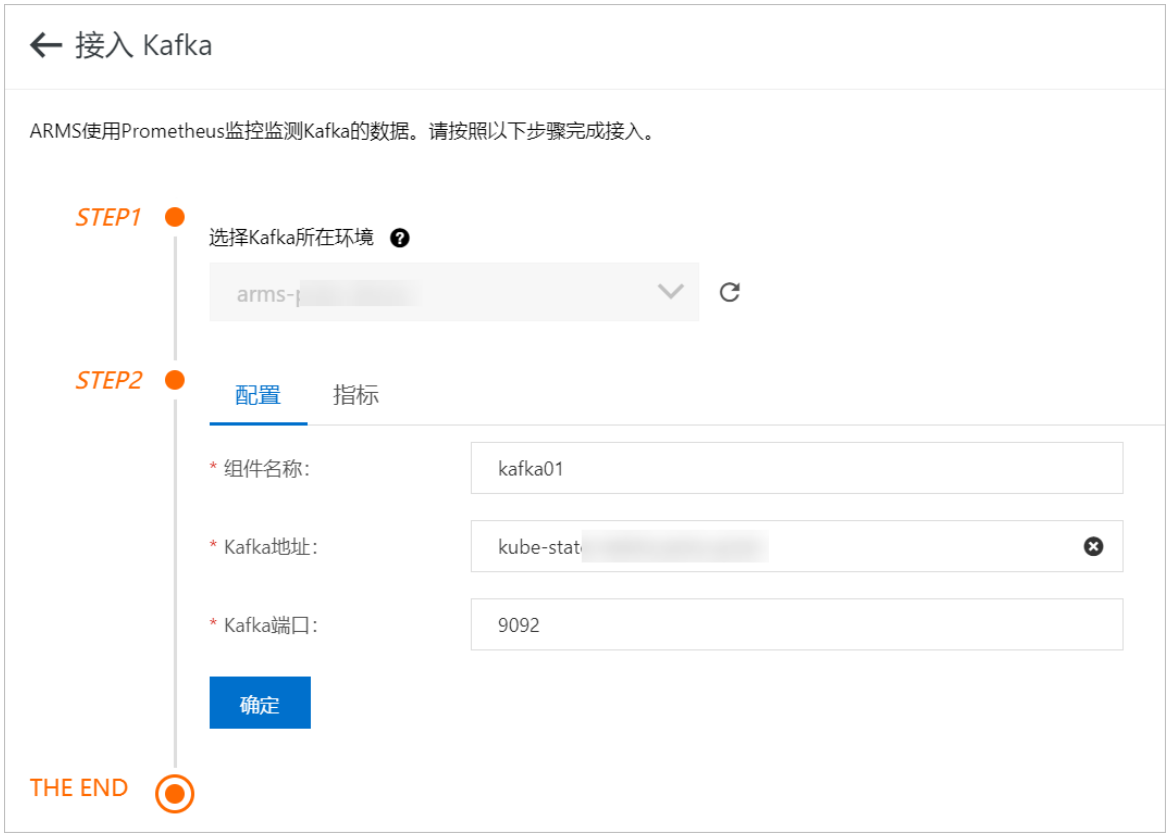
功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加Kafka类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。

2. 在**接入中心**面板中单击Kafka组件图标。
3. 在**接入Kafka**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**。

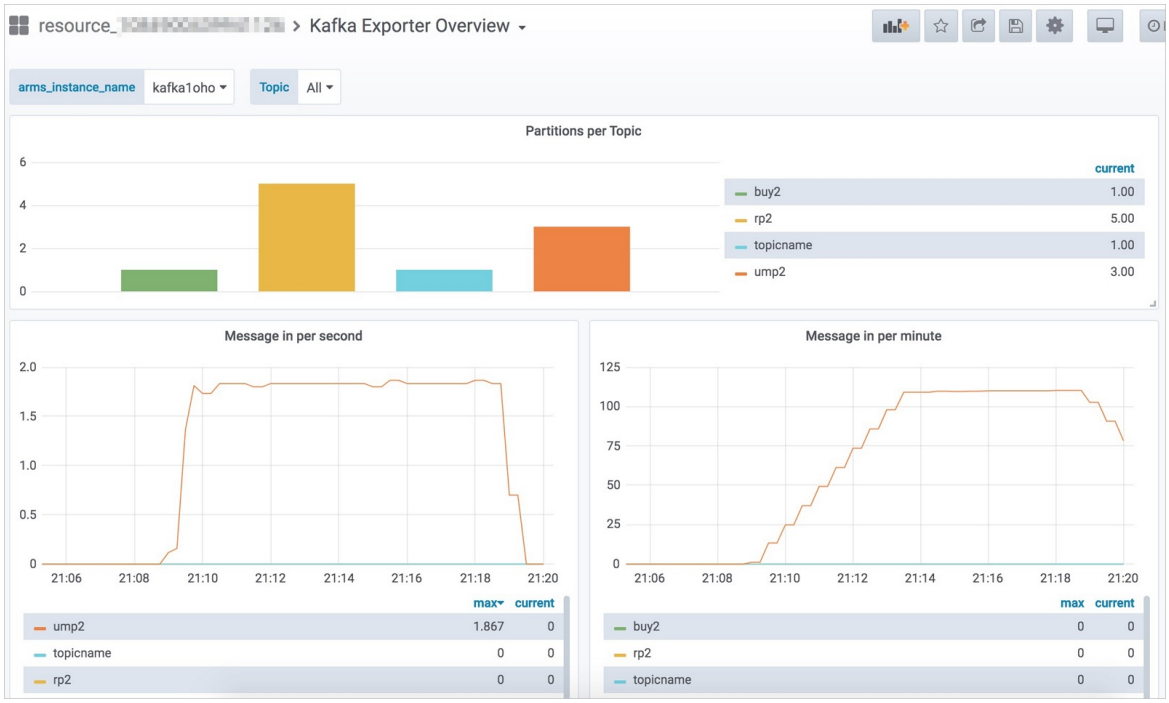


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
Kafka地址	Kafka的连接地址。
Kafka端口	Kafka的端口号，例如：9092。

说明 在**接入Kafka**面板**STEP2**区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> **说明** 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.7. 使用阿里云Prometheus监控RabbitMQ

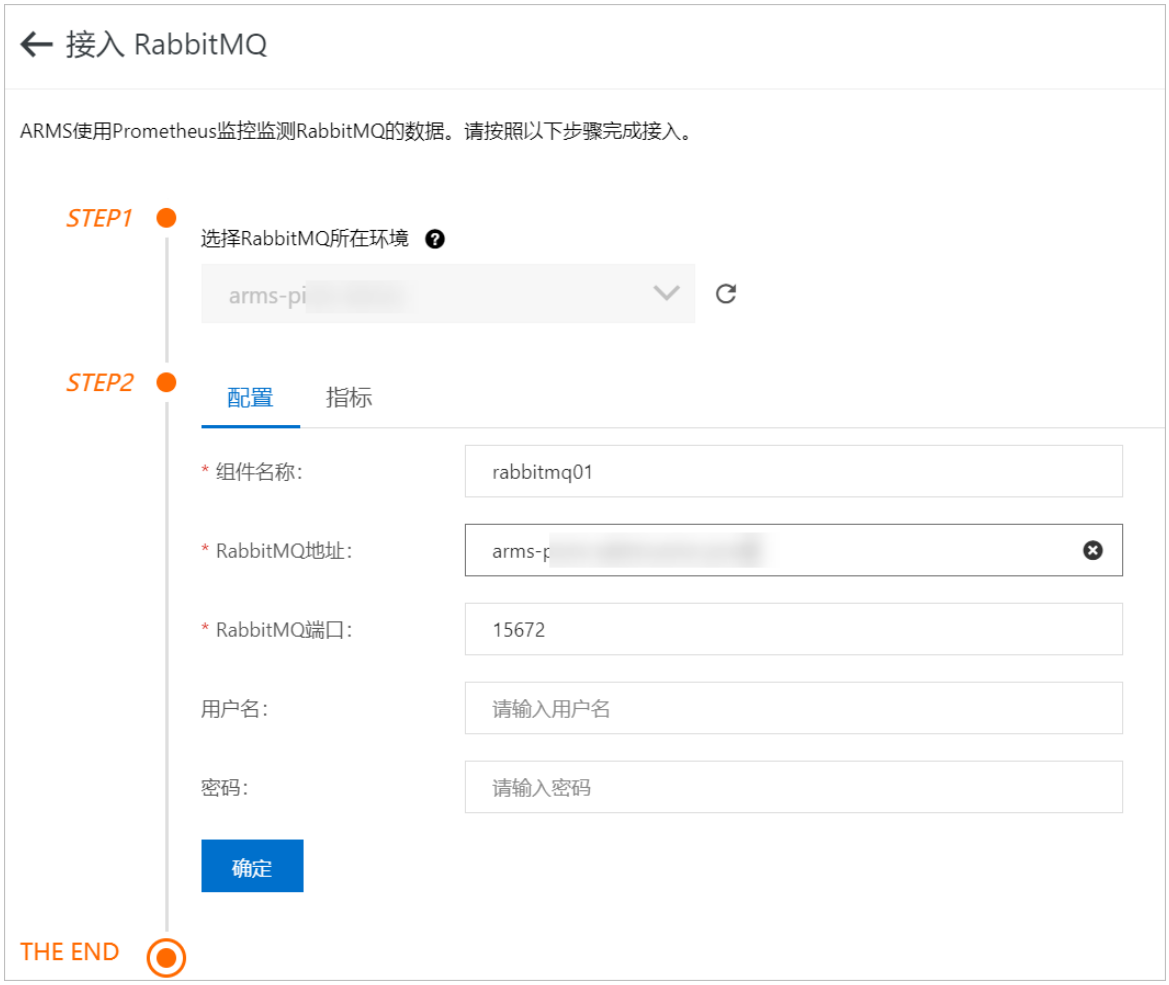
阿里云Prometheus监控提供一键安装配置RabbitMQ类型的组件，并提供开箱即用的专属监控大盘。

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加RabbitMQ类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击RabbitMQ组件图标。
3. 在**接入RabbitMQ**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**。

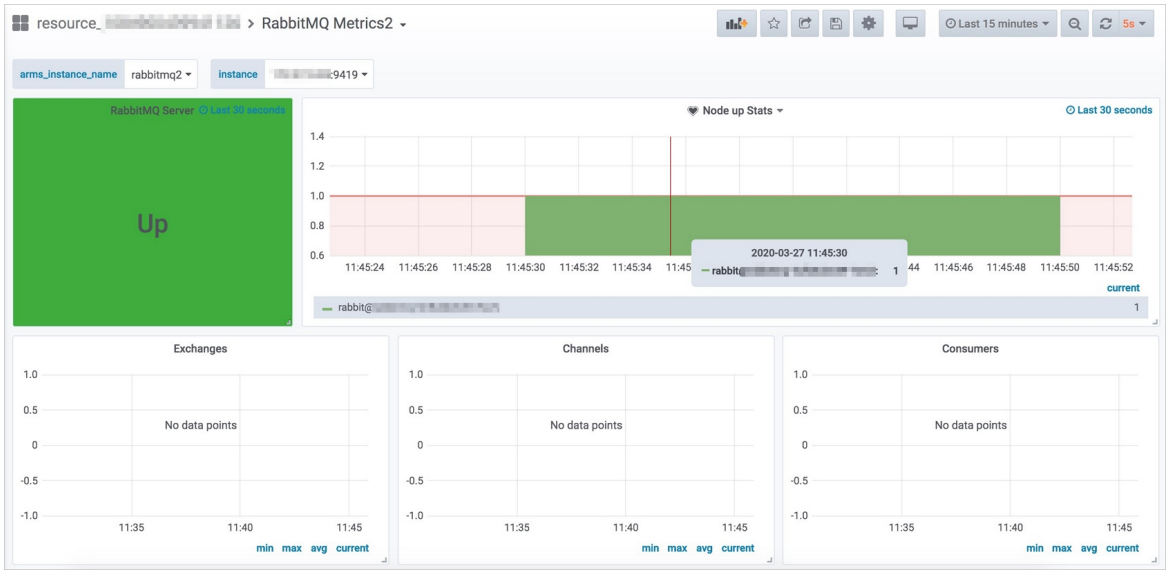


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
RabbitMQ地址	RabbitMQ的连接地址。
RabbitMQ端口	RabbitMQ的端口号，例如：15672。
用户名	RabbitMQ的用户名称。
密码	RabbitMQ的密码。

> 说明　在接入RebbitMQ面板STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> 说明　单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.8. 使用阿里云Prometheus监控RocketMQ

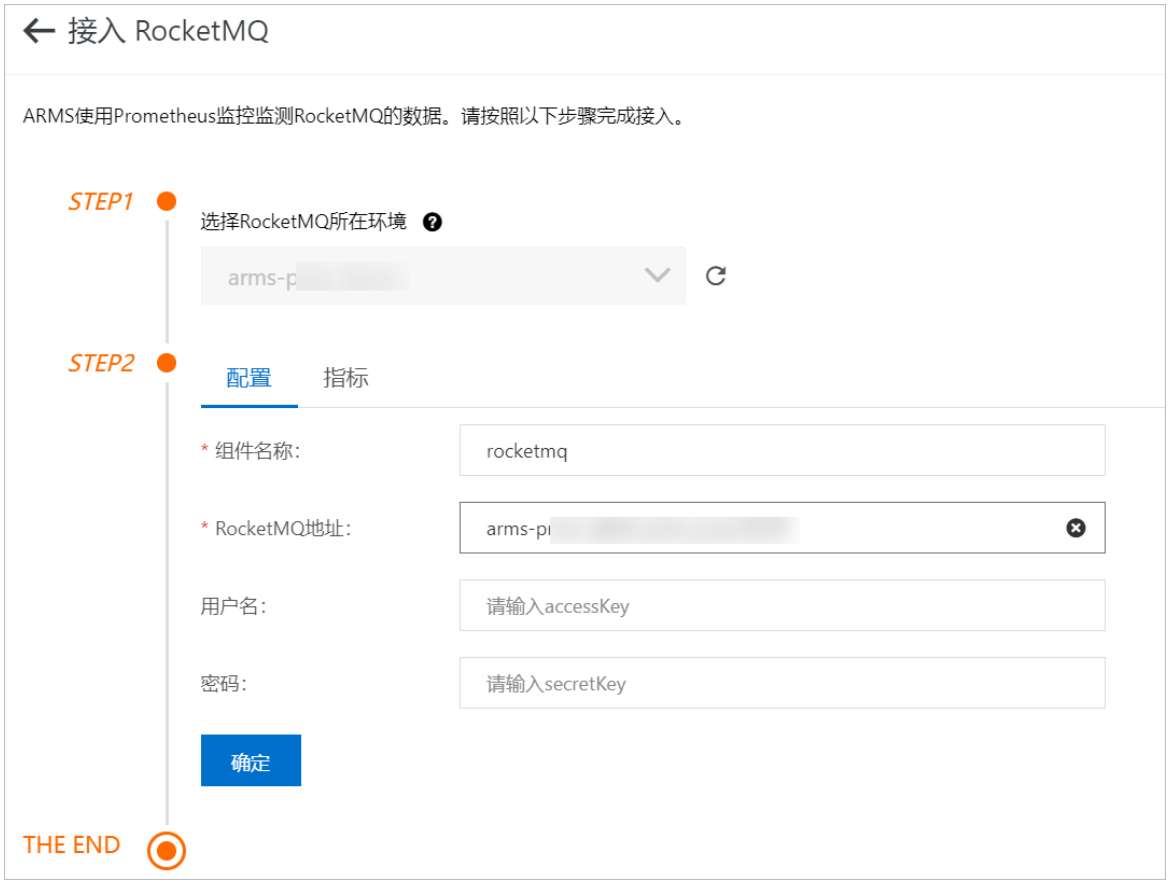
阿里云Prometheus监控提供一键安装配置RocketMQ类型的组件，并提供开箱即用的专属监控大盘。

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加RocketMQ类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击RocketMQ组件图标。
3. 在**接入RocketMQ**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**。

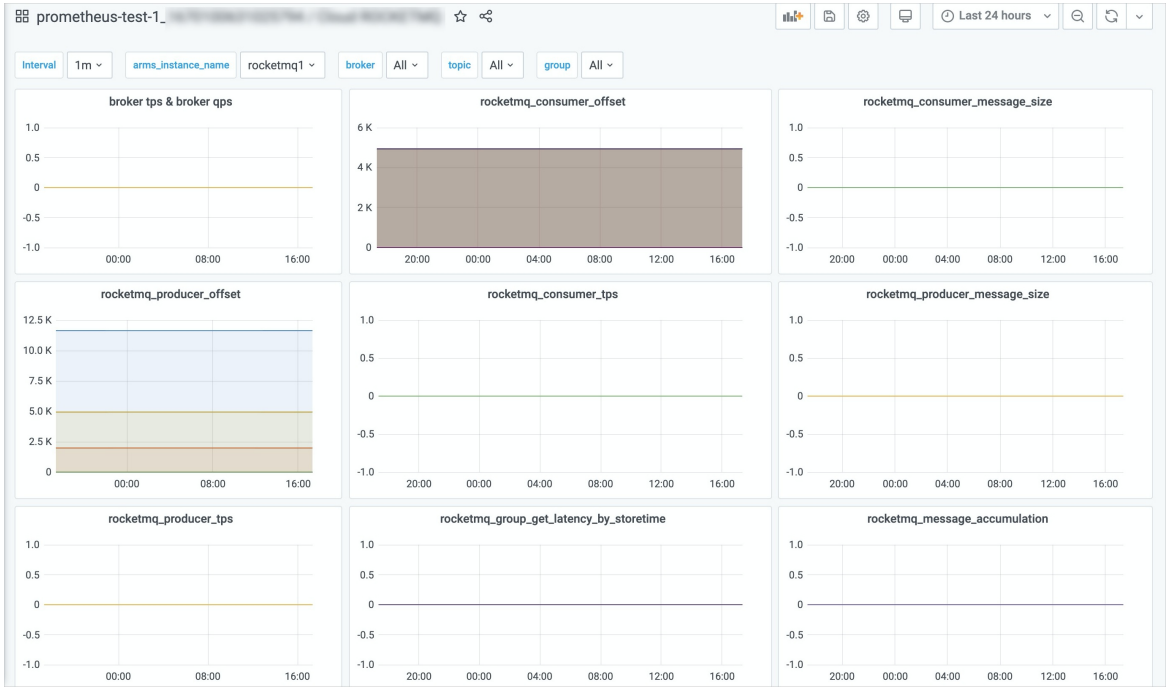


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
RocketMQ地址	RocketMQ的连接地址。格式为：**连接地址:端口号**，多个地址之间使用半角逗号（,）分隔。 **说明** 支持部署在容器服务Kubernetes版的RocketMQ地址联想。
用户名	RocketMQ的AccessKey。
密码	RocketMQ的SecretKey。

> 说明　在接入RocketMQ面板STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> 说明　单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

相关文档

- 其他类型的组件接入

6.9. 使用阿里云Prometheus监控PostgreSQL

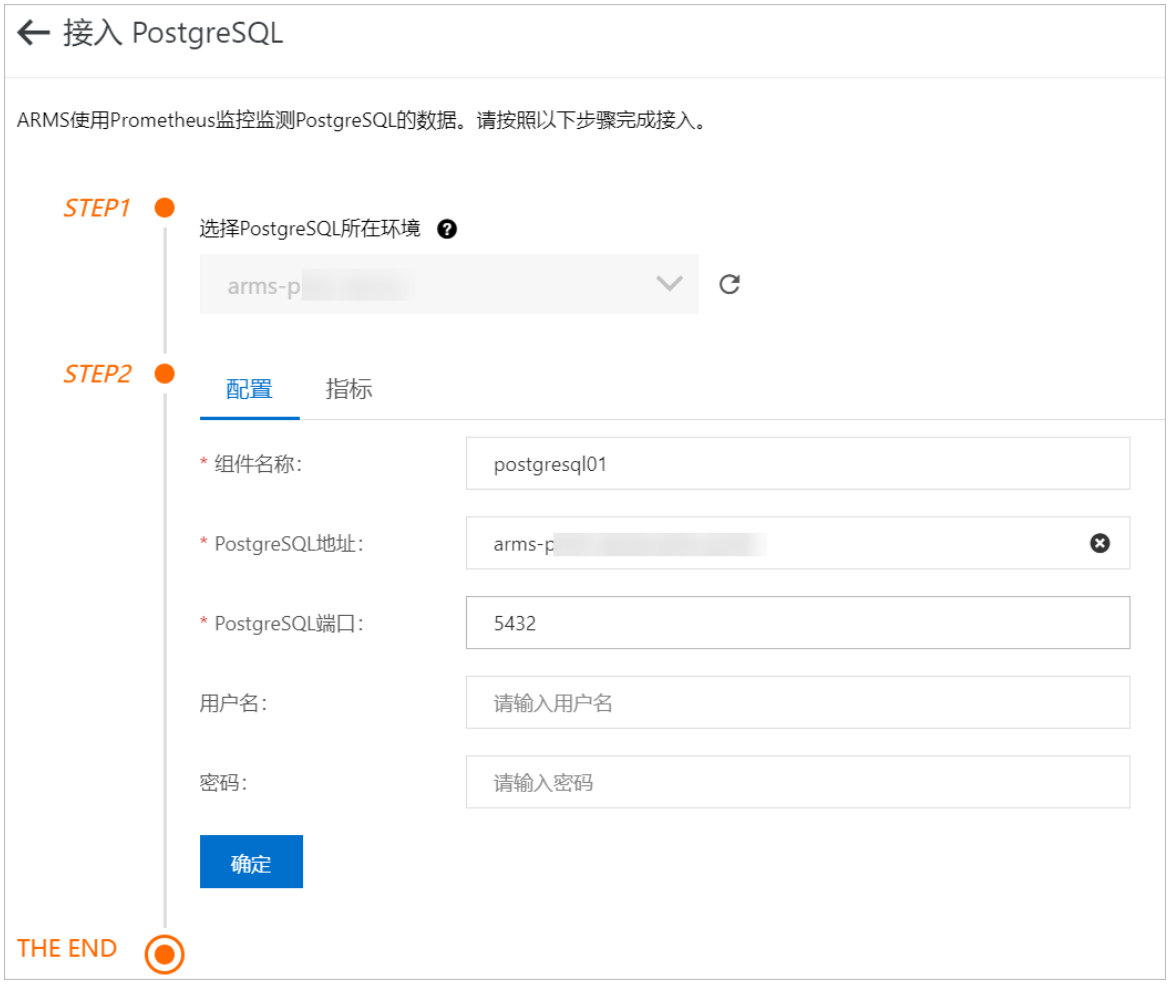
阿里云Prometheus监控提供一键安装配置PostgreSQL类型的组件，并提供开箱即用的专属监控大盘。

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加PostgreSQL类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击PostgreSQL组件图标。
3. 在**接入PostgreSQL**面板**STEP2**的**配置**页签输入各项参数，并单击**确定**。



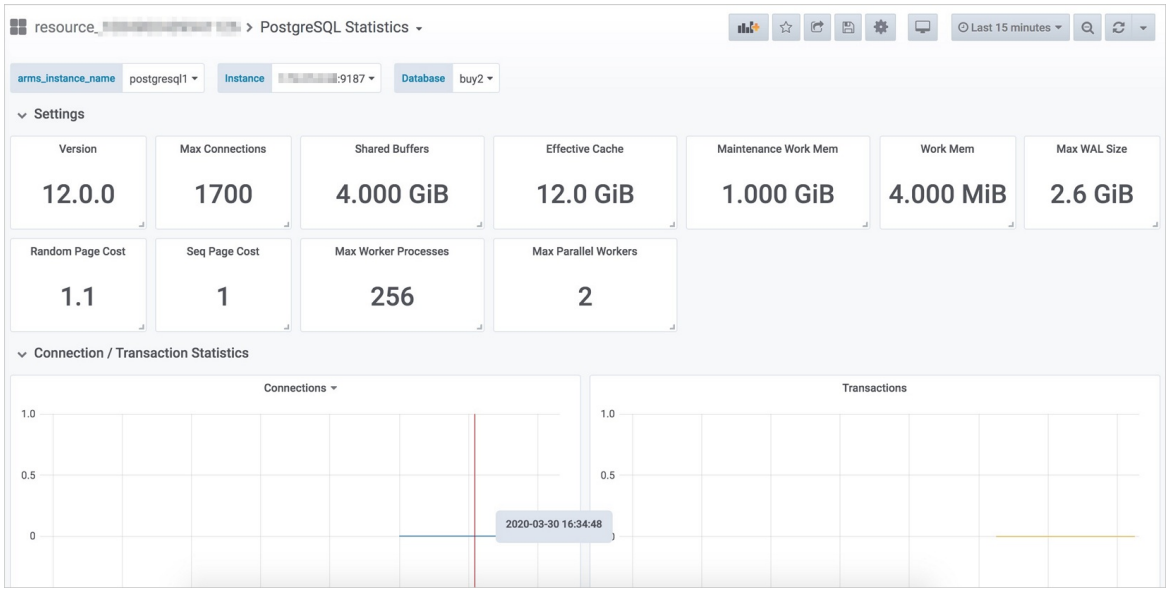
参数	描述

参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
PostgreSQL地址	PostgreSQL的连接地址。
PostgreSQL端口	PostgreSQL的端口号，例如：5432。
用户名	PostgreSQL的用户名称。
密码	PostgreSQL的密码。

说明 在**接入PostgreSQL面板**STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



说明 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.10. 使用阿里云Prometheus监控ZooKeeper

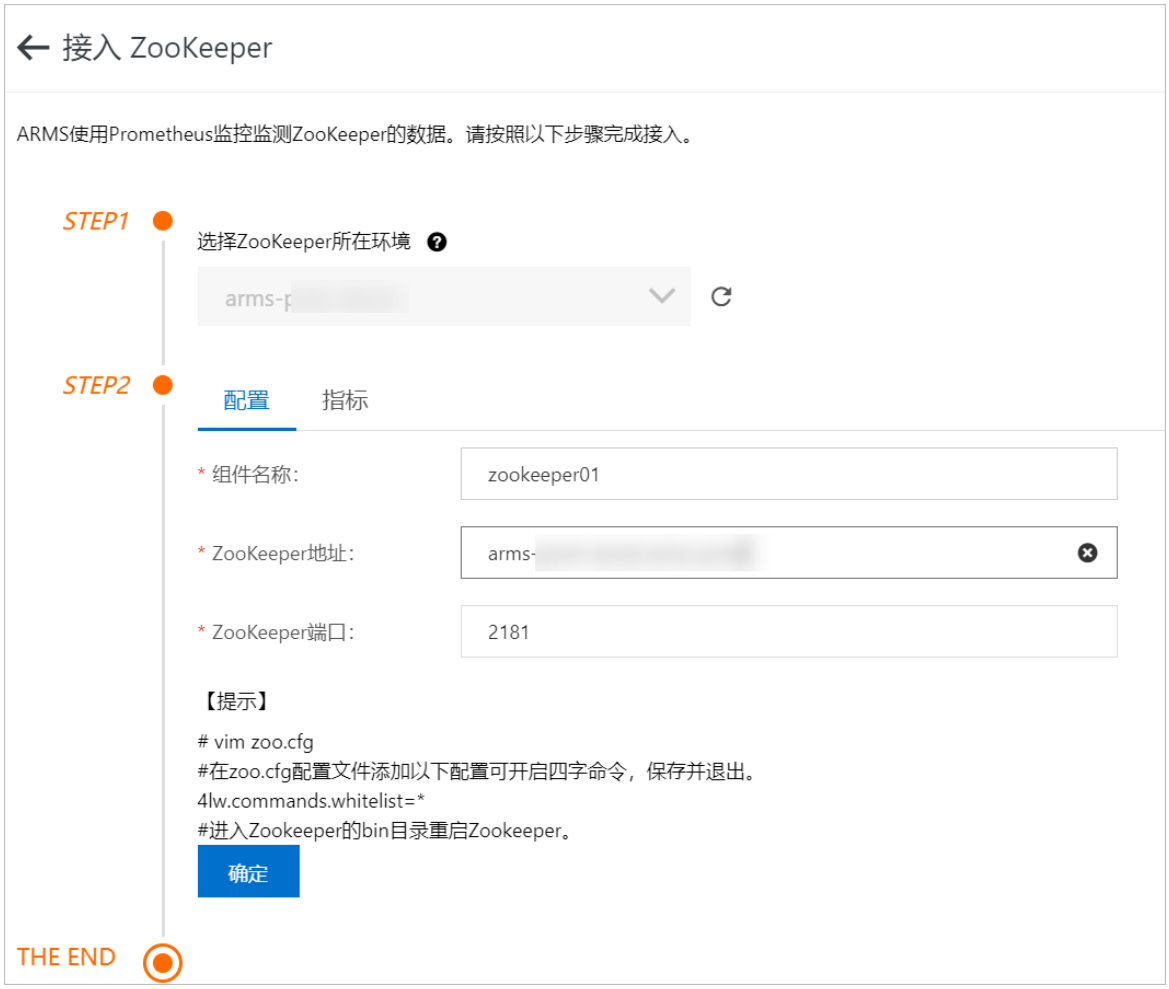
阿里云Prometheus监控提供一键安装配置ZooKeeper类型的组件，并提供开箱即用的专属监控大盘。

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加ZooKeeper类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击ZooKeeper组件图标。
3. 在**接入ZooKeeper**面板STEP2区域的**配置**页签输入各项参数，并单击**确定**。



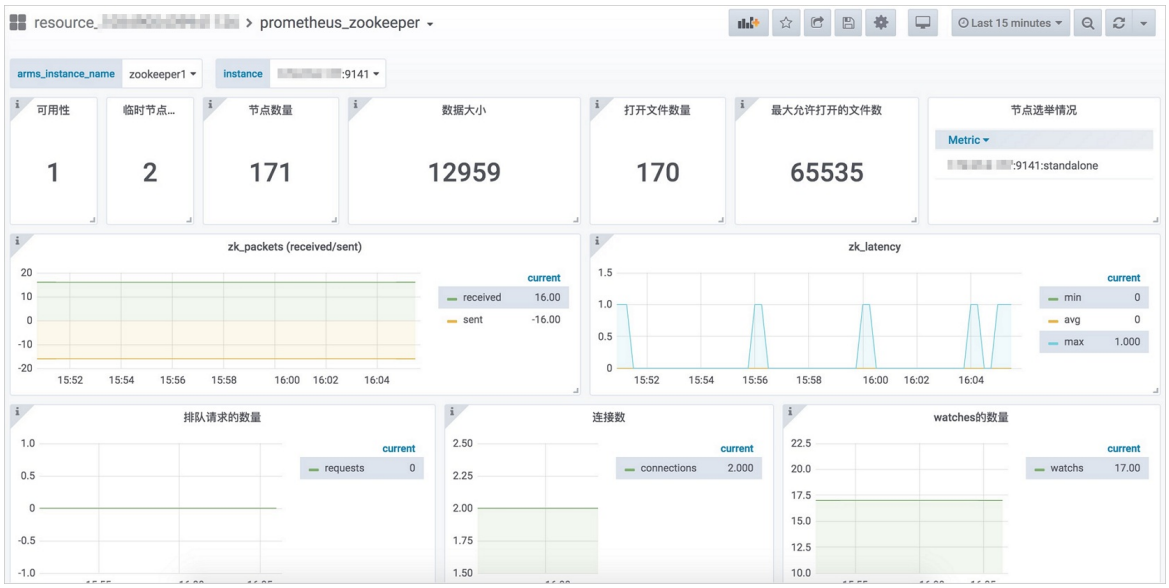
参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
ZooKeeper地址	ZooKeeper的连接地址。
ZooKeeper端口	ZooKeeper的端口号，例如：2181。

说明

- 执行 `vim zoo.cfg` 命令，在*zoo.cfg*配置文件中添加 `4lw.commands.whitelist=*` 配置并保存退出，然后进入ZooKeeper的*bin*目录重启ZooKeeper即可开启四字命令。
- 在**接入ZooKeeper**面板STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> **说明** 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.11. 使用阿里云Prometheus监控Nginx（旧版）

阿里云Prometheus监控提供一键安装配置Nginx类型的组件，并提供开箱即用的专属监控大盘。本文介绍旧版Nginx类型组件的安装配置详情。

背景信息

- 旧版Nginx类型组件安装的是nginx-module-vts模块。
- 旧版Nginx类型组件采集的Nginx指标如下表所示。

指标	类型	描述
nginx_server_requests	Server	Server请求数
nginx_server_bytes	Server	Server字节数

指标	类型	描述
nginx_server_cache	Server	Server缓存
nginx_filter_requests	Filter	Filter请求数
nginx_filter_bytes	Filter	Filter字节数
nginx__filter_responseMsec	Filter	Filter响应时间
nginx_upstream_requests	Upstreams	上行请求数
nginx_upstream_bytes	Upstreams	上行字节数
nginx_upstream_responseMsec	Upstreams	上行响应时间

前提条件

您已成功安装并运行Nginx服务，之后需要安装nginx-module-vts模块。安装nginx-module-vts模块的具体操作如下。

1. 下载nginx-module-vts模块。

```
shell> git clone git://github.com/vozlt/nginx-module-vts.git
```

2. 编译配置。
   i. 在nginx编译时添加nginx-module-vts模块。

```
--add-module=/path/to/nginx-module-vts
```

   ii. 下载官方软件包并编译进nginx-module-vts模块。

```
./configure --user=www --group=www --prefix=/usr/local/nginx --with-http_sysguard_m
odule  --add-module=nginx-module-vts
```

3. 安装nginx-module-vts模块。

```
make && make install
```

4. Nginx Conf配置。更改Nginx Conf的配置，并添加监控接口。

```
http {
    vhost_traffic_status_zone;
  vhost_traffic_status_filter_by_host on;
    ...
    server {
        ...
        location /status {
            vhost_traffic_status_display;
            vhost_traffic_status_display_format html;
        }
    }
}
```

这里建议您同时执行如下命令打开vhost过滤。

```
vhost_traffic_status_filter_by_host on
```

> **说明** 开启此功能，当Nginx配置有多个server_name的情况下，系统会根据不同的server_name进行流量的统计，否则默认会把流量全部计算到第一个server_name上。

若您不需要统计流量的server区域，可以执行如下命令禁用vhost_traffic_status。

```
server {
...
vhost_traffic_status off;
...
}
```

5. 验证nginx-module-vts模版是否安装成功。

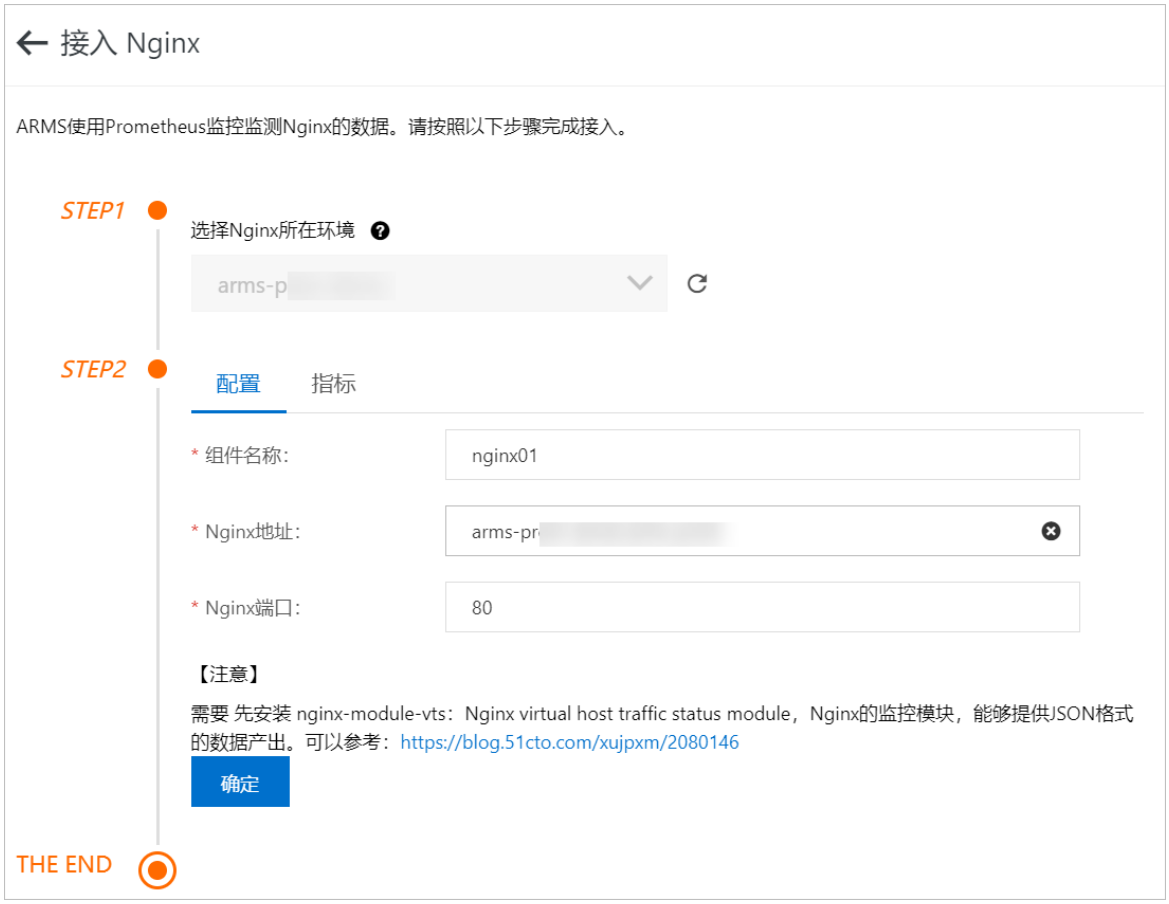
```
curl http://127.0.0.1/status
```

功能入口

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。

添加Nginx类型的组件

1. 在**组件监控**页面，单击右上角的**添加组件监控**。
2. 在**接入中心**面板中单击Nginx组件图标。
3. 在**接入Nginx**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**。



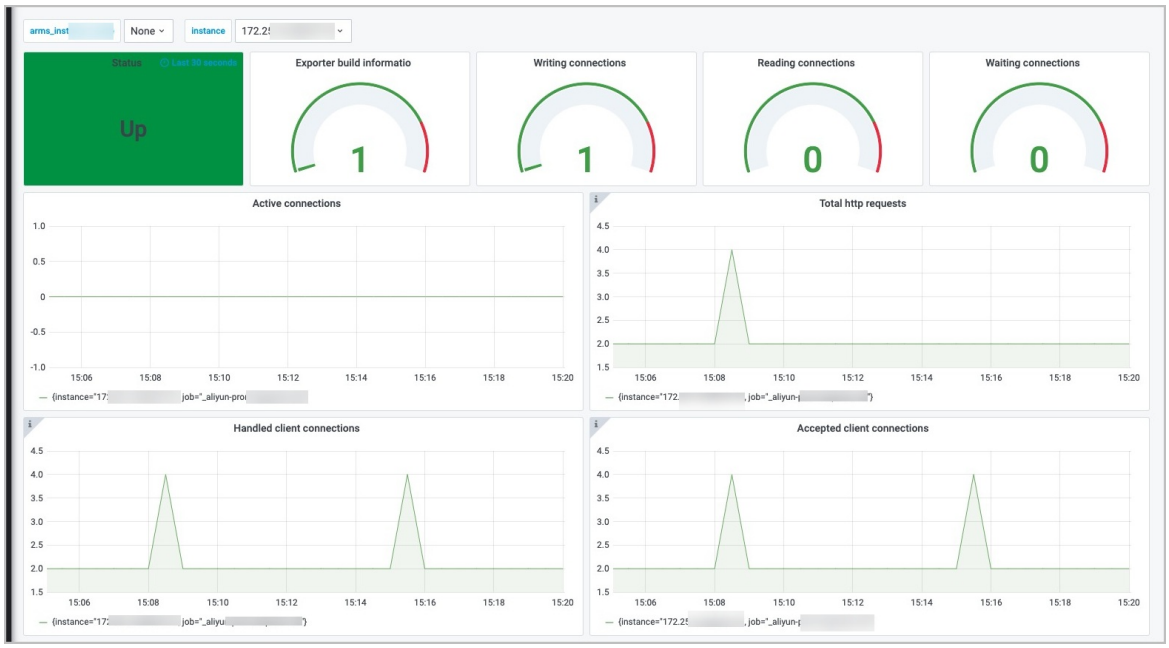
参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
Nginx地址	Nginx的连接地址。
Nginx端口	Nginx的端口号，例如：80。

注意

- 您需要先安装Nginx的监控模块nginx-module-vts：Nginx virtual host traffic status module，此模块可以提供JSON格式的数据产出。
- 在**接入Nginx**面板STEP2区域的**指标**页签可查看监控指标。

4. 在**组件监控**页面，会显示已接入的组件实例。

5. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



> **说明** 单击该组件**实例名称**，也可查看该组件的监控指标数据。

相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.12. 使用阿里云Prometheus监控Nginx（新版）

阿里云Prometheus监控提供一键安装配置Nginx类型的组件，并提供开箱即用的专属监控大盘。本文介绍新版Nginx类型组件的安装配置详情。

前提条件

您已成功安装并运行Nginx服务。

背景信息

- Nginx状态监控模块ngx_http_stub_status_module是统计Nginx服务所接收和处理的请求数量的模块。
- 新版Nginx类型Exporter安装的是ngx_http_stub_status_module模块。
- 新版Nginx类型Exporter采集的Nginx指标如下表所示。

指标	描述
nginx_connections_accepted	接受的客户端连接总数
nginx_connections_active	当前客户端连接数
nginx_connections_handled	Handled状态的连接数
nginx_connections_reading	读取客户端的连接数
nginx_connections_waiting	等待中的客户端连接数
nginx_connections_writing	回写客户端的连接数
nginx_http_requests_total	客户端请求总数
nginx_up	Nginx Exporter是否正常运行
nginxexporter_build_info	Nginx Exporter的构建信息

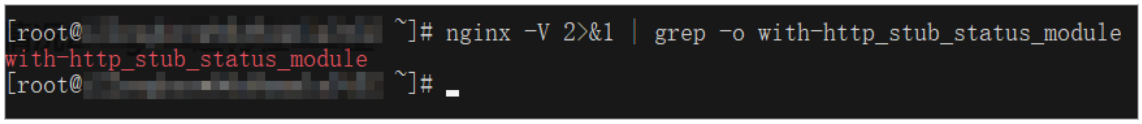
步骤一：安装ngx_http_stub_status_module模块

如果您的Nginx服务运行在云服务器ECS，则按照以下步骤安装Nginx类型的组件。

1. 检查状态监控模块ngx_http_stub_status_module是否已安装。

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

   - 出现以下提示则表示已安装ngx_http_stub_status_module模块。



   - 若未出现以上提示，则说明未安装ngx_http_stub_status_module模块，可执行以下命令安装此模块。

```
wget http://nginx.org/download/nginx-1.13.12.tar.gz
tar xfz nginx-1.13.12.tar.gz
cd nginx-1.13.12/
./configure --with-http_stub_status_module
make
make install
```

2. 启用ngx_http_stub_status_module模块查询Nginx状态。

```
location /nginx_status {
  stub_status on;
  allow 127.0.0.1;  #only allow requests from localhost
  deny all;   #deny all other hosts
 }
```

> **说明**
>
> - Location地址请严格命名为 `nginx_status` 。
> - `allow 127.0.0.1` 和 `deny all` 表示仅允许本地访问。若需允许Nginx Exporter访问，则可将这两行代码注释，或者将 `127.0.0.1` 设置为Nginx Exporter的IP地址。

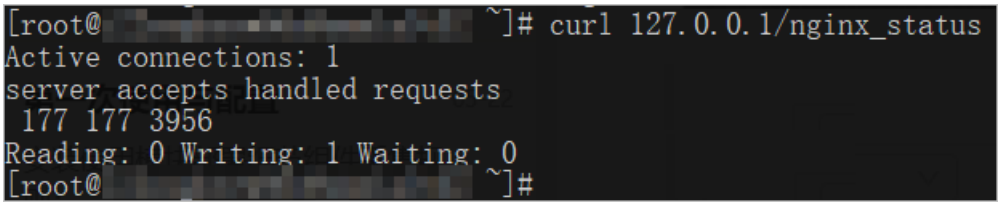
3. 重启Nginx。

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

4. （可选）验证ngx_http_stub_status_module模块是否已成功启动。

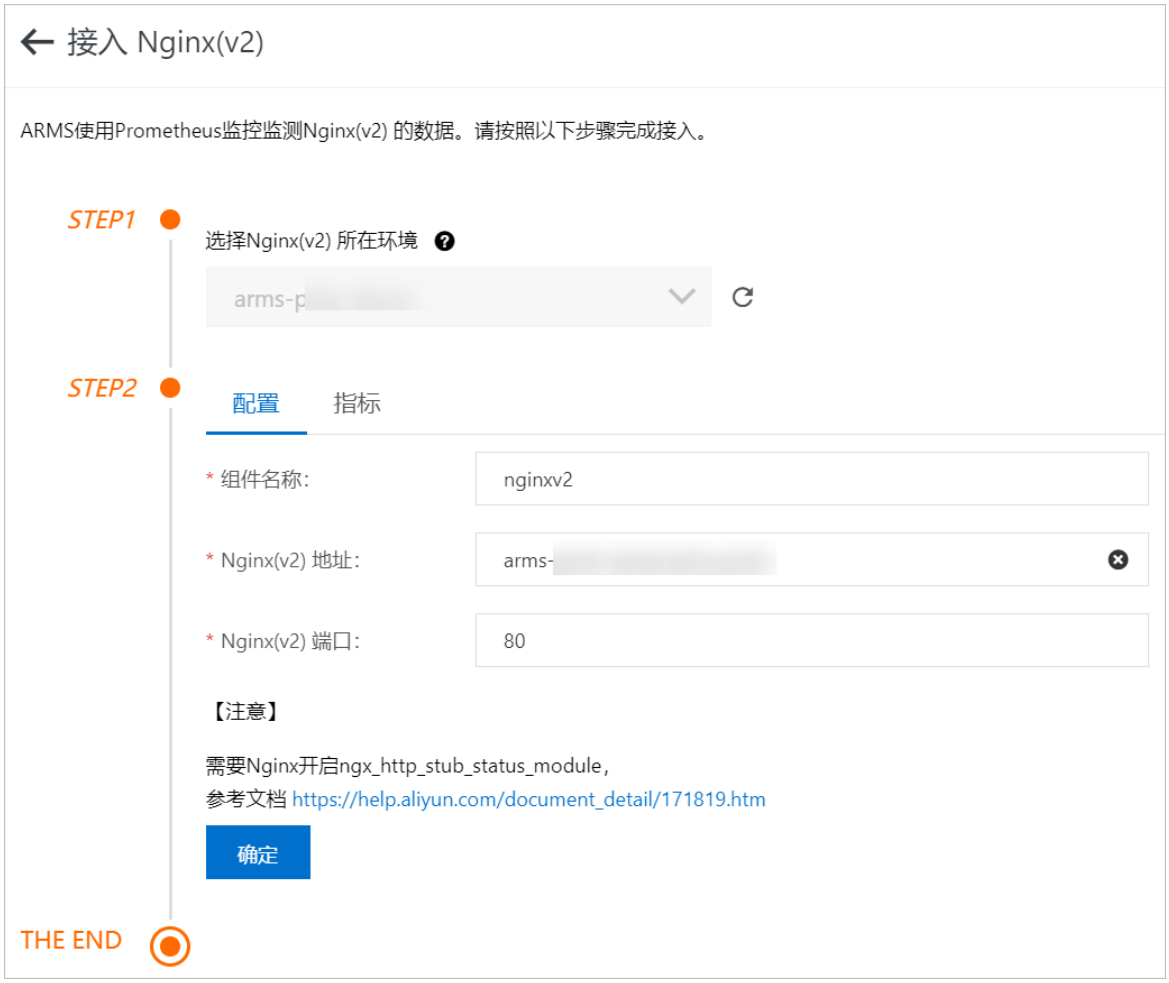
```
curl http://127.0.0.1/nginx_status
```

出现以下提示则表示ngx_http_stub_status_module模块已成功启动。



步骤二：添加新版Nginx类型的组件

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后根据需要单击容器服务、Kubernetes或者ECS类型的Prometheus实例名称。
3. 在左侧导航栏单击**组件监控**。
4. 在**组件监控**页面，单击右上角的**添加组件监控**。
5. 在**接入中心**面板中单击Nginx（v2）组件图标。
6. 在**接入Nginx(v2)**面板**STEP2**区域的**配置**页签输入各项参数，并单击**确定**。

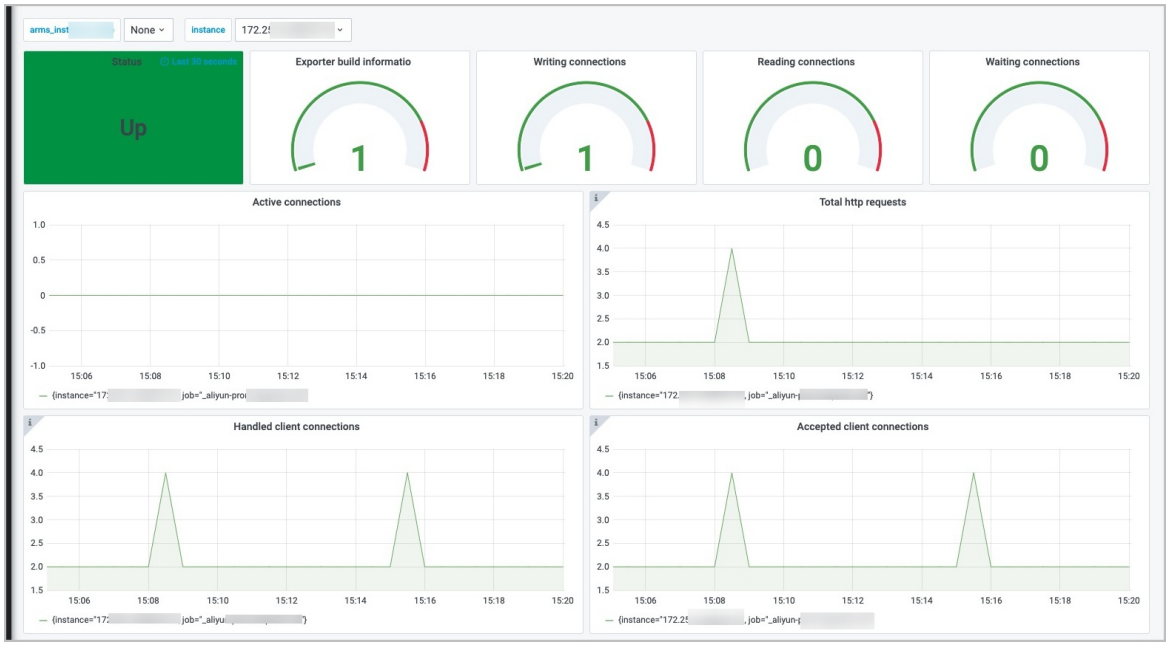


参数	描述
组件名称	组件的名称命名规范要求如下： ○ 仅可包含小写字母、数字和短划线（-），且短划线不可出现在开头或结尾。 ○ 名称具有唯一性。 **说明** 默认名称由组件类型及数字后缀组成。
Nginx(v2) 地址	Nginx的连接地址。
Nginx(v2) 端口	Nginx的端口号，例如：80。

说明 在**接入Nginx(v2)**面板STEP2区域的**指标**页签可查看监控指标。

7. 在**组件监控**页面，会显示已接入的组件实例。

8. 单击该组件实例**大盘**列的大盘，查看该组件的监控指标数据。



相关操作

在**组件监控**页面，可对已添加的组件执行以下操作：

- 单击**操作**列的**删除**，可删除已添加的组件。
- 单击**操作**列的**日志**，可查看组件的运行日志。
- 单击**操作**列的**详情**，可查看组件的详情，包括组件的环境变量和描述信息。

6.13. 其他类型的组件接入

阿里云Prometheus监控提供一键安装和配置数据库类型、消息类型、HTTP服务器类型以及其他类型的组件，并提供开箱即用的专属监控大盘。

阿里云Prometheus将陆续支持其他各种类型组件的一键接入功能，如您所需的组件还未支持，请先使用手动方式安装组件、配置服务发现并创建大盘。本文以MySQL为例，演示其他阿里云Prometheus暂未支持的开源组件的接入操作。

Prometheus组件列表请参见Exporters and integrations。

前提条件

- 已创建阿里云容器服务K8s集群。具体操作，请参见创建Kubernetes专有版集群。
- 阿里云容器服务K8s集群已接入阿里云Prometheus监控。具体操作，请参见Prometheus实例 for 容器服务。
- VPC实例已接入阿里云Prometheus监控。具体操作，Prometheus实例 for VPC。

操作流程

通过阿里云Prometheus手动监控MySQL的操作流程如下图所示。

步骤一：部署应用

将Prometheus官方提供的mysqld-exporter镜像部署至容器服务K8s集群，以便抓取MySQL数据。操作步骤如下：

1. 登录容器服务管理控制台。
2. 在左侧导航栏，选择**集群**。
3. 在**集群列表**页面，找到目标集群，在其右侧**操作**列单击**应用管理**。
4. 创建容器组。
   i. 在左侧导航栏，选择**工作负载 > 无状态**。
   ii. 在**无状态**页面，单击**使用YAML创建资源**。
   iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

> **说明** 请将*<yourMySQLUsername>*、*<yourMySQLPassword>*、*<IP>*、*<port>*替换为MySQL的对应值。

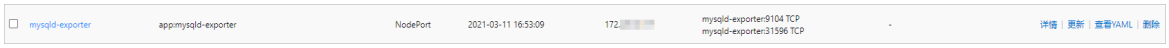
```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: mysqld-exporter
  labels:
    app: mysqld-exporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysqld-exporter
  template:
    metadata:
      labels:
        app: mysqld-exporter
    spec:
      containers:
      - name: mysqld-exporter
        imagePullPolicy: Always
        env:
          - name: DATA_SOURCE_NAME
            value: "<yourMySQLUsername>:<yourMySQLPassword>@(<IP>:<port>)/"
        image: prom/mysqld-exporter
        ports:
        - containerPort: 9104
          name: mysqld-exporter
```

无状态页面显示创建的容器组。

5. 创建服务。
   i. 在左侧导航栏，选择**网络 > 服务**。
   ii. 在**服务**页面，单击**使用YAML创建资源**。
   iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: mysqld-exporter
  name: mysqld-exporter
spec:
  ports:
  - name: mysqld-exporter
    port: 9104
    protocol: TCP
    targetPort: 9104
  type: NodePort
  selector:
    app: mysqld-exporter
```

服务页面显示创建的服务。



步骤二：配置服务发现

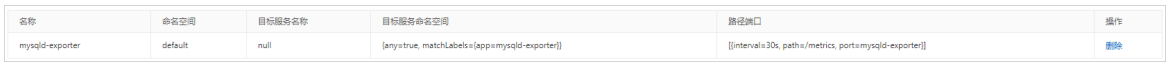
配置阿里云Prometheus监控的服务发现以接收MySQL数据的操作步骤如下：

> **注意** 请确认：
> - 阿里云容器服务K8s集群已接入阿里云Prometheus监控。具体操作，请参见Prometheus实例 for 容器服务。
> - VPC实例已接入阿里云Prometheus监控。具体操作，Prometheus实例 for VPC。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，并在目标集群右侧的**操作**列单击**设置**。
3. 在设置页面，单击**服务发现**页签，在**配置**页签下，单击**ServiceMonitor**页签。
4. 在**ServiceMonitor**页签下，单击**添加ServiceMonitor**。
5. 在**添加ServiceMonitor**对话框中输入以下内容，然后单击**确定**。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  #  填写一个唯一名称
  name: tomcat-demo
  #  填写目标命名空间
  namespace: default
spec:
  endpoints:
  - interval: 30s
    #  填写service.yaml中Prometheus Exporter对应的Port的Name字段的值
    port: tomcat-monitor
    #  填写Prometheus Exporter对应的Path的值
    path: /metrics
  namespaceSelector:
    any: true
    #  Nginx Demo的命名空间
  selector:
    matchLabels:
      #  填写service.yaml的Label字段的值以定位目标service.yaml
      app: tomcat
```

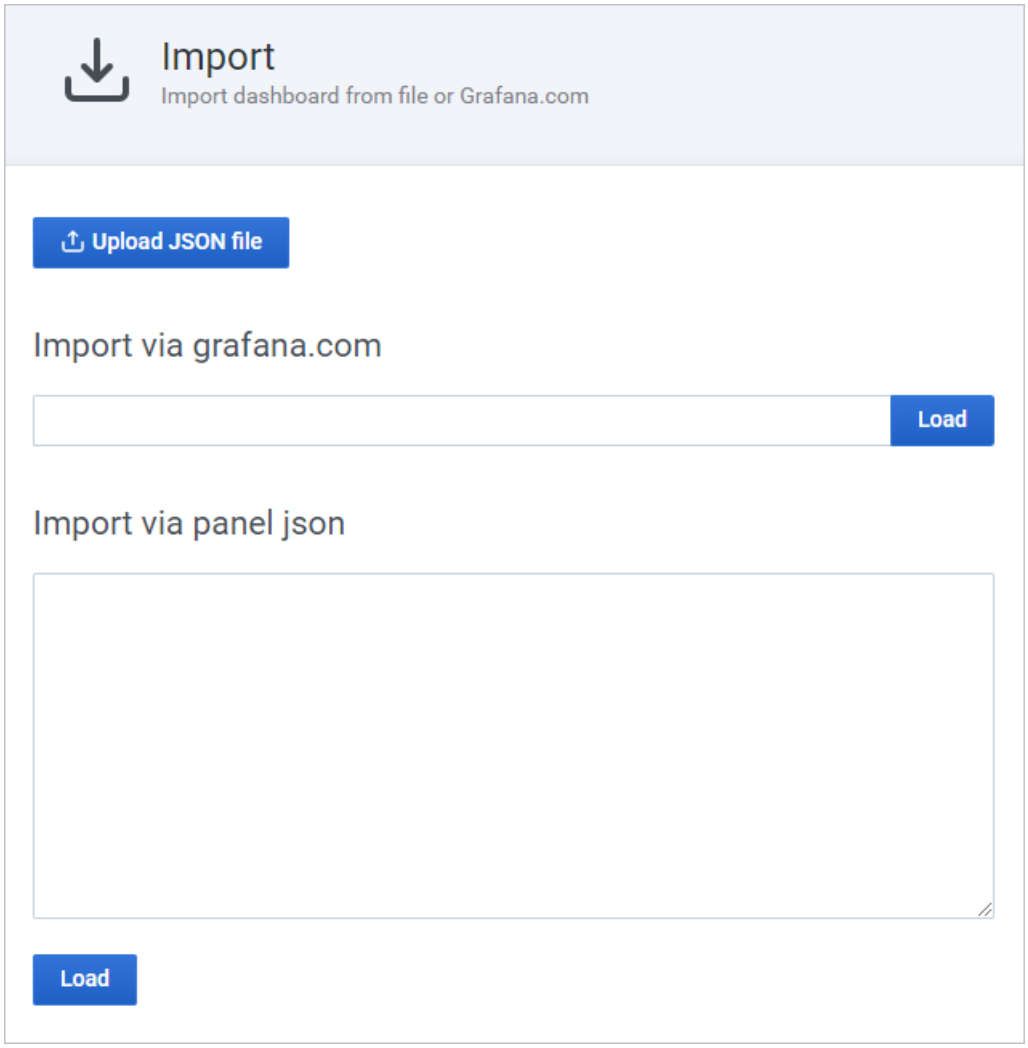
ServiceMonitor页签下显示配置的服务发现。



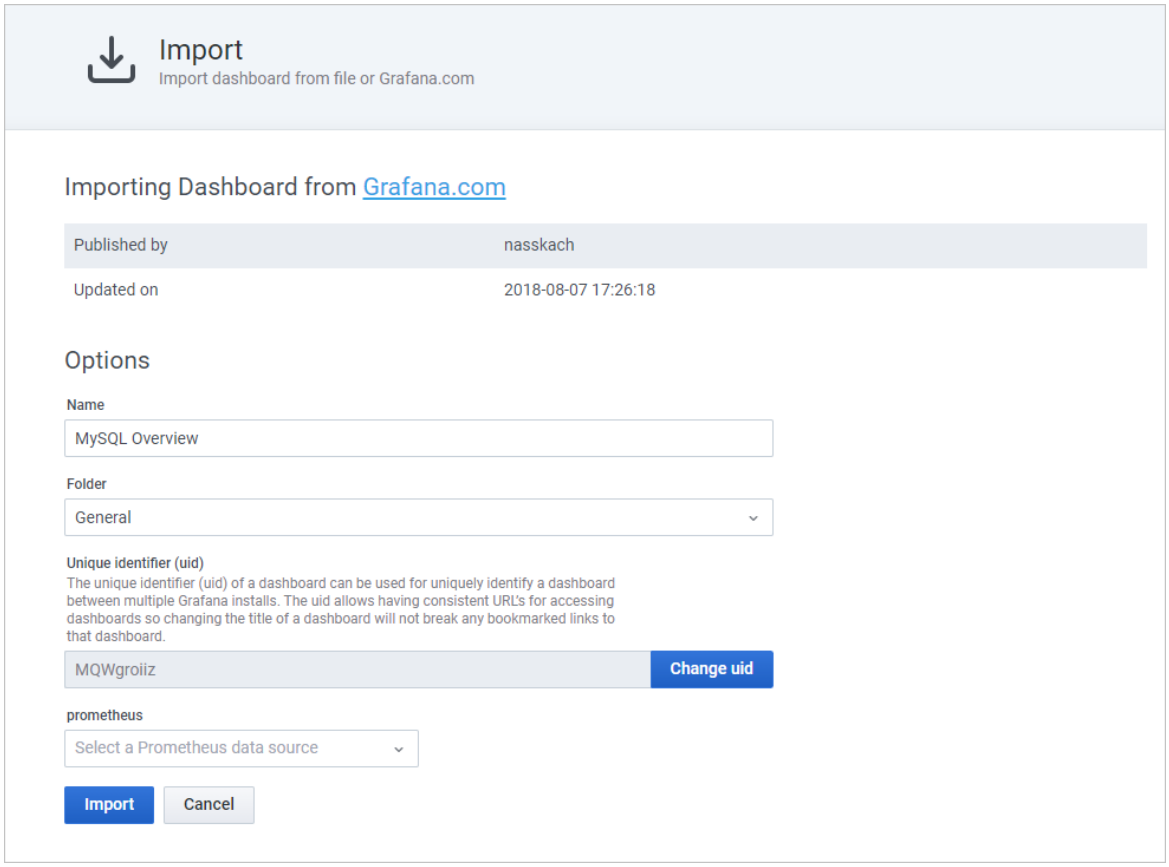
步骤三：配置大盘

配置Grafana大盘以展示数据的操作步骤如下：

1. 打开Grafana大盘概览页。
2. 在左侧导航栏选择+ > Import。
3. 在Import页面的Import via grafna.com文本框，输入Prometheus提供的MySQL大盘模板的ID*7362*，然后在其右侧单击Load。



4. 在Import页面设置以下信息，然后单击Import。

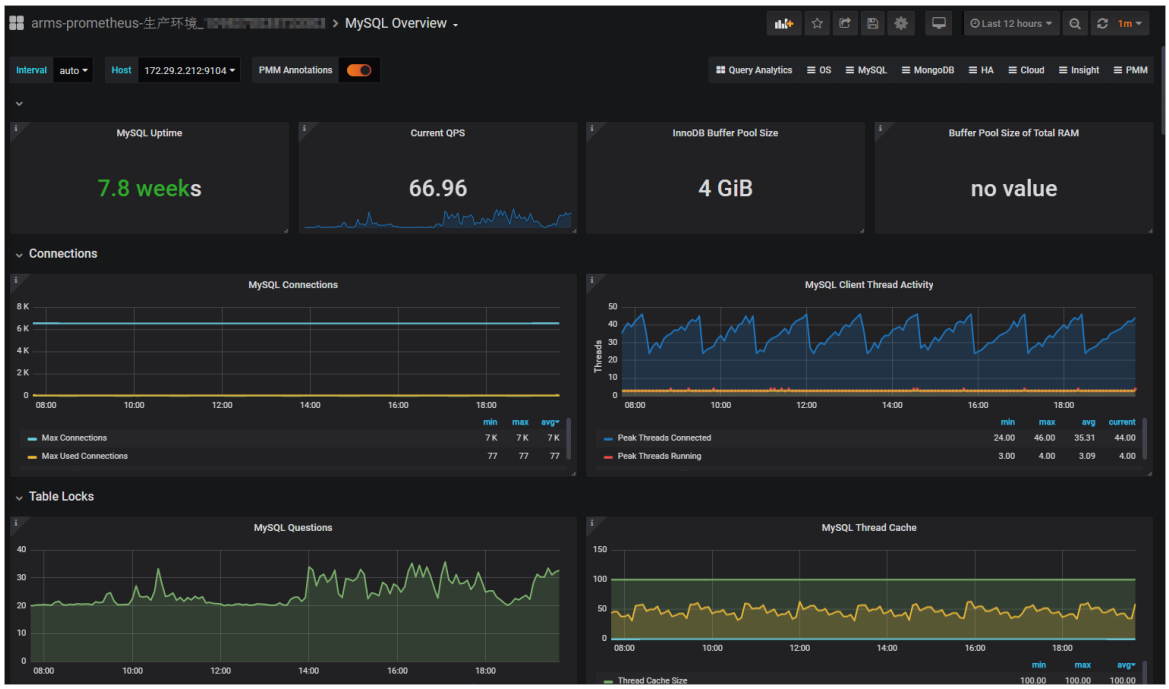


i. 在**Name**文本框中输入自定义的大盘名称。

ii. 在**Folder**下拉列表中选择您的阿里云容器服务K8s集群。

iii. 在**prometheus**下拉列表中选择您的阿里云容器服务K8s集群。

配置完毕后的Grafana大盘如图所示。



步骤四：创建Prometheus监控报警

为监控指标创建报警的操作步骤如下：

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
3. 在左侧导航栏，选择**报警配置**。
4. 在报警配置页面右上角，单击**创建报警**。
5. 在**创建报警**面板，执行以下操作：
   i. （可选）从**告警模板**下拉列表，选择模板。
   ii. 在**规则名称**文本框，输入规则名称，例如：网络接收压力报警。
   iii. 在**告警表达式**文本框，输入告警表达式。例如：`(sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m]))) > 0.01`。

   > **注意** PromQL语句中包含的 `$` 符号会导致报错，您需要删除包含$符号的语句中 `=` 左右两边的参数及 `=` 。例如：将 `sum (rate (container_network_receive_bytes_total{instance=~"^$HostIp.*"}[1m]))` 修改为 `sum (rate (container_network_receive_bytes_total[1m]))` 。

   iv. 在**持续时间**文本框，输入持续时间N，当连续N分钟满足告警条件的时候才触发告警。例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。

   > **说明** 持续N分钟满足告警条件是指在连续N分钟内，您上面设置的PromQL语句条件都能满足。Prometheus默认数据采集周期为15s，如果没有连续N×60/15=4N个数据点满足告警条件（PromQL语句），就不会发送告警。如Prometheus告警规则默认持续时间为1分钟，既只有连续4个数据点都满足告警条件（PromQL语句）才会触发告警。如果您想在任何一个数据点满足告警条件（PromQL语句）就发送告警，请修改持续时间为0分钟。

   v. 在**告警消息**文本框，输入告警消息。
   vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。
   vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息`，例如：`message:{{$labels.pod_name}}重启`。

   您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：
   viii. 从**通知策略**下拉列表，选择通知策略。

   如何创建通知策略，请参见通知策略。
   ix. 单击**确定**。

报警配置页面显示创建的报警。

7.客户端接入

7.1. 通过阿里云Prometheus监控JVM

通过在应用中埋点来暴露JVM数据，使用阿里云Prometheus监控采集JVM数据，借助Prometheus Grafana大盘来展示JVM数据，并创建报警，即可实现利用Prometheus监控JVM的目的。本文以阿里云容器服务K8s集群和阿里云容器镜像服务为例，介绍如何通过Prometheus监控JVM。

前提条件

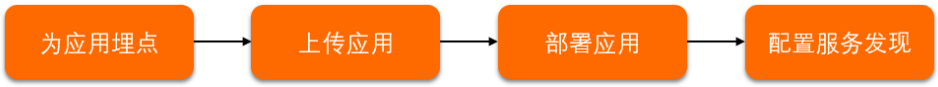
- 阿里云容器服务K8s集群已接入阿里云Prometheus监控。如何接入，请参见Prometheus实例 for 容器服务。
- 已创建阿里云容器镜像服务镜像仓库。如何创建，请参见步骤二：创建镜像仓库。

Demo

如需快速体验如何通过Prometheus监控JVM，您可以使用已埋点的Demo项目。

操作流程

通过阿里云Prometheus监控JVM的操作流程如下图所示。



步骤一：为应用埋点

为应用埋点以暴露JVM数据的操作步骤如下：

1. 在*pom.xml*文件中添加Maven依赖。

```
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>simpleclient_hotspot</artifactId>
    <version>0.6.0</version>
</dependency>
```

2. 在应用代码中添加初始化JVM Exporter的方法。

```
@PostConstruct
    public void initJvmExporter() {
        io.prometheus.client.hotspot.DefaultExports.initialize();
    }
```

   您可以参考Demo项目的*/src/main/java/com/monitise/prometheus_demo/DemoController.java*文件。

3. 在*application.properties*文件中配置用于Prometheus监控的端口和路径。

```
management.port: 8081
endpoints.prometheus.path: prometheus-metrics
```

   您可以参考Demo项目的*/src/main/resources/application.properties*文件。

4. 在应用代码中添加打开HTTP端口的方法。

```
@SpringBootApplication
// sets up the prometheus endpoint /prometheus-metrics
@EnablePrometheusEndpoint
// exports the data at /metrics at a prometheus endpoint
@EnableSpringBootMetricsCollector
public class PrometheusDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(PrometheusDemoApplication.class, args);
    }
}
```

您可以参考Demo项目的*/src/main/java/com/monitise/prometheus_demo/PrometheusDemoApplication.java*文件。

步骤二：上传应用

将完成埋点的应用制作成镜像并上传至阿里云容器镜像服务的镜像仓库的操作步骤如下：

1. 执行以下命令重新编译模块。

   ```
   mvn clean install -DskipTests
   ```

2. 执行以下命令构建镜像。

   ```
   docker build -t <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> . --no-cache
   ```

   示例命令：

   ```
   docker build -t promethues-demo:v0 . --no-cache
   ```

3. 执行以下命令为镜像打标。

   ```
   sudo docker tag <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
   ```

   示例命令：

   ```
   sudo docker tag promethues-demo:v0 registry.cn-hangzhou.aliyuncs.com/testnamespace/promethues-demo:v0
   ```

4. 执行以下命令将镜像推送至镜像仓库。

   ```
   sudo docker push <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
   ```

   示例命令：

   ```
   sudo docker push registry.cn-hangzhou.aliyuncs.com/testnamespace/promethues-demo:v0
   ```

   容器镜像服务控制台的**镜像版本**页面显示上传的应用镜像。

步骤三：部署应用

将应用部署至容器服务K8s集群的操作步骤如下：

1. 登录容器服务管理控制台。

2. 在左侧导航栏，选择**集群**。
3. 在**集群列表**页面，找到目标集群，在其右侧**操作**列单击**应用管理**。
4. 创建容器组。
   i. 在左侧导航栏，选择**工作负载 > 无状态**。
   ii. 在**无状态**页面，单击**使用YAML创建资源**。
   iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: prometheus-demo
spec:
  replicas: 2
  template:
    metadata:
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/prometheus-metrics'
        prometheus.io/port: '8081'
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        imagePullPolicy: Always
        image: <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
        ports:
        - containerPort: 8080
          name: tomcat-normal
        - containerPort: 8081
          name: tomcat-monitor
```

示例代码：

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: prometheus-demo
  labels:
    app: tomcat
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/prometheus-metrics'
        prometheus.io/port: '8081'
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        imagePullPolicy: Always
        image: registry.cn-hangzhou.aliyuncs.com/peiyu-test/prometheus-demo:v0
        ports:
        - containerPort: 8080
          name: tomcat-normal
        - containerPort: 8081
          name: tomcat-monitor
```

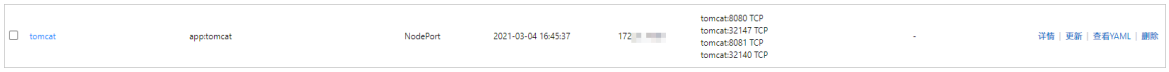
无状态页面显示创建的容器组。

5. 创建服务。
   i. 在左侧导航栏，选择**网络 > 服务**。
   ii. 在**服务**页面，单击**使用YAML创建资源**。

iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: tomcat
  name: tomcat
  namespace: default
spec:
  ports:
  - name: tomcat-normal
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: tomcat-monitor
    port: 8081
    protocol: TCP
    targetPort: 8081
  type: NodePort
  selector:
    app: tomcat
```

服务页面显示创建的服务。



步骤四：配置服务发现

通过ServiceMonitor配置阿里云Prometheus监控的服务发现以采集JVM数据。具体操作，请参见通过ServiceMonitor创建服务发现。

7.2. 通过阿里云Prometheus监控Go应用

通过在应用中埋点来暴露应用数据，使用阿里云Prometheus监控抓取数据，借助Prometheus Grafana大盘来展示数据，并创建报警，即可实现利用Prometheus监控Go应用的目的。本文以阿里云容器服务K8s集群和阿里云容器镜像服务为例，介绍如何通过Prometheus监控Go应用。

前提条件

在开始本教程前，确保您已经完成了以下操作：

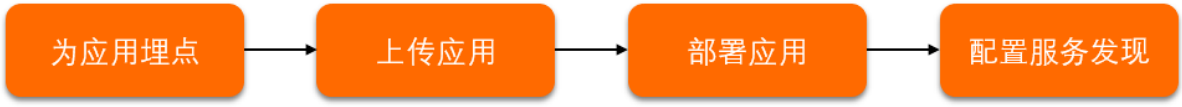
- 阿里云容器服务K8s集群已接入阿里云Prometheus监控。如何接入，请参见Prometheus实例 for 容器服务。
- 已创建阿里云容器镜像服务镜像仓库。如何创建，请参见步骤二：创建镜像仓库。

Demo

如需快速体验如何通过Prometheus监控Go应用，您可以使用已埋点的Demo工程。

操作流程

通过阿里云Prometheus监控Go应用的操作流程如下图所示。



步骤一：为应用埋点

为应用埋点以暴露Go应用数据的操作步骤如下：

1. 将监控包导入Go应用。

```
import (
    "fmt"
    "github.com/prometheus/client_golang/prometheus/promhttp"
    "net/http"
    "strconv"
)
```

2. 将监控接口绑定至promhttp.Handler()。

```
http.Handle(path, promhttp.Handler()) //初始化一个HTTP Handler。
```

步骤二：上传应用

将完成埋点的应用制作成镜像并上传至阿里云容器镜像服务的镜像仓库的操作步骤如下：

1. 执行以下命令重新编译模块。

```
go build
```

2. 执行以下命令构建镜像。

```
docker build -t <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> . --no-cache
```

示例命令：

```
docker build -t prometheus-go-demo:v0 . --no-cache
```

3. 执行以下命令为镜像打标。

```
sudo docker tag <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
```

示例命令：

```
sudo docker tag prometheus-go-demo:v0 registry.cn-hangzhou.aliyuncs.com/testnamespace/prometheus-go-demo:v0
```

4. 执行以下命令将镜像推送至镜像仓库。

```
sudo docker push <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
```

示例命令：

```
sudo docker push registry.cn-hangzhou.aliyuncs.com/testnamespace/prometheus-go-demo:v0
```

容器镜像服务控制台的**镜像版本**页面显示上传的应用镜像。

步骤三：部署应用

将应用部署至容器服务K8s集群的操作步骤如下：

1. 登录容器服务管理控制台。
2. 在左侧导航栏，选择**集群**。
3. 在**集群列表**页面，找到目标集群，在其右侧**操作**列单击**应用管理**。
4. 创建容器组。
   i. 在左侧导航栏，选择**工作负载 > 无状态**。
   ii. 在**无状态**页面，单击**使用YAML创建资源**。
   iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: prometheus-go-demo
  labels:
    app: go-exporter
spec:
  replicas: 2
  selector:
    matchLabels:
      app: go-exporter
  template:
    metadata:
      labels:
        app: go-exporter
    spec:
      containers:
      - name: prometheus-go-demo
        imagePullPolicy: Always
        image: <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
        ports:
        - containerPort: 8077
          name: arms-go-demo
```

示例代码：

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: prometheus-go-demo
  labels:
    app: go-exporter
spec:
  replicas: 2
  selector:
    matchLabels:
      app: go-exporter
  template:
    metadata:
      labels:
        app: go-exporter
    spec:
      containers:
      - name: prometheus-go-demo
        imagePullPolicy: Always
        image: registry.cn-hangzhou.aliyuncs.com/fuling/prometheus-go-demo:v0
        ports:
        - containerPort: 8077
          name: arms-go-demo
```

无状态页面显示创建的容器组。

5. 创建服务。
   i. 在左侧导航栏，选择**网络 > 服务**。
   ii. 在**服务**页面，单击**使用YAML创建资源**。
   iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prometheus-go-demo
  name: prometheus-go-demo
spec:
  ports:
  - name: arms-go-demo
    port: 8077
    protocol: TCP
    targetPort: 8077
  type: NodePort
  selector:
    app: prometheus-go-demo
```

服务页面显示创建的服务。

步骤四：配置服务发现

配置阿里云Prometheus监控的服务发现以抓取Go应用数据的操作步骤如下：

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标实例名称。
3. 在左侧导航树单击**服务发现**，然后单击**配置**页签。
4. 在**配置**页面单击**ServiceMonitor**页签，然后单击**添加ServiceMonitor**。
5. 在**添加ServiceMonitor**对话框中输入以下内容，然后单击**确定**。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  # 填写一个唯一名称
  name: prometheus-go-demo
  # 填写目标命名空间
  namespace: default
spec:
  endpoints:
  - interval: 30s
    # 填写Prometheus Exporter对应的Port的Name字段的值
    port: arms-go-demo
    # 填写Prometheus Exporter对应的Path的值
    path: /metrics
  namespaceSelector:
    any: true
    # Demo的命名空间
  selector:
    matchLabels:
      app: prometheus-go-demo
```

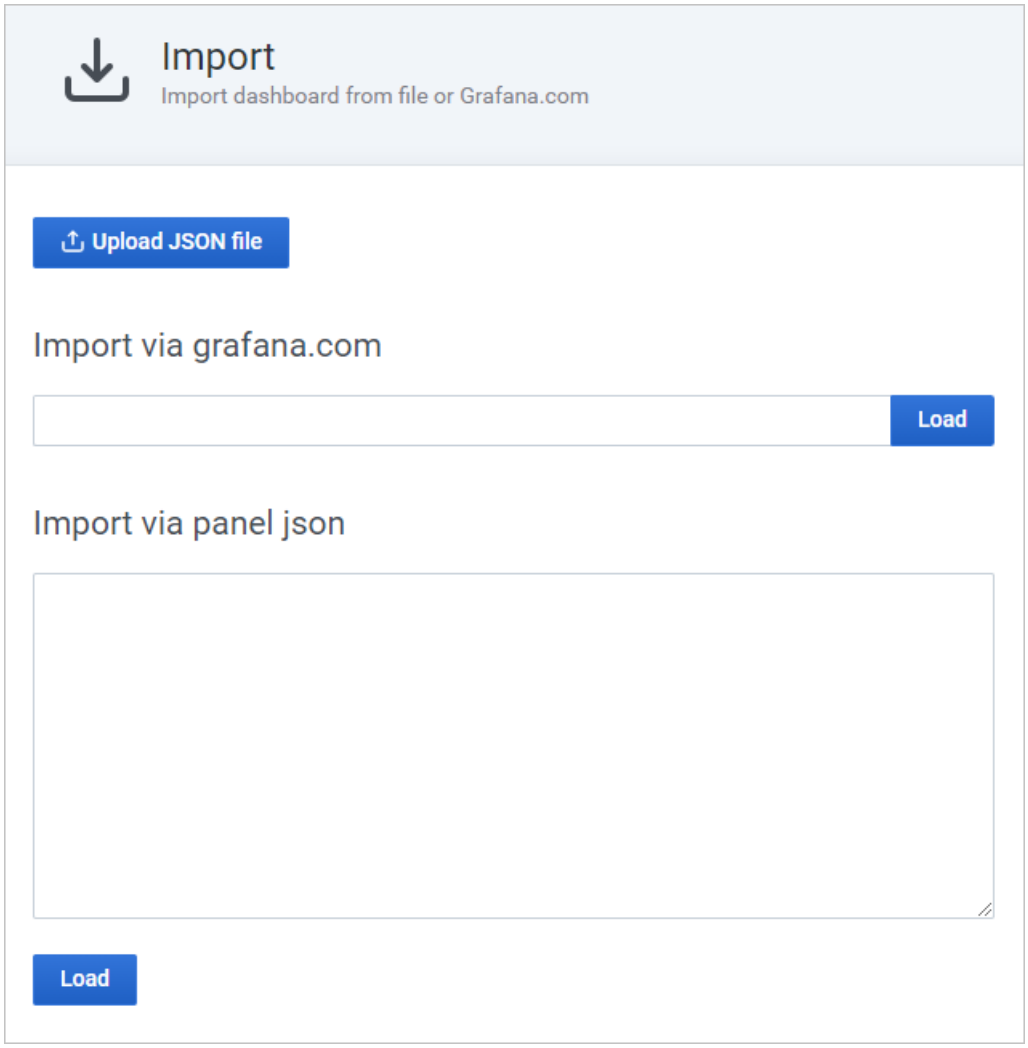
ServiceMonitor页签下显示配置的服务发现。

名称	命名空间	目标服务名称	目标服务命名空间	路径端口	操作
prometheus-go-demo	default	{matchLabels:{app:prometheus-go-demo}}	{any:true}	[{interval:30s, path:/metrics, port:arms-go-demo}]	删除

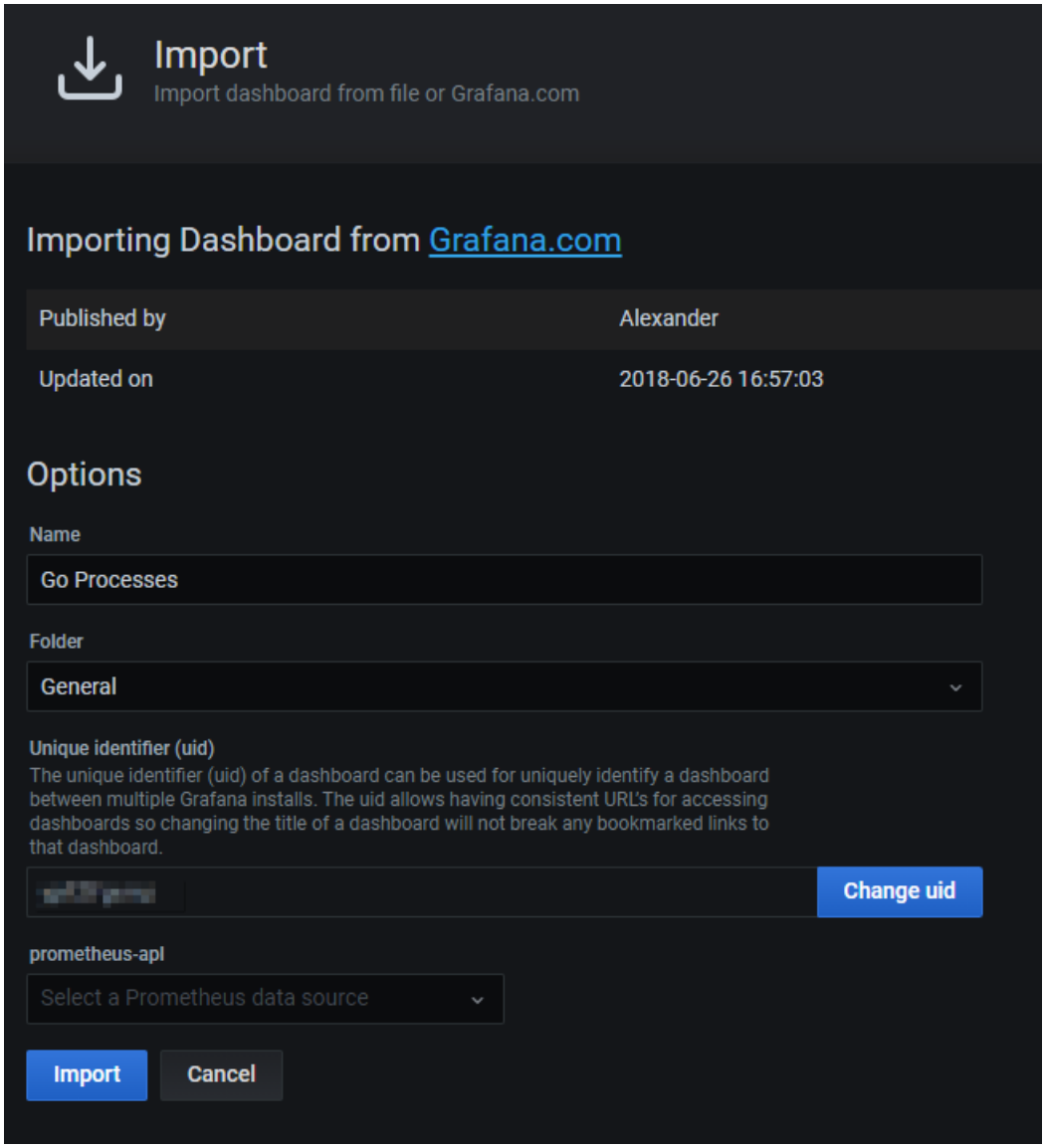
步骤五：配置大盘

配置Grafana大盘以展示数据的操作步骤如下：

1. 打开Grafana大盘概览页。
2. 在左侧导航栏选择**+ > Import**。
3. 在**Import**页面的**Import via grafna.com**文本框，输入Prometheus提供的Go应用大盘模板ID*6671*，然后在其右侧，单击**Load**。



4. 在Import页面输入以下信息，然后单击Import。

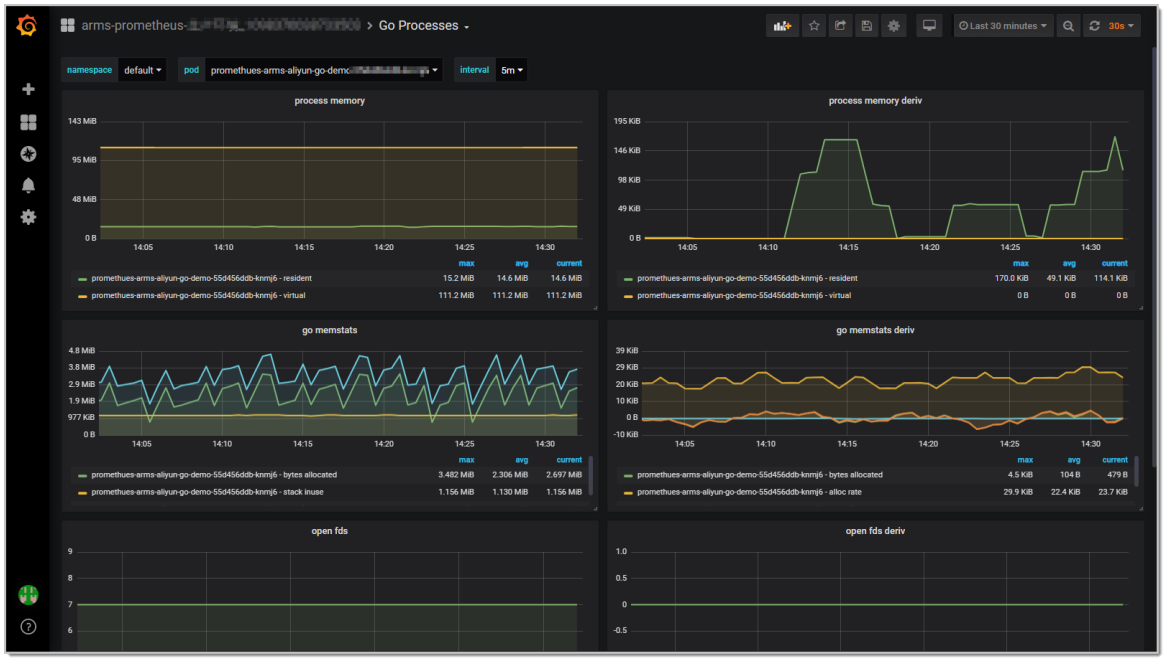


i. 在**Name**文本框中输入自定义的大盘名称。

ii. 从**Folder**列表中，选择您的阿里云容器服务K8s集群。

iii. 在**prometheus-apl**下拉框中选择您的阿里云容器服务K8s集群。

配置完毕后的Grafana大盘如图所示。



步骤六：创建Prometheus监控报警

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
3. 在左侧导航栏，选择**报警配置**。
4. 在报警配置页面右上角，单击**创建报警**。
5. 在**创建报警**面板，执行以下操作：
   i. （可选）从**告警模板**下拉列表，选择模板。
   ii. 在**规则名称**文本框，输入规则名称，例如：网络接收压力报警。
   iii. 在**告警表达式**文本框，输入告警表达式。例如：`(sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m]))) > 0.01`。

   > 注意　PromQL语句中包含的`$`符号会导致报错，您需要删除包含$符号的语句中`=`左右两边的参数及`=`。例如：将`sum (rate (container_network_receive_bytes_total{instance=~"^$HostIp.*"}[1m]))`修改为`sum (rate (container_network_receive_bytes_total[1m]))`。

   iv. 在**持续时间**文本框，输入持续时间N，当连续N分钟满足告警条件的时候才触发告警。例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。

   > 说明　持续N分钟满足告警条件是指在连续N分钟内，您上面设置的PromQL语句条件都能满足。Prometheus默认数据采集周期为15s，如果没有连续N×60/15=4N个数据点满足告警条件（PromQL语句），就不会发送告警。如Prometheus告警规则默认持续时间为1分钟，既只有连续4个数据点都满足告警条件（PromQL语句）才会触发告警。如果您想在任何一个数据点满足告警条件（PromQL语句）就发送告警，请修改持续时间为0分钟。

   v. 在**告警消息**文本框，输入告警消息。

vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。

vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息`，例如：`message:{{$labels.pod_name}}重启`。

您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：

viii. 从**通知策略**下拉列表，选择通知策略。

如何创建通知策略，请参见通知策略。

ix. 单击**确定**。

报警配置页面显示创建的报警。

8.云服务监控接入

8.1. 管理云服务监控接入

创建云服务类型的Prometheus实例后，您可以在**云服务采集列表**页面为该实例增加或删除云服务。本文介绍如何管理云服务的接入。

前提条件

云服务已接入Prometheus监控。具体操作，请参见Prometheus实例 for 云服务。

操作步骤

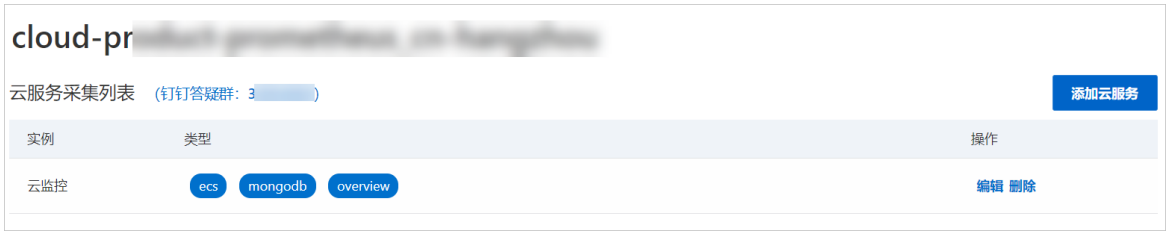
1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。

   Prometheus监控页面显示了所有Prometheus实例，其中监控对象类型为云服务的实例为Prometheus云服务实例。

3. 单击Prometheus云服务实例名称。
4. 在左侧导航栏单击**云服务接入**。

   云服务采集列表页面显示当前已经接入的云服务。

   > **说明** 同时支持在**Prometheus监控**页面通过选择容器服务类型的Prometheus实例或者Kubernetes类型的Prometheus实例，然后在**云服务接入** > **云服务采集列表**页面调整云服务接入。

5. 在右侧**云服务采集列表**页面，可以根据需求调整云服务接入。
   - 如果需要增加或删除云服务，单击云服务右侧**操作**列的**编辑**或单击**云服务采集列表**右上角的**添加云服务**，在**接入云服务**面板的**云服务**页签中重新选择云服务，然后单击**确定**。

     > **说明** 调整云服务接入时，在**云服务**页签右上角会自动勾选**同时采集云服务标签指标**，勾选后采集的云服务监控指标会关联上配置的标签。若您不需要监控指标关联标签，可自行取消勾选。

   - 如果需要删除所有云服务的接入，单击云服务右侧**操作**列的**删除**，在弹出的**确认**对话框中单击**确认**。

> **说明** 删除所有云服务的接入后，并不会删除云服务类型的Prometheus实例，因此在**Prometheus监控**页面依然会显示云服务类型的Prometheus实例。

8.2. 接入并监控云服务（旧版）

阿里云Prometheus监控集成了阿里云云监控，您可以在Prometheus监控中监控当前云账号在指定地域下的云服务。

接入说明

Prometheus监控现已支持直接接入云服务，无需依赖Kubernetes集群。具体详情，请参见Prometheus实例 for 云服务。

您仍可以继续使用原有Kubernetes云服务监控功能，但此功能将逐步停止服务，推荐您使用直接接入云服务的方式实现监控和报警。

9.管理大盘

9.1. 大盘列表

阿里云Prometheus监控通过Grafana大盘展示监控的指标数据。在**大盘列表**页面，您可以查看不同类型的Prometheus实例下的所有大盘信息和对应的核心指标，并根据需要对目标大盘进行升级。

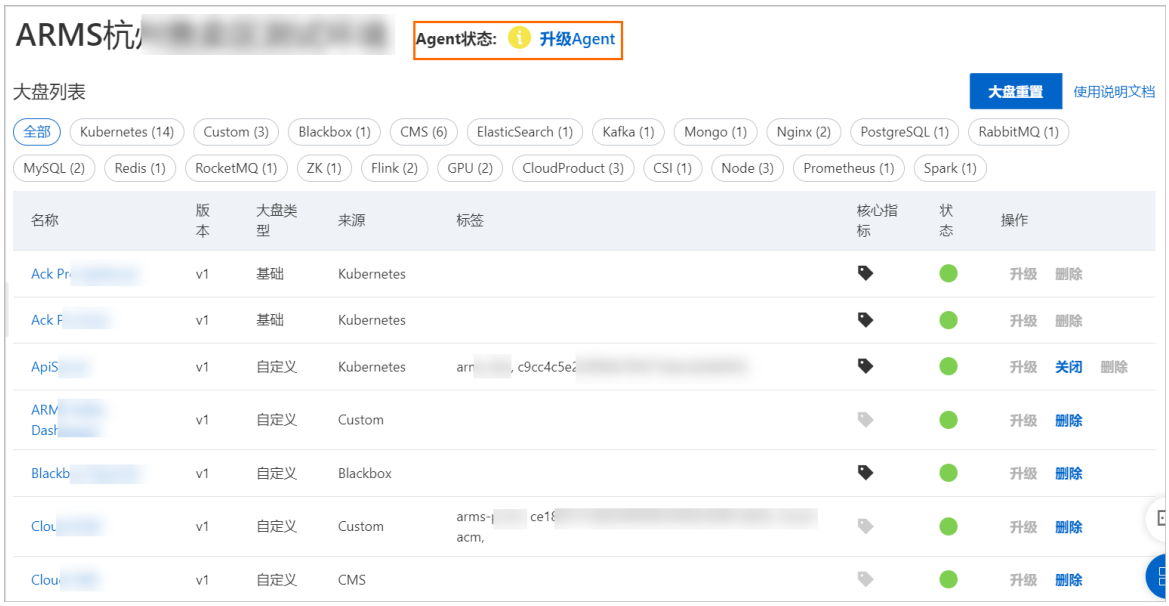
前提条件

您的服务已接入Prometheus监控。具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC
- Prometheus实例 for 云服务

查看大盘列表

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，单击需要查看大盘列表信息的Prometheus实例名称。
   在**大盘列表**页面会列出该Prometheus实例的所有大盘的名称、版本、大盘类型、来源、标签、核心指标和状态。
   以容器服务类型的Prometheus实例为例，在大盘列表页面，查看大盘信息以及Agent当前的状态和版本信息，同时可以单击**升级Agent**及时升级Agent版本。



参数	说明
名称	该Prometheus实例的大盘名称。
版本	大盘当前的版本，Prometheus监控会对基础大盘进行不定期升级。

参数	说明
大盘类型	大盘的类型，包括： ○ 基础：Prometheus监控预置的Grafana大盘。 ○ 自定义：自定义的Grafana大盘。 **说明** 若预置的Grafana大盘无法满足您的需求，您可以自定义Grafana大盘。具体操作，请参见通过阿里云Prometheus自定义Grafana大盘。
来源	Grafana大盘中监控数据的来源。 **说明** 不同类型的Prometheus实例，其Grafana大盘的监控数据来源有所不同。
标签	监控对象内设置的标签。
核心指标	大盘核心指标的信息。
状态	大盘启用状态。

以云服务类型的Prometheus实例为例，其大盘列表如下图所示。

> 说明
> - 来源为CMS的大盘数据来源于通过云监控接入Prometheus监控的云服务，这类大盘数据会随着云服务接入变更而更新。
> - 来源为CloudProduct的大盘数据来源为直接接入Prometheus监控的云服务。

相关操作

在**大盘列表**页面，您可以执行以下操作：

- 查看监控指标
  单击大盘名称，可以在Grafana中查看大盘监控的指标的实时信息。有关基础大盘具体监控指标说明，请参见基础大盘说明。
- 查看核心指标
  将鼠标悬浮于目标大盘核心指标列的图标上，可以查看核心指标信息。单击**查看详情**，可以在**核心指标详情**对话框查看核心指标的类型和描述信息。
- 关闭大盘
  单击大盘名称右侧**操作**列的**关闭**，在弹出的对话框中单击**关闭**，可以关闭该大盘。
- 删除大盘
  单击大盘名称右侧**操作**列的**删除**，在弹出的对话框中单击**删除**，可以删除该大盘。
- 升级大盘版本
  - 升级大盘：在大盘名称上有**New**图标的大盘右侧**操作**列，单击**升级**，在弹出的对话框中单击**升级**，可以升级该大盘。
  - 重置基础大盘：单击**大盘列表**页面右上角的**大盘重置**，在弹出的对话框中单击**重置**，可以将所有基础大盘更新为最新版本。具体操作，请参见重置大盘。

> 说明
> - **升级**和**大盘重置**操作都可以对大盘版本进行升级。**升级**操作是对大盘版本按照版本顺序依次升级，而**大盘重置**操作可以跨版本直接将基础大盘升级至最新版本。若没有及时升级或者重置大盘为最新版本，可能导致当前大盘展示的数据不准确。
> - 云服务类型的Prometheus实例的大盘不支持**大盘重置**操作。

9.2. 重置大盘

创建Prometheus实例之后，系统会默认自动创建相关的基础大盘，您可以通过大盘列表页面的大盘重置功能，重置默认创建的基础大盘。

前提条件

您的服务已接入Prometheus监控。具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC

适用场景

- 一键升级全量默认大盘。

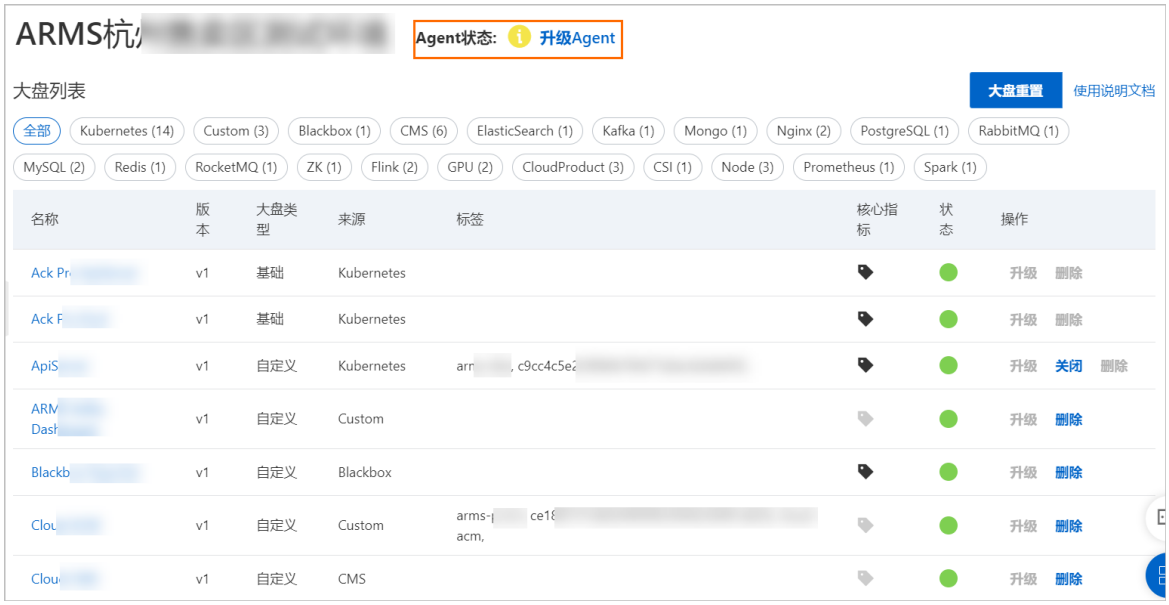
- 已修改的基础大盘想要恢复为默认大盘，通过大盘重置一键恢复。

操作步骤

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，单击需要重置大盘的Prometheus实例名称。

   > **说明** 云服务类型的Prometheus实例的大盘不支持**大盘重置**操作。

3. 单击**大盘列表**页面右上角的**大盘重置**，在弹出的对话框中单击**重置**，可以将所有基础大盘更新为最新版本。



> **说明**
>
> - 重置大盘操作会将Prometheus监控提供的默认基础大盘重置到最新版本，不会影响您的自定义大盘。如果您没有修改过默认基础大盘，则对您的业务不会产生任何影响。
> - 若因环境问题，导致重置大盘后同一基础大盘存在多个版本，单击**操作**列的**升级**即可恢复。更多大盘操作，请参见大盘列表。

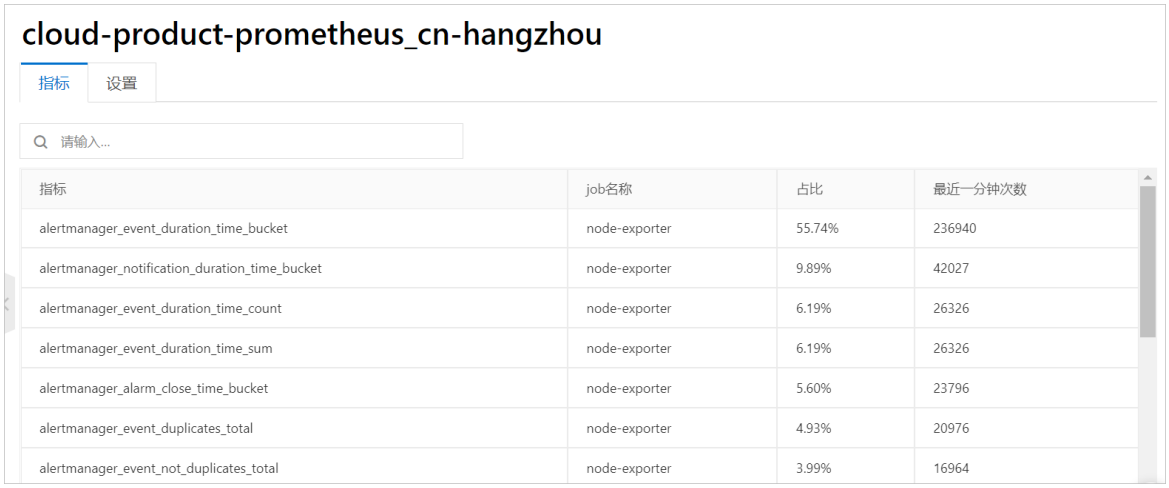
10.管理Prometheus实例

10.1. 配置指标

本文以如何查看云服务类型和容器类型的Prometheus实例的指标为例，介绍如何查看Prometheus实例指标以及如何配置废弃指标。

查看云服务类型的Prometheus实例指标

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择目标地域，然后单击**实例类型**为**Prometheus 实例 for 云服务**对应的Prometheus实例名称。
3. 在左侧导航栏单击**设置**。
4. 在右侧页面，单击**指标**页签。
   指标页签下显示指标的名称、占比以及最近一分钟次数。

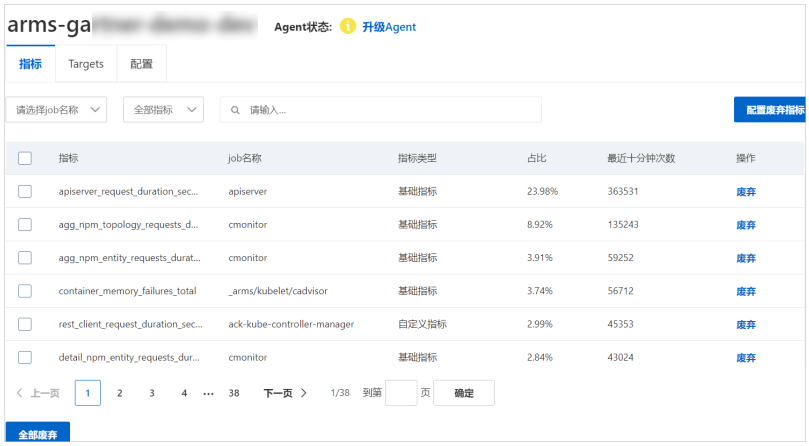


   - job名称：该指标的名称。
   - 占比：该指标数量在所有指标中的占比。
   - 最近一分钟次数：最近一分钟指标落库存储的数量。

查看容器服务类型的Prometheus实例指标

查看Prometheus其他类型实例的指标与查看云服务类型的Prometheus实例指标操作不同，此处以如何查看容器服务类型的Prometheus实例的指标为例，具体操作如下：

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择目标地域，然后单击**实例类型**为**Prometheus 实例 for 容器服务**对应的Prometheus实例名称。
3. 在左侧导航栏单击**服务发现**。
4. 在右侧页面，单击**指标**页签。
   指标页签下显示指标的名称、Job名称、指标类型、占比以及最近十分钟次数。
   您可以通过Job名称或指标类型筛选指标，也可以通过输入关键字进行模糊搜索。



参数	说明
指标	指标的名称。
Job名称	指标所在的任务名称。
指标类型	○ 基础指标 ○ 自定义指标
占比	该指标数量在所有指标中的占比。
最近十分钟次数	最近十分钟指标落库存储的数量。

配置废弃指标

当您的Prometheus实例不再需要监控某些自定义指标的数据时，为了避免这些自定义指标继续产生费用，您可以废弃这些自定义指标。已经配置为废弃的自定义指标将不再继续产生费用。

> **说明** 云服务类型的Prometheus实例不支持配置废弃指标操作。

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后单击Prometheus实例名称。
3. 在左侧导航栏单击**服务发现**。
4. 在右侧页面，单击**指标**页签，然后配置废弃指标：
   - 配置单条废弃指标
     a. 在需要废弃的指标的右侧**操作**列，单击**废弃**。
     b. 在弹出的**配置废弃指标**对话框中，单击**确定**。
   - 配置多条废弃指标
     a. 单击页面右上角的**配置废弃指标**。
     b. 在弹出的**配置废弃指标**对话框中，输入要废弃的指标的名称，然后单击**确定**。
   - 废弃全部指标
     a. 单击页面左下角的**全部废弃**。
     b. 在弹出的确认对话框中，单击**确定**。

> **说明** 对于自建的Kubernetes集群，配置废弃指标的具体操作，请参见配置废弃指标。

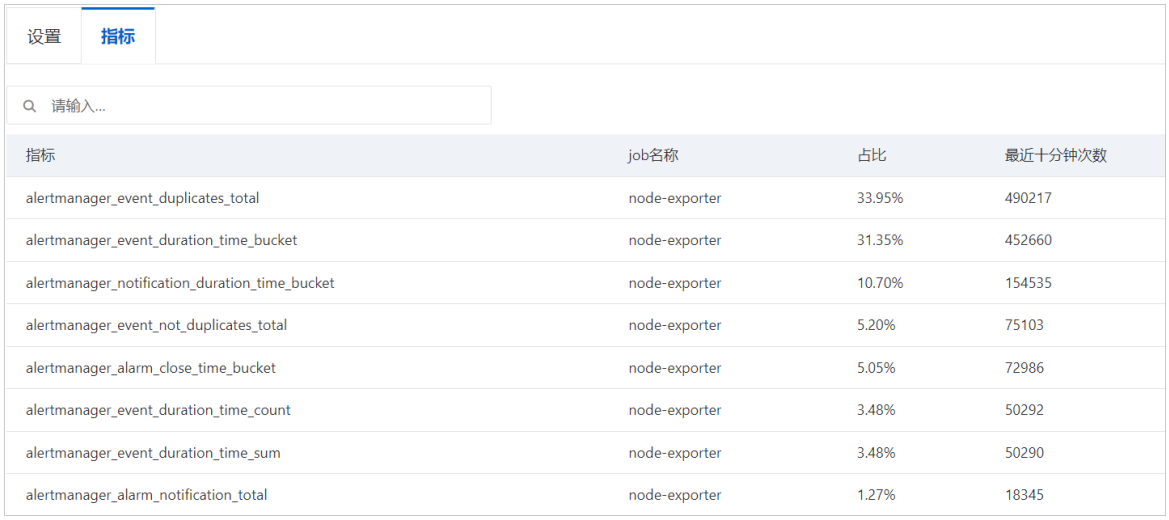
10.2. 云服务指标

本文介绍如何查看云服务指标。

操作步骤

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。

   Prometheus监控页面显示了所有接入Prometheus监控的集群，其中监控对象类型为云服务的实例为Prometheus云服务实例。
3. 单击Prometheus云服务实例名称。
4. 在左侧导航栏单击**设置**。
5. 在右侧页面，单击**指标**页签。
   指标页签下显示指标的名称、占比以及最近十分钟次数。



   - job名称：该指标的名称。
   - 占比：该指标数量在所有指标中的占比。
   - 最近十分钟次数：最近十分钟指标落库存储的数量。

10.3. 云服务设置

在云服务的**设置**页签，您可以获取当前Prometheus实例下的Remote Read地址、Remote Write地址和HTTP API地址。

前提条件

您的服务已接入Prometheus监控。具体操作，请参见Prometheus实例 for 云服务。

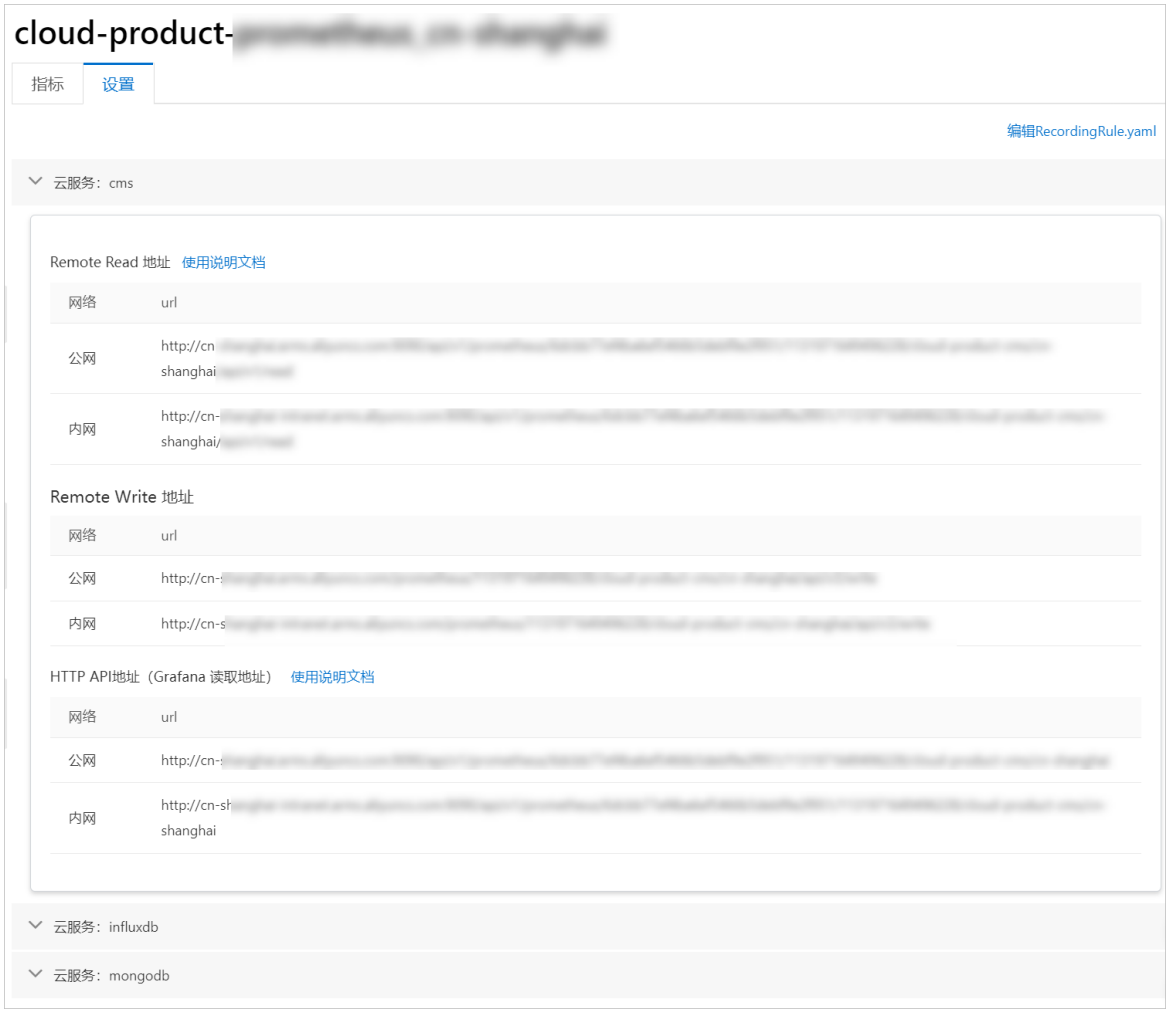
功能入口

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。

   Prometheus监控页面显示了所有接入Prometheus监控的集群，其中监控对象类型为云服务的实例为

Prometheus云服务实例。

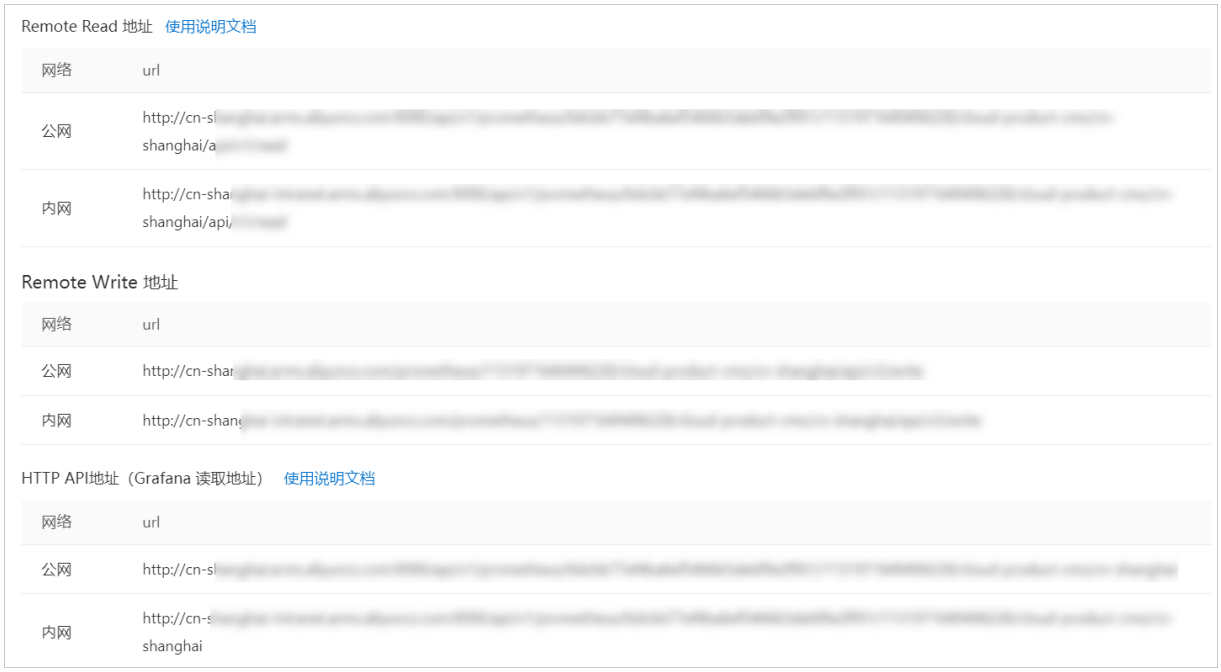
3. 单击Prometheus云服务实例名称。
4. 在左侧导航栏单击**设置**，在右侧页面单击**设置**页签。
5. 在**设置**页签，单击目标接入类型的云服务。



查看地址

在**设置**页签您可以查看云服务下的Remote Read地址、Remote Write地址和HTTP API地址。

> **说明** 根据云服务中接入的产品类型不同，对应的Remote Read地址、Remote Write地址和HTTP API地址也不相同。



- 您可以使用Remote Read地址和Remote Write地址，将自建Prometheus的监控数据存储到阿里云Prometheus实例中。具体操作，请参见Prometheus实例 for Remote Write。
- 您可以使用HTTP API地址，将阿里云Prometheus监控数据接入本地Grafana。具体操作，请参见将阿里云Prometheus监控数据接入本地Grafana。

10.4. 配置Prometheus监控采集规则

您可以通过编辑*prometheus.yaml*或添加ServiceMonitor的方式为应用配置Prometheus监控的采集规则。

前提条件

已成功为应用安装Prometheus监控插件，详情请参见Prometheus实例 for 容器服务。

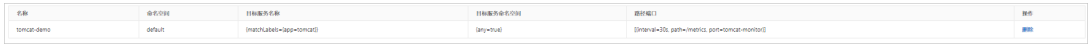
操作步骤

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择容器服务K8s集群所在的地域，并在目标集群右侧的**操作**列单击**设置**。
3. 为应用配置Prometheus监控采集规则分为以下两种情况。
   - 需要监控部署在K8s集群内的应用的业务数据，例如订单信息。操作步骤如下：
     a. 在左侧导航树单击**服务发现**，然后单击**配置**页签。
     b. 在**配置**页面单击ServiceMonitor页签，然后单击**添加ServiceMonitor**。

c. 在**添加ServiceMonitor**对话框中输入以下内容，然后单击**确定**。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  #  填写一个唯一名称
  name: tomcat-demo
  #  填写目标命名空间
  namespace: default
spec:
  endpoints:
  - interval: 30s
    #  填写service.yaml中Prometheus Exporter对应的Port的Name字段的值
    port: tomcat-monitor
    #  填写Prometheus Exporter对应的Path的值
    path: /metrics
  namespaceSelector:
    any: true
    #  Demo的命名空间
  selector:
    matchLabels:
      #  填写service.yaml的Label字段的值以定位目标service.yaml
      app: tomcat
```

ServiceMonitor页签下显示配置的服务发现。



- 需要监控部署在K8s集群之外的业务数据，如Redis连接数。操作步骤如下：

a. 在设置页面，单击**Prometheus设置**页签。

b. 在*Prometheus.yaml*中输入以下内容，然后单击**保存**。

```
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default
is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is ever
y 1 minute.
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
```

10.5. 服务发现

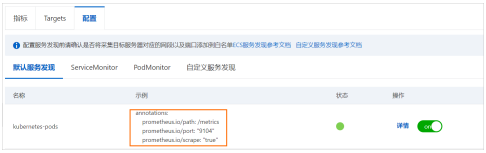
10.5.1. 管理Kubernetes集群服务发现

您可以在服务发现页签中查看ARMS Prometheus监控内置的服务发现，也可以通过添加ServiceMonitor配置Prometheus监控的采集规则。添加ServiceMonitor适用场景包括监控Kubernetes集群内的应用的业务数据，例如订单信息。

管理默认服务发现

默认服务发现为ARMS Prometheus监控内置的服务发现功能，在接入ARMS Prometheus监控时自动开启。当前默认服务发现指标采集对象为Kubernetes集群下所有Namespace包含的Pod。当Pod包含以下注解时，默认服务发现会自动采集该Pod的指标信息并计费：

- prometheus.io/path: /metrics
- prometheus.io/port: "9104"
- prometheus.io/scrape: "true"



1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择目标地域，然后单击**实例类型**为**Prometheus 实例 for 容器服务**对应的Prometheus实例名称。
3. 在左侧导航栏，单击**服务发现**。
4. 在**服务发现**页面单击**配置**页签，然后单击**默认服务发现**页签。

在**默认服务发现**页签下，您还可以执行以下操作：

- 单击**操作**列的**详情**，可以查看默认服务发现的YAML配置详情。
- 打开或关闭**操作**列的开关即可打开或关闭默认服务发现。

查看默认服务发现

更多操作

管理ServiceMonitor

您可以选择手动添加ServiceMonitor，并根据相关配置进行指标采集。ServiceMonitor的指标采集会产生计费。

1. 在**配置**页签，单击**ServiceMonitor**页签，然后单击**添加ServiceMonitor**。
2. 在**添加ServiceMonitor**对话框，输入配置，然后单击**确定**。

   示例配置：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  #  填写一个唯一名称。
  name: tomcat-demo
  #  填写目标命名空间。
  namespace: default
spec:
  endpoints:
  - interval: 30s
    #  填写service.yaml中Prometheus Exporter对应的Port的Name字段的值。
    port: tomcat-monitor
    #  填写Prometheus Exporter对应的Path的值。
    path: /metrics
  namespaceSelector:
    any: true
    #  Nginx Demo的命名空间。
  selector:
    matchLabels:
      #  填写service.yaml的Label字段的值以定位目标service.yaml。
      app: tomcat
```

配置完成后，ServiceMonitor页签下显示配置的服务发现。

在ServiceMonitor页签下，您可以执行以下操作：

- 打开或关闭**操作**列的开关即可打开或关闭自定义的ServiceMonitor资源。您也可以单击**全部关闭**一键关闭所有添加的ServiceMonitor资源。
- 单击**操作**列的**删除**，可以删除自定义的ServiceMonitor资源。

添加ServiceMonitor

更多操作

10.5.2. 管理VPC服务发现

您可以在服务发现页签中查看ARMS Prometheus监控内置的服务发现，也可以通过添加自定义服务配置Prometheus监控的采集规则。

管理默认服务发现

默认服务发现为ARMS Prometheus监控内置的服务发现功能，在接入ARMS Prometheus监控时自动开启并开始计费。当前支持以下2个默认服务发现：

- exporter-service-discovery：按配置规则在**组件监控**页面添加相关组件。
- vpc-ecs-service-discovery：按配置规则自动发现ECS提供的采集服务。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择目标地域，然后单击**实例类型**为**Prometheus 实例 for VPC**对应的Prometheus实例名称。
3. 在左侧导航栏，单击**服务发现**。
4. 在**服务发现**页面单击**配置**页签，然后单击**默认服务发现**页签。

在**默认服务发现**页签下，您还可以执行以下操作：

- 单击**操作**列的**详情**，可以查看默认服务发现的YAML配置文件。
- 打开或关闭**操作**列的开关即可打开或关闭默认服务发现。
- 对于vpc-ecs-service-discovery服务发现，单击**操作**列的**详情**，可以修改YAML配置文件的 `metrics_path` 和 `port` 参数，修改完成后需要单击**确认**。

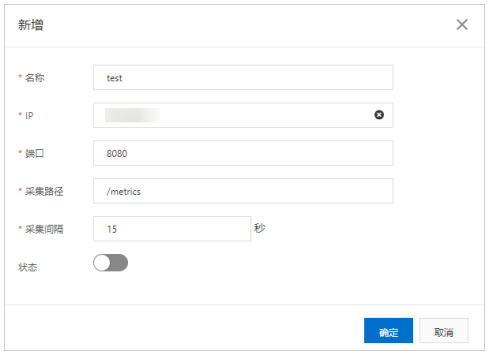
查看默认服务发现

更多操作

管理自定义服务

您可以选择手动添加自定义服务，并根据相关配置进行指标采集。自定义服务的指标采集会产生计费。

1. 在**配置**页签，单击**自定义服务**页签，然后单击**新增**。
2. 在**新增**对话框，输入采集的指标参数，然后单击**确定**。



   - 名称：自定义指标采集服务的名称。
   - IP：指标的IP。
   - 端口：指标的端口。
   - 采集路径：采集的指标路径。
   - 采集间隔：指标采集的时间间隔。
   - 状态：开启或关闭当前指标采集服务。

   配置完成后，**自定义服务**页签下显示配置的服务发现。

在**自定义服务**页签下，您可以执行以下操作：

- 单击**操作**列的**编辑**，可以修改自定义的服务发现。
- 单击**操作**列的**删除**，可以删除自定义的服务发现。

添加自定义服务

更多操作

10.6. 探针设置

在**设置**页签，您可以查看Prometheus探针的基本信息和健康检查结果、设置Agent副本数、重启探针。

前提条件

您的服务已接入Prometheus监控。具体操作，请参见Prometheus实例 for 容器服务。

功能入口

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，单击目标K8s实例名称。
4. 在左侧导航栏单击**设置**，在右侧页面单击**设置**页签。

查看Prometheus探针

在**设置**页签的**Prometheus探针**区域，您可以查看Prometheus探针的状态、版本，Helm版本和Agent副本数。

在**设置**页签的**Prometheus探针**区域，您可以执行以下操作：

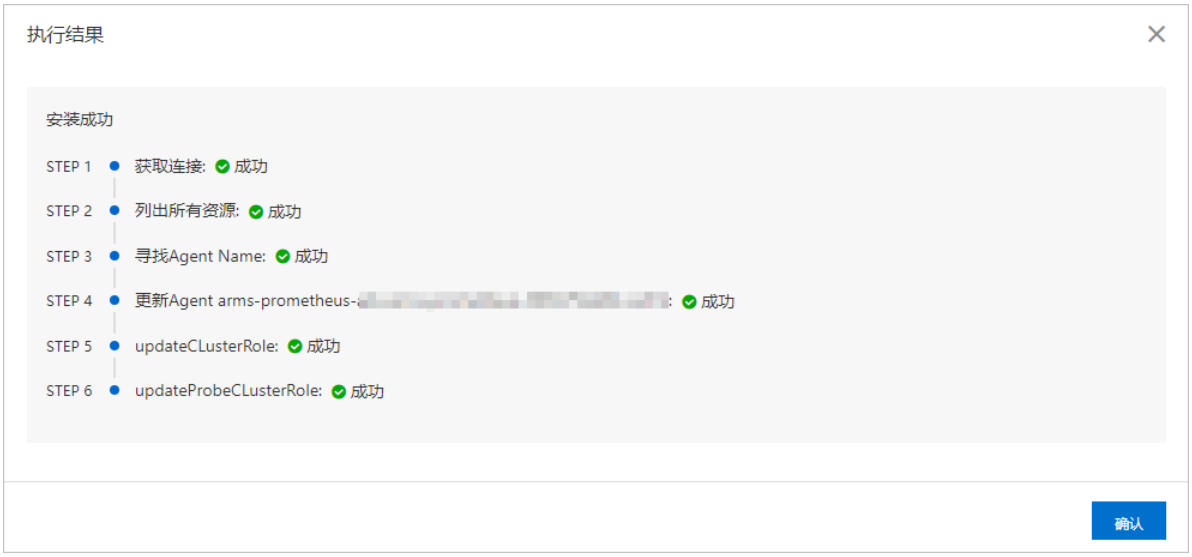
- 单击**Prometheus探针**区域右上角的**编辑Prometheus.yaml**，可以更新Prometheus的*Prometheus.yaml*文件。
- 单击**Prometheus探针**区域右上角的**编辑RecordingRule.yaml**，可以更新Prometheus的*RecordingRule.yaml*文件。具体操作，请参见编辑RecordingRule.yaml。
  阿里云Prometheus监控的*RecordingRule.yaml*只需配置Rule Group。 暂不支持Recording Rule的Remote Write。配置示例如下：

```
groups:
  - name: cpu-node
    interval: 30s
    rules:
      - expr: avg by (job, instance, mode) (rate(apiserver_request_total[5m]))
        record: job_instance_mode:apiserver_request_total:avg_rate5m
```

- 单击**Helm版本**区域的**升级**，在弹出的对话框中单击**确定**，可以升级Helm版本。Helm版本的详细信息，

请参见Helm版本说明。

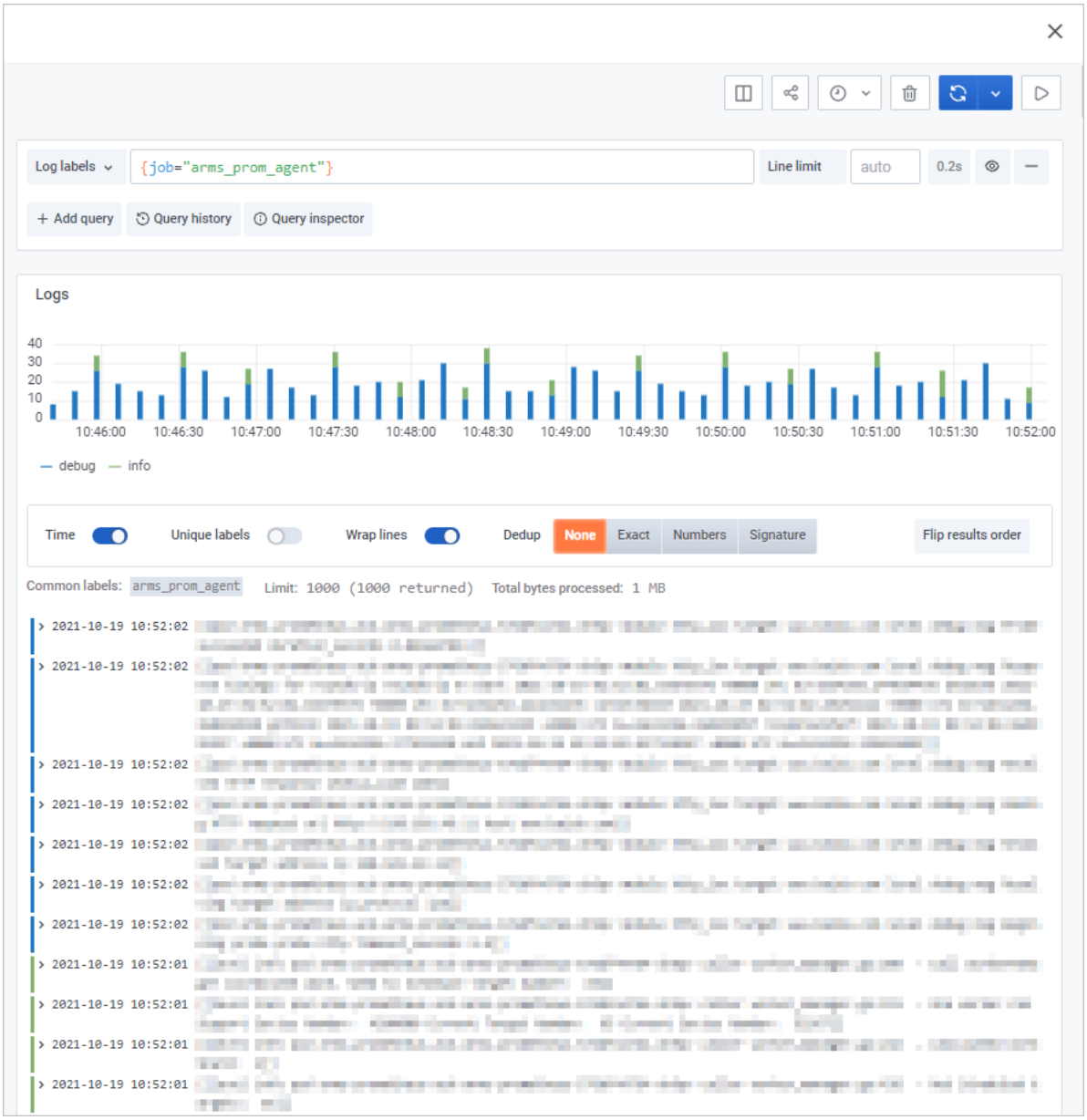
- 单击**操作**列的**重启**，可以将当前探针重启。
  重启完成后会弹出**执行结果**对话框。



- 单击**操作**列的**副本数**，在弹出的**设置Agent副本数**对话框设置Agent副本数，然后单击**确定**。

- 单击**操作**列的**健康检查**，在弹出的**健康检查结果**对话框查看探针的健康检查结果。

- 单击**操作**列的**探针日志**，在弹出的面板中查看探针的日志。



10.7. 获取Remote Read、Remote Write和HTTP API地址

在**设置**页签，您可以获取当前Prometheus实例下的Remote Read地址、Remote Write地址和HTTP API地址。本文以如何获取云服务类型和容器类型的Prometheus实例的地址为例，介绍如何获取这些地址。

前提条件

您的服务已接入Prometheus监控。具体操作，请参见

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC
- Prometheus实例 for 云服务

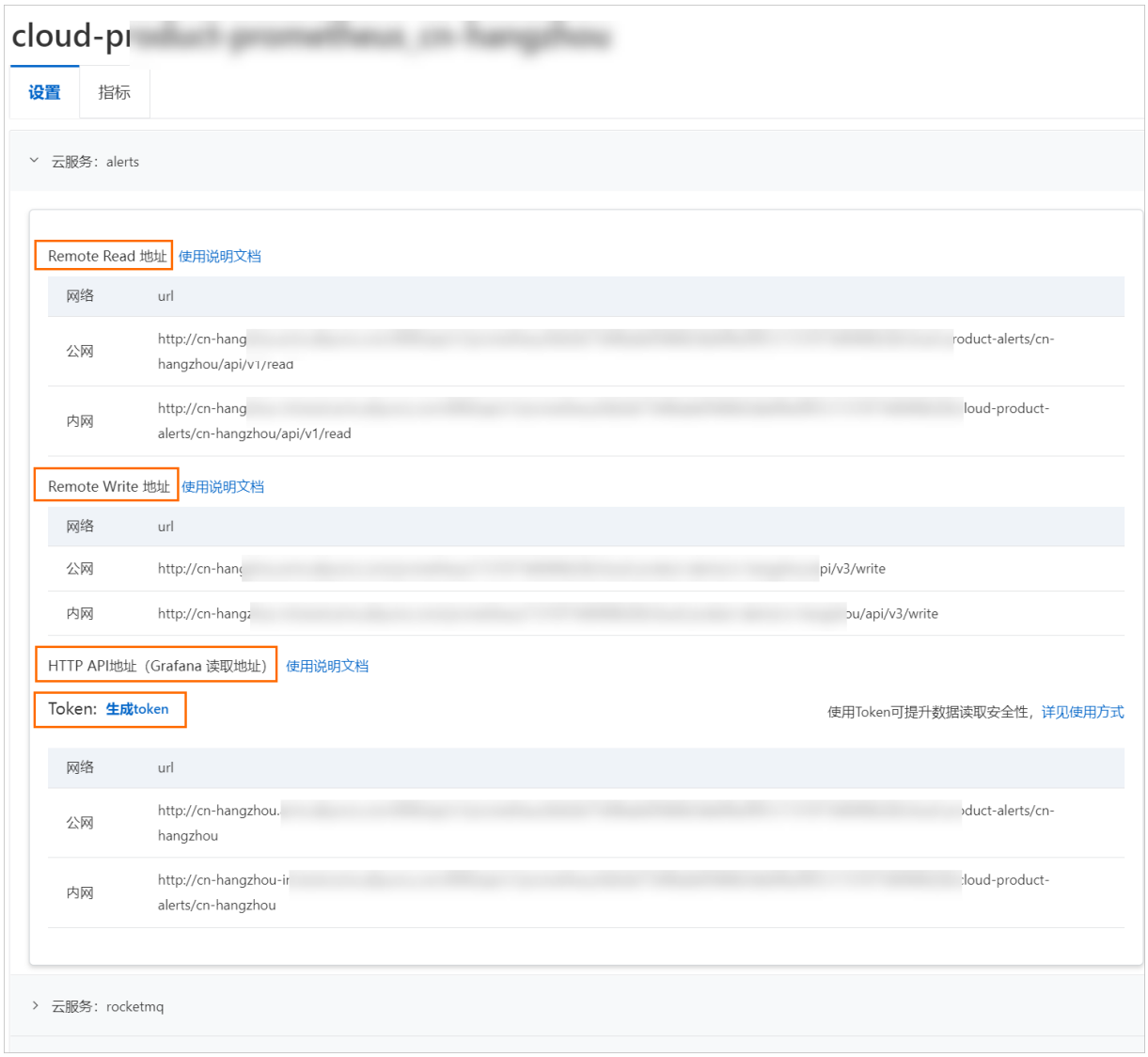
。

功能入口

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，单击目标Prometheus实例名称。
4. 在左侧导航栏单击**设置**，在右侧页面单击**设置**页签。

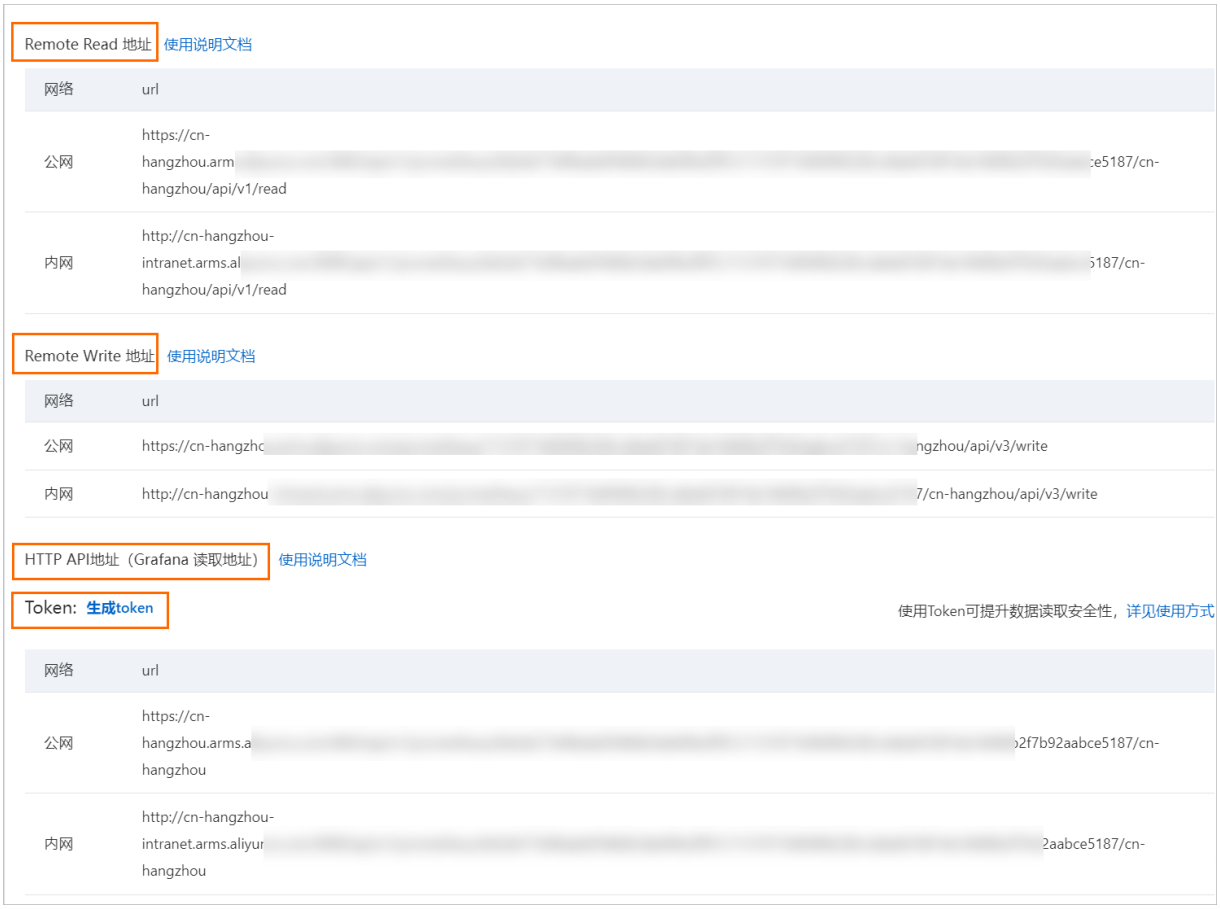
查看云服务类型Prometheus实例的地址

在**设置**页签，单击目标云服务右侧的 › 图标，您可以查看该云服务下的Remote Read地址、Remote Write地址和HTTP API地址。同时在**Token**区域，您还可以获取Prometheus实例的鉴权Token。



查看容器服务类型Prometheus实例的地址

在**设置**页签，您可以查看该云服务下的Remote Read地址、Remote Write地址和HTTP API地址。同时在**Token**区域，您还可以获取Prometheus实例的鉴权Token。获取其他类型Prometheus实例下地址的方式与容器类型的一致。



后续操作

- 阿里云Prometheus接入自建Grafana，使用Prometheus Metrics实现HPA等场景需要远程读取ARMS Prometheus存储数据；自建Prometheus、纳管集群（注册集群）等场景需要远程写入ARMS Prometheus存储。在远程读写Prometheus数据时，使用鉴权Token可以提高数据读写的安全性。阿里云Prometheus接入自建Grafana时使用Token的操作，请参见将阿里云Prometheus监控数据接入本地Grafana。注册集群接入Prometheus监控的操作，请参见公网Kubernetes集群接入Prometheus监控。
- 您可以使用Remote Read地址和Remote Write地址，将自建Prometheus的监控数据存储到阿里云Prometheus中。具体操作，请参见Prometheus实例 for Remote Write。
- 您可以使用HTTP API地址，将阿里云Prometheus监控数据接入本地Grafana。具体操作，请参见将阿里云Prometheus监控数据接入本地Grafana。

10.8. 升级组件版本

本文介绍阿里云容器服务Kubernetes集群类型的Prometheus实例如何升级ack-arms-prometheus组件版本。

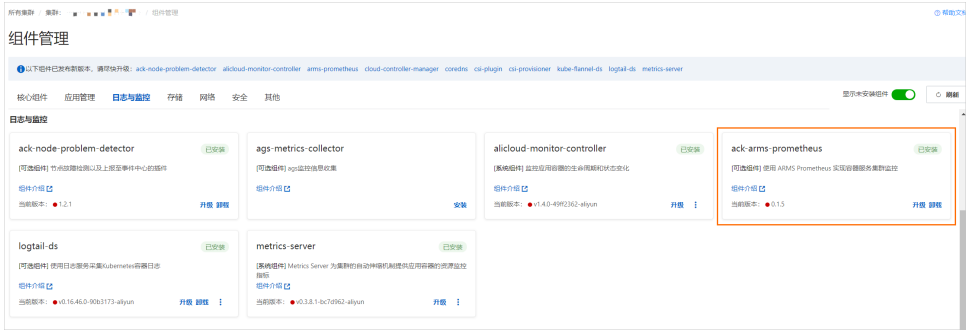
若您的ARMS Prometheus Helm版本为v0.1.4及以下，或者在升级过程中，出现资源冲突报错，您需要先卸载旧版本的Helm，然后在容器服务控制台的**运维管理 > 组件管理**页面重新安装ack-arms-prometheus组件。若您在升级过程中遇到任何问题，欢迎搜索钉钉账号（aliprometheus）联系技术支持人员获取帮助 。

- 卸载清理文档请参见常见问题。
- 安装ack-arms-prometheus组件的操作，请参见通过ACK控制台Prometheus监控开启Prometheus监控。

操作步骤

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面，选择目标集群，并在目标集群右侧**操作**列下，选择**更多 > 组件管理**。
4. 在**组件管理**页面的**日志与监控**区域，选择ack-arms-prometheus组件，然后单击**升级**。

> **说明** 如果ack-arms-prometheus组件显示未安装或升级失败，请先卸载并重新安装ack-arms-prometheus组件。
>
> - 卸载清理文档请参见常见问题。
> - 安装ack-arms-prometheus组件的操作，请参见通过ACK控制台Prometheus监控开启Prometheus监控。



相关文档

- Helm版本说明
-

10.9. 编辑RecordingRule.yaml

本文介绍如何配置预聚合（Recording Rule）以及如何查看Recording Rule指标。

背景信息

预聚合（Recording Rule）可以对落地的指标数据做二次开发。某些查询可能需要在查询端进行大量的计算，导致查询端压力过大，您可以配置预聚合规则将计算过程提前到写入端，减少查询端资源占用，尤其在大规模集群和复杂业务场景下可以有效的降低PromQL的复杂度，从而提高查询性能，解决用户配置以及查询慢的问题。

> **说明** Recording Rule的配置与开源Prometheus相同，以规则组（Rule Group）的形式存在，每个规则组可以有多条规则（Rules），聚合规则的名称必须符合 Prometheus指标名称规范。相同组中的规则以一定的间隔顺序执行，预聚合后的指标按照新的规则名字存入远端数据库。

配置Recording Rule

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后单击需要配置Recording Rule的Prometheus实例名称。
3. 在左侧导航栏的**设置**页面单击**设置**页签，然后单击**编辑RecordingRule.yaml**。

4. 在**编辑RecordingRule.yaml**对话框，删除默认模板并输入预聚合规则，然后单击**保存**。

> **说明** 同一个集群只需要配置一份RecordingRule.yaml，不同规则组(Rule Group)的名字必须不同。

预聚合规则示例：

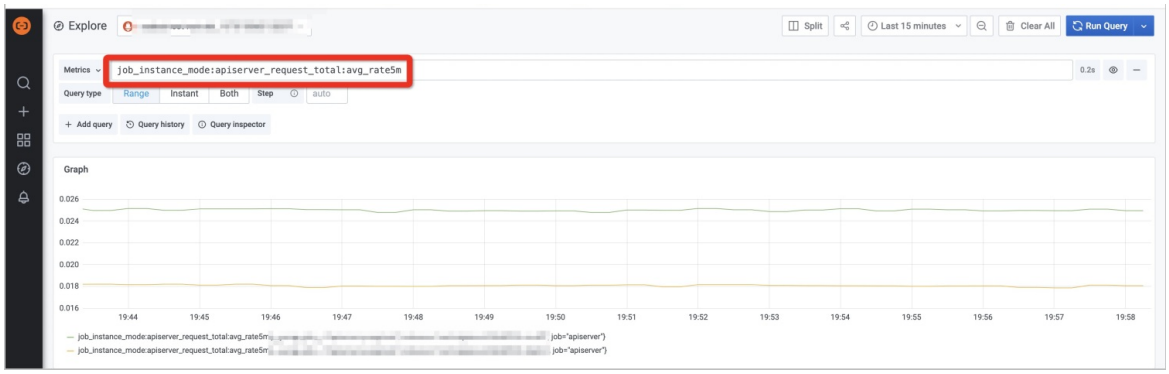
```
groups:
  - name: apiserver_request_total
    interval: 60s
    rules:
      - record: job_instance_mode:apiserver_request_total:avg_rate5m
        expr: avg by (job, instance, mode) (rate(apiserver_request_total[5m]))
        labels:
          team: operations
      - record: job:apiserver_request_total:sum_rate10m
        expr: sum by (job)(rate(apiserver_request_total[10m]))
        labels:
          team: operations
```

参数	说明
groups	规则组。一份RecordingRule.yaml可以配置多组规则组。
name	规则组名称。规则组名称必须唯一。

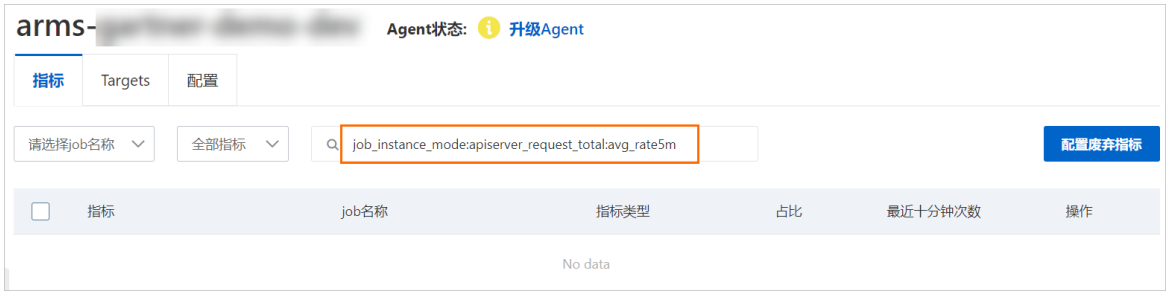
参数	说明
interval	（可选）规则组的执行周期。默认60s。
rules	规则。一个规则组可以包含多条规则。
record	规则的名称。聚合规则的名称必须符合 Prometheus指标名称规范。 **说明** 预聚合规则命名的推荐格式：`level:metric:operations` • `level`：表示聚合级别，以及规则的输出标签。 • `metric`：表示指标的名称。 • `operations`：应用于指标的操作列表，最新的操作在前面。
expr	计算表达式。Prometheus监控将通过该表达式计算得出预聚合指标。计算表达式必须符合PromQL。
lables	（可选）指标的标签。

查看Recording Rule指标

1. 以管理员账号登录本地Grafana系统。
2. 在左侧导航栏中选择**Explore**。
3. 在**Explore**右侧选择对应的Kubernetes集群名称，然后在**Metrics**右侧输入Recording Rule指标名称查看该指标的数据详情。



1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后单击需要查看Recording Rule指标的Prometheus实例名称。
3. 在左侧导航栏的**服务发现**页面，选择**指标**页签，然后您可以在搜索框中输入对应的Recording Rule指标名称来查看该指标的详细信息。



方式一：从Grafana大盘查看

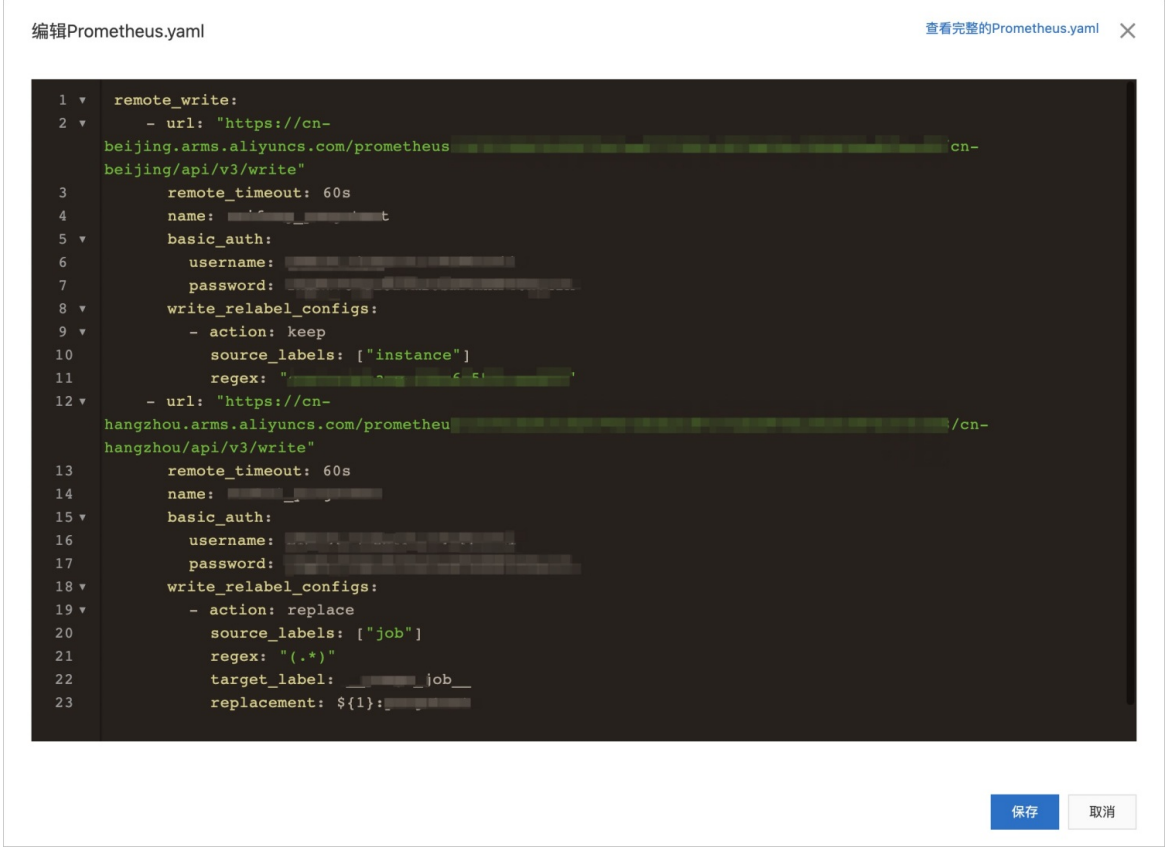
方式二：从Prometheus监控控制台查看

Recording Rule支持Remote Write

阿里云Prometheus监控的Recording Rule同开源Prometheus一样，支持Remote Write功能。

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后单击需要配置Remote Write的Prometheus实例名称。
3. 在左侧导航栏的**设置**页面单击**设置**页签，然后单击**编辑Prometheus.yaml**。

4. 在弹出的**编辑Prometheus.yaml**对话框中配置Remote Write。



> **说明** 配置Remote Write的方法与开源Prometheus配置方法相同。支持多组Remote Write配置，同时支持Write Relabel Config。

Remote Write配置同时对Agent和Recording Rule组件生效。若只需要将Recording Rule生成的指标远程写（Remote Write）到其他库，可配置对应的Write Relabel Config。示例如下：

```
remote_write:
  - url: "https://xxxx/api/v1/prom/write?db=dbname&u=username&p=password"     //远程写数据库地址
    write_relabel_configs:
      - source_labels: [__name__]
        regex: job_instance_mode:apiserver_request_total:avg_rate5m
        action: keep
```

> **说明** 当前配置只会将Recording Rule指标（即job_instance_mode:apiserver_request_total:avg_rate5m）远程写（Remote Write）到其他库。

相关操作

对Recording Rule生成的新指标Remote Write，URL需要使用公网地址。如需对Recording Rule组件添加网络白名单，请根据地域添加对应的白名单。

北京： 101.200.XX.XX
杭州： 118.31.XX.XX
上海： 106.14.XX.XX
深圳： 8.129.XX.XX
张家口： 39.103.XX.XX
青岛： 139.129.XX.XX
成都： 47.108.XX.XX
香港： 47.242.XX.XX
新加坡： 47.241.XX.XX

10.10. 卸载Prometheus监控插件

如需停止对Kubernetes集群的Prometheus监控，请按照以下步骤卸载Prometheus监控插件。

操作步骤

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面单击目标集群右侧**操作**列**卸载**。
4. 在**确认**对话框中单击**确认**。
   卸载插件完毕后，**Prometheus监控**页面中的集群将会消失。接下来需要前往容器服务管理控制台确认卸载是否成功。

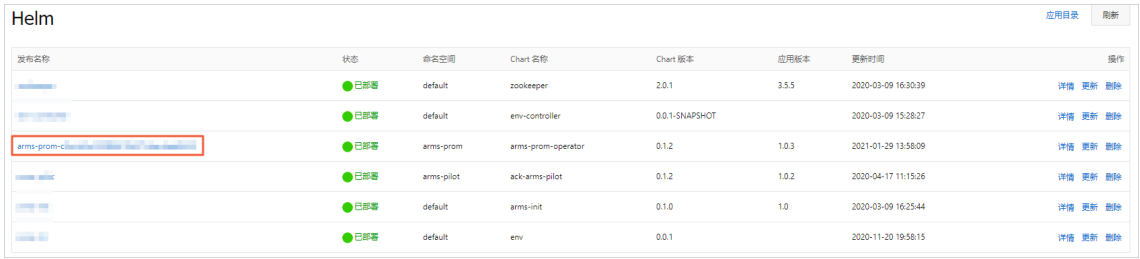
结果验证

注意 部署在容器服务Kubernetes版的Prometheus插件卸载验证步骤如下，部署在开源Kubernetes的验证步骤，请参见Kubernetes官方文档。

1. 登录容器服务管理控制台。
2. 在左侧导航栏单击**集群**，然后单击需停止监控的集群名称。

   说明 本文中关于容器服务管理控制台的步骤描述仅适用于新版控制台。如果您使用的是旧版控制台，请在左侧导航栏选择**集群 > 集群**，然后在页面右上角单击**体验新版**。

3. 在左侧导航栏选择**应用 > Helm**，并根据情况采取以下任一操作：
   - 如果**Helm**页面没有 `arms-prom-****` 记录，则说明监控插件卸载成功，您无需采取任何操作。
   - 如果**Helm**页面有 `arms-prom-****` 记录，请在其右侧的**操作**列中单击**删除**。



11.全局配置

11.1. GlobalView聚合实例（旧版）

阿里云Prometheus监控提供地域级别的GlobalView聚合实例的功能。GlobalView聚合实例功能可以为您提供在当前地域下所有Prometheus实例的一个虚拟聚合实例。针对这个虚拟聚合实例可以实现统一的指标查询和告警。

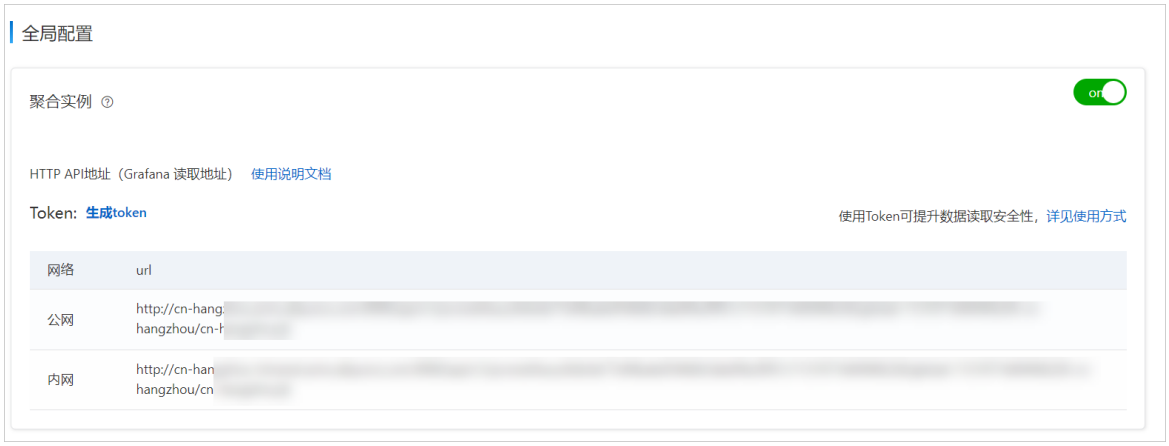
> **说明** 目前旧版聚合实例功能将逐步停止服务，在停止服务之前您仍可以继续使用，这里推荐您使用新版创建聚合实例的方式实现监控和报警，具体内容请参见Prometheus实例 for GlobalView。

前提条件

- 已创建Prometheus实例，具体操作，请参见：
  - Prometheus实例 for 容器服务
  - Prometheus实例 for Kubernetes
  - Prometheus实例 for Remote Write
  - Prometheus实例 for VPC
  - Prometheus实例 for 云服务
- 您已在本地成功安装Grafana软件，请参见Grafana。

查看聚合实例

1. 登录Prometheus控制台。
2. 在左侧导航栏选择**全局配置**。在页面左上角选择目标地域。
3. 查看聚合实例的HTTP API地址。您可以使用HTTP API地址，将阿里云Prometheus监控的GlobalView聚合实例数据接入本地Grafana。实现统一查看指标数据。



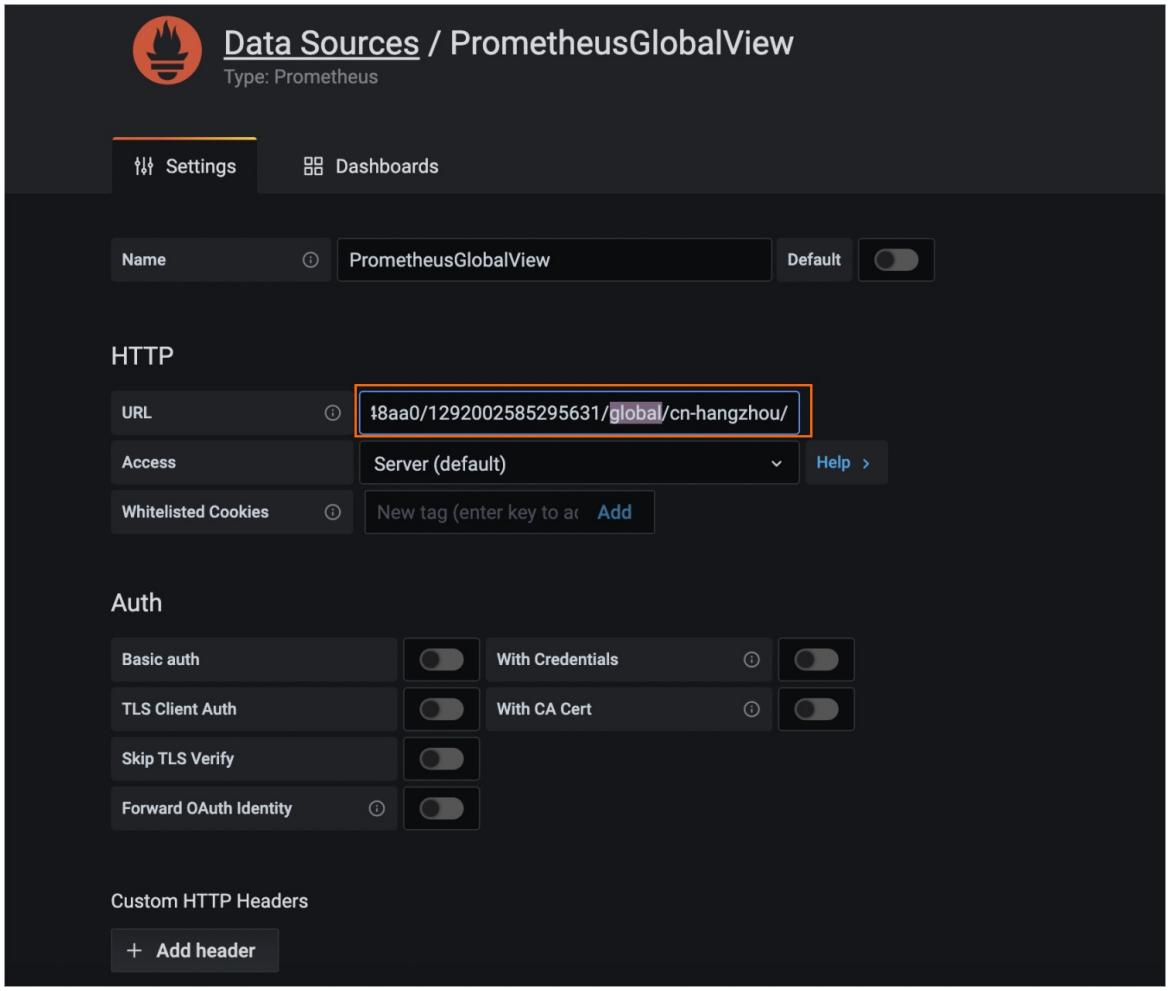
通过Grafana添加Prometheus监控的GlobalView聚合实例

将在查看聚合实例中获得的API接口地址添加为本地Grafana的数据源即可实现目标。请按照以下步骤添加数据源：

1. 以管理员账号登录本地Grafana系统。
2. 在左侧导航栏中选择**Configuration > Data Sources**。

> 说明 仅管理员可以看到此菜单。

3. 在Data Sources页签上单击Add data source。
4. 在Add data source页面上单击PrometheusGlobalView。
5. 在Settings页签的Name字段输入自定义的名称，在URL字段粘贴查看聚合实例中获得的API接口地址。

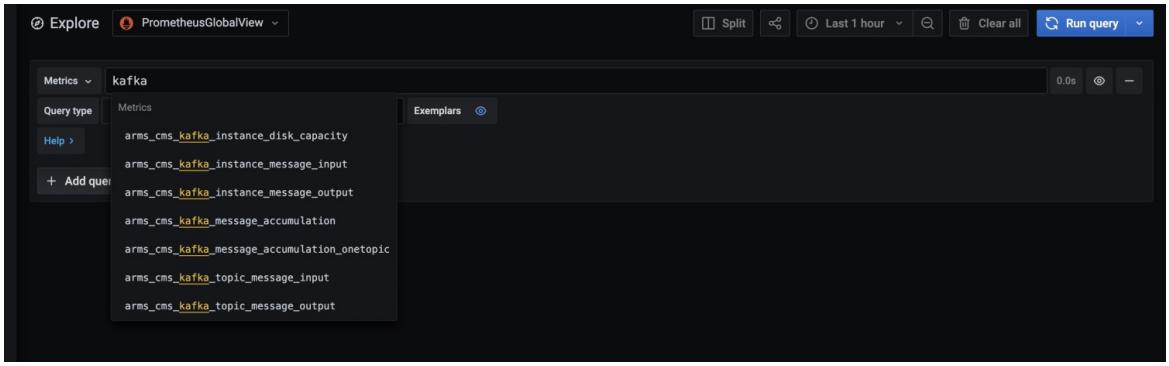


6. 单击页签底部的Save & Test，完成添加Prometheus监控的GlobalView聚合实例。

查看指标数据

通过Grafana添加Prometheus监控的GlobalView聚合实例后，可以通过Explorer查看Prometheus监控、云服务、应用监控等服务的所有指标。

1. 登录本地Grafana系统。
2. 在左侧导航栏中选择Explorer。
3. 在Explorer右侧下拉框中选择PrometheusGlobalView，在Metrics字段输入指标名称并按回车键。
4. 显示相应指标的图表，查看GlobalView聚合实例的所有指标数据。若未显示任何指标数据，请检查填写的接口地址是否正确，或者数据源是否有Prometheus监控数据。


Explore
PrometheusGlobalView
Split
Last 1 hour
Clear all
Run query
Metrics
kafka
0.0s
Query type
Metrics
Exemplars
Help
arms_cms_kafka_instance_disk_capacity
arms_cms_kafka_instance_message_input
arms_cms_kafka_instance_message_output
arms_cms_kafka_message_accumulation
arms_cms_kafka_message_accumulation_onetopic
arms_cms_kafka_topic_message_input
arms_cms_kafka_topic_message_output

12.健康巡检

背景信息

操作步骤

1.

12.1. 创建自定义巡检

健康巡检定期对监控的服务进行连接测试，帮助您掌握服务的健康状况，及时发现异常，从而采取针对性的有效措施。您可以为指定服务创建自定义健康巡检。

前提条件

您的服务已接入Prometheus监控。具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC
- Prometheus实例 for 云服务

操作步骤

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，根据需要单击容器服务或者Kubernetes类型的Prometheus实例名称。
4. 在左侧导航栏，单击**健康巡检**。
5. 在**巡检**页签右上角，单击**新建巡检**，然后在弹出的对话框中设置巡检参数。

> **说明** 您在检查点中需要输入IP或者域名，例如192.168.0.1或者www.aliyun.com。

12.2. 创建ACK Service巡检

健康巡检定期对监控的服务进行连接测试，帮助您掌握服务的健康状况，及时发现异常，从而采取针对性的有效措施。您可以通过批量导入Prometheus监控的ACK Service数据，快速创建ACK Service巡检。

前提条件

您的服务已接入Prometheus监控。具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC
- Prometheus实例 for 云服务

创建ACK Service巡检

1. 登录Prometheus控制台。

2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，根据需要单击容器服务或者Kubernetes类型的Prometheus实例名称。
4. 在左侧导航栏，单击**健康巡检**。
5. 在**巡检**页签右上角，单击**ACK Service巡检**，然后在弹出的对话框中选择要导入的ACK Service数据，并单击**确定**。

> **说明** ACK Service支持巡检的类型：
> - 内部端点类型：TCP、PING。
> - 外部端点类型：TCP、PING、HTTP。

12.3. 创建云服务巡检

健康巡检定期对监控的服务进行连接测试，帮助您掌握服务的健康状况，及时发现异常，从而采取针对性的有效措施。您可以通过批量导入Prometheus监控的云服务数据，快速创建云服务巡检。

前提条件

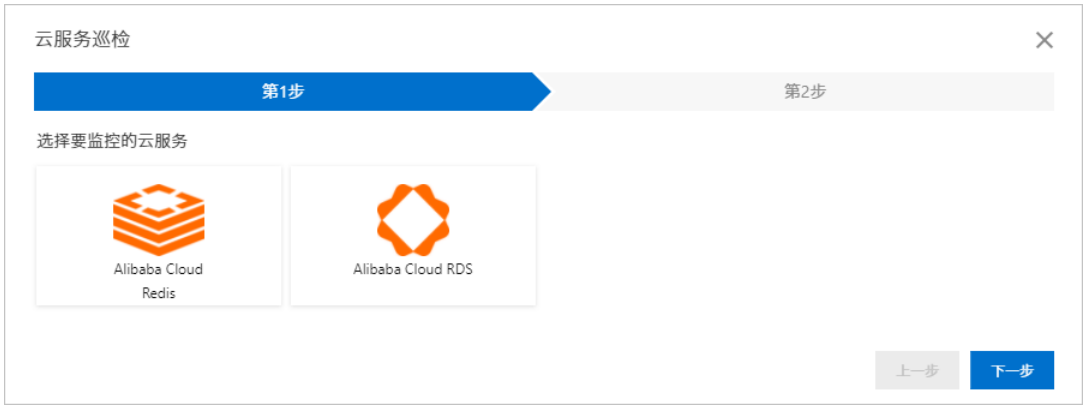
您的服务已接入Prometheus监控。具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC
- Prometheus实例 for 云服务

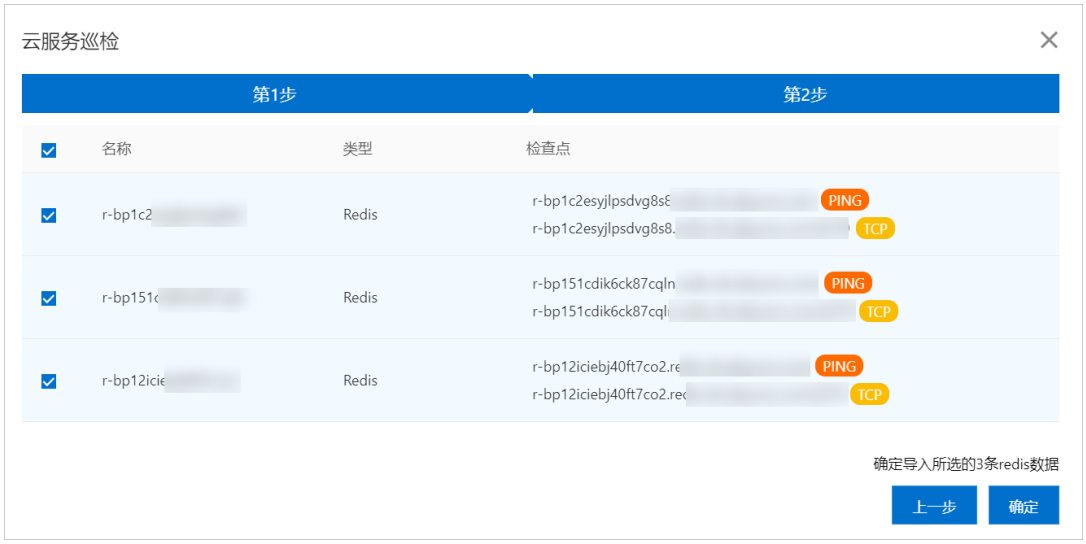
创建云服务巡检

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，根据需要单击容器服务或者Kubernetes类型的Prometheus实例名称。
4. 在左侧导航栏，单击**健康巡检**。
5. 在**巡检**页签右上角，单击**云服务巡检**，然后在弹出的对话框中执行以下操作。
   i. 在**第1步**选择要导入数据的云服务，然后单击**下一步**。

> **说明** Prometheus目前支持导入Alibaba Cloud Redis数据和Alibaba Cloud RDS数据。



ii. 在**第2步**选择要导入的云服务数据，然后单击**确定**。



说明 默认选择导入所有的云服务数据，请您根据实际需要进行调整。

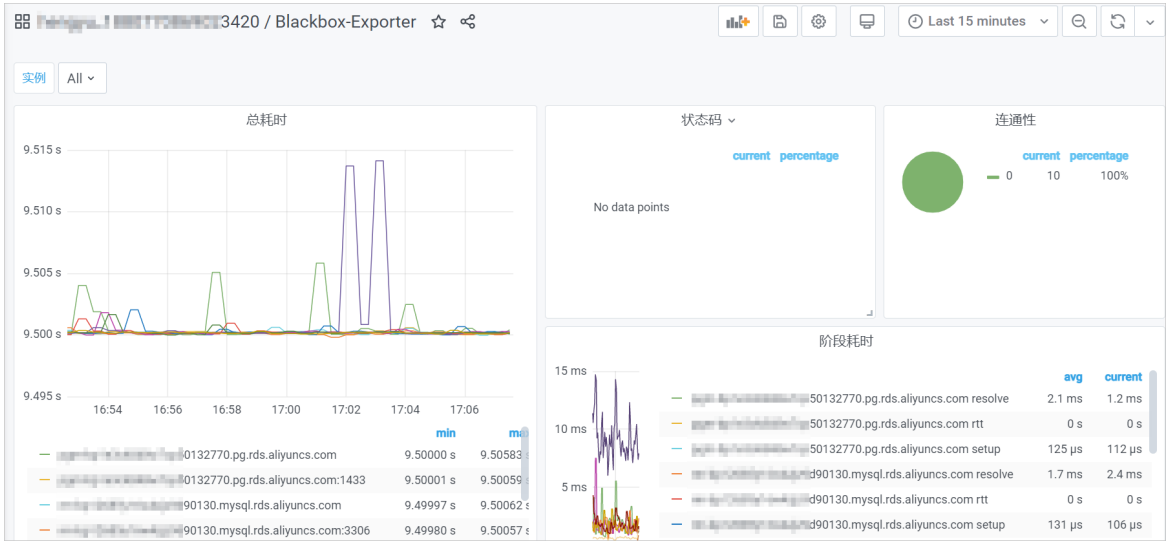
12.4. 管理健康巡检

在**健康巡检**页面，您可以查看已创建巡检的大盘信息，并对巡检执行编辑、删除等操作。

查看健康巡检大盘

创建健康巡检后，系统将为其自动创建Grafana大盘。您可以通过该大盘查看服务的健康指标。

1. 在左侧导航栏，单击**大盘列表**。
2. 在**大盘列表**页面的**名称**列，单击**Blackbox-Exporter**。



说明 您还可以单击巡检名称直接跳转至大盘查看服务的健康指标。

启用健康巡检报警

创建健康巡检后，系统将为其自动创建未启用的报警规则。您需要开启报警规则，以确保巡检异常时能收到报警通知。

1. 在左侧导航栏，单击**报警配置**。
2. 在页面的**报警名称**列，找到**健康巡检**，在其右侧**操作**列，单击**开启**。

编辑巡检

您可以编辑巡检的参数设置。

1. 在**巡检**页签下，找到要编辑的检查点，在其右侧**操作**列，单击**编辑**。
2. 在**编辑巡检**对话框，重新设置巡检参数。

> **说明** 您可以在检查点中重新输入IP或者域名，例如192.168.0.1或者www.aliyun.com。

删除巡检

您可以在巡检页签下删除不需要进行巡检的检查点。当前支持单个删除和批量删除两种操作模式。

13.报警（旧版）

13.1. 创建报警

阿里云Prometheus监控提供开箱即用的报警规则，您也可以自定义针对特定监控对象的报警规则。当规则被触发时，系统会以您指定的报警方式向报警联系人分组发送报警信息，以提醒报警联系人采取必要的问题解决措施。

前提条件

- 已成功创建监控任务，请参见Prometheus实例 for 容器服务。
- 创建联系人，请参见创建联系人。

操作步骤

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
3. 在左侧导航栏，选择**报警配置**。
4. 在报警配置页面右上角，单击**创建报警**。
5. 在**创建报警**面板，执行以下操作：
   i. （可选）从**告警模板**下拉列表，选择模板。
   ii. 在**规则名称**文本框，输入规则名称，例如：网络接收压力报警。
   iii. 在**告警表达式**文本框，输入告警表达式。例如：`(sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m]))) > 0.01`。

   > **注意** PromQL语句中包含的 `$` 符号会导致报错，您需要删除包含$符号的语句中 `=` 左右两边的参数及 `=` 。例如：将 `sum (rate (container_network_receive_bytes_total{instance=~"^$HostIp.*"}[1m]))` 修改为 `sum (rate (container_network_receive_bytes_total[1m]))` 。

   iv. 在**持续时间**文本框，输入持续时间N，当连续N分钟满足告警条件的时候才触发告警。例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。

   > **说明** 持续N分钟满足告警条件是指在连续N分钟内，您上面设置的PromQL语句条件都能满足。Prometheus默认数据采集周期为15s，如果没有连续N×60/15=4N个数据点满足告警条件（PromQL语句），就不会发送告警。如Prometheus告警规则默认持续时间为1分钟，既只有连续4个数据点都满足告警条件（PromQL语句）才会触发告警。如果您想在任何一个数据点满足告警条件（PromQL语句）就发送告警，请修改持续时间为0分钟。

   v. 在**告警消息**文本框，输入告警消息。
   vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。

vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息`，例如：`message:{{$labels.pod_name}}重启`。

您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：

viii. 从**通知策略**下拉列表，选择通知策略。

如何创建通知策略，请参见通知策略。

ix. 单击**确定**。

报警配置页面显示创建的报警。

相关操作

您可以在**通知策略**页面中设置分派规则、通知方式和通知联系人，具体操作，请参见通知策略。

13.2. 管理报警

阿里云Prometheus监控提供开箱即用的报警规则，在报警配置页面上，您可以管理账号下的所有报警规则。

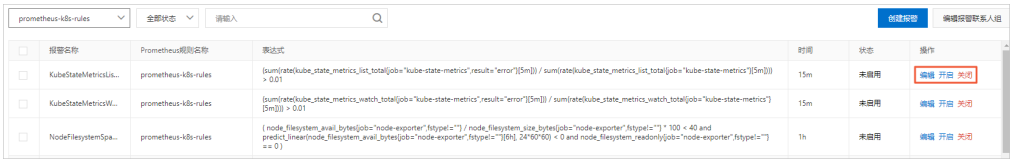
管理报警规则

您可以对报警规则执行开启、关闭、编辑等操作。

1. 登录Prometheus控制台。
2. 在顶部菜单栏选择目标地域，然后单击目标K8s集群名称。
3. 在左侧导航栏中单击**报警配置Beta**。
4. （可选）在**报警配置Beta**页面设置筛选条件，或者在搜索框中输入报警名称并单击**搜索**。

> **说明** 您可以输入报警名称的一部分内容进行模糊搜索。

5. 在搜索结果列表的**操作**列中，按需对目标报警规则采取以下操作：
   - 如需编辑报警规则，请单击**编辑**，在**编辑报警**对话框中编辑报警规则，并单击**确认**。**编辑报警**对话框中详细的参数说明，请参见创建报警。
   - 如需启动未启用的报警规则，请单击**开启**，然后在**状态**列中查看启动状态。
   - 如需停用已启用的报警规则，请单击**关闭**，然后在**状态**列中查看停用状态。



删除报警规则

报警仅支持按规则删除，即同一规则的所有报警会一起删除。

1. 在左侧导航栏中单击**设置**，然后单击**规则**页签。
2. 单击目标规则**操作**列的**删除**。
3. 在弹出的**确认**对话框中单击**删除**。

相关文档

- 创建报警
- 创建联系人

13.3. 云服务报警配置

Prometheus监控为接入的云服务提供预置报警规则，若预置的报警规则不满足您的业务需求，您还可以创建自定义报警。本文介绍如何查看预置报警和创建自定义报警。

前提条件

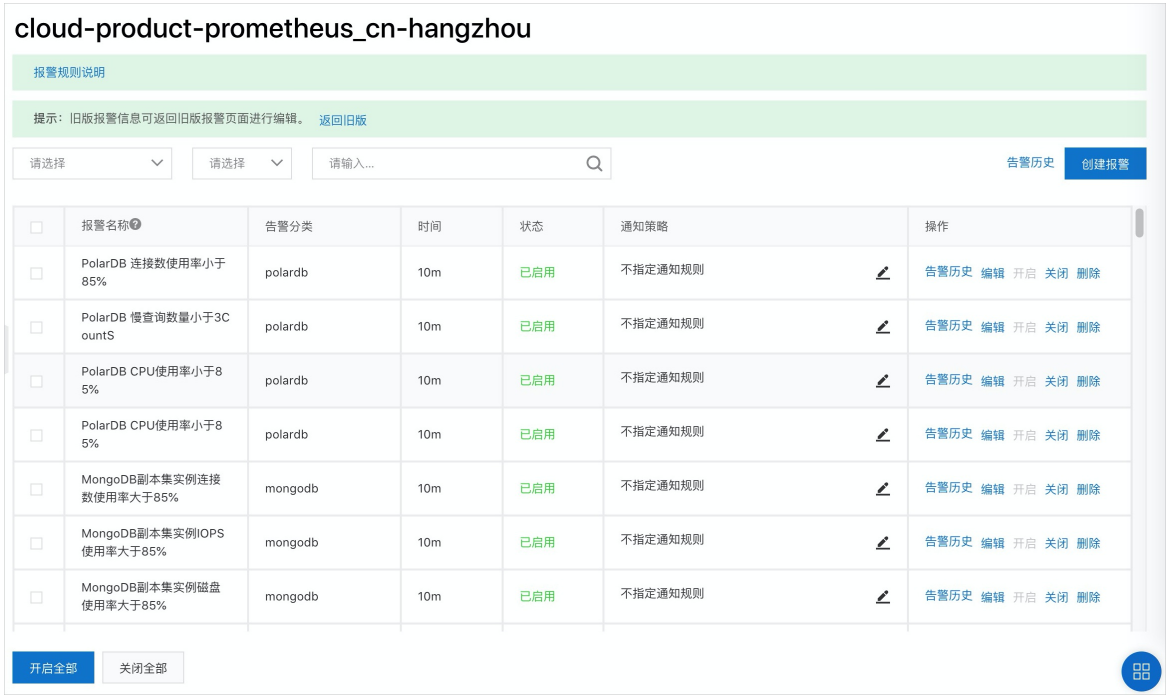
- 云服务已接入Prometheus监控，请参见Prometheus实例 for 云服务。
- 创建联系人，请参见创建联系人。

查看预置告警

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。

   Prometheus监控页面显示了所有接入Prometheus监控的集群，其中监控对象类型为云服务的实例为Prometheus云服务实例。
3. 单击Prometheus云服务实例名称。
4. 在左侧导航栏单击**报警配置**。

   报警配置页面显示了云服务的预置报警。



创建报警

如果预置的报警不满足您的业务需求，您还可以创建自定义告警。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。

   Prometheus监控页面显示了所有接入Prometheus监控的集群，其中监控对象类型为云服务的实例为Prometheus云服务实例。

3. 单击Prometheus云服务实例名称。
4. 在左侧导航栏单击**报警配置**。
5. 在报警配置页面右上角，单击**创建报警**。
6. 在**创建报警**面板，执行以下操作：
   i. （可选）从**告警模板**下拉列表，选择模板。
   ii. 在**规则名称**文本框，输入规则名称，例如：网络接收压力报警。
   iii. 在**告警表达式**文本框，输入告警表达式。例如：`(sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m]))) > 0.01`。

   > 注意
   > - PromQL语句中包含的 `$` 符号会导致报错，您需要删除包含 `$` 符号的语句中 `=` 符号左右两边的参数及 `=` 符号。例如：将 `sum (rate (container_network_receive_bytes_total{instance=~"^$HostIp.*"}[1m]))` 修改为 `sum (rate (container_network_receive_bytes_total[1m]))`。
   > - 阿里云Prometheus支持多种云服务指标的监控，Prometheus监控的云服务指标详情，请参见云服务指标。

   iv. 在**持续时间**文本框，输入持续时间N，当连续N分钟满足告警条件的时候才触发告警。例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。

   > **说明** 持续N分钟满足告警条件是指在连续N分钟内，您上面设置的PromQL语句条件都能满足。Prometheus默认数据采集周期为15s，如果没有连续4N（N×60/15=4N）个数据点满足告警条件（PromQL语句），就不会发送告警。假设Prometheus告警规则默认持续时间为1分钟，则只有连续4个数据点都满足告警条件（PromQL语句）才会触发告警。如果您想在任何一个数据点满足告警条件（PromQL语句）就发送告警，请修改持续时间为0分钟。

   v. 在**告警消息**文本框，输入告警消息。
   vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。
   vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息`，例如：`message:{{$labels.pod_name}}重启`。

   您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：
   viii. 从**通知策略**下拉列表，选择通知策略。

   创建通知策略的操作，请参见通知策略。
   ix. 单击**确定**。

报警配置页面显示创建的报警。

14.告警管理（新版）

14.1. Prometheus告警规则

通过创建Prometheus监控告警规则，您可以制定针对特定Prometheus实例的告警规则。当告警规则设置的条件满足后，系统会产生对应的告警事件。如果想要收到通知，需要进一步配置对应的通知策略以生成告警并且以短信、邮件、电话、钉群机器人、企业微信机器人或者Webhook等方式发送通知。

前提条件

已创建Prometheus实例，具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write
- Prometheus实例 for VPC
- Prometheus实例 for 云服务

功能入口

1. 登录Prometheus控制台。
2. 在左侧导航栏，单击**告警规则列表**。
3. 在**Prometheus告警规则**页面的右上角单击**创建Prometheus告警规则**。

通过静态阈值创建Prometheus告警规则

静态阈值检查类型提供了系统预设的告警指标，通过选择已有的告警指标，您可以通过语义化的方式快速创建对应指标项的告警规则。

1. 在**创建Prometheus告警规则**页面设置以下告警参数。

参数	说明	示例
告警名称	告警的名称。	生产集群-容器CPU使用率告警
检测类型	选择静态阈值。	静态阈值
Prometheus实例	选择需要创建告警的Prometheus实例。	生产集群
告警分组	选择告警分组。 不同Prometheus类型支持的告警分组不同，告警分组备选项会随着选择的Prometheus实例类型的不同产生变化。	Kubernetes负载
告警指标	选择想要配置告警的指标，每个告警分组对应不同的指标。	容器CPU使用率
告警条件	基于告警指标预置内容设置告警事件产生条件。	当容器CPU使用率 大于 *80*%时，满足告警条件。

参数	说明	示例
筛选条件	根据告警指标，设置当前配置的告警规则所适用的范围，即所有符合筛选条件的资源满足此条告警规则时，均会产生告警事件。 可选筛选条件包括： ○ **遍历**：告警规则适用于当前Prometheus实例下的所有资源。筛选条件默认为遍历。 ○ **等于**：选择该条件后，需继续输入具体资源名称。所创建的告警规则将仅适用于对应资源。不支持同时填写多个资源。 ○ **不等于**：选择该条件后，需继续输入具体资源名称。所创建的告警规则将适用于除该资源之外的其他资源。不支持同时填写多个资源。 ○ **正则匹配**：选择该条件后，按需输入正则表达式匹配相应的资源名称。所创建的告警规则将适用于符合该正则表达式的所有资源。 ○ **正则不匹配**：选择该条件后，按需输入正则表达式匹配相应的资源名称。所创建的告警规则将过滤符合该正则表达式的所有资源。 **说明** 完成筛选条件设置后，会弹出**数据预览**区域。	实例IP：遍历

参数	说明	示例
数据预览	**数据预览**区域展示告警条件对应的PromQL语句，并以时序曲线的形式展示当前告警规则配置的监控指标的值。 默认仅展示一个资源的实时值，您可以在该区域的筛选框中选择目标资源以及时间区间来查看不同时间区间和不同资源的值。 **说明** ○ 告警阈值将会以一条红色直线的形式显示在时序曲线中，满足告警阈值的时序曲线显示为深红色，不满足告警阈值的时序曲线显示为蓝色。 ○ 将鼠标悬浮于时序曲线上，可以查看对应时间点的资源详情。 ○ 在时序曲线上选中一段时间，可以查看对应时间段的时序曲线。	无
持续时间	○ 当告警条件满足时，直接产生告警事件：有任何一个数据点满足阈值，就会产生告警事件。 ○ 当告警条件满足持续N分钟时，才产生告警事件：即只有当满足阈值的时间大于等于N分钟时，才产生告警事件。	1
告警等级	自定义告警等级。默认告警等级为**默认**，告警严重程度从默认、P4、P3、P2、P1逐级上升。	默认
告警内容	用户收到的告警信息。您可以使用Go template语法在告警内容中自定义告警参数变量。	*命名空间：{{$labels.namespace}} / Pod：{{$labels.pod_name}} / 容器：{{$labels.container}} CPU使用率{{$labels.metrics_params_opt_label_value}} {{$labels.metrics_params_value}}%，当前值{{ printf "%.2f" $value }}%*
高级设置		

参数	说明	示例
快速指定通知策略	○ 不指定通知策略：若选择此选项，当完成创建告警规则后，您可以在**通知策略**页面新建通知策略并指定匹配规则和匹配条件（如告警规则名称等）来匹配该告警规则。当该告警规则被触发产生告警事件后，告警信息会被发送给通知策略中指定的联系人或联系人组。更多信息，请参见通知策略。 ○ 指定某个通知策略：若选择此项，ARMS会自动在对应的通知策略添加一条匹配规则，匹配规则内容为告警规则ID（以告警规则名称的方式呈现），以确保当前告警规则产生的告警事件一定可以被选择的通知策略匹配到。 **注意** 快速指定通知策略只能保证当前告警规则产生的告警事件一定能够被所选的通知策略匹配到并且产生对应告警。但是，当前告警规则产生的事件同时也可能被其它设置了模糊匹配的通知策略匹配到并且产生告警。告警规则产生的告警事件和通知策略之间是多对多的匹配关系。	不指定通知规则
标签	设置告警标签，设置的标签可用作通知策略匹配规则的选项。	无
注释	设置告警的注释。	无

2. 设置完成后单击**保存**。

通过自定义PromQL创建Prometheus告警规则

如果需要对静态阈值中系统预设指标之外的指标进行监控，您可以使用自定义PromQL检测类型来创建告警规则。

1. 在**创建Prometheus告警规则**页面设置以下告警参数。

参数	说明	示例
告警名称	告警的名称。	Pod的CPU使用率大于8%
检测类型	设置为自定义PromQL。	自定义PromQL
Prometheus实例	选择需要创建告警的Prometheus实例。	无

参数	说明	示例
参考指标	可选。参考指标中包括了常见指标的自定义PromQL配置方法，您可以选择已有的类似指标来进行填充，然后参考对应指标的配置方式进行修改以完成告警配置。 参考指标参数会根据选择的Prometheus实例类型自动过滤支持的告警指标。	Pod磁盘使用率告警
自定义PromQL语句	使用PromQL语句设置告警则表达式。	*max(container_fs_usage_bytes{pod!="", namespace!="arms-prom",namespace!="monitoring"}) by (pod_name, namespace, device)/max(container_fs_limit_bytes{pod!=""}) by (pod_name,namespace, device) * 100 > 90*
数据预览	**数据预览**区域展示了满足PromQL告警表达式的集群资源的指标时序曲线。 默认展示所有满足PromQL告警表达式资源的告警数据，您可以在该区域的筛选框中选择目标资源以及时间区间进行数据展示。 **说明** ○ 将鼠标悬浮于时序曲线上，可以查看对应时间点的资源详情。 ○ 在时序曲线上选中一段时间，可以查看对应时间段的时序曲线。	无
持续时间	○ 当告警条件满足时，直接产生告警事件：有任何一个数据点满足阈值，就会产生告警事件。 ○ 当告警条件满足持续N分钟时，才产生告警事件：即只有当满足阈值的时间大于等于N分钟时，才产生告警事件。	1
告警等级	自定义告警等级。默认告警等级为**默认**，告警严重程度从默认、P4、P3、P2、P1逐级上升。	默认
告警内容	用户收到的告警信息。您可以使用Go template语法在告警内容中自定义告警参数变量。	*命名空间：{{$labels.namespace}}/Pod：{{$labels.pod_name}}/磁盘设备：{{$labels.device}} 使用率超过90%，当前值{{ printf "%.2f" $value }}%*
高级设置		

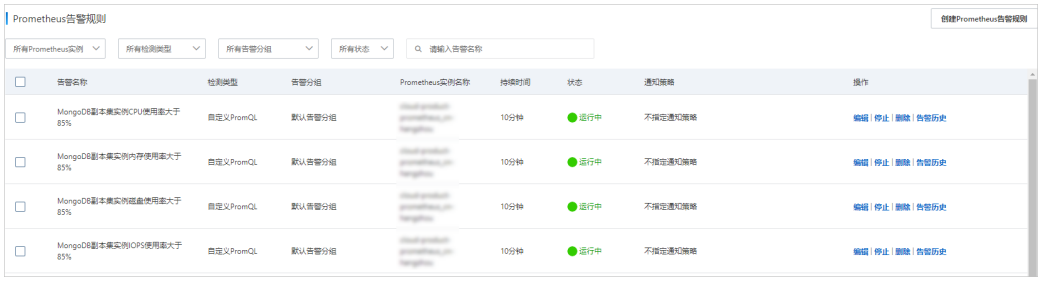
参数	说明	示例
快速指定通知策略	○ 不指定通知规则：若选择此选项，当完成创建告警规则后，您可以在**通知策略**页面新建通知策略并指定匹配规则和匹配条件（如告警规则名称等）来匹配该告警规则。当该告警规则被触发产生告警事件后，告警信息会被发送给通知策略中指定的联系人或联系人组。更多信息，请参见通知策略。 ○ 指定某个通知策略：若选择此项，ARMS会自动在对应的通知策略添加一条匹配规则，匹配规则内容为告警规则ID（以告警规则名称的方式呈现），以确保当前告警规则产生的告警事件一定可以被选择的通知策略匹配到。 **注意** 快速指定通知策略只能保证当前告警规则产生的告警事件一定能够被所选的通知策略匹配到并且产生对应告警。但是，当前告警规则产生的事件同时也可能被其它设置了模糊匹配的通知策略匹配到并且产生告警。告警规则产生的告警事件和通知策略之间是多对多的匹配关系。	不指定通知规则
标签	设置告警标签，设置的标签可用作通知策略匹配规则的选项。	无
注释	设置告警的注释。	无

2. 设置完成后单击**保存**。

管理告警规则

在**Prometheus告警规则**页面上，您可以对告警规则执行启动、停止、编辑、删除、查看告警详情等操作。

1. （可选）在**Prometheus告警规则**页面设置告警过滤条件，或者在搜索框中输入告警名称并单击搜索图标。
2. 在搜索结果列表的**操作**列中，按需对目标告警规则采取以下操作：



- 如需编辑告警规则，请单击**编辑**，在**编辑告警**页面中编辑告警规则，并单击**保存**。
- 如需删除告警规则，请单击**删除**，并在**提示**对话框中单击**确认**。

- 如需启动已停止的告警规则，请单击**启动**，并在**提示**对话框中单击**确认**。
- 如需停止已启动的告警规则，请单击**停止**，并在**提示**对话框中单击**确认**。
- 如需查看告警事件历史和告警发送历史，请单击**告警历史**，在**告警事件历史**和**告警发送历史**页面上查看相关记录。

相关文档

- 查看告警事件历史
- 通知策略

14.2. Prometheus告警规则模板

当用户拥有多个跨地域的Prometheus实例，并且需要为其中多个Prometheus实例创建相同的Prometheus告警规则时，可以使用Prometheus告警模板功能。本文介绍如何创建并管理Prometheus告警规则模板。

背景信息

对于多个跨地域的Prometheus实例，当需要为这些Prometheus实例创建告警规则时，如果每个Prometheus实例都单独创建，工作量较大并且难以同步管理。为了解决这个问题，阿里云Prometheus监控提供了告警规则模板功能，可以帮助用户快速为多个Prometheus实例创建告警规则，并且可以统一管理，降低用户管理多个Prometheus实例告警规则的成本。

Prometheus告警规则模板的主要功能

- 创建Prometheus告警规则模板
- 编辑Prometheus告警规则模版
- 应用模板
  - Prometheus实例选择模式
  - 标签控制器模式
- 删除Prometheus告警规则模版
- 查看通过模板创建的告警规则
  - 批量启动告警规则
  - 批量停止告警规则
  - 批量删除告警规则
- 批量应用模版
- 批量删除模版

创建Prometheus告警规则模板

1. 登录Prometheus控制台。
2. 在左侧导航栏中单击**Prometheus告警规则模板**。
3. 在**Prometheus告警规则模板**页面的右上角单击**告警规则模板**。
4. 在**创建Prometheus告警规则**页面设置以下参数。

   您可以选择通过静态阈值或自定义PromQL创建Prometheus告警规则模板。

   - 静态阈值检查类型提供了系统预设的告警指标，通过选择已有的告警指标，您可以通过语义化的方式快速创建对应指标项的告警规则。
   - 如果需要对静态阈值中系统预设指标之外的指标进行监控，您可以使用自定义PromQL检测类型来创建告警规则。

静态阈值检查类型

参数	说明	示例
告警规则模板名称	告警规则模板的名称。	生产集群-容器CPU使用率告警
告警规则模板描述	非必填。对模板进行描述，可以用于记录模板的含义、适用场景、备注等。	无
检测类型	选择静态阈值。	静态阈值
告警分组	选择告警分组。	Kubernetes负载
告警指标	选择想要配置告警的指标，每个告警分组对应不同的指标。	容器CPU使用率
告警条件	基于告警指标预置内容设置告警事件产生条件。	当容器CPU使用率 大于 *80*%时，满足告警条件。
筛选条件	根据告警指标，设置当前配置的告警规则所适用的范围，即所有符合筛选条件的资源满足此条告警规则时，均会产生告警事件。 可选筛选条件包括： ○ **遍历**：告警规则适用于当前Prometheus实例下的所有资源。筛选条件默认为遍历。 ○ **等于**：选择该条件后，需继续输入具体资源名称。所创建的告警规则将仅适用于对应资源。不支持同时填写多个资源。 ○ **不等于**：选择该条件后，需继续输入具体资源名称。所创建的告警规则将适用于除该资源之外的其他资源。不支持同时填写多个资源。 ○ **正则匹配**：选择该条件后，按需输入正则表达式匹配相应的资源名称。所创建的告警规则将适用于符合该正则表达式的所有资源。 ○ **正则不匹配**：选择该条件后，按需输入正则表达式匹配相应的资源名称。所创建的告警规则将过滤符合该正则表达式的所有资源。	实例IP：遍历
持续时间	○ 当告警条件满足时，直接产生告警事件：有任何一个数据点满足阈值，就会产生告警事件。 ○ 当告警条件满足持续N分钟时，才产生告警事件：即只有当满足阈值的时间大于等于N分钟时，才产生告警事件。	1

参数	说明	示例
告警等级	自定义告警等级。默认告警等级为**默认**，告警严重程度从默认、P4、P3、P2、P1逐级上升。	默认
告警内容	用户收到的告警信息。您可以使用Go template语法在告警内容中自定义告警参数变量。	*命名空间: {{$labels.namespace}} / Pod: {{$labels.pod_name}} / 容器: {{$labels.container}} CPU使用率{{$labels.metrics_params_opt_label_value}} {{$labels.metrics_params_value}}%, 当前值{{ printf "%.2f" $value }}%*
高级设置		
标签	设置告警标签，设置的标签可用作通知策略匹配规则的选项。	无
注释	设置告警的注释。	无

自定义PromQL检测类型

参数	说明	示例
告警规则模板名称	告警规则模板的名称。	Pod的CPU使用率大于8%
告警规则模板描述	非必填。对模板进行描述，可以用于记录模板的含义、适用场景、备注等。	无
检测类型	设置为自定义PromQL。	自定义PromQL
自定义PromQL语句	使用PromQL语句设置告警则表达式。	*max(container_fs_usage_bytes{pod!="", namespace!="arms-prom",namespace!="monitoring"}) by (pod_name, namespace, device)/max(container_fs_limit_bytes{pod!=""}) by (pod_name,namespace, device) * 100 > 90*
持续时间	◦ 当告警条件满足时，直接产生告警事件：有任何一个数据点满足阈值，就会产生告警事件。 ◦ 当告警条件满足持续N分钟时，才产生告警事件：即只有当满足阈值的时间大于等于N分钟时，才产生告警事件。	1
告警等级	自定义告警等级。默认告警等级为**默认**，告警严重程度从默认、P4、P3、P2、P1逐级上升。	默认
告警内容	用户收到的告警信息。您可以使用Go template语法在告警内容中自定义告警参数变量。	*命名空间: {{$labels.namespace}}/Pod: {{$labels.pod_name}}/磁盘设备: {{$labels.device}} 使用率超过90%，当前值{{ printf "%.2f" $value }}%*

参数	说明	示例
高级设置		
标签	设置告警标签，设置的标签可用作通知策略匹配规则的选项。	无
注释	设置告警的注释。	无

应用模板

创建Prometheus告警模板后，您可以通过应用模板来为Prometheus实例创建告警规则，或者根据现有模板更新Prometheus实例的告警规则。

Prometheus告警模板目前提供两种模板应用模式。

1. 在控制台的**Prometheus告警规则模板**页面，单击需要应用的告警规则模板右侧的**应用模板**。
2. 在**应用模板**页面的**Prometheus实例选择模式**页签，选择Prometheus实例，然后单击**确认**。

   > **说明** 您可以通过Prometheus实例名称、地域和Prometheus实例类型筛选需要使用该告警模板的Prometheus实例。

3. 在弹出的提示框中选择是否更新已创建的告警规则，然后单击**确认**。

   ARMS将会使用当前告警规则模板在选中的Prometheus实例中创建告警规则。

   - 未选中**更新已经创建的告警规则**：如果选中的Prometheus实例已经存在通过当前告警模板创建的告警规则，那么在创建告警规则时将会提示**已经使用此模板创建过告警规则，不进行更新**，此时告警规则将不会更新。
   - 选中**更新已经创建的告警规则**：如果选中的Prometheus实例已经存在通过当前告警模板创建的告警规则，那么此时会根据最新的告警模板内容更新Prometheus告警规则。

     > **注意** 如果对应的告警规则单独修改过，并且保留了告警规则与模板的联系，那么单独修改的内容会将被告警规则模板覆盖而丢失。

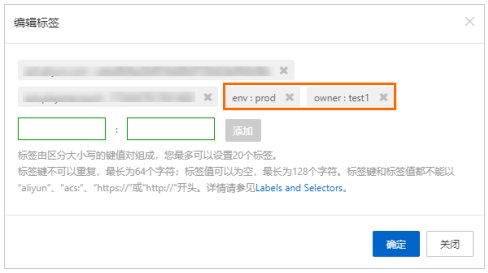
利用容器服务为集群添加的标签，在**标签控制器模式**页签通过标签匹配的方式筛选符合条件的集群，每个模板可以单独配置，配置后就会直接为符合标签匹配条件的Prometheus实例创建告警规则。

Prometheus告警规则模板会根据标签动态更新告警规则：

- 当模板发生变化时，满足标签匹配条件的集群对应的告警规则会根据模板自动更新。
- 修改通过标签控制器模式创建的告警规则时，如果保留了告警规则和模板之间的映射关系，告警规则会被自动覆盖。
- 当容器服务集群的标签发生变化时，与集群对应的Prometheus实例标签也会变化，Prometheus告警模板也会根据标签匹配条件动态删除或创建告警规则。

1. 在容器服务管理控制台为集群添加标签。
   i. 在左侧导航栏单击**集群**。
   ii. 在**集群列表**页面，将鼠标悬浮于目标集群名称右侧的图标上，然后单击**编辑标签**。

iii. 在**编辑标签**对话框中添加集群标签，然后单击**确定**。



2. 在控制台的**Prometheus告警规则模板**页面，单击需要应用的告警规则模板右侧的**应用模板**。
3. 在**应用模板**页面，单击**标签控制器模式**页签，设置匹配的标签和匹配表达式。

Prometheus实例选择模式

标签控制器模式

删除模板

当Prometheus告警模板不需要使用时，可以删除Prometheus告警模板。删除模板时，您可以根据需要，选择是否保留由此告警模板创建的告警规则。

1. 在控制台的**Prometheus告警规则模板**页面，单击目标告警规则模板右侧的**删除**。
2. 在弹出的提示对话框中选择是否同时删除从模板创建的告警规则，然后单击**确认**。
   - 选中**同时删除从模版创建的告警规则**：将会删除与模版相关联的Prometheus告警规则。如果通过模板创建的Prometheus告警规则被手动编辑过，并且保存时选择了**不保留模板与规则的映射关系**，则对应规则不会被删除。
   - 为选中**同时删除从模版创建的告警规则**：通过模版创建的Prometheus告警规则将会保留。

查看通过模板创建的告警规则

您可以查看并批量管理通过模板创建的告警规则。

1. 在控制台的**Prometheus告警规则模板**页面，单击目标告警规则模板右侧的**查看目标创建规则**。
2. 在**从模板创建的告警规则**页面批量管理告警规则。
   - 批量启动告警：选中需要启动的告警规则，然后单击**批量启动告警**。
   - 批量停止告警：选中需要停止的告警规则，然后单击**批量停止告警**。
   - 批量删除告警：选中需要删除的告警规则，然后单击**批量删除告警**。

注意事项

编辑通过Prometheus告警规则模板创建的Prometheus告警规则后，在保存时需要选择是否保留模板与规则的映射关系。

- **保留此告警规则与告警规则模板的映射关系**：保留映射关系后，如果在应用对应告警规则模板时选择了当前规则对应的Prometheus实例，并且选择了强制更新时，本次编辑内容可能会丢失。
- **解除此告警规则与告警规则模板的映射关系**：解除映射关系后，此告警规则会被视为独立规则。建议修改告警规则名称，否则使用对应告警规则模板再次应用到该Prometheus实例时，会因为告警规则名称重复而导致创建失败。

14.3. 查看告警发送历史

通过**告警发送历史**页面，您可以筛选并查看根据通知策略分派的告警内容，并管理告警。

功能入口

1. 登录Prometheus控制台。
2. 在左侧导航栏选择**告警管理 > 告警发送历史**。

告警

告警页面显示了告警的等级、名称、处理人、创建时间和状态。

在**告警**页面，您可以执行以下操作：

- 设置筛选字段，然后单击**搜索**，查看对应的告警发送历史。

  > **说明** 打开更多开关，可以设置更多的过滤筛选字段。

字段	说明
告警名称	创建的告警规则的名称。
告警状态	告警目前的处理状态，共有以下3种状态： ○ 待认领 ○ 处理中 ○ 已解决
告警等级	告警的严重级别。Prometheus告警规则可以在创建时指定告警规则产生的告警的级别。应用监控告警规则和前端监控告警规则产生的告警都是默认级别。 告警级别有以下几种： ○ 默认 ○ P1 ○ P2 ○ P3 ○ P4 ○ P5
处理人	告警的处理人。
更多	
通知策略	告警对应的通知策略。

字段	说明
集成类型	告警事件对应的集成类型。 ○ PROMETHEUS：自建Prometheus上报的告警事件。 ○ ARMS：ARMS所有产品上报的告警事件。 ○ 日志服务：日志服务上报的告警事件。 ○ 云监控：云监控上报的告警事件。 ○ 自定义集成：自定义集成上报的告警事件。 ○ ARMS-前端监控：ARMS前端监控上报的告警事件。 ○ ARMS-应用监控：ARMS应用监控上报的告警事件。 ○ ARMS-Prometheus：ARMS Prometheus上报的告警事件。 ○ ARMS-自定义监控：ARMS自定义监控上报的告警事件。 ○ ARMS-链路追踪：ARMS链路追踪上报的告警事件。
创建时间	告警产生的时间段。

- 单击告警名称，可以查看目标告警的详细信息。更多信息，请参见告警详情。
- 对于未解决的告警，可以认领、解决、指定告警处理人或修改告警等级。具体操作，请参见处理告警。

告警详情

告警详情页面显示了告警等级、发送信息、状态，以及告警基本信息、关联事件和活动记录。

在**告警详情**页面，您可以执行以下操作：

对于未解决的告警，可以认领、解决、指定告警处理人或修改告警等级。具体操作，请参见处理告警。

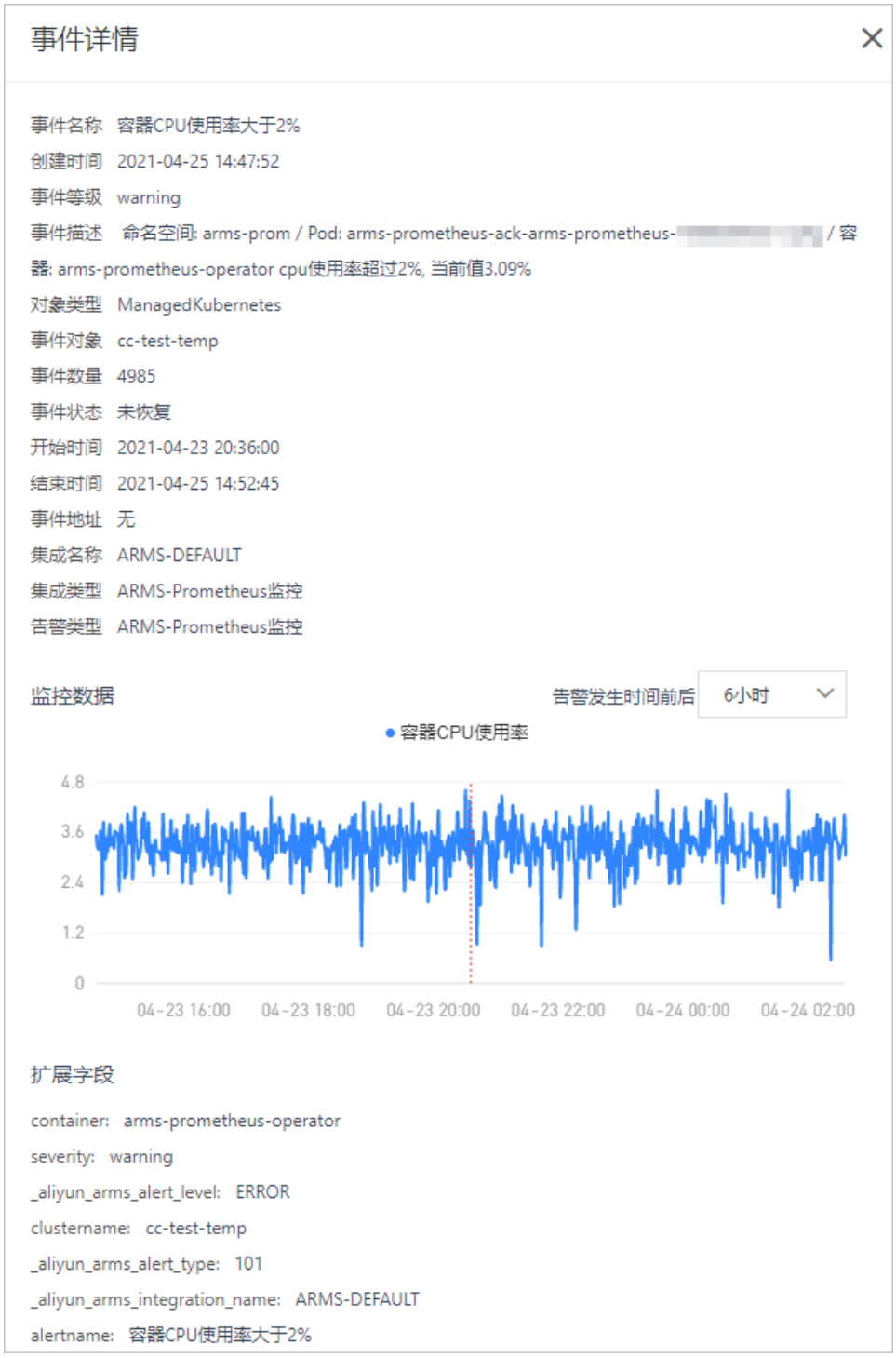
在**详情**、**事件**和**活动**页签可以分别查看以下信息：

- **详情**页签显示了告警创建时间、告警对象、处理人和通知人。
- **事件**页签显示了告警关联事件、事件创建时间和事件状态。单击事件名称，可以查看目标事件的详细信息。更多信息，请参见事件详情。

- **活动**页签显示了告警的活动记录。

事件详情

事件详情面板显示了事件的基本信息、监控数据和扩展字段。



在事件详情面板的**监控数据**区域，您可以执行以下操作：

- 在**监控数据**区域右上角，设置数据显示的时间段为告警发生时间前后的6小时、12小时或1天。
- 使用光标选中一段时间，可以查看指定时间段的监控数据。单击**重置**，可以将曲线图恢复。

处理告警

在控制台中，对于未解决的告警，可以认领、解决、指定告警处理人或修改告警等级。

1. 绑定手机号。

i. 在钉钉群中单击告警卡片中的**查看详情**。

ii. 在钉钉群中首次查看告警信息时会弹出**绑定手机号**面板，输入手机号，并单击**发送验证码**。

iii. 收到验证码后输入验证码，然后单击**确定**。

2. 登录Prometheus控制台。
3. 在左侧导航栏选择**告警管理 > 告警发送历史**。
4. 在**告警**页面右上角的**选择操作人**下拉框选择操作人。

> **说明**
> - 操作人的名称为钉钉昵称。
> - 由于一个阿里云账号可能存在多人同时使用的情况，所以需要通过选择操作人来区分实际每次操作告警的人员。

5. 在目标告警右侧，或者单击目标告警，在**告警详情**页面：
   - 单击**认领**，可以将自己设置为当前告警的处理人。
   - 单击**解决**，可以关闭目标告警。
   - 单击更多图标，然后单击**指定告警处理人**，可以指定钉钉群里的联系人为告警处理人。
   - 单击更多图标，然后单击**修改告警等级**，可以修改告警等级。

14.4. 查看告警事件历史

通过**事件列表**页面，您可以筛选并查看所有告警事件。

功能入口

1. 登录Prometheus控制台。
2. 在左侧导航栏选择**告警管理 > 告警事件历史**。

事件列表

事件列表页面显示了未恢复告警和已恢复告警的事件名称、通知策略、创建时间、事件数量、事件状态、事件对象和对象类型。

在**事件列表**页面，您可以执行以下操作：

- 设置筛选字段，然后单击**搜索**，可以查看对应的告警事件。

> **说明** 打开更多开关，可以设置更多的过滤筛选字段。

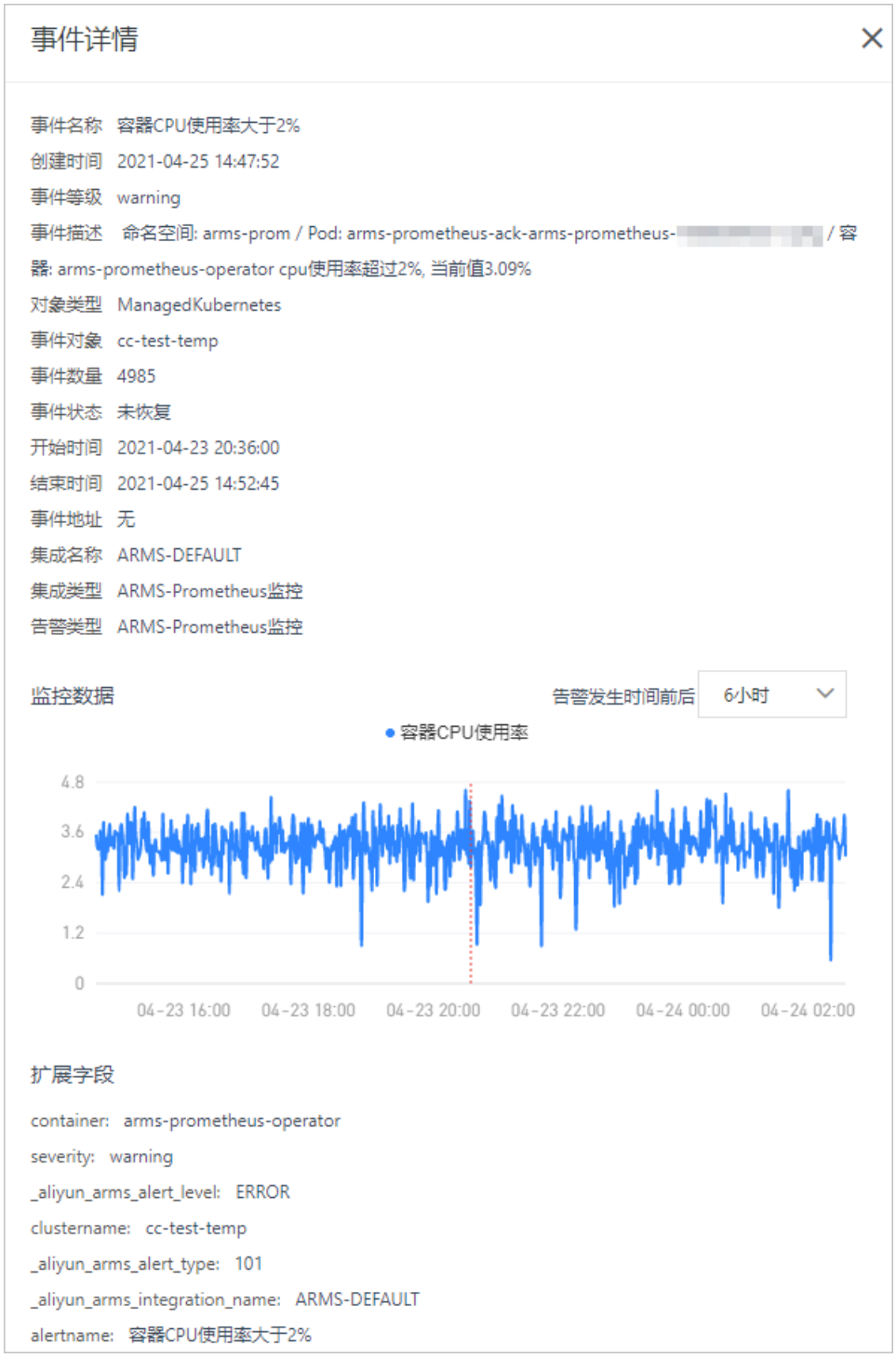
字段	说明
事件名称	创建的告警规则的名称。

字段	说明
集成名称	告警事件对应的集成名称。 ○ ARMS-DEFAULT ○ PROMETHEUS集成 ○ 云监控集成 ○ 日志服务集成 ○ 自定义集成
事件状态	告警事件的状态，共有以下3种状态： ○ 未恢复：告警事件持续被触发。 ○ 静默：在设置的自动恢复告警事件的时间内，和第一条告警事件名称和等级相同的告警事件的状态为静默。 ○ 已恢复：在设置的时间内，告警事件不再触发。
更多	
事件对象	监控任务名称或集群名称。
对象类型	前端站点、应用或Prometheus告警。
集成类型	告警事件对应的集成类型。 ○ PROMETHEUS：自建Prometheus上报的告警事件。 ○ ARMS：ARMS所有产品上报的告警事件。 ○ 日志服务：日志服务上报的告警事件。 ○ 云监控：云监控上报的告警事件。 ○ 自定义集成：自定义集成上报的告警事件。 ○ ARMS-前端监控：ARMS前端监控上报的告警事件。 ○ ARMS-应用监控：ARMS应用监控上报的告警事件。 ○ ARMS-Prometheus：ARMS Prometheus上报的告警事件。 ○ ARMS-自定义监控：ARMS自定义监控上报的告警事件。 ○ ARMS-链路追踪：ARMS链路追踪上报的告警事件。

- 单击事件所在行，查看目标事件的详细信息。更多信息，请参见事件详情。
- 单击页面右上角的**发送测试事件**，在弹出的对话框中设置集成名称和事件内容，可以发送一次告警事件测试到指定集成中。

事件详情

事件详情面板显示了事件的基本信息、监控数据和扩展字段。



在事件详情面板的**监控数据**区域，您可以执行以下操作：

- 在**监控数据**区域右上角，设置数据显示的时间段为告警发生时间前后的6小时、12小时或1天。
- 使用光标选中一段时间，可以查看指定时间段的监控数据。单击**重置**，可以将曲线图恢复。

14.5. 通知策略

通过设置通知策略，您可以制定针对告警事件的分派条件。当分派条件被触发时，系统会以您指定的通知方式向处理人发送告警信息，以提醒处理人采取必要的问题解决措施。

前提条件

创建联系人，具体操作，请参见创建联系人。

新建通知策略

1. 登录Prometheus控制台。
2. 在左侧导航栏中选择**告警管理 > 通知策略**。
3. 在**通知策略列表**页面单击右上角的**创建通知策略**。
4. 在右侧区域顶部文本框中输入通知策略名称。
5. 在**匹配告警事件规则**区域的进行以下操作。
   i. 选择告警过滤来源或不过滤来源。
      - 指定过滤来源：通知策略会针对指定来源的告警事件进行分派条件过滤并发送通知。
      - 不过滤来源：通知策略会针对所有告警事件进行分派条件过滤并发送通知。

ii. 设置分派条件表达式，您可以自定义标签或选择已有的标签。例如：alertname等于内存使用率。

已有的标签包括以下三种：

- 告警规则表达式指标中携带的标签。
- Prometheus监控通过告警规则创建的标签，请参见Prometheus告警规则。
- 系统自带的默认标签，默认标签说明如下。

标签	说明
alertname	告警名称，格式为：告警名称_集群名称。
_aliyun_arms_alert_level	告警等级。
_aliyun_arms_alert_type	告警类型。
_aliyun_arms_alert_rule_id	告警规则对应的ID。
_aliyun_arms_region_id	地域ID。
_aliyun_arms_userid	用户ID。
_aliyun_arms_involvedObject_type	关联对象子类型，如ManagedKubernetes，ServerlessKubernetes。
_aliyun_arms_involvedObject_kind	关联对象分类，如app，cluster。
_aliyun_arms_involvedObject_id	关联对象ID。
_aliyun_arms_involvedObject_name	关联对象名称。

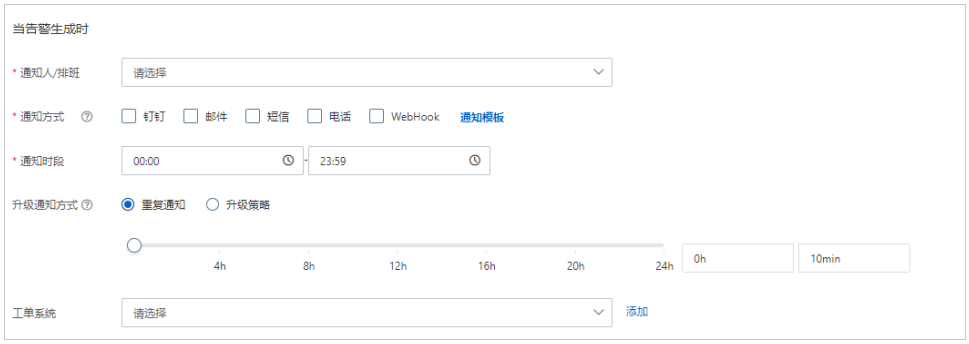
> **说明**
>
> - 如果需同时满足多个分派条件才告警，则单击**+条件**编辑第二条分派条件。
> - 如果需满足任意一个匹配告警事件规则就告警，则单击**+添加规则**编辑第二条匹配告警事件规则。

6. 在**事件处理**区域，设置以下参数。

> **说明** 智能降噪功能说明，请参见配置智能降噪。

参数	说明
处理方式	○ 生成告警：将监测到的告警发送给联系人。 ○ 不告警（丢弃事件）：监测到告警后不发送给联系人。 **说明** 如果同时设置了两个相同的通知策略且处理方式分别设置为*生成告警*和*不告警（丢弃事件）*，则*不告警（丢弃事件）*的优先级高于*生成告警*，即不会给联系人发送告警信息。
高级配置	
事件分组	○ 不分组：所有告警会以一条信息发送给处理人。 ○ 指定相同字段内容的事件分到一个组：设置分组字段，相同字段的告警内容会分别通过独立信息发送给处理人。
告警自动恢复	当告警下面全部事件都恢复时，告警状态是否自动恢复为已解决。当告警恢复时，系统将会发送通知给处理人。

7. 在**当告警生成时**区域，设置以下参数。



参数	说明

参数	说明
通知人/排班	通知人支持设置联系人、联系人组、IM机器人和排班表。 ○ 联系人的创建方法，请参见创建联系人。 ○ 排班表的创建方法，请参见排班策略。
通知方式	通知方式支持钉钉、邮件、WebHook、短信和电话，可以同时选择多种方式。 **说明** ○ 未验证手机号的联系人无法使用电话通知方式。验证手机号的操作，请参见验证手机号。 ○ 单击**通知模板**，可以在**通知模板**对话框中设置邮件、短信和电话的通知信息格式。
通知时段	告警会在设置的通知时段内重复发送告警通知。
升级通知方式	对于长期未解决的告警可以选择重复通知或升级通知来提醒联系人及时解决。 ○ 重复通知：选择重复通知后，需要设置告警发送频率。所有告警会在指定通知时段内以设置的告警发送频率循环发送告警信息直至告警恢复。 ○ 升级策略：选择升级策略后，需要选择已有的升级策略或添加一个新的升级策略。具体操作，请参见升级策略。 **说明** 单击**详情**，可以查看当前选中的升级策略的详细信息。
工单系统	选择告警需要推送到的工单系统。工单系统的集成，请参见通过Jira账号信息集成Jira工单系统。

8. 设置完成后，单击右上角的图标。

管理通知策略

通知策略新建完成后会在**通知策略列表**中显示，您可以在**通知策略列表**中执行以下操作：

- 单击通知策略名称右侧的更多图标，选择**编辑**、**停用**、**启用**、**复制**或**删除**可以管理该通知策略。
- 单击目标通知策略，在右侧页面右上角选择相应的图标，可以编辑、刷新、复制和删除该通知策略。

14.6. 升级策略

对于长期未解决的告警，可以选择升级通知来提醒联系人及时解决。在通知策略中添加升级策略后，系统会以您指定的通知方式向处理人发送告警信息，以提醒处理人采取必要的问题解决措施。

前提条件

创建联系人，具体操作，请参见创建联系人。

新建升级策略

1. 登录Prometheus控制台。

2. 在左侧导航栏中选择**告警管理 > 升级策略**。
3. 在**升级策略列表**区域单击**新增策略**。
4. 在右侧区域顶部文本框中输入升级策略名称。
5. 在升级规则区域，设置通知条件，即当告警在一段时间内未被认领或未被解决时发送告警通知。例如：当告警10分钟未认领时发送升级通知。

> **说明** 单击**+添加规则**，可以新增一条升级规则。

参数	说明
通知人	通知人支持设置联系人、联系人组、钉群和排班表。 ○ 联系人的创建方法，请参见创建联系人。 ○ 排班表的创建方法，请参见排班策略。
通知方式	通知方式支持钉钉、邮件、WebHook、短信和电话，可以同时选择多种方式。 **说明** 未验证手机号的联系人无法使用电话通知方式。验证手机号的操作，请参见验证手机号。
通知时段	满足通知条件的告警会在设置的通知时段内重复发送告警通知。
重复次数	重复发送告警通知的次数。当告警不满足通知条件后，告警通知将不再发送。 **说明** 至少发送一次告警，即设置为0时也会发送一次告警。

6. 设置完成后，单击右上角的图标。

管理升级策略

升级策略新建完成后会在**升级策略列表**中显示，您可以在**升级策略列表**中执行以下操作：

- 单击升级策略名称右侧的更多图标，选择**编辑**、**停用**、**启用**、**复制**或**删除**可以管理该升级策略。
- 单击目标升级策略，在右侧页面右上角选择相应的图标，可以编辑、刷新、复制和删除该升级策略。

14.7. 联系人管理

14.7.1. 联系人

通知策略的分派规则被触发时可以向您指定的联系人发送通知。联系人支持通过电话、短信、邮件和钉钉等方式接收告警通知。

创建联系人

1. 登录Prometheus控制台。
2. 在左侧导航栏中选择**告警管理 > 联系人**。
3. 在**联系人**页签上，单击右上角的**新建联系人**。

4. 在**新建联系人**对话框中输入联系人姓名，根据实际需求输入联系人手机号码、邮箱或钉钉机器人地址，设置是否接收系统通知，然后单击**确认**。

参数	说明
姓名	自定义联系人姓名。
手机号码	设置联系人的手机号码后，可以通过电话和短信的方式接收告警通知。 **说明** 仅验证过的手机号码可以在通知策略中使用电话的通知方式，验证手机号的操作，请参见验证手机号。
邮箱	设置联系人的邮箱地址后，可以通过邮箱接收告警通知。
钉钉机器人	设置联系人的钉钉机器人地址后，可以钉钉群中接收告警通知。获取钉钉机器人地址的方法，请参见获取钉钉机器人Webhook地址。
钉钉机器人是否发送每日统计	选中后，需要输入每日统计信息发送的时间点，使用英文逗号（,）分隔多个发送时间点，时间点格式为 `HH:SS` 。告警管理将在设置的时间点发送今日产生告警的总数、解决数和待解决数。

注意

- 手机号码、邮箱和钉钉机器人至少填写一项。每个手机号码或邮箱只能用于一个联系人。
- 最多可添加100个联系人。

验证手机号

仅验证过的手机号可以在通知策略中使用电话的通知方式。

1. 登录Prometheus控制台。
2. 在左侧导航栏中选择**告警管理 > 联系人**。
3. 在**联系人**页签上，选择为一个或多个联系人验证手机号。
   - 如需为单个联系人验证手机号：单击未验证手机号右侧的**未验证**。
   - 如需为多个联系人批量验证手机号：在复选框中选中需要验证手机号的联系人，然后单击**批量验证**。

     系统将会给各联系人发送验证手机号短信。
4. 使用浏览器打开短信中的链接。

5. 在验证页面确认手机号信息，然后单击**验证**。

管理联系人

创建联系人后，您可以在**联系人**页签查询、编辑或删除联系人：

- 如需搜索联系人，请在**联系人**页签上，从搜索下拉框中选择**姓名**、**手机号**或**Email**，然后在搜索框中输入联系人姓名、手机号码或邮箱的关键字，并单击图标。
- 如需编辑联系人，请单击联系人右侧**操作**列中的**编辑**，在**编辑联系人**对话框中编辑信息，然后单击**确认**。
- 如需删除单个联系人，请单击联系人右侧**操作**列中的**删除**，然后在提示对话框中单击**确认**。
- 如需删除多个联系人，请选择目标联系人，单击**批量删除**，然后在提示对话框中单击**确认**。

14.7.2. 联系人组

创建通知策略时，您可以将联系人组指定为通知对象，当通知策略的分派规则被触发时，告警管理会向该联系人组中的联系人通过电话、短信、邮件和钉钉等方式发送告警通知。

前提条件

已创建联系人，具体操作，请参见创建联系人。

创建联系人组

1. 登录Prometheus控制台。

2. 在左侧导航栏中选择**告警管理 > 联系人**。
3. 在**联系人**页面单击**联系人组**页签，并单击右上角的**新建联系组**。
4. 在**新建联系组**对话框中输入**组名**，选择**报警联系人**，并单击**确认**。

管理联系人组

创建联系人组后，您可以在**联系人组**页签查询、编辑或删除联系人：

- 如需搜索联系组，请在**联系人组**页签的搜索框中输入联系人组名称的全部或部分字符，并单击图标。
- 如需编辑联系组，请单击联系人组右侧的图标，并在**编辑联系组**对话框中编辑相关信息，然后单击**确认**。

  > **说明** 仅支持编辑当前用户创建的联系人组。

- 如需查看联系组中的联系人信息，请单击联系人组左侧的图标展开联系组。

  > **说明** 您可以在展开模式下移除联系组中当前用户创建的联系人。如需移除，请单击目标联系人**操作**列中的**移除**。

- 如需删除联系组，请单击联系人组右侧的图标，然后在弹出的提示对话框中单击**确认**。

  > **注意**
  > - 删除联系组之前，请确保目标联系组没有添加至通知策略中，否则可能导致告警通知无法发送。
  > - 仅支持删除当前用户创建的联系人组。

14.7.3. 钉钉机器人

在告警管理中创建钉钉机器人后，您可以在通知策略中指定对应的钉钉群用于接收告警。当通知策略的分派规则被触发时，系统会自动向您指定的钉钉群发送告警通知。钉钉群收到通知后，您可以在钉钉中对告警进行管理。

前提条件

已在钉钉客户端创建用于接收告警通知的钉钉群。

在创建告警通知钉群

1. 在钉钉群中创建自定义机器人并获取机器人Webhook地址，具体操作，请参见获取钉钉机器人Webhook地址。
2. 登录Prometheus控制台。
3. 在左侧导航栏中选择**告警管理 > 联系人**。
4. 在**联系人**页面单击**IM机器人**页签。
5. 在**IM机器人**页签上，单击右上角的**新建机器人**。
6. 在**新建机器人**对话框中设置以下参数，然后单击**确认**。

> **注意** 在通知策略中需要选择**通知方式**为**钉钉**才能在钉钉群中接收告警，具体操作，请参见通知策略。

参数	说明
名称	自定义钉钉机器人的名称。
类型	选择**类型**为**钉钉**。
机器人地址	输入钉钉机器人的Webhook地址。
机器人是否发送每日统计	选中后，需要输入每日统计信息发送的时间点，使用英文逗号（,）分隔多个发送时间点，时间点格式为 `HH:SS` 。告警管理将在设置的时间点发送今日产生告警的总数、解决数和待解决数。

在钉钉群中管理告警

在钉钉群中收到告警通知后，您可以在钉钉群里查看并管理告警。更多信息，请参见在告警通知群中处理告警。

相关操作

创建机器人后，您可以在**IM机器人**页签查询、编辑或删除机器人：

- 如需搜索指定的机器人，在搜索框中输入机器人名称的关键字，然后单击图标。
- 如需编辑机器人，单击机器人右侧**操作**列中的**编辑**，在**编辑机器人**对话框中修改信息，然后单击**确认**。
- 如需删除机器人，单击机器人右侧**操作**列中的**删除**，并在提示对话框中单击**确认**。

14.7.4. 企业微信机器人

在告警管理中创建企业微信机器人后，您可以在通知策略中指定对应的企业微信群用于接收告警。当通知策略的分派规则被触发时，系统会自动向您指定的企业微信群发送告警通知。企业微信群收到通知后，您可以在企业微信群中对告警进行管理。

前提条件

已在企业微信中创建用于接收告警通知的企业微信群。

步骤一：获取机器人Webhook地址

1. 打开并登录企业微信。
2. 单击企业微信群右上角的 `...` 图标，然后单击**添加群机器人**。
3. 在**添加机器人**页面单击**新创建一个机器人**。
4. 在**创建机器人**页面自定义机器人名称，然后单击**添加机器人**。
5. 单击**复制地址**，保存企业微信机器人的Webhook地址。

步骤二：创建机器人

1. 登录Prometheus控制台。
2. 在左侧导航栏中选择**告警管理 > 联系人**。
3. 在**联系人**页面单击**IM机器人**页签。
4. 在**IM机器人**页签上，单击右上角的**新建机器人**。
5. 在**新建机器人**对话框中设置以下参数，然后单击**确认**。

参数	说明
名称	自定义企业微信机器人的名称。
类型	选择**类型**为**企业微信**。
机器人地址	输入企业微信机器人的Webhook地址。
机器人是否发送每日统计	选中后，需要输入每日统计信息发送的时间点，使用英文逗号（,）分隔多个发送时间点，时间点格式为 `HH:SS` 。告警管理将在设置的时间点发送今日产生告警的总数、解决数和待解决数。

在企业微信群中管理告警

在企业微信群中收到告警通知后，您可以在企业微信群里查看并管理告警。更多信息，请参见在告警通知群中处理告警。

相关操作

创建机器人后，您可以在**IM机器人**页签查询、编辑或删除机器人：

- 如需搜索指定的机器人，在搜索框中输入机器人名称的关键字，然后单击 图标。
- 如需编辑机器人，单击机器人右侧**操作**列中的**编辑**，在**编辑机器人**对话框中修改信息，然后单击**确认**。
- 如需删除机器人，单击机器人右侧**操作**列中的**删除**，并在提示对话框中单击**确认**。

14.7.5. 飞书机器人

在告警管理中创建飞书机器人后，您可以在通知策略中指定对应的飞书群用于接收告警。当通知策略的匹配告警事件规则被触发时，系统会自动向您指定的飞书群发送告警通知。飞书群收到通知后，您可以在飞书群中对告警进行管理。

步骤一：获取机器人Webhook地址

1. 打开并登录飞书。
2. （可选）单击+图标，然后单击**创建群组**，新建一个用于发送告警的群组。
3. 单击飞书群右侧的图标，然后单击**群机器人**。
4. 在**群机器人**面板单击**添加机器人**，然后选择自定义机器人。
5. 在机器人配置页设置机器人名称和描述，然后单击**添加**。
6. 复制Webhook地址，然后选中**自定义关键词**并输入关键词为告警。

7. 单击**完成**。

步骤二：创建机器人

1. 登录Prometheus控制台。
2. 在左侧导航栏中选择**告警管理 > 联系人**。
3. 在**联系人**页面单击**IM机器人**页签。
4. 在**IM机器人**页签上，单击右上角的**新建机器人**。
5. 在**新建机器人**对话框中设置以下参数，然后单击**确认**。

参数	说明
名称	自定义飞书机器人的名称。
类型	选择**类型**为**飞书**。
机器人地址	输入飞书机器人的Webhook地址。
机器人是否发送每日统计	选中后，需要输入每日统计信息发送的时间点，使用英文逗号（,）分隔多个发送时间点，时间点格式为`HH:SS`。告警管理将在设置的时间点发送今日产生告警的总数、解决数和待解决数。

步骤三：设置通知策略

新建或编辑通知策略，选择**通知人**为上一步创建的飞书机器人。具体操作，请参见通知策略。

在飞书群中管理告警

在飞书群中收到告警通知后，您可以在飞书群里查看并管理告警。更多信息，请参见在告警通知群中处理告警。

相关操作

创建机器人后，您可以在**IM机器人**页签查询、编辑或删除机器人：

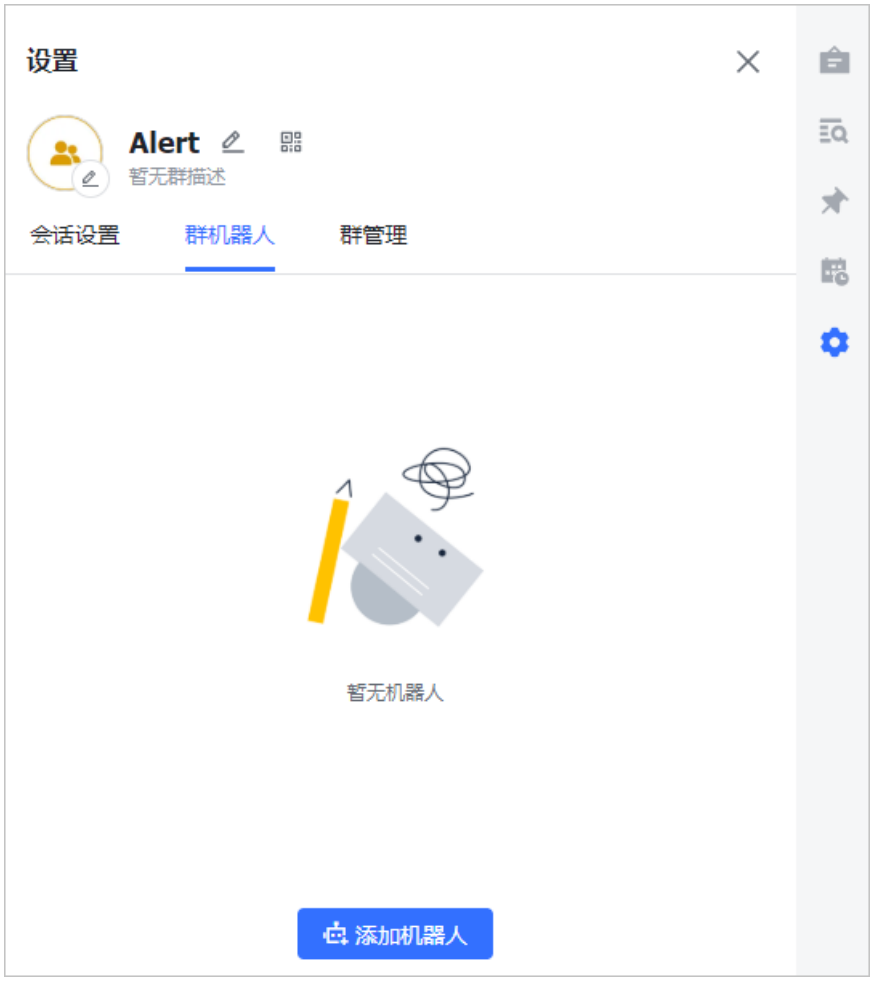
- 如需搜索指定的机器人，在搜索框中输入机器人名称的关键字，然后单击 图标。
- 如需编辑机器人，单击机器人右侧**操作**列中的**编辑**，在**编辑机器人**对话框中修改信息，然后单击**确认**。
- 如需删除机器人，单击机器人右侧**操作**列中的**删除**，并在提示对话框中单击**确认**。

14.7.6. 通过Webhook自定义告警通知人

创建通知策略时，您可以将告警通知发送到自定义的Webhook地址中。告警管理支持对飞书、微信、钉钉等群组发送Webhook告警，本文以飞书为例，介绍如何创建Webhook告警。

步骤一：获取Webhook地址

1. 打开并登录飞书。
2. 单击+图标，然后单击**创建群组**，新建一个用于发送告警的群组。
3. 单击群组设置图标，然后单击**群机器人**页签。
4. 在**群机器人**页签单击**添加机器人**。



5. 在**添加机器人**面板选择**Custom Bot**。

6. 在配置页设置显示名称和描述，然后单击**添加**。

7. 在添加情况区域单击**复制链接**，然后单击**完成**。

步骤二：创建Webhook联系人

1. 登录Prometheus控制台。
2. 在左侧导航栏中选择**告警管理 > 联系人**。
3. 在**联系人**页签上，单击右上角的**新建webhook**。
4. 在**创建Webhook**对话框中输入配置信息。

   基本参数描述如下所示。

参数	说明
Webhook名称	必填，自定义Webhook名称。
Post和Get	必填，设置请求方法。URL不可超过100个字符。 此例中选择Post，并将步骤一：获取Webhook地址中保存的Webhook地址粘贴至右侧文本框。
Header和Param	非必填，设置请求头，不可超过200个字符。 单击**+添加**，可以添加其他Header信息或Param信息。默认请求头为Content-Type: text/plain; charset=UTF-8，Header和Param个数总数不能超过6个。 此例中设置以下两个Header: ◦ Arms-Content-Type：json ◦ Content-Type：application/json

参数	说明
通知模板	告警触发时发送的通知模板，非必填，在Post方法下出现，可使用$content占位符输出通知内容，不可超过500个字符。更多信息，请参见通知模板的变量说明。 通知模板如下： `{` `"告警名称":"{{ .commonLabels.alertname }}{{if .commonLabels.clustername }}",` `"集群名称":"{{ .commonLabels.clustername }} {{ end }}{{if eq "app" .commonLabels._aliyun_arms_involvedObject_kind }}",` `"应用名称":"{{ .commonLabels._aliyun_arms_involvedObject_name }} {{ end }}",` `"通知策略":"{{ .dispatchRuleName }}",` `"告警时间":"{{ .startTime }}",` `"告警内容":"{{ for .alerts }} {{ .annotations.message }} {{ end }}"` `}` 此处以飞书为例可以设置如下文本格式： `{` `"msg_type": "text",` `"content": {` `"text": "告警名称: {{ .commonLabels.alertname }}\n{{if .commonLabels.clustername }}集群名称: {{ .commonLabels.clustername }}\n{{ end }}{{if eq "app" .commonLabels._aliyun_arms_involvedObject_kind }}应用名称: {{ .commonLabels._aliyun_arms_involvedObject_name }}\n{{ end }}通知策略: {{ .dispatchRuleName }} \n告警时间: {{ .startTime }} \n告警内容: {{ for .alerts }} {{ .annotations.message }}\n {{ end }}"` `}` `}`

参数	说明
恢复模板	告警恢复时发送的通知模板，非必填，在Post方法下出现，可使用$content占位符输出通知内容，不可超过500个字符。更多信息，请参见通知模板的变量说明。 恢复模板如下： `{` `"告警名称":"{{ .commonLabels.alertname }}{{if .commonLabels.clustername }}",` `"集群名称":"{{ .commonLabels.clustername }} {{ end }}{{if eq "app" .commonLabels._aliyun_arms_involvedObject_kind }}",` `"应用名称":"{{ .commonLabels._aliyun_arms_involvedObject_name }} {{ end }}",` `"通知策略":"{{ .dispatchRuleName }}",` `"恢复时间":"{{ .endTime }}",` `"告警内容":"{{ for .alerts }} {{ .annotations.message }} {{ end }}"` `}` 此处以飞书为例可以设置如下文本格式： `{` `"msg_type": "text",` `"content": {` `"text": "告警名称: {{ .commonLabels.alertname }}\n{{if .commonLabels.clustername }}集群名称: {{ .commonLabels.clustername }}\n{{ end }}{{if eq "app" .commonLabels._aliyun_arms_involvedObject_kind }}应用名称: {{ .commonLabels._aliyun_arms_involvedObject_name }}\n{{ end }}恢复时间: {{ .startTime }} \n通知策略: {{ .dispatchRuleName }} \n恢复告警内容: {{ for .alerts }} {{ .annotations.message }}\n {{ end }}"` `}` `}`

5. （可选）单击**测试**，验证配置是否成功。
6. 单击**创建**。

步骤三：设置通知策略

在通知策略中需要选择**通知方式**为**WebHook**才能在自定义的Webhook中接收告警。

1. 新建或编辑通知策略，选择**通知人**为Webhook联系人，在**通知方式**区域选中**WebHook**。具体操作，请参见通知策略。

> **说明** Webhook告警的超时时间为5秒，如果发出请求后5秒内没有返回，即没有收到告警信息，则表示发送失败。

14.7.7. 获取钉钉机器人Webhook地址

设置钉群或在联系人中设置钉钉机器人时需要先在钉钉群中获取自定义机器人Webhook地址。本文介绍如何获取钉钉机器人Webhook地址。

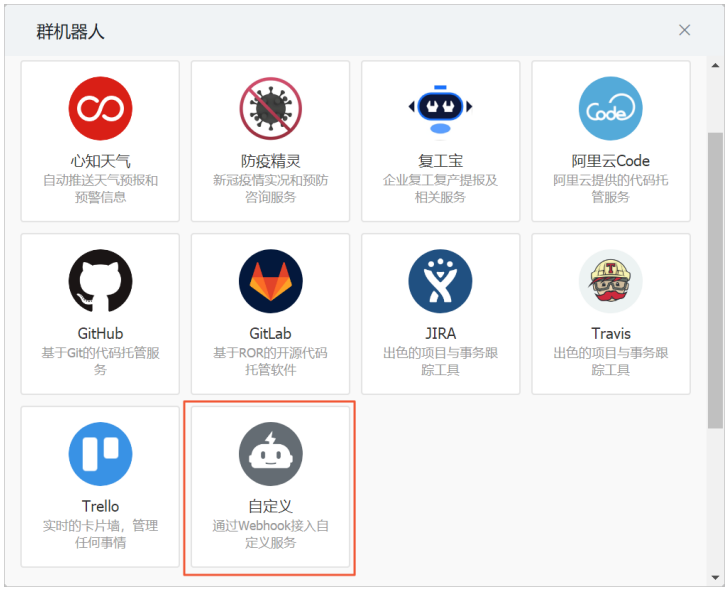
前提条件

已创建用于接收告警通知的钉钉群。

操作步骤

请按照以下步骤在钉钉群中添加自定义钉钉机器人并获取Webhook地址。

1. 在PC版钉钉上打开您想要添加报警机器人的钉钉群，并单击右上角的群设置图标。
2. 在**群设置**面板中单击**智能群助手**。
3. 在**智能群助手**面板单击**添加机器人**。
4. 在**群机器人**对话框单击**添加机器人**区域的**+**图标，然后选择添加**自定义**。



5. 在**机器人详情**对话框单击**添加**。
6. 在**添加机器人**对话框中执行以下操作。

   i. 设置机器人头像和名字。
   ii. **安全设置**选中**自定义关键词**，设置关键词为告警。
   iii. 选中**我已阅读并同意《自定义机器人服务及免责条款》**。

iv. 单击**完成**。

> **说明** 更多关于钉钉机器人的操作，请参见自定义机器人接入。

7. 在**添加机器人**对话框中复制生成的机器人Webhook地址，然后单击**完成**。

15.参考信息

15.1. 基础指标说明

阿里云Prometheus监控按照指标上报次数收费。指标分为两种类型：基础指标和自定义指标。其中，基础指标不收费，自定义指标于2020年01月06日开始收费。
Prometheus监控支持的容器集群基础指标如下表所示。

- Prometheus状态信息的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	promhttp_metric_handler_requests_in_flight
	go_memstats_mallocs_total
	go_memstats_lookups_total
	go_memstats_last_gc_time_seconds
	go_memstats_heap_sys_bytes
	go_memstats_heap_released_bytes
	go_memstats_heap_objects
	go_memstats_heap_inuse_bytes
	go_memstats_heap_idle_bytes
	go_memstats_heap_alloc_bytes
	go_memstats_gc_sys_bytes
	go_memstats_gc_cpu_fraction
	go_memstats_frees_total
	go_memstats_buck_hash_sys_bytes
	go_memstats_alloc_bytes_total
	go_memstats_alloc_bytes
	scrape_duration_seconds
	go_info
	go_goroutines
	scrape_samples_post_metric_relabeling
	go_gc_duration_seconds_sum

任务名称（Job Name）	指标名称
_arms-prom/kubelet/1	go_gc_duration_seconds_count
	blackbox_exporter_config_last_reload_successful
	blackbox_exporter_config_last_reload_success_timestamp_seconds
	scrape_samples_scraped
	blackbox_exporter_build_info
	arms_prometheus_target_scrapes_sample_out_of_order_total
	arms_prometheus_target_scrapes_sample_out_of_bounds_total
	arms_prometheus_target_scrapes_sample_duplicate_timestamp_total
	scrape_series_added
	arms_prometheus_target_scrapes_exceeded_sample_limit_total
	arms_prometheus_target_scrapes_cache_flush_forced_total_arms-prom/kubelet/1
	arms_prometheus_target_scrape_pools_total
	statsd_metric_mapper_cache_gets_total
	statsd_metric_mapper_cache_hits_total
	statsd_metric_mapper_cache_length
	arms_prometheus_target_scrape_pools_failed_total
	up
	arms_prometheus_target_scrape_pool_reloads_total
	arms_prometheus_target_scrape_pool_reloads_failed_total

- API Server任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称

任务名称（Job Name）	指标名称
apiserver	apiserver_request_duration_seconds_bucket（默认废弃）
	apiserver_admission_controller_admission_duration_seconds_bucket
	apiserver_request_total
	rest_client_requests_total
	apiserver_admission_webhook_admission_duration_seconds_bucket
	apiserver_current_inflight_requests
	up
	apiserver_admission_webhook_admission_duration_seconds_count
	scrape_samples_post_metric_relabeling
	scrape_samples_scraped
	scrape_series_added
	scrape_duration_seconds

- Ingress任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	nginx_ingress_controller_request_duration_seconds_bucket
	nginx_ingress_controller_response_duration_seconds_bucket
	nginx_ingress_controller_response_size_bucket
	nginx_ingress_controller_request_size_bucket
	nginx_ingress_controller_bytes_sent_bucket
	go_gc_duration_seconds
	nginx_ingress_controller_nginx_process_connections
	nginx_ingress_controller_request_duration_seconds_sum

任务名称（Job Name）	指标名称
	nginx_ingress_controller_request_duration_seconds_count
	nginx_ingress_controller_bytes_sent_sum
	nginx_ingress_controller_request_size_sum
	nginx_ingress_controller_response_duration_seconds_count
	nginx_ingress_controller_response_duration_seconds_sum
	nginx_ingress_controller_response_size_count
	nginx_ingress_controller_bytes_sent_count
	nginx_ingress_controller_response_size_sum
	nginx_ingress_controller_request_size_count
	promhttp_metric_handler_requests_total
	nginx_ingress_controller_nginx_process_connections_total
	go_memstats_mcache_sys_bytes
	go_memstats_lookups_total
	go_threads
	go_memstats_sys_bytes
	go_memstats_last_gc_time_seconds
	go_memstats_heap_sys_bytes
	go_memstats_heap_objects
	go_memstats_heap_inuse_bytes
	go_memstats_heap_idle_bytes
	go_memstats_heap_alloc_bytes
	go_memstats_gc_sys_bytes
	promhttp_metric_handler_requests_in_flight
	go_memstats_stack_sys_bytes

任务名称（Job Name）	指标名称
arms-ack-ingress	go_memstats_stack_inuse_bytes
	go_memstats_gc_cpu_fraction
	go_memstats_frees_total
	go_memstats_buck_hash_sys_bytes
	go_memstats_alloc_bytes_total
	go_memstats_alloc_bytes
	nginx_ingress_controller_nginx_process_num_procs
	go_info
	go_memstats_mallocs_total
	go_memstats_other_sys_bytes
	go_goroutines
	scrape_samples_post_metric_relabeling
	scrape_samples_scraped
	process_virtual_memory_max_bytes
	process_virtual_memory_bytes
	scrape_duration_seconds
	go_memstats_heap_released_bytes
	go_gc_duration_seconds_sum
	go_memstats_next_gc_bytes
	go_gc_duration_seconds_count
	nginx_ingress_controller_config_hash
	nginx_ingress_controller_config_last_reload_successful
	nginx_ingress_controller_config_last_reload_successful_timestamp_seconds
	nginx_ingress_controller_ingress_upstream_latency_seconds_count

任务名称（Job Name）	指标名称
	nginx_ingress_controller_ingress_upstream_latency_seconds_sum
	process_start_time_seconds
	nginx_ingress_controller_nginx_process_cpu_seconds_total
	scrape_series_added
	nginx_ingress_controller_nginx_process_oldest_start_time_seconds
	nginx_ingress_controller_nginx_process_read_bytes_total
	nginx_ingress_controller_nginx_process_requests_total
	nginx_ingress_controller_nginx_process_resident_memory_bytes
	nginx_ingress_controller_nginx_process_virtual_memory_bytes
	nginx_ingress_controller_nginx_process_write_bytes_total
	nginx_ingress_controller_requests
	go_memstats_mcache_inuse_bytes
	nginx_ingress_controller_success
	process_resident_memory_bytes
	process_open_fds
	process_max_fds
	process_cpu_seconds_total
	go_memstats_mspan_sys_bytes
	up
	go_memstats_mspan_inuse_bytes
	nginx_ingress_controller_ssl_expire_time_seconds
	nginx_ingress_controller_leader_election_status

- CoreDNS任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	coredns_forward_request_duration_seconds_bucket
	coredns_dns_request_size_bytes_bucket
	coredns_dns_response_size_bytes_bucket
	coredns_kubernetes_dns_programming_duration_seconds_bucket
	coredns_dns_request_duration_seconds_bucket
	coredns_plugin_enabled
	coredns_health_request_duration_seconds_bucket
	go_gc_duration_seconds
	coredns_forward_responses_total
	coredns_forward_request_duration_seconds_sum
	coredns_forward_request_duration_seconds_count
	coredns_dns_requests_total
	coredns_forward_conn_cache_misses_total
	coredns_dns_responses_total
	coredns_cache_entries
	coredns_cache_hits_total
	coredns_forward_conn_cache_hits_total
	coredns_forward_requests_total
	coredns_dns_request_size_bytes_sum
	coredns_dns_response_size_bytes_count
	coredns_dns_response_size_bytes_sum
	coredns_dns_request_size_bytes_count
	scrape_duration_seconds
	scrape_samples_scraped
	scrape_series_added

任务名称（Job Name）	指标名称
arms-ack-coredns	
	up
	scrape_samples_post_metric_relabeling
	go_memstats_lookups_total
	go_memstats_last_gc_time_seconds
	go_memstats_heap_sys_bytes
	coredns_build_info
	go_memstats_heap_released_bytes
	go_memstats_heap_objects
	go_memstats_heap_inuse_bytes
	go_memstats_heap_idle_bytes
	go_memstats_heap_alloc_bytes
	go_memstats_gc_sys_bytes
	go_memstats_sys_bytes
	go_memstats_stack_sys_bytes
	go_memstats_mallocs_total
	go_memstats_gc_cpu_fraction
	go_memstats_stack_inuse_bytes
	go_memstats_frees_total
	go_memstats_buck_hash_sys_bytes
	go_memstats_alloc_bytes_total
	go_memstats_alloc_bytes
	coredns_cache_misses_total
	go_memstats_other_sys_bytes
	go_memstats_mcache_inuse_bytes
	go_goroutines

任务名称（Job Name）	指标名称
	process_virtual_memory_max_bytes
	process_virtual_memory_bytes
	go_gc_duration_seconds_sum
	go_gc_duration_seconds_countarms-ack-coredns
	go_memstats_next_gc_bytes
	coredns_dns_request_duration_seconds_count
	coredns_reload_failed_total
	coredns_panics_total
	coredns_local_localhost_requests_total
	coredns_kubernetes_dns_programming_duration_seconds_sum
	coredns_kubernetes_dns_programming_duration_seconds_count
	coredns_dns_request_duration_seconds_sum
	coredns_hosts_reload_timestamp_seconds
	oredns_health_request_failures_total
	process_start_time_seconds
	process_resident_memory_bytes
	process_open_fds
	process_max_fds
	process_cpu_seconds_total
	coredns_health_request_duration_seconds_sum
	coredns_health_request_duration_seconds_count
	go_memstats_mspan_sys_bytes
	coredns_forward_max_concurrent_rejects_total
	coredns_forward_healthcheck_broken_total
	go_memstats_mcache_sys_bytes

任务名称（Job Name）	指标名称
	go_memstats_mspan_inuse_bytes
	go_threads
	go_info

- 采集自Kube-State-Metrics任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	kube_pod_container_status_waiting_reason
	kube_pod_status_phase
	kube_pod_container_status_last_terminated_reason
	kube_pod_container_status_terminated_reason
	kube_pod_status_ready
	kube_node_status_condition
	kube_pod_container_status_running
	kube_pod_container_status_restarts_total
	kube_pod_container_info
	kube_pod_container_status_waiting
	kube_pod_container_status_terminated
	kube_pod_labels
	kube_pod_owner
	kube_pod_info
	kube_pod_container_resource_limits
	kube_persistentvolume_status_phase
	kube_pod_container_resource_requests_memory_bytes
	kube_pod_container_resource_requests_cpu_cores
	kube_pod_container_resource_limits_memory_bytes
	kube_node_status_capacity

任务名称（Job Name）	指标名称
_kube-state-metrics	kube_service_info
	kube_pod_container_resource_limits_cpu_cores
	kube_deployment_status_replicas_updated
	kube_deployment_status_replicas_unavailable
	kube_deployment_spec_replicas
	kube_deployment_created
	kube_deployment_metadata_generation
	kube_deployment_status_replicas
	kube_deployment_labels
	kube_deployment_status_observed_generation
	kube_deployment_status_replicas_available
	kube_deployment_spec_strategy_rollingupdate_max_unavailable
	kube_daemonset_status_desired_number_scheduled
	kube_daemonset_updated_number_scheduled
	kube_daemonset_status_number_ready
	kube_daemonset_status_number_misscheduled
	kube_daemonset_status_number_available
	kube_daemonset_status_current_number_scheduled
	kube_daemonset_created
	kube_node_status_allocatable_cpu_cores
	kube_node_status_capacity_memory_bytes
	kube_node_spec_unschedulable
	kube_node_status_allocatable_memory_bytes
	kube_node_labels
	kube_node_info

任务名称（Job Name）	指标名称
	kube_namespace_labels
	kube_node_status_capacity_cpu_cores
	kube_node_status_capacity_pods
	kube_node_status_allocatable_pods
	kube_node_spec_taint
	kube_statefulset_status_replicas
	kube_statefulset_replicas
	kube_statefulset_created
	up
	scrape_samples_scraped
	scrape_duration_seconds
	scrape_samples_post_metric_relabeling
	scrape_series_added

- Kubelet任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	rest_client_request_duration_seconds_bucket
	apiserver_client_certificate_expiration_seconds_bucket
	kubelet_pod_worker_duration_seconds_bucket
	kubelet_pleg_relist_duration_seconds_bucket
	workqueue_queue_duration_seconds_bucket
	rest_client_requests_total
	go_gc_duration_seconds
	process_cpu_seconds_total
	process_resident_memory_bytes
	kubernetes_build_info
	kubelet_node_name

任务名称（Job Name）	指标名称
_arms/kubelet/metric	kubelet_certificate_manager_client_ttl_seconds
	kubelet_certificate_manager_client_expiration_renew_errors
	scrape_duration_seconds
	go_goroutines
	crape_samples_post_metric_relabeling
	scrape_samples_scraped
	scrape_series_added
	up
	apiserver_client_certificate_expiration_seconds_count
	workqueue_adds_total
	workqueue_depth

- Cadvisor任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	container_memory_failures_total（默认废弃）
	container_memory_rss
	container_spec_memory_limit_bytes
	container_memory_failcnt
	container_memory_cache
	container_memory_swap
	container_memory_usage_bytes
	container_memory_max_usage_bytes
	container_cpu_load_average_10s
	container_fs_reads_total（默认废弃）
	container_fs_writes_total（默认废弃）
	container_network_transmit_errors_total

任务名称（Job Name）	指标名称
_arms/kubelet/cadvisor	container_network_receive_bytes_total
	container_network_transmit_packets_total
	container_network_receive_errors_total
	container_network_receive_bytes_total
	container_network_receive_errors_total
	container_network_transmit_errors_total
	container_memory_working_set_bytes
	container_cpu_usage_seconds_total
	container_fs_reads_bytes_total
	container_fs_writes_bytes_total
	container_spec_cpu_quota
	container_cpu_cfs_periods_total
	container_cpu_cfs_throttled_periods_total
	container_cpu_cfs_throttled_seconds_total
	container_fs_inodes_free
	container_fs_io_time_seconds_total
	container_fs_io_time_weighted_seconds_total
	container_fs_limit_bytes
	container_tasks_state（默认废弃）
	container_fs_read_seconds_total（默认废弃）
	container_fs_write_seconds_total（默认废弃）
	container_fs_usage_bytes
	container_fs_inodes_total
	container_fs_io_current
	scrape_duration_seconds
	scrape_samples_scraped

任务名称（Job Name）	指标名称
	machine_cpu_cores
	machine_memory_bytes
	scrape_samples_post_metric_relabeling
	scrape_series_added
	up
_arms-prom/kube-apiserver/cadvisor	scrape_duration_seconds
	up
	scrape_samples_scraped
	scrape_samples_post_metric_relabeling
	scrape_series_added

- ACK Scheduler任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
ack-scheduler	rest_client_request_duration_seconds_bucket
	scheduler_pod_scheduling_attempts_bucket
	rest_client_requests_total
	scheduler_pending_pods
	scheduler_scheduler_cache_size
	up

- etcd任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
etcd	etcd_disk_backend_commit_duration_seconds_bucket
	up
	etcd_server_has_leader
	etcd_debugging_mvcc_keys_total
	etcd_debugging_mvcc_db_total_size_in_bytes

任务名称（Job Name）	指标名称
	etcd_server_leader_changes_seen_total

- Node任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	node_filesystem_size_bytes
	node_filesystem_readonly
	node_filesystem_free_bytes
	node_filesystem_avail_bytes
	node_cpu_seconds_total
	node_network_receive_bytes_total
	node_network_receive_errs_total
	node_network_transmit_bytes_total
	node_network_receive_packets_total
	node_network_transmit_drop_total
	node_network_transmit_errs_total
	node_network_up
	node_network_transmit_packets_total
	node_network_receive_drop_total
	go_gc_duration_seconds
	node_load5
	node_filefd_allocated
	node_exporter_build_info
	node_disk_written_bytes_total
	node_disk_writes_completed_total
	node_disk_write_time_seconds_total
	node_nf_conntrack_entries
	node_nf_conntrack_entries_limit

任务名称（Job Name）	指标名称
node-exporter	node_processes_max_processes
	node_processes_pids
	node_sockstat_TCP_alloc
	node_sockstat_TCP_inuse
	node_sockstat_TCP_tw
	node_timex_offset_seconds
	node_timex_sync_status
	node_uname_info
	node_vmstat_pgfault
	node_vmstat_pgmajfault
	node_vmstat_pgpgin
	node_vmstat_pgpgout
	node_disk_reads_completed_total
	node_disk_read_time_seconds_total
	process_cpu_seconds_total
	node_disk_read_bytes_total
	node_disk_io_time_weighted_seconds_total
	node_disk_io_time_seconds_total
	node_disk_io_now
	node_context_switches_total
	node_boot_time_seconds
	process_resident_memory_bytes
	node_intr_total
	node_load1
	go_goroutines
	scrape_duration_seconds

任务名称（Job Name）	指标名称
	node_load15
	scrape_samples_post_metric_relabeling
	node_netstat_Tcp_PassiveOpens
	scrape_samples_scraped
	node_netstat_Tcp_CurrEstab
	scrape_series_added
	node_netstat_Tcp_ActiveOpens
	node_memory_MemTotal_bytes
	node_memory_MemFree_bytes
	node_memory_MemAvailable_bytes
	node_memory_Cached_bytes
	up
	node_memory_Buffers_bytes

- GPU任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
	go_gc_duration_seconds
	promhttp_metric_handler_requests_total
	scrape_series_added
	up
	scrape_duration_seconds
	scrape_samples_scraped
	scrape_samples_post_metric_relabeling
	go_memstats_mcache_inuse_bytes
	process_virtual_memory_max_bytes
	process_virtual_memory_bytes
	process_start_time_seconds

任务名称（Job Name）	指标名称
gpu-exporter	go_memstats_next_gc_bytes
	go_memstats_heap_objects
	process_resident_memory_bytes
	process_open_fds
	process_max_fds
	go_memstats_other_sys_bytes
	go_gc_duration_seconds_count
	go_memstats_heap_alloc_bytes
	process_cpu_seconds_total
	nvidia_gpu_temperature_celsius
	go_memstats_stack_inuse_bytes
	nvidia_gpu_power_usage_milliwatts
	nvidia_gpu_num_devices
	nvidia_gpu_memory_used_bytes
	nvidia_gpu_memory_total_bytes
	go_memstats_stack_sys_bytes
	nvidia_gpu_memory_allocated_bytes
	nvidia_gpu_duty_cycle
	nvidia_gpu_allocated_num_devices
	promhttp_metric_handler_requests_in_flight
	go_memstats_sys_bytes
	go_memstats_gc_sys_bytes
	go_memstats_gc_cpu_fraction
	go_memstats_heap_released_bytes
	go_memstats_frees_total
	go_threads

任务名称（Job Name）	指标名称
	go_memstats_mspan_sys_bytes
	go_memstats_buck_hash_sys_bytes
	go_memstats_alloc_bytes_total
	go_memstats_heap_sys_bytes
	go_memstats_mspan_inuse_bytes
	go_memstats_alloc_bytes
	go_info
	go_memstats_last_gc_time_seconds
	go_memstats_heap_inuse_bytes
	go_memstats_mcache_sys_bytes
	go_memstats_lookups_total
	go_memstats_mallocs_total
	go_gc_duration_seconds_sum
	go_goroutines
	go_memstats_heap_idle_bytes

- PV任务类型（Job）的任务名称和基础指标如下：

任务名称（Job Name）	指标名称
k8s-csi-cluster-pv	cluster_pvc_detail_num_total
	cluster_pv_detail_num_total
	cluster_pv_status_num_total
	cluster_scrape_collector_success
	cluster_scrape_collector_duration_seconds
	alibaba_cloud_storage_operator_build_info
	cluster_pvc_status_num_total
	scrape_duration_seconds
	scrape_samples_post_metric_relabeling

任务名称（Job Name）	指标名称
	scrape_samples_scraped
	scrape_series_added
	up
k8s-csi-node-pv	cluster_scrape_collector_duration_seconds
	cluster_scrape_collector_success
	alibaba_cloud_csi_driver_build_info
	up
	scrape_series_added
	scrape_samples_post_metric_relabeling
	scrape_samples_scraped
	scrape_duration_seconds

Prometheus实例 for 云服务（即云服务类型的Prometheus实例）支持的基础指标如下表所示。

指标分类	指标名称	指标说明
ECS	cpu_util_lization	（ECS）CPU使用率
	internet_in_rate	（ECS）公网流入流量平均速率
	internet_out_rate	（ECS）公网流出流量平均速率
	disk_read_bps	（ECS）所有磁盘读取BPS
	disk_write_bps	（ECS）所有磁盘每秒读取次数
	vpc_public_ip_internet_in_Rate	（ECS）IP维度公网流入平均速率
	vpc_public_ip_internet_out_Rate	（ECS）IP维度公网流出带宽使用率
	cpu_total	（Agent）cpu.total
	memory_totalspace	（Agent）memory.total.space
	memory_usedutilization	（Agent）memory.used.utilization
	diskusage_utilization	（Agent）disk.usage.utilization_device
	cpu_usage_average	CPU使用率
	disk_usage	磁盘使用率

指标分类	指标名称	指标说明
RDS	iops_usage	IOPS使用率
	connection_usage	连接数使用率
	data_delay	只读实例延迟
	memory_usage	内存使用率
	mysql_network_in_new	MySQL网络流入带宽
	mysql_network_out_new	MySQL网络流出带宽
	mysql_active_sessions	MySQL_ActiveSessions
	sqlserver_network_in_new	SQLServer网络流入带宽
	sqlserver_network_out_new	SQLServer网络流出带宽
NAT	snat_connection	SNAT连接数
	snat_connection_drop_limit	历史累积最大限制丢弃连接数
	snat_connection_drop_rate_limit	历史累积新建限制丢弃连接数
	net_rx_rate	流入带宽
	net_tx_rate	流出带宽
	net_rx_pkgs	流入包速率
	net_tx_pkgs	流出包速率
RocketMQ	consumer_lag_gid	消息堆积
	receive_message_count_gid	Consumer（GroupId） 每分钟接收消息数量
	send_message_count_gid	Producer（GroupId） 每分钟发送消息的数量
	consumer_lag_topic	消息堆积（GroupID&Topic）
	receive_message_count_topic	Consumer（GroupId&Topic） 每分钟接收消息数量
	send_message_count_topic	Producer（GroupId&Topic） 每分钟发送消息数量
	receive_message_count	每分钟接收消息数量
	send_message_count	每分钟发送消息数量

指标分类	指标名称	指标说明
SLB	healthy_server_count	后端健康ECS实例个数
	unhealthy_server_count	后端异常ECS实例个数
	packet_tx	每秒流入数据包数
	packet_rx	每秒流出数据包数
	traffic_rx_new	流入带宽
	traffic_tx_new	流出带宽
	active_connection	TCP活跃连接数
	inactive_connection	端口非活跃连接数
	new_connection	TCP新建连接数
	max_connection	端口并发连接数
	instance_active_connection	实例活跃连接数
	instance_new_connection	实例每秒新建连接数
	instance_max_connection	实例每秒最大并发连接数
	instance_drop_connection	实例每秒丢失连接数
	instance_traffic_rx	实例每秒入bit数
	instance_traffic_tx	实例每秒出bit数
E-MapReduce（EMR）	active_applications	active状态的作业个数
	active_users	active的用户数
	aggregate_containers_allocated	总共分配的container个数
	aggregate_containers_released	总共释放的container个数
	allocated_containers	分配的container个数
	apps_completed	已完成的作业数
	apps_failed	失败的作业数
	apps_killed	被杀死的作业数
	apps_pending	等待的作业数
	apps_running	运行中的作业数

指标分类	指标名称	指标说明
	apps_submitted	提交的作业数
	available_mb	当前队列当前可用的内存大小
	available_vcores	当前队列可用的VCore个数
	pending_containers	等待的container个数
	reserved_containers	预留的container个数
EIP	net_rx_rate	流入带宽
	net_tx_rate	流出带宽
	net_rx_pkgs_rate	流入包速率
	net_tx_pkgs_rate	流出包速率
	out_ratelimit_drop_speed	限速丢包速率
OSS	availability	可用性
	request_valid_rate	有效请求率
	success_rate	成功请求占比
	network_error_rate	网络错误请求占比
	total_request_count	总请求数
	valid_count	有效请求数
	internet_send	公网流出流量
	internet_recv	公网流入流量
	intranet_send	内网流出流量
	intranet_recv	内网流入流量
	success_count	成功请求总数
	network_error_count	网络错误请求总数
	client_timeout_count	客户端超时错误请求总数
	node_cpu_utilization	Elasticsearch实例节点CPU使用率
	node_heap_memory_utilization	Elasticsearch实例节点HeapMemory使用率
	node_stats_exception_log_count	Exception次数

指标分类	指标名称	指标说明
Elasticsearch（ES）	node_stats_full_gc_collection_count	FullGc次数
	node_disk_utilization	Elasticsearch实例节点磁盘使用率
	node_load_1m	节点Load_1m
	cluster_query_qps	集群查询QPS
	cluster_index_qps	ClusterIndexQPS
Logstash	cpu_percent	Logstash实例节点CPU使用率
	node_heap_memory	节点内存使用量
	node_disk_usage	Logstash实例节点磁盘使用率
	cpu_utilization	CPU使用率
	connection_count	连接数
	logic_qps	逻辑QPS
	logic_rt	逻辑RT
	memory_utilization	内存利用率
	network_input_traffic	网络输入带宽
	network_output_traffic	网络输出带宽
	physics_qps	物理QPS
	physics_rt	物理RT
	thread_count	活跃线程数
	com_insert_select	私有RDS_MySQL每秒InsertSelect量
	com_replace	私有RDS_MySQL每秒Replace量
	com_replace_select	私有RDS_MySQL每秒ReplaceSelect量
	com_select	私有RDS_MySQL每秒Select量
	com_update	私有RDS_MySQL每秒Update量
	conn_usage	私有RDS_MySQL连接数利用率
	cpu_usage	私有RDS_MySQL CPU使用率
	disk_usage	私有RDS_MySQL磁盘使用率

指标分类	指标名称	指标说明
DRDS	ibuf_dirty_ratio	私有RDS_MySQL_BP脏页百分率
	ibuf_pool_reads	私有RDS_MySQL每秒物理读次数
	ibuf_read_hit	私有RDS_MySQL_BP读命中率
	ibuf_request_r	私有RDS_MySQL每秒逻辑读次数
	ibuf_request_w	私有RDS_MySQL每秒逻辑写次数
	ibuf_use_ratio	私有RDS_MySQL_BP利用率
	inno_data_read	私有RDS_MySQL_InnoDB每秒读取数据量
	inno_data_written	私有RDS_MySQL_InnoDB每秒写入数据量
	inno_row_delete	私有RDS_MySQL_InnoDB每秒删除行数
	inno_row_insert	私有RDS_MySQL_InnoDB每秒插入行数
	inno_row_readed	私有RDS_MySQL_InnoDB每秒读取行数
	inno_row_update	私有RDS_MySQL_InnoDB每秒更新行数
	innodb_log_write_requests	私有RDS_MySQL_InnoDB每秒日志写请求次数
	innodb_log_writes	私有RDS_MySQL_InnoDB每秒日志物理写次数
	innodb_os_log_fsyncs	私有RDS_MySQL_InnoDB每秒日志fsync量
	input_traffic_ps	私有RDS_MySQL网络流入带宽
	iops_usage	私有RDS_MySQL IOPS利用率
	mem_usage	私有RDS_MySQL内存利用率
	output_traffic_ps	私有RDS_MySQL网络流出带宽
	qps	私有RDS_MySQL每秒查询量
	slave_lag	私有RDS_MySQL只读实例延迟
	slow_queries	私有RDS_MySQL每秒慢查询量
	tb_tmp_disk	私有RDS_MySQL每秒创建临时表数量
	instance_disk_capacity	实例磁盘使用率
	instance_message_input	实例消息生产量

指标分类	指标名称	指标说明
Kafka	instance_message_output	实例消息消费量
	topic_message_input	Topic消息生产量
	topic_message_output	Topic消息消费量
MongoDB	cpu_utilization	CPU使用率
	memory_utilization	内存使用百分比
	disk_utilization	磁盘使用率
	iops_utilization	IOPS使用率
	qps	每秒请求数
	connect_amount	连接数使用量
	instance_disk_amount	实例占用磁盘空间量
	data_disk_amount	数据占用磁盘空间量
	log_disk_amount	日志占用磁盘空间量
	intranet_in	内网网络入流量
	intranet_out	内网网络出流量
	number_requests	请求数
	op_insert	Insert操作次数
	op_query	Query操作次数
	op_update	Update操作次数
	op_delete	Delete操作次数
	op_getmore	Getmore操作次数
	op_command	Command操作次数
	active_connections	活跃连接数
	blks_read_delta	数据块读取数
	cluster_active_sessions	活跃连接数
	cluster_connection_utilization	连接数使用率
	cluster_cpu_utilization	CPU使用率

指标分类	指标名称	指标说明
PolarDB	cluster_data_io	每秒存储引擎IO吞吐量
	cluster_data_iops	每秒存储引擎IO次数
	cluster_mem_hit_ratio	内存命中率
	cluster_memory_utilization	内存使用率
	cluster_qps	每秒查询数量
	cluster_slow_queries_ps	每秒慢查询数量
	cluster_tps	每秒事务数
	conn_usage	连接使用率
	cpu_total	CPU使用率
	db_age	数据库最大年龄
	instance_connection_utilization	实例连接数使用率
	instance_cpu_utilization	实例CPU使用率
	instance_input_bandwidth	实例输入带宽
	instance_memory_utilization	实例内存使用率
	instance_output_bandwidth	实例输出带宽
	mem_usage	内存利用率
	pls_data_size	pg数据盘大小
	pls_iops	pg IOPS
	pls_iops_read	pg读IOPS
	pls_iops_write	pg写IOPS
	pls_pg_wal_dir_size	pg WAL日志大小
	pls_throughput	pg IO吞吐
	pls_throughput_read	pg读IO吞吐
	pls_throughput_write	pg写IO吞吐
	swell_time	pg膨胀点
	tps	pg TPS

指标分类	指标名称	指标说明
	cluster_iops	每秒IO次数
Redis	intranet_in_ratio	写入带宽使用率
	intranet_out_ratio	读取带宽使用率
	failed_count	操作失败数
	cpu_usage	CPU使用率
	used_memory	内存使用量
	used_connection	已用连接数
	used_qps	已用QPS数量

Prometheus监控支持的消息队列RocketMQ版云服务的基础指标如下表所示。

指标分类	指标名称	指标说明
生产者	rocketmq_producer_requests	发送相关API调用次数
	rocketmq_producer_messages	发送消息量
	rocketmq_producer_message_size_bytes	发送消息的总大小
	rocketmq_producer_send_success_rate	发送消息成功率
	rocketmq_producer_failure_api_calls	发送API调用失败次数
	rocketmq_producer_send_rt_milliseconds_avg	发送消息耗时平均值
	rocketmq_producer_send_rt_milliseconds_min	发送消息耗时最小值
	rocketmq_producer_send_rt_milliseconds_max	发送消息耗时最大值
	rocketmq_producer_send_rt_milliseconds_p95	发送消息耗时P95值
	rocketmq_producer_send_rt_milliseconds_p99	发送消息耗时P99值
	rocketmq_consumer_requests	消费消息相关API调用次数
	rocketmq_consumer_send_back_requests	消费者消费失败回发接口调用次数
	rocketmq_consumer_send_back_messages	消费者消费失败回发的消息
	rocketmq_consumer_messages	消费消息量
	rocketmq_consumer_message_size_bytes	消费消息量大小（一分钟累积量）

指标分类	指标名称	指标说明
消费者	rocketmq_consumer_ready_and_inflight_messages	消息消费滞后量（包括已就绪消息量和处理中消息量）
	rocketmq_consumer_ready_messages	已就绪消息量
	rocketmq_consumer_inflight_messages	处理中消息量
	rocketmq_consumer_queue_time_milliseconds	消息排队时间
	rocketmq_consumer_message_await_time_milliseconds_avg	消息在消费者客户端等待处理资源耗时平均值
	rocketmq_consumer_message_await_time_milliseconds_min	消息在消费者客户端等待处理资源耗时最小值
	rocketmq_consumer_message_await_time_milliseconds_max	消息在消费者客户端等待处理资源耗时最大值
	rocketmq_consumer_message_await_time_milliseconds_p95	消息在消费者客户端等待处理资源耗时P95值
	rocketmq_consumer_message_await_time_milliseconds_p99	消息在消费者客户端等待处理资源耗时P99值
	rocketmq_consumer_message_process_time_milliseconds_avg	消费者处理消息耗时平均值
	rocketmq_consumer_message_process_time_milliseconds_min	消费者处理消息耗时最小值
	rocketmq_consumer_message_process_time_milliseconds_max	消费者处理消息耗时最大值
	rocketmq_consumer_message_process_time_milliseconds_p95	消费者处理消息耗时P95值
	rocketmq_consumer_message_process_time_milliseconds_p99	消费者处理消息耗时P99值
	rocketmq_consumer_consume_success_rate	消费消息成功率
	rocketmq_consumer_failure_api_calls	消费API调用失败次数
	rocketmq_consumer_to_dlq_messages	进死信消息量

相关文档

- 配置废弃指标
-

15.2. 报警规则说明

Prometheus监控报警规则包括ARMS报警规则、K8s报警规则、MongoDB报警规则、MySQL报警规则、Nginx报警规则、Redis报警规则。

ARMS报警规则

报警名称	表达式	采集数据时间（分钟）	报警触发条件
PodCpu75	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, "pod_name", "$1", "pod", "(.*)")) by (pod_name))>75	7	Pod的CPU使用率大于75%。
PodMemory75	100 * (sum(container_memory_working_set_bytes) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_memory_bytes, "pod_name", "$1", "pod", "(.*)")) by (pod_name))>75	5	Pod的内存使用率大于75%。
pod_status_no_running	sum (kube_pod_status_phase{phase!="Running"}) by (pod,phase)	5	Pod的状态为未运行。
PodMem4GbRestart	(sum (container_memory_working_set_bytes{id!="/"})by (pod_name,container_name) /1024/1024/1024)>4	5	Pod的内存大于4GB。
PodRestart	sum (increase (kube_pod_container_status_restarts_total{}[2m])) by (namespace,pod) >0	5	Pod重启。

K8s报警规则

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeStateMetricsListErrors	(sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m]))) > 0.01	15	Metric List出错。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeStateMetricsWatchErrors	(sum(rate(kube_state_metrics_watch_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_watch_total{job="kube-state-metrics"}[5m]))) > 0.01	15	Metric Watch出错。
NodeFilesystemAlmostOutOfSpace	(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)	60	Node文件系统即将无空间。
NodeFilesystemSpaceFillingUp	(node_filesystem_avail_bytes{job="node-exporter",fstype!=""} / node_filesystem_size_bytes{job="node-exporter",fstype!=""} * 100 < 40 and predict_linear(node_filesystem_avail_bytes{job="node-exporter",fstype!=""}[6h], 24*60*60) < 0 and node_filesystem_readonly{job="node-exporter",fstype!=""} == 0)	60	Node文件系统空间即将占满。
NodeFilesystemFilesFillingUp	(node_filesystem_files_free{job="node-exporter",fstype!=""} / node_filesystem_files{job="node-exporter",fstype!=""} * 100 < 40 and predict_linear(node_filesystem_files_free{job="node-exporter",fstype!=""}[6h], 24*60*60) < 0 and node_filesystem_readonly{job="node-exporter",fstype!=""} == 0)	60	Node文件系统文件即将占满。
NodeFilesystemAlmostOutOfFiles	(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)	60	Node文件系统几乎无文件。
NodeNetworkReceiveErrs	increase(node_network_receive_errs_total[2m]) > 10	60	Node网络接收错误。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
NodeNetworkTransmitErrs	increase(node_network_transmit_errs_total[2m]) > 10	60	Node网络传输错误。
NodeHighNumberConntrackEntriesUsed	(node_nf_conntrack_entries / node_nf_conntrack_entries_limit) > 0.75	无	使用大量Conntrack条目。
NodeClockSkewDetected	(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)	10	出现时间偏差。
NodeClockNotSynchronising	min_over_time(node_timex_sync_status[5m]) == 0	10	出现时间不同步。
KubePodCrashLooping	rate(kube_pod_container_status_restarts_total{job="kube-state-metrics"}[15m]) * 60 * 5 > 0	15	出现循环崩溃。
KubePodNotReady	sum by (namespace, pod) (max by(namespace, pod) (kube_pod_status_phase{job="kube-state-metrics", phase=~"Pending|Unknown"}) * on(namespace, pod) group_left(owner_kind) max by(namespace, pod, owner_kind) (kube_pod_owner{owner_kind!="Job"})) > 0	15	Pod未准备好。
KubeDeploymentGenerationMismatch	kube_deployment_status_observed_generation{job="kube-state-metrics"} != kube_deployment_metadata_generation{job="kube-state-metrics"}	15	出现部署版本不匹配。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeDeploymentReplicasMismatch	(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)	15	出现部署副本不匹配。
KubeStatefulSetReplicasMismatch	(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)	15	状态集副本不匹配。
KubeStatefulSetGenerationMismatch	kube_statefulset_status_observed_generation{job="kube-state-metrics"} != kube_statefulset_metadata_generation{job="kube-state-metrics"}	15	状态集版本不匹配。
KubeStatefulSetUpdateNotRolledOut	max without (revision) (kube_statefulset_status_current_revision{job="kube-state-metrics"} unless kube_statefulset_status_update_revision{job="kube-state-metrics"}) * (kube_statefulset_replicas{job="kube-state-metrics"} != kube_statefulset_status_replicas_updated{job="kube-state-metrics"})	15	状态集更新未退出。
KubeDaemonSetRolloutStuck	kube_daemonset_status_number_ready{job="kube-state-metrics"} / kube_daemonset_status_desired_number_scheduled{job="kube-state-metrics"} < 1.00	15	DaemonSet退出回退。
KubeContainerWaiting	sum by (namespace, pod, container) (kube_pod_container_status_waiting_reason{job="kube-state-metrics"}) > 0	60	容器等待。
KubeDaemonSetNotScheduled	kube_daemonset_status_desired_number_scheduled{job="kube-state-metrics"} - kube_daemonset_status_current_number_scheduled{job="kube-state-metrics"} > 0	10	DaemonSet无计划。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeDaemonSetMisScheduled	kube_daemonset_status_number_misscheduled{job="kube-state-metrics"} > 0	15	Daemon缺失计划。
KubeCronJobRunning	time() - kube_cronjob_next_schedule_time{job="kube-state-metrics"} > 3600	60	若Cron任务完成时间大于1小。
KubeJobCompletion	kube_job_spec_completions{job="kube-state-metrics"} - kube_job_status_succeeded{job="kube-state-metrics"} > 0	60	任务完成。
KubeJobFailed	kube_job_failed{job="kube-state-metrics"} > 0	15	任务失败。
KubeHpaReplicasMismatch	(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	15	HPA副本不匹配。
KubeHpaMaxedOut	kube_hpa_status_current_replicas{job="kube-state-metrics"} == kube_hpa_spec_max_replicas{job="kube-state-metrics"}	15	HPA副本超过最大值。
KubeCPUOvercommit	sum(namespace:kube_pod_container_resource_requests_cpu_cores:sum{}) / sum(kube_node_status_allocatable_cpu_cores) > (count(kube_node_status_allocatable_cpu_cores)-1) / count(kube_node_status_allocatable_cpu_cores)	5	CPU过载。
KubeMemoryOvercommit	sum(namespace:kube_pod_container_resource_requests_memory_bytes:sum{}) / sum(kube_node_status_allocatable_memory_bytes) > (count(kube_node_status_allocatable_memory_bytes)-1) / count(kube_node_status_allocatable_memory_bytes)	5	存储过载。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeCPUQuotaOvercommit	sum(kube_resourcequota{job="kube-state-metrics", type="hard", resource="cpu"}) / sum(kube_node_status_allocatable_cpu_cores) > 1.5	5	CPU额度过载。
KubeMemoryQuotaOvercommit	sum(kube_resourcequota{job="kube-state-metrics", type="hard", resource="memory"}) / sum(kube_node_status_allocatable_memory_bytes{job="node-exporter"}) > 1.5	5	存储额度过载。
KubeQuotaExceeded	kube_resourcequota{job="kube-state-metrics", type="used"} / ignoring(instance, job, type) (kube_resourcequota{job="kube-state-metrics", type="hard"} > 0) > 0.90	15	若配额超过限制。
CPUThrottlingHigh	sum(increase(container_cpu_cfs_throttled_periods_total{container!="", }[5m])) by (container, pod, namespace) / sum(increase(container_cpu_cfs_periods_total{}[5m])) by (container, pod, namespace) > (25 / 100)	15	CPU过热。
KubePersistentVolumeFillingUp	kubelet_volume_stats_available_bytes{job="kubelet", metrics_path="/metrics"} / kubelet_volume_stats_capacity_bytes{job="kubelet", metrics_path="/metrics"} < 0.03	1	存储卷容量即将不足。
KubePersistentVolumeErrors	kube_persistentvolume_status_phase{phase=~"Failed|Pending",job="kube-state-metrics"} > 0	5	存储卷容量出错。
KubeVersionMismatch	count(count by (gitVersion) (label_replace(kubernetes_build_info{job!~"kube-dns|coredns"},"gitVersion","$1","gitVersion","(v[0-9]*.[0-9]*.[0-9]*).*"))) > 1	15	版本不匹配。
KubeClientErrors	(sum(rate(rest_client_requests_total{code=~"5.."}[5m])) by (instance, job) / sum(rate(rest_client_requests_total[5m])) by (instance, job)) > 0.01	15	客户端出错。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeAPIErrorBudgetBurn	sum(apiserver_request:burnrate1h) > (14.40 * 0.01000) and sum(apiserver_request:burnrate5m) > (14.40 * 0.01000)	2	API错误过多。
KubeAPILatencyHigh	(cluster:apiserver_request_duration_seconds:mean5m{job="apiserver"} > on (verb) group_left() (avg by (verb) (cluster:apiserver_request_duration_seconds:mean5m{job="apiserver"} >= 0) + 2*stddev by (verb) (cluster:apiserver_request_duration_seconds:mean5m{job="apiserver"} >= 0))) > on (verb) group_left() 1.2 * avg by (verb) (cluster:apiserver_request_duration_seconds:mean5m{job="apiserver"} >= 0) and on (verb,resource) cluster_quantile:apiserver_request_duration_seconds:histogram_quantile{job="apiserver",quantile="0.99"} > 1	5	API延迟过高。
KubeAPIErrorsHigh	sum(rate(apiserver_request_total{job="apiserver",code=~"5.."}[5m])) by (resource,subresource,verb) / sum(rate(apiserver_request_total{job="apiserver"}[5m])) by (resource,subresource,verb) > 0.05	10	API错误过多。
KubeClientCertificateExpiration	apiserver_client_certificate_expiration_seconds_count{job="apiserver"} > 0 and on(job) histogram_quantile(0.01, sum by (job, le) (rate(apiserver_client_certificate_expiration_seconds_bucket{job="apiserver"}[5m]))) < 604800	无	客户端认证过期。
AggregatedAPIErrors	sum by(name, namespace) (increase(aggregator_unavailable_apiservice_count[5m])) > 2	无	聚合API出错。
AggregatedAPIDown	sum by(name, namespace) (sum_over_time(aggregator_unavailable_apiservice[5m])) > 0	5	聚合API下线。
KubeAPIDown	absent(up{job="apiserver"} == 1)	15	API下线。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
KubeNodeNotReady	kube_node_status_condition{job="kube-state-metrics",condition="Ready",status="true"} == 0	15	Node未准备好。
KubeNodeUnreachable	kube_node_spec_taint{job="kube-state-metrics",key="node.kubernetes.io/unreachable",effect="NoSchedule"} == 1	2	Node无法获取。
KubeletTooManyPods	max(max(kubelet_running_pod_count{job="kubelet", metrics_path="/metrics"}) by(instance) * on(instance) group_left(node) kubelet_node_name{job="kubelet", metrics_path="/metrics"}) by(node) / max(kube_node_status_capacity_pods{job="kube-state-metrics"} != 1) by(node) > 0.95	15	Pod过多。
KubeNodeReadinessFlapping	sum(changes(kube_node_status_condition{status="true",condition="Ready"}[15m])) by (node) > 2	15	准备状态变更次数过多。
KubeletPlegDurationHigh	node_quantile:kubelet_pleg_relist_duration_seconds:histogram_quantile{quantile="0.99"} >= 10	5	PLEG持续时间过长。
KubeletPodStartUpLatencyHigh	histogram_quantile(0.99, sum(rate(kubelet_pod_worker_duration_seconds_bucket{job="kubelet", metrics_path="/metrics"}[5m])) by (instance, le)) * on(instance) group_left(node) kubelet_node_name{job="kubelet", metrics_path="/metrics"} > 60	15	Pod启动延迟过高。
KubeletDown	absent(up{job="kubelet", metrics_path="/metrics"} == 1)	15	Kubelet下线。
KubeSchedulerDown	absent(up{job="kube-scheduler"} == 1)	15	Kubelet日程下线。
KubeControllerManagerDown	absent(up{job="kube-controller-manager"} == 1)	15	Controller Manager下线。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
TargetDown	100 * (count(up == 0) BY (job, namespace, service) / count(up) BY (job, namespace, service)) > 10	10	目标下线。
NodeNetworkInterfaceFlapping	changes(node_network_up{job="node-exporter",device!~"veth.+"}[2m]) > 2	2	网络接口状态变更过频繁。

MongoDB报警规则

报警名称	表达式	采集数据时间（分钟）	报警触发条件
MongodbReplicationLag	avg(mongodb_replset_member_optime_date{state="PRIMARY"}) - avg(mongodb_replset_member_optime_date{state="SECONDARY"}) > 10	5	复制延迟过长。
MongodbReplicationHeadroom	(avg(mongodb_replset_oplog_tail_timestamp - mongodb_replset_oplog_head_timestamp) - (avg(mongodb_replset_member_optime_date{state="PRIMARY"}) - avg(mongodb_replset_member_optime_date{state="SECONDARY"}))) <= 0	5	复制余量不足。
MongodbReplicationStatus3	mongodb_replset_member_state == 3	5	复制状态为3。
MongodbReplicationStatus6	mongodb_replset_member_state == 6	5	复制状态为6。
MongodbReplicationStatus8	mongodb_replset_member_state == 8	5	复制状态为8。
MongodbReplicationStatus10	mongodb_replset_member_state == 10	5	复制状态为10。
MongodbNumberCursorsOpen	mongodb_metrics_cursor_open{state="total_open"} > 10000	5	打开数字光标数量过多。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
MongodbCursorsTimeouts	sum (increase increase(mongodb_metrics_cursor_timed_out_total[10m]) > 100	5	若光标超。
MongodbTooManyConnections	mongodb_connections{state="current"} > 500	5	连接过多。
MongodbVirtualMemoryUsage	(sum(mongodb_memory{type="virtual"}) BY (ip) / sum(mongodb_memory{type="mapped"}) BY (ip)) > 3	5	虚拟内存使用率过高。

MySQL报警规则

报警名称	表达式	采集数据时间（分钟）	报警触发条件
MySQL is down	mysql_up == 0	1	MySQL下线。
open files high	mysql_global_status_innodb_num_open_files > (mysql_global_variables_open_files_limit) * 0.75	1	打开文件数量偏高。
Read buffer size is bigger than max. allowed packet size	mysql_global_variables_read_buffer_size > mysql_global_variables_slave_max_allowed_packet	1	读取缓存区超过数据包最大限制。
Sort buffer possibly missconfigured	mysql_global_variables_innodb_sort_buffer_size <256*1024 or mysql_global_variables_read_buffer_size > 4*1024*1024	1	排序缓冲区可能存在配置错误。
Thread stack size is too small	mysql_global_variables_thread_stack <196608	1	线程堆栈太小。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
Used more than 80% of max connections limited	mysql_global_status_max_used_connections > mysql_global_variables_max_connections * 0.8	1	使用超过80%连接限制。
InnoDB Force Recovery is enabled	mysql_global_variables_innodb_force_recovery != 0	1	启用强制恢复。
InnoDB Log File size is too small	mysql_global_variables_innodb_log_file_size < 16777216	1	日志文件过小。
InnoDB Flush Log at Transaction Commit	mysql_global_variables_innodb_flush_log_at_trx_commit != 1	1	在事务提交时刷新日志。
Table definition cache too small	mysql_global_status_open_table_definitions > mysql_global_variables_table_definition_cache	1	表定义缓存过小。
Table open cache too small	mysql_global_status_open_tables >mysql_global_variables_table_open_cache * 99/100	1	表打开缓存过小。
Thread stack size is possibly too small	mysql_global_variables_thread_stack < 262144	1	线程堆栈可能过小。
InnoDB Buffer Pool Instances is too small	mysql_global_variables_innodb_buffer_pool_instances == 1	1	缓冲池实例过小。
InnoDB Plugin is enabled	mysql_global_variables_ignore_builtin_innodb == 1	1	插件启用。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
Binary Log is disabled	mysql_global_variables_log_bin != 1	1	二进制日志禁用。
Binlog Cache size too small	mysql_global_variables_binlog_cache_size < 1048576	1	缓存过小。
Binlog Statement Cache size too small	mysql_global_variables_binlog_stmt_cache_size <1048576 and mysql_global_variables_binlog_stmt_cache_size > 0	1	声明缓存过小。
Binlog Transaction Cache size too small	mysql_global_variables_binlog_cache_size <1048576	1	交易缓存过小。
Sync Binlog is enabled	mysql_global_variables_sync_binlog == 1	1	二进制日志启用。
IO thread stopped	mysql_slave_status_slave_io_running != 1	1	IO线程停止。
SQL thread stopped	mysql_slave_status_slave_sql_running == 0	1	SQL线程停止。
Mysql_Too_Many_Connections	rate(mysql_global_status_threads_connected[5m])>200	5	连接过多。
Mysql_Too_Many_slow_queries	rate(mysql_global_status_slow_queries[5m])>3	5	慢查询过多。
Slave lagging behind Master	rate(mysql_slave_status_seconds_behind_master[1m]) >30	1	从机表现落后于主机。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
Slave is NOT read only(Please ignore this warning indicator.)	mysql_global_variables_read_only != 0	1	从机权限不是只读。

Nginx报警规则

报警名称	表达式	采集数据时间（分钟）	报警触发条件
NginxHighHttp4xxErrorRate	sum(rate(nginx_http_requests_total{status=~"^4.."}[1m])) / sum(rate(nginx_http_requests_total[1m])) * 100 > 5	5	HTTP 4xx错误率过高。
NginxHighHttp5xxErrorRate	sum(rate(nginx_http_requests_total{status=~"^5.."}[1m])) / sum(rate(nginx_http_requests_total[1m])) * 100 > 5	5	HTTP 5xx错误率过高。
NginxLatencyHigh	histogram_quantile(0.99, sum(rate(nginx_http_request_duration_seconds_bucket[30m])) by (host, node)) > 10	5	延迟过高。

Redis报警规则

报警名称	表达式	采集数据时间（分钟）	报警触发条件
RedisDown	redis_up == 0	5	Redis下线。
RedisMissingMaster	count(redis_instance_info{role="master"}) == 0	5	Master缺失。
RedisTooManyMasters	count(redis_instance_info{role="master"}) > 1	5	Master过多。
RedisDisconnectedSlaves	count without (instance, job) (redis_connected_slaves) - sum without (instance, job) (redis_connected_slaves) - 1 > 1	5	Slave连接断开。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
RedisReplicationBroken	delta(redis_connected_slaves[1m]) < 0	5	复制中断。
RedisClusterFlapping	changes(redis_connected_slaves[5m]) > 2	5	副本连接识别变更。
RedisMissingBackup	time() - redis_rdb_last_save_timestamp_seconds > 60 * 60 * 24	5	备份中断。
RedisOutOfMemory	redis_memory_used_bytes / redis_total_system_memory_bytes * 100 > 90	5	内存不足。
RedisTooManyConnections	redis_connected_clients > 100	5	连接过多。
RedisNotEnoughConnections	redis_connected_clients < 5	5	连接不足。
RedisRejectedConnections	increase(redis_rejected_connections_total[1m]) > 0	5	连接被拒绝。

15.3. Helm命令参数说明

Prometheus支持通过Operator方式安装K8s集群，本文介绍Operator安装过程中的Helm命令的各参数说明。

使用Operator方式安装K8s集群的Helm示例命令如下：

```
helm install arms-prom-operator aliyun/ack-arms-prometheus \
                --namespace arms-prom \
                --set controller.cluster_id=123 \
                --set controller.uid="456" \
                --set controller.region_id=eu-central-1 \
                --set controller.vpc_prefix=registry.
```

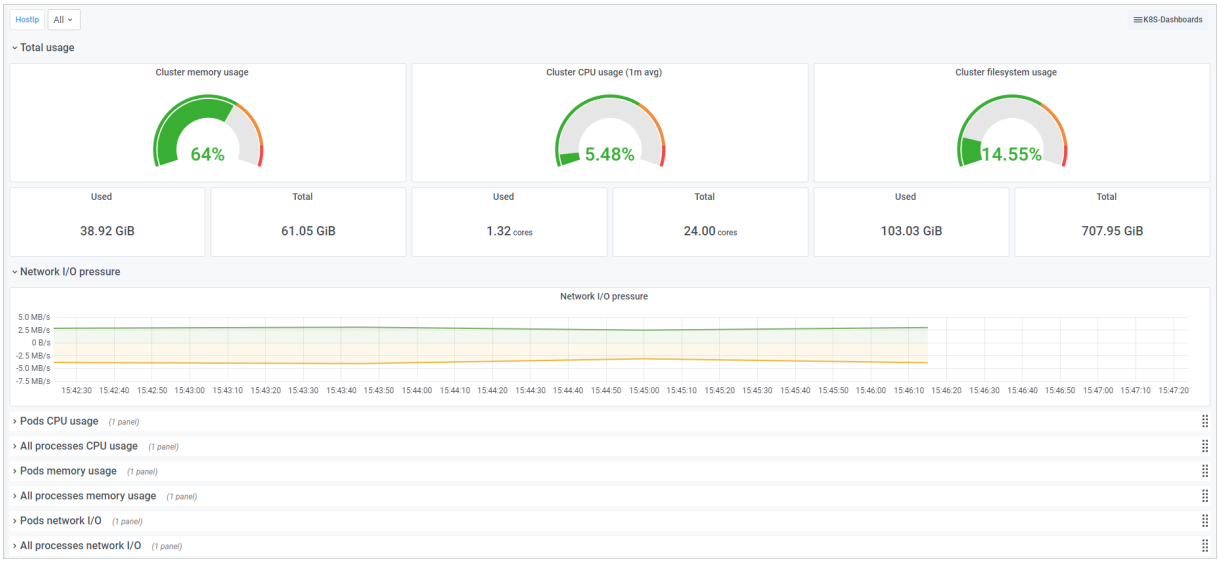
参数	说明
namespace	Helm目标命名空间。
controller.cluster_id	当前K8s所在集群的Cluster ID。
controller.uid	阿里云账号的User ID。

参数	说明
controller.region_id	当前K8s所在集群的Region ID。
controller.vpc_prefix	选择镜像拉取的网络环境。 • `--set controller.vpc_prefix=registry-vpc.`：从阿里云内网拉取镜像。阿里云Prometheus默认从阿里云内网拉取镜像，如果您的镜像存储在阿里云内网，则可以不用配置此参数。 • `--set controller.vpc_prefix=registry.`：从公网拉取镜像。

15.4. 基础大盘说明

Prometheus提供了多种开箱即用的预置监控大盘，您可以通过这些大盘查看丰富的Prometheus监控指标，并按需更改大盘数据的时间区间、刷新频率等属性。本文主要说明Kubernetes Overview、Deployment、Pod和Node Details基础大盘，并简单介绍下Grafana的常用操作。

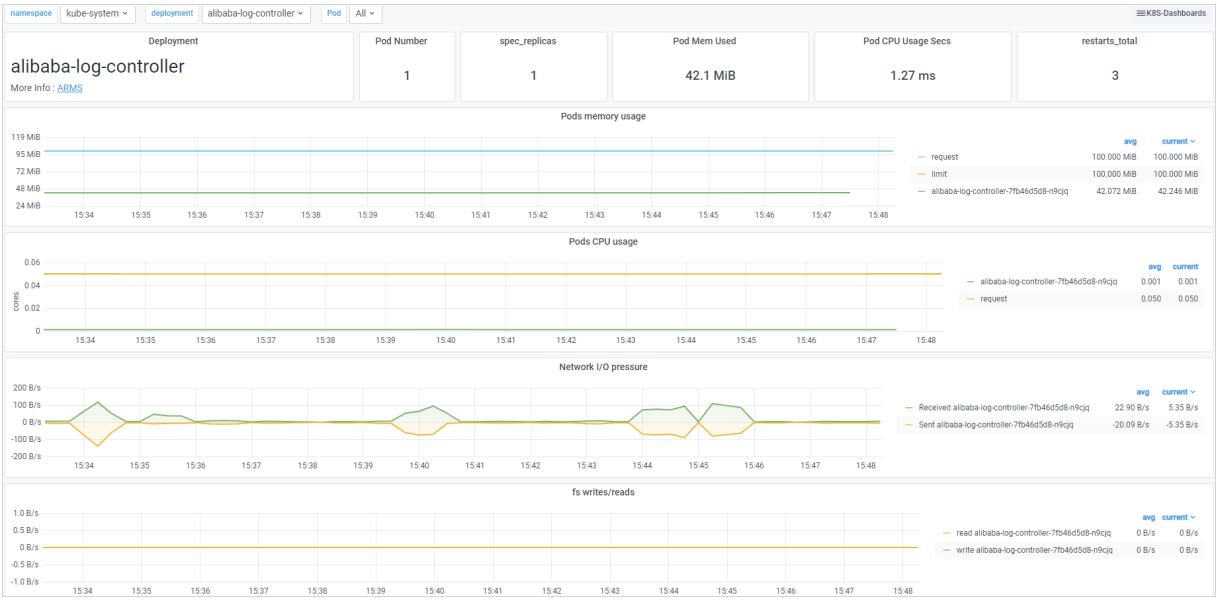
Kubernetes Overview大盘



该大盘展示的监控指标主要包括：

- 总体使用量信息：例如集群CPU使用率、集群内存使用率、集群文件系统使用率。
- CPU信息：例如Pod CPU使用率、全部进程CPU使用率。
- 内存信息：例如Pod内存使用率、全部进程内存使用率。
- 网络信息：例如网络I/O压力、Pod网络I/O、所有进程网络I/O。

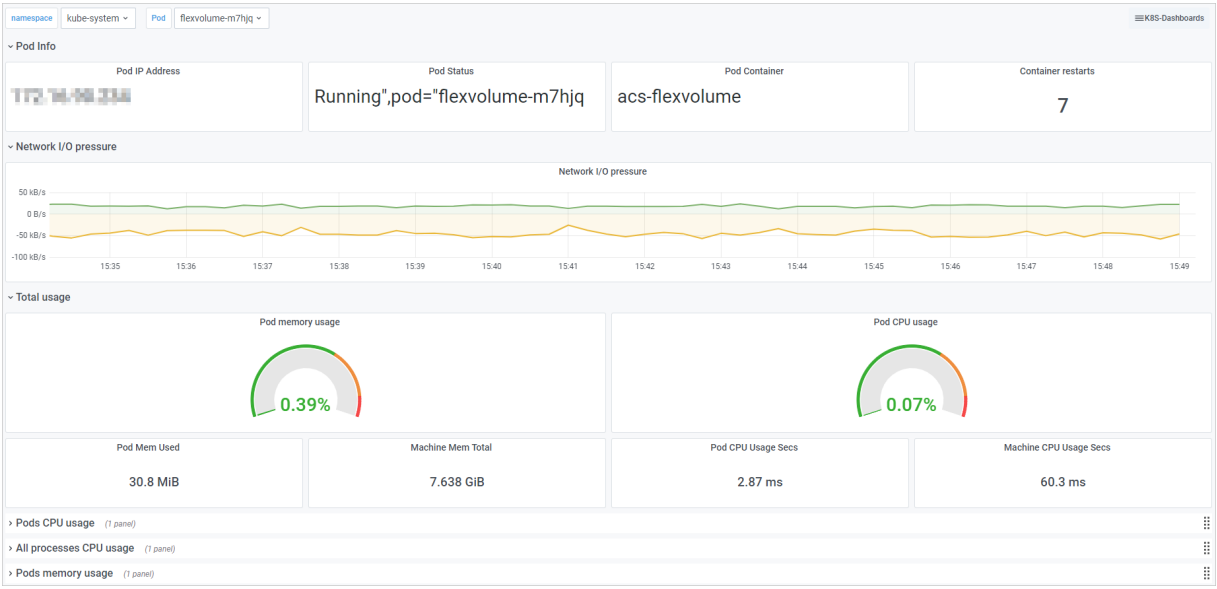
Deployment大盘



该大盘展示的监控指标主要包括：

- 概览：Pod数量、副本数量、部署CPU使用率、部署内存使用量、重启次数。
- 详情：CPU使用率、内存使用率、全部进程网络I/O、FS读写情况。

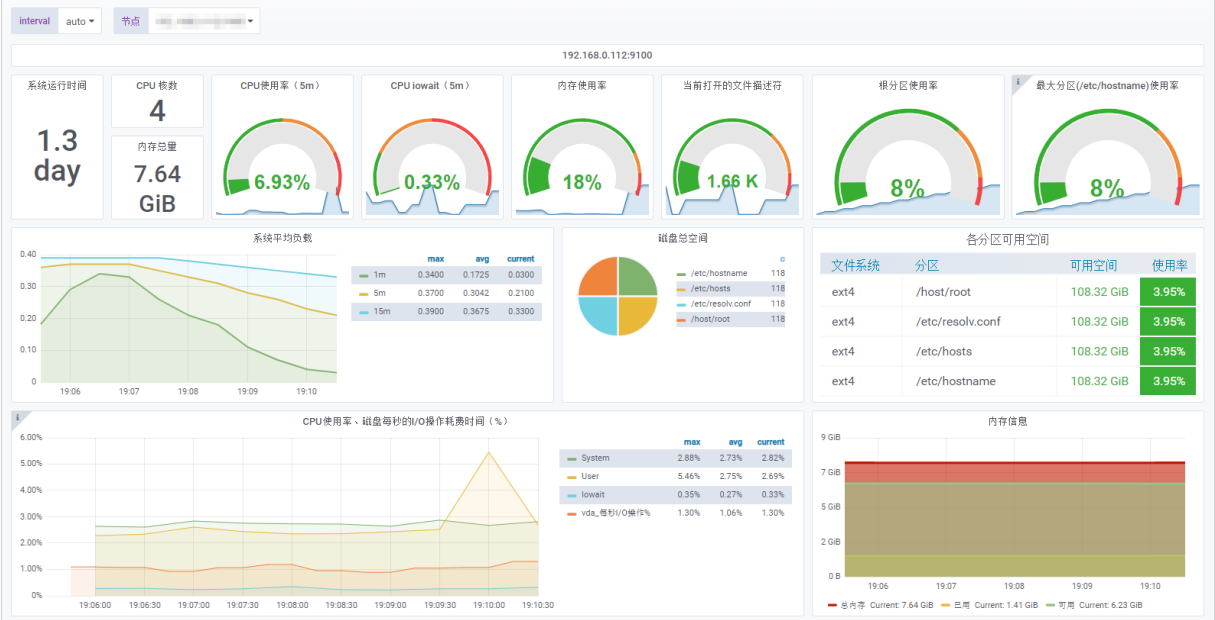
Pod大盘



该大盘展示的监控指标主要包括：

- Pod基本信息：Pod IP地址、Pod状态、Pod容器、容器重启次数。
- 总体使用量信息：例如Pod CPU使用率、Pod内存使用率。
- CPU信息：例如Pod CPU使用率、全部进程CPU使用率。
- 内存信息：例如Pod内存使用率、全部进程内存使用率。
- 网络信息：例如网络I/O压力、Pod网络I/O、所有进程网络I/O。

Node Details大盘



该大盘展示的监控指标主要包括：

- CPU信息：例如CPU核数、CPU使用率、CPU I/O等待时间。
- 内存信息：例如内存总量、内存使用率。
- 磁盘信息：例如磁盘总空间、磁盘IO读写时间、磁盘读写速率。
- 网络信息：例如网络流量、TCP连接情况。

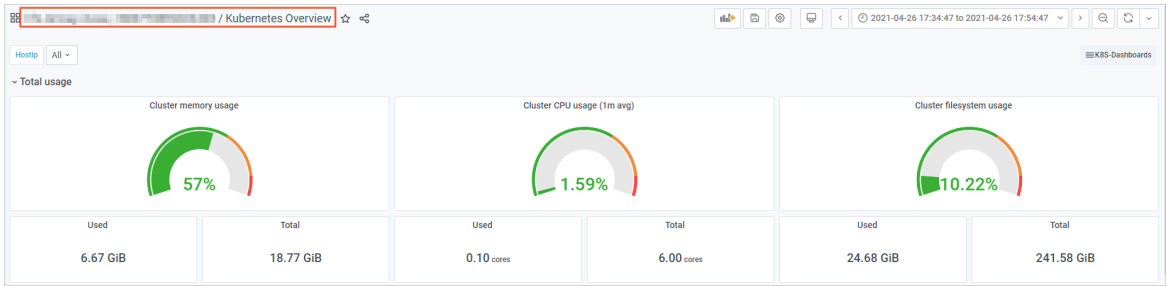
大盘常用操作

> **说明** 本文只介绍了Grafana中常用的基本操作，更详细的使用方法，请参见Grafana文档。

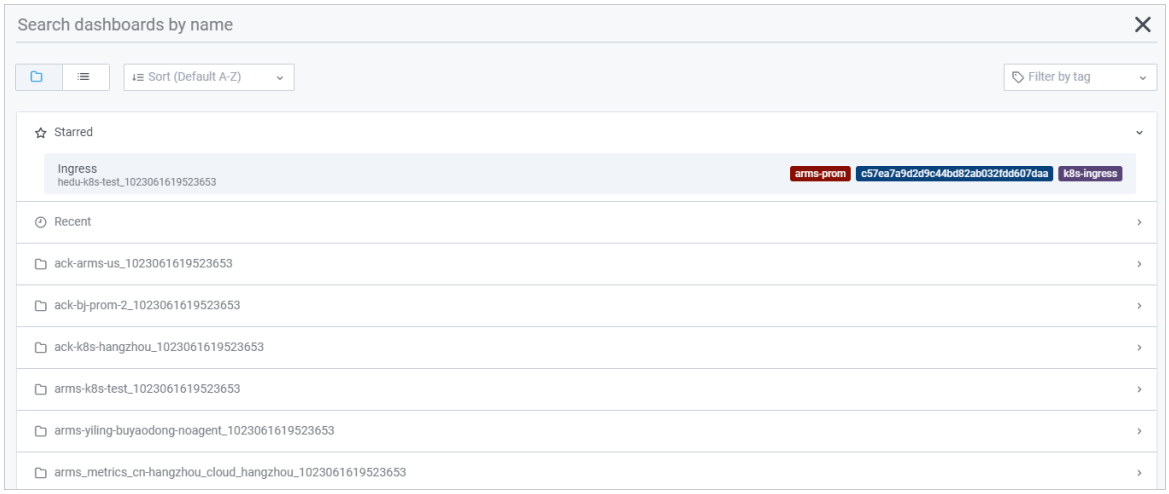
对大盘的常用操作如下所示：

切换大盘

1. 在Grafana页面顶部，单击大盘名称。



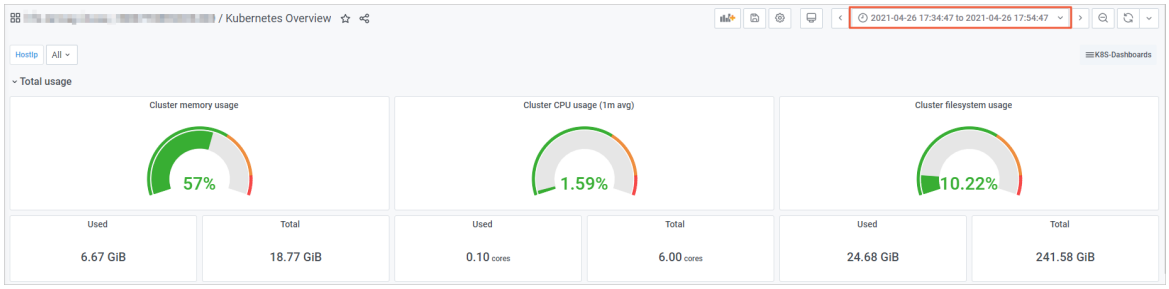
2. 在顶部搜索栏中输入大盘名称来，或者单击右上角的Filter by tag下拉菜单筛选出带有指定Tag（标签）的大盘。



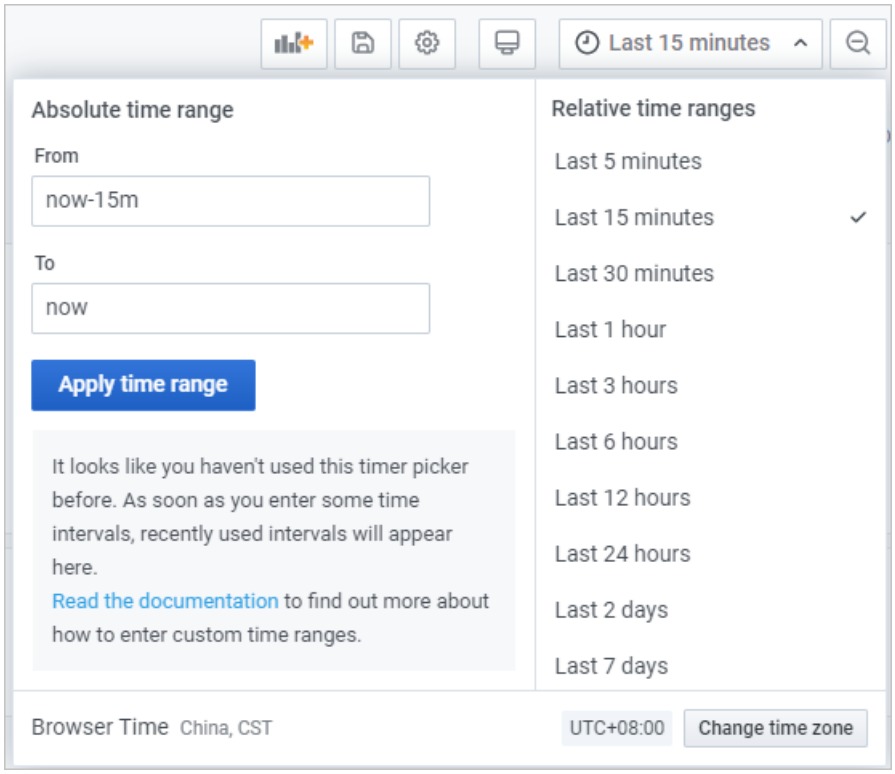
3. 单击需要查看的大盘名称。

设置时间区间和刷新频率

1. 在Grafana页面右上角，单击时间选择框。



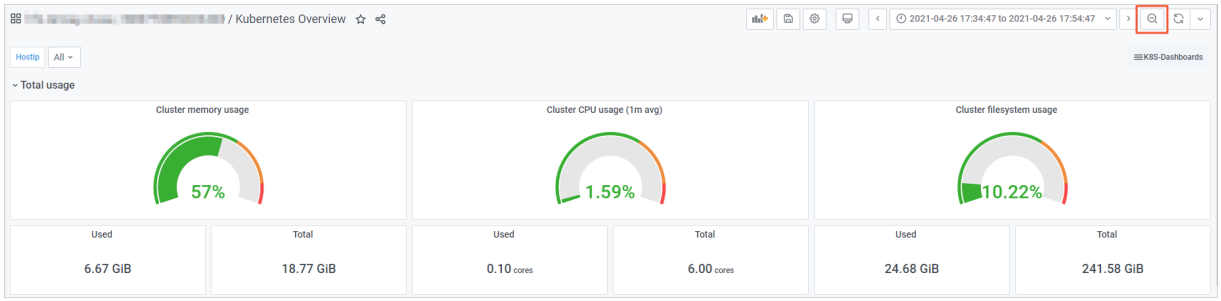
2. 在弹出的对话框中选择预定义的监控数据相对时间区间，例如过去5分钟、过去12小时、过去30天等，也可以通过设置时间起点和终点来设置自定义的绝对时间区间。



扩大时间区间

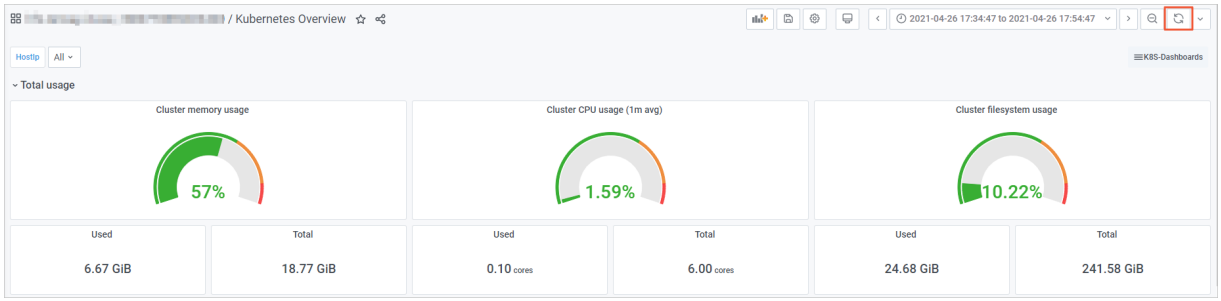
在Grafana页面右上角，单击 图标。

每单击一次该扩大按钮，时间区间就会扩大为当前的两倍，且时间起点迁移和终点后移的幅度相等。例如，假设当前选择的时间区间为过去10分钟，则单击一次该扩大按钮后，时间区间的前面和后面将会各延长5分钟。



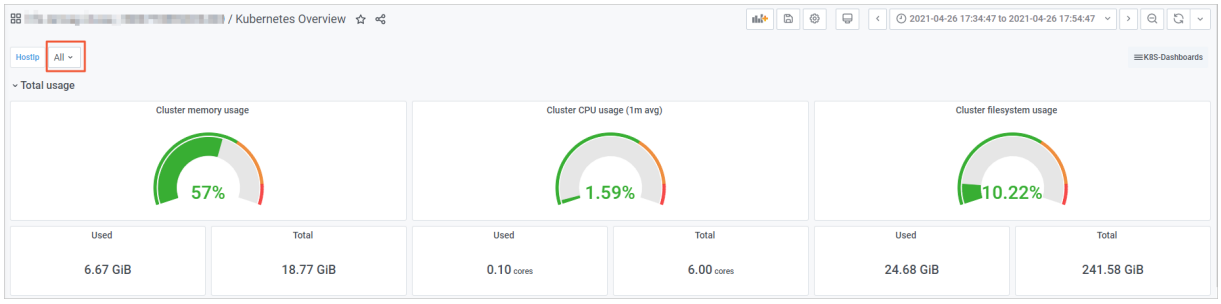
手动刷新

在Grafana页面右上角，单击 图标，刷新当前大盘中所有面板的监控数据。



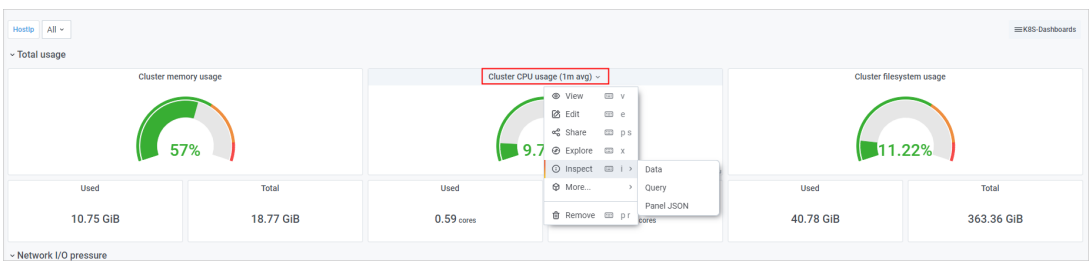
筛选监控数据

在Grafana页面，选择此下拉菜单中的选项即可筛选当前大盘显示的监控数据。



大盘面板常用操作

单击面板顶部的面板名称后，可进行以下操作：



- 全屏查看当前面板：单击**View**，或按快捷键V。再次按快捷键V或Esc即可退出全屏模式。
- 编辑当前面板：单击**Edit**，或按快捷键E。在**Edit Panel**页面修改当前面板的信息，然后单击右上角的**Apply**。
- 分享当前面板：单击**Share**，或依次按下P和S打开分享对话框，获得当前面板的分享链接、嵌入链接或快照链接。
- 探索当前面板：单击**Explore**，或按快捷键X。在**Explore**页面检查指标、排查故障或探索数据。
- 将当前面板的数据导出为CSV文件：选择**Inspect > Data**，然后在**Data**页签中单击**Download CSV**导出数据。
- 查看当前面板的查询指标：选择**Inspect > Query**，然后在**Query**页签中查看请求和响应。
- 获得当前面板的JSON代码：选择**Inspect > Panel JSON**，然后在**JSON**页签中拷贝JSON代码。
- 拷贝并粘贴当前面板：选择**More > Duplicate**，或依次按下P和D即可拷贝当前面板并自动粘贴至当前仪表盘中。
- 拷贝当前面板：选择**More > Copy**，可以拷贝当前面板。

15.5. Helm版本说明

本文主要介绍阿里云Prometheus监控的监控组件Helm的版本发布说明。

Helm版本号	Agent镜像版本号	发布时间	功能概述
v1.1.5	arms-prom-operator:v3.1.0	2022年05月	• 支持集成中心。 • 支持超大规模集群（>1万节点）。 • 支持设置非Prometheus监控控制台创建的ServiceMonitor和PodMonitor同步。 • 支持配置非Prometheus监控控制台创建的ServiceMonitor与PodMonitor声明式服务发现。 `Annotations: arms.prometheus.io/discovery=true|false` • 支持Agent HPA副本数上限可参数化配置。 • 支持编辑Prometheus监控基础指标Job部分字段。 • 支持在线校验ServiceMontor、PodMonitor及Prometheus.yaml相关配置文件。 • 优化CPU、内存资源使用与系统稳定性。
v1.1.0	arms-prom-operator:v3.0.0	2021年10月	• 支持PodMonitor。 • 支持自定义Namespace。 • CMS支持GPU数据。 • 支持Agent日志在线化。

Helm版本号	Agent镜像版本号	发布时间	功能概述
v1.0.0	arms-prom-operator:v3.0.0	2021年09月	• 按目标抓取量调度。 • 按量HPA能力。 • 性能优化。 • Bug修复：修复CMS磁盘使用率数据为0的问题。
v0.1.8	arms-prom-operator:v0.1	2021年07月	• 优化升级能力。 • 优化自建K8s公网接入。 • Bug修复：修复云服务Region标签不准确问题。
v0.1.5	arms-prom-operator:v0.1	2020年10月	• 支持阿里云容器服务Kubernetes版v1.18集群。 • 支持镜像Region从内网地址拉取。
v0.1.4	arms-prom-operator:v0.1	2020年07月	• 开箱即用的K8s容器监控，包括Pod监控、Node监控和Resource监控等，主要用于监控应用所在的K8s容器运行时。 • 白屏化的组件监控，包括MySQL、Redis、Kafka、ZooKeeper和Nginx等常见的9种组件监控，主要用于监控应用依赖中间件的场景。 • 全托管的Prometheus监控系统，包括Prometheus.yaml采集规则、Grafana大盘和告警系统，可以满足自建Prometheus迁移阿里云的需求场景。 • Bug修复：修复鉴权访问Bug。
v0.1.3	arms-prom-operator:v0.1	2020年04月	增加Agent资源使用限制。
v0.1.2	arms-prom-operator:v0.1	2019年08月	初始发布版本。

15.6. Remote Write和Remote Read地址使用说明

阿里云Prometheus监控提供了Remote Write和Remote Read两个标准接口，您可以通过该接口远程存储Prometheus监控数据。本文以开源Prometheus将监控数据写入阿里云Prometheus监控服务为例介绍如何使用Remote Read地址和Remote Write地址。

前提条件

已创建Prometheus实例，具体操作，请参见：

- Prometheus实例 for 容器服务
- Prometheus实例 for Kubernetes
- Prometheus实例 for Remote Write

- Prometheus实例 for VPC
- Prometheus实例 for 云服务

（可选）

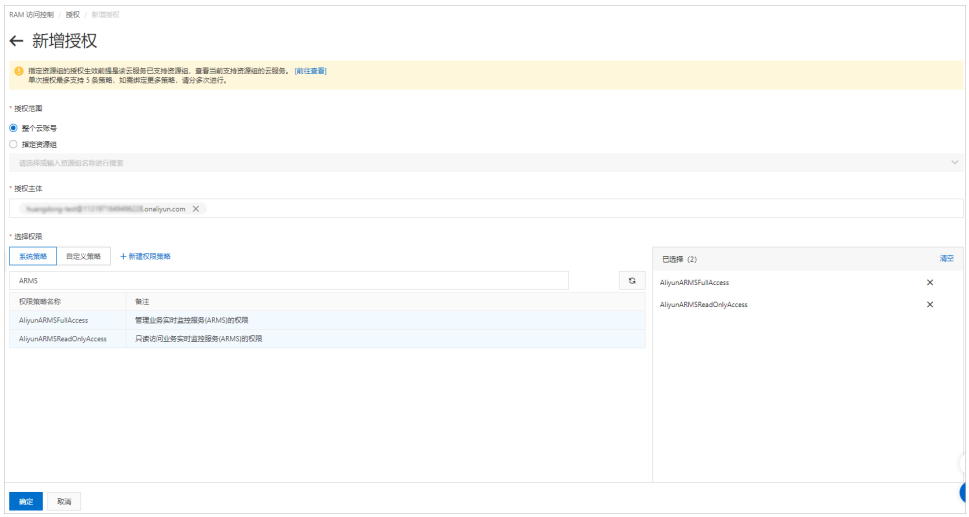
（可选）步骤一：为RAM用户（子账号）授予ARMS读写权限

如果您的阿里云Prometheus实例是由阿里云账号（主账号）创建，且您需要使用RAM用户（子账号）的AccessKey ID和AccessKey Secret进行远程读写，则需要先为RAM用户授予ARMS的读写权限。

1. 使用阿里云账号（主账号）登录RAM控制台。
2. 在左侧导航栏选择**权限管理 > 授权**。
3. 在**授权**页面单击**新增授权**。
4. 在**新增授权**页面，设置授权主体为需要被授权的RAM用户。
5. 在**选择权限**区域通过搜索查找ARMS的权限策略，单击权限策略名称将权限策略添加至右侧**已选择**区域，然后单击**确定**。

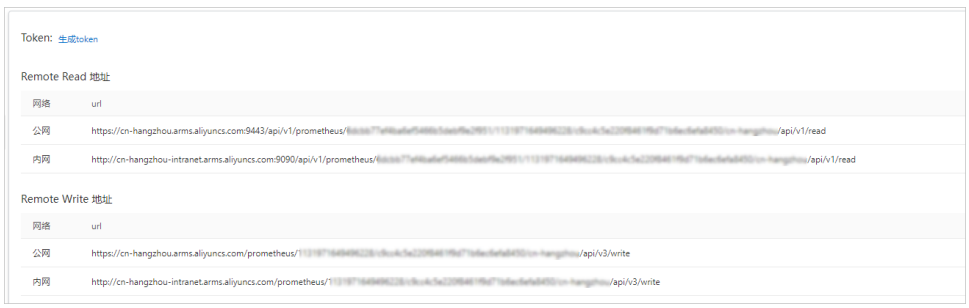
   需要添加以下权限策略：

   - AliyunARMSFullAccess
   - AliyunARMSReadOnlyAccess



步骤二：获取Remote Read和Remote Write地址

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面顶部选择Prometheus实例所在的地域，并在目标集群右侧的**操作**列单击**设置**。
3. 在右侧页面单击**设置**页签。
4. 在**设置**页签上，根据需求复制公网或内网的Remote Write和Remote Read地址。



步骤三：配置Prometheus

1. 安装Prometheus，安装方法请参见Prometheus官方文档。
2. 打开*Prometheus.yaml*配置文件，并在文件末尾增加以下内容，将 `remote_write` 和 `remote_read` 链接替换为步骤二中获取的地址，然后保存文件。

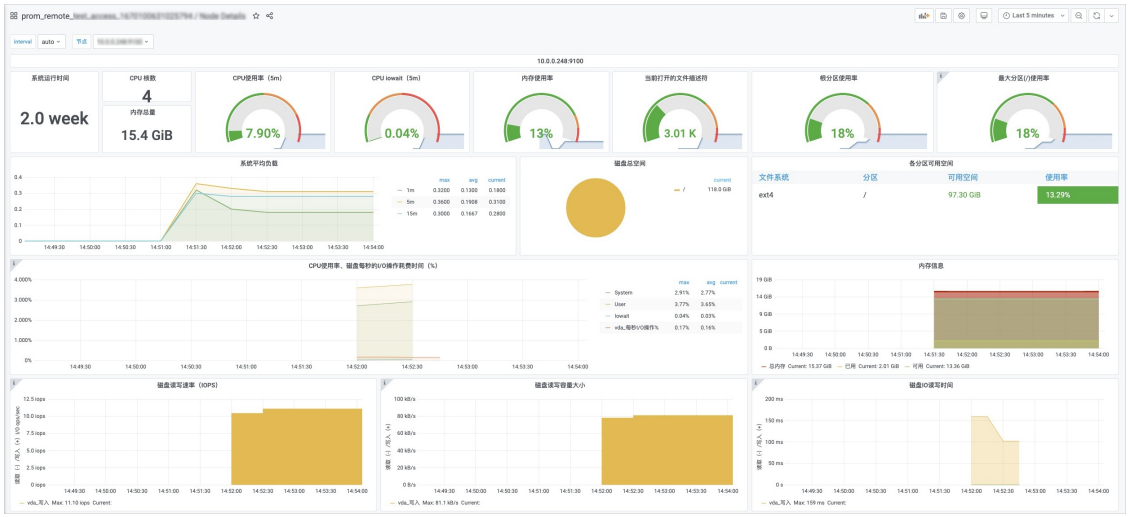
```
global:
  scrape_interval:     15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
remote_write:
  // 替换为您的Remote Write地址。
  - url: "http://ts-xxxxxxxxxxx.hitsdb.rds.aliyuncs.com:3242/api/prom_write"
    basic_auth:
      //username和password分别对应您阿里云账号的AccessKey ID和AccessKey Secret。
      username: access-key-id
      password: access-key-secret
remote_read:
  // 替换为您的Remote Read地址。
  - url: "http://ts-xxxxxxxxxxx.hitsdb.rds.aliyuncs.com:3242/api/prom_read"
    read_recent: true
```

> **说明**
>
> - 阿里云Prometheus监控提供了公网、内网两类Remote Read和Remote Write地址。其中公网、内网地址均需要配置Username、Password。
> - Username、Password分别对应您阿里云账号的AccessKey ID和AccessKey Secret。获取AccessKey的操作，请参见获取AccessKey。
> - 如果您的阿里云Prometheus实例是由阿里云账号（主账号）创建，且您需要使用RAM用户（子账号）的AccessKey ID和AccessKey Secret进行远程读写，则需要先为RAM用户授予ARMS的读写权限。具体操作，请参见步骤一。

查看开源Prometheus监控数据

运行开源Prometheus监控后，您可以在Grafana大盘上查看监控数据。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏选择地域，然后单击开源Prometheus写入的阿里云Prometheus实例名称。
3. 在**大盘列表**页面单击需要查看的大盘。



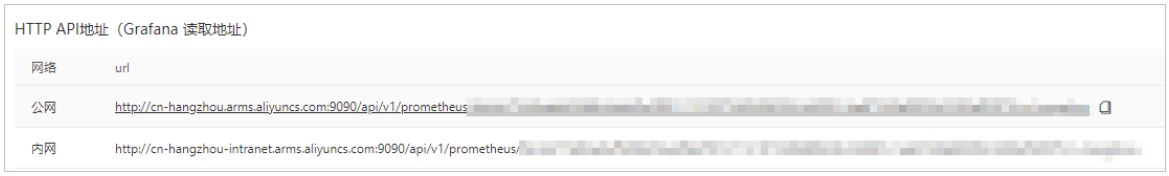
15.7. HTTP API地址使用说明

阿里云Prometheus监控提供了HTTP API地址，您可以通过该地址将阿里云Prometheus实例的监控数据接入自建的Grafana，也可以获取阿里云Prometheus监控数据进行二次开发。本文介绍如何通过HTTP API地址接入自建的Grafana和获取阿里云Prometheus监控数据。

获取HTTP API地址

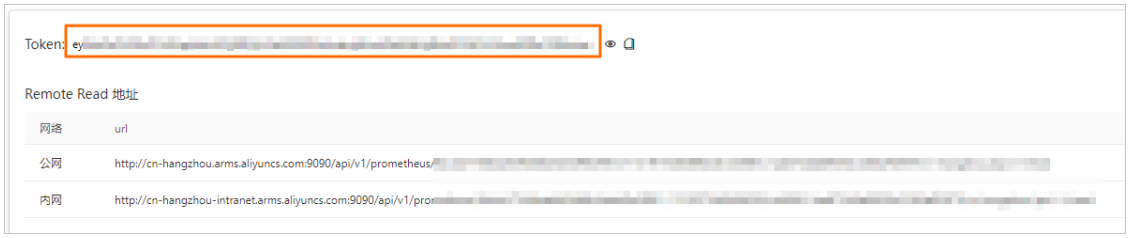
1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面顶部选择Prometheus实例所在的地域，并在目标集群右侧的**操作**列单击**设置**。
3. 在右侧页面单击**设置**页签。
4. 在**设置**页签上，根据需求复制公网或内网的HTTP API地址。

> **说明** 如果是云服务类型的Prometheus实例，请根据接入云服务的产品类型选择对应的HTTP API地址。



5. （可选）如果您需要提高Grafana数据读取的安全性，可以单击**生成token**，获取Prometheus实例的鉴权Token。

> **注意** 生成Token后，在Grafana中添加数据源时必须配置Token，否则无法读取Prometheus监控数据。

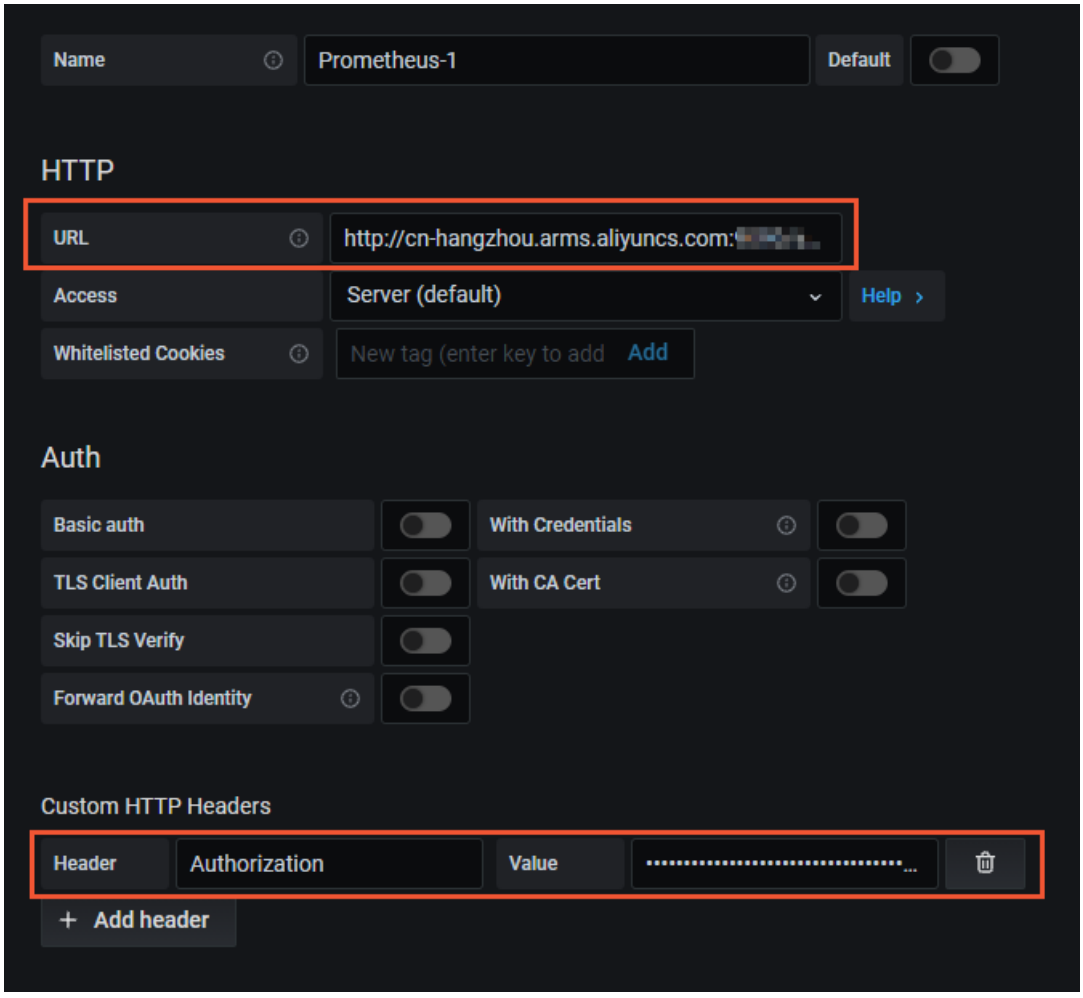


接入自建Grafana

1. 配置Grafana数据源。
   i. 以管理员账号登录本地Grafana系统。
   ii. 在左侧导航栏中选择Configuration > Data Sources。

   > 说明 仅管理员可以看到此菜单。

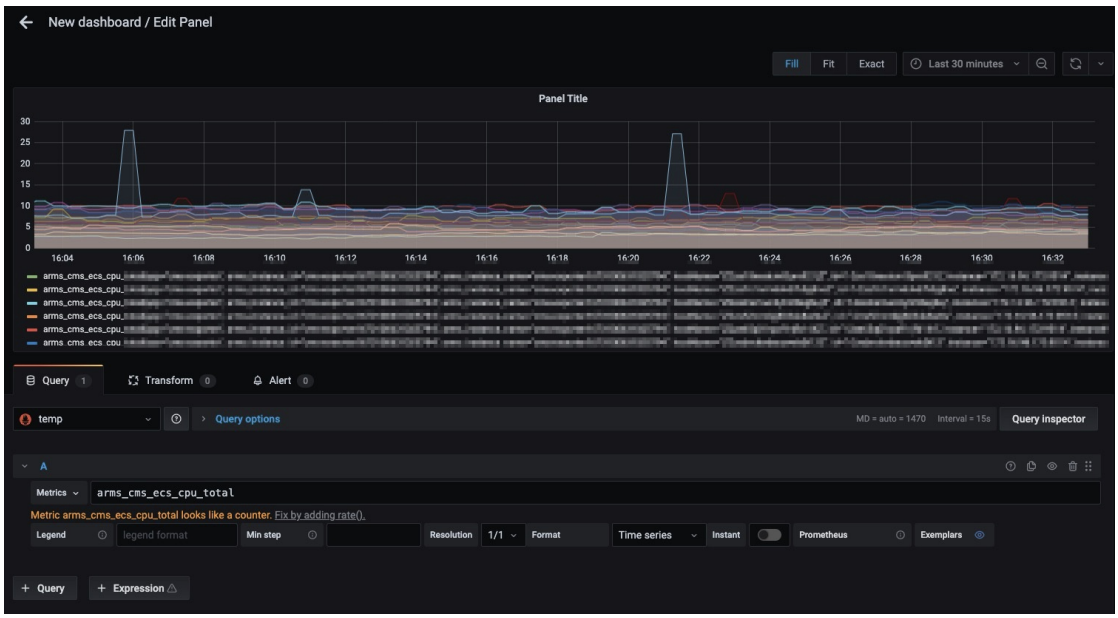
   iii. 在Data Sources页签上单击Add data source。
   iv. 在Add data source页面上单击Prometheus。
   v. 在Settings页签的Name字段输入自定义的名称，在URL字段粘贴获取HTTP API地址中获得的HTTP API地址。
   vi. （可选）在Custom HTTP Headers区域单击+Add header，设置Header为Authorization，设置Value为获取HTTP API地址中获取的鉴权Token。



   vii. 单击页签底部的Save & Test。

2. 验证结果。
   i. 登录本地Grafana系统。
   ii. 在左侧导航栏中选择+ > Create。
   iii. 在New dashboard页面单击Add an empty panel。

iv. 在Edit Panel页面的Query页签的下拉框中选择步骤1中添加的数据源，在A区域的Metrics字段输入指标名称并按回车。
如果能显示出相应指标的图表，则说明操作成功。否则请检查填写的接口地址或Token是否正确，以及数据源是否有Prometheus监控数据。



获取阿里云Prometheus监控数据

调用阿里云Prometheus监控数据的请求示例如下。更多使用HTTP API获取Prometheus监控数据的操作，请参见Prometheus HTTP API。

```
GET {HTTP API}/api/v1/query
Accept: application/json
Content-Type: application/json
Authorization: {Token}
{
  "query":"arms_prometheus_target_interval_length_seconds_sum",
  "time":"1635302655",
  "timeout":"1000"
}
```

> **说明** `{HTTP API}` 和 `{Token}` 请替换为获取HTTP API地址中获取的HTTP API和鉴权Token。

返回示例：

```
{
    "status": "success",
    "data": {
        "resultType": "vector",
        "result": [
            {
                "metric": {
                    "__name__": "arms_prometheus_target_interval_length_seconds_sum",
                    "instance": "localhost:9335",
                    "interval": "15s",
                    "job": "_arms-prom/kubelet/1"
                },
                "value": [
                    1635302655,
                    "146655.24420603667"
                ]
            },
            {
                "metric": {
                    "__name__": "arms_prometheus_target_interval_length_seconds_sum",
                    "instance": "localhost:9335",
                    "interval": "30s",
                    "job": "_arms-prom/kubelet/1"
                },
                "value": [
                    1635302655,
                    "879810.747346541"
                ]
            },
            {
                "metric": {
                    "__name__": "arms_prometheus_target_interval_length_seconds_sum",
                    "instance": "localhost:9335",
                    "interval": "20s",
                    "job": "_arms-prom/kubelet/1"
                },
                "value": [
                    1635302655,
                    "73320.13578499513"
                ]
            }
        ]
    }
```


15.8. Agent水平伸缩（HPA）自动扩容能力说明

阿里云Prometheus监控支持Agent副本数水平伸缩（HPA）自动扩容的能力。由于配置的Agent副本数量不足，导致Agent不断产生内存溢出发生重启。因此Prometheus监控新增Agent副本数的HPA自动扩容功能，可以自动调整Agent副本数。

Helm及Agent镜像版本号具备的自动扩容能力

Helm版本号	Agent镜像版本号	是否具备自动扩容能力
v1.0.0	arms-prom-operator:v3.0.0	是
≤v0.1.8	arms-prom-operator:v0.1	否

> **说明** Helm及Agent镜像版本的详细说明，请参见Helm版本说明。

以下两种情况Agent副本数会进行自动扩容：

- 当Agent单副本运行时：其Master副本既需要执行Targets服务发现又需要执行Targets抓取，如果因为Metrics量级过多导致Agent因为OOM而终止进程时，Agent副本数会一次性自动扩容为3个。
- 当Agent多副本运行时：其Master副本仅需执行Targets服务发现，由Worker副本执行Targets抓取。当Worker副本内存使用超过60%时，会进行Targets抓取任务再分配，同时计算出所需的Worker副本数，实现自动扩容。

> **说明** 每个Agent可以抓取的Metrics数量级上限为4,200,000，内存使用上限为60%。

15.9. mysqld_exporter访问MySQL数据库所需的权限说明

若您需要获取MySQL数据库类型的监控指标数据，需要在MySQL数据库中开通相关的权限，将mysqld_exporter连接到MySQL数据库，本文介绍如何设置MySQL数据库的mysqld_exporter权限。

在MySQL数据库中为mysqld_exporter创建一个用户，用户密码可以自行设置。然后执行如下命令，为performance_schema.* 表添加读权限。

```
mysql> GRANT REPLICATION CLIENT, PROCESS ON *.* TO
'mysqld_exporter'@'localhost' identified by 'arms_prometheus2022';
mysql> GRANT SELECT ON performance_schema.* TO 'mysqld_exporter'@'localhost';
mysql> FLUSH PRIVILEGES;
```

> **说明** `mysqld_exporter` 和 `arms_prometheus2022` 是自定义的用户名称和密码，请根据实际情况替换。

16.API参考

16.1. Prometheus监控

16.1.1. AddGrafana

调用AddGrafana接口集成ARMS Prometheus监控的大盘。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AddGrafana	系统规定参数，取值为 `AddGrafana` 。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	阿里云容器服务Kubernetes版的Kubernetes集群的ID。
Integration	String	是	asm	ARMS支持的软件缩写。可选值（不区分大小写）： `ASM` 、 `IoT` 和 `Flink` 。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Data	String	success	操作是否成功。
RequestId	String	1A9C645C-C83F-4C9D-8CCB-29BEC9E1****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AddGrafana
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&Integration=asm
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<AddGrafanaResponse>
   <RequestId>1A9C645C-C83F-4C9D-8CCB-29BEC9E1****</RequestId>
   <Data>success</Data>
</AddGrafanaResponse>
```

`JSON` 格式

```
{
    "RequestId": "1A9C645C-C83F-4C9D-8CCB-29BEC9E1****",
    "Data": "success"
}
```

错误码

访问错误中心查看更多错误码。

16.1.2. DeleteGrafanaResource

调用DeleteGrafanaResource接口删除ARMS Prometheus监控集群中的Grafana大盘资源。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteGrafanaResource	系统规定参数。取值：**DeleteGrafanaResource**。
RegionId	String	是	cn-hangzhou	地域ID。
ClusterName	String	是	clusterNameOfTest	集群名称。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	集群ID。

返回数据

名称	类型	示例值	描述
Data	String	delete success.	返回信息。
RequestId	String	771DC66C-C5E0-59BC-A983-DD18FEE9EFFA	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteGrafanaResource
&RegionId=cn-hangzhou
&ClusterName=clusterNameOfTest
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DeleteGrafanaResourceResponse>
    <Data>delete success.</Data>
    <RequestId>771DC66C-C5E0-59BC-A983-DD18FEE9EFFA</RequestId>
</DeleteGrafanaResourceResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : "delete success.",
  "RequestId" : "771DC66C-C5E0-59BC-A983-DD18FEE9EFFA"
}
```

错误码

访问错误中心查看更多错误码。

16.1.3. AddIntegration

调用AddIntegration接口集成ARMS Prometheus监控的大盘以及采集规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AddIntegration	系统规定参数，取值为 `AddIntegration` 。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	阿里云容器服务Kubernetes版的Kubernetes集群的ID。

名称	类型	是否必选	示例值	描述
Integration	String	是	asm	ARMS支持的软件缩写。可选值（不区分大小写）：`ASM`、`IoT`和`Flink`。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Data	String	success	操作是否成功。
RequestId	String	1A9C645C-C83F-4C9D-8CCB-29BEC9E1****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AddIntegration
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&Integration=asm
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<AddIntegrationResponse>
   <RequestId>1A9C645C-C83F-4C9D-8CCB-29BEC9E1****</RequestId>
   <Data>success</Data>
</AddIntegrationResponse>
```

`JSON` 格式

```
{
    "RequestId": "1A9C645C-C83F-4C9D-8CCB-29BEC9E1****",
    "Data": "success"
}
```

错误码

访问错误中心查看更多错误码。

16.1.4. DeleteIntegration

调用DeleteIntegration接口来删除Integration接入的采集规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteIntegration	系统规定参数。取值：**DeleteIntegration**。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	阿里云容器服务Kubernetes版的Kubernetes集群的ID。
RegionId	String	是	cn-hangzhou	地域ID。
Integration	String	是	asm	ARMS支持的软件缩写。可选值（不区分大小写）：`ASM`、`IoT`和`Flink`。

返回数据

名称	类型	示例值	描述
RequestId	String	1A9C645C-C83F-4C9D-8CCB-29BEC9E1****	请求ID。
Data	String	success	操作是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteIntegration
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&RegionId=cn-hangzhou
&Integration=asm
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DeleteIntegrationResponse>
    <RequestId>1A9C645C-C83F-4C9D-8CCB-29BEC9E1****</RequestId>
    <Data>success</Data>
</DeleteIntegrationResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1A9C645C-C83F-4C9D-8CCB-29BEC9E1****",
  "Data" : "success"
}
```

错误码

访问错误中心查看更多错误码。

16.1.5. GetIntegrationState

调用GetIntegrationState接口来获取Integration的接入状态。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetIntegrationState	系统规定参数。取值：**GetIntegrationState**。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	阿里云容器服务Kubernetes版的Kubernetes集群的ID。
RegionId	String	是	cn-hangzhou	地域ID。
Integration	String	是	asm	ARMS支持的软件缩写。可选值（不区分大小写）：`ASM`、`IoT`和`Flink`。

返回数据

名称	类型	示例值	描述
RequestId	String	1A9C645C-C83F-4C9D-8CCB-29BEC9E1****	请求ID。
State	Boolean	true	Integration的接入状态。 • `true`：表示已接入。 • `false`：表示未接入。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetIntegrationState
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&RegionId=cn-hangzhou
&Integration=asm
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<GetIntegrationStateResponse>
    <RequestId>1A9C645C-C83F-4C9D-8CCB-29BEC9E1****</RequestId>
    <State>true</State>
</GetIntegrationStateResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "1A9C645C-C83F-4C9D-8CCB-29BEC9E1****",
  "State" : true
}
```

错误码

访问错误中心查看更多错误码。

16.1.6. GetPrometheusApiToken

调用GetPrometheusApiToken接口获取集成ARMS Prometheus监控所需的Token。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetPrometheusApiToken	系统规定参数，取值为 `GetPrometheusApiToken` 。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述

名称	类型	示例值	描述
RequestId	String	1A9C645C-C83F-4C9D-8CCB-29BEC9E1****	请求ID。
Token	String	6dcbb77ef4ba6ef5466b5debf9e2****	集成ARMS Prometheus监控所需的Token。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetPrometheusApiToken
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<GetPrometheusApiTokenResponse>
   <RequestId>1A9C645C-C83F-4C9D-8CCB-29BEC9E1****</RequestId>
   <Token>6dcbb77ef4ba6ef5466b5debf9e2****</Token>
</GetPrometheusApiTokenResponse>
```

`JSON` 格式

```
{
    "RequestId": "1A9C645C-C83F-4C9D-8CCB-29BEC9E1****",
    "Token": "6dcbb77ef4ba6ef5466b5debf9e2****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.7. ListDashboards

调用ListDashboards接口获取集群的Grafana大盘的列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListDashboards	系统规定参数。取值：ListDashboards。
RegionId	String	是	cn-hangzhou	地域ID。

名称	类型	是否必选	示例值	描述
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	阿里云容器服务Kubernetes版的Kubernetes集群的ID。
ClusterType	String	否	Node	集群类型，必选。虚拟集群可通过集群类型查询大盘列表。InfluxDB类云产品统一传 `cloud-product-prometheus` 。
Title	String	否	ApiServer	指定大盘标题。大盘标题可能会修改，建议使用**DashboardName**查询。
Product	String	否	xxxx	云产品Code，可选。 当 `clusterType=cloud-product-prometheus` 时该字段必传。目前已经接入的云产品有： • InfluxDB • MongoDB • DLA • SAE
RecreateSwitch	Boolean	否	false	创建或者查询虚拟集群开关参数，可以对老数据兼容控制。
DashboardName	String	否	k8s-node-overview	大盘唯一名称，可筛选查询指定名称的大盘。相对于**Title**参数，**Title**可能会发生变化，name不会。并且支持指定多个name，以半角逗号（,）分隔，例如： `k8s-event,k8s-overview` 。同一个大盘名称会有多个版本，如果要指定版本，可以在name后面增加版本信息，例如： `k8s-event:v1,k8s-overview:latest` 。

返回数据

名称	类型	示例值	描述
RequestId	String	2A0CEDF1-06FE-44AC-8E21-21A5BE65****	请求ID。
DashboardVos	Array of DashboardVos		Grafana大盘信息。
Type	String	dash-db	Grafana大盘类型，包括： • `dash-db` ：大盘 • `dash-folder` ：文件夹（可包含大盘）
Time	String	1590136924	Grafana大盘创建时间的时间戳。

名称	类型	示例值	描述
NeedUpdate	Boolean	false	大盘是否有新版本可以升级。
Kind	String	BASIC	大盘种类，为BASIC、THIRD、LIMIT、CUSTOM的其中一种。
Url	String	http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&refresh=60s	Grafana大盘的完整URL。
HttpsUrl	String	http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&refresh=60s	Grafana大盘URL。
DashboardType	String	Node	大盘类型，作用与Exporter一致，但是字段含义更明确。
Exporter	String	Nginx	Exporter接入源的类型，包括： • Prometheus • Node • GPU • Redis • MySQL • Kafka • Nginx（v2） • Nginx • ZooKeeper • MongoDB • RabbitMQ • PostgreSQL • Kubernetes • Client Library • Elasticsearch • RocketMQ
Version	String	v2	大盘版本，与name形成唯一键，确定一个大盘。
IsArmsExporter	Boolean	false	是否属于ARMS提供的Exporter： • `true`：是ARMS提供的Exporter。 • `false`：不是ARMS提供的Exporter。

名称	类型	示例值	描述
HttpUrl	String	http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&refresh=60s	Grafana大盘的URL。
Title	String	ApiServer	Grafana大盘标题。
Name	String	k8s-node-overview	大盘名称，与**Title**不同，不会修改。
Id	String	1100**	Grafana大盘ID，仅在安装Grafana大盘时是唯一的。
Uid	String	1131971649496228-*****-59	安装多个Grafana大盘时的大盘唯一标识符，是展示在页面上的唯一业务ID。
Tags	Array of String	["arms-k8s","ccc8ce1fe0c9543629e39ee657e34*****"]	Grafana大盘标签。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListDashboards
&RegionId=cn-hangzhou
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&ClusterType=Node
&Title=ApiServer
&Product=xxxx
&RecreateSwitch=false
&DashboardName=k8s-node-overview
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ListDashboardsResponse>
    <RequestId>2A0CEDF1-06FE-44AC-8E21-21A5BE65****</RequestId>
    <DashboardVos>
        <Type>dash-db</Type>
        <Time>1590136924</Time>
        <NeedUpdate>false</NeedUpdate>
        <Kind>BASIC</Kind>
        <Url>http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&amp;refresh=60s</Url>
        <HttpsUrl>http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&amp;refresh=60s</HttpsUrl>
        <DashboardType>Node</DashboardType>
        <Exporter>Nginx</Exporter>
        <Version>v2</Version>
        <IsArmsExporter>false</IsArmsExporter>
        <HttpUrl>http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&amp;refresh=60s</HttpUrl>
        <Title>ApiServer</Title>
        <Name>k8s-node-overview</Name>
        <Id>1100**</Id>
        <Uid>1131971649496228-*****-59</Uid>
        <Tags>["arms-k8s","ccc8ce1fe0c9543629e39ee657e34****"]</Tags>
    </DashboardVos>
</ListDashboardsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "2A0CEDF1-06FE-44AC-8E21-21A5BE65****",
  "DashboardVos" : {
    "Type" : "dash-db",
    "Time" : 1590136924,
    "NeedUpdate" : false,
    "Kind" : "BASIC",
    "Url" : "http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&amp;refresh=60s",
    "HttpsUrl" : "http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&amp;refresh=60s",
    "DashboardType" : "Node",
    "Exporter" : "Nginx",
    "Version" : "v2",
    "IsArmsExporter" : false,
    "HttpUrl" : "http://g.console.aliyun.com/d/1131971649496228-*****-59/ApiServer?orgId=3**&amp;refresh=60s",
    "Title" : "ApiServer",
    "Name" : "k8s-node-overview",
    "Id" : "1100**",
    "Uid" : "1131971649496228-*****-59",
    "Tags" : "[\"arms-k8s\",\"ccc8ce1fe0c9543629e39ee657e34****\"]"
  }
}
```

错误码

访问错误中心查看更多错误码。

16.1.8. CreatePrometheusAlertRule

调用CreatePrometheusAlertRule接口创建告警规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreatePrometheusAlertRule	系统规定参数。取值：**CreatePrometheusAlertRule**。
RegionId	String	是	cn-hangzhou	地域ID。
AlertName	String	是	Prometheus_Alert	告警规则名称。

名称	类型	是否必选	示例值	描述
ClusterId	String	是	c0bad479465464e1d8c1e641b0afb****	集群ID。
Type	String	否	Kubernetes组件告警	告警规则类型。
NotifyType	String	否	ALERT_MANAGER	通知类型。取值： • `ALERT_MANAGER`（默认）：通过告警运维中心通知。 • `DISPATCH_RULE`：指定通知策略进行通知。
DispatchRuleId	Long	否	10282	通知策略ID，当**NotifyType**指定为`DISPATCH_RULE`时必填。
Expression	String	是	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(.*)\")) by (pod_name))>75	告警表达式，需要使用PromQL语句。
Duration	String	是	1m	持续时间，范围在1m~1440m，单位为分钟。
Message	String	是	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	告警消息，支持按照{{$labels.xxx}}格式来引用标签。
Labels	String	否	[{"Value": "critical","Name": "severity"}]	标签JSON串。需要设置标签的Name和Value。
Annotations	String	否	[{"Value": "xxx","Name": "description"}]	注释JSON串。需要设置注释的Name和Value。

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
PrometheusAlertRule	Object		返回结构体。
Status	Integer	1	告警规则启用状态。取值： • `1`：开启。 • `0`：关闭。
Type	String	Kubernetes组件告警	告警规则类型。
NotifyType	String	ALERT_MANAGER	通知类型。取值： • `ALERT_MANAGER`：通过告警运维中心通知。 • `DISPATCH_RULE`：指定通知策略进行通知。
Expression	String	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(.*)\")) by (pod_name))>75	告警表达式。
Message	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	告警消息，支持按照{{$labels.xxx}}格式来引用标签。
Duration	String	1m	持续时间，范围在1m~1440m，单位为分钟。
DispatchRuleId	Long	10282	通知策略ID。
AlertName	String	Prometheus_Alert	告警规则名称。
AlertId	Long	3888704	告警规则ID。
ClusterId	String	c0bad479465464e1d8c1e641b0afb****	集群ID。
Labels	Array of Label		告警规则的标签。

名称	类型	示例值	描述
Name	String	severity	标签的名称。
Value	String	critical	标签的值。
Annotations	Array of Annotation		告警规则的注释。
Name	String	message	注释的名称。
Value	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	注释的值。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreatePrometheusAlertRule
&RegionId=cn-hangzhou
&AlertName=Prometheus_Alert
&ClusterId=c0bad479465464e1d8c1e641b0afb****
&Type=Kubernetes组件告警
&NotifyType=ALERT_MANAGER
&DispatchRuleId=10282
&Expression=100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(lab
el_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(
.*)\")) by (pod_name))>75
&Duration=1m
&Message=${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%
&Labels=[{"Value": "critical","Name": "severity"}]
&Annotations=[{"Value": "xxx","Name": "description"}]
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<CreatePrometheusAlertRuleResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <PrometheusAlertRule>
        <Status>1</Status>
        <Type>Kubernetes组件告警</Type>
        <NotifyType>ALERT_MANAGER</NotifyType>
        <Expression>100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod
\", \"(.*)\")) by (pod_name))&amp;gt;75</Expression>
        <Message>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Message>
        <Duration>1m</Duration>
        <DispatchRuleId>10282</DispatchRuleId>
        <AlertName>Prometheus_Alert</AlertName>
        <AlertId>3888704</AlertId>
        <ClusterId>c0bad479465464e1d8c1e641b0afb****</ClusterId>
        <Labels>
            <Name>severity</Name>
            <Value>critical</Value>
        </Labels>
        <Annotations>
            <Name>message</Name>
            <Value>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Value>
        </Annotations>
    </PrometheusAlertRule>
</CreatePrometheusAlertRuleResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "PrometheusAlertRule" : {
    "Status" : 1,
    "Type" : "Kubernetes组件告警",
    "NotifyType" : "ALERT_MANAGER",
    "Expression" : "100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \\\"pod_name\\\", \\\"$1\\\
", \\\"pod\\\", \\\"(.*)\\\")) by (pod_name))&amp;gt;75",
    "Message" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
    "Duration" : "1m",
    "DispatchRuleId" : 10282,
    "AlertName" : "Prometheus_Alert",
    "AlertId" : 3888704,
    "ClusterId" : "c0bad479465464e1d8c1e641b0afb****",
    "Labels" : {
      "Name" : "severity",
      "Value" : "critical"
    },
    "Annotations" : {
      "Name" : "message",
      "Value" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%"
    }
  }
}
```

错误码

访问错误中心查看更多错误码。

16.1.9. ListPrometheusAlertRules

调用ListPrometheusAlertRules接口查看Prometheus告警规则列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListPrometheusAlertRules	系统规定参数。取值：**ListPrometheusAlertRules**。
Name	String	否	Prometheus_Alert	告警规则名称。
RegionId	String	是	cn-hangzhou	地域ID。

名称	类型	是否必选	示例值	描述
ClusterId	String	是	c0bad479465464e1d8c1e641b0afb****	集群ID。
Type	String	否	自定义	告警规则类型。
Status	Integer	否	1	告警规则启用状态，取值： • `1`：开启 • `0`：关闭
MatchExpressions	String	否	[{"key":"severity","value":"critical","operator":"re"}]	标签匹配条件的JSON串。关于此字段的详细说明参见下文**关于参数MatchExpressions的补充说明**。

关于参数MatchExpressions的补充说明

JSON串示例及说明

```
[
{
    "key":"severity",                          // 标签的Key。
    "value":"critical",                        // 标签的Value。
    "operator":"re"                            // eq：等于；re：匹配正则。
}
]
```

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
PrometheusAlertRules	Array of PrometheusAlertRule		返回结构体。
Status	Integer	1	告警规则启用状态，取值： • `1`：开启 • `0`：关闭
Type	String	自定义	告警规则类型。
NotifyType	String	ALERT_MANAGER	通知类型， 取值： • `ALERT_MANAGER`：通过报警运维中心通知。 • `DISPATCH_RULE`：指定通知策略进行通知。

名称	类型	示例值	描述
Expression	String	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(.*)\")) by (pod_name))>75	告警表达式。
Message	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	告警通知消息，支持按照{{$labels.xxx}}格式来引用标签。
Duration	String	1m	持续时间，范围在1m~1440m，单位为分钟。
DispatchRuleId	Long	10282	通知策略ID，当**NotifyType**指定为 `DISPATCH_RULE` 时显示此参数。
AlertName	String	Prometheus_Alert	告警规则名称。
AlertId	Long	3888704	告警规则ID。
ClusterId	String	c0bad479465464e1d8c1e641b0afb****	集群ID。
Labels	Array of Label		告警规则的标签。
Name	String	severity	标签的名称。
Value	String	critical	标签的值。
Annotations	Array of Annotation		告警规则的注释。
Name	String	message	注释的名称。
Value	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	注释的值。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListPrometheusAlertRules
&ClusterId=c0bad479465464e1d8c1e641b0afb****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ListPrometheusAlertRulesResponse>
    <PrometheusAlertRules>
        <Status>1</Status>
        <NotifyType>ALERT_MANAGER</NotifyType>
        <Type>自定义</Type>
        <AlertId>3888704</AlertId>
        <AlertName>Prometheus_Alert</AlertName>
        <Message>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Message>
        <ClusterId>c0bad479465464e1d8c1e641b0afb****</ClusterId>
        <Expression>100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod
\", \"(.*)\")) by (pod_name))&amp;gt;75</Expression>
        <DispatchRuleId>10282</DispatchRuleId>
        <Duration>1m</Duration>
        <Labels>
            <Value>critical</Value>
            <Name>severity</Name>
        </Labels>
        <Annotations>
            <Value>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Value>
            <Name>message</Name>
        </Annotations>
    </PrometheusAlertRules>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
</ListPrometheusAlertRulesResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "PrometheusAlertRules" : {
    "Status" : 1,
    "NotifyType" : "ALERT_MANAGER",
    "Type" : "自定义",
    "AlertId" : 3888704,
    "AlertName" : "Prometheus_Alert",
    "Message" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
    "ClusterId" : "c0bad479465464e1d8c1e641b0afb****",
    "Expression" : "100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \\\"pod_name\\\", \\\"$1\\\
", \\\"pod\\\", \\\"(.*)\\\")) by (pod_name))&amp;gt;75",
    "DispatchRuleId" : 10282,
    "Duration" : "1m",
    "Labels" : {
      "Value" : "critical",
      "Name" : "severity"
    },
    "Annotations" : {
      "Value" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
      "Name" : "message"
    }
  },
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.10. DescribePrometheusAlertRule

调用DescribePrometheusAlertRule接口查看Prometheus告警规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribePrometheusAlertRule	系统规定参数。取值：**DescribePrometheusAlertRule**。
AlertId	Long	是	3888704	告警规则ID，您可以在ListPrometheusAlertRules API接口的返回数据中查看。

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
PrometheusAlertRule	Object		返回结构体。
Status	Integer	1	告警规则启用状态，取值： • `1`：开启 • `0`：关闭
Type	String	Kubernetes组件告警	告警规则类型。
NotifyType	String	ALERT_MANAGER	通知类型， 取值： • `ALERT_MANAGER`：通过告警运维中心通知。 • `DISPATCH_RULE`：指定通知策略进行通知。
Expression	String	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(.*)\")) by (pod_name))>75	告警表达式。
Message	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	告警通知消息，支持按照{{$labels.xxx}}格式来引用标签。
Duration	String	1m	持续时间，范围在1m~1440m，单位为分钟。
DispatchRuleId	Long	10282	通知策略ID，当**NotifyType**指定为 `DISPATCH_RULE` 时显示此参数。
AlertName	String	Prometheus_Alert	告警规则名称。
AlertId	Long	3888704	告警规则ID。
ClusterId	String	c0bad479465464e1d8c1e641b0afb****	集群ID。

名称	类型	示例值	描述
Labels	Array of Label		告警规则的标签。
Name	String	severity	标签的名称。
Value	String	critical	标签的值。
Annotations	Array of Annotation		告警规则的注释。
Name	String	message	注释的名称。
Value	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	注释的值。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribePrometheusAlertRule
&AlertId=3888704
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DescribePrometheusAlertRuleResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <PrometheusAlertRule>
        <Status>1</Status>
        <NotifyType>ALERT_MANAGER</NotifyType>
        <Type>Kubernetes组件告警</Type>
        <AlertId>3888704</AlertId>
        <AlertName>Prometheus_Alert</AlertName>
        <Message>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Message>
        <ClusterId>c0bad479465464e1d8c1e641b0afb****</ClusterId>
        <Expression>100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod
\", \"(.*)\")) by (pod_name))&amp;gt;75</Expression>
        <DispatchRuleId>10282</DispatchRuleId>
        <Duration>1m</Duration>
        <Labels>
            <Value>critical</Value>
            <Name>severity</Name>
        </Labels>
        <Annotations>
            <Value>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Value>
            <Name>message</Name>
        </Annotations>
    </PrometheusAlertRule>
</DescribePrometheusAlertRuleResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "PrometheusAlertRule" : {
    "Status" : 1,
    "NotifyType" : "ALERT_MANAGER",
    "Type" : "Kubernetes组件告警",
    "AlertId" : 3888704,
    "AlertName" : "Prometheus_Alert",
    "Message" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
    "ClusterId" : "c0bad479465464e1d8c1e641b0afb****",
    "Expression" : "100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \\\"pod_name\\\", \\\"$1\\\
", \\\"pod\\\", \\\"(.*)\\\")) by (pod_name))&amp;gt;75",
    "DispatchRuleId" : 10282,
    "Duration" : "1m",
    "Labels" : {
      "Value" : "critical",
      "Name" : "severity"
    },
    "Annotations" : {
      "Value" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
      "Name" : "message"
    }
  }
}
```

错误码

访问错误中心查看更多错误码。

16.1.11. UpdatePrometheusAlertRule

调用UpdatePrometheusAlertRule接口更新告警规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	UpdatePrometheusAlertRule	系统规定参数。取值：**UpdatePrometheusAlertRule**。
RegionId	String	是	cn-hangzhou	地域ID。
AlertName	String	是	Prometheus_Alert	告警规则名称。

名称	类型	是否必选	示例值	描述
ClusterId	String	是	c0bad479465464e1d8c1e641b0afb****	集群ID。
Type	String	否	Kubernetes组件告警	自定义告警规则类型。
NotifyType	String	否	ALERT_MANAGER	通知类型， 取值： • `ALERT_MANAGER` ：通过告警运维中心通知。 • `DISPATCH_RULE` ：指定通知策略进行通知。
DispatchRuleId	Long	否	10282	通知策略ID，当**NotifyType**指定为 `DISPATCH_RULE` 时必填。
Expression	String	是	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(.*)\")) by (pod_name))>75	告警表达式，需要使用PromQL语句。
Duration	String	是	1m	持续时间，范围在1m~1440m，单位为分钟。
Message	String	是	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	告警消息，支持按照{{$labels.xxx}}格式来引用标签。
Labels	String	否	[{"Value": "critical","Name": "severity"}]	标签JSON串。需要设置标签的Name和Value。
Annotations	String	否	[{"Value": "xxx","Name": "description"}]	注释JSON串。需要设置注释的Name和Value。
AlertId	Long	是	3888704	告警规则ID，您可以在ListPrometheusAlertRules API接口的返回数据中查看。

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
PrometheusAlertRule	Object		返回结构体。
Status	Integer	1	告警规则启用状态，取值： • `1`：开启 • `0`：关闭
Type	String	Kubernetes组件告警	告警规则类型。
NotifyType	String	ALERT_MANAGER	通知类型， 取值： • `ALERT_MANAGER`：通过报警运维中心通知。 • `DISPATCH_RULE`：指定通知策略进行通知。
Expression	String	100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(.*)\")) by (pod_name))>75	告警表达式。
Message	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	告警通知消息，支持按照{{$labels.xxx}}格式来引用标签。
Duration	String	1m	持续时间，范围在1m~1440m，单位为分钟。
DispatchRuleId	Long	10282	通知策略ID。
AlertName	String	Prometheus_Alert	告警规则名称。
AlertId	Long	3888704	告警规则ID。
ClusterId	String	c0bad479465464e1d8c1e641b0afb****	集群ID。

名称	类型	示例值	描述
Labels	Array of Label		告警规则的标签。
Name	String	severity	标签的名称。
Value	String	critical	标签的值。
Annotations	Array of Annotation		告警规则的注释。
Name	String	message	注释的名称。
Value	String	${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%	注释的值。

示例

请求示例

```
http(s)://[Endpoint]/?Action=UpdatePrometheusAlertRule
&AlertId=3888704
&AlertName=Prometheus_Alert
&ClusterId=c0bad479465464e1d8c1e641b0afb****
&Duration=1m
&Expression=100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) / sum(lab
el_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod\", \"(
.*)\")) by (pod_name))>75
&Message=${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<UpdatePrometheusAlertRuleResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <PrometheusAlertRule>
        <Status>1</Status>
        <NotifyType>ALERT_MANAGER</NotifyType>
        <Type>Kubernetes组件告警</Type>
        <AlertId>3888704</AlertId>
        <AlertName>Prometheus_Alert</AlertName>
        <Message>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Message>
        <ClusterId>c0bad479465464e1d8c1e641b0afb****</ClusterId>
        <Expression>100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \"pod_name\", \"$1\", \"pod
\", \"(.*)\")) by (pod_name))&amp;gt;75</Expression>
        <DispatchRuleId>10282</DispatchRuleId>
        <Duration>1m</Duration>
        <Labels>
            <Value>critical</Value>
            <Name>severity</Name>
        </Labels>
        <Annotations>
            <Value>${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%</Value>
            <Name>message</Name>
        </Annotations>
    </PrometheusAlertRule>
</UpdatePrometheusAlertRuleResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "PrometheusAlertRule" : {
    "Status" : 1,
    "NotifyType" : "ALERT_MANAGER",
    "Type" : "Kubernetes组件告警",
    "AlertId" : 3888704,
    "AlertName" : "Prometheus_Alert",
    "Message" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
    "ClusterId" : "c0bad479465464e1d8c1e641b0afb****",
    "Expression" : "100 * (sum(rate(container_cpu_usage_seconds_total[1m])) by (pod_name) /
sum(label_replace(kube_pod_container_resource_limits_cpu_cores, \\\"pod_name\\\", \\\"$1\\\
", \\\"pod\\\", \\\"(.*)\\\")) by (pod_name))&amp;gt;75",
    "DispatchRuleId" : 10282,
    "Duration" : "1m",
    "Labels" : {
      "Value" : "critical",
      "Name" : "severity"
    },
    "Annotations" : {
      "Value" : "${{$labels.pod_name}}CPU使用率大于80%，当前值{{$value}}%",
      "Name" : "message"
    }
  }
}
```

错误码

访问错误中心查看更多错误码。

16.1.12. DeletePrometheusAlertRule

调用DeletePrometheusAlertRule接口删除Prometheus告警规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeletePrometheusAlertRule	系统规定参数。取值：**DeletePrometheusAlertRule**。
AlertId	Long	是	3888704	告警规则ID，您可以在ListPrometheusAlertRules API接口的返回数据中查看。

返回数据

名称	类型	示例值	描述
Success	Boolean	true	是否删除成功。取值： • `true`：删除成功 • `false`：删除失败
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeletePrometheusAlertRule
&AlertId=3888704
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DeletePrometheusAlertRuleResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <Success>true</Success>
</DeletePrometheusAlertRuleResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "Success" : true
}
```

错误码

访问错误中心查看更多错误码。

16.1.13. ListPrometheusAlertTemplates

调用ListPrometheusAlertTemplates接口查看Prometheus告警模板列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListPrometheusAlertTemplates	系统规定参数。取值：**ListPrometheusAlertTemplates**。
ClusterId	String	否	c0bad479465464e1d8c1e641b0afb****	集群ID。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
PrometheusAlertTemplates	Array of PrometheusAlertTemplate		返回结构体。
Type	String	节点	告警规则类型。
Description	String	节点 {{ $labels.instance }} 可用内存不足10%，当前可用内存 {{ $value }}%	告警消息，支持按照{{$labels.xxx}}格式来引用标签。
Expression	String	node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 < 10	告警表达式。
Version	String	1.0	告警规则版本。
Duration	String	1m	持续时间，范围在1m~1440m，单位为分钟。
AlertName	String	节点内存可用率不足10%	告警规则名称。
Labels	Array of Label		告警规则的标签。
Name	String	severity	标签的名称。
Value	String	warning	标签的值。
Annotations	Array of Annotation		告警规则的注释。

名称	类型	示例值	描述
Name	String	message	注释的名称。
Value	String	节点 {{ $labels.instance }} 可用内存不足10%，当前可用内存 {{ $value }}%	注释的值。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListPrometheusAlertTemplates
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ListPrometheusAlertTemplatesResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <PrometheusAlertTemplates>
        <Type>节点</Type>
        <Description>节点 {{ $labels.instance }} 可用内存不足10%，当前可用内存 {{ $value }}%</D
escription>
        <AlertName>节点内存可用率不足10%</AlertName>
        <Version>1</Version>
        <Expression>node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 &amp;
lt; 10</Expression>
        <Duration>1m</Duration>
        <Labels>
            <Value>warning</Value>
            <Name>severity</Name>
        </Labels>
        <Annotations>
            <Value>节点 {{ $labels.instance }} 可用内存不足10%，当前可用内存 {{ $value }}%</Val
ue>
            <Name>message</Name>
        </Annotations>
    </PrometheusAlertTemplates>
</ListPrometheusAlertTemplatesResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "PrometheusAlertTemplates" : {
    "Type" : "节点",
    "Description" : "节点 {{ $labels.instance }} 可用内存不足10%，当前可用内存 {{ $value }}%",
    "AlertName" : "节点内存可用率不足10%",
    "Version" : 1,
    "Expression" : "node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 &amp;
lt; 10",
    "Duration" : "1m",
    "Labels" : {
      "Value" : "warning",
      "Name" : "severity"
    },
    "Annotations" : {
      "Value" : "节点 {{ $labels.instance }} 可用内存不足10%，当前可用内存 {{ $value }}%",
      "Name" : "message"
    }
  }
}
```

错误码

访问错误中心查看更多错误码。

16.1.14. ListActivatedAlerts

调用ListActivatedAlerts接口查询已经触发的告警列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListActivatedAlerts	系统规定参数。取值：ListActivatedAlerts。
CurrentPage	Integer	是	1	查询结果分页的页码。默认为 `1` 。
PageSize	Integer	是	10	查询结果分页的每页项目数量。默认为 `10` 。
RegionId	String	是	cn-hangzhou	地域ID。
Filter	String	否	{"alertname":"容器CPU使用率大于80%"}	筛选条件，格式为 `{"key":"value"}` 。需要设置筛选条件的 `key` 和 `value` 。

返回数据

名称	类型	示例值	描述
Page	Struct		返回结构体。
Alerts	Array of Alert		告警信息。
AlertId	String	3888704	告警规则ID。
AlertName	String	容器CPU使用率大于80%	告警规则名称。
AlertType	String	ARMS-Prometheus监控	告警类型。
Count	Integer	598	告警事件接受次数。
CreateTime	Long	1616466300000	告警规则创建时间的时间戳。
DispatchRules	Array of DispatchRule		通知策略。
RuleId	Integer	7021	通知策略ID。
RuleName	String	容器CPU使用率大于80%的通知策略	通知策略名称。
EndsAt	Long	1616502540000	告警结束时间。

名称	类型	示例值	描述
ExpandFields	Map	"severity": "critical", "_aliyun_arms_alert_level": "ERROR", "pod": "night-test-group-1-1-5f5d6f4d84-pszns", "_aliyun_arms_alert_type": "101", "_aliyun_arms_integration_name": "测试集成-prometheus", "alertname": "PodRestart_jiubiantestphp2", "_aliyun_arms_userid": "1131971649496228", "_aliyun_arms_involvedObject_name": "jiubiantestphp2", "_aliyun_arms_involvedObject_id": "ccafb2763cfa7415eb2e2a60a74b1f825", "_aliyun_arms_region_id": "cn-beijing", "_aliyun_arms_involvedObject_kind": "cluster", "_aliyun_arms_product_type": "PROMETHEUS", "namespace": "default", "_aliyun_arms_integration_id": "80", "_aliyun_arms_involvedObject_type": "ManagedKubernetes", "_aliyun_arms_alert_rule_id": "3612229"	扩展字段（标签），标签来源包括： • 报警规则表达式指标中携带的标签。 • 通过报警规则创建的标签。 • ARMS系统自带的默认标签。
IntegrationName	String	testphp2	告警关联对象名称。
IntegrationType	String	PROMETHEUS	告警来源集成的类型。
InvolvedObjectKind	String	cluster	告警关联对象类型。

名称	类型	示例值	描述
InvolvedObject Name	String	测试集成-prometheus	告警来源集成的名称。
Message	String	报警名称: PodRestart_testphp2，\n Pod night-test-group-1-1-5f5d6f4d84-pszns is restart, Value: 133.33%, 1.33%	告警描述信息。
Severity	String	critical	告警等级。取值: • `critical`：严重。 • `error`：错误。 • `warn`：警告。 • `page`：通知。
StartsAt	Long	1616466300000	告警开始时间。
Status	String	Active	告警状态。取值: • `Active`：未恢复。 • `Inhibited`：抑制。 • `Silenced`：静默。 • `Resolved`：已恢复。
Page	Integer	1	查询结果分页页码。
PageSize	Integer	20	查询结果分页的每页项目数量。
Total	Integer	5	查询结果总数。
RequestId	String	BDB74B8F-4123-482A-ABB7-7F440349****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListActivatedAlerts
&CurrentPage=1
&PageSize=10
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<ListActivatedAlertsResponse>
  <RequestId>BDB74B8F-4123-482A-ABB7-7F440349****</RequestId>
  <Page>
        <PageSize>20</PageSize>
        <Total>5</Total>
        <Page>1</Page>
        <Alerts>
              <Status>Active</Status>
              <AlertName>容器CPU使用率大于80%</AlertName>
              <Message>报警名称：PodRestart_testphp2，\n Pod night-test-group-1-1-5f5d6f4d84-
pszns is restart, Value: 133.33%, 1.33%</Message>
              <InvolvedObjectKind>cluster</InvolvedObjectKind>
              <CreateTime>1616466300000</CreateTime>
              <Severity>critical</Severity>
              <Count>598</Count>
              <ExpandFields>       "severity": "critical",           "_aliyun_arms_alert_le
vel": "ERROR",           "pod": "night-test-group-1-1-5f5d6f4d84-pszns",           "_aliyun
_arms_alert_type": "101",           "_aliyun_arms_integration_name": "测试集成-prometheus",
"alertname": "PodRestart_jiubiantestphp2",           "_aliyun_arms_userid": "11319716494962
28",           "_aliyun_arms_involvedObject_name": "jiubiantestphp2",           "_aliyun_ar
ms_involvedObject_id": "ccafb2763cfa7415eb2e2a60a74b1f825",           "_aliyun_arms_region_
id": "cn-beijing",           "_aliyun_arms_involvedObject_kind": "cluster",           "_ali
yun_arms_product_type": "PROMETHEUS",           "namespace": "default",           "_aliyun_
arms_integration_id": "80",           "_aliyun_arms_involvedObject_type": "ManagedKubernete
s",           "_aliyun_arms_alert_rule_id": "3612229"</ExpandFields>
              <InvolvedObjectName>测试集成-prometheus</InvolvedObjectName>
              <EndsAt>1616502540000</EndsAt>
              <AlertType>ARMS-Prometheus监控</AlertType>
              <IntegrationName>testphp2</IntegrationName>
              <AlertId>3888704</AlertId>
              <StartsAt>1616466300000</StartsAt>
              <IntegrationType>PROMETHEUS</IntegrationType>
              <DispatchRules>
                    <RuleId>7021</RuleId>
                    <RuleName>容器CPU使用率大于80%的通知策略</RuleName>
              </DispatchRules>
        </Alerts>
  </Page>
</ListActivatedAlertsResponse>
```

`JSON` 格式

```
{
    "RequestId": "BDB74B8F-4123-482A-ABB7-7F440349****",
    "Page": {
        "PageSize": 20,
        "Total": 5,
        "Page": 1,
        "Alerts": {
            "Status": "Active",
            "AlertName": "容器CPU使用率大于80%",
            "Message": "报警名称：PodRestart_testphp2，\\n Pod night-test-group-1-1-5f5d6f4d8
4-pszns is restart, Value: 133.33%, 1.33%",
            "InvolvedObjectKind": "cluster",
            "CreateTime": 1616466300000,
            "Severity": "critical",
            "Count": 598,
            "ExpandFields": "\"severity\": \"critical\",           \"_aliyun_arms_alert_lev
el\": \"ERROR\",           \"pod\": \"night-test-group-1-1-5f5d6f4d84-pszns\",           \"
_aliyun_arms_alert_type\": \"101\",           \"_aliyun_arms_integration_name\": \"测试集成-
prometheus\",           \"alertname\": \"PodRestart_jiubiantestphp2\",           \"_aliyun_
arms_userid\": \"1131971649496228\",           \"_aliyun_arms_involvedObject_name\": \"jiub
iantestphp2\",           \"_aliyun_arms_involvedObject_id\": \"ccafb2763cfa7415eb2e2a60a74b
1f825\",           \"_aliyun_arms_region_id\": \"cn-beijing\",           \"_aliyun_arms_inv
olvedObject_kind\": \"cluster\",           \"_aliyun_arms_product_type\": \"PROMETHEUS\",
\"namespace\": \"default\",           \"_aliyun_arms_integration_id\": \"80\",           \"
_aliyun_arms_involvedObject_type\": \"ManagedKubernetes\",           \"_aliyun_arms_alert_r
ule_id\": \"3612229\"",
            "InvolvedObjectName": "测试集成-prometheus",
            "EndsAt": 1616502540000,
            "AlertType": "ARMS-Prometheus监控",
            "IntegrationName": "testphp2",
            "AlertId": 3888704,
            "StartsAt": 1616466300000,
            "IntegrationType": "PROMETHEUS",
            "DispatchRules": {
                "RuleId": 7021,
                "RuleName": "容器CPU使用率大于80%的通知策略"
            }
        }
    }
}
```


错误码

访问错误中心查看更多错误码。

16.1.15. GetRecordingRule

获取集群的RecordingRule聚合规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetRecordingRule	系统规定参数。取值：**GetRecordingRule**。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	集群ID。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
Data	String	--- groups: - name: "recording_demo" rules: - expr: "sum(jvm_memory_max_bytes)" record: "rate_coredns_demo"	获取到的集群RecordingRule聚合规则。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetRecordingRule
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<GetRecordingRuleResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <Data>--- groups: - name: "recording_demo"   rules:   - expr: "sum(jvm_memory_max_bytes
)"     record: "rate_coredns_demo"</Data>
</GetRecordingRuleResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "Data" : "--- groups: - name: \"recording_demo\"   rules:   - expr: \"sum(jvm_memory_max_
bytes)\"     record: \"rate_coredns_demo\""
}
```

错误码

访问错误中心查看更多错误码。

16.1.16. AddRecordingRule

调用AddRecordingRule接口创建或者更新RecordingRule规则。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AddRecordingRule	系统规定参数。取值：**AddRecordingRule**。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	集群ID。
RuleYaml	String	是	groups: - name: "recording_demo" rules: - expr: "sum(jvm_memory_max_bytes)" record: "rate_coredns_demo"	自定义的RecordingRule聚合规则。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	9FEA6D00-317F-45E3-9004-7FB8B0B7****	请求ID。
Data	String	success	响应状态。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AddRecordingRule
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&RuleYaml=groups: - name: "recording_demo"   rules:   - expr: "sum(jvm_memory_max_bytes)"
record: "rate_coredns_demo"
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<AddRecordingRuleResponse>
    <RequestId>9FEA6D00-317F-45E3-9004-7FB8B0B7****</RequestId>
    <Data>success</Data>
</AddRecordingRuleResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "9FEA6D00-317F-45E3-9004-7FB8B0B7****",
  "Data" : "success"
}
```

错误码

访问错误中心查看更多错误码。

16.1.17. GetAuthToken

调用GetAuthToken接口获取公网读写鉴权Token。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetAuthToken	系统规定参数。取值：**GetAuthToken**。
ClusterId	String	是	cc7a37ee31aea4ed1a059eff8034b****	阿里云容器服务Kubernetes版的Kubernetes集群的ID。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
Data	String	success	操作是否成功。
RequestId	String	1A9C645C-C83F-4C9D-8CCB-29BEC9E1****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetAuthToken
&ClusterId=cc7a37ee31aea4ed1a059eff8034b****
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<GetAuthTokenResponse>
    <Data>success</Data>
    <RequestId>1A9C645C-C83F-4C9D-8CCB-29BEC9E1****</RequestId>
</GetAuthTokenResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : "success",
  "RequestId" : "1A9C645C-C83F-4C9D-8CCB-29BEC9E1****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.18. AddPrometheusGlobalViewByAliClusterIds

调用AddPrometheusGlobalViewByAliClusterIds接口增加ARMS Prometheus监控的聚合实例。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AddPrometheusGlobalViewByAliClusterIds	系统规定参数。取值：**AddPrometheusGlobalViewByAliClusterIds**。
RegionId	String	是	cn-hangzhou	地域ID。
GroupName	String	是	zyGlobalView	聚合实例名称。
ClusterIds	String	是	cd1d55bef19904324a20ed0ebb86caa5c,c5b48729918ab4745a24482ac29d0973a,c00a94896641449098bf24931e4166003,cd174485c09384060ba542bc1be1185a4	集群ID列表，可以是多个，需要用英文逗号（,）分隔。

返回数据

名称	类型	示例值	描述
Data	Object		返回结构体。
Success	Boolean	true	查询是否成功： • `true`：成功。 • `false`：失败。
Msg	String	success	附加说明信息。
Info	String	{regionId: 实例所属region,globalViewClusterId: 实例Id,failedClusterIds: 添加失败的AliClusterId（一般是clusterId有误，或者跨大洲添加）}	Info级别信息。
RequestId	String	3A0EA2AF-C9B3-555C-B9D5-5DD8F5EF98A9	请求的ID。用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AddPrometheusGlobalViewByAliClusterIds
&RegionId=cn-hangzhou
&GroupName=zyGlobalView
&ClusterIds=cd1d55bef19904324a20ed0ebb86caa5c,c5b48729918ab4745a24482ac29d0973a, c00a948966
41449098bf24931e4166003, cd174485c09384060ba542bc1be1185a4
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<AddPrometheusGlobalViewByAliClusterIdsResponse>
    <Data>
        <Success>true</Success>
        <Msg>success</Msg>
        <Info>{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加
失败的AliClusterId（一般是clusterId有误，或者跨大洲添加）}</Info>
    </Data>
    <RequestId>3A0EA2AF-C9B3-555C-B9D5-5DD8F5EF98A9</RequestId>
</AddPrometheusGlobalViewByAliClusterIdsResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : {
    "Success" : true,
    "Msg" : "success",
    "Info" : "{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加
失败的AliClusterId（一般是clusterId有误，或者跨大洲添加）}"
  },
  "RequestId" : "3A0EA2AF-C9B3-555C-B9D5-5DD8F5EF98A9"
}
```

错误码

访问错误中心查看更多错误码。

16.1.19. RemoveAliClusterIdsFromPrometheusGlobalView

调用RemoveAliClusterIdsFromPrometheusGlobalView接口移除ARMS Prometheus监控聚合实例的某些数据源。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	RemoveAliClusterIdsFromPrometheusGlobalView	系统规定参数。取值：**RemoveAliClusterIdsFromPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GroupName	String	是	zyGlobalView	聚合实例名称。
GlobalViewClusterId	String	是	global-v2-cn-1670100631025794-amaykca4	聚合实例ID。
ClusterIds	String	是	cd1d55bef19904324a20ed0ebb86caa5c,c5b48729918ab4745a24482ac29d0973a,c00a94896641449098bf24931e4166003,cd174485c09384060ba542bc1be1185a4	集群ID列表，可以是多个，需要用英文逗号（,）分隔。

返回数据

名称	类型	示例值	描述
Data	Object		返回结构体。
Success	Boolean	true	操作是否成功： • `true`：操作成功 • `false`：操作失败
Msg	String	OK	附加说明信息。
Info	String	{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加失败的AliClusterId（一般是clusterId有误）}	Info级别信息。
RequestId	String	F7781D4A-2818-41E7-B7BB-79D809E9****	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=RemoveAliClusterIdsFromPrometheusGlobalView
&RegionId=cn-hangzhou
&GroupName=zyGlobalView
&GlobalViewClusterId=global-v2-cn-1670100631025794-amaykca4
&ClusterIds=cd1d55bef19904324a20ed0ebb86caa5c,c5b48729918ab4745a24482ac29d0973a, c00a948966
41449098bf24931e4166003, cd174485c09384060ba542bc1be1185a4
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<RemoveAliClusterIdsFromPrometheusGlobalViewResponse>
    <Data>
        <Success>true</Success>
        <Msg>OK</Msg>
        <Info>{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加
失败的AliClusterId（一般是clusterId有误）}</Info>
    </Data>
    <RequestId>F7781D4A-2818-41E7-B7BB-79D809E9****</RequestId>
</RemoveAliClusterIdsFromPrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : {
    "Success" : true,
    "Msg" : "OK",
    "Info" : "{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加
失败的AliClusterId（一般是clusterId有误）}"
  },
  "RequestId" : "F7781D4A-2818-41E7-B7BB-79D809E9****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.20. AddAliClusterIdsToPrometheusGlobalView

调用AddAliClusterIdsToPrometheusGlobalView接口增加ARMS Prometheus监控聚合实例的数据源。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AddAliClusterIdsToPrometheusGlobalView	系统规定参数。取值：**AddAliClusterIdsToPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GroupName	String	是	zyGlobalView	聚合实例名称。
GlobalViewClusterId	String	是	global-v2-cn-1478326682034601-vss8pd0i	聚合实例ID。
ClusterIds	String	是	cd1d55bef19904324a20ed0ebb86caa5c,c5b48729918ab4745a24482ac29d0973a,c00a94896641449098bf24931e4166003,cd174485c09384060ba542bc1be1185a4	集群ID列表，可以是多个，需要用英文逗号（,）分隔。

返回数据

名称	类型	示例值	描述
Data	Object		返回结构体。
Success	Boolean	true	是否删除成功。 • `true`：成功 • `false`：失败
Msg	String	OK	附加说明信息。
Info	String	{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加失败的AliClusterId（一般是clusterId有误，或者跨大洲添加）}	Info级别信息。

名称	类型	示例值	描述
RequestId	String	F7781D4A-2818-41E7-B7BB-79D809E9****	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AddAliClusterIdsToPrometheusGlobalView
&RegionId=cn-hangzhou
&GroupName=zyGlobalView
&GlobalViewClusterId=global-v2-cn-1478326682034601-vss8pd0i
&ClusterIds=cd1d55bef19904324a20ed0ebb86caa5c,c5b48729918ab4745a24482ac29d0973a, c00a948966
41449098bf24931e4166003, cd174485c09384060ba542bc1be1185a4
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<AddAliClusterIdsToPrometheusGlobalViewResponse>
    <Data>
        <Success>true</Success>
        <Msg>OK</Msg>
        <Info>{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加
失败的AliClusterId（一般是clusterId有误，或者跨大洲添加）}</Info>
    </Data>
    <RequestId>F7781D4A-2818-41E7-B7BB-79D809E9****</RequestId>
</AddAliClusterIdsToPrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : {
    "Success" : true,
    "Msg" : "OK",
    "Info" : "{regionId: 实例所属region, globalViewClusterId: 实例Id, failedClusterIds: 添加
失败的AliClusterId（一般是clusterId有误，或者跨大洲添加）}"
  },
  "RequestId" : "F7781D4A-2818-41E7-B7BB-79D809E9****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.21. AddPrometheusGlobalView

调用AddPrometheusGlobalView接口增加ARMS Prometheus监控的聚合实例。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AddPrometheusGlobalView	系统规定参数。取值：**AddPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GroupName	String	是	zyGlobalView	聚合实例名称。
Clusters	String	是	[{ "sourceName": "数据源名称-ArmsPrometheus No.1", "sourceType":"AlibabaPrometheus", "userId":"UserID", "clusterId":"ClusterId", }, { "sourceName": "数据源名称 - MetrcStore No.2", "sourceType":"MetricStore", "dataSource":"MetricStore的 remote read 地址", "extras":{ "username":"BasicAuthUsername", "password":"BasicAuthPassword" } }, { "sourceName": "Custom ", "sourceType":"CustomPrometheus", "dataSource":"自建Prometheus数据源 remoteread地址", "extras":{ "username":"BasicAuthUsername", "password":"Basi	聚合实例列表，为JSON格式字符串。

名称	类型	是否必选	示例值	描述
			cAuthPassword" } }, { "sourceName": "Other one ", "sourceType":"Others", "dataSource":"其他数据源如Tencent remoteread地址", "headers":{ "AnyHeaderToFill":"需要填充的Headers" }, "extras":{ "username":"BasicAuthUsername", "password":"BasicAuthPassword" } } // ..... more addre]	

返回数据

名称	类型	示例值	描述
Data	Object		返回结构体。
Success	Boolean	true	查询是否成功。 • `true`：成功。 • `false`：失败。
Msg	String	OK	附加说明信息。
Info	String	{regionId: 实例所属region, globalViewClusterId: 实例Id, failedInstances: 数据源json list中，添加失败的单个json的list}	Info级别信息。
RequestId	String	34ED024E-9E31-434A-9E4E-D9D15C3****	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AddPrometheusGlobalView
&RegionId=cn-hangzhou
&GroupName=zyGlobalView
&Clusters=[         {             "sourceName": "数据源名称- ArmsPrometheus No.1",
"sourceType":"AlibabaPrometheus",             "userId":"UserID",             "clusterId":"C
lusterId",         },         {             "sourceName": "数据源名称 - MetrcStore No.2",
"sourceType":"MetricStore",             "dataSource":"MetricStore的 remote read 地址",
"extras":{                 "username":"BasicAuthUsername",                 "password":"Basi
cAuthPassword"             }         },         {             "sourceName": "Custom ",
"sourceType":"CustomPrometheus",             "dataSource":"自建Prometheus数据源 remoteread地
址",             "extras":{                 "username":"BasicAuthUsername",
"password":"BasicAuthPassword"             }         },         {             "sourceName":
"Other one ",             "sourceType":"Others",             "dataSource":"其他数据源如Tencen
t remoteread地址",             "headers":{                 "AnyHeaderToFill":"需要填充的Heade
rs"             },             "extras":{                 "username":"BasicAuthUsername",
"password":"BasicAuthPassword"             }         }   // ....... more addre ]
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<AddPrometheusGlobalViewResponse>
    <Data>
        <Success>true</Success>
        <Msg>OK</Msg>
        <Info>{regionId: 实例所属region, globalViewClusterId: 实例Id, failedInstances: 数据源
json list中，添加失败的单个json的list}</Info>
    </Data>
    <RequestId>34ED024E-9E31-434A-9E4E-D9D15C3****</RequestId>
</AddPrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : {
    "Success" : true,
    "Msg" : "OK",
    "Info" : "{regionId: 实例所属region, globalViewClusterId: 实例Id, failedInstances: 数据源
json list中，添加失败的单个json的list}"
  },
  "RequestId" : "34ED024E-9E31-434A-9E4E-D9D15C3****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.22. GetPrometheusGlobalView

调用GetPrometheusGlobalView接口增加ARMS Prometheus监控的聚合实例，获取指定聚合实例的详细数据源情况。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetPrometheusGlobalView	系统规定参数。取值：**GetPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GlobalViewClusterId	String	是	global-v2-cn-1478326682034601-vss8pd0i	聚合实例ID。

返回数据

名称	类型	示例值	描述
Data	String	{ "clusterId":"聚合实例ClusterId", "groupName":"聚合实例名称", "dataSources":[{ "sourceName":"数据源名称-ArmsPrometheus No.1", "sourceType":"AlibabaPrometheus", "userId":"UserID", "clusterId":"ClusterId" }, // more datasources] }	返回结构体。
RequestId	String	743AD493-D006-53BD-AAEC-DDCE7FB68EA7	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetPrometheusGlobalView
&RegionId=cn-hangzhou
&GlobalViewClusterId=global-v2-cn-1478326682034601-vss8pd0i
&公共请求参数
```

正常返回示例

XML 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<GetPrometheusGlobalViewResponse>
    <Data>{     "clusterId":"聚合实例ClusterId",     "groupName":"聚合实例名称",     "dataSour
ces":[         {             "sourceName":"数据源名称- ArmsPrometheus No.1",             "so
urceType":"AlibabaPrometheus",             "userId":"UserID",             "clusterId":"Clus
terId"         },       // more datasources     ] }</Data>
    <RequestId>743AD493-D006-53BD-AAEC-DDCE7FB68EA7</RequestId>
</GetPrometheusGlobalViewResponse>
```

JSON 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : "{     \"clusterId\":\"聚合实例ClusterId\",     \"groupName\":\"聚合实例名称\",
\"dataSources\":[         {             \"sourceName\":\"数据源名称- ArmsPrometheus No.1\",
\"sourceType\":\"AlibabaPrometheus\",             \"userId\":\"UserID\",             \"clus
terId\":\"ClusterId\"         },       // more datasources     ] }",
  "RequestId" : "743AD493-D006-53BD-AAEC-DDCE7FB68EA7"
}
```

错误码

访问错误中心查看更多错误码。

16.1.23. ListPrometheusGlobalView

调用ListPrometheusGlobalView接口增加ARMS Prometheus监控的聚合实例，获取聚合实例列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	ListPrometheusGlobalView	系统规定参数。取值：**ListPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述

名称	类型	示例值	描述
Data	String	[{groupName: "聚合实例名称", clusterId: "global-v2-clusterid", endpoint: "cn-hangzhou"}, // ..... more items]	聚合实例列表，为JSON格式字符串。
RequestId	String	DBDCE95A-A0DD-5FC5-97CC-EEFC3D814385	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=ListPrometheusGlobalView
&RegionId=cn-hangzhou
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<ListPrometheusGlobalViewResponse>
    <Data>[   {groupName: "聚合实例名称", clusterId: "global-v2-clusterid", endpoint: "cn-han
gzhou"}, //   ..... more items ]</Data>
    <RequestId>DBDCE95A-A0DD-5FC5-97CC-EEFC3D814385</RequestId>
</ListPrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : "[   {groupName: \"聚合实例名称\", clusterId: \"global-v2-clusterid\", endpoint: \
"cn-hangzhou\"}, //   ..... more items ]",
  "RequestId" : "DBDCE95A-A0DD-5FC5-97CC-EEFC3D814385"
}
```

错误码

访问错误中心查看更多错误码。

16.1.24. DeletePrometheusGlobalView

调用DeletePrometheusGlobalView接口删除Prometheus聚合实例。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeletePrometheusGlobalView	系统规定参数。取值：**DeletePrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GlobalViewClusterId	String	是	global-v2-cn-1670100631025794-amaykca4	聚合实例ID。

返回数据

名称	类型	示例值	描述
Data	String	{"Success":true,"Msg":"OK"}	JSON格式的返回结果。
RequestId	String	337B8F7E-0A64-5768-9225-E9B3CF******	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeletePrometheusGlobalView
&RegionId=cn-hangzhou
&GlobalViewClusterId=global-v2-cn-1670100631025794-amaykca4
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<DeletePrometheusGlobalViewResponse>
    <Data>{"Success":true,"Msg":"OK"}</Data>
    <RequestId>337B8F7E-0A64-5768-9225-E9B3CF******</RequestId>
</DeletePrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : "{\"Success\":true,\"Msg\":\"OK\"}",
  "RequestId" : "337B8F7E-0A64-5768-9225-E9B3CF******"
}
```

错误码

访问错误中心查看更多错误码。

16.1.25. AppendInstancesToPrometheusGlobalView

调用AppendInstancesToPrometheusGlobalView接口增加ARMS Prometheus监控聚合实例中的数据源，将数据源添加到Prometheus聚合实例。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	AppendInstancesToPrometheusGlobalView	系统规定参数。取值：**AppendInstancesToPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GroupName	String	是	zyGlobalView	聚合实例名称。
			[{ "sourceName": "数据源名称-ArmsPrometheus No.1", "sourceType":"AlibabaPrometheus", "userId":"UserID", "clusterId":"ClusterId", }, { "sourceName": "数据源名称 - MetrcStore No.2", "sourceType":"MetricStore", "dataSource":"MetricStore的 remote read 地址", "extras":{ "username":"Basi	

名称	类型	是否必选	示例值	描述
Clusters	String	是	cAuthUsername", "password":"BasicAuthPassword" } }, { "sourceName": "Custom ", "sourceType":"CustomPrometheus ", "dataSource":"自建Prometheus数据源 remoteread地址", "extras":{ "username":"BasicAuthUsername", "password":"BasicAuthPassword" } }, { "sourceName": "Other one ", "sourceType":"Others", "dataSource":"其他数据源如Tencent remoteread地址", "headers":{ "AnyHeaderToFill":"需要填充的Headers" }, "extras":{ "username":"BasicAuthUsername", "password":"BasicAuthPassword" } } // ..... more addre]	聚合实例列表，为JSON格式的字符串。
GlobalViewClusterId	String	是	global-v2-cn-1670100631025794-6gjc0qgb	聚合实例ID。

返回数据

名称	类型	示例值	描述
Data	Object		返回结构体。
Success	Boolean	True	操作是否成功： • `true`：操作成功 • `false`：操作失败
Msg	String	OK	附加说明信息。

名称	类型	示例值	描述
Info	String	{regionId: 实例所属region, globalViewClusterId: 实例Id, failedInstances: 数据源json list中，添加失败的单个json的list}	Info级别信息。
RequestId	String	27E653FA-5958-45BE-8AA9-14D884DC****	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=AppendInstancesToPrometheusGlobalView
&RegionId=cn-hangzhou
&GroupName=zyGlobalView
&Clusters=[         {             "sourceName": "数据源名称- ArmsPrometheus No.1",
"sourceType":"AlibabaPrometheus",             "userId":"UserID",             "clusterId":"C
lusterId",         },         {             "sourceName": "数据源名称 - MetrcStore No.2",
"sourceType":"MetricStore",             "dataSource":"MetricStore的 remote read 地址",
"extras":{                 "username":"BasicAuthUsername",                 "password":"Basi
cAuthPassword"             }         },         {             "sourceName": "Custom ",
"sourceType":"CustomPrometheus",             "dataSource":"自建Prometheus数据源 remoteread地
址",             "extras":{                 "username":"BasicAuthUsername",
"password":"BasicAuthPassword"             }         },         {             "sourceName":
"Other one ",             "sourceType":"Others",             "dataSource":"其他数据源如Tencen
t remoteread地址",             "headers":{                 "AnyHeaderToFill":"需要填充的Heade
rs"             },             "extras":{                 "username":"BasicAuthUsername",
"password":"BasicAuthPassword"             }         }   // ....... more addre ]
&GlobalViewClusterId=global-v2-cn-1670100631025794-6gjc0qgb
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<AppendInstancesToPrometheusGlobalViewResponse>
    <Data>
        <Success>true</Success>
        <Msg>OK</Msg>
        <Info>{regionId: 实例所属region, globalViewClusterId: 实例Id, failedInstances: 数据源
json list中，添加失败的单个json的list}</Info>
    </Data>
    <RequestId>27E653FA-5958-45BE-8AA9-14D884DC****</RequestId>
</AppendInstancesToPrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : {
    "Success" : true,
    "Msg" : "OK",
    "Info" : "{regionId: 实例所属region, globalViewClusterId: 实例Id, failedInstances: 数据源json list中，添加失败的单个json的list}"
  },
  "RequestId" : "27E653FA-5958-45BE-8AA9-14D884DC****"
}
```

错误码

访问错误中心查看更多错误码。

16.1.26. RemoveSourcesFromPrometheusGlobalView

调用RemoveSourcesFromPrometheusGlobalView接口移除ARMS Prometheus监控聚合实例中的数据源，仅支持删除非阿里数据源。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	RemoveSourcesFromPrometheusGlobalView	系统规定参数。取值：**RemoveSourcesFromPrometheusGlobalView**。
RegionId	String	是	cn-hangzhou	地域ID。
GroupName	String	是	zyGlobalView	聚合实例名称
GlobalViewClusterId	String	是	global-v2-cn-1478326682034601-vss8pd0i	聚合实例ID。
SourceNames	String	是	localPrometheusClusterName,testCumterPrometheusName	自定义数据源的SourceName列表，可以是多个，需要用英文逗号（,）分隔。

返回数据

名称	类型	示例值	描述
Data	Object		返回结构体。
Success	Boolean	True	操作是否成功： • `true`：操作成功 • `false`：操作失败
Msg	String	OK	附加说明信息。
Info	String	{regionId: 实例所属region, globalViewClusterId: 实例Id}	Info级别信息。
RequestId	String	9319A57D-2D9E-472A-B69B-CF3CD16D****	请求ID，用于定位日志，排查问题。

示例

请求示例

```
http(s)://[Endpoint]/?Action=RemoveSourcesFromPrometheusGlobalView
&RegionId=cn-hangzhou
&GroupName=zyGlobalView
&GlobalViewClusterId=global-v2-cn-1478326682034601-vss8pd0i
&SourceNames=localPrometheusClusterName,testCumterPrometheusName
&公共请求参数
```

正常返回示例

`XML` 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<RemoveSourcesFromPrometheusGlobalViewResponse>
    <Data>
        <Success>true</Success>
        <Msg>OK</Msg>
        <Info>{regionId: 实例所属region, globalViewClusterId: 实例Id}</Info>
    </Data>
    <RequestId>9319A57D-2D9E-472A-B69B-CF3CD16D****</RequestId>
</RemoveSourcesFromPrometheusGlobalViewResponse>
```

`JSON` 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : {
    "Success" : true,
    "Msg" : "OK",
    "Info" : "{regionId: 实例所属region, globalViewClusterId: 实例Id}"
  },
  "RequestId" : "9319A57D-2D9E-472A-B69B-CF3CD16D****"
}
```

错误码

访问错误中心查看更多错误码。

16.2. 报警

16.2.1. CreateDispatchRule

调用CreateDispatchRule接口创建分派策略。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateDispatchRule	系统规定参数。取值：CreateDispatchRule。

名称	类型	是否必选	示例值	描述
DispatchRule	String	是	{ "system": false, "ruleid": 10282, "name": "Prometheus Alert", "labelMatchExpressionGrid": { "labelMatchExpressionGroups": [{ "labelMatchExpressions": [{ "key": "_aliyun_arms_involvedObject_kind", "value": "app", "operator": "eq" }] }] }, "dispatchType": "CREATE_ALERT/DISCARD_ALERT", "isRecover": true, "groupRules": [{ "groupId": 1, "groupingFields": ["alertname"], "groupWait": 10, "groupInterval": 15, "repeatInterval": 20 }], "notifyRules": [{ "notifyObjects": [{ "notifyType": "ARMS_CONTACT", "name": "JohnDoe", "notifyObjectId": 1 }, { "notifyType": "ARMS_CONTACT_GROUP", "name": "JohnDoe_group", "notifyObjectId": 2 }], "notifyChannels": ["dingTalk","wechat","webhook","email"] },], }	分派条件的配置JSON串。关于此字段的详细说明参见下文**关于参数DispatchRule的补充说明**。
RegionId	String	是	cn-hangzhou	地域ID。

关于参数DispatchRule的补充说明

JSON串示例及说明

```
{
  "system": false,         //分派条件是否可编辑。true：不可编辑；false：可编辑。
  "ruleid": 10282,           //分派规则ID。
  "name": "Prometheus Alert",    //分派策略名称。
  "labelMatchExpressionGrid": {
    "labelMatchExpressionGroups": [      //设置分派条件。
      {
        "labelMatchExpressions": [
          {
            "key": "_aliyun_arms_involvedObject_kind",    //分派条件标签，详见下一节。
            "value": "app",                               //标签取值。
            "operator": "eq"                              //eq：等于；re：匹配正则。
          }
        ]
      }
    ]
  },
  "dispatchType": "CREATE_ALERT/DISCARD_ALERT",     //告警处理方式。CREATE_ALERT：就是生成报警；
DISCARD_ALERT：丢弃报警事件，即不告警。
  "isRecover": true,                //是否发送恢复的告警。true：发送；false：不发送。
  "groupRules": [                   //设置事件分组。
    {
      "groupId": 1,                //分组ID。
      "groupingFields": [          //指定相同字段内容的事件分到一个组：设置分组字段，相同字段的告警
内容会分别通过独立信息发送给处理人。
        "alertname"
      ],
      "groupWait": 10,                    //分组等待时间：收到第一个告警后会等待设置的时间，等待分组
时间后收到的所有告警会以一条信息发送给处理人。
      "groupInterval": 15,                //分组间隔时间：在重复告警静默时间内，如果有新告警产生，等
待设置的时间后就会直接发送新的告警信息。
      "repeatInterval": 20                //重复告警静默时间：所有告警会以设置的时间间隔循环发送告警
信息直至告警消失。
    }
  ],
  "notifyRules": [             //设置通知规则。
    {
      "notifyObjects": [
        {
          "notifyType": "ARMS_CONTACT",      //ARMS_CONTACT：联系人；ARMS_CONTACT_GROUP：联系
人组。
          "name": "JohnDoe",                 //联系人或联系人组的名称。
          "notifyObjectId": 1                //联系人或联系人组的ID。
        },
        {
          "notifyType": "ARMS_CONTACT_GROUP",
          "name": "JohnDoe_group",
          "notifyObjectId": 2
        }
      ],
      "notifyChannels":["dingTalk","wechat","webhook","email"]      //通知方式：dingTalk（钉
钉）、sms（短信）、webhook、email（邮件）、wechat（微信）。
    },
  ],
```

```
}
```

分派标签取值枚举

- `_aliyun_arms_userid`：用户ID
- `_aliyun_arms_involvedObject_kind`：关联对象类型
- `_aliyun_arms_involvedObject_id`：关联对象ID
- `_aliyun_arms_involvedObject_name`：关联对象名称
- `_aliyun_arms_alert_name`：告警名称
- `_aliyun_arms_alert_rule_id`：告警规则对应的ID
- `_aliyun_arms_alert_type`：告警类型
- `_aliyun_arms_alert_level`：告警等级

返回数据

名称	类型	示例值	描述
DispatchRuleId	Long	10413	分派策略ID。
RequestId	String	A5EC8221-08F2-4C95-9AF1-49FD998C****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateDispatchRule
&DispatchRule=
{
  "system": false,
  "ruleid": 10282,
  "name": "Prometheus Alert",
  "labelMatchExpressionGrid": {
    "labelMatchExpressionGroups": [
      {
        "labelMatchExpressions": [
          {
            "key": "_aliyun_arms_involvedObject_kind",
            "value": "app",
            "operator": "eq"
          }
        ]
      }
    ]
  },
  "dispatchType": "CREATE_ALERT/DISCARD_ALERT",
  "isRecover": true,
  "groupRules": [
    {
      "groupId": 1,
      "groupingFields": [
        "alertname"
      ],
      "groupWait": 10,
      "groupInterval": 15,
      "repeatInterval": 20
    }
  ],
  "notifyRules": [
    {
      "notifyObjects": [
        {
          "notifyType": "ARMS_CONTACT",
          "name": "JohnDoe",
          "notifyObjectId": 1
        },
        {
          "notifyType": "ARMS_CONTACT_GROUP",
          "name": "JohnDoe_group",
          "notifyObjectId": 2
        }
      ],
      "notifyChannels":["dingTalk","wechat","webhook","email"]
    },
  ],
}
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<CreateDispatchRuleResponse>
      <RequestId>A5EC8221-08F2-4C95-9AF1-49FD998C****</RequestId>
      <DispatchRuleId>10413</DispatchRuleId>
</CreateDispatchRuleResponse>
```

`JSON` 格式

```
{
    "RequestId": "A5EC8221-08F2-4C95-9AF1-49FD998C****",
    "DispatchRuleId": 10413
}
```

错误码

访问错误中心查看更多错误码。

16.2.2. DescribeDispatchRule

调用DescribeDispatchRule接口查询分派策略信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DescribeDispatchRule	系统规定参数。取值：DescribeDispatchRule。
RegionId	String	是	cn-hangzhou	地域ID。
Id	String	否	12345	分派策略ID。

返回数据

名称	类型	示例值	描述
DispatchRule	Struct		返回结构体。
GroupRules	Array of GroupRule		事件分组。
GroupId	Long	1	分组ID。
GroupInterval	Long	15	分组间隔时间。
GroupWaitTime	Long	10	分组等待时间。

名称	类型	示例值	描述
GroupingFields	List	_aliyun_arms_involvedObject_kind	分组字段列表。
LabelMatchExpressionGrid	Struct		分派规则。
LabelMatchExpressionGroups	Array of LabelMatchExpressionGroup		分派条件集合。
LabelMatchExpressions	Array of LabelMatchExpression		分派规则的条件。
Key	String	_aliyun_arms_involvedObject_kind	分派条件标签： • `_aliyun_arms_userid`：用户ID • `_aliyun_arms_involvedObject_kind`：关联对象类型 • `_aliyun_arms_involvedObject_id`：关联对象ID • `_aliyun_arms_involvedObject_name`：关联对象名称 • `_aliyun_arms_alert_name`：告警名称 • `_aliyun_arms_alert_rule_id`：告警规则对应的ID • `_aliyun_arms_alert_type`：告警类型 • `_aliyun_arms_alert_level`：告警等级
Operator	String	eq	选项： • `eq`：等于 • `re`：匹配正则
Value	String	app	标签取值。
Name	String	Prometheus Alert	分派策略名称。
NotifyRules	Array of NotifyRule		通知方式集合。
NotifyChannels	List	email	通知方式： • `dingTalk` • `sms` • `webhook` • `email` • `wechat`

名称	类型	示例值	描述
NotifyObjects	Array of NotifyObject		通知对象集合。
Name	String	JohnDoe	联系人或联系人组的名称。
NotifyObjectId	String	1	联系人或联系人组的ID。
NotifyType	String	CONTACT	通知对象类型： • `CONTACT`：联系人 • `CONTACT_GROUP`：联系人组
RuleId	Long	10282	分派规则ID。
State	String	true	是否启用该分派策略。 • `true`：启用 • `false`：关闭
RequestId	String	34ED024E-9E31-434A-9E4E-D9D15C3****	请求ID。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DescribeDispatchRule
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DescribeDispatchRuleResponse>
  <RequestId>34ED024E-9E31-434A-9E4E-D9D15C3****</RequestId>
  <DispatchRule>
        <GroupRules>
              <GroupInterval>15</GroupInterval>
              <GroupWaitTime>10</GroupWaitTime>
              <GroupId>1</GroupId>
        </GroupRules>
        <GroupRules>
              <GroupingFields>_aliyun_arms_involvedObject_kind</GroupingFields>
        </GroupRules>
        <State>true</State>
        <RuleId>10282</RuleId>
        <LabelMatchExpressionGrid>
              <LabelMatchExpressionGroups>
                    <LabelMatchExpressions>
                          <Operator>eq</Operator>
                          <Value>app</Value>
                          <Key>_aliyun_arms_involvedObject_kind</Key>
                    </LabelMatchExpressions>
              </LabelMatchExpressionGroups>
        </LabelMatchExpressionGrid>
        <NotifyRules>
              <NotifyObjects>
                    <NotifyType>CONTACT</NotifyType>
                    <NotifyObjectId>1</NotifyObjectId>
                    <Name>JohnDoe</Name>
              </NotifyObjects>
        </NotifyRules>
        <NotifyRules>
              <NotifyChannels>email</NotifyChannels>
        </NotifyRules>
        <Name>Prometheus Alert</Name>
  </DispatchRule>
</DescribeDispatchRuleResponse>
```

`JSON` 格式

```
{
    "RequestId": "34ED024E-9E31-434A-9E4E-D9D15C3****",
    "DispatchRule": {
        "GroupRules": [
            {
                "GroupInterval": 15,
                "GroupWaitTime": 10,
                "GroupId": 1
            },
            {
                "GroupingFields": "_aliyun_arms_involvedObject_kind"
            }
        ],
        "State": true,
        "RuleId": 10282,
        "LabelMatchExpressionGrid": {
            "LabelMatchExpressionGroups": {
                "LabelMatchExpressions": {
                    "Operator": "eq",
                    "Value": "app",
                    "Key": "_aliyun_arms_involvedObject_kind"
                }
            }
        },
        "NotifyRules": [
            {
                "NotifyObjects": {
                    "NotifyType": "CONTACT",
                    "NotifyObjectId": 1,
                    "Name": "JohnDoe"
                }
            },
            {
                "NotifyChannels": "email"
            }
        ],
        "Name": "Prometheus Alert"
    }
}
```


错误码

访问错误中心查看更多错误码。

16.2.3. UpdateDispatchRule

调用UpdateDispatchRule接口修改分派策略。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	UpdateDispatchRule	系统规定参数。取值：UpdateDispatchRule。
DispatchRule	String	是	{ "id": 123, "system": false, "ruleid": 10282, "name": "Prometheus Alert", "labelMatchExpressionGrid": { "labelMatchExpressionGroups": [{ "labelMatchExpressions": [{ "key": "_aliyun_arms_involvedObject_kind", "value": "app", "operator": "eq" }] }] }, "dispatchType": "CREATE_ALERT/DISCARD_ALERT", "isRecover": true, "groupRules": [{ "groupId": 1, "groupingFields": ["alertname"], "groupWait": 10, "groupInterval": 15, "repeatInterval": 20 }], "notifyRules": [{ "notifyObjects": [{ "notifyType": "ARMS_CONTACT", "name": "JohnDoe", "notifyObjectId": 1 }, { "notifyType": "ARMS_CONTACT_GROUP", "name": "JohnDoe_group", "notifyObjectId": 2 }], "notifyChannels": ["dingTalk","wechat","webhook","email"] },], }	分派条件的配置JSON串。关于此字段的详细说明参见下文**关于参数DispatchRule的补充说明**。

名称	类型	是否必选	示例值	描述
RegionId	String	是	cn-hangzhou	地域ID。

关于参数DispatchRule的补充说明

JSON串示例及说明

```
{
  "id": 123,           //分派策略ID。
  "system": false,         //分派条件是否可编辑：true（不可编辑）、false（可编辑）。
  "ruleid": 10282,           //分派规则ID。
  "name": "Prometheus Alert",    //分派策略名称。
  "labelMatchExpressionGrid": {
    "labelMatchExpressionGroups": [      //设置分派条件。
      {
        "labelMatchExpressions": [
          {
            "key": "_aliyun_arms_involvedObject_kind",    //分派条件标签，详见下一节。
            "value": "app",                               //标签取值。
            "operator": "eq"                              //eq：等于；re：匹配正则。
          }
        ]
      }
    ]
  },
  "dispatchType": "CREATE_ALERT/DISCARD_ALERT",     //告警处理方式：CREATE_ALERT（就是生成报警）
，DISCARD_ALERT（丢弃报警事件，即不告警）
  "isRecover": true,                //是否发生恢复的告警。true（发生），false（不发送）。
  "groupRules": [                   //设置事件分组。
    {
      "groupId": 1,                //分组ID。
      "groupingFields": [          //指定相同字段内容的事件分到一个组：设置分组字段，相同字段的告警
内容会分别通过独立信息发送给处理人。
        "alertname"
      ],
      "groupWait": 10,                    //分组等待时间：收到第一个告警后会等待设置的时间，等待分组
时间后收到的所有告警会以一条信息发送给处理人。
      "groupInterval": 15,                //分组间隔时间：在重复告警静默时间内，如果有新告警产生，等
待设置的时间后就会直接发送新的告警信息。
      "repeatInterval": 20                //重复告警静默时间：所有告警会以设置的时间间隔循环发送告警
信息直至告警消失。
    }
  ],
  "notifyRules": [             //设置通知规则。
    {
      "notifyObjects": [
        {
          "notifyType": "ARMS_CONTACT",      //ARMS_CONTACT：联系人；ARMS_CONTACT_GROUP：联系
人组。
          "name": "JohnDoe",                 //联系人或联系人组的名称。
          "notifyObjectId": 1                //联系人或联系人组的ID。
        },
        {
          "notifyType": "ARMS_CONTACT_GROUP",
```

```
            "name": "JohnDoe_group",
            "notifyObjectId": 2
          }
        ],
        "notifyChannels":["dingTalk","wechat","webhook","email"]      //通知方式：dingTalk（钉钉）、sms（短信）、webhook、email（邮件）、wechat（微信）。
      },
    ],
}
```

分派标签取值枚举

- `_aliyun_arms_userid`：用户ID
- `_aliyun_arms_involvedObject_kind`：关联对象类型
- `_aliyun_arms_involvedObject_id`：关联对象ID
- `_aliyun_arms_involvedObject_name`：关联对象名称
- `_aliyun_arms_alert_name`：告警名称
- `_aliyun_arms_alert_rule_id`：告警规则对应的ID
- `_aliyun_arms_alert_type`：告警类型
- `_aliyun_arms_alert_level`：告警等级

返回数据

名称	类型	示例值	描述
RequestId	String	A5EC8221-08F2-4C95-9AF1-49FD998C****	请求ID。
Success	Boolean	true	分派策略是否修改成功。 • `true`：修改成功 • `false`：修改失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateDispatchRule
&DispatchRule=
{
  "id": 123,
   "system": false,
  "ruleid": 10282,
  "name": "Prometheus Alert",
  "labelMatchExpressionGrid": {
    "labelMatchExpressionGroups": [
      {
        "labelMatchExpressions": [
          {
            "key": "_aliyun_arms_involvedObject_kind",
            "value": "app",
            "operator": "eq"
          }
        ]
      }
    ]
  },
  "dispatchType": "CREATE_ALERT/DISCARD_ALERT",
  "isRecover": true,
  "groupRules": [
    {
      "groupId": 1,
      "groupingFields": [
        "alertname"
      ],
      "groupWait": 10,
      "groupInterval": 15,
      "repeatInterval": 20
    }
  ],
  "notifyRules": [
    {
      "notifyObjects": [
        {
          "notifyType": "ARMS_CONTACT",
          "name": "JohnDoe",
          "notifyObjectId": 1
        },
        {
          "notifyType": "ARMS_CONTACT_GROUP",
          "name": "JohnDoe_group",
          "notifyObjectId": 2
        }
      ],
      "notifyChannels":["dingTalk","wechat","webhook","email"]
    },
  ],
}
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<UpdateDispatchRuleResponse>
      <RequestId>A5EC8221-08F2-4C95-9AF1-49FD998C****</RequestId>
      <Success>true</Success>
</UpdateDispatchRuleResponse>
```

`JSON` 格式

```
{
    "RequestId": "A5EC8221-08F2-4C95-9AF1-49FD998C****",
    "Success": true
}
```

错误码

访问错误中心查看更多错误码。

16.2.4. DeleteDispatchRule

调用DeleteDispatchRule接口删除指定ID的分派策略。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteDispatchRule	系统规定参数。取值：DeleteDispatchRule。
Id	String	是	12345	分派策略ID。
RegionId	String	是	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	16AF921B-8187-489F-9913-43C808B4****	请求ID。
Success	Boolean	true	是否删除成功。 • `true` 删除成功 • `false` 删除失败

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteDispatchRule
&Id=12345
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DeleteDispatchRuleResponse>
      <RequestId>16AF921B-8187-489F-9913-43C808B4****</RequestId>
      <Success>true</Success>
</DeleteDispatchRuleResponse>
```

`JSON` 格式

```
{
    "RequestId": "16AF921B-8187-489F-9913-43C808B4****",
    "Success": true
}
```

错误码

访问错误中心查看更多错误码。

17.最佳实践

17.1. 监控VPC网络下ECS实例中的Java应用

本文介绍在VPC网络下的ECS实例接入阿里云Prometheus监控后，如何监控ECS实例中的Java应用。

前提条件

- 已创建Java应用。
  Spring Boot作为最主流的Java Web框架，在其生态中有着丰富的组件支持，可以通过Actuator和Micrometer很好的与阿里云Prometheus监控对接，因此，本文以Spring Boot Java应用为例。如果您还没有创建Java工程，可使用Java工程脚手架直接创建一个Java Maven Project。
- 已将VPC网络下ECS实例接入Prometheus监控。具体操作，请参见Prometheus实例 for VPC。

> **说明** 本文中的参数取值均为示例，您可以根据实际情况进行修改。

操作步骤

1. 在Project的*pom.xml*中添加以下依赖。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
   <version>2.3.7.RELEASE</version>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
  <version>1.5.1</version>
</dependency>
```

> **说明**
> - Spring Boot Actuator为Spring Boot提供了一系列产品级的特性：监控应用程序、收集元数据、运行情况或者数据库状态。更多详情，请参见Spring Boot文档。
> - Micrometer作为底层的度量工具，是监控度量的门面，相当于SLF4J在日志框架中的作用，其支持按照各种格式来暴露数据，包括Prometheus格式。

2. 修改Spring Boot配置文件。
   - 如果您的Spring Boot配置文件为application.properties格式，请参考以下命令修改配置。

```
# 应用名称
spring.application.name=demo
# 应用服务Web访问端口
server.port=8080
#可选配置
#management.endpoints.enabled-by-default=true
#management.endpoints.web.base-path=/actuator
#暴露Prometheus数据端点  /actuator/prometheus
management.endpoints.web.exposure.include=prometheus
#暴露的Prometheus数据中添加application label
management.metrics.tags.application=demo
```

- 如果您的Spring Boot配置文件为application.yml格式，请参考以下命令修改配置。

```
server:
  port: 8080
spring:
  application:
    name: spring-demo
management:
  endpoints:
    web:
      exposure:
        include: 'prometheus' # 暴露/actuator/prometheus
  metrics:
    tags:
      application: demo
```

3. 检查服务。

   启动应用后通过浏览器访问以下地址进行测试。

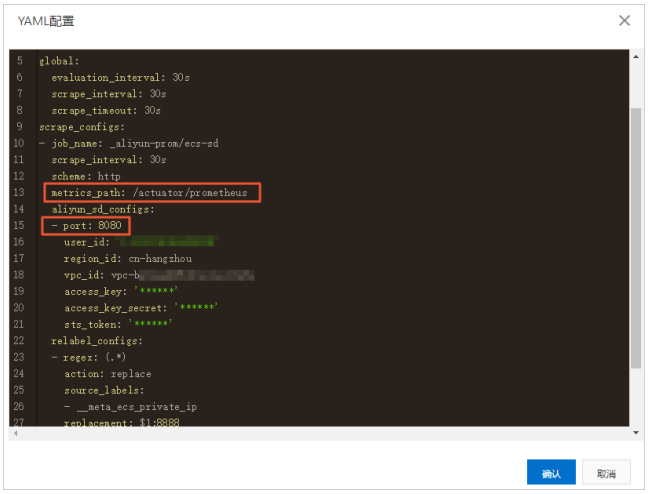
```
http://localhost:8080/actuator/prometheus
```

> **说明**
>
> - 请根据实际情况替换命令中的 `localhost` 和端口。
> - 建议在其他ECS实例上也进行地址测试（http://*[ECS IP]:port*/actuator/prometheus），检查连接是否通畅，避免被安全组限制。

   预计可得到以下返回结果。

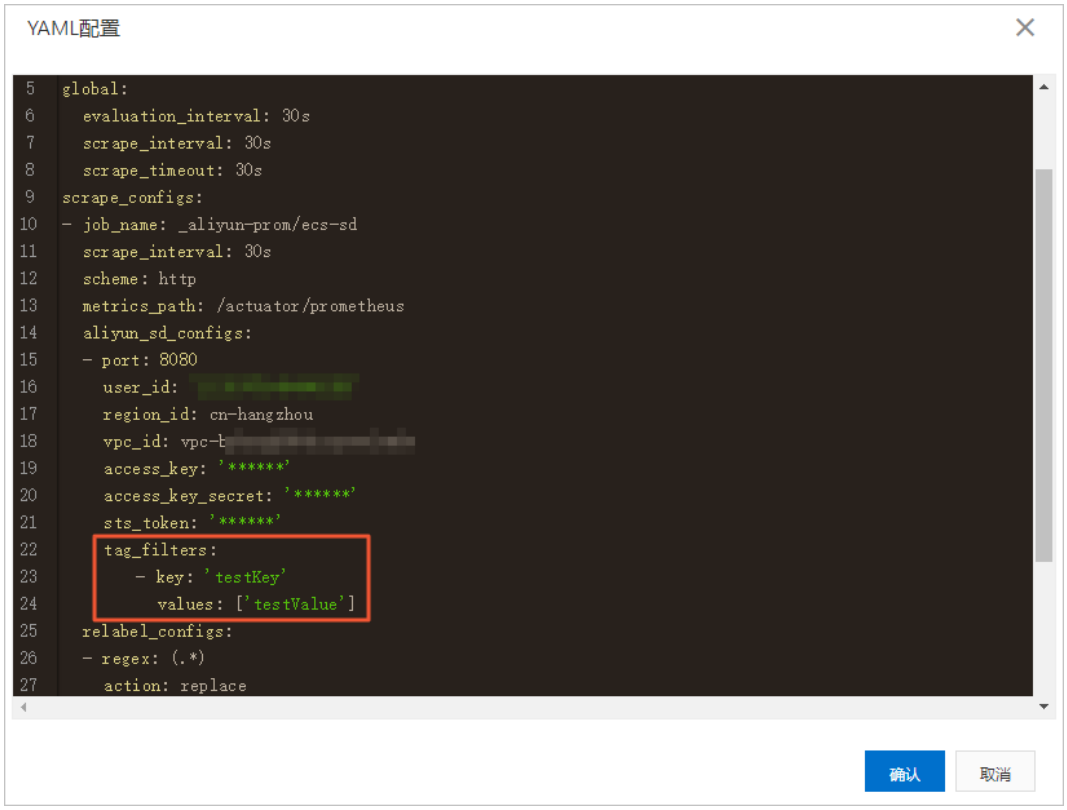
```
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for t
he Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{application="demo",area="heap",id="G1 Eden Space",} 1.300234
24E8
jvm_memory_committed_bytes{application="demo",area="heap",id="G1 Old Gen",} 1.28974848E
8
jvm_memory_committed_bytes{application="demo",area="nonheap",id="Metaspace",} 4.9627136
E7
jvm_memory_committed_bytes{application="demo",area="heap",id="G1 Survivor Space",} 9437
184.0
jvm_memory_committed_bytes{application="demo",area="nonheap",id="CodeHeap 'non-profiled
nmethods'",} 7077888.0
jvm_memory_committed_bytes{application="demo",area="nonheap",id="Compressed Class Space
",} 6680576.0
jvm_memory_committed_bytes{application="demo",area="nonheap",id="CodeHeap 'non-nmethods
'",} 2555904.0
# HELP jvm_threads_states_threads The current number of threads having NEW state
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{application="demo",state="waiting",} 11.0
jvm_threads_states_threads{application="demo",state="blocked",} 0.0
jvm_threads_states_threads{application="demo",state="timed-waiting",} 7.0
jvm_threads_states_threads{application="demo",state="runnable",} 14.0
jvm_threads_states_threads{application="demo",state="new",} 0.0
jvm_threads_states_threads{application="demo",state="terminated",} 0.0
```

4. 添加服务发现。
   i. 登录Prometheus控制台。
   ii. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域，单击目标VPC类型的Prometheus实例的名称。
   iii. 在左侧导航栏单击**服务发现**，然后单击**配置**页签。
   iv. 在**配置**页签可以通过以下两种方式添加服务发现。
      a. 在**默认服务发现**页签，单击vpc-ecs-service-discovery右侧的**详情**。
      b. 在**YAML配置**对话框中修改以下内容，然后单击**确认**。
         - 将默认的端口8888改为实际的端口，例如：8080。
         - 将默认的路径/metrics改为实际的路径，例如：/actuator/prometheus。



此处会采集当前VPC网络下所有ECS实例上的8080/actuator/prometheus端点。
如果您只希望采集部分ECS实例，则可以通过配置 `tag_filters` 来对ECS实例按标签进行过滤。

```
#格式如下
   tag_filters:
        - key: 'testKey'
          values: ['testValue']
```



然后在ECS管理控制台中为ECS实例添加对应的标签。具体操作，请参见编辑实例标签。

a. 在**自定义服务发现**页签，单击**添加**。

b. 在弹出的**添加**对话框，输入采集的指标参数，然后单击**保存**。

方法一：修改默认服务发现。

方法二：自定义服务发现。

5. 创建Grafana大盘。

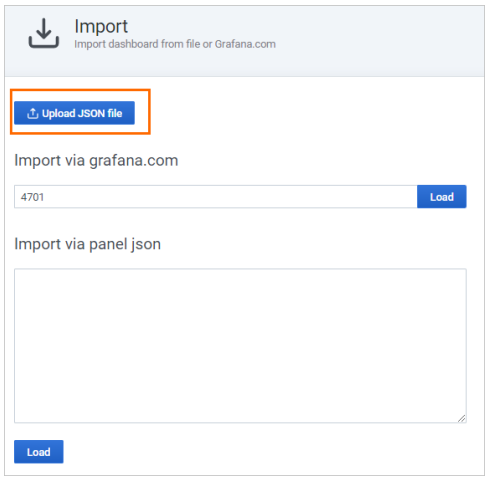
i. 在控制台左侧导航栏单击**大盘列表**。

ii. 在**大盘列表**页面单击右上角的**创建大盘**。

iii. 在左侧导航栏选择**+ > Import**。

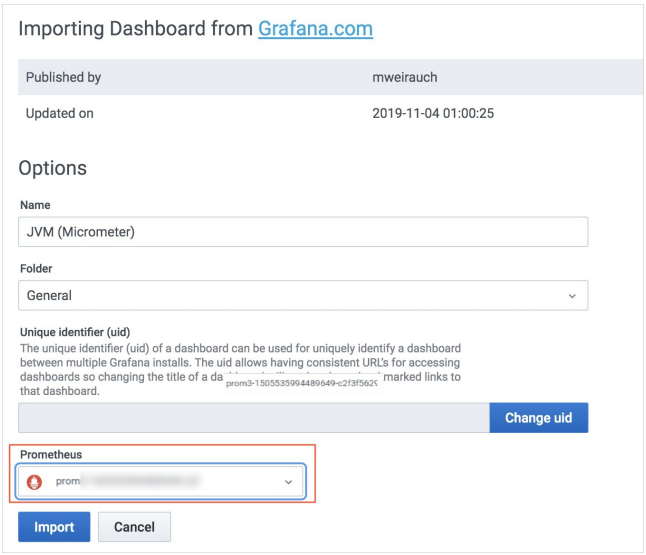
iv. 在**Import**页面的**Import via grafana.com**文本框，输入Prometheus提供的JVM大盘模板的ID*4701*，然后在其右侧单击**Load**。

> **说明** 如需获取其他Grafana大盘模板，请参见Dashboards。

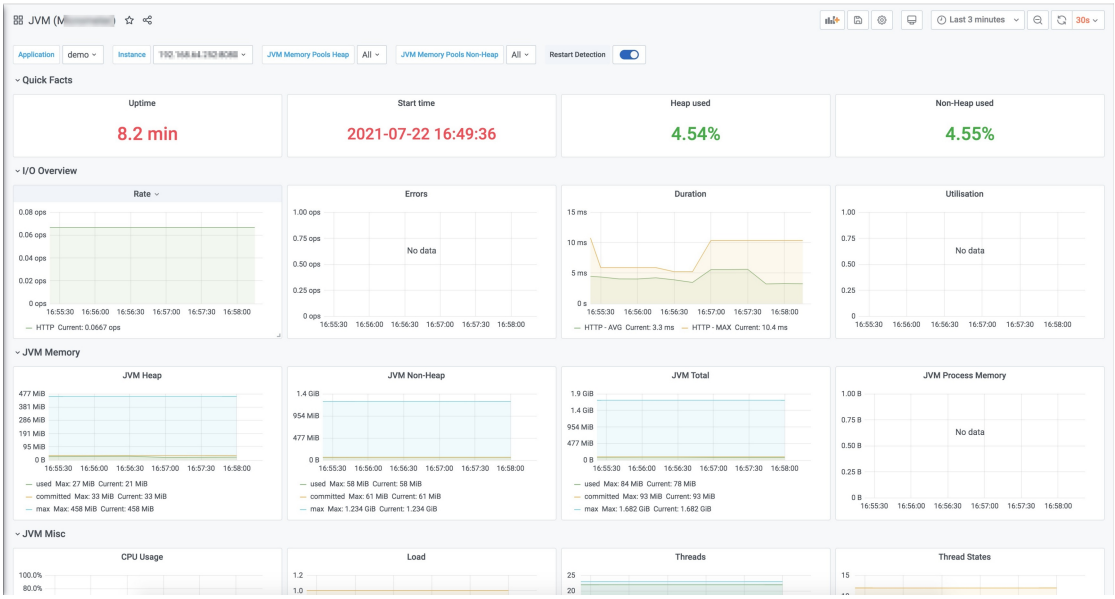


v. 在**Prometheus**下拉列表，选择您的VPC网络下的数据源，然后单击**Import**。

VPC网络下的数据源名称格式为：vpc-****。



导入成功后即可查看Grafana大盘。



相关操作

如果您需要监控Java应用中的某个API请求次数，可以执行以下操作。

1. 创建一个示例Controller，Java文件内容如下。

```
@RestController
@RequestMapping("/v1")
public class IndexController {
    @Autowired
    MeterRegistry registry;
    private Counter counter_index;
    @PostConstruct
    private void init(){
        counter_index = registry.counter("demo_app_requests_method_count", "method", "I
ndexController.index");
    }
    @RequestMapping(value = "/index")
    public Object index(){
        try{
            counter_index.increment();
        } catch (Exception e) {
            return e;
        }
        return counter_index.count();
    }
}
```

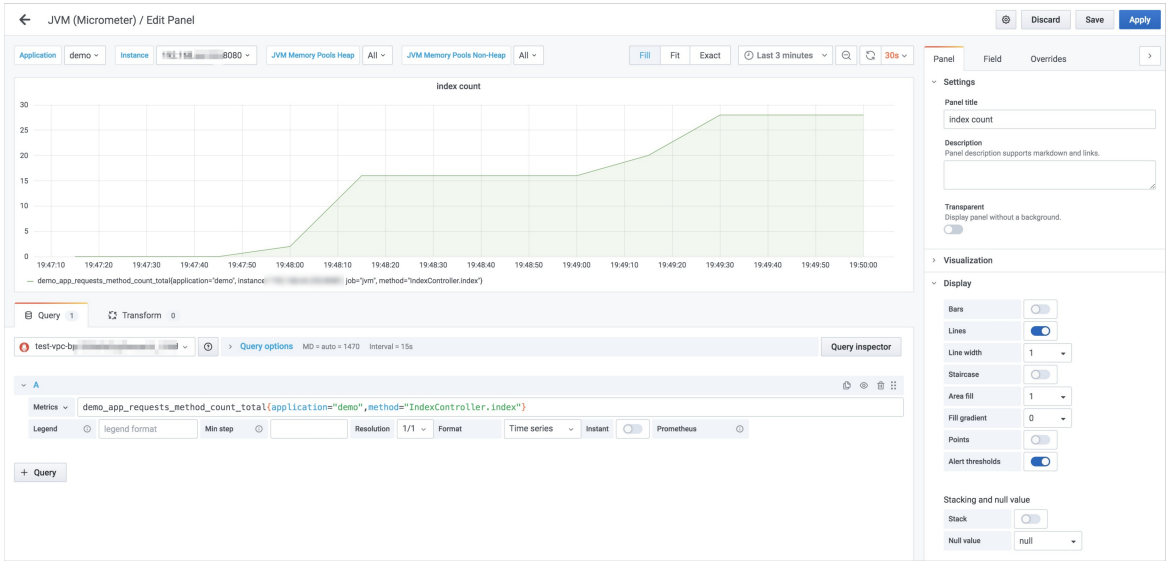
2. 执行以下命令测试Java示例代码是否正常运行。

   如果命令能够访问，则说明Java示例正常工作。

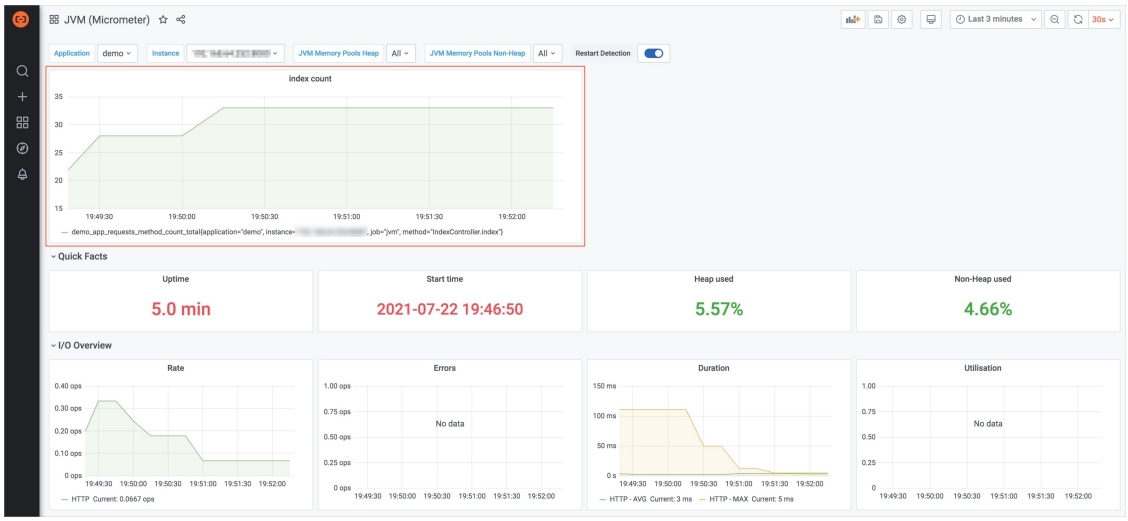
```
curl http://127.0.0.1:8080/v1/index
```

3. 进入上面创建的Grafana大盘页面。
4. 单击页面右上角的  图标，然后单击**+Add new panel**。
5. 在**Edit Panel**页面的**Query**区域的下拉列表中选择VPC网络下的ECS数据源。在**A**折叠面板的**Metrics**右侧文本框输入以下PromQL查询语句。

```
demo_app_requests_method_count_total{application="demo",method="IndexController.index"}
```



6. 在右侧**Panel**页签输入面板的名称，例如：index count。
7. 单击右上角**Save**，在弹出的对话框中单击**Save**。
   配置完毕后，在Grafana大盘新增的index count面板中即可查看API请求次数。



17.2. 监控VPC网络下ECS实例中的Node节点

本文介绍在VPC网络下的ECS实例接入阿里云Prometheus监控后，如何监控ECS实例中的Node节点。

前提条件

- 已创建ECS（Node）。
- 已将VPC网络下ECS实例接入Prometheus监控。具体操作，请参见Prometheus实例 for VPC。

> **说明** 本文中的参数取值均为示例，您可以根据实际情况进行修改。

操作步骤

1. 下载Node Exporter。
   - 通过Prometheus官网下载。
   - 通过命令下载。

     ```
     wget https://github.com/prometheus/node_exporter/releases/download/v1.2.0/node_exporter-1.2.0.linux-amd64.tar.gz
     ```

2. 安装Node Exporter。
   i. 解压下载的Node Exporter。

      ```
      tar -zvxf node_exporter-1.2.0.linux-amd64.tar.gz -C /usr/local/
      ```

   ii. 重命名解压后的文件。

      ```
      mv /usr/local/node_exporter-1.2.0.linux-amd64 /usr/local/node_exporter
      ```

iii. 启动Node Exporter。

```
cd /usr/local/node_exporter
nohup ./node_exporter &
```

iv. 检查9100端口是否被占用。

```
ss -naltp | grep 9100
```

返回信息如下时，表示9100端口未被占用。

```
LISTEN 0      4096               *:9100             *:*    users:(("node_exporter",
pid=1420**,fd=3))
```

如果9100端口被占用，则执行以下命令调整启动参数中的端口。例如，端口可以调整为9999。

```
nohup ./node_exporter  --web.listen-address=":9999"&
```

3. 检查服务。

启动Node后通过命令访问以下地址。

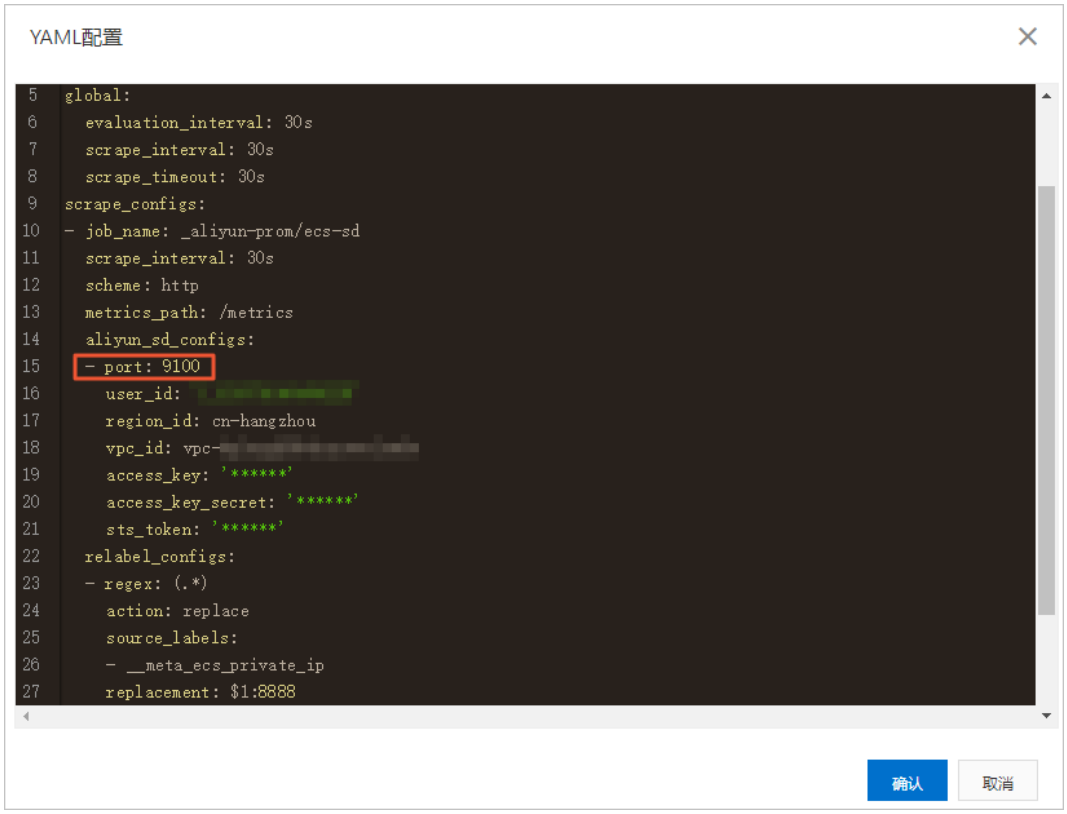
```
curl http://localhost:9100/metrics
```

> **说明**
>
> - 请根据实际情况替换命令中的 `localhost` 和端口。
> - 建议在其他ECS实例上也进行地址访问测试（http://*[ECS IP]*:9100/metrics），检查连接是否通畅，避免被安全组限制。

预计可得到以下返回结果。

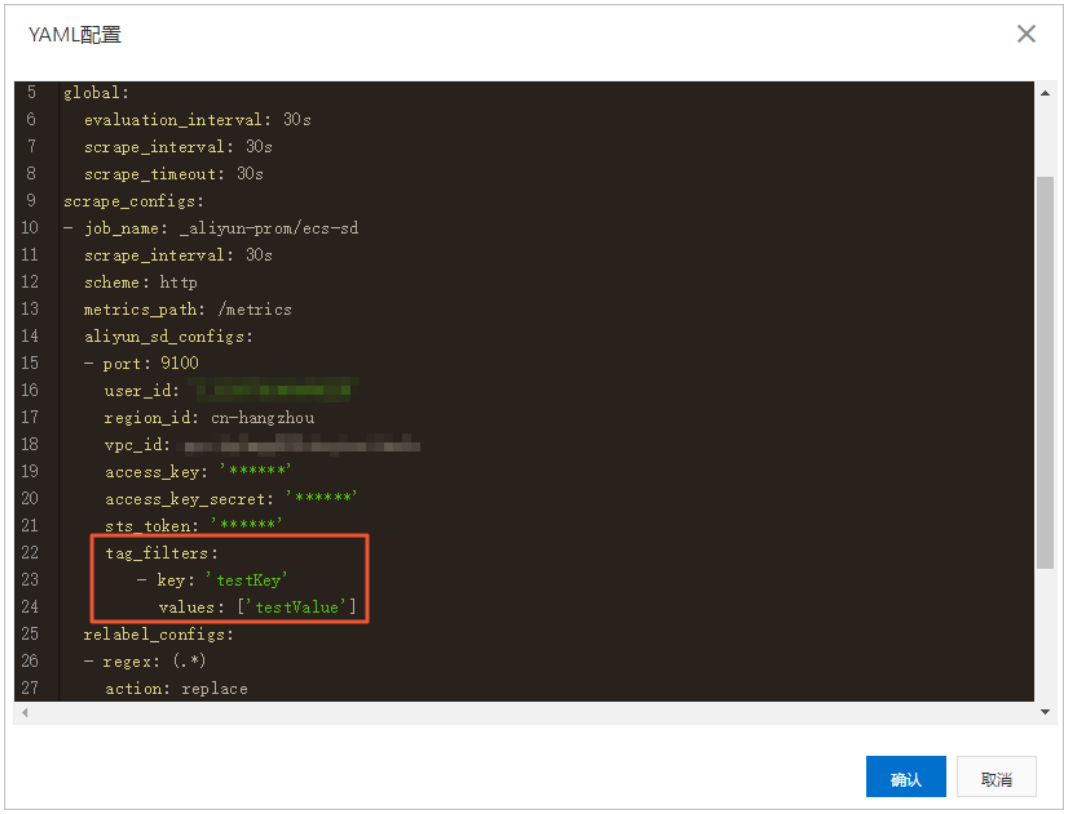
```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cyc
les.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.16.6"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.386192e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.386192e+06
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket has
h table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 4562
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 754
```

4. 添加服务发现。
   i. 登录Prometheus控制台。
   ii. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域，单击目标VPC类型的Prometheus实例的名称。
   iii. 在左侧导航栏单击**服务发现**，然后单击**配置**页签。
   iv. 在**配置**页签可以通过以下两种方式添加服务发现。
      a. 在**默认服务发现**页签，单击vpc-ecs-service-discovery右侧的**详情**。
      b. 在**YAML配置**对话框中修改以下内容，然后单击**确认**。
      将默认的端口8888改为实际的端口，例如：9100。



此处会采集当前VPC网络下所有ECS实例上的9100/metrics端点。
如果您只希望采集部分ECS实例，则可以通过配置 `tag_filters` 来对ECS实例按标签进行过滤。

```
#格式如下
   tag_filters:
       - key: 'testKey'
         values: ['testValue']
```



然后在ECS管理控制台中为ECS实例添加对应的标签。具体操作，请参见编辑实例标签。

a. 在**自定义服务发现**页签，单击**添加**。

b. 在弹出的**添加**对话框，输入采集的指标参数，然后单击**保存**。

方法一：修改默认服务发现。

方法二：自定义服务发现。

5. 创建Grafana大盘。

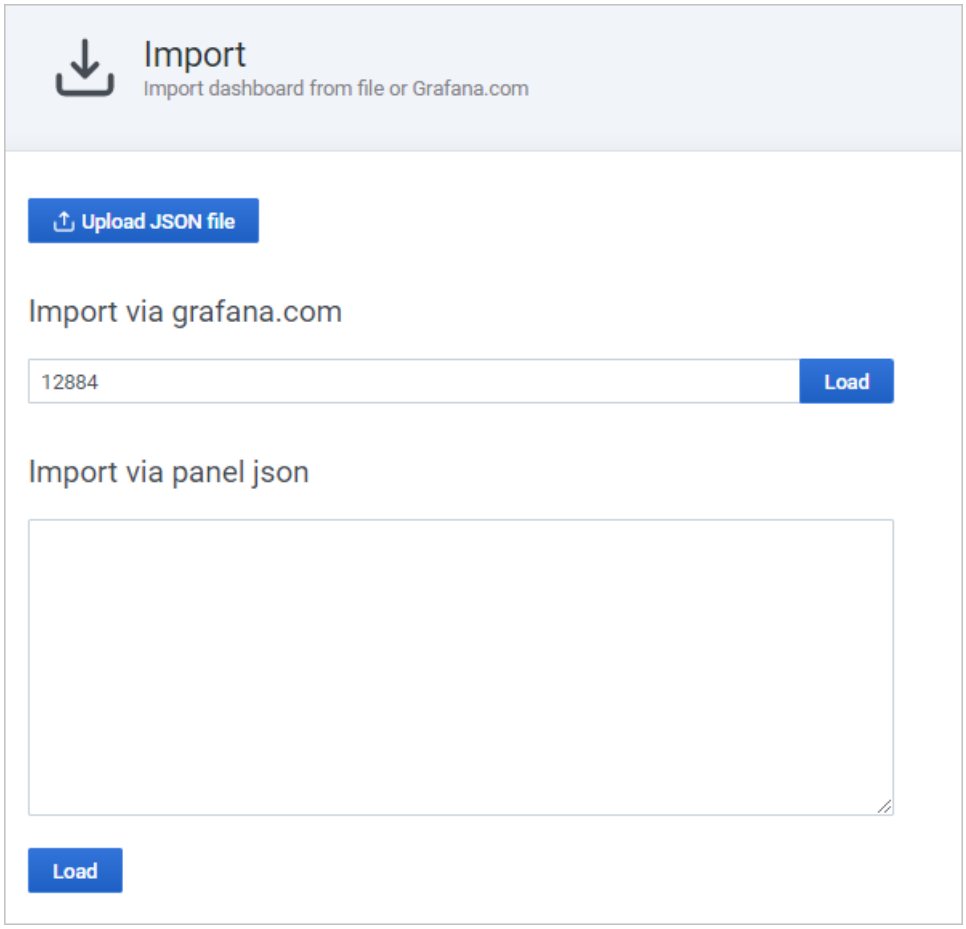
i. 在控制台左侧导航栏单击**大盘列表**。

ii. 在**大盘列表**页面单击右上角的**创建大盘**。

iii. 在左侧导航栏选择**+ > Import**。

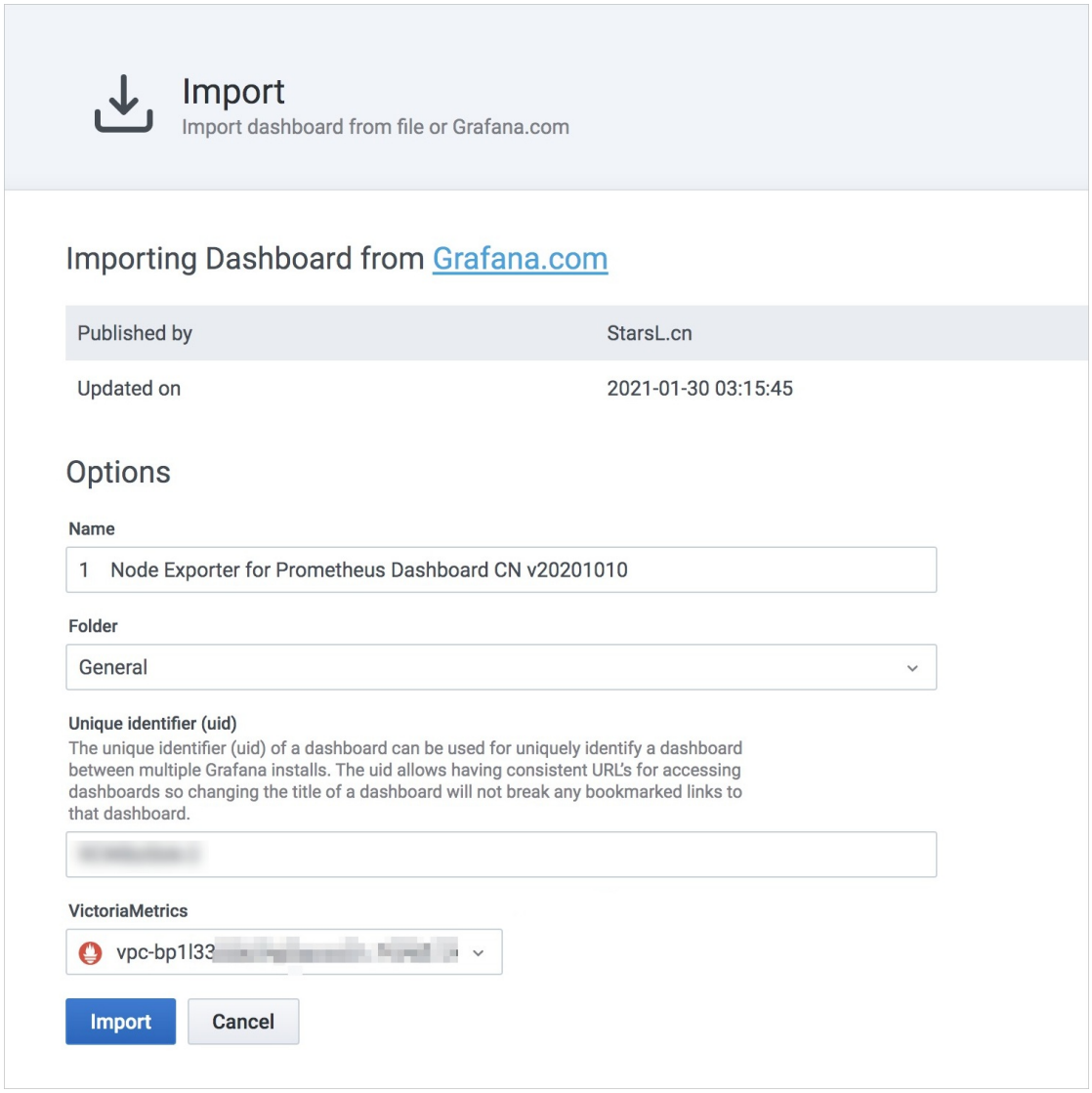
iv. 在Import页面的Import via grafana.com文本框，输入Prometheus提供的Node大盘模板的ID*1884*，然后在其右侧单击Load。

> **说明** 如需获取其他Grafana大盘模板，请参见Dashboards。

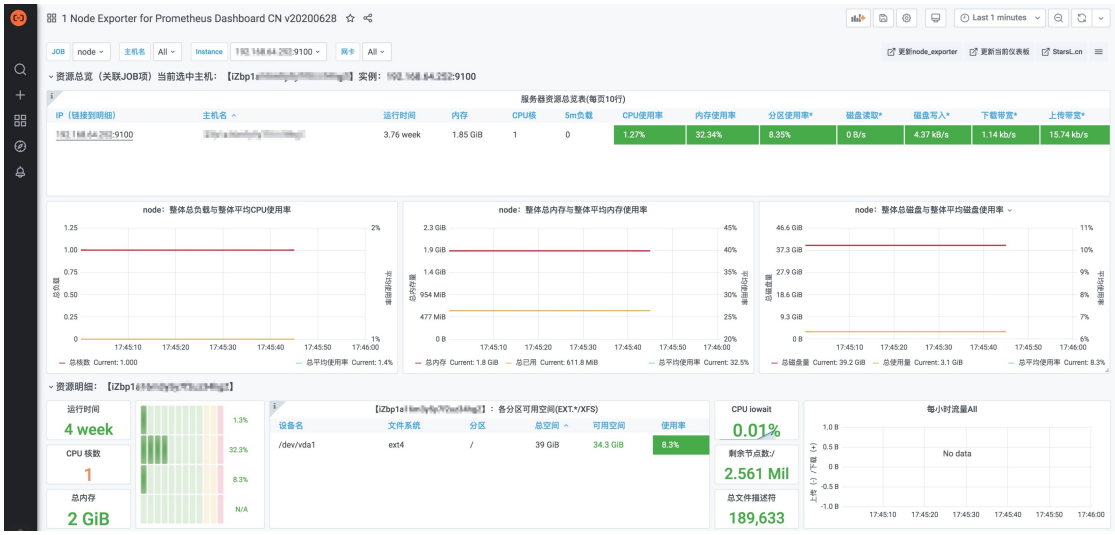


v. 在Prometheus下拉列表，选择您的VPC网络下的数据源，然后单击Import。

VPC网络下的数据源名称格式为：vpc-****。



导入成功后即可查看Grafana大盘。



17.3. 将阿里云Prometheus监控数据接入本地Grafana

如果您需要在本地的Grafana系统中查看阿里云Prometheus监控数据，可以利用Prometheus监控提供的专用API接口轻松实现此目的。本文介绍了将Prometheus监控数据接入本地Grafana的实现方法。

前提条件

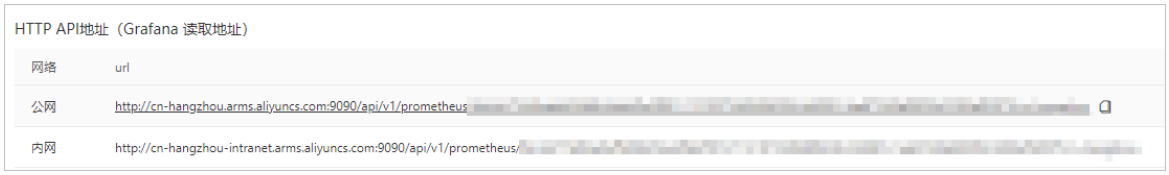
- 已创建Prometheus实例，具体操作，请参见：
  - Prometheus实例 for 容器服务
  - Prometheus实例 for Kubernetes
  - Prometheus实例 for Remote Write
  - Prometheus实例 for VPC
  - Prometheus实例 for 云服务
- 您已在本地成功安装Grafana软件，请参见Grafana。

步骤一：获取Prometheus监控提供的专用API接口

该专用API接口是连接Prometheus和本地Grafana的纽带。请按照以下步骤获取该接口：

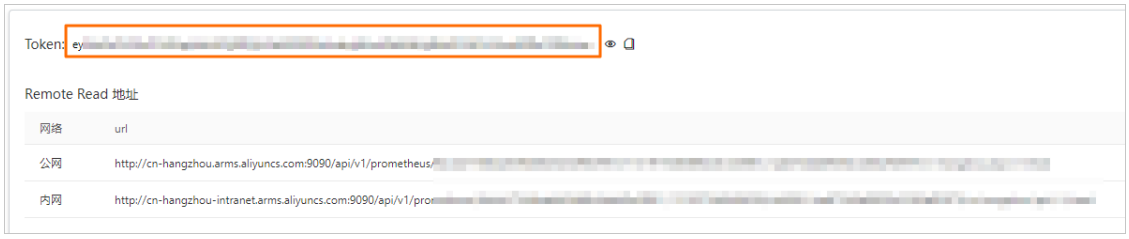
1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择Prometheus实例所在的地域，并在目标集群右侧的**操作**列单击**设置**。
3. 在右侧页面单击**设置**页签。
4. 在**设置**页签上，根据需求复制公网或私网的HTTP API地址。

> **说明** 如果是云服务类型的Prometheus实例，请根据接入云服务的产品类型选择对应的HTTP API地址。



5. （可选）如果您需要提高Grafana数据读取的安全性，可以单击**生成token**，获取Prometheus实例的鉴权Token。

> **注意** 生成Token后，在Grafana中添加数据源时必须配置Token，否则可能无法读取Prometheus监控数据。



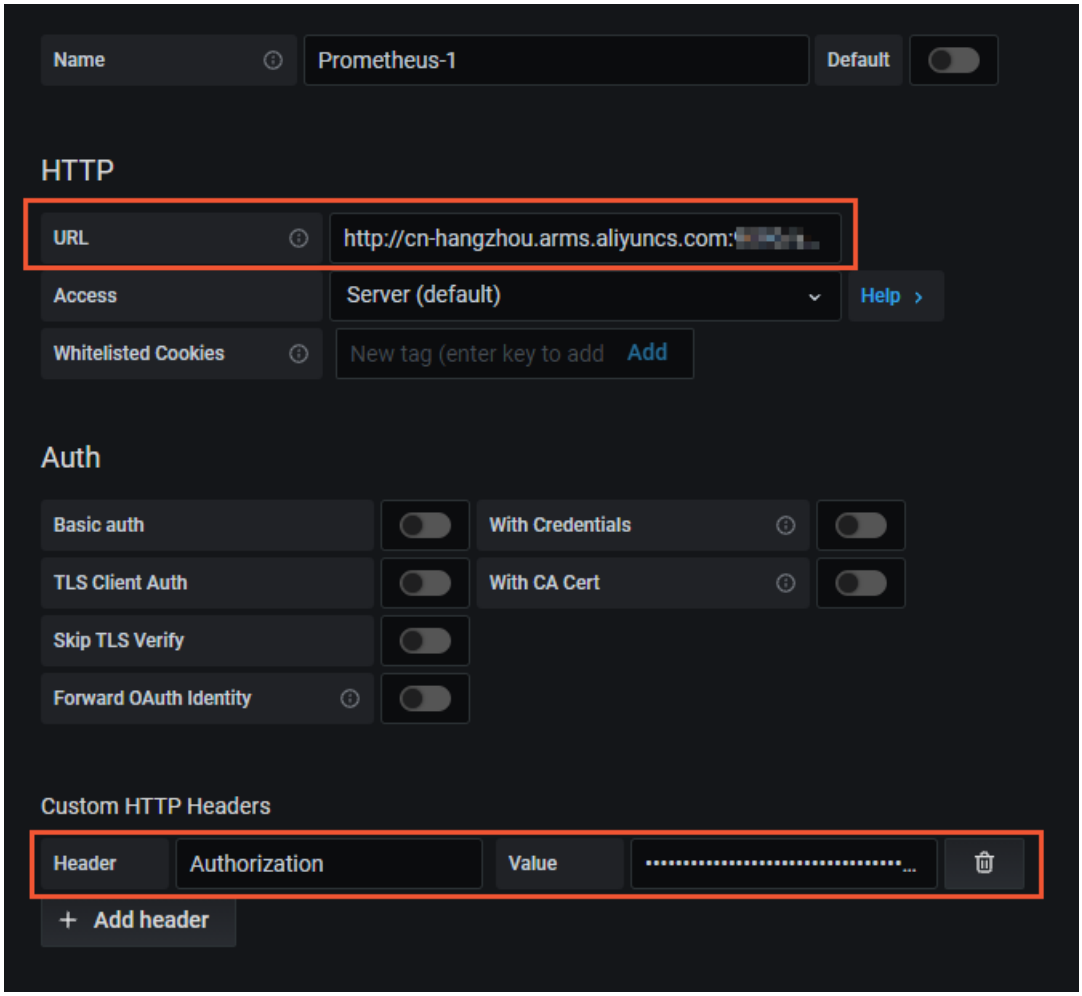
步骤二：在本地Grafana添加数据源

将步骤一获得的API接口地址添加为本地Grafana的数据源即可实现目标。请按照以下步骤添加数据源：

1. 以管理员账号登录本地Grafana系统。
2. 在左侧导航栏中选择Configuration > Data Sources。

   > 说明　仅管理员可以看到此菜单。

3. 在Data Sources页签上单击Add data source。
4. 在Add data source页面上单击Prometheus。
5. 在Settings页签的Name字段输入自定义的名称，在URL字段粘贴步骤一：获取Prometheus监控提供的专用API接口获得的API接口地址。
6. （可选）在Custom HTTP Headers区域单击+Add header，设置Header为Authorization，设置Value为步骤一：获取Prometheus监控提供的专用API接口获取的鉴权Token。



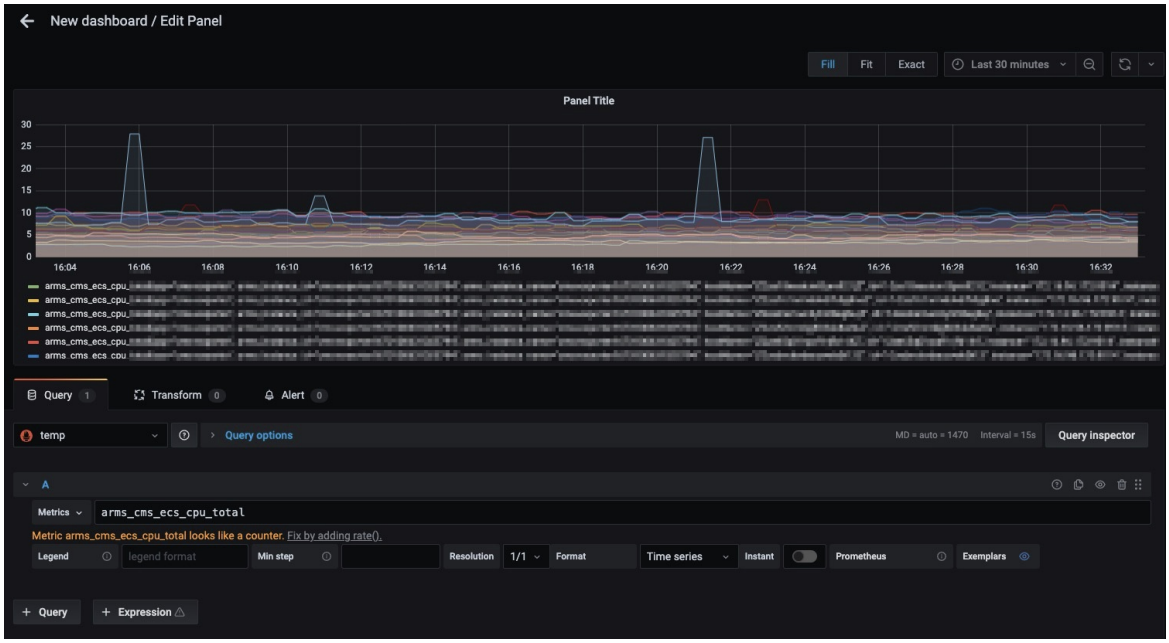
7. 单击页签底部的Save & Test。

验证结果

请按照以下步骤验证操作是否成功：

1. 登录本地Grafana系统。
2. 在左侧导航栏中选择+ > Create。
3. 在New dashboard页面单击Add an empty panel。

4. 在Edit Panel页面的Query页签的下拉框中选择步骤二中添加的数据源，在A区域的Metrics字段输入指标名称并按回车。
   如果能显示出相应指标的图表，则说明操作成功。否则请检查填写的接口地址或Token是否正确，以及数据源是否有Prometheus监控数据。



17.4. 开始使用日志监控

对于高度定制化的业务场景，可以通过创建日志监控任务来自由统计所需指标，生成需要的数据与报表，灵活地配置报警。

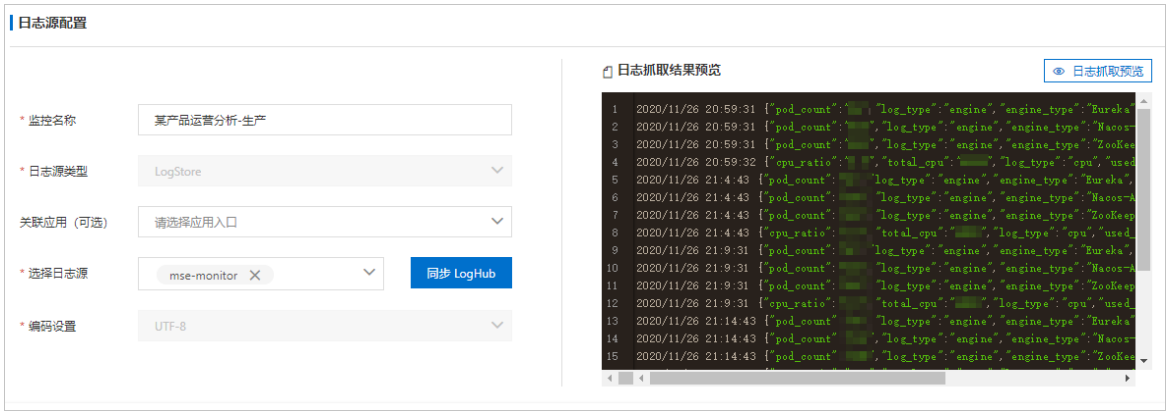
背景信息

ARMS日志监控支持完全自定义的监控任务，其流程包含配置数据源和配置指标两个关键步骤，如下图所示。

日志监控任务的创建流程

创建日志监控

1. 登录ARMS控制台。
2. 在左侧导航栏选择**业务监控 > 日志接入**。
3. 在**日志接入**页面右上角单击**创建日志监控**。
4. 在**新建日志监控**页面的**日志源配置**区域设置如下参数。

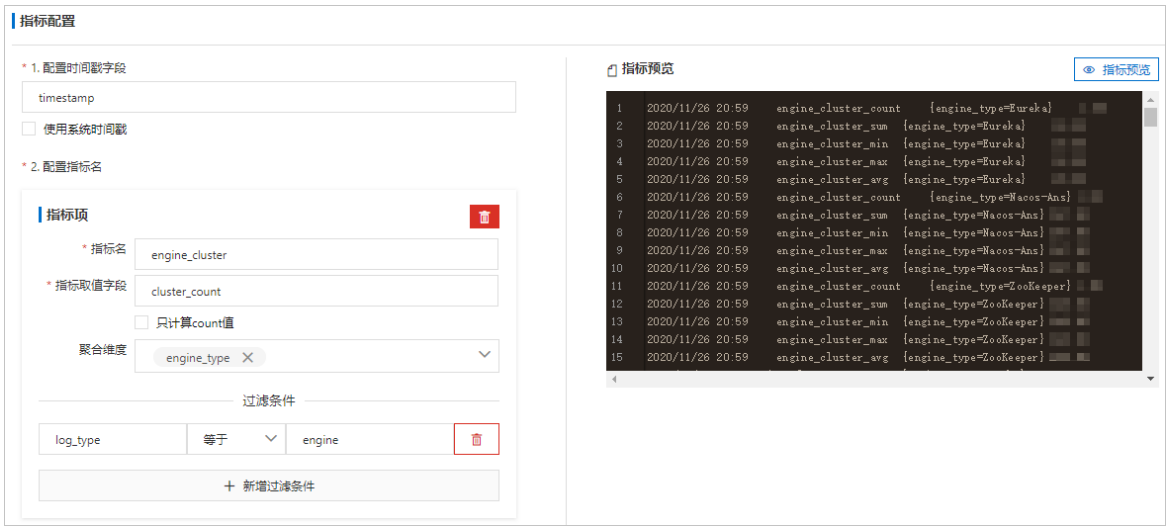


参数	说明
监控名称	设置监控名称。
关联应用（可选）	选择需要关联的应用监控下的应用。设置关联应用后可以在**日志接入**页面按应用筛选过滤。
选择日志源	如果下拉框中没有可选的日志源，单击**同步LogHub**。 选择日志源后日志会自动抓取，您也可以在**日志抓取结果预览**区域单击右上角的**日志抓取预览**。

说明

- 同步日志源前，请确认您已开通阿里云日志服务，且当前账号为阿里云账号或已被授权访问日志服务的RAM用户。同步LogHub的操作，请参见同步LogHub数据源。
- ARMS会从选择的机器日志中抓取部分数据（最多20条）。由于需要建立预抓取的临时通道，一般需要30秒左右。
- 如果预抓取日志不成功，请检查选择的日志源是否正确。

5. 在**指标配置**区域配置时间戳字段和指标名，配置完成后单击**保存监控配置**。
   在**配置指标名**区域单击**新增指标**，设置指标名、指标取值字段和聚合维度；单击**新增过滤条件**可以增加该指标的过滤条件。



指标参数	说明
配置时间戳字段	选择日志中的时间戳字段。如果日志中没有时间戳字段，可以选中**使用系统时间戳**，即仅监控当前时间点的数据。
配置指标名	单击**新增指标**可以增加多个监控指标。
指标名	设置指标名称。
指标取值字段	是衡量目标的度量，选择日志中的数值类型字段名。ARMS的指标对应于实时计算后的Count、Max、Min、Sum、Avg等值。如果只需要统计日志行数，选中**只计算count值**。
聚合维度	是衡量目标的思维角度，选择日志里的非数值类型字段。例如想统计每个品牌的销售额，那么品牌名称就是聚合维度字段。最多支持选择8个聚合维度。
过滤条件	单击**新增过滤条件**可以增加多个过滤条件。 选择日志中的某一个字段作为过滤指标。例如：log_type等于engine。

6. （可选）单击**报警配置**行，在**报警配置**区域单击**创建报警**，设置报警规则。然后单击右上角的**保存**。
   报警规则可以直接输入规则表达式，也可以按提示设置表达式。表达式需要使用PromQL语句。

   > **说明** 如果触发已开启的报警，报警通知会发送至联系人组PrometheusGroup。

管理日志监控

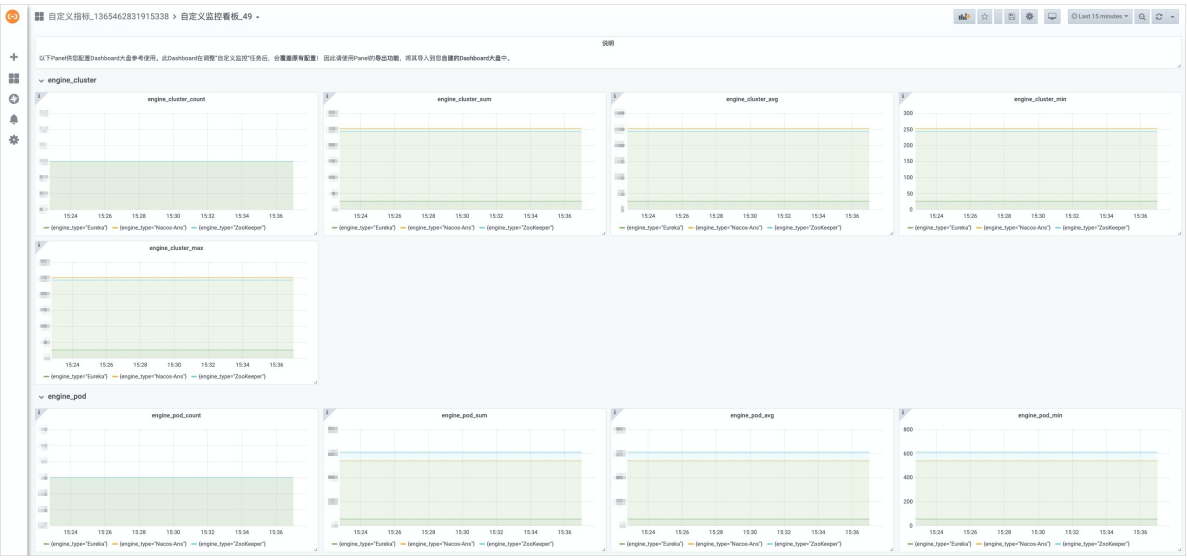
在**日志接入**页面可以查看创建的所有日志监控。

- 在页面上方搜索框内输入监控名或指标名称，可以按监控名或指标名称进行筛选。
- 单击监控名称所在行可以展开或合并该监控的指标信息。
- 单击各监控区域右上角的**编辑**，在**日志监控设置**页面的**规则配置**页签可以修改监控信息，在**删除**页签可以删除该监控。

  > **注意** 此操作将会清除该日志监控的所有数据，且删除之后无法恢复。

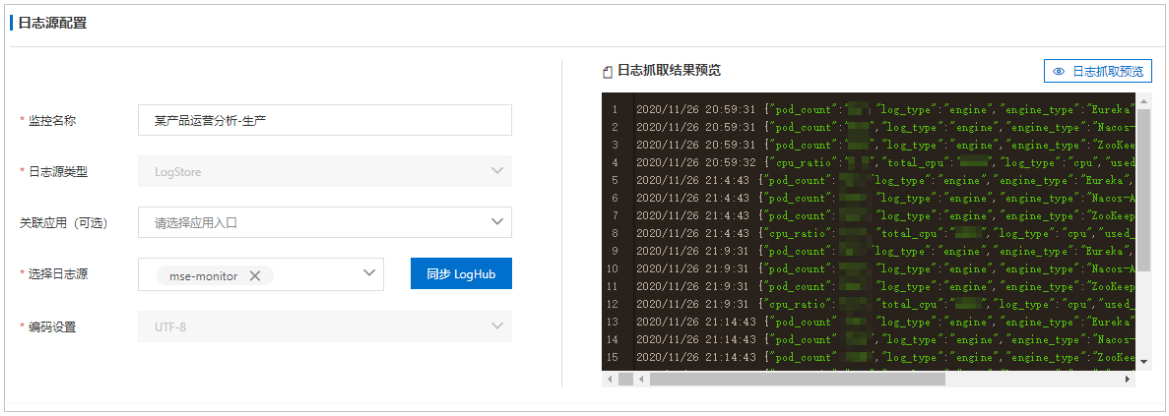
- 单击各监控区域右上角的**更多**可以启动或停止该监控。
- 单击各监控区域右上角的**看板**可以以大盘样式展示监控数据。

  > **说明** 修改日志监控会重置大盘配置，如果您修改了大盘配置请及时保存。



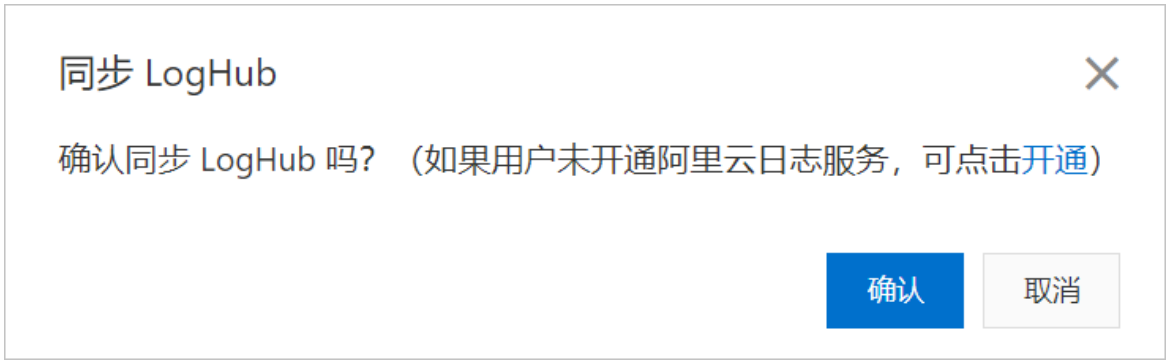
同步LogHub数据源

1. 登录ARMS控制台。
2. 在左侧导航栏选择**业务监控 > 日志接入**。
3. 在**日志接入**页面右上角单击**创建日志监控**。
4. 在**新建日志监控**页面的**日志源配置**区域单击**同步LogHub**。



5. （可选）如果此前未授权ARMS读取LogHub数据，则在弹出的**提示**对话框中，单击**确认**。

6. 在**同步LogHub**对话框中单击**确认**。



17.5. 将Prometheus监控接入PagerDuty

PagerDuty是一款为企业IT部门提供事件响应的软件。您可以将Prometheus监控接入PagerDuty从而触发自动事件或追踪服务变化。

前提条件

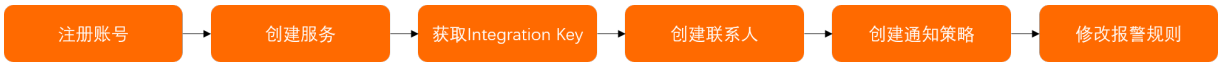
- 您的K8s集群已接入Prometheus监控。具体操作，请参见Prometheus实例 for 容器服务。
- 您已创建报警规则，且报警规则处于启用状态并被触发。具体操作，请参见创建报警。

背景信息

PagerDuty是一款为企业IT部门提供事件响应的软件。当服务出现问题时，PagerDuty支持以电话、短信、邮件等方式通知企业IT部门。关于PagerDuty的更多信息，请参见PagerDuty官网。

操作流程

将Prometheus监控接入PagerDuty的操作流程如下图所示。

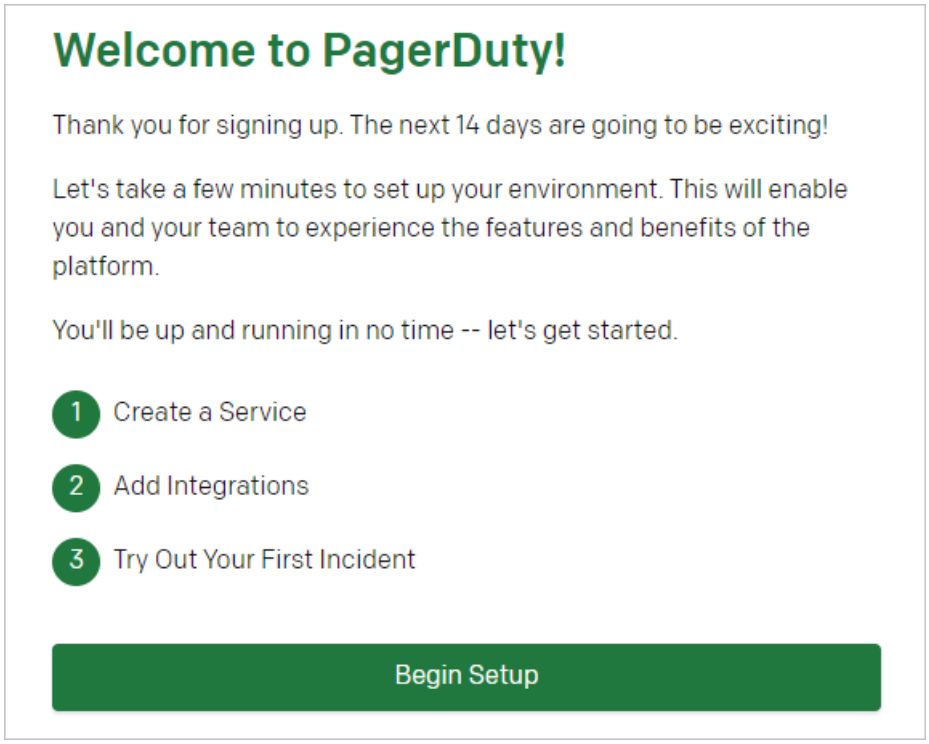


步骤一：注册账号

注册14天免费试用的PagerDuty账号的操作步骤如下：

1. 打开PagerDuty注册页面。
2. 在**Try PagerDuty**配置向导区域，执行以下操作：
   i. 输入邮箱，然后单击**GET STARTED**。
   ii. 输入姓名，然后单击**NEXT STEP**。
   iii. 输入密码，然后单击**NEXT STEP**。
   iv. 输入子域名，选中服务协议，然后单击**CREATE ACCOUNT**。

   注册完成后跳转到PagerDuty欢迎页面。



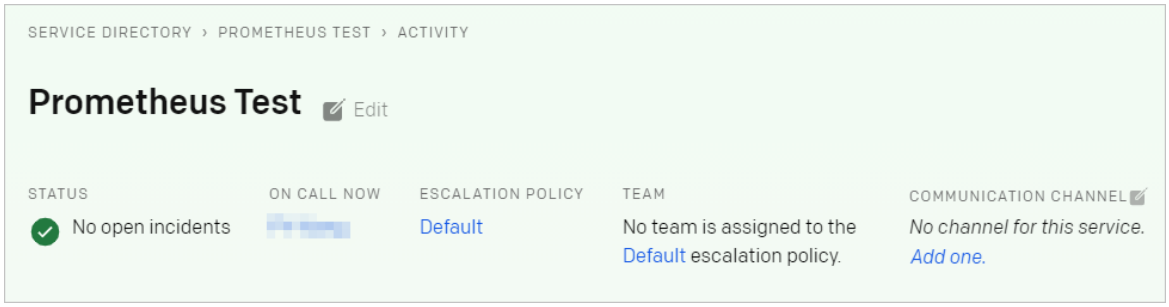
步骤二：创建服务

在PagerDuty控制台为Promethues监控创建对应的服务的操作步骤如下：

1. 登录PagerDuty控制台。
2. 在顶部菜单栏，选择**Services > Service Directory**。
3. 在**Service Directory**页面，单击**New Service**。
4. 在**Add a Service**页面，输入服务名称，选择**Integration Type**为**Prometheus**，然后单击**Add Service**。

> **说明** 您可以根据业务需求设置Service的其他参数。

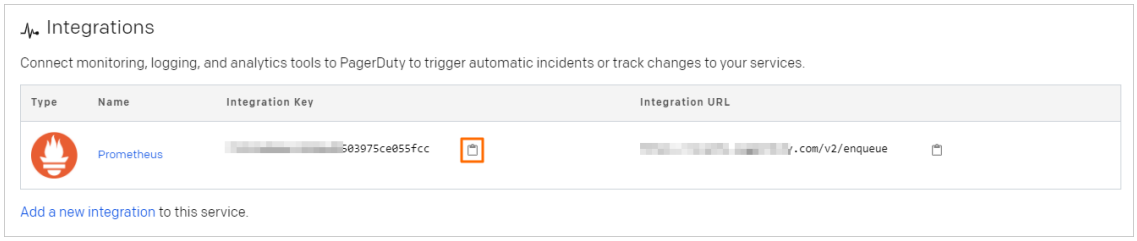
跳转的页面显示创建的服务的信息。



步骤三：获取Integration Key

获取用于将Prometheus监控接入PagerDuty的Integration Key的操作步骤如下：

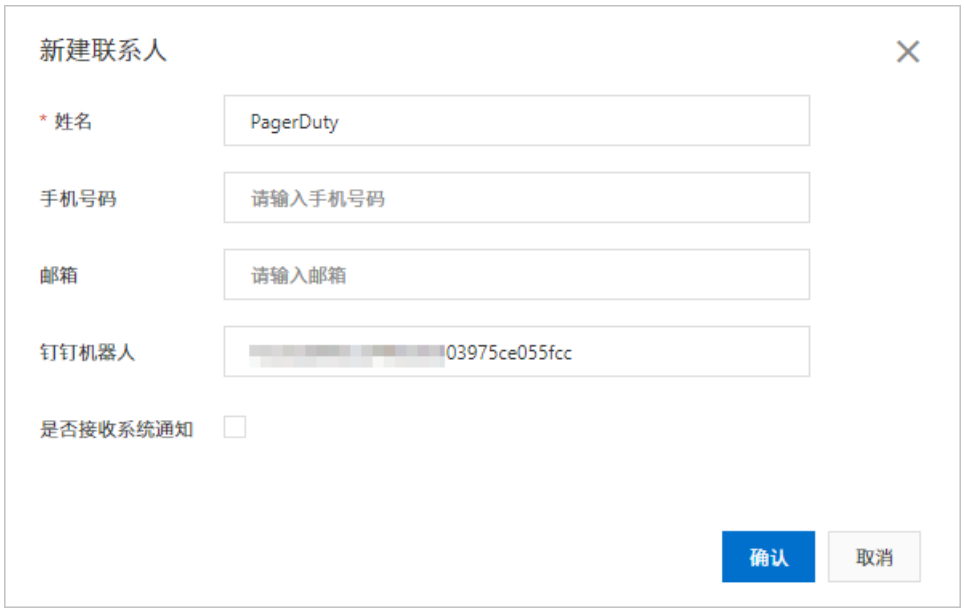
1. 在创建的服务的页面下方，单击**Integrations**页签。
2. 在**Integrations**区域，找到Prometheus服务，在其右侧**Integration Key**列，单击复制图标将Integration Key复制到剪贴板。



步骤四：创建联系人

使用Integration Key创建联系人的操作步骤如下：

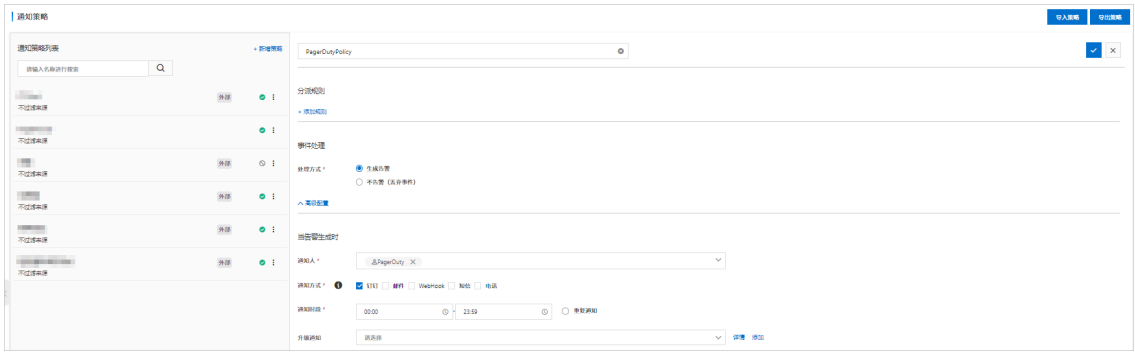
1. 登录Prometheus控制台。
2. 在左侧导航栏，选择**报警管理 > 联系人管理**。
3. 在**联系人管理**页面，单击**新建联系人**。
4. 在**新建联系人**对话框，输入姓名，将**钉钉机器人**设置为获取的Integration Key，然后单击**确认**。



步骤五：创建通知策略

为联系人创建通知策略的操作步骤如下：

1. 登录Prometheus控制台。
2. 在左侧导航栏，单击**通知策略**。
3. 在**通知策略**页面的**通知策略列表**区域，单击**新增策略**。
4. 在**通知策略**页面右侧，输入通知策略的名称，在**事件处理**区域，选择**处理方式**为**生成告警**，在**当告警生成时**区域，选择**通知人**为创建的联系人，选择**通知方式**为**钉钉**，设置**通知时段**，然后在右上角，单击确认图标。



步骤六：修改报警规则

将报警规则的通知策略修改为创建的通知策略的操作步骤如下：

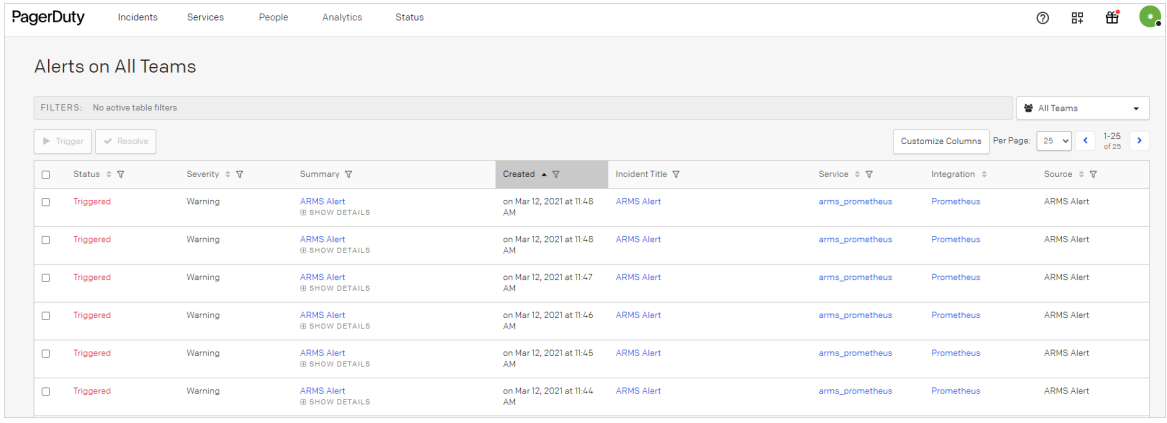
1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击目标K8s集群的名称。
3. 在左侧导航栏，单击**报警配置**。
4. 在报警配置页面，找到目标报警规则，在其右侧**操作**列，单击**编辑**。
5. 在**编辑报警**面板的**通知策略**下拉列表，选择创建的通知策略，然后单击**确定**。

结果验证

您可以在PagerDuty控制台查看报警以验证是否成功接入。

（可选）

1. 登录PagerDuty控制台。
2. 在顶部菜单栏，选择**Incidents > Alerts**。
   Alerts on All Teams页面显示触发的报警。



3. （可选）如需查看Alert的详细信息，在**Summary**列，单击目标Alert的名称。

17.6. 通过阿里云Prometheus自定义Grafana大盘

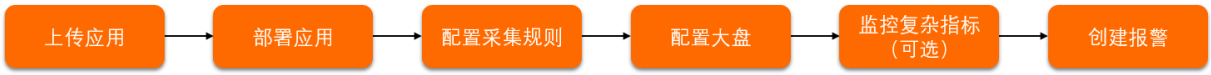
阿里云Prometheus监控可以通过Grafana大盘来展示监控数据，并且支持自定义创建Grafana大盘和从Grafana官网导入大盘两种方式。本文以阿里云容器服务K8s集群和阿里云容器镜像服务为例，介绍通过自定义Grafana大盘展示监控数据。

前提条件

- 阿里云容器服务K8s集群已接入阿里云Prometheus监控。如何接入，请参见Prometheus实例 for 容器服务。
- 已创建阿里云容器镜像服务镜像仓库。如何创建，请参见步骤二：创建镜像仓库。

操作流程

通过阿里云Prometheus自定义Grafana大盘的操作流程如下图所示。



步骤一：上传应用

将应用制作成镜像并上传至阿里云容器镜像服务的镜像仓库的操作步骤如下：

1. 执行以下命令重新编译模块。

   ```
   mvn clean install -DskipTests
   ```

2. 执行以下命令构建镜像。

   ```
   docker build -t <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> . --no-cache
   ```

   示例命令：

   ```
   docker build -t promethues-demo:v0 . --no-cache
   ```

3. 执行以下命令为镜像打标。

   ```
   sudo docker tag <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
   ```

   示例命令：

   ```
   sudo docker tag promethues-demo:v0 registry.cn-hangzhou.aliyuncs.com/testnamespace/promethues-demo:v0
   ```

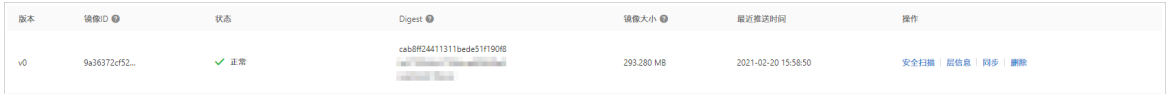
4. 执行以下命令将镜像推送至镜像仓库。

   ```
   sudo docker push <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
   ```

   示例命令：

   ```
   sudo docker push registry.cn-hangzhou.aliyuncs.com/testnamespace/promethues-demo:v0
   ```

   容器镜像服务控制台的**镜像版本**页面显示上传的应用镜像。



步骤二：部署应用

将应用部署至容器服务K8s集群的操作步骤如下：

1. 登录容器服务管理控制台。
2. 在左侧导航栏，选择**集群**。
3. 在**集群列表**页面，找到目标集群，在其右侧**操作**列单击**应用管理**。
4. 创建容器组。

i. 在左侧导航栏，选择**工作负载 > 无状态**。

ii. 在**无状态**页面，单击**使用模板创建**。

iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: prometheus-demo
spec:
  replicas: 2
  template:
    metadata:
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/prometheus-metrics'
        prometheus.io/port: '8081'
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        imagePullPolicy: Always
        image: <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
        ports:
        - containerPort: 8080
          name: tomcat-normal
        - containerPort: 8081
          name: tomcat-monitor
```

示例代码：

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: prometheus-demo
  labels:
    app: tomcat
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/path: '/prometheus-metrics'
        prometheus.io/port: '8081'
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        imagePullPolicy: Always
        image: registry.cn-hangzhou.aliyuncs.com/peiyu-test/prometheus-demo:v0
        ports:
        - containerPort: 8080
          name: tomcat-normal
        - containerPort: 8081
          name: tomcat-monitor
```

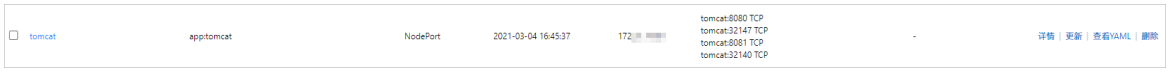
无状态页面显示创建的容器组。

5. 创建服务。
   i. 在左侧导航栏，选择**服务与路由 > 服务**。
   ii. 在**服务**页面，单击**使用YAML创建资源**。

iii. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: tomcat
  name: tomcat
  namespace: default
spec:
  ports:
  - name: tomcat-normal
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: tomcat-monitor
    port: 8081
    protocol: TCP
    targetPort: 8081
  type: NodePort
  selector:
    app: tomcat
```

服务页面显示创建的服务。



步骤三：配置采集规则

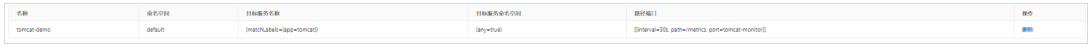
阿里云Prometheus会默认监控CPU信息、内存信息、网络信息等。如果您需要监控默认数据外的其他数据，例如订单信息，那么需要为应用自定义Prometheus监控采集规则。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标实例名称。
3. 为应用配置Prometheus监控采集规则分为以下两种情况。
   - 需要监控部署在K8s集群内的应用的业务数据，例如订单信息。操作步骤如下：
     a. 在左侧导航树单击**服务发现**，然后单击**配置**页签。
     b. 在**配置**页面单击ServiceMonitor页签，然后单击**添加**ServiceMonitor。

c. 在**添加ServiceMonitor**对话框中输入以下内容，然后单击**确定**。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  #  填写一个唯一名称
  name: tomcat-demo
  #  填写目标命名空间
  namespace: default
spec:
  endpoints:
  - interval: 30s
    #  填写service.yaml中Prometheus Exporter对应的Port的Name字段的值
    port: tomcat-monitor
    #  填写Prometheus Exporter对应的Path的值
    path: /metrics
  namespaceSelector:
    any: true
    #  Demo的命名空间
  selector:
    matchLabels:
      #  填写service.yaml的Label字段的值以定位目标service.yaml
      app: tomcat
```

ServiceMonitor页签下显示配置的服务发现。



- 需要监控部署在K8s集群之外的业务数据，如Redis连接数。操作步骤如下：

  a. 在设置页面，单击**Prometheus设置**页签。

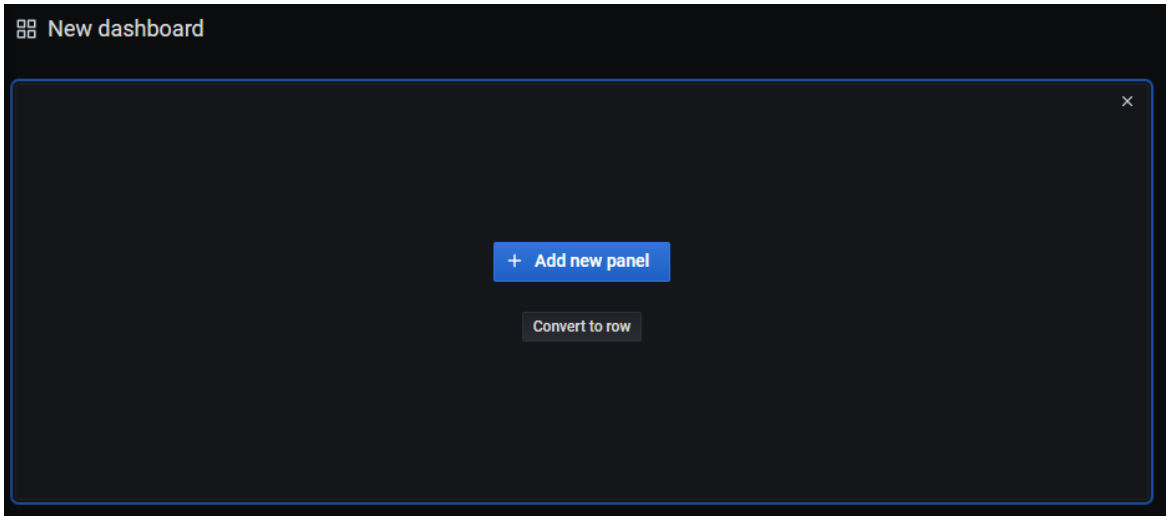
  b. 在*Prometheus.yaml*中输入以下内容，然后单击**保存**。

```
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default
is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is ever
y 1 minute.
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
```

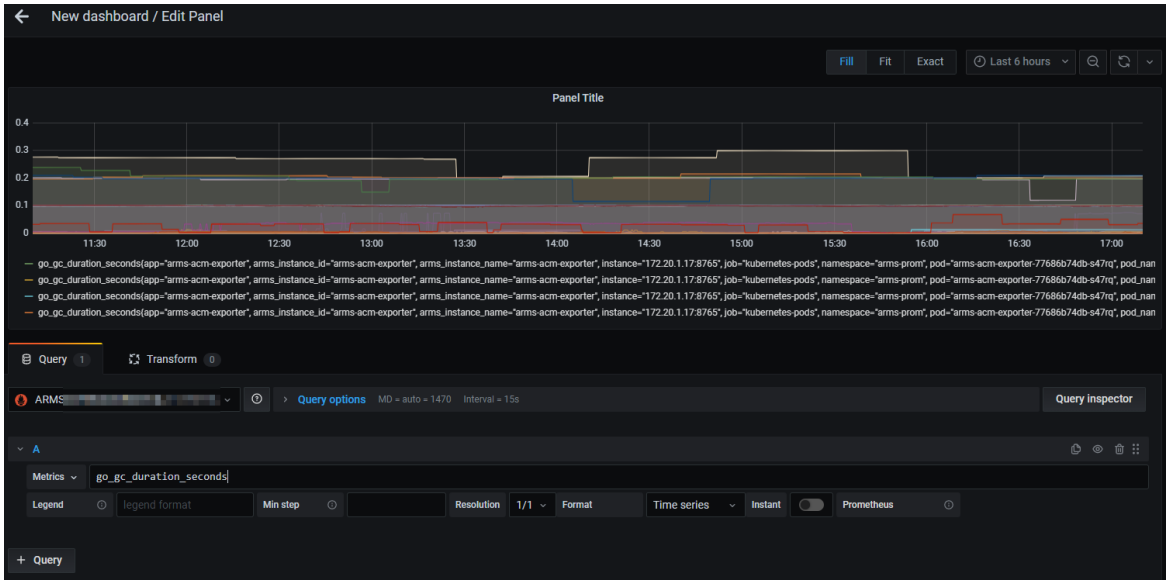
步骤四：配置大盘

配置Grafana大盘以展示数据的操作步骤如下：

1. 打开Grafana大盘概览页。
2. 在左侧导航栏选择**+ > Dashboard**。
3. 在**New dashboard**页面中单击**Add new panel**。

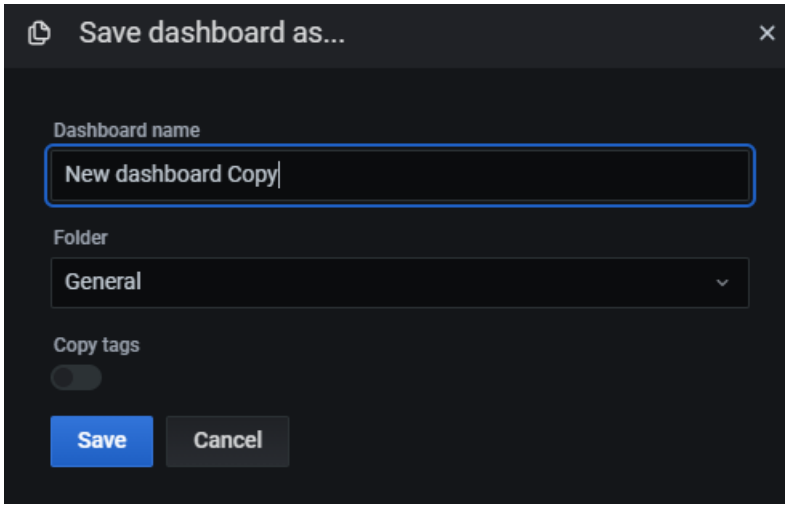


4. 在Edit Panel页面的Query区域的下拉列表中选择集群。在A折叠面板的Metrics下拉列表中选择监控指标，例如：go_gc_duration_seconds。

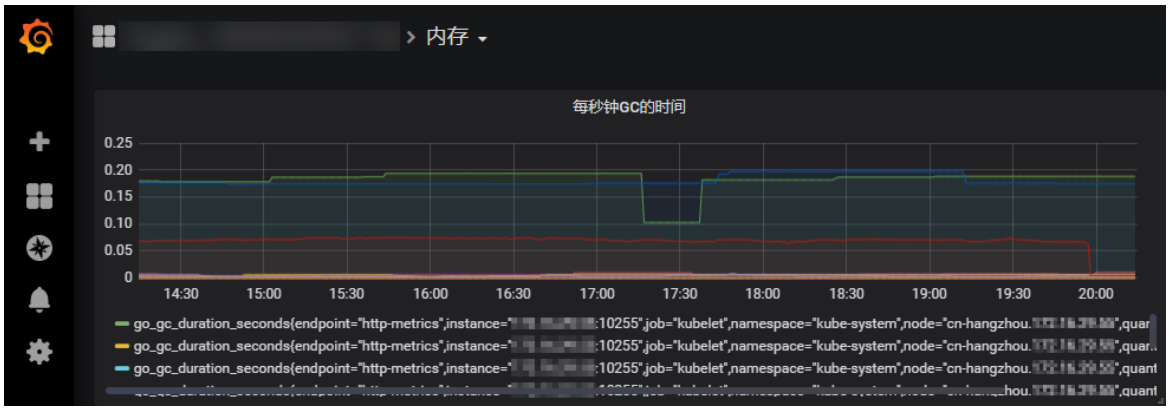


5. 在页面右侧的区域，填写图表名称，选择大盘的可视化类型，如图形、表格、热点图等，并根据您的需求配置其他参数。

6. 单击右上角Save，在弹出的对话框中输入大盘名称，并选择集群，然后单击Save。
   您可根据需要自行创建多个大盘和图表。



配置完毕后的Grafana大盘如图所示。

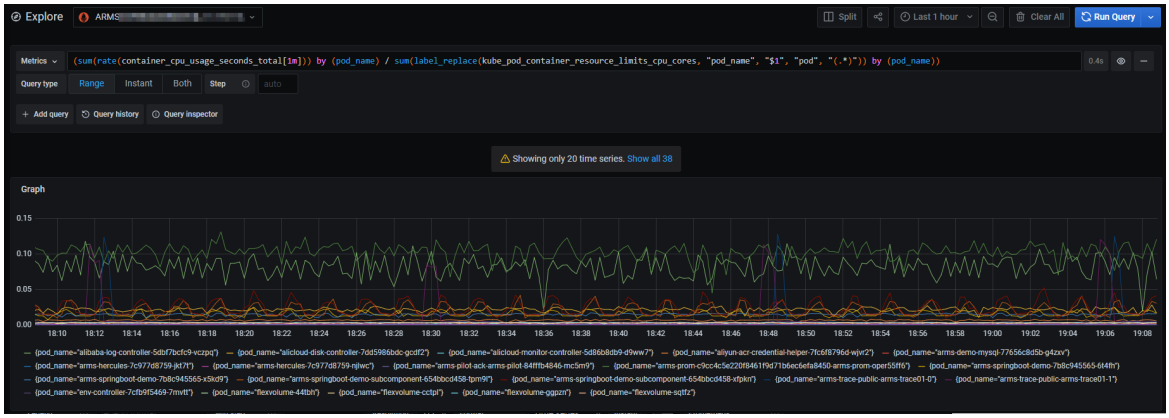


（可选）

步骤五：监控复杂指标

如果需要监控涉及复杂运算的指标，您需要在Prometheus监控中进行数据调试，从而得到相应的PromQL语句。

1. 打开Grafana大盘概览页。
2. 在左侧导航栏单击Explore图标。
3. 在Explore页面顶部下拉框中选择集群，然后在Metrics输入框中输入PromQL语句，单击右上角的Run Query进行调试。



4. 调试成功后，您可以参考上述步骤继续添加大盘或图表，具体操作，请参见步骤四：配置大盘。

步骤六：创建报警

为监控指标创建报警的操作步骤如下：

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
3. 在左侧导航栏，选择**报警配置**。
4. 在报警配置页面右上角，单击**创建报警**。
5. 在**创建报警**面板，执行以下操作：
   i. （可选）从**告警模板**下拉列表，选择模板。
   ii. 在**规则名称**文本框，输入规则名称，例如：网络接收压力报警。
   iii. 在**告警表达式**文本框，输入告警表达式。例如：`(sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m])) / sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m]))) > 0.01`。

   > 注意　PromQL语句中包含的 `$` 符号会导致报错，您需要删除包含$符号的语句中 `=` 左右两边的参数及 `=` 。例如：将 `sum (rate (container_network_receive_bytes_total{instance=~"^$HostIp.*"}[1m]))` 修改为 `sum (rate (container_network_receive_bytes_total[1m]))` 。

   iv. 在**持续时间**文本框，输入持续时间N，当连续N分钟满足告警条件的时候才触发告警。例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。

   > 说明　持续N分钟满足告警条件是指在连续N分钟内，您上面设置的PromQL语句条件都能满足。Prometheus默认数据采集周期为15s，如果没有连续N×60/15=4N个数据点满足告警条件（PromQL语句），就不会发送告警。如Prometheus告警规则默认持续时间为1分钟，既只有连续4个数据点都满足告警条件（PromQL语句）才会触发告警。如果您想在任何一个数据点满足告警条件（PromQL语句）就发送告警，请修改持续时间为0分钟。

   v. 在**告警消息**文本框，输入告警消息。
   vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。
   vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息`，例如：`message:{{$labels.pod_name}}重启`。

   您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：
   viii. 从**通知策略**下拉列表，选择通知策略。

   如何创建通知策略，请参见通知策略。
   ix. 单击**确定**。

报警配置页面显示创建的报警。

相关文档

- 大盘列表
-

17.7. 通过ServiceMonitor创建服务发现

阿里云Prometheus支持使用CRD ServiceMonitor的方式来满足您自定义服务发现的采集需求。通过使用ServiceMonitor，您可以自行定义Pod发现的Namespace范围以及通过 `matchLabel` 来选择监听的Service。本文将基于SpringBoot框架演示如何通过ServiceMonitor创建服务发现。

Demo

您可以通过下载Demo工程，同步体验通过ServiceMonitor创建服务发现的完整过程。

步骤一：创建基础代码依赖

1. 创建一个Maven应用，并在*pom.xml*文件中添加以下依赖。

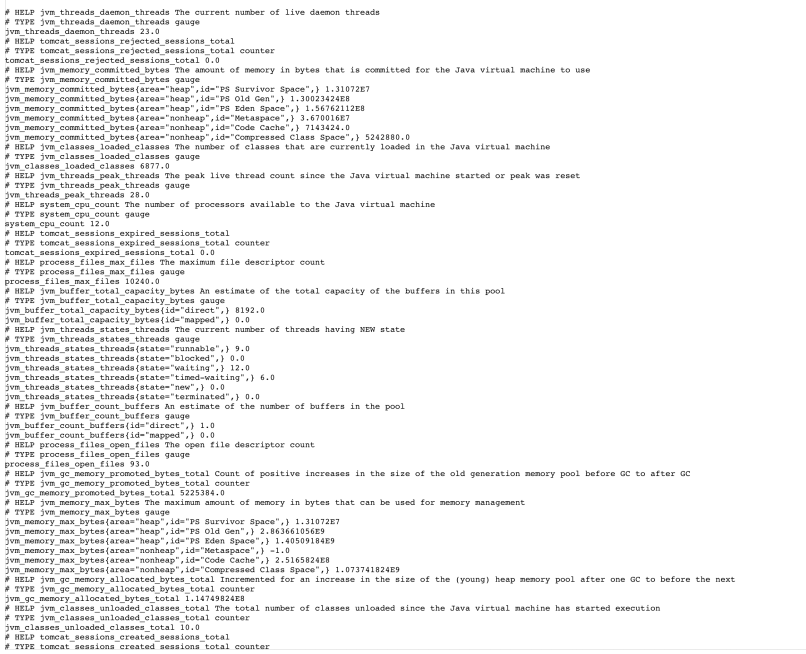
```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
        <version>1.6.6</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

2. 在项目的*src/resources/application.properties*文件中添加以下配置。

```
management.endpoints.web.exposure.include=prometheus
```

3. 启动项目，通过浏览器访问 `http://{host}:{port}/actuator/prometheus` 。

   获取到对应的JVM监控，返回示例如下：



```
# HELP jvm_threads_daemon_threads The current number of live daemon threads
# TYPE jvm_threads_daemon_threads gauge
jvm_threads_daemon_threads 23.0
# HELP tomcat_sessions_rejected_sessions_total
# TYPE tomcat_sessions_rejected_sessions_total counter
tomcat_sessions_rejected_sessions_total 0.0
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="PS Survivor Space",} 1.31072E7
jvm_memory_committed_bytes{area="heap",id="PS Old Gen",} 1.30023424E8
jvm_memory_committed_bytes{area="heap",id="PS Eden Space",} 1.56762112E8
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 3.670016E7
jvm_memory_committed_bytes{area="nonheap",id="Code Cache",} 7143424.0
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 5242880.0
# HELP jvm_classes_loaded_classes The number of classes that are currently loaded in the Java virtual machine
# TYPE jvm_classes_loaded_classes gauge
jvm_classes_loaded_classes 6877.0
# HELP jvm_threads_peak_threads The peak live thread count since the Java virtual machine started or peak was reset
# TYPE jvm_threads_peak_threads gauge
jvm_threads_peak_threads 28.0
# HELP system_cpu_count The number of processors available to the Java virtual machine
# TYPE system_cpu_count gauge
system_cpu_count 12.0
# HELP tomcat_sessions_expired_sessions_total
# TYPE tomcat_sessions_expired_sessions_total counter
tomcat_sessions_expired_sessions_total 0.0
# HELP process_files_max_files The maximum file descriptor count
# TYPE process_files_max_files gauge
process_files_max_files 10240.0
# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="direct",} 8192.0
jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
# HELP jvm_threads_states_threads The current number of threads having NEW state
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{state="runnable",} 9.0
jvm_threads_states_threads{state="blocked",} 0.0
jvm_threads_states_threads{state="waiting",} 12.0
jvm_threads_states_threads{state="timed-waiting",} 6.0
jvm_threads_states_threads{state="new",} 0.0
jvm_threads_states_threads{state="terminated",} 0.0
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="direct",} 1.0
jvm_buffer_count_buffers{id="mapped",} 0.0
# HELP process_files_open_files The open file descriptor count
# TYPE process_files_open_files gauge
process_files_open_files 93.0
# HELP jvm_gc_memory_promoted_bytes_total Count of positive increases in the size of the old generation memory pool before GC to after GC
# TYPE jvm_gc_memory_promoted_bytes_total counter
jvm_gc_memory_promoted_bytes_total 5225384.0
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for memory management
# TYPE jvm_memory_max_bytes gauge
jvm_memory_max_bytes{area="heap",id="PS Survivor Space",} 1.31072E7
jvm_memory_max_bytes{area="heap",id="PS Old Gen",} 2.863661056E9
jvm_memory_max_bytes{area="heap",id="PS Eden Space",} 1.40509184E9
jvm_memory_max_bytes{area="nonheap",id="Metaspace",} -1.0
jvm_memory_max_bytes{area="nonheap",id="Code Cache",} 2.5165824E8
jvm_memory_max_bytes{area="nonheap",id="Compressed Class Space",} 1.073741824E9
# HELP jvm_gc_memory_allocated_bytes_total Incremented for an increase in the size of the (young) heap memory pool after one GC to before the next
# TYPE jvm_gc_memory_allocated_bytes_total counter
jvm_gc_memory_allocated_bytes_total 1.1474982424E8
# HELP jvm_classes_unloaded_classes_total The total number of classes unloaded since the Java virtual machine has started execution
# TYPE jvm_classes_unloaded_classes_total counter
jvm_classes_unloaded_classes_total 10.0
# HELP tomcat_sessions_created_sessions_total
# TYPE tomcat_sessions_created_sessions_total counter
```

步骤二：部署Kubernetes集群

1. 构建一个镜像，并将构建镜像的Dockerfile文件上传至镜像仓库。
2. 参考以下内容创建Deployment。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: micrometer-prometheus
  namespace: default
  labels:
    app: demo-prometheus
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-prometheus
  template:
    metadata:
      labels:
        app: demo-prometheus
    spec:
      containers:
        - name: micrometer-prometheus
          image: manjusakalza/micrometer-prometheus:latest
          ports:
            - containerPort: 8080
```

3. 参考以下内容创建Service。

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-metrics-demo
  namespace: default
  labels:
    micrometer-prometheus-discovery: 'true'
spec:
  selector:
    app: demo-prometheus
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      name: metrics
```

步骤三：创建ServiceMonitor

ServiceMonitor的YAML文件示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: micrometer-demo
  namespace: default
spec:
  endpoints:
    - interval: 15s
      path: /actuator/prometheus
      port: metrics
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      micrometer-prometheus-discovery: 'true'
```

在这段YAML文件中，各代码段的含义如下：

- `metadata` 下的 `name` 和 `namespace` 将指定ServiceMonitor所需的一些关键元信息。
- `spec` 的 `endpoints` 为服务端点，代表Prometheus所需的采集Metrics的地址。 `endpoints` 为一个数组，同时可以创建多个 `endpoints` 。每个 `endpoints` 包含三个字段，每个字段的含义如下：
  - `interval` ：指定Prometheus对当前 `endpoints` 采集的周期。单位为秒，在本次示例中设定为 `15s` 。
  - `path` ：指定Prometheus的采集路径。在本次示例中，指定为 `/actuator/prometheus` 。
  - `port` ：指定采集数据需要通过的端口，设置的端口为步骤二创建Service时端口所设置的 `name` 。在本次示例中，设定为 `metrics` 。
- `spec` 的 `namespaceSelector` 为需要发现的Service的范围。 `namespaceSelector` 包含两个互斥字段，字段的含义如下：
  - `any` ：有且仅有一个值 `true` ，当该字段被设置时，将监听所有符合Selector过滤条件的Service的变动。

- `matchNames`：数组值，指定需要监听的`namespace`的范围。例如，只想监听default和arms-prom两个命名空间中的Service，那么`matchNames`设置如下：

  ```
  namespaceSelector:
    matchNames:
    - default
    - arms-prom
  ```

- `spec`的`selector`用于选择Service。
  在本次示例所使用的Service有micrometer-prometheus-discovery: 'true' Label，所以`selector`设置如下：

  ```
  selector:
    matchLabels:
      micrometer-prometheus-discovery: 'true'
  ```

阿里云Prometheus可以通过以下两种方式创建ServiceMonitor，请选择其中一种方式创建。

通过控制台创建ServiceMonitor

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，单击目标实例名称。
4. 在左侧导航栏单击**服务发现**，然后单击**配置**页签。
5. 在**配置**页面，单击**ServiceMonitor**页签，然后单击**添加ServiceMonitor**。
6. 在**添加ServiceMonitor**对话框，输入YAML文件内容，然后单击**确定**。

通过命令创建ServiceMonitor

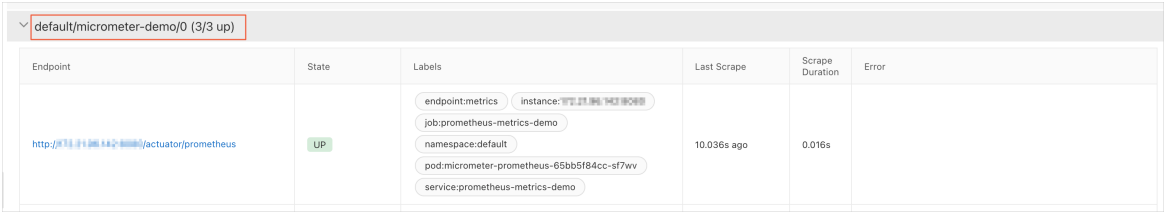
1. 将写好的YAML文件保存至本地。
2. 执行以下命令使YAML文件生效。

   ```
   kubectl apply -f {YAML文件所在的路径}
   ```

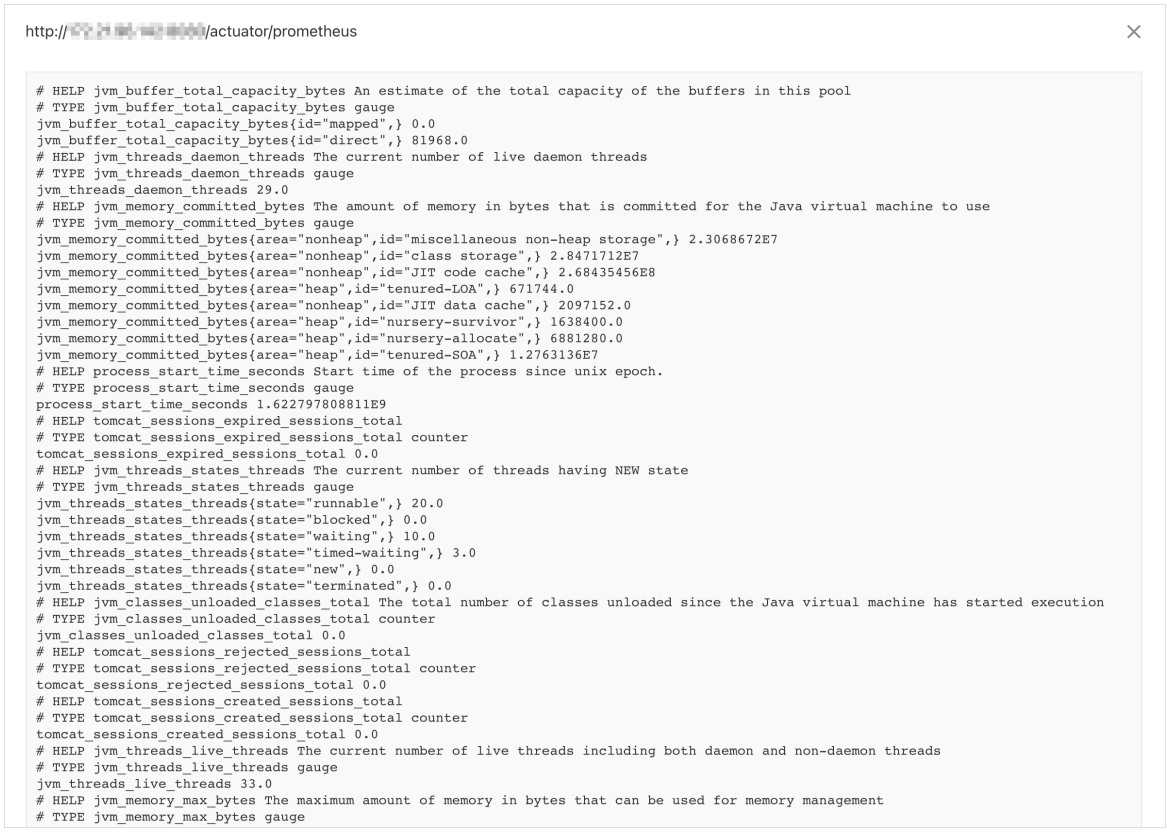
步骤四：验证ServiceMonitor

通过以下操作，验证Prometheus是否成功进行服务发现。

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，单击目标Kubernetes集群名称。
4. 在左侧导航栏单击**服务发现**，然后单击**Targets**页签。
   在**Targets**页签，查看是否存在名称为*{namespace}/{serviceMonitorName}/x*的Target。

5. 单击*{namespace}/{serviceMonitorName}/x*所在行展开Target，然后单击Endpoint链接。
   验证Metrics是否正常。



相关文档

- 通过阿里云Prometheus自定义Grafana大盘
- 创建报警
-
-

17.8. 公网Kubernetes集群接入Prometheus监控

本文介绍如何将公网的Kubernetes集群接入Prometheus监控，并开启鉴权。

前提条件

自建Kubernetes集群已接入阿里云本地数据中心集群。具体操作，请参见创建注册集群并接入本地数据中心集群。

适用场景

- 非阿里云环境的Kubernetes集群接入Prometheus监控。
- 阿里云环境上的Kubernetes集群因为特殊原因需要开启公网数据接入。

> **说明** 如果您的Kubernetes集群已接入阿里云内网，集群接入Prometheus监控的操作，请参见Prometheus实例 for Kubernetes。

步骤一：安装Prometheus Agent

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建prometheus实例**。
3. 在**新建prometheus实例**页面，单击**自建Kubernetes集群**区域。
4. 在**接入自建Kubernetes集群**面板右上角选择Kubernetes集群需要接入的地域，然后完成以下操作：
   i. 自定义Prometheus监控实例的名称，然后单击**新建**。
   ii. 执行以下命令添加阿里云的Helm Repository。

> **注意** 不同地域添加阿里云的Helm Repository的命令不同，请根据实际地域替换命令中的 `{region_id}` ，或直接在**接入自建Kubernetes集群**面板获取准确的添加命令。

```
helm repo add aliyun http://aliacs-k8s-{region_id}.oss-{region_id}.aliyuncs.com/app
/charts-incubator/
```

   iii. 执行**安装Prometheus探针**区域的命令为自建Kubernetes集群安装探针。

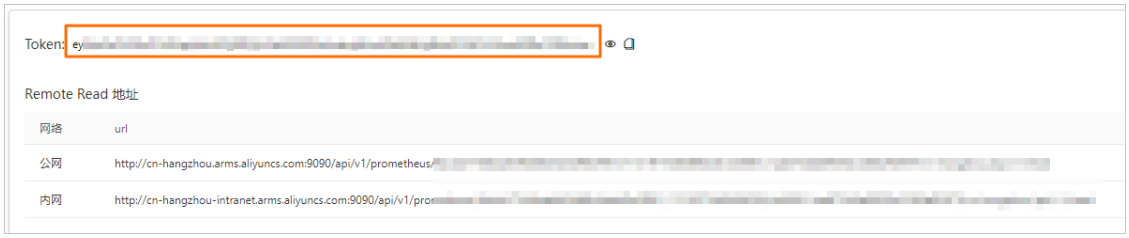
```
helm install arms-prom-operator aliyun/ack-arms-prometheus \
  --namespace arms-prom \
  --set controller.cluster_id=$CLUSTER_ID \     //请在安装Prometheus探针区域获取集群ID。
  --set controller.uid="***" \                  //请在安装Prometheus探针区域获取UID。
  --set controller.region_id=*** \              //请在安装Prometheus探针区域获取Region ID。
  --set controller.vpc_prefix=registry.         //从公网拉取镜像。如果您的镜像存储在阿里云内网，则可以不用配置此参数。
```

> **说明** 关于Helm命令的参数说明，请参见Helm命令参数说明。

步骤二：生成鉴权Token

公网Kubernetes集群远程写入阿里云Prometheus监控时需要做Token校验。

1. 登录Prometheus控制台。
2. 在顶部菜单栏，选择地域。
3. 在**Prometheus监控**页面，单击Kubernetes集群名称。
4. 在左侧导航栏单击**设置**，在右侧页面单击**设置**页签。
5. 在**设置**页签单击**生成token**，复制并保存Token。



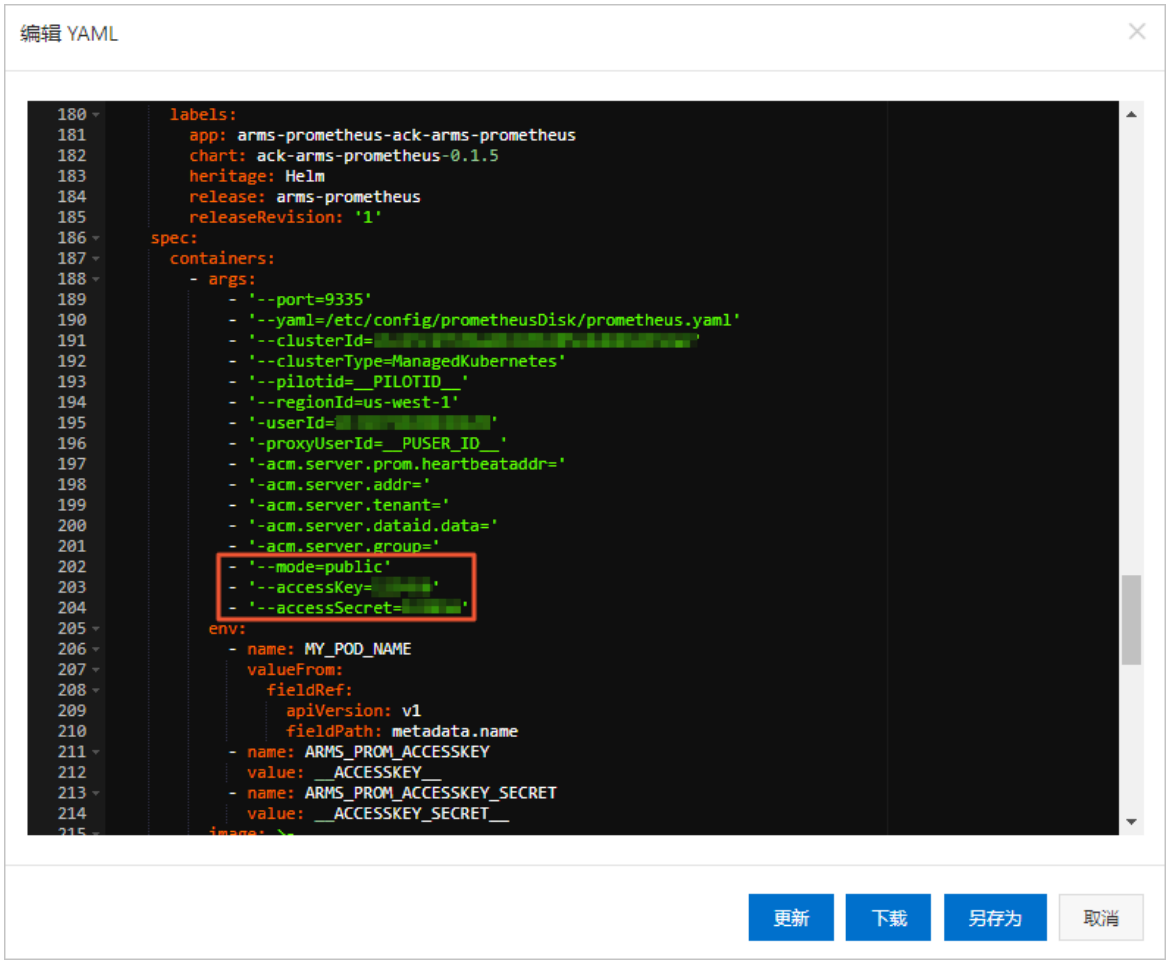
步骤三：配置Kubernetes集群

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。

3. 在**集群列表**页面单击目标集群的名称或右侧**操作**列的**详情**。
4. 在左侧导航栏选择**工作负载 > 无状态**。
5. 在**无状态**页面顶部选择命名空间为arms-prom。
6. 在名称为arms-prometheus-ack-arms-prometheus的Deployment右侧**操作**列，选择**更多 > 查看YAML**。
7. 在**编辑YAML**对话框的 `args` 字段中新增以下参数。

```
- '--mode=public'
- '--accessKey=***'
- '--accessSecret=***'
```

> **说明** `accessKey` 和 `accessSecret` 的值请替换为步骤二中完成生成Token操作所使用的阿里云账号的AccessKey ID和AccessKey Secret。获取AccessKey的操作，请参见获取AccessKey。配置AccessKey和AccessSecret后，Prometheus监控后台将会自动校验步骤二中获取的Token信息。

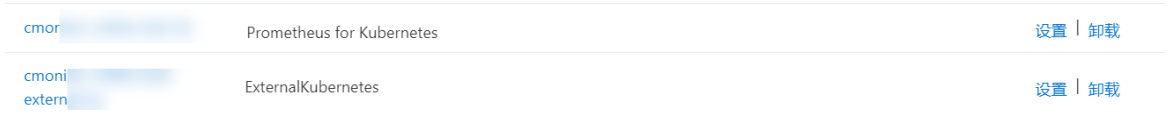


8. YAML文件修改完成后，单击**更新**。

结果验证

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Kubernetes集群所在的地域。

查看已接入的自建Kubernetes集群是否存在监控数据。



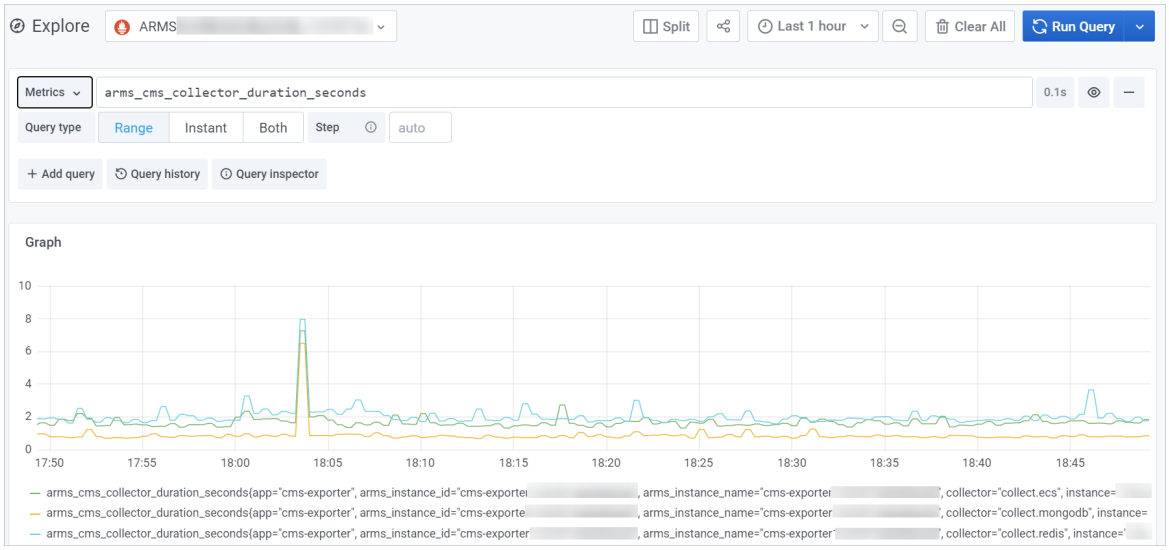
17.9. 使用智能检测算子发现异常数据

Prometheus监控可以通过智能检测算子算法自动地发现KPI时间序列数据中的异常波动，实现时间序列的异常检测，为后续的告警、自动止损、根因分析等提供决策依据，本文提供了Prometheus实例通过在Grafana大盘页面使用智能检测算子发现异常数据波动的操作方法。

检测Prometheus实例的异常数据波动

1. 登录Prometheus控制台。
2. 在页面左上角选择目标地域，然后在Prometheus实例列表中单击目标实例名称对应的已安装大盘。
3. 在左侧导航栏单击Explore图标，然后在左上角的Explore右侧下拉列表选择对应的数据源。
4. 在Metrics下拉列表中选择目标指标，可查看当前指标正常的时序数据，例如目标指标为：

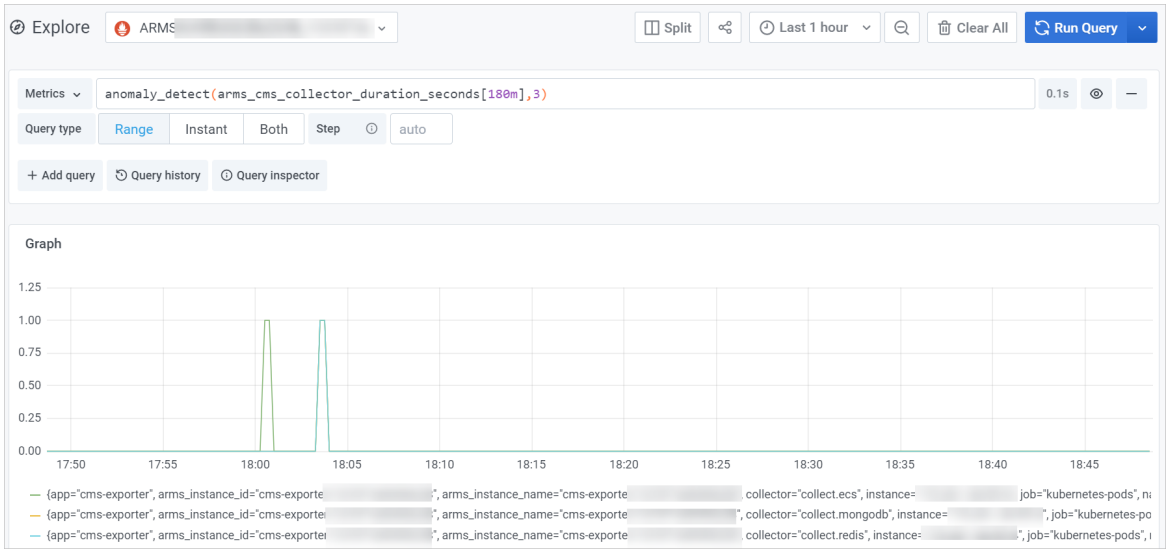
```
arms_cms_collector_duration_seconds
```



5. 在Metrics右侧文本框输入以下PromQL查询语句示例，即异常检测算子。可检测出当前指标在某些时段的数据异常波动情况。

```
anomaly_detect(arms_cms_collector_duration_seconds[180m], 3)
```

> **说明**
>
> - arms_cms_collector_duration_seconds：为上一步骤中的目标指标名称，请根据实际情况替换。
> - 输入的PromQL查询语句数据类型必须是Range vector类型，因此需要在指标名称后增加时间范围“[180m]”，其中时间范围默认选择“[180m]”，参数默认选择“3”。如果提前执行了其他聚合函数操作，则需要将默认的时间范围选择变更为“[180m:]”，使其数据类型变为Range vector类型，例如：anomaly_detect(sum(node_memory_free_bytes)[180m:], 3)。



17.10. 对于自建Kubernetes集群如何自定义Prometheus配置

自建的Kubernetes集群接入Prometheus监控后，无法直接通过控制台进行Prometheus相关配置，而是需要自行通过Kubectl命令方式配置Promethues.yaml、废弃指标、设置Agent副本数以及升级Helm版本。

前提条件

已接入自建的Kubernetes集群，具体操作，请参见Prometheus实例 for Kubernetes。

编辑promethues.yaml文件

1. 创建一个promethues.yaml文件，您可以配置服务发现，实现远程读写地址等。示例如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name:
arms-prom-prometheus-yaml
  namespace: arms-prom
  labels:
    target: arms
    type: prometheus-yaml
data:
  promYaml: |
    remote_write:
       // 替换为您的Remote Write地址。
       - url: "http://ts-xxxxxxxxxxxx.hitsdb.rds.aliyuncs.com:3242/api/prom_write"
         basic_auth:
           //username和password分别对应您阿里云账号的AccessKey ID和AccessKey Secret。
           username: access-key-id
           password: access-key-secret
```

> **说明**
> - `remote_write:` 需要替换为您的Remote Write地址。获取该地址的具体操作，请参见获取Remote Write地址。
> - `username` 、`password` 分别对应您阿里云账号的AccessKey ID和AccessKey Secret。获取AccessKey的操作，请参见获取AccessKey。

2. 在对应目录执行以下命令：

```
kubectl apply -f prometheus.yaml
```

配置废弃指标

当您不再需要采集某些自定义指标时，为了避免这些自定义指标继续产生费用，您可以废弃这些自定义指标。已经配置为废弃的自定义指标将不再继续产生费用。

通过创建drop-metric.yaml文件配置废弃指标。示例如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: arms-prom-drop-metric
  namespace: arms-prom
  labels:
    target: arms
    type: drop-metric
data:
  dropMetric: |
    container_memory_rss
    container_memory_failures_total
    apiserver_request_total
    kube_pod_container_status_waiting_reason
    container_cpu_load_average_10s
    container_memory_max_usage_bytes
```

> **说明** `dropMetric:` 之后的内容就是您需要废弃的指标，请您根据实际情况进行替换。

设置Agent副本数

若您配置的Agent副本数量不足，导致Agent不断产生内存溢出发生重启。您可以调整Agent副本数实现自动扩容。

执行以下命令调整Agent副本数。示例如下：

```
kubectl scale deployment arms-prometheus-ack-arms-prometheus --replicas 3 -n arms-prom
```

> **说明** 命令中的 `3` 为当前设置的Agent副本数，请您根据实际情况进行替换，建议扩容的副本数≥3。

升级Helm版本

阿里云Prometheus监控支持升级Helm版本。查看Helm的版本信息，请参见升级组件版本。

1. 执行以下命令，查看Helm当前版本信息。

```
helm list -n arms-prom
```

2. 执行以下命令，查看Helm最新版本信息。

```
helm search repo ack-arms-prometheus -n arms-prom
```

回显信息如下：

```
wangtengfei@B-F4DHLVDL-0245 temp % helm list -n arms-prom
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                           APP VERSION
arms-prom       arms-prom       1               2021-12-08 17:27:00.075031725 +0800 CST deployed        ack-arms-prometheus-1.1.0       1.0.9
wangtengfei@B-F4DHLVDL-0245 temp % helm search repo ack-arms-prometheus -n arms-prom
NAME                            CHART VERSION   APP VERSION     DESCRIPTION
aliyun/ack-arms-prometheus      1.1.0           1.0.8           ARMS Prometheus Operator
wangtengfei@B-F4DHLVDL-0245 temp %
```

3. 如上图所示，若和的回显信息中查询到的Helm版本号一致，则不需要升级Helm版本，否则，执行以下命令升级Helm版本。

```
helm upgrade arms-prometheus aliyun/ack-arms-prometheus -n arms-prom
```

4. 执行以下命令，可查看Helm历史版本记录。

```
helm history arms-prometheus -n arms-prom
```

回显信息如下：

```
wangtengfei@B-F4DHLVDL-0245 temp % helm search repo ack-arms-prometheus -n arms-prom
NAME                            CHART VERSION   APP VERSION     DESCRIPTION
aliyun/ack-arms-prometheus      1.1.0           1.0.8           ARMS Prometheus Operator
wangtengfei@B-F4DHLVDL-0245 temp % helm history arms-prometheus -n arms-prom
REVISION        UPDATED                         STATUS          CHART                           APP VERSION     DESCRIPTION
1               Tue Nov  2 10:50:21 2021        superseded      ack-arms-prometheus-1.1.0       1.0.8           Install complete
2               Wed Nov 17 15:43:16 2021        superseded      ack-arms-prometheus-1.1.0       1.0.8           Rollback to 1
3               Wed Nov 17 15:57:01 2021        deployed        ack-arms-prometheus-1.1.0       1.0.8           Upgrade complete
wangtengfei@B-F4DHLVDL-0245 temp % helm search repo ack-arms-prometheus -n arms-prom
NAME                            CHART VERSION   APP VERSION     DESCRIPTION
aliyun/ack-arms-prometheus      1.1.0           1.0.8           ARMS Prometheus Operator
wangtengfei@B-F4DHLVDL-0245 temp % helm list  -n arms-prom
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                           APP VERSION
arms-prometheus arms-prom       3               2021-11-17 15:57:01.959674 +0800 CST    deployed        ack-arms-prometheus-1.1.0       1.0.8
wangtengfei@B-F4DHLVDL-0245 temp % helm history arms-prometheus -n arms-prom
REVISION        UPDATED                         STATUS          CHART                           APP VERSION     DESCRIPTION
1               Tue Nov  2 10:50:21 2021        superseded      ack-arms-prometheus-1.1.0       1.0.8           Install complete
2               Wed Nov 17 15:43:16 2021        superseded      ack-arms-prometheus-1.1.0       1.0.8           Rollback to 1
3               Wed Nov 17 15:57:01 2021        deployed        ack-arms-prometheus-1.1.0       1.0.8           Upgrade complete
wangtengfei@B-F4DHLVDL-0245 temp %
```

17.11. 配置Prometheus for VPC服务发现

本文介绍配置Prometheus for VPC服务发现，实现监控采集ECS上暴露的Metric指标服务。

前提条件

已创建Prometheus实例 for VPC，具体操作，请参见Prometheus实例 for VPC。

步骤一：开启ECS服务发现

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建prometheus实例**。
3. 在**新建prometheus实例**页面，单击目标Prometheus实例名称。
4. 在左侧导航栏单击**服务发现**，然后单击**配置**页签。
5. 在**名称**为vpc-ecs-service-discovery的**操作**列单击off。然后在弹出的提示框中单击**开启**，开启ECS服务发现。

开启ECS服务发现后，将会开启ECS数据采集。

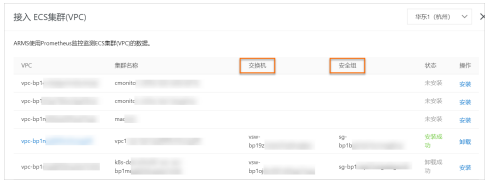
步骤二：修改ECS服务发现相关配置

1. 在**名称**为vpc-ecs-service-discovery的**操作**列单击**详情**。
2. 在弹出的**Yaml配置**对话框中修改metrics_path或者port。如下图所示，若您的服务端口为9090/metrics，只需要修改port值为9090。

（可选）步骤三：检查安全组

如果您无法采集到ECS的监控数据，通常是由于安全组访问规则限制导致，建议您可以按照以下步骤优先检查安全组访问规则。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建prometheus实例**。
3. 在**新建prometheus实例**页面，单击**Prometheus实例 for VPC**区域。
   在**接入ECS集群(VPC)**面板显示当前地域下的所有VPC列表。
4. 在**接入ECS集群(VPC)**面板的目标VPC右侧**安全组**和**交换机**列，可以获取创建该Prometheus实例时选择的安全组和交换机信息。



5. 登录ECS管理控制台，在上一步骤中获取的对应安全组中添加允许规则。

> **说明** 添加允许规则时，**授权对象**的源IP地址为安装Prometheus监控时选择的交换机对应的网段（即上一步骤中获取的交换机对应的网段）。

（可选）步骤四：过滤ECS

如果您的VPC内只有部分ECS提供指标查询服务，并且只需要采集部分ECS的数据，可选择过滤ECS，具体如下。

1. 登录Prometheus控制台。

2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建prometheus实例**。
3. 在**新建prometheus实例**页面，单击目标Prometheus实例名称。
4. 在左侧导航栏单击**服务发现**，然后单击**配置**页签。
5. 在**名称**为vpc-ecs-service-discovery的**操作**列单击**详情**。
6. 在弹出的**Yaml配置**对话框中输入Prometheus.yaml，然后单击**确定**。如下所示的代码段是阿里云ECS服务发现配置样例Prometheus.yaml内容，具体请参见样例Prometheus.yaml。

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 30s
scrape_configs:
- job_name: _aliyun-prom/ecs-sd
  honor_timestamps: true
  scrape_interval: 30s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: https
  aliyun_sd_configs:
    - port:  8888                                    # 服务发现后的prometheus抓取采集点port
      user_id: <aliyun userId>                       # Aliyun用户身份表示id userId，填写会为discovery target带上__meta_ecs_user_id的label，可不填写
      refresh_interval: 30s
      region_id: cn-hangzhou                         # 设置获取ECS的regionId
      access_key: <aliyun ak>                        # Aliyun鉴权字段AK
      access_key_secret: <aliyun sk>                 # Aliyun鉴权字段SK
      tag_filters:                                   # Aliyun ECS tag filter， 按tagKey tagValue匹配筛选实例
       - key: 'testK'
         values: ['*', 'test1*']
       - key: 'testM'
         values: ['test2*']
#      limit: 40                                     # 从接口取到的最大实例个数限制，不填为获取所有ECS实例
  relabel_configs:
#   1. 手动设置使用ECS的哪种IP
#   默认ECS会按 经典网络公网IP > 经典网络内网IP > VPC网络公网IP > VPC网络内网IP 的顺序查找并赋予此ECS的采集IP，此时的采集点port为aliyun_sd_configs.port设置
#   用户可用过一下relabel设置，手动设置ECS的采集IP
    - source_labels: [__meta_ecs_public_ip]        # 经典网络公网ip __meta_ecs_public_ip
#    - source_labels: [__meta_ecs_inner_ip]        # 经典网络内网ip __meta_ecs_inner_ip
#    - source_labels: [__meta_ecs_eip]             # VPC网络 公网ip __meta_ecs_eip
#    - source_labels: [__meta_ecs_private_ip]      # VPC网络 内网ip __meta_ecs_private_ip
      regex: (.*)
      target_label: __address__
      replacement: $1:<port>                        # 注意此处为手动设置relabel时的采集port
#   2. 按ECS属性过滤 keep为只保留此条件筛选到的target，drop为过滤掉此条件筛选到的target
#   __meta_ecs_instance_id          实例id
#   __meta_ecs_region_id            实例regionId 注意配置中aliyun_sd_configs.region_id决定了获取的ECS的regionId
#   __meta_ecs_status               实例状态 Running：运行中、Starting：启动中、Stopping：停止中、Stopped：已停止
#   __meta_ecs_zone_id              实例区域id
#   __meta_ecs_network_type         实例网络类型  classic：经典网络、vpc：VPC
#   __meta_ecs_tag_<TagKey>         实例tag TagKey为tag的名
    - source_labels:  ["__meta_ecs_instance_id"]
      regex: ".+"       # or other value regex
      action: keep      # keep / drop
```

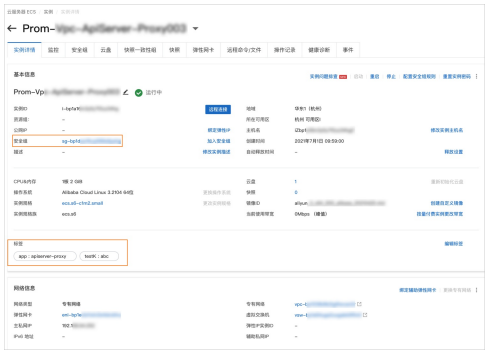
例如，若您只需要采集标签名为app，标签值为myNginx的ECS，则可以进行如下配置。

```
tag_filters:
   - key: 'app'
     values: ['myNginx']
```

（可选）步骤五：指标Relabel

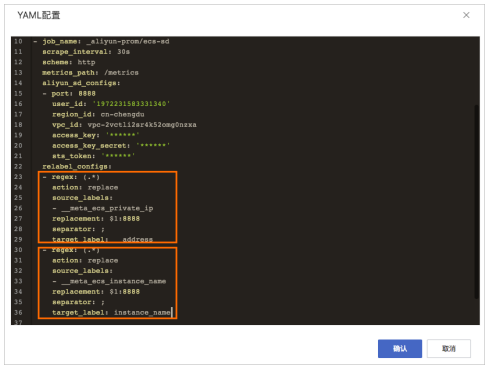
如果您需要在监控采集的数据中获取主机名以及标签信息，可参考如下操作。

1. 登录ECS管理控制台，查看ECS的标签信息。

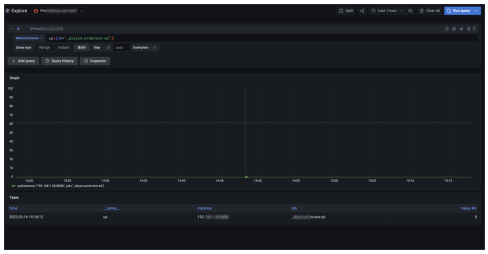


2. 登录Prometheus控制台。
3. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建prometheus实例**。
4. 在**新建prometheus实例**页面，单击目标Prometheus实例名称。
5. 在左侧导航栏单击**服务发现**，然后单击**配置**页签。
6. 在**名称**为vpc-ecs-service-discovery的**操作**列单击**详情**。
7. 在弹出的**Yaml配置**对话框中输入如下Prometheus.yaml，配置需要采集的标签及主机名，然后单击**确定**。

```
relabel_configs:
- regex: (.*)
  action: replace
  source_labels:
  - __meta_ecs_private_ip
  replacement: $1:88889
  separator: ;
  target_label: __address__
- regex: (.*)
  action: replace
  source_labels:
  - __meta_ecs_tag_app
  replacement: $1
  separator: ;
  target_label: tag_app
- regex: (.*)
  action: replace
  source_labels:
  - __meta_ecs_instance_name
  replacement: $1
  separator: ;
  target_label: instance_name
```



8. 查看采集到的监控数据。
   i. 打开Grafana大盘概览页。
   ii. 在左侧导航栏单击Explore图标。
   iii. 在Explore页面顶部下拉框中选择集群，然后在Metrics输入框中输入PromQL语句，查看采集到的监控数据。



17.12. 使用阿里云Prometheus监控腾讯云产品

本文介绍如何使用阿里云Prometheus监控服务实现对腾讯云云产品的监控。

前提条件

- 已创建阿里云容器服务K8s集群。具体操作，请参见创建Kubernetes专有版集群。
- 阿里云容器服务K8s集群已接入阿里云Prometheus监控。具体操作，请参见Prometheus实例 for 容器服务。

背景信息

利用开源的腾讯云监控Exporter（qcloud exporter）可以将腾讯云监控支持的产品监控指标自动批量导出为Prometheus格式，使用Prometheus拉取qcloud exporter指标即可实现对腾讯云产品的监控。

腾讯云云监控（Cloud Monitor，简称CM）是一项可对腾讯云云产品资源实时监控和告警的服务，为用户提供统一监控云服务器、云数据库等所有云产品的平台。qcloud exporter是Prometheus Third-party exporters一种实现，当前支持的腾讯云产品列表如下：

产品	命名空间	支持的指标
MongoDB	QCE/CMONGO	指标详情
CDB	QCE/CDB	指标详情
Redis标准版	QCE/REDIS	暂无指标详情说明

产品	命名空间	支持的指标
Redis集群版	QCE/REDIS_CLUSTER	暂无指标详情说明
Redis内存版监控指标	QCE/REDIS_MEM	指标详情
CVM	QCE/CVM	指标详情
COS	QCE/COS	指标详情
CDN	QCE/CDN	指标详情
CLB（公网）	QCE/LB_PUBLIC	指标详情
CLB（7层）	QCE/LOADBALANCE	指标详情
NAT	QCE/NAT_GATEWAY	指标详情
物理专线	QCE/DC	指标详情
专用通道	QCE/DCX	指标详情
云硬盘	QCE/CBS	指标详情
SqlServer	QCE/SQLSERVER	指标详情
MariaDB	QCE/MARIADB	指标详情
Elasticsearch	QCE/CES	指标详情
CMQ队列服务	QCE/CMQ	指标详情
CMQ主题订阅	QCE/CMQTOPIC	指标详情
PostgreSQL	QCE/POSTGRES	指标详情
CKafka实例	QCE/CKAFKA	指标详情
Memcached	QCE/MEMCACHED	暂无指标详情说明
Lighthouse	QCE/LIGHTHOUSE	暂无指标详情说明
分布式数据库TDSQL MySQL实例	QCE/TDMYSQL	指标详情
弹性公网IP	QCE/LB	指标详情

操作流程

通过阿里云Prometheus配置监控腾讯云的操作流程如下图所示。

步骤一：部署qcloud exporter

1. 构造镜像。

```
git clone https://github.com/tencentyun/tencentcloud-exporter.git
make build
```

2. 定义产品实例配置。
   - 配置云API的credential认证信息
   - 配置产品products指标、实例导出信息

   例如，若您需要导出云服务器CVM所有指标和实例信息，您可以执行如下代码段。

```
credential:
  access_key: "access_key"  #云API的SecretId
  secret_key: "secret_key"  #云API的SecretKey
  region: "ap-nanjing"      #实例所在区域信息
rate_limit: 15              #云监控拉数据接口最大限制，20/秒，1200/分钟，https://cloud.tencent.com/document/product/248/31014
products:
  - namespace: QCE/CVM      #指标详情：https://cloud.tencent.com/document/product/248/6843
    all_metrics: true       #导出支持的所有指标
    all_instances: true     #导出region下的所有实例
    #only_include_metrics: []
    #only_include_instances: [ins-xxxxxxxx]
    extra_labels: [InstanceId, InstanceName] #将实例的字段作为指标的lables导出
    #statistics_types: [last]
    #period_seconds: 60
    #metric_name_type: 2
```

> **说明** 更多qcloud.yaml配置详情，请参见 tencentcloud-exporter。

3. 部署qcloud exporter。

   将以上配置文件构建到Docker镜像并将镜像上传到镜像仓库，如DockerHub、阿里云容器镜像服务ACR。

   i. 登录容器服务管理控制台。
   ii. 在左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面，单击目标集群右侧**操作**列**应用管理**。

iv. 创建容器组。

a. 在左侧导航栏，选择**工作负载 > 无状态**。

b. 在**无状态**页面，单击**使用YAML创建资源**。

c. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  generation: 5
  labels:
    app: qcloud-exporter-demo
  name: qcloud-exporter-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: qcloud-exporter-demo
  template:
    metadata:
      labels:
        app: qcloud-exporter-demo
    spec:
      containers:
        - args:
            - '--config.file=/usr/local/etc/qcloud-cvm-product.yml'
          image: 'registry.cn-hangzhou.aliyuncs.com/fuling/qcloud-exporter:v0.1
'
          imagePullPolicy: Always
          name: qcloud-exporter
          ports:
            - containerPort: 9123
              name: web-normal
              protocol: TCP
```

无状态页面会显示创建的容器组。

v. 创建服务。

a. 在左侧导航栏，选择**服务与路由 > 服务**。

b. 在**服务**页面，单击**使用YAML创建资源**。

c. 在**创建**页面的**模板**代码框输入以下内容，然后单击**创建**。

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: qcloud-exporter-demo
  name: qcloud-exporter-demo-svc
  namespace: default
spec:
  ports:
    - name: qcloud-exporter-metrics
      port: 9123
      protocol: TCP
      targetPort: 9123
  selector:
    app: qcloud-exporter-demo
```

服务页面会显示创建的服务。

步骤二：配置服务发现

配置阿里云Prometheus监控的服务发现以接收qcloud exporter数据的操作步骤如下：

> **注意** 请确认阿里云容器服务K8s集群已接入Prometheus监控。具体操作，请参见Prometheus实例 for 容器服务。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标实例名称。
3. 在左侧导航树单击**服务发现**，然后单击**配置**页签。
4. 在**配置**页面单击**ServiceMonitor**页签，然后单击**添加ServiceMonitor**，在弹出的**添加ServiceMonitor**对话框中输入以下内容，单击**确定**。

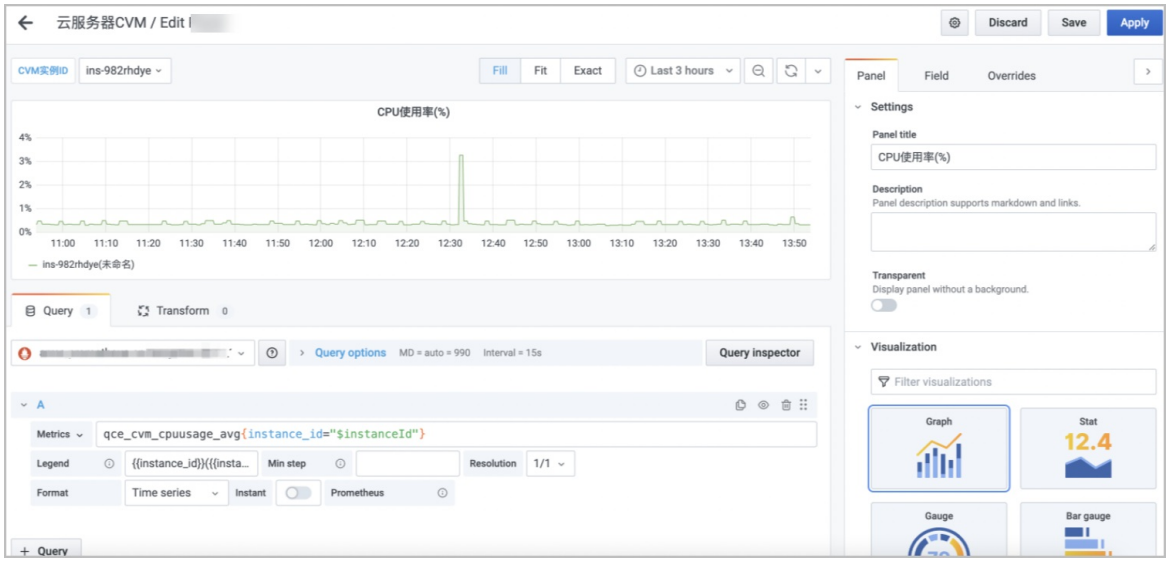
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: qcloud-exporter-sm
  namespace: default
spec:
  endpoints:
  - interval: 60s
    path: /metrics
    port: qcloud-exporter-metrics
    scrapeTimeout: 60s
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      app: qcloud-exporter-demo
```

ServiceMonitor页面会显示配置的服务发现。

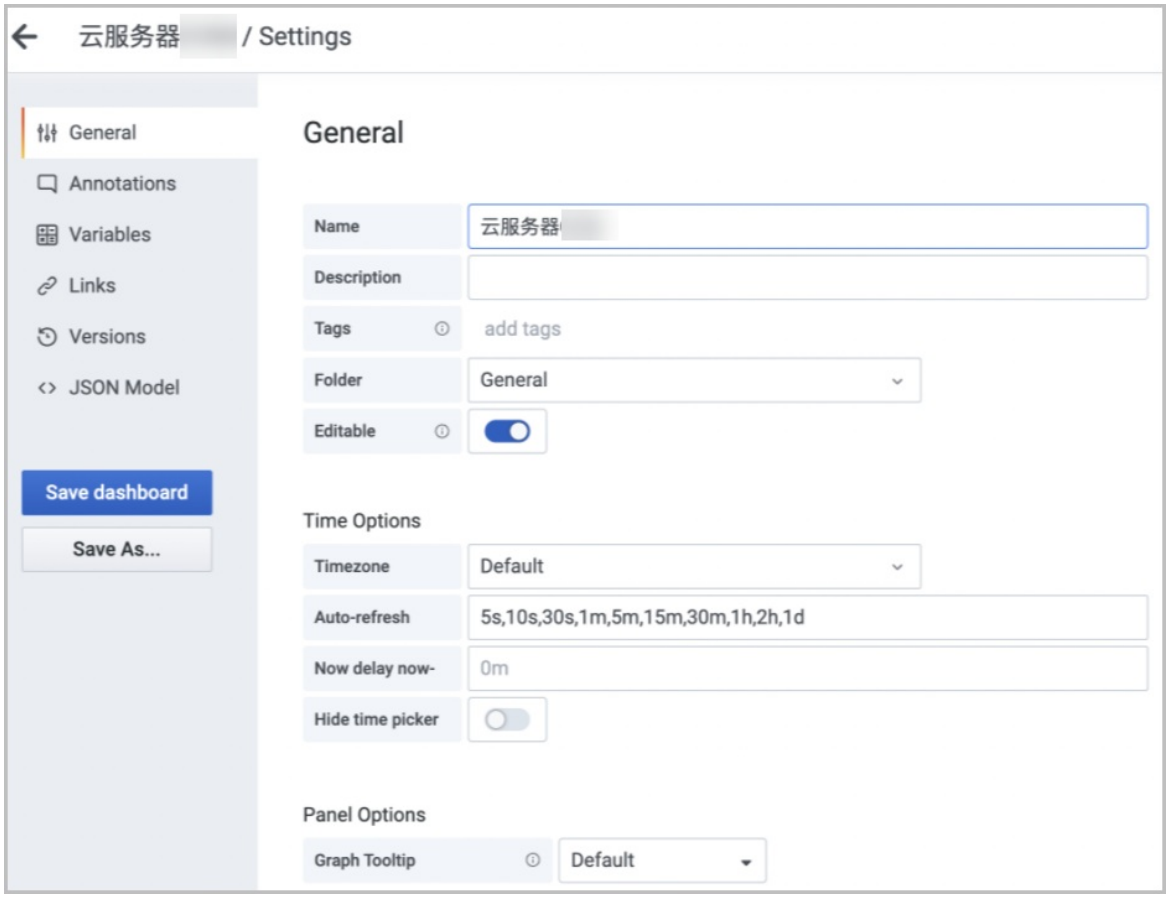
步骤三：配置Grafana大盘

配置Grafana大盘以展示监控数据的操作步骤如下：

1. 打开Grafana大盘概览页。
2. 在左侧导航栏选择+ > Dashboard。
3. 在Dashboard页面单击+Add new panel，在Edit Panel页面的Query区域的下拉列表中选择Prometheus数据源，然后在A折叠面板的Metrics中填写PromQL，然后在页面右侧Panel页签中设置Panel title等配置项，查看您所需要的监控数据信息。如下图所示展示了CPU使用率的监控数据详情。



4. 参见上一步骤，继续添加Panel、Row、Variables等，然后设置大盘名称，单击Save Dashboard。



配置完毕后的Grafana大盘如下图所示。

步骤四：创建Prometheus监控报警

1. 登录Prometheus控制台。

2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
3. 在左侧导航栏，选择**报警配置**。
4. 在报警配置页面右上角，单击**创建报警**。
5. 在**创建报警**面板，执行以下操作：
   i. （可选）从**告警模板**下拉列表，选择模板。
   ii. 在**规则名称**文本框，输入规则名称，例如：网络接收压力报警。
   iii. 在**告警表达式**文本框，输入告警表达式。例如：`qce_cvm_cpuusage_avg >= 80` 。
   iv. 在**持续时间**文本框，输入持续时间N，当连续N分钟满足告警条件的时候才触发告警。例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。
   v. 在**告警消息**文本框，输入告警消息。
   vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。
   vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息` ，例如：`message:{{$labels.pod_name}}重启` 。
   您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：
   - 报警规则表达式指标中携带的标签。
   - 通过报警规则创建的标签。
   viii. 从**通知策略**下拉列表，选择通知策略。
   如何创建通知策略，请参见通知策略。
   ix. 单击**确定**。

报警配置页面显示创建的报警。

17.13. Spring Boot应用如何快速接入Prometheus监控

在使用Spring Boot应用过程中，为了对系统的状态进行持续地观测，您可以将Spring Boot应用接入Prometheus监控。本文介绍如何将Spring Boot应用快速接入Prometheus监控。

背景信息

对于开发者而言，大部分传统SSM结构的MVC应用背后的糟糕体验都是来自于搭建项目时的大量配置，稍有不慎就可能导致配置出错。为了解决这个问题，Spring Boot应运而生。Spring Boot的核心价值就是自动配置，只要存在相应Jar包，Spring Boot可以自动配置，如果默认配置不能满足需求，您还可以替换掉自动配置类，使用自定义配置快速构建企业级应用程序。

构建Spring Boot应用以及该应用上线之后，您需要对该应用进行监测。一般来说，搭建一套完整易用的监测系统主要包含以下几个关键部分。

目前，行业常见的收集监测数据方式主要分为推送（Push）和抓取（Pull）两个模式。以越来越广泛应用的Prometheus监测体系举例，Prometheus监控就是以抓取（Pull）模式运行的典型系统。应用及基础设施的监测数据以OpenMetrics标准接口的形式暴露给Prometheus监控，然后由Prometheus监控进行定期抓取并长期存储。

OpenMetrics，是云原生、高度可扩展的指标协议。 OpenMetrics定义了大规模上报云原生指标的事实标准，并支持文本表示协议和Protocol Buffers协议，文本表示协议在其中更为常见，也是在Prometheus监控进行数据抓取时默认采用的协议。下图是一个基于OpenMetrics格式的指标表示格式样例。

```
# TYPE acme_http_router_request_seconds summary
# UNIT acme_http_router_request_seconds seconds
# HELP acme_http_router_request_seconds Latency though all of ACME's HTTP request router.
acme_http_router_request_seconds_sum{path="/api/v1",method="GET"} 9036.32
acme_http_router_request_seconds_count{path="/api/v1",method="GET"} 807283.0
acme_http_router_request_seconds_created{path="/api/v1",method="GET"} 1605281325.0
acme_http_router_request_seconds_sum{path="/api/v2",method="POST"} 479.3
acme_http_router_request_seconds_count{path="/api/v2",method="POST"} 34.0
acme_http_router_request_seconds_created{path="/api/v2",method="POST"} 1605281325.0
# TYPE go_goroutines gauge
# HELP go_goroutines Number of goroutines that currently exist.
go_goroutines 69
# TYPE process_cpu_seconds counter
# UNIT process_cpu_seconds seconds
# HELP process_cpu_seconds Total user and system CPU time spent in seconds.
process_cpu_seconds_total 4.20072246e+06
# EOF
```

说明 指标的数据模型由指标（Metric）名，以及一组Key/Value标签（Label）定义的，具有相同的度量名称以及标签属于相同时序集合。例如acme_http_router_request_seconds_sum{path="/api/v1",method="GET"} 可以表示指标名为acme_http_router_request_seconds_sum，标签method值为POST的一次采样点数据。采样点内包含一个Float64值和一个毫秒级的UNIX时间戳。随着时间推移，这些收集起来的采样点数据将在图表上实时绘制动态变化的线条。

目前，对于云原生体系下的绝大多数基础组件能够支持OpenMetrics的文本协议格式暴露指标，对于暂不能支持自身暴露指标的组件， Prometheus社区也存在极其丰富的Prometheus Exporter供开发及运维人员使用。这些组件（或Exporter）通过响应来自Prometheus监控的定期抓取请求来及时地将自身的运行状况记录到Prometheus监控以便后续的处理及分析。对于应用开发者，您还可以通过Prometheus监控的多语言SDK，进行代码埋点，将自身的业务指标也接入到上述的Prometheus生态当中。

在获取应用或基础设施运行状态、资源使用情况，以及服务运行状态等直观信息后，通过查询和分析多类型、多维度信息能够方便的对节点进行跟踪和比较。同时，通过标准易用的可视化大盘去获知当前系统的运行状态。比较常见的解决方案就是Grafana，作为开源社区中目前热度很高的数据可视化解决方案，Grafana提供了丰富的图表形式与模板。在阿里云Prometheus监控服务中，也为您提供了基于Grafana全托管版的监测数据查询、分析及可视化。

当业务即将出现故障时，监测系统需要迅速反应并通知管理员，从而能够对问题进行快速的处理或者提前预防问题的发生，避免出现对业务的影响。当问题发生后，管理员需要对问题进行认领和处理。通过对不同监测指标以及历史数据的分析，能够找到并解决根源问题。

线程池改造

接入流程概述

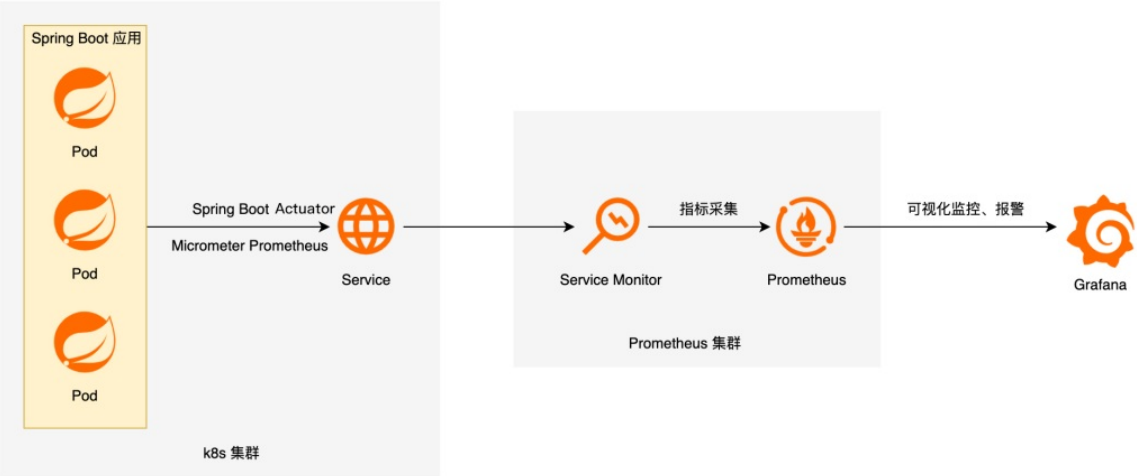
针对Spring Boot应用，社区提供了开箱即用的Spring Boot Actuator框架，方便Java开发者进行代码埋点和监测数据收集、输出。从Spring Boot 2.0开始，Actuator将底层改为Micrometer，同时提供了更强、更灵活的监测能力。Micrometer是一个监测门面，可以类比成监测界的Slf4j，借助Micrometer，应用则能够对接各种监测系统。例如，AppOptics、Datadog、Elastic、InfluxDB以及Prometheus监控等。

Micrometer在将Prometheus监控指标对接到Java应用的指标时，支持应用开发者用三个类型的语义来映射：

Micrometer指标类型	Prometheus监控指标类型	典型用途
Counter	Counter	计数器，单调递增场景。例如，统计PV和UV，接口调用次数等。
Gauge	Gauge	持续波动的变量。例如，资源使用率、系统负载、请求队列长度等。
Timer	Histogram	统计数据分布。例如，统计某接口调用延时的P50、P90、P99等。
DistributionSummary	Summary	统计数据分布，与Histogram用途类似。

- Micrometer中的Counter指标类型对应于Prometheus监控中的Counter指标类型，用来描述一个单调递增的变量。如某个接口的访问次数、缓存命中或者访问总次数等。Timer在逻辑上蕴含了Counter，即如果使用Timer采集每个接口的响应时间，必然也会采集访问次数。因此无需为某个接口同时指定Timer与Counter两个指标。
- MicroMeter中的Gauge指标类型对应于Prometheus监控中的Gauge指标类型，用来描述在一个范围内持续波动的变量。如CPU使用率、线程池任务队列数等。
- MicroMeter中的Timer指标类型对应于Prometheus监控中的Histogram，用来描述与时间相关的数据。如某个接口RT时间分布等。
- Micrometer中的DistributionSummary指标类型对应Prometheus监控中的Summary指标类型 ，与Histogram类似，Summary也是用于统计数据分布的，但由于数据的分布情况是在客户端计算完成后再传入Prometheus监控进行存储，因此Summary的结果无法在多个机器之间进行数据聚合，无法统计全局视图的数据分布，使用起来有一定局限性。

当您需要把部署在Kubernetes集群中的Spring Boot应用接入到Prometheus监控时，需要按照代码埋点>部署应用>服务发现这个流程来进行。



首先，您需要在代码中引入Spring Boot Actuator相关Maven依赖，并对您需要监测的数据进行注册，或对Controller内的方法打上响应的注解。

其次，您需要将埋点后的应用部署在Kubernetes中，并向Prometheus监控注册向应用拉取监测数据的端点（即Prometheus监控的服务发现）。阿里云Prometheus服务提供了使用ServiceMonitor CRD进行服务发现的方法。

最后，在目标应用的监测采集端点被Prometheus监控成功发现后，您就可以在Grafana上配置数据源及相应的大盘。同时您也可以根据某些关键指标进行对应的告警配置。

最终目标

通过将部署在Kubernetes集群中的Spring Boot应用接入到Prometheus监控，希望能够实现以下几点目标：

- 监测系统的入口：Frontend服务是一个基于SpringMVC开发的入口应用，承接外部的客户流量，这里主要关注的是外部接口的关键RED指标。例如，调用率Rate、失败数Error、请求耗时Duration。
- 监测系统的关键链路：对后端服务critical path上的对象进行监测。例如，线程池的队列情况、进程内Guava Cache缓存的命中情况。
- 实现对业务强相关的自定义指标进行监测。例如，某个接口的UV等。
- 实现对JVM GC及内存使用情况进行监测。
- 实现对上述指标进行统一汇聚展示、以及配置关键指标的告警。

这里选取一个基于Spring Boot和Spring Cloud Alibaba构建的云原生微服务应用，为您介绍部署在Kubernetes集群上的Spring Boot微服务应用如何进行Prometheus接入的具体接入流程。

步骤一：引入Spring Boot Actuator依赖，进行初始配置

1. 执行如下代码段，引入Spring Boot Actuator的相关依赖。

```
<!-- spring-boot-actuator依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<!-- prometheus依赖 -->
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

2. 在application.properties中添加相关配置暴露监测数据端口。例如，端口为8091。

```
# application.properties添加以下配置用于暴露指标
spring.application.name=frontend
management.server.port=8091
management.endpoints.web.exposure.include=*
management.metrics.tags.application=${spring.application.name}
```

配置成功后，即可访问该应用的8091端口，然后您可以在该端口的/actuator/prometheus路径中获取OpenMetrics标准的监测数据。

步骤二：代码埋点及改造

若要获取某个API接口的RED指标，您需要在对应的接口方法上打@Timed注解。这里以index页面接口为例打@Timed注解，如下代码段所示。

```
@Timed(value = "main_page_request_duration", description = "Time taken to return main page"
, histogram = true)
@ApiOperation(value = "首页", tags = {"首页操作页面"})
@GetMapping("/")
public String index(Model model) {
    model.addAttribute("products", productDAO.getProductList());
    model.addAttribute("FRONTEND_APP_NAME", Application.APP_NAME);
    model.addAttribute("FRONTEND_SERVICE_TAG", Application.SERVICE_TAG);
    model.addAttribute("FRONTEND_IP", registration.getHost());
    model.addAttribute("PRODUCT_APP_NAME", PRODUCT_APP_NAME);
    model.addAttribute("PRODUCT_SERVICE_TAG", PRODUCT_SERVICE_TAG);
    model.addAttribute("PRODUCT_IP", PRODUCT_IP);
    model.addAttribute("new_version", StringUtils.isBlank(env));
    return "index.html";
}
```

说明 其中value即为暴露到/actuator/prometheus中的指标名字，histogram=true表示暴露这个接口请求时长的histogram直方图类型指标，便于您后续计算P90、P99等请求时间分布情况。

若您的应用中使用了进程内缓存库（例如，最常见的Guava Cache库等）且需要追踪进程内缓存的运行状况，您可以按照Micrometer提供的修饰方法，对于待监测的关键对象进行封装。

1. 注入MeterRegistry，这里注入的具体实现是PrometheusMeterRegistry，由Spring Boot自行注入即可。
2. 使用工具类API包装本地缓存，即如下图中的GuavaCacheMetrics.monitor。
3. 开启缓存数据记录，即调用.recordStats()方法。
4. 为Cache对象命名，用于生成对应的指标。

```
@Resource
private static MeterRegistry meterRegistry;

private static final LoadingCache<String, String> REFRESH_CACHE = GuavaCacheMetrics.monitor(
        meterRegistry,
        CacheBuilder.newBuilder()
                .recordStats()
                .build(new CacheLoader<String, String>() {
                    @Override
                    public String load(String key) {
                        // 此处是具体的load操作，由开发者自行实现
                        return "";
                    }
                }),
        cacheName: "refresh-cache");
```

1. 注入MeterRegistry，这里注入的具体实现是PrometheusMeterRegistry。
2. 使用工具类API包装线程池。
3. 为线程池命名，用于生成对应的指标。

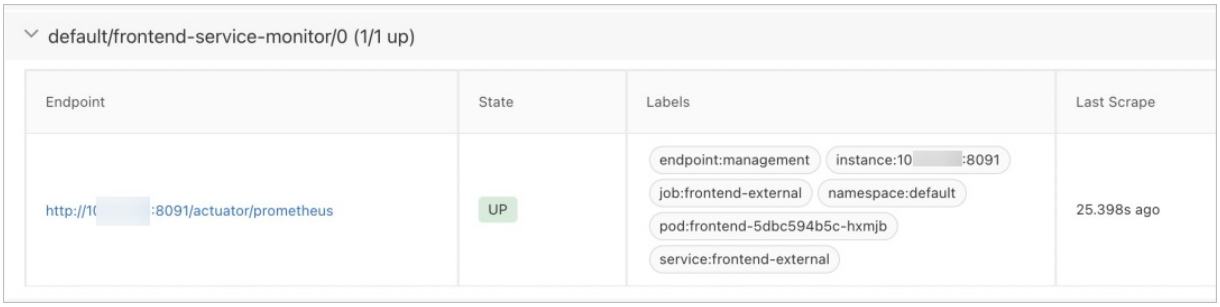
```
@Resource
private static MeterRegistry meterRegistry;

private static final ScheduledExecutorService REFRESH_EXECUTOR = ExecutorServiceMetrics.monitor(
        meterRegistry,
        Executors.newScheduledThreadPool( corePoolSize: 1,
                new ThreadFactory() {
                    public Thread newThread(Runnable r) {
                        Thread thread = new Thread(r);
                        thread.setDaemon(true);
                        thread.setName("dubbo.outlier.refresh-" + thread.getId());
                        return thread;
                    }
                }),
        executorServiceName: "refresh-executor"
);
```

在开发过程中还会涉及许多业务强相关的自定义指标，为了监测这些指标，在往Bean中注入MeterRegistry后，您还需要按照需求和对应场景构造Counter、Gauge或Timer来进行数据统计，并将其注册到MeterRegistry进行指标暴露，示例如下。

```
@Service
public class DemoService {
    Counter visitCounter;
    public DemoService(MeterRegistry registry) {
        visitCounter = Counter.builder("visit_counter")
            .description("Number of visits to the site")
            .register(registry);
    }
    public String visit() {
        visitCounter.increment();
        return "Hello World!";
    }
}
```

至此，您对应用的代码改造工作已全部完成。然后您需要将应用镜像重新构建并部署到已安装Prometheus监控的Kubernetes集群中，并在Prometheus监控控制台中配置ServiceMonitor，进行服务发现。更多信息，请参见Prometheus实例 for 容器服务和管理Kubernetes集群服务发现。

ServiceMonitor配置完成后，您可以在Targets列表中查看到刚注册的Service应用。

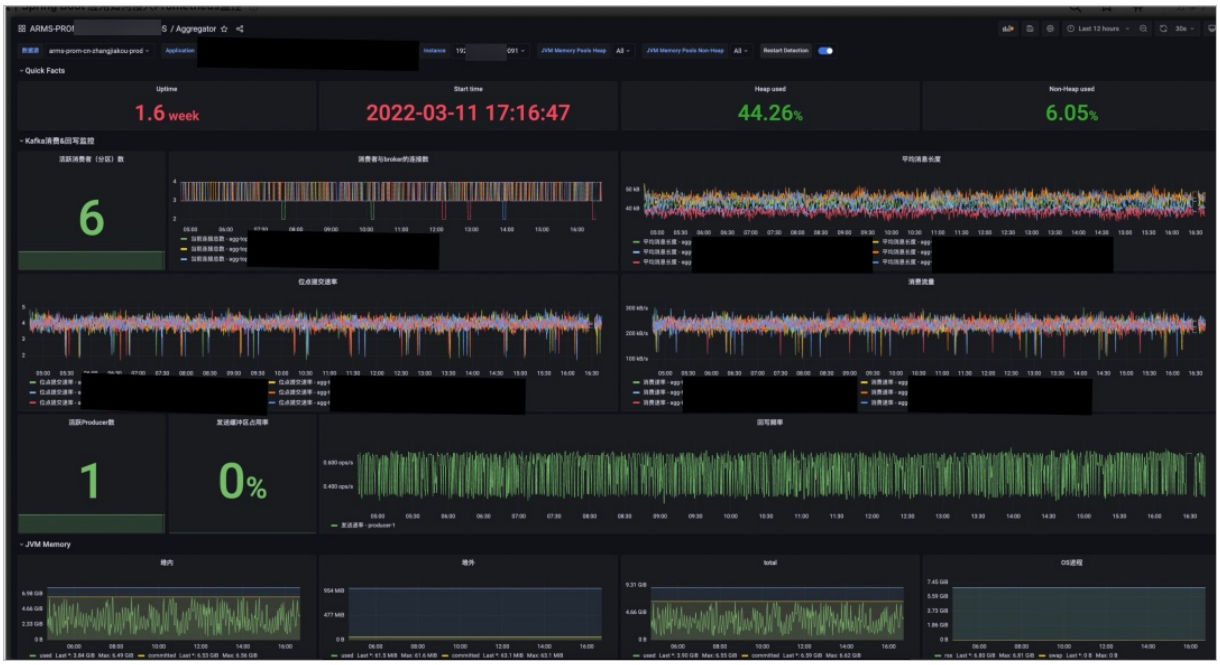


步骤三：看板配置

应用的监测数据已成功收集并存储到Prometheus监控，因此您可以配置相应的大盘及告警来查看监控到的数据。这里，为您提供以下两个Grafana社区中的开源大盘模板来构建您自己的业务监测模板。

- Spring Boot 2.1 Statistics
- JVM (Micrometer)

借助以上模板以及Prometheus监控内置的Grafana服务，您可以根据自己的需求，将日常开发和运维过程中需要重点关注的指标展示在同一个Grafana Dashboard页面上，创建属于您的个性化大盘，便于日常监测。例如，这里基于上述模板和自身业务构建了一个真实的大盘，包含总览、组件运行时间，内存使用率、堆内堆外内存、分代GC情况等。

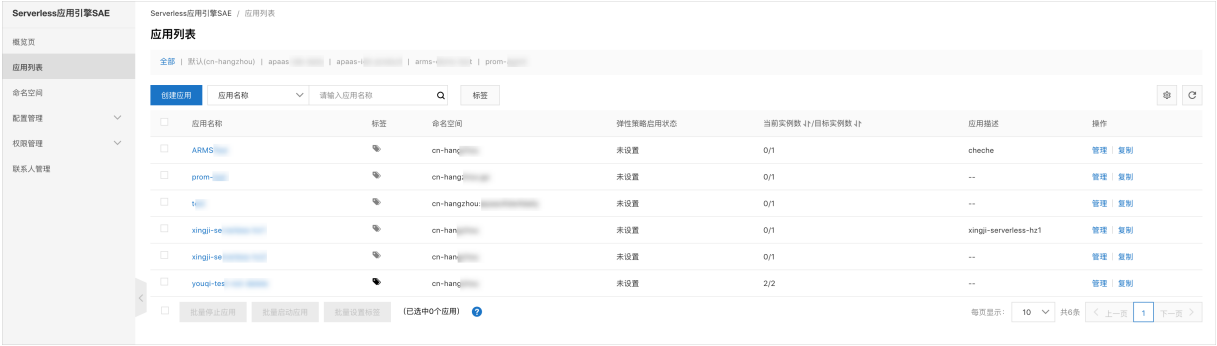


17.14. VPC网络下的SAE应用如何接入阿里云Prometheus监控

本文介绍如何将VPC网络下的SAE应用接入阿里云Prometheus监控。

步骤一：创建SAE应用并获取VPC、交换机等信息

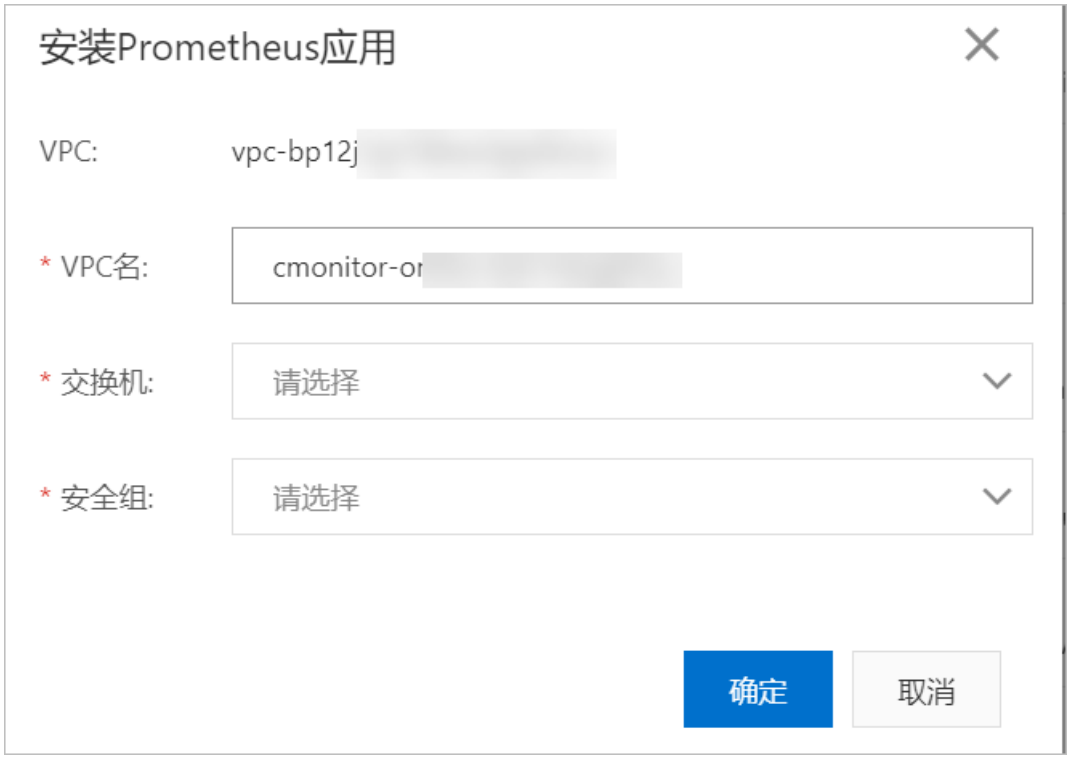
在SAE控制台中创建应用并暴露Metrics（可以为其他地址）服务。具体操作，请参见在SAE控制台使用WAR包部署Java Web应用。



在SAE控制台的应用列表页面已获取应用的命名空间、VPC、交换机、安全组等信息。

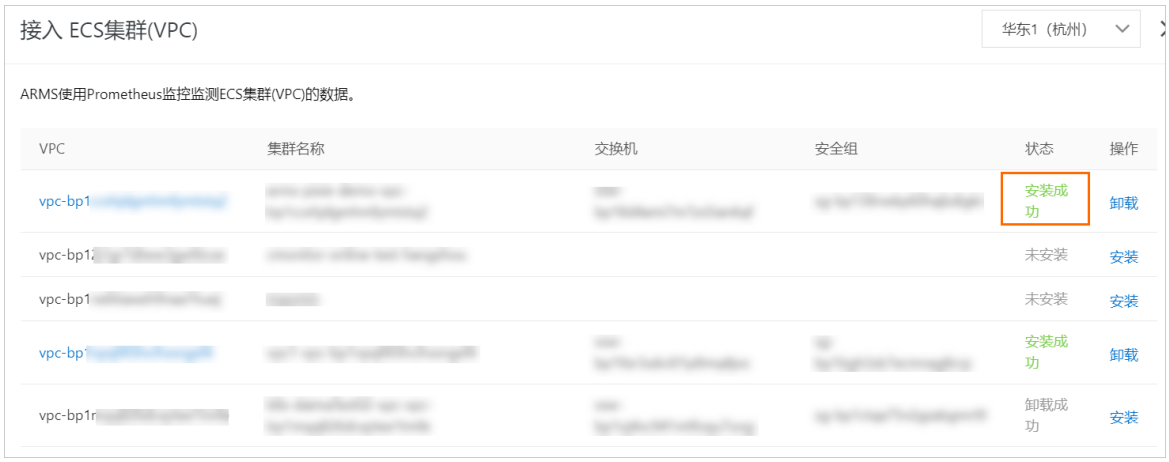
步骤二：安装Prometheus For VPC

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击**新建prometheus实例**。
3. 在**新建prometheus实例**页面，单击**Prometheus实例 for VPC**区域。
   在**接入 ECS集群(VPC)**面板显示当前地域下的所有VPC列表。
4. 单击步骤一中获取的目标VPC右侧**操作**列的**安装**。
5. 在**安装Prometheus应用**对话框中，输入VPC名称，选择交换机和安全组，然后单击**确定**。



> **说明**
>
> - VPC：为步骤一中获取的VPC。
> - 交换机和安全组：建议您选择与步骤一中获取的交换机和安全组保持一致。
> - 安全组规则：您还需要添加网络规则，确保SAE应用和Prometheus监控之间的网络通畅。

安装成功后，对应VPC右侧**状态**列显示安装成功。



步骤三：配置SAE服务发现

1. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击步骤二中安装的Prometheus实例 for VPC**操作**列的**设置**。
2. 在**设置**页面单击**编辑Prometheus.yaml**，在弹出的**编辑Prometheus.yaml**对话框中输入如下代码段，创建SAE服务发现。

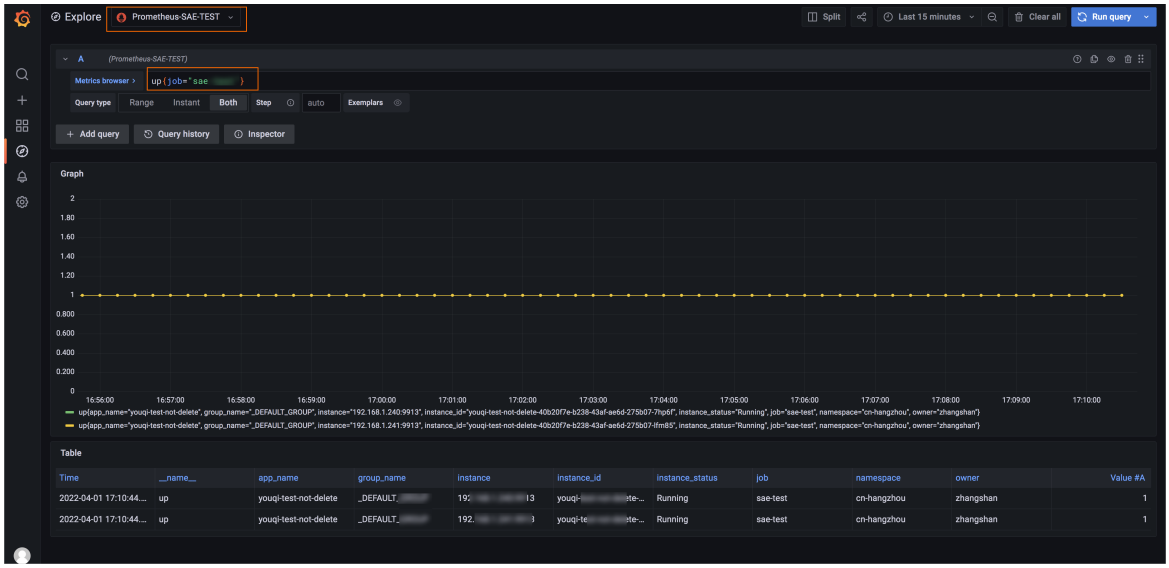
```
global:
  scrape_interval: 30s
  scrape_timeout: 30s
  evaluation_interval: 30s
scrape_configs:
## job名称 可以自定义修改
- job_name: sae-test
  honor_timestamps: true
  scrape_interval: 30s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  ## SAE专用服务发现类型
  aliyun_sae_sd_configs:
      ## 请将端口号调整为您的应用对应的端口号
    - port:  9913
      ## 请根据你的实际情况填写用户信息
      user_id: *********
      access_key: *********
      access_key_secret: ********
      refresh_interval: 30s
      ## 请根据步骤1中应用的信息填写
      region_id: cn-hangzhou
      vpc id: vpc-bp156863cbb2z17bhyzav
```

```
      namespace_id: ["cn-hangzhou"]
      app_name: ["test1","test2"]
      ##
      tag_filters:
        - key: environment
          values: ["daily","publish"]
        - key: owner
          values: ["zhangshan"]
  relabel_configs:
    - source_labels: [__meta_sae_private_ip]
      regex: (.*)
      target_label: __address__
      ##将端口号调整为您的应用对应的端口号
      replacement: $1:9913
    - source_labels: [__meta_sae_namespace_id]
      regex: (.*)
      target_label: namespace
      replacement: $1
    - source_labels: [__meta_sae_app_name]
      regex: (.*)
      target_label: app_name
      replacement: $1
    - source_labels: [__meta_sae_group_name]
      regex: (.*)
      target_label: group_name
      replacement: $1
    - source_labels: [__meta_sae_instance_id]
      regex: (.*)
      target_label: instance_id
      replacement: $1
    - source_labels: [__meta_sae_status]
      regex: (.*)
      target_label: instance_status
      replacement: $1
      ## 用户在应用上自定义的标签(格式:__meta_sae_tag_[tagName])
    - source_labels: [__meta_sae_tag_owner]
      regex: (.*)
      target_label: owner
      replacement: $1
```

步骤四：查看监控数据

1. 在左侧导航栏单击**大盘列表**，然后在**大盘列表**页面单击**创建大盘**，进入Grafana页面。
2. 在左侧导航栏单击**Explore**，然后在**Explore**页面选择VPC对应的数据源，然后输入Metrics查询命令，例如，up{job="sae-test"} 。

> **说明** 其中job名称为步骤三中创建的。



检查数据采集是否符合预期，若未成功采集到数据，请您检查安全组网络规则和Prometheus.yaml配置。

17.15. 如何实现集群内ServiceMonitor和PodMonitor的同步

本文介绍如何实现集群内ServiceMonitor和PodMonitor的同步。

前提条件

- 您的Prometheus Helm版本已升级至v1.1.5或以上。升级Helm版本的具体操作，请参见升级组件版本。
- （可选）如果您需要让通过集群命令创建的ServiceMonitor或PodMonitor能够被阿里云Prometheus监控发现到，那么您需要在创建的时候添加如下示例的**annotation**。

```
annotations:
  arms.prometheus.io/discovery: 'true' // 其中'true'被发现，false不会被发现
```

背景信息

阿里云Prometheus监控会默认发现集群内的ServiceMonitor以及PodMonitor并采集数据。如果您的集群通过命令行已创建了ServiceMonitor或PodMonitor，或者您的集群已安装了开源Prometheus自带的ServiceMonitor或PodMonitor，那么这些采集任务可能会和阿里云Prometheus监控默认的采集任务重复，导致不必要的资源消耗，同时会产生相应的费用。

为了避免产生不必要的成本以及资源的消耗，阿里云Prometheus监控提供了一键同步集群内ServiceMonitor和PodMonitor的开关。您可根据需求决定是否进行ServiceMonitor或PodMonitor同步。打开同步开关之后，您在集群创建的ServiceMonitor或PodMonitor将会被阿里云Prometheus监控发现；否则，阿里云Prometheus监控仅会发现通过Prometheus监控控制台创建的ServiceMonitor或PodMonitor。

操作步骤

同步集群内ServiceMonitor和PodMonitor的操作步骤类似，这里以同步集群内的ServiceMonitor为例为您介绍具体的操作。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择地域，然后单击目标Prometheus实例名称。
3. 在左侧导航栏，单击**服务发现**。

4. 在**服务发现**页面单击**配置**页签，然后单击**ServiceMonitor**页签，打开**自动同步集群内ServiceMonitor**开关即可。

> **说明**
>
> - 若您需要同步集群内的PodMonitor，此时只需在配置页面单击**PodMonitor**页签并打开**自动同步集群内PodMonitor**开关即可。
> - 您在编辑ServiceMonitor或PodMonitor时，请勿删除**annotations**字段信息，否则会影响到您数据的采集。

18.常见问题

18.1. 常见问题概述

本章节汇总了使用阿里云Prometheus监控的常见问题。

本页目录

- 接入ARMS Prometheus监控后，为什么会产生额外的费用？
- 如果不需要某些自定义指标，应该如何避免收费？
- 为什么在创建Grafana大盘时，没有Kubelet和API Server的监控指标？
- 为什么Exporter对应的大盘看不到具体的指标？
- 如何禁止默认报警自动创建
- 如何开启报警自动启用
- 如何查看采集到的数据？
- 为什么获取不到CSI组件的Metric？
- Grafana、Istio和HPA等第三方系统如何集成Prometheus监控？
- 为什么ACK集群已删除，Prometheus Agent没有同步删除？
- 如何关闭对云数据库MongoDB版的监控？
- 如何关闭对实时计算Flink版的监控？
- 为什么在容器、节点、Pod中得到的内存值不一致？

18.2. 计费相关

18.2.1. 接入ARMS Prometheus监控后，为什么会产生额外的费用？

这可能是由于ARMS Prometheus监控内置的服务发现功能采集到了您的数据指标，因而产生相关费用。您可以通过关闭相应的服务发现功能来避免产生费用。

ARMS Prometheus监控提供的服务发现功能包含默认服务发现和自定义ServiceMonitor两部分。默认服务发现在接入后自动开启并进行指标采集。ServiceMonitor需进行手动添加，并根据配置进行指标采集。

有关服务发现功能的更多详细信息以及如何关闭服务发现，请参考管理Kubernetes集群服务发现。

18.2.2. 如果不需要某些自定义指标，应该如何避免收费？

当您的Prometheus实例不再需要监控某些自定义指标的数据时，为了避免这些自定义指标继续产生费用，您可以废弃这些自定义指标。配置废弃指标的具体操作，请参见配置废弃指标。已经配置为废弃的自定义指标将不再继续产生费用。

18.3. 大盘相关

18.3.1. 为什么在创建Grafana大盘时，没有Kubelet和API Server的监控指标？

目前只采集了部分免费基础指标，采集指标列表如下：

采集指标列表 ❯

18.3.2. 为什么Exporter对应的大盘看不到具体的指标？

1. 参考如何查看采集到的数据？排查Exporter数据输出是否正常。
   如果存在异常，请按以下步骤排查。
2. 登录Prometheus控制台。
3. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
4. 在左侧导航栏，单击**组件监控**。
5. 在需要排查的Exporter右侧**操作**列，单击**日志**。
   查看是否有报错日志：
   - 有：查看对应开源社区Exporter是否有解决方案。开源社区Exporter链接如下：
     - MySQL
     - Mongo
     - RabbitMQ

       > **说明** 默认仅支持RabbitMQ 3.6以上版本。

     - Kafka
     - Redis
     - PostgreSQL
     - Zookeeper
   - 没有：请使用钉钉通过搜索钉钉账号 `arms160804` 联系技术支持。

18.4. 报警相关

18.4.1. 如何禁止默认报警自动创建

Prometheus监控会为接入的应用自动创建默认报警。关于这些自动创建的默认报警，请参见报警规则说明。

控制默认报警自动创建的参数为defaultAlert，取值为*true*时表示自动创建默认的报警，取值为*false*时表示不自动创建默认的报警。

对于已接入Prometheus监控的应用，如果您希望禁止默认报警自动创建，您需要将Prometheus监控插件的defaultAlert设置为*false*。在禁止默认报警自动创建后，建议您手动删除Prometheus监控之前为您自动创建的默认报警规则。

为容器服务K8s集群进行设置的操作步骤如下：

1. 将Prometheus监控插件的defaultAlert设置为*false*。

i. 登录容器服务管理控制台。
ii. 在控制台左侧导航栏中，单击**集群**。
iii. 在**集群列表**页面中，选择目标集群，并在目标集群右侧的**操作**列下，单击**应用管理**。
iv. 在**无状态**页面，选择**命名空间**为arms-prom，找到以arms-prom开头的deployment，例如arms-prom-ack-arms-prometheus，在其右侧**操作**列，单击**编辑**。
v. 在**编辑**页面的**生命周期**区域的**启动执行**的**参数**文本框中，添加*-defaultAlert=false*，然后在页面右上角，单击**更新**。
Prometheus监控插件完成更新后，等待3分钟~5分钟，Prometheus监控不再为该集群创建默认报警。

2. （可选）删除默认报警规则。
i. 登录Prometheus控制台。
ii. 在**Prometheus监控**页面，单击目标K8s集群的名称。
iii. 在左侧导航栏，单击**告警规则**。然后在对应告警规则**操作**列单击**删除**，删除默认报警规则。

18.4.2. 如何开启报警自动启用

报警的状态默认为关闭。

控制报警自动启用的参数为alert，取值为*true*时表示报警自动启用，取值为*false*时表示报警不自动启用。

对于已接入Prometheus监控的应用，如果您希望创建报警后，报警自动启用，您需要将Prometheus监控插件的alert设置为*true*。

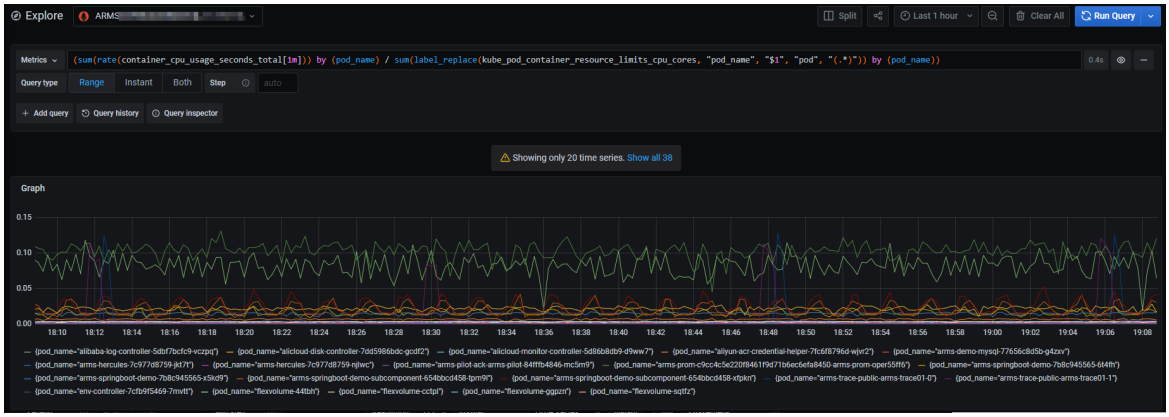
为容器服务K8s集群进行设置的操作步骤如下：

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，选择目标集群，并在目标集群右侧的**操作**列下，单击**应用管理**。
4. 在**无状态**页面，选择**命名空间**为arms-prom，找到以arms-prom开头的deployment，例如arms-prom-ack-arms-prometheus，在其右侧**操作**列，单击**编辑**。
5. 在**编辑**页面的**生命周期**区域的**启动执行**的**参数**文本框中，添加*-alert=true*，然后在页面右上角，单击**更新**。
Prometheus监控插件完成更新后，等待3分钟~5分钟，所有报警的**状态**显示启用。

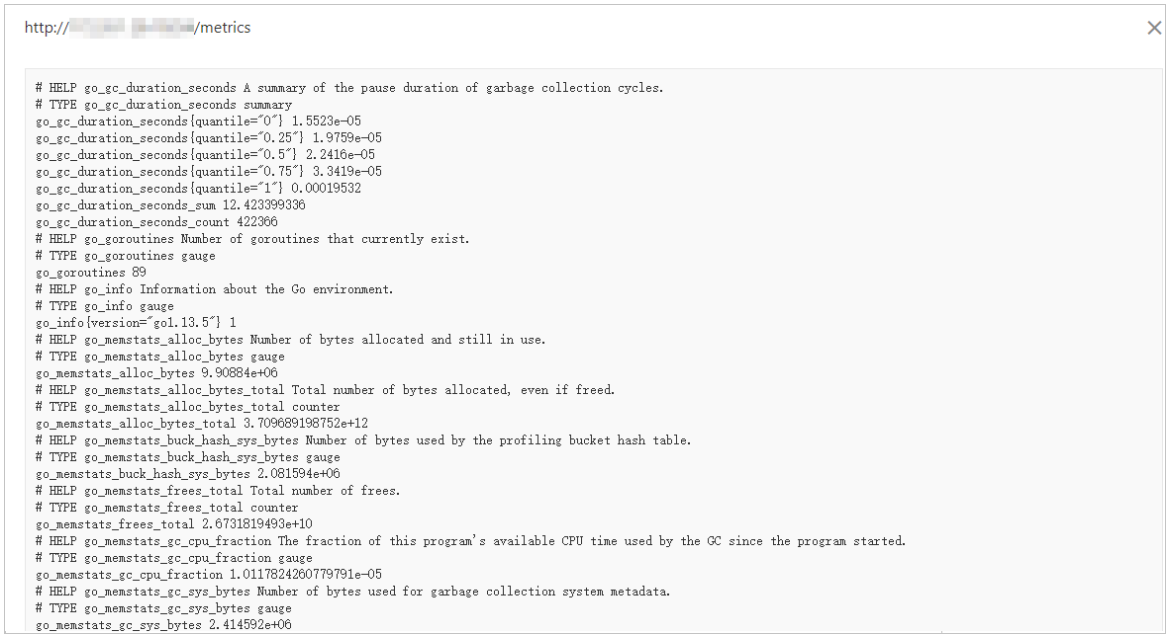
18.5. 其他

18.5.1. 如何查看采集到的数据？

1. 打开Grafana大盘概览页。
2. 在左侧导航栏单击Explore图标。
3. 在Explore页面顶部下拉框中选择集群，然后在Metrics输入框中输入PromQL语句，单击右上角的Run Query进行调试。



1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择K8s集群所在的地域，单击目标K8s集群的名称。
3. 在左侧导航栏，单击**服务发现**。
4. 在右侧页面，单击**Targets**页签。
5. 单击目标Target，并单击目标Endpoint链接，获取原始输出的Metric。



方法一：

方法二：

18.5.2. 为什么获取不到CSI组件的Metric？

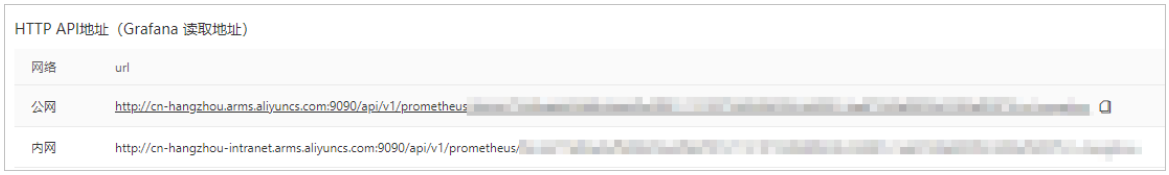
请检查ACK CSI组件版本是否为V1.18.8.45或以上，如果不是，请更新相关组件。更多信息，请参考使用csi-plugin组件监控节点侧存储资源。

18.5.3. Grafana、Istio和HPA等第三方系统如何集成Prometheus监控？

Grafana、Istio和HPA等第三方系统集成Prometheus监控时，需要获取Prometheus监控的API接口地址。可以按照以下操作步骤获取API接口地址：

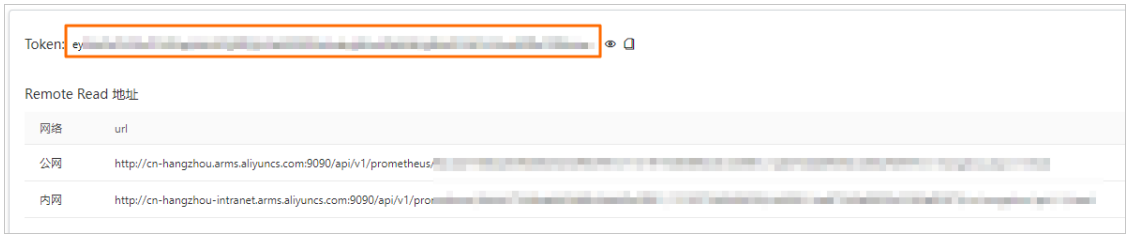
1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面左上角选择Prometheus实例所在的地域，并在目标集群右侧的**操作**列单击**设置**。
3. 在右侧页面单击**设置**页签。
4. 在**设置**页签上，根据需求复制公网或私网的HTTP API地址。

> **说明** 如果是云服务类型的Prometheus实例，请根据接入云服务的产品类型选择对应的HTTP API地址。



5. （可选）如果您需要提高Grafana数据读取的安全性，可以单击**生成token**，获取Prometheus实例的鉴权Token。

> **注意** 生成Token后，在Grafana中添加数据源时必须配置Token，否则可能无法读取Prometheus监控数据。



获取到API接口地址后，将其添加到Grafana、Istio和HPA等第三方系统中，即可集成Prometheus监控。完整的Grafana集成Prometheus监控的操作请参见将阿里云Prometheus监控数据接入本地Grafana。

18.5.4. 为什么ACK集群已删除，Prometheus Agent没有同步删除？

背景

在同一地域下，删除ACK集群后出现以下情况：

- 在Prometheus控制台集群列表页面出现若干置灰集群。
- 重新创建同名集群失败。

解决方案

在Prometheus控制台中卸载Prometheus监控插件，具体操作，请参见卸载Prometheus监控插件。

18.5.5. 如何关闭对云数据库MongoDB版的监控？

若您不再需要对阿里云云数据库MongoDB版进行监控，您可以按照以下步骤卸载云数据库MongoDB版

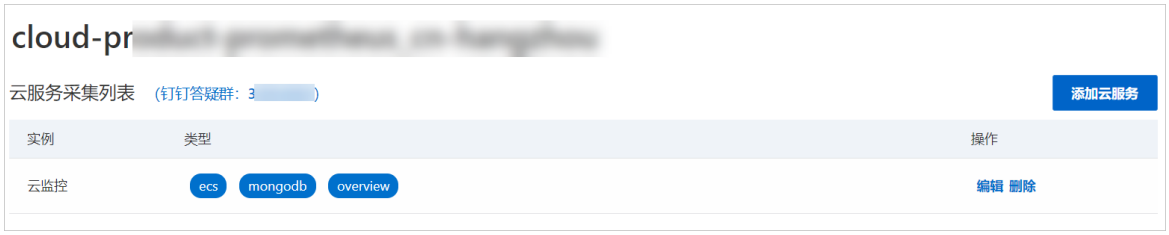
若您不再需要对阿里云云数据库MongoDB版进行监控，您可以按照以下步骤卸载云数据库MongoDB版。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。

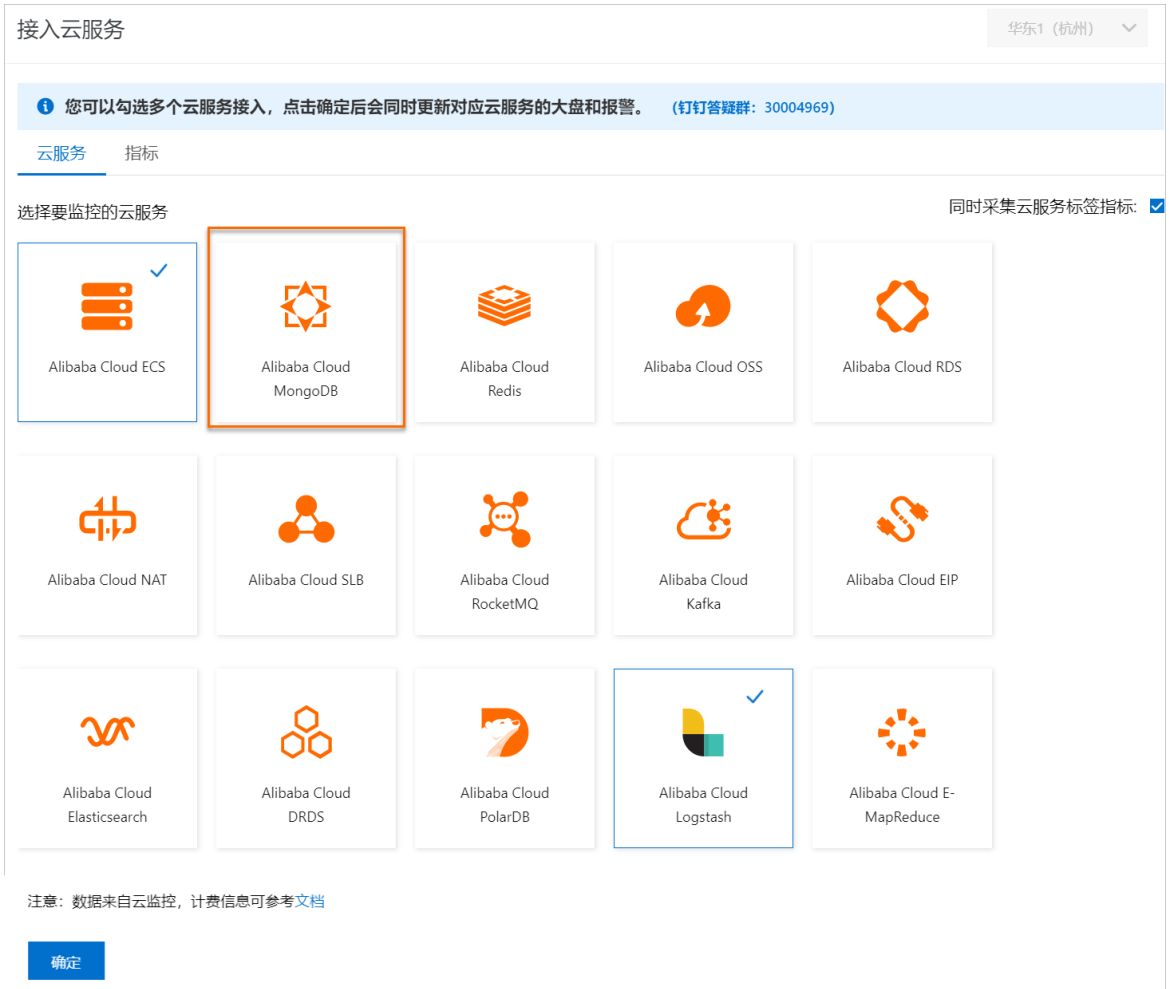
   Prometheus监控页面显示了所有Prometheus实例，其中实例类型为Prometheus for 云服务对应的实例为Prometheus云服务实例。

3. 单击Prometheus云服务实例名称。
4. 在左侧导航栏单击**云服务接入**。

   云服务采集列表页面显示当前已经接入的云服务。



5. 单击**云监控**实例**操作**列的**编辑**，或者单击右上角的**添加云服务**。
   在展开的**云服务**页签，取消勾选**Alibaba Cloud MongoDB**，然后单击**确定**，可关闭对云数据库MongoDB版的监控。



18.5.6. 如何关闭对实时计算Flink版的监控？

若您不再需要对实时计算Flink版进行监控，您可以按照以下步骤卸载实时计算Flink版。

1. 登录Prometheus控制台。
2. 在**Prometheus监控**页面的顶部菜单栏，选择Prometheus实例所在的地域。
   Prometheus监控页面显示了所有Prometheus实例。
3. 单击实例类型为**Prometheus for Flink Serverless**的实例对应**操作**列的**卸载**，然后在弹出的对话框中单击**确认**。

18.5.7. 为什么在容器、节点、Pod中得到的内存值不一致？

Pod命令如何计算内存使用量

执行 `kubectl top pod` 命令得到的结果，并不是容器服务中container_memory_usage_bytes指标的内存使用量，而是指标container_memory_working_set_bytes的内存使用量，计算方式如下：

- container_memory_usage_bytes = container_memory_rss + container_memory_cache + kernel memory
- container_memory_working_set_bytes = container_memory_usage_bytes - total_inactive_file（未激活的匿名缓存页）
- container_memory_working_set_bytes是容器真实使用的内存量，也是资源限制limit时的重启判断依据

kubectl top node命令和top node命令的区别

在容器中执行 `kubectl top node` 命令，得到的CPU和内存值，并不是节点上所有Pod的总和，因此不可以直接相加汇总结果值。

在容器的节点上执行 `top node` 命令，得到的CPU和内存值，是节点所在的物理机上cgroup根目录下的汇总统计值。

以上两个命令操作得到的结果值不可以进行直接对比，因为两者的计算逻辑不同。

例如，主机监控就是在容器节点上执行 `top node` 的结果：node_memory_MemAvailable_bytes。主机监控的相关指标监控项说明，请参见监控项说明。计算方式如下：

- 当/proc/meminfo中存在 `MemAvailable` 时的计算方式为：（total-MemAvailable）/ total×100%
- 当/proc/meminfo中无 `MemAvailable` 时的计算方式为：（total-free-buffers-cached）/ total×100%

kubectl top pod命令和top pod命令的区别

在容器中执行 `kubectl top pod` 命令，得到的CPU和内存值，并不是节点上所有Pod的总和，因此不可以直接相加汇总结果值。

在容器的Pod上执行 `top pod` 命令，得到的CPU和内存值，是Pod所在的物理机上cgroup根目录下的汇总统计值，而非Pod可分配的CPU和内存值，即使对容器的Pod做了资源限制limit。

以上两个命令操作得到的结果值不可以进行直接对比，因为两者的计算逻辑不同。

进程RSS表示进程使用的所有物理内存值（file_rss + anon_rss），即Anonymous Pages和Mapped Pages的内存值总和（包含共享内存）；进程cgroup RSS表示Anonymous和Swap Cache Memory的内存值总和（不包含共享内存）。两者都不包含File Cache。

19.相关协议

19.1. Prometheus服务等级协议

本产品最新版服务等级协议，请在阿里云服务等级协议汇总页查找获取。

19.2. Prometheus监控服务试用条款

版本生效日期：2020年09月15日。
欢迎您申请试用阿里云服务

提示条款

欢迎您与阿里云计算有限公司（以下简称“阿里云”）共同签署本服务试用条款（下称“本条款”）并试用阿里云服务！

条款前所列索引关键词仅为帮助您理解该条款表达的主旨之用，不影响或限制本条款的含义或解释。为维护您自身权益，建议您仔细阅读各条款具体表述。

【审慎阅读】您在同意本条款之前，应当认真阅读本条款。请您务必审慎阅读、充分理解各条款的内容，特别是免除或者限制责任的条款、法律适用和争议解决条款，这些条款将以粗体下划线标识，您应重点阅读。如您对条款有任何疑问，可以向客服和相关业务部门进行咨询。

【签约动作】当您阅读并点击同意本条款或以其他方式选择接受本条款后，即表示您已充分阅读、理解并接受本条款的全部内容，并与阿里云达成一致。本条款自您通过网络页面点击确认或以其他方式选择接受本条款之日起成立。阅读本条款的过程中，如果您不同意本条款或其中任何条款约定，请勿进行签约动作。

通用服务条款

1. 签约主体及条款范围
   本服务条款是您与阿里云计算有限公司就您试用阿里云Prometheus监控服务所签署的服务条款。
2. 服务内容
   本条款中“服务”指：阿里云网站和客户端（以下单独或统称“阿里云网站”）向您所展示的，您申请试用、且经阿里云同意向您提供的技术及网络支持服务。
3. 服务费用
   i. 您理解并同意，阿里云目前为您免费提供服务，即您开通或使用服务，并不需向阿里云支付费用；阿里云不排除日后收取费用的可能，届时阿里云将提前10个自然日通过在网站内合适版面发布公告或发送站内通知等方式公布收费政策及规范。
   ii. 阿里云进行收费后，如您仍使用阿里云服务的，应按届时有效的收费政策付费并应遵守届时阿里云公布的有效的服务条款。如在收费后，您拒绝支付服务费的，阿里云有权不再向您提供服务，并有权利不再继续保留您的业务数据。
4. 您的权利和义务
   i. 成功开通服务后，您有权要求阿里云按照本条款以及阿里云网站相关页面所展示的服务说明、技术规范等内容向您提供服务。
   ii. 就阿里云服务的使用应符合阿里云的《服务使用规则》以及本条款。
   iii. 您对自己存放在阿里云云平台上的数据以及进入和管理阿里云云平台上各类产品与服务的口令、密码的完整性和保密性负责。因您维护不当或保密不当致使上述数据、口令、密码等丢失或泄漏所引起的损失和后果均由您承担。

iv. 您须依照《网络安全法》、《互联网信息服务管理办法》等法律法规的规定保留自己网站的访问日志记录，包括发布的信息内容及其发布时间、互联网地址（IP）、域名等，国家有关机关依法查询时应配合提供。您将承担未按规定保留相关记录而引起的相应法律责任。

v. 为了数据的安全，您应负责您数据的备份工作。阿里云的产品或服务可能会为您配置数据备份的功能或工具，您负责操作以完成备份。

vi. 您应对您的用户业务数据的来源及内容负责，阿里云提示您谨慎判断数据来源及内容的合法性。您将承担因您的用户业务数据内容违反法律法规、部门规章或国家政策而造成的相应结果及责任。

vii. 您理解并同意，中华人民共和国的国家秘密受法律保护，您有保守中华人民共和国的国家秘密的义务；您使用阿里云服务应遵守相关保密法律法规的要求，并不得危害中华人民共和国国家秘密的安全。

viii. 您还应仔细阅读并遵守阿里云在网站页面上展示的相应服务说明、技术规范、使用流程、操作文档等内容（以上简称“操作指引”），依照相关操作指引进行操作。您将承担违反相关操作指引所引起的后果；同时，阿里云郑重提示您，请把握风险谨慎操作。

5. 阿里云的权利、义务

i. 阿里云应按照约定提供服务。

ii. 服务期限内，阿里云将为您提供如下售后服务：

a. 阿里云将提供7×24电话咨询服务以及在线工单咨询服务，解答您在使用中的问题。

b. 阿里云将为您提供故障支持服务，您应通过在线工单申报故障。阿里云将及时就您非人为操作所出现的故障提供支持，但因您的人为原因和/或不可抗力、以及其他非阿里云控制范围内的事项除外。

iii. 阿里云仅负责操作系统以下的底层部分及阿里云提供的软件的运营维护，即服务的相关技术架构及阿里云提供的操作系统等。操作系统之上部分（如您在系统上安装的应用程序）由您负责。此外，您自行升级操作系统可能会造成宕机等不良影响，请把握风险并谨慎操作。

iv. 您了解阿里云无法保证其所提供的服务毫无瑕疵（如阿里云安全产品并不能保证您的硬件或软件的绝对安全），但阿里云承诺不断提升服务质量及服务水平。所以您同意：即使阿里云提供的服务存在瑕疵，但上述瑕疵是当时行业技术水平所无法避免的，其将不被视为阿里云违约。您同意和阿里云一同合作解决上述瑕疵问题。

v. 阿里云的某些服务可能具备账户授权管理功能，即您可将您对服务的全部或部分操作权限授权给您指定的一个或多个被授权账户，此种情况下，任一被授权账户下进行的所有操作行为，均将被视为您通过本人账户所进行的行为，都将由您承担相应的责任和由此产生的服务费用。

vi. 您理解并认可，阿里云将为您提供基于某些服务的安全防护（如“云盾安骑士服务”）以及管理与监控的相关功能及服务（如“云监控”），尽管阿里云对该等服务经过详细的测试，但并不能保证其与所有的软硬件系统完全兼容，亦不能保证其软件及服务的完全准确性。如果出现不兼容及软件错误的情况，您应立即关闭或停止使用相关功能，并及时联系阿里云，获得技术支持。

6. 用户业务数据

i. 阿里云理解并认可，您通过阿里云提供的服务，加工、存储、上传、下载、分发以及通过其他方式处理的数据，均为您的用户业务数据，您完全拥有您的用户业务数据。

ii. 就用户业务数据，阿里云除执行您的服务要求外，不进行任何未获授权的使用及披露。但以下情形除外：

a. 在国家有关机关依法查询或调阅用户业务数据时，阿里云具有按照相关法律法规或政策文件要求提供配合，并向第三方或者行政、司法等机构披露的义务。

b. 您和阿里云另行协商一致。

iii. 您可自行对您的用户业务数据进行删除、更改等操作。如您自行释放服务或删除数据的，阿里云将删除您的数据，按照您的指令不再保留该等数据。就数据的删除、更改等操作，您应谨慎操作。

iv. 当服务期届满、服务提前终止（包括双方协商一致提前终止，其他原因导致的提前终止等）或您发生欠费时，除法律法规明确规定、主管部门要求或双方另有约定外，阿里云仅在一定的缓冲期（以您所订购的服务适用的专有条款、产品文档、服务说明等所载明的时限为准）内继续存储您的用户业务数据（如有），缓冲期届满阿里云将删除所有用户业务数据，包括所有缓存或者备份的副本，不再保留您的任何用户业务数据。

v. 用户业务数据一经删除，即不可恢复。您应自行承担数据因此被删除所引发的后果和责任，您理解并同意，阿里云没有继续保留、导出或者返还用户业务数据的义务。

vi. 根据您与阿里云协商一致，阿里云在您选定的数据中心存储用户业务数据。阿里云恪守对用户的安全承诺，根据适用的法律保护用户存储在阿里云数据中心的数据。

7. 知识产权

i. 在本条款项下一方向对方提供的任何资料、技术或技术支持、软件、服务等的知识产权均属于提供一方或其合法权利人所有。 除提供方或合法权利人明示同意外，另一方无权复制、传播、转让、许可或提供他人使用上述知识成果，否则应承担相应的责任。

ii. 您应保证提交阿里云的素材、对阿里云服务的使用及使用阿里云服务所产生的成果未侵犯任何第三方的合法权益。阿里云应保证向您提供的服务未侵犯任何第三方的合法权益。

iii. 如果第三方机构或个人对您使用阿里云服务所涉及的相关素材的知识产权归属提出质疑或投诉，或对您使用的阿里云服务的知识产权的归属提出质疑或投诉，您和阿里云均有责任出具相关知识产权证明材料，并配合对方的相关投诉处理工作。对于因此引起的索赔、诉讼或可能向其提起诉讼，违约方应负责解决，承担费用和损失，以及使另一方免责。

8. 保密条款

i. 本服务条款所称保密信息，是指一方（以下简称“接受方”）从对方（以下简称“披露方”）取得的、获知的、或因双方履行本条款而产生的商业秘密（包括财务秘密）、技术秘密、经营诀窍和（或）其他应予保密的信息和资料（包括产品资料，产品计划，价格，财务及营销规划，业务战略，客户信息，客户数据，研发，软件，硬件，API应用数据接口，技术说明，设计，特殊公式，特殊算法等），无论上述信息和资料以何种形式或载于何种载体，无论披露方在披露时是否以口头、图像或书面等方式表明其具有保密性。

ii. 双方应采取适当措施妥善保存对方提供的保密信息，措施的审慎程度不少于其保护自身的保密信息时的审慎程度。双方仅能将保密信息用于与本条款项下的有关用途或目的。

iii. 双方保证保密信息仅可在各自一方从事该业务的负责人和雇员范围内知悉，并严格限制接触上述保密信息的员工遵守本条之保密义务。

iv. 本条上述限制条款不适用于以下情况：

a. 在签署本条款之时或之前，该保密信息已以合法方式属接受方所有。

b. 保密信息在通知给接受方时，已经公开或能从公开领域获得。

c. 保密信息是接受方从与其没有保密或不透露义务的第三方获得的。

d. 在不违反本条款约定责任的前提下，该保密信息已经公开或能从公开领域获得。

e. 该保密信息是接受方或其关联或附属公司独立开发，而且未从通知方或其关联或附属公司获得的信息中获益。

f. 接受方应法院或其它法律、行政管理部门要求（通过口头提问、询问、要求资料或文件、传唤、民事或刑事调查或其他程序）因而透露保密信息。

g. 接受方为向行政管理部门、行业协会等机构申请某项业务资质、获得某项认定、或符合国家、行业标准/认证，需结合对方情况向前述机构提交材料或进行说明的而披露的信息，在该等情况下，接受方应秉持必要情况下最少披露原则及要求因此获知保密信息的机构按不低于本条款的标准予以保密。

v. 您和阿里云都应尽最大的努力保护上述保密信息不被泄露。一旦发现有上述保密信息泄露事件，双方应合作采取一切合理措施避免或者减轻损害后果的产生。如因此给对方造成损失的，应赔偿因此给对方造成的直接经济损失。

9. 服务的终止与变更

i. 发生下列情形之一的，服务期限提前终止：

a. 双方协商一致提前终止的。

b. 经阿里云提前通知，结束公测期或免费试用期的。

c. 您严重违反本条款（包括，您严重违反相关法律法规规定，或您严重违反本条款项下之任一承诺内容等），阿里云有权提前终止服务直至清除您的全部数据。

d. 您理解并充分认可，虽然阿里云已经建立（并将根据技术的发展不断完善）必要的技术措施来防御包括计算机病毒、网络入侵和攻击破坏（包括DDoS）等危害网络安全事项或行为（以下统称该等行为），但鉴于网络安全技术的局限性、相对性以及该等行为的不可预见性，因此如因您网站遭遇该等行为而给阿里云或者阿里云的其他网络或服务器（包括本地及外地和国际的网络、服务器等）带来危害，或影响阿里云与国际互联网或者阿里云与特定网络、服务器及阿里云内部的通畅联系，阿里云可决定暂停或终止服务。

ii. 您理解并认可，为技术升级、服务体系升级、或因经营策略调整或配合国家重大技术、法规政策等变化，阿里云不保证永久的提供某种服务，并有权变更所提供服务的形式、规格或其他方面（如服务的价格和计费模式），在终止该种服务或进行此种变更前，阿里云将提前以网站公告、站内信、邮件或短信等一种或多种方式进行事先通知。

10. 违约责任

i. 您违反本条款中的承诺、保证条款、服务使用规则或义务的任一内容，或阿里云根据其判断认为您的使用行为存在异常的，阿里云均有权就其情节，根据独立判断并单方采取以下措施中的一种或多种：

a. 限制、中止使用服务。

b. 终止提供服务，终止本条款。

c. 追究您的法律责任。

d. 其他阿里云认为适合的处理措施。

阿里云依据前述约定采取中止服务、终止服务等措施而造成的用户损失将由您承担。

ii. 如因您违反有关法律法规或者本条款、相关规则之规定，使阿里云遭受任何损失、受到其他用户、任何第三方的索赔或任何行政管理部门的处罚，您应对阿里云、其他用户或相关第三方的实际损失进行全额赔偿，包括合理的律师费用。

iii. 您理解且同意，鉴于计算机、互联网的特殊性，下述情况不属于阿里云违约：

a. 阿里云在进行系统及服务器配置、维护、升级时，需要短时间中断服务。

b. 由于Internet上的通路阻塞造成您网站访问速度下降。

iv. 责任的限制及免除。
您应理解并同意，虽然阿里云服务会提供服务可用性和可靠性支撑，但在试用期间（免费或不免费），阿里云将不对任何服务可用性、可靠性做出承诺。阿里云亦不对您使用试用服务工作或结果承担任何责任。

v. 在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性的损害，包括您使用阿里云服务而遭受的利润损失承担责任（即使您已被告知该等损失的可能性）。

vi. 在法律允许的情况下，阿里云在本条款项下所承担的损失赔偿责任不超过就该服务过往12个月所缴纳的服务费用的总和。

11. 通知

i. 您在使用阿里云服务时，您应该向阿里云提供真实有效的联系方式（包括您的电子邮件地址、联系电话、联系地址等），对于联系方式发生变更的，您有义务及时更新有关信息，并保持可被联系的状态。您接收站内信、系统消息的会员账号（包括子账号），也作为您的有效联系方式。

ii. 阿里云将向您的上述联系方式的其中之一或其中若干向您送达各类通知，而此类通知的内容可能对您的权利义务产生重大的有利或不利影响，请您务必及时关注。

iii. 阿里云通过上述联系方式向您发出通知，其中以电子的方式发出的书面通知，包括但不限于公告，向您提供的联系电话发送手机短信，向您提供的电子邮件地址发送电子邮件，向您的账号发送系统消息以及站内信信息，在发送成功后即视为送达；以纸质载体发出的书面通知，按照提供联系地址交邮后的第五个自然日即视为送达。

iv. 您应当保证所提供的联系方式是准确、有效的，并进行实时更新。如果因提供的联系方式不确切，或不及时告知变更后的联系方式，使法律文书无法送达或未及时送达，由您自行承担由此可能产生的法律后果。

12. 不可抗力

i. 因不可抗力或者其他意外事件，使得本服务条款的履行不可能、不必要或者无意义的，遭受不可抗力、意外事件的一方不承担责任。

ii. 不可抗力、意外事件是指不能预见、不能克服并不能避免且对一方或双方当事人造成重大影响的客观事件，包括但不限于自然灾害如洪水、地震、瘟疫流行等以及社会事件如战争、动乱、政府行为、电信主干线路中断、黑客、网路堵塞、电信部门技术调整和政府管制等。

13. 法律适用及争议解决

i. 本条款之订立、生效、解释、修订、补充、终止、执行与争议解决均适用中华人民共和国大陆法律；如法律无相关规定的，参照商业惯例及/或行业惯例。

ii. 您因使用阿里云服务所产生及与阿里云服务有关的争议，由阿里云与您协商解决。协商不成时，任何一方均可向被告所在地有管辖权的人民法院提起诉讼。

14. 附则

i. 本条款的附件，以及阿里云在阿里云网站相关页面上的服务说明、价格说明和您确认同意的订购页面（包括特定产品的专用条款、服务说明、操作文档等）均为本条款不可分割的一部分。如遇不一致之处，以（1）服务说明、价格说明、其他订购页面，（2）专用条款和4.3. 附件，（3）本条款通用条款的顺序予以适用。

ii. 如本条款内容发生变动，阿里云应通过提前30天在阿里云网站的适当版面公告向您提示修改内容；如您继续使用阿里云服务，则视为您接受阿里云所做的相关修改。

iii. 阿里云有权经提前将本条款的权利义务全部或者部分转移给阿里云的关联公司。

iv. 阿里云于您过失或违约时放弃本条款规定的权利，不应视为其对您的其他或以后同类之过失或违约行为弃权。

v. 本条款任一条款被视为废止、无效或不可执行，该条应视为可分的且并不影响本条款其余条款的有效性及可执行性。

vi. 本条款项下之保证条款、保密条款、知识产权条款、法律适用及争议解决条款等内容，不因本条款的终止而失效。

19.3. Prometheus监控服务专家版协议

版本生效日期：2018年2月26日。

欢迎您与阿里云计算有限公司（以下简称“阿里云”）共同签署本《Prometheus监控服务专家版协议》（下称“本协议”）并使用阿里云服务！

协议中条款前所列索引关键词仅为帮助您理解该条款表达的主旨之用，不影响或限制本协议条款的含义或解释。为维护您自身权益，建议您仔细阅读各条款具体表述。

【审慎阅读】您在同意本协议之前，应当认真阅读本协议。请您务必审慎阅读、充分理解各条款的内容，特别是免除或者限制责任的条款、法律适用和争议解决条款，这些条款将以粗体下划线标识，您应重点阅读。如您对协议有任何疑问，可以向客服和相关业务部门进行咨询。

【签约动作】当您阅读并点击同意本协议或以其他方式选择接受本协议后，即表示您已充分阅读、理解并接受本协议的全部内容，并与阿里云达成一致。本协议自您通过网络页面点击确认或以其他方式选择接受本协议之日起成立。阅读本协议的过程中，如果您不同意本协议或其中任何条款约定，请勿进行签约动作。

通用服务条款

1. 签约主体及协议范围
   本服务协议是您与阿里云计算有限公司就您使用阿里云Prometheus监控服务所签署的服务协议。
2. 服务内容
   本条款中“服务”指：阿里云网站和客户端（以下单独或统称“阿里云网站”）所展示的Prometheus监控服务以及相关的技术及网络支持服务。
3. 服务费用
   i. 服务费用将在您订购页面予以列明公示，您可自行选择具体服务类型并按列明的价格予以支付。您可选择先付费或后付费的服务。
   ii. 先付费：
      a. 在您付费之后，阿里云才开始为您提供服务。您未在下单后立即付费的，订单将为您保留7天，7天内您仍未付费或者7天内阿里云服务售罄的，订单失效，订单失效后阿里云与您就服务所达成的合意失效。
      b. 服务期满双方愿意继续合作的，您至少应在服务期满前7天内支付续费款项，以使服务得以继续进行。
   iii. 后付费：您先使用后付费。具体扣费规则请查看阿里云网站上的页面展示且以页面公布的后付费服务当时有效的计费模式、标准为准。
   iv. 阿里云保留在您未按照约定支付全部费用之前不向您提供服务和/或技术支持，或者终止服务和/或技术支持的权利。同时，阿里云保留对您的欠费要求您按日承担万分之五的违约金以及追究其他法律责任的权利。
   v. 您完全理解阿里云价格体系中所有的赠送服务项目或优惠活动均为阿里云在正常服务价格之外的一次性特别优惠，赠送的服务项目或优惠活动不可折价、冲抵服务价格。
4. 您的权利和义务
   i. 成功订购服务后，您有权要求阿里云按照本服务协议以及阿里云网站相关页面所展示的服务说明、技术规范等内容向您提供服务。
   ii. 您订购阿里云的服务后，您可享受免费的售后服务。除此之外阿里云并提供其他付费的技术服务。
   iii. 您应按照阿里云的页面提示及本服务协议的约定支付相应服务费用。
   iv. 就阿里云服务的使用应符合附件阿里云的《服务使用规则》以及本服务协议。
   v. 您对自己存放在阿里云云平台上的数据以及进入和管理阿里云云平台上各类产品与服务的口令、密码的完整性和保密性负责。因您维护不当或保密不当致使上述数据、口令、密码等丢失或泄漏所引起的损失和后果均由您自行承担。
   vi. 您须依照《互联网信息服务管理办法》、《互联网电子公告服务管理规定》等法律法规的规定保留自己网站的访问日志记录，包括发布的信息内容及其发布时间、互联网地址（IP）、域名等，国家有关机关依法查询时应配合提供。您自行承担未按规定保留相关记录而引起的法律责任。
   vii. 为了数据的安全，您应负责您数据的备份工作。阿里云的产品或服务可能会为您配置数据备份的功能或工具，您负责操作以完成备份。

viii. 您应对您的用户业务数据的来源及内容负责，阿里云提示您谨慎判断数据来源及内容的合法性。因您的用户业务数据内容违反法律法规、部门规章或国家政策而造成的相应结果均由您承担。

ix. 您理解并同意，中华人民共和国的国家秘密受法律保护，您有保守中华人民共和国的国家秘密的义务。您使用阿里云服务应遵守相关保密法律法规的要求，并不得危害中华人民共和国国家秘密的安全。

x. 您还应仔细阅读并遵守阿里云在网站页面上展示的相应服务说明、技术规范、使用流程、操作文档等内容（以上简称“操作指引”），依照相关操作指引进行操作。您将自行承担违反相关操作指引所引起的后果。同时，阿里云郑重提示您，请把握风险谨慎操作。

5. 阿里云的权利、义务

i. 阿里云应按照约定提供服务。

ii. 服务期限内，阿里云将为您提供如下售后服务：

a. 阿里云将提供7×24电话咨询服务以及在线工单咨询服务，解答您在使用中的问题。

b. 阿里云将为您提供故障支持服务，您应通过在线工单申报故障。阿里云将及时就您非人为操作所出现的故障提供支持，但因您的人为原因和/或不可抗力、以及其他非阿里云控制范围内的事项除外。
您还可通过阿里云获得其他付费的售后服务，具体详见阿里云的网站相关页面的收费售后服务内容。

iii. 阿里云仅负责操作系统以下的底层部分及阿里云提供的软件的运营维护，即服务的相关技术架构及阿里云提供的操作系统等。操作系统之上部分（如您在系统上安装的应用程序）由您自行负责。此外，您自行升级操作系统可能会造成宕机等不良影响，请自行把握风险并谨慎操作。

iv. 您了解阿里云无法保证其所提供的服务毫无瑕疵（如阿里云安全产品并不能保证您的硬件或软件的绝对安全），但阿里云承诺不断提升服务质量及服务水平。所以您同意：即使阿里云提供的服务存在瑕疵，但上述瑕疵是当时行业技术水平所无法避免的，其将不被视为阿里云违约。您同意和阿里云一同合作解决上述瑕疵问题。

v. 阿里云的某些服务可能具备账户授权管理功能，即您可将您对服务的全部或部分操作权限授权给您指定的一个或多个被授权账户，此种情况下，任一被授权账户下进行的所有操作行为，均将被视为您通过本人账户所进行的行为，都将由您承担相应结果，并承担相应的服务费用。

vi. 您理解并认可，阿里云将为您提供基于某些服务的安全防护（如“云盾安骑士服务”）以及管理与监控的相关功能及服务（如“云监控”），尽管阿里云对该等服务经过详细的测试，但并不能保证其与所有的软硬件系统完全兼容，亦不能保证其软件及服务的完全准确性。如果出现不兼容及软件错误的情况，您应立即关闭或停止使用相关功能，并及时联系阿里云，获得技术支持。

6. 用户业务数据

i. 阿里云理解并认可，您通过阿里云提供的服务，加工、存储、上传、下载、分发以及通过其他方式处理的数据，均为您的用户业务数据，您完全拥有您的用户业务数据。

ii. 就用户业务数据，阿里云除执行您的服务要求外，不进行任何未获授权的使用及披露。但以下情形除外：

a. 在国家有关机关依法查询或调阅用户业务数据时，阿里云具有按照相关法律法规或政策文件要求提供配合，并向第三方或者行政、司法等机构披露的义务。

b. 您和阿里云另行协商一致。

iii. 您可自行对您的用户业务数据进行删除、更改等操作。如您自行释放服务或删除数据的，阿里云将删除您的数据，按照您的指令不再保留该等数据。就数据的删除、更改等操作，您应谨慎操作。

iv. 当服务期届满、服务提前终止（包括双方协商一致提前终止，其他原因导致的提前终止等）或您发生欠费时，除法律法规明确约定、主管部门要求或双方另有约定外，阿里云仅在一定的缓冲期（以您所订购的服务适用的产品文档、服务说明等所载明的时限为准）内继续存储您的用户业务数据（如有），缓冲期届满阿里云将删除所有用户业务数据，包括所有缓存或者备份的副本，不再保留您的任何用户业务数据。

v. 用户业务数据一经删除，即不可恢复。您应自行承担数据因此被删除所引发的后果和责任，您理解并同意，阿里云没有继续保留、导出或者返还用户业务数据的义务。

vi. 根据您与阿里云协商一致，阿里云在您选定的数据中心存储用户业务数据。阿里云恪守对用户的安全承诺，根据适用的法律保护用户存储在阿里云数据中心的数据。

7. 知识产权

i. 在本协议项下一方向对方提供的任何资料、技术或技术支持、软件、服务等的知识产权均属于提供一方或其合法权利人所有。 除提供方或合法权利人明示同意外，另一方无权复制、传播、转让、许可或提供他人使用上述知识成果，否则应承担相应的责任。

ii. 您应保证提交阿里云的素材、对阿里云服务的使用及使用阿里云服务所产生的成果未侵犯任何第三方的合法权益。阿里云应保证向您提供的服务未侵犯任何第三方的合法权益。

iii. 如果第三方机构或个人对您使用阿里云服务所涉及的相关素材的知识产权归属提出质疑或投诉，或对您使用的阿里云的服务的知识产权的归属提出质疑或投诉，您和阿里云均有责任出具相关知识产权证明材料，并配合对方相关投诉处理工作。对于因此引起的索赔、诉讼或可能向其提起诉讼，违约方应负责解决，承担费用和损失，以及使另一方完全免责。

8. 保密条款

i. 本服务条款所称保密信息，是指一方（以下简称“接受方”）从对方（以下简称“披露方”）取得的、获知的、或因双方履行本协议而产生的商业秘密（包括财务秘密）、技术秘密、经营诀窍和（或）其他应予保密的信息和资料（包括产品资料、产品计划、价格、财务及营销规划、业务战略、客户信息、客户数据、研发、软件、硬件、API应用数据接口、技术说明、设计、特殊公式、特殊算法等），无论上述信息和资料以何种形式或载于何种载体，无论披露方在披露时是否以口头、图像或书面等方式表明其具有保密性。

ii. 双方应采取适当措施妥善保存对方提供的保密信息，措施的审慎程度不少于其保护自身的保密信息时的审慎程度。双方仅能将保密信息用于与本协议项下的有关用途或目的。

iii. 双方保证保密信息仅可在各自一方从事该业务的负责人和雇员范围内知悉，并严格限制接触上述保密信息的员工遵守本条之保密义务。

iv. 本条上述限制条款不适用于以下情况：

a. 在签署本协议之时或之前，该保密信息已以合法方式属接受方所有。

b. 保密信息在通知给接受方时，已经公开或能从公开领域获得。

c. 保密信息是接受方从与其没有保密或不透露义务的第三方获得的。

d. 在不违反本协议约定责任的前提下，该保密信息已经公开或能从公开领域获得。

e. 该保密信息是接受方或其关联或附属公司独立开发，而且未从通知方或其关联或附属公司获得的信息中获益。

f. 接受方应法院或其它法律、行政管理部门要求（通过口头提问、询问、要求资料或文件、传唤、民事或刑事调查或其他程序）因而透露保密信息。

g. 接受方为向行政管理部门、行业协会等机构申请某项业务资质、获得某项认定、或符合国家、行业标准/认证，需结合对方情况向前述机构提交材料或进行说明的而披露的信息，在该等情况下，接受方应秉持必要情况下最少披露原则及要求因此获知保密信息的机构按不低于本协议的标准予以保密。

v. 您和阿里云都应尽最大的努力保护上述保密信息不被披露。一旦发现有上述保密信息泄露事件，双方应合作采取相应的合理措施避免或者减轻损害后果的产生。如因此给对方造成损失的，应赔偿因此给对方造成的直接经济损失。

9. 服务的开通、终止与变更

i. 先付费的服务：

a. 您付费后服务即开通，开通后您获得阿里云向您发送的登录、使用服务的密钥、口令即可使用服务，服务期限自开通之时起算（而非自您获得登录、使用服务的密钥、口令时起算）。

b. 以包年包月形式售卖的服务，服务期限至订购的期限届满为止。以资源包（或套餐包）形式售卖的服务，服务期限则至您订购的资源包服务期限到期或资源包中的服务被使用完毕为止（以前述二者早发生为准）。

c. 您应在服务期限内将资源包的服务数量使用完毕，如资源包的服务期限届满，您已订购但未使用完毕的服务将被作废且阿里云将不提供其他替代或补充。

d. 您对于服务的使用将优先消耗订购的资源包，除法定及双方另行约定外，如资源包中的各项服务使用完毕或者服务期限到期，且您未继续订购资源包服务但持续使用此项服务的，阿里云将视为您使用阿里云以后付费形式售卖的该服务（如有），阿里云将持续计费并根据计费结果予以扣划服务费用。

ii. 后付费的服务：
除非另有其他约定或您未结清其他应付款项的，您开通服务即可使用阿里云的服务。您应确保您的账户余额充足，以便持续使用服务至法律规定或本服务条款约定的终止情形出现时为止。

iii. 发生下列情形之一的，服务期限提前终止：

a. 双方协商一致提前终止的。

b. 您严重违反本协议（包括您严重违反相关法律法规规定，或您严重违反本协议项下之任一承诺内容等），阿里云有权提前终止服务直至清除您的全部数据。

c. 您理解并充分认可，虽然阿里云已经建立（并将根据技术的发展不断完善）必要的技术措施来防御包括计算机病毒、网络入侵和攻击破坏（如DDoS等）危害网络安全事项或行为（以下统称该等行为），但鉴于网络安全技术的局限性、相对性以及该等行为的不可预见性，因此如因您网站遭遇该等行为而给阿里云或者阿里云的其他网络或服务器（包括本地及外地和国际的网络、服务器等）带来危害，或影响阿里云与国际互联网或者阿里云与特定网络、服务器及阿里云内部的通畅联系，阿里云可决定暂停或终止服务。如果终止服务的，将按照实际提供服务月份计算（不足一个月的按天计）服务费用，将剩余款项（如有）返还。

d. 阿里云可提前30天在阿里云网站上通告或给您发网站内通知或书面通知的方式终止本服务服务协议。
届时阿里云应将您已支付但未消费的款项退还至您的阿里云账户。

iv. 您理解并认可，为技术升级、服务体系升级、或因经营策略调整或配合国家重大技术、法规政策等变化，阿里云不保证永久的提供某种服务，并有权变更所提供服务的形式、规格或其他方面（如服务的价格和计费模式），在终止该种服务或进行此种变更前，阿里云将尽最大努力且提前以网站公告、站内信、邮件或短信等一种或多种方式进行事先通知。

10. 违约责任

i. 您违反本协议中的承诺、保证条款、服务使用规则或义务的任一内容，或阿里云根据其判断认为您的使用行为存在异常的，阿里云均有权就其情节，判断并采取以下措施中的一种或多种：

a. 限制、中止使用服务。

b. 终止提供服务，终止本协议。

c. 追究您的法律责任。

d. 其他阿里云认为适合的处理措施。

阿里云依据前述约定采取中止服务、终止服务等措施而造成的用户损失由您承担。

ii. 如因您违反有关法律法规或者本协议、相关规则之规定，使阿里云遭受任何损失、受到其他用户、任何第三方的索赔或任何行政管理部门的处罚，您应对阿里云、其他用户或相关第三方的实际损失进行全额赔偿，包括合理的律师费用。

iii. 您理解且同意，鉴于计算机、互联网的特殊性，下述情况不属于阿里云违约：

a. 阿里云在进行系统及服务器配置、维护、升级时，需要短时间中断服务。

b. 由于Internet上的通路阻塞造成您网站访问速度下降。

iv. 如果因阿里云原因造成您连续72小时不能正常使用服务的，您可终止接受服务，但非阿里云控制之内的原因引起的除外。

v. 在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性的损害，包括您使用阿里云服务而遭受的利润损失承担责任（即使您已被告知该等损失的可能性）。

vi. 在法律允许的情况下，阿里云在本协议项下所承担的损失赔偿责任不超过就该服务过往12个月所缴纳的服务费用的总和。

11. 通知

i. 您在使用阿里云服务时，您应该向阿里云提供真实有效的联系方式（包括您的电子邮件地址、联系电话、联系地址等），对于联系方式发生变更的，您有义务及时更新有关信息，并保持可被联系的状态。您接收站内信、系统消息的会员账号（包括子账号），也作为您的有效联系方式。

ii. 阿里云将向您的上述联系方式的其中之一或其中若干向您送达各类通知，而此类通知的内容可能对您的权利义务产生重大的有利或不利影响，请您务必及时关注。

iii. 阿里云通过上述联系方式向您发出通知，其中以电子的方式发出的书面通知，包括但不限于公告，向您提供的联系电话发送手机短信，向您提供的电子邮件地址发送电子邮件，向您的账号发送系统消息以及站内信信息，在发送成功后即视为送达；以纸质载体发出的书面通知，按照提供联系地址交邮后的第五个自然日即视为送达。

iv. 您应当保证所提供的联系方式是准确、有效的，并进行实时更新。如果因提供的联系方式不确切，或不及时告知变更后的联系方式，使法律文书无法送达或未及时送达，由您自行承担由此可能产生的法律后果。

12. 不可抗力

i. 因不可抗力或者其他意外事件，使得本服务条款的履行不可能、不必要或者无意义的，遭受不可抗力、意外事件的一方不承担责任。

ii. 不可抗力、意外事件是指不能预见、不能克服并不能避免且对一方或双方当事人造成重大影响的客观事件，包括但不限于自然灾害如洪水、地震、瘟疫流行等以及社会事件如战争、动乱、政府行为、电信主干线路中断、黑客、网路堵塞、电信部门技术调整和政府管制等。

13. 法律适用及争议解决

i. 本协议之订立、生效、解释、修订、补充、终止、执行与争议解决均适用中华人民共和国大陆法律；如法律无相关规定的，参照商业惯例及/或行业惯例。

ii. 您因使用阿里云服务所产生及与阿里云服务有关的争议，由阿里云与您协商解决。协商不成时，任何一方均可向被告所在地有管辖权的人民法院提起诉讼。

14. 附则

    i. 本协议的附件，以及阿里云在阿里云网站相关页面上的服务说明、价格说明和您确认同意的订购页面（包括特定产品的专用条款、服务说明、操作文档等）均为本协议不可分割的一部分。如遇不一致之处，以（1）服务说明、价格说明、其他订购页面，（2）专用条款和4.3. 附件，（3）本协议通用条款的顺序予以适用。

    ii. 如本协议内容发生变动，阿里云应通过提前30天在阿里云网站的适当版面公告向您提示修改内容；如您继续使用阿里云服务，则视为您接受阿里云所做的相关修改。

    iii. 阿里云有权经提前将本协议的权利义务全部或者部分转移给阿里云的关联公司。

    iv. 阿里云于您过失或违约时放弃本协议规定的权利，不应视为其对您的其他或以后同类之过失或违约行为弃权。

    v. 本协议任一条款被视为废止、无效或不可执行，该条应视为可分的且并不影响本协议其余条款的有效性及可执行性。

    vi. 本协议项下之保证条款、保密条款、知识产权条款、法律适用及争议解决条款等内容，不因本协议的终止而失效。

Prometheus监控服务专家版专用条款

到期及欠费

1. 如包年、包月和包周等固定服务期限到期的：

    i. 就包年、包月和包周的Prometheus监控服务，服务期到期后，如您继续使用阿里云服务的，您应及时续费。

    ii. 您服务到期后，如未续费，阿里云将在到期时暂停您就该服务的操作权限，但仍保留实例、继续存储您的数据7日（自操作权限被暂停之日的暂停开始时刻至第7日相同时刻为期限届满）；如7日届满仍未续费，阿里云将释放您的实例，并删除实例数据。

2. 就按量付费的Prometheus监控服务，您应及时充值、缴纳服务费用以保证服务的持续使用，如您发生欠费：

    i. 自账户首次欠费之时起，阿里云将暂停您对账户下所有按量付费服务的操作权限，但仍保留实例、继续存储您的数据7日（自操作权限被暂停之日的暂停开始时刻至第7日相同时刻为期限届满）。

    ii. 如自您首次欠费发生之时起7日届满，您仍未成功充值并足以支付所欠服务费用的，阿里云将释放您的欠费实例，并删除数据。

附件

服务使用规则

1. 您承诺，如果您利用阿里云提供的服务进行经营或非经营的活动需要获得国家有关部门的许可或批准的，应获得该有关的许可或批准。包括但不限于以下内容：

    i. 如果您在云服务器服务上开办了多个网站，须保证所开办的全部网站均获得国家有关部门的许可或批准。

    ii. 如您网站提供非经营性互联网信息服务的，必须办理非经营性网站备案，并保证所提交的所有备案信息真实有效，在备案信息发生变化时及时在备案系统中提交更新信息。

    iii. 如您网站提供经营性互联网信息服务的，还应自行在当地通信管理部门取得经营性网站许可证。

    iv. 如您提供BBS等电子公告服务的，也需根据相关法规政策要求备案或获得相应批准。

    v. 如您经营互联网游戏网站的，您应依法获得网络文化经营许可证。

    vi. 如您经营互联网视频网站的，您应依法获得信息网络传播视听节目许可证。

vii. 若您从事新闻、出版、教育、医疗保健、药品和医疗器械等互联网信息服务，依照法律、行政法规以及国家有关规定须经有关主管部门审核同意，在申请经营许可或者履行备案手续前，应当依法经有关主管部门审核同意。
您理解并认可，以上列举并不能穷尽您进行经营或非经营活动需要获得国家有关部门的许可或批准的全部类型，您应获得有关的许可或批准，并应符合国家及地方不时颁布相关法律法规之要求。

2. 除阿里云明示许可外，您不应修改、翻译、改编、出租、转许可、在信息网络上传播或转让阿里云提供的服务或软件，也不得逆向工程、反编译或试图以其他方式发现阿里云提供的服务或软件的源代码。
3. 您不应不散布电子邮件广告、垃圾邮件（SPAM）：不利用阿里云提供的服务散发大量不受欢迎的或者未经请求的电子邮件、电子广告或包含反动、色情等有害信息的电子邮件。
4. 您不应利用阿里云提供的资源和服务上传（Upload）、下载（Download）、储存、发布如下信息或者内容，不为他人发布该等信息提供任何便利（包括但不限于设置URL、BANNER链接等）：
   i. 违反国家规定的政治宣传和/或新闻信息。
   ii. 涉及国家秘密和/或安全的信息。
   iii. 封建迷信和/或淫秽、色情、下流的信息或教唆犯罪的信息。
   iv. 博彩有奖、赌博游戏、“私服”、“外挂”等非法互联网出版活动。
   v. 违反国家民族和宗教政策的信息。
   vi. 妨碍互联网运行安全的信息。
   vii. 侵害他人合法权益的信息和/或其他有损于社会秩序、社会治安、公共道德的信息或内容。
   viii. 其他违反法律法规、部门规章或国家政策的内容。
5. 您不应大量占用，亦不得导致如程序或进程等大量占用阿里云云计算资源（如云服务器、网络带宽、存储空间等）所组成的平台（以下简称“云平台”）中服务器内存、CPU或者网络带宽资源（比如但不限于互联网挖矿等行为），并给阿里云云平台或者阿里云的其他用户的网络、服务器（包括但不限于本地及外地和国际的网络、服务器等）、产品/应用等带来严重的、不合理的负荷，影响阿里云与国际互联网或者阿里云与特定网络、服务器及阿里云内部正常通畅的联系，或者导致阿里云云平台产品与服务或者阿里云的其他用户的服务器宕机、死机或者用户基于云平台的产品/应用不可访问等。
6. 您不应进行任何破坏或试图破坏网络安全的行为（包括但不限于钓鱼、黑客、网络诈骗、网站或空间中含有或涉嫌散播：病毒、木马、恶意代码，及通过虚拟服务器对其他网站、服务器进行涉嫌攻击行为如扫描、嗅探、ARP欺骗、DOS等）。
7. 您不应进行任何改变或试图改变阿里云提供的系统配置或破坏系统安全的行为。
8. 除非您购买了专门用于此目的的服务，您不利用本服务从事DDoS防护、DNS防护等防护售卖业务。
9. 不利用阿里云的服务提供任何不经网络审查或依靠技术手段成为境内获取境外非法信息的途径。
10. 您不应在阿里云服务或平台之上安装、使用盗版软件，您对自己行为（如自行安装的软件和进行的操作）承担责任。