ALIBABA CLOUD

阿里云

应用实时监控服务 ARMS
Prometheus监控

文档版本：20200901

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

1.什么是Prometheus监控？

ARMS 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。

Prometheus简介

Prometheus是一套开源的系统监控和报警框架，灵感源自Google的Borgmon监控系统。2012年，SoundCloud的Google前员工创造了Prometheus，并作为社区开源项目进行开发。2015年，该项目正式发布。2016年，Prometheus加入云原生计算基金会（Cloud Native Computing Foundation），成为受欢迎度仅次于Kubernetes的项目。

Prometheus具有以下特性：

- 多维的数据模型（基于时间序列的Key、Value键值对）
- 灵活的查询和聚合语言PromQL
- 提供本地存储和分布式存储
- 通过基于HTTP的Pull模型采集时间序列数据
- 可利用Pushgateway（Prometheus的可选中间件）实现Push模式
- 可通过动态服务发现或静态配置发现目标机器
- 支持多种图表和数据大盘

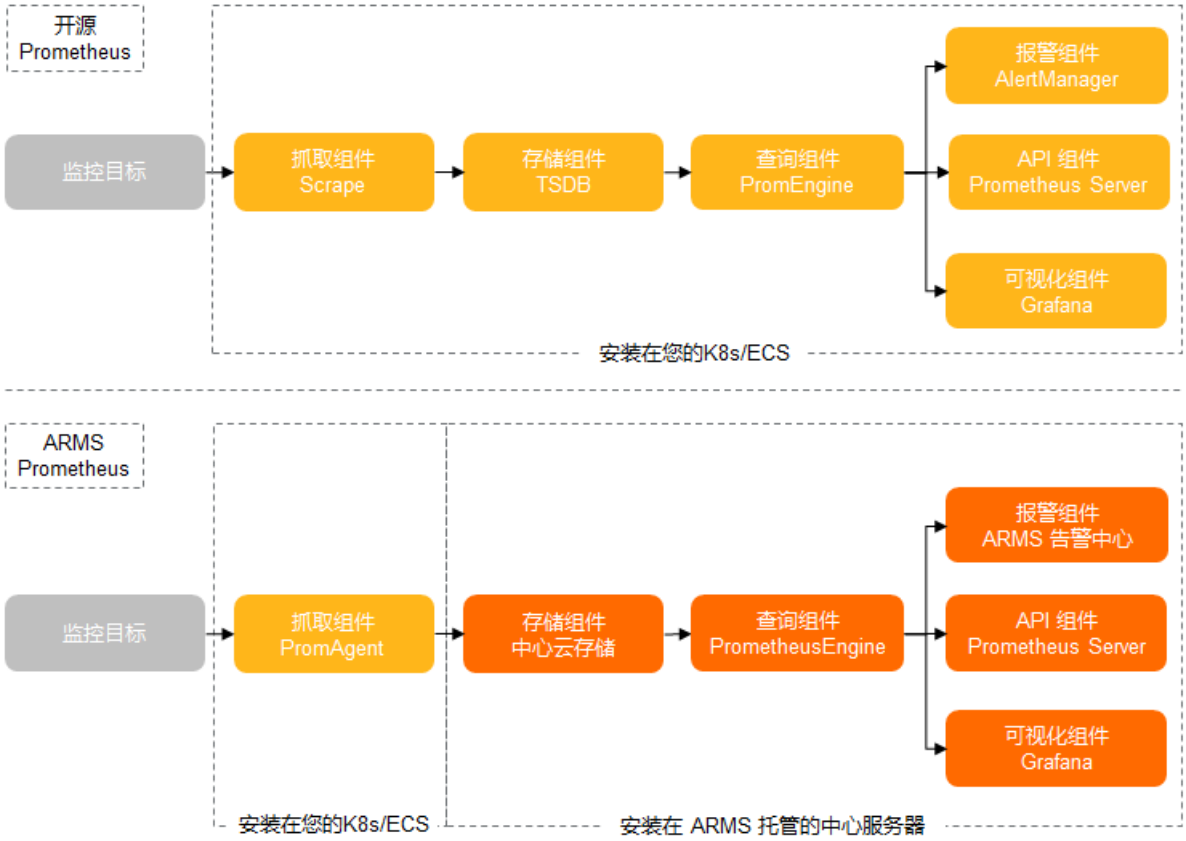
ARMS Prometheus监控和开源Prometheus监控对比的优势

整体而言，与开源Prometheus监控相比，ARMS Prometheus监控的优势体现为：

- 更轻量、更稳定、更准确
- 数据量无上限
- 完全兼容开源生态
- 节省成本

更轻量更稳定更准确

- 与开源Prometheus监控相比，ARMS Prometheus监控的整体结构更加轻量化。您无需自行搭建Prometheus监控系统，仅需安装ARMS Prometheus监控探针PromAgent即可开始监控业务。



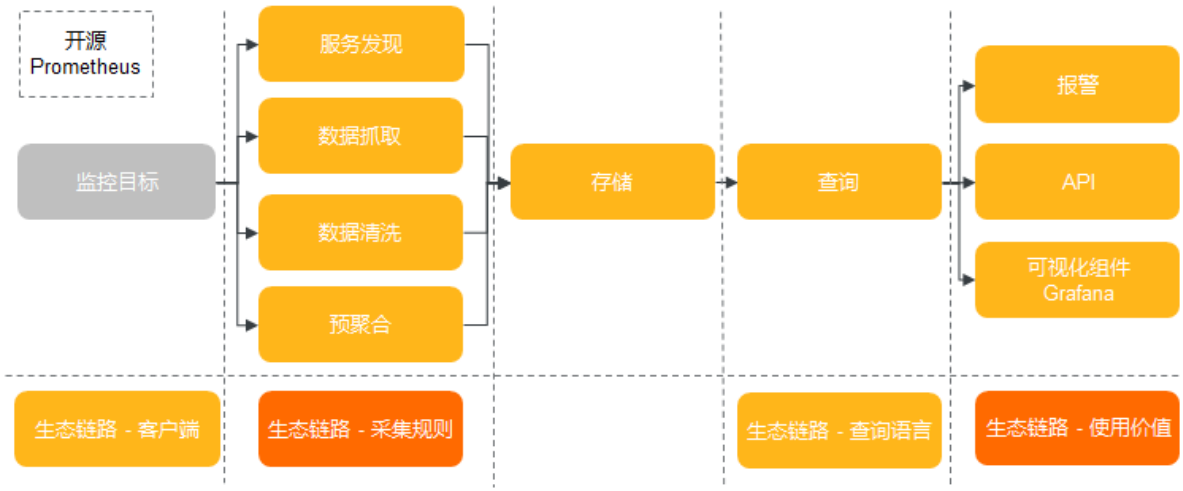
- 在系统稳定性方面，开源Prometheus监控一般会占用16GB~128GB的内存，而ARMS Prometheus监控仅占用200MB~1GB的内存和1核CPU。相比开源Prometheus监控，ARMS Prometheus监控更加稳定。
- 在抓取和写入数据的准确性方面，开源Prometheus监控仅抓取1次数据，并且瞬时写入存储组件时存在丢弃逻辑。而ARMS Prometheus监控抓取数据会重试多次，持续并发写入存储组件，不存在丢弃逻辑。

数据量无上限

- 开源Prometheus监控的数据采集能力上限为百万条Metrics级别，而ARMS Prometheus监控的数据采集能力可以按照K8s副本数水平扩展，从而均衡分解采集任务。
- 开源Prometheus监控的数据存储能力上限受本地磁盘大小的限制，而ARMS Prometheus监控使用中心云存储服务，理论上存储能力无上限。

完全兼容开源生态

- ARMS Prometheus监控完全兼容Prometheus监控开源生态链路中的客户端和查询语言部分，兼容并优化开源生态链路中的采集规则和使用价值部分。



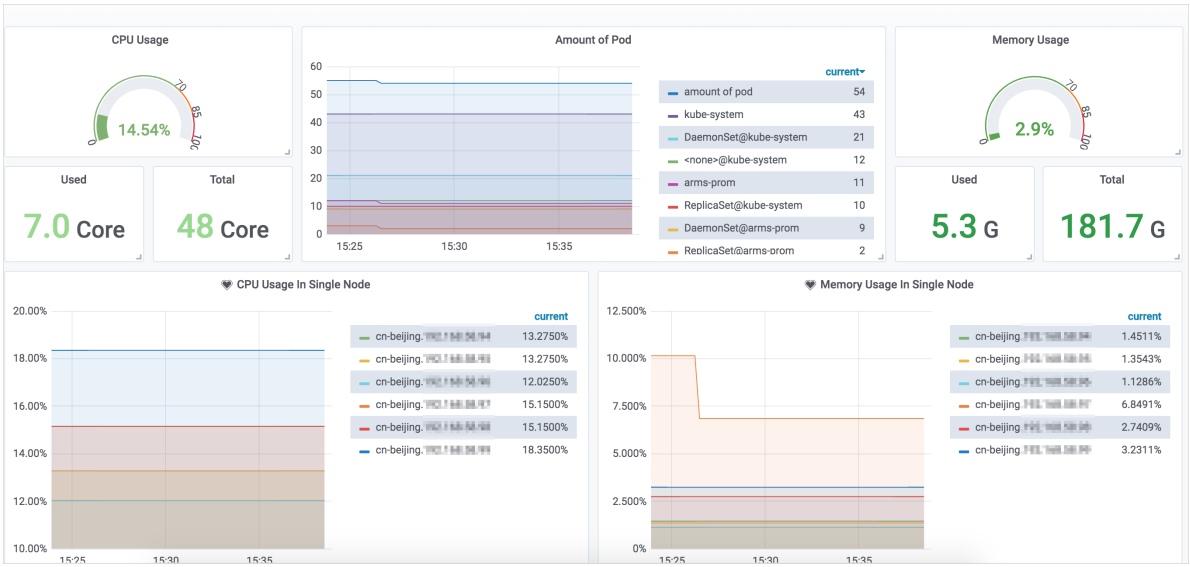
- ARMS Prometheus监控兼容并提供三种主流采集规则的实现，包括标准开源*prometheus.yaml*采集规则配置文件、适合自定义K8s内监控的采集规则ServiceMonitor、以及默认采集规则Annotation。与开源Prometheus监控相比，ARMS Prometheus监控无需重启，即可使用*prometheus.yaml*配置文件动态更新采集规则。在*Deployment*文件里也无需编写多行代码，仅需增加以下3个Annotation注解即可。

```
prometheus.io/scrape: "true"
prometheus.io/port: "9090"
prometheus.io/path: "/metrics"
```

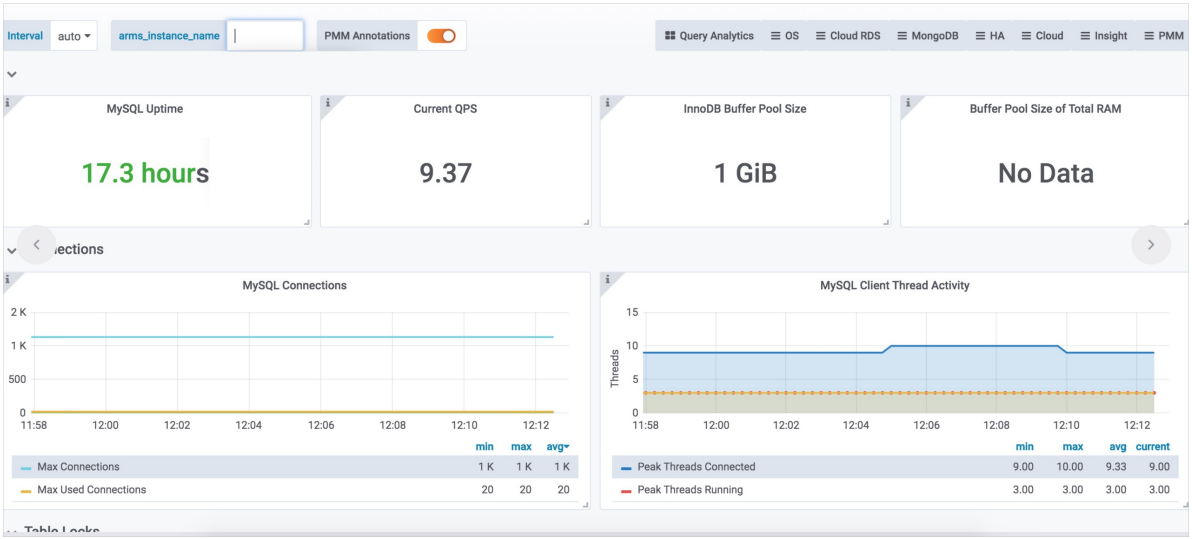
- ARMS Prometheus监控兼容可视化Grafana。通过配置Prometheus HTTP API URL即可在Grafana中完成数据源的多租户隔离，以及Grafana大盘的多租户隔离。ARMS Prometheus监控还兼容Grafana的Explore数据调试模块。
- ARMS Prometheus监控兼容开源Prometheus监控的HTTP API模块，完整支持query、query_range和labelValues 3个标准数据查询接口，并通过在数据URL中添加*/userId/clusterId/regionId/*这组ID达到多租户隔离的效果。
- ARMS Prometheus监控虽然使用ARMS已有的告警系统，但完全兼容开源Prometheus监控的告警规则PromQL。

节省成本

- ARMS Prometheus监控支持默认K8s监控。在您安装默认K8s监控后，ARMS Prometheus监控会自动为您创建默认的Exporter、采集规则、Grafana大盘以及ARMS告警。您的时间成本可由原来使用开源Prometheus监控K8s的3天左右降低至10分钟左右。



- ARMS Prometheus监控支持开源组件监控。您仅需输入阿里云账号的AccessKey ID和AccessKey Secret，以及RDS和Redis组件的账号和密码，ARMS Prometheus监控即可为您默认生成这些组件的Exporter，并为您创建默认的组件大盘。您的时间成本可由原来使用开源Prometheus监控开源组件的7天左右降低至3分钟左右。



- ARMS Prometheus监控支持一键安装、一键卸载，以及通过健康检查功能调试Prometheus监控。您的时间成本可由原来使用开源Prometheus监控的1天左右降低至3分钟左右。

相关文档

- Prometheus 监控商用通知

2.版本说明

本文主要介绍Prometheus监控相关内容的最新动态。

2020年4月

版本号	镜像地址	变更时间	变更内容
arms-prom-operator:v0.1	*registry.cn-hangzhou.aliyuncs.com/arms-docker-repo/arms-prom-operator:v0.1*	2020年4月16日	新功能如下所示： • 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。

3.开始使用Prometheus监控

对于部署在阿里云容器服务Kubernetes版中的Kubernetes集群，您可以在ARMS中为其一键安装Prometheus监控插件，此后即可通过ARMS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

- 在阿里云容器服务Kubernetes版中创建Kubernetes集群，详情请参见快速创建Kubernetes托管版集群。
- 开通和升级ARMS

安装ARMS Prometheus监控组件

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，选择**市场 > 应用目录**。
3. 在**应用目录**页面的**阿里云应用**页签中，单击**ack-arms-prometheus**。
4. 在**应用目录 - ack-arms-prometheus**页面，单击**参数**。
5. 在**参数**页签，配置集群ID参数cluster_id。

   > **说明**
   >
   > ACK自动匹配目标集群ID。在控制台左侧导航栏选择**集群**进入**集群列表**，目标集群的名称下面的字符串即为集群ID。

6. 在**应用目录 - ack-arms-prometheus**右侧的创建面板中，选择目标集群，然后单击**创建**。

> **说明** 命名空间和发布名称默认为arms-prom。

打开Prometheus监控大盘

1. 登录ARMS控制台。
2. 在左侧导航栏单击**Prometheus监控**。
3. 在**Prometheus监控**页面左上角选择目标地域，并单击**已安装大盘**列中的链接，即可在浏览器新窗口中打开对应的监控大盘。

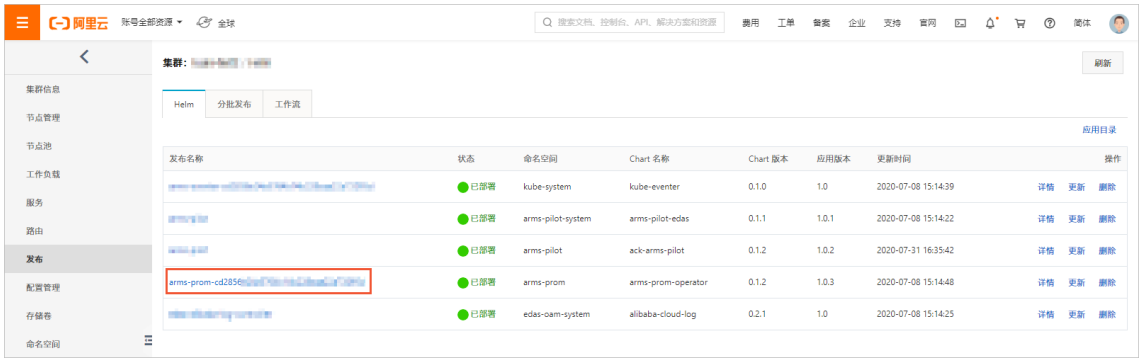
卸载监控插件

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。

1. 登录ARMS控制台。
2. 在左侧导航栏单击**Prometheus监控**。
3. 在**Prometheus监控**页面左上角选择目标地域，单击**操作**列中的**卸载**，并在**确认**对话框中单击**确认**。
   卸载插件完毕后，**已安装大盘**列中的大盘将会消失。接下来需要前往容器服务管理控制台确认卸载是否成功。
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。

   > **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**。

6. 在左侧导航栏单击**发布**，并根据情况采取以下任一操作：
   - 如果**发布**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**发布**页面有 `arms-prom-****` 记录，请在其右侧的**操作**列中单击**删除**。



相关文档

- 什么是Prometheus监控？

4.接入指南

4.1. Exporter接入

4.1.1. 使用ARMS Prometheus监控Redis

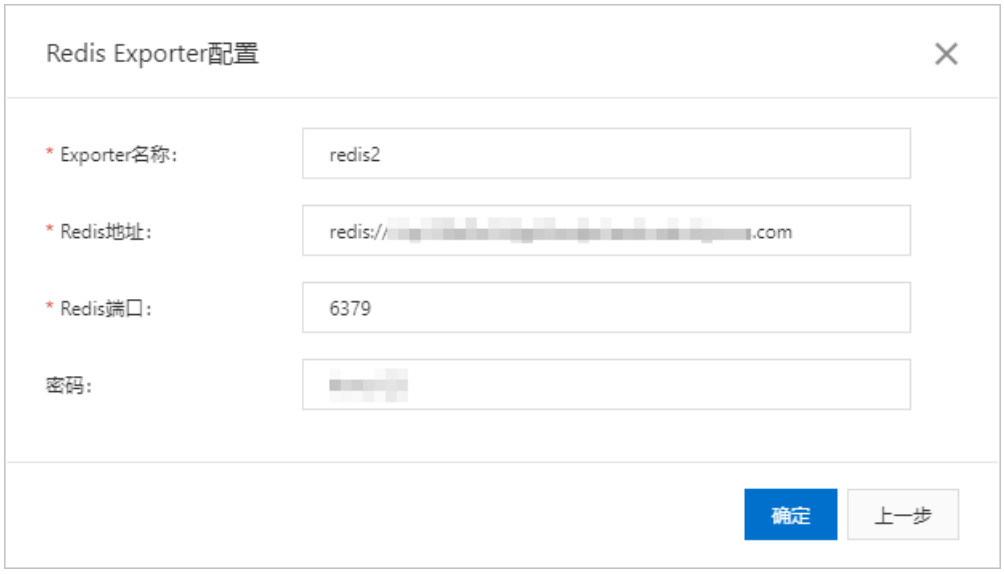
ARMS Prometheus监控提供一键安装配置Redis类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

添加Redis类型的Exporter

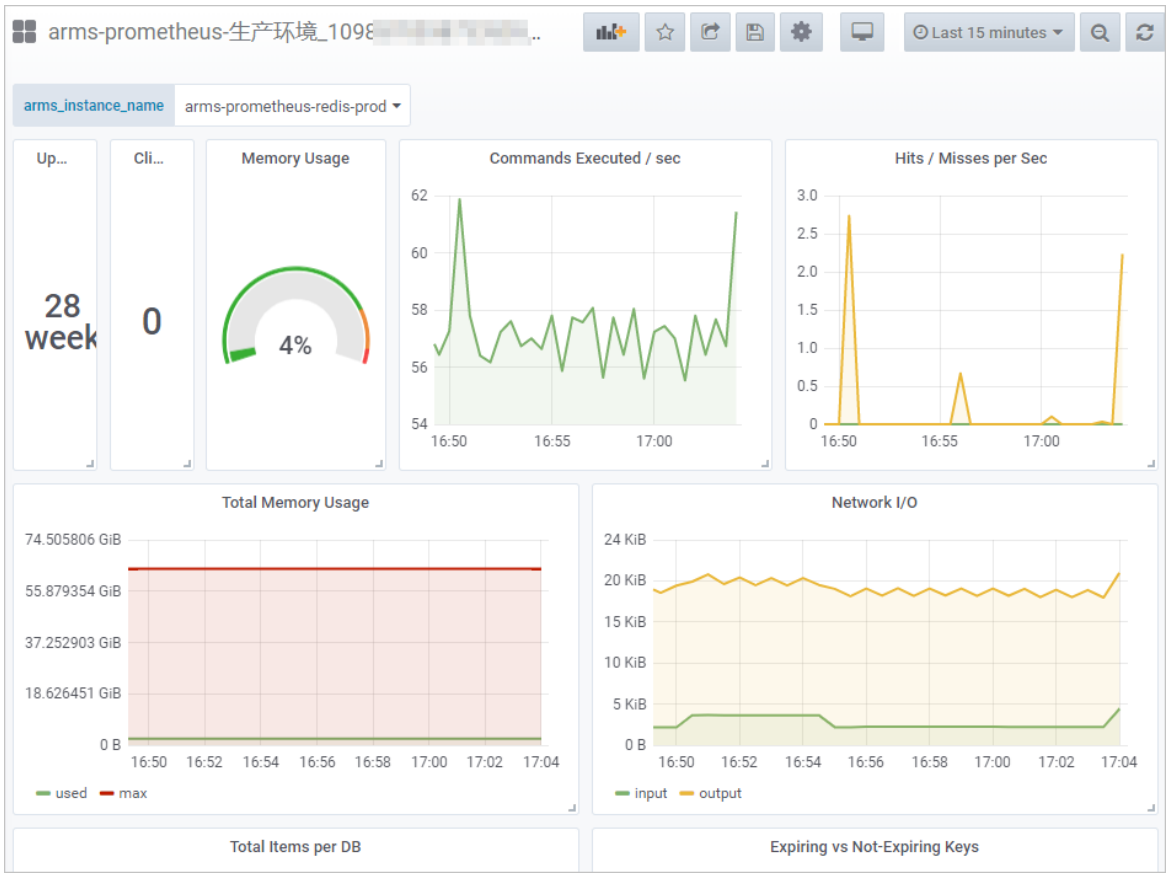
1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在Redis Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter类型及数字后缀组成。
Redis地址	Redis的连接地址。

参数	描述
Redis端口	Redis的端口号，例如：6379。
密码	Redis的连接密码。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.2. 使用ARMS Prometheus监控MySQL

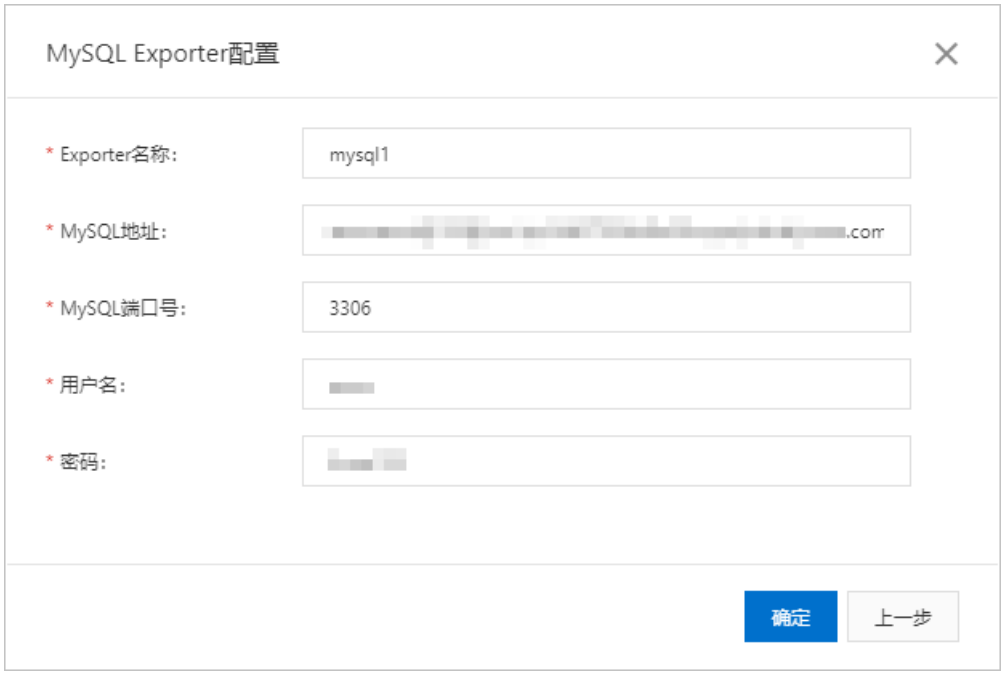
ARMS Prometheus监控提供一键安装配置MySQL类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

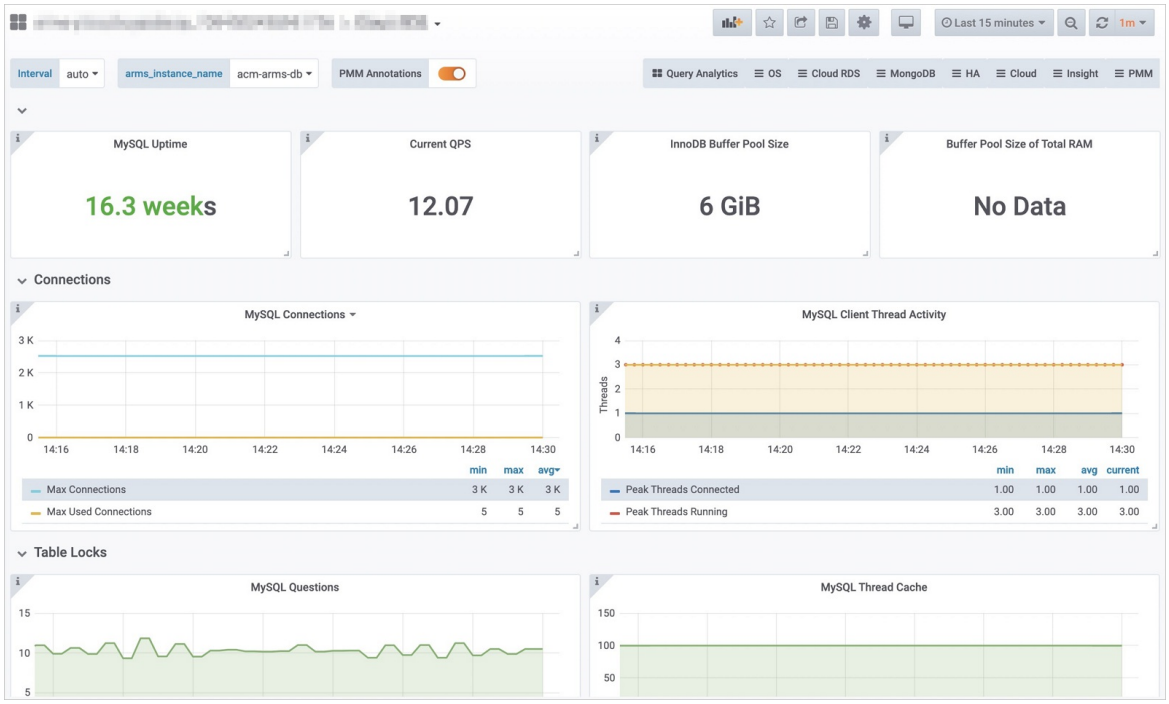
添加MySQL类型的Exporter

1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在MySQL Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter类型及数字后缀组成。
MySQL地址	MySQL的连接地址。
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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.3. 使用ARMS Prometheus监控ElasticSearch

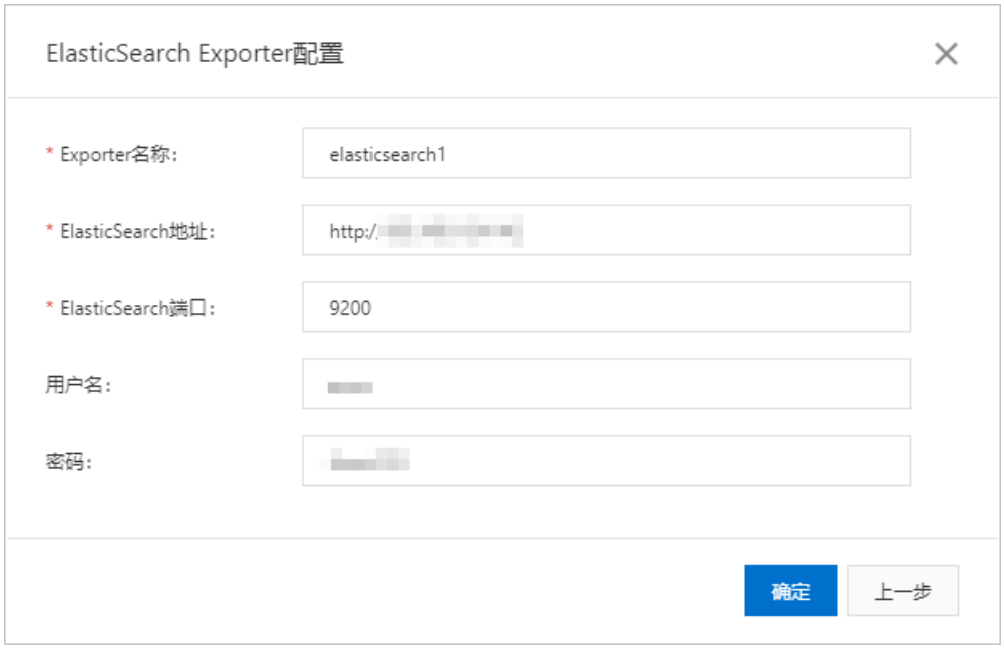
ARMS Prometheus监控提供一键安装配置ElasticSearch类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

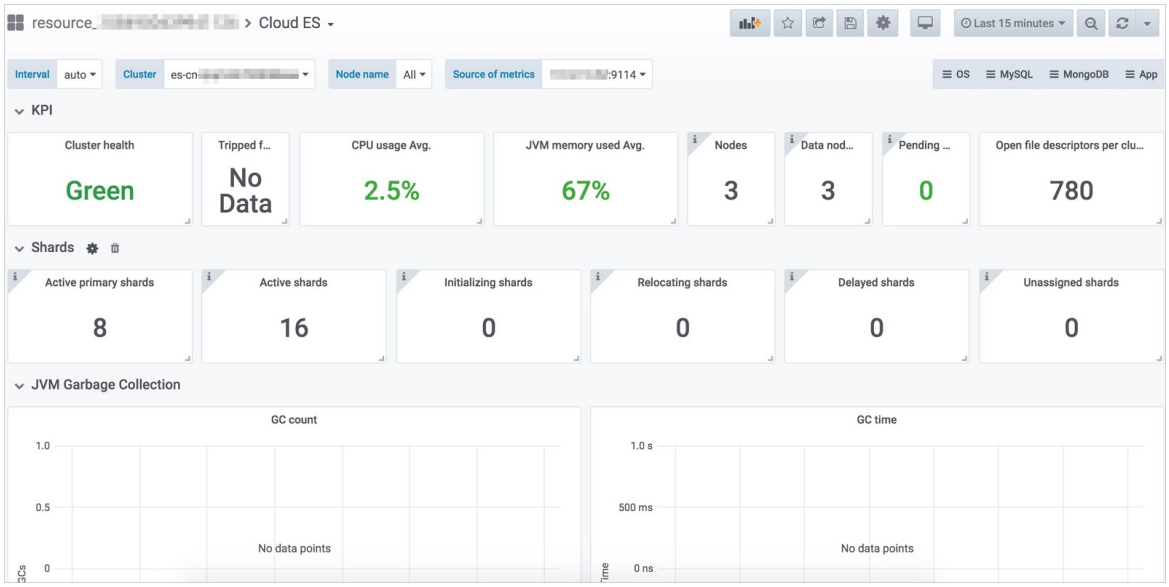
添加ElasticSearch类型的Exporter

1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在ElasticSearch Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.4. 使用ARMS Prometheus监控MongoDB

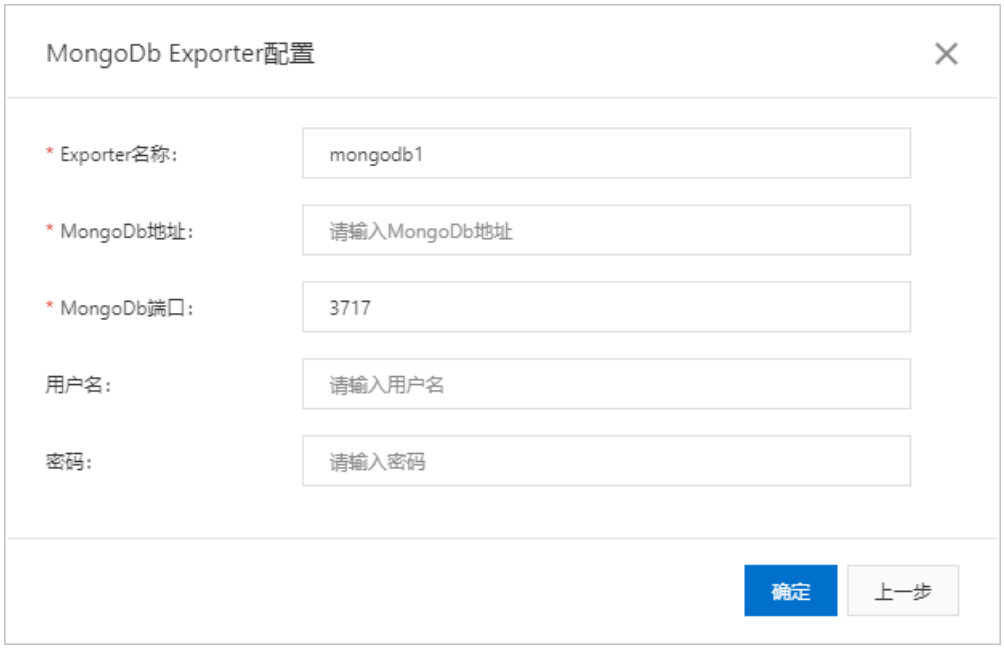
ARMS Prometheus监控提供一键安装配置MongoDB类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

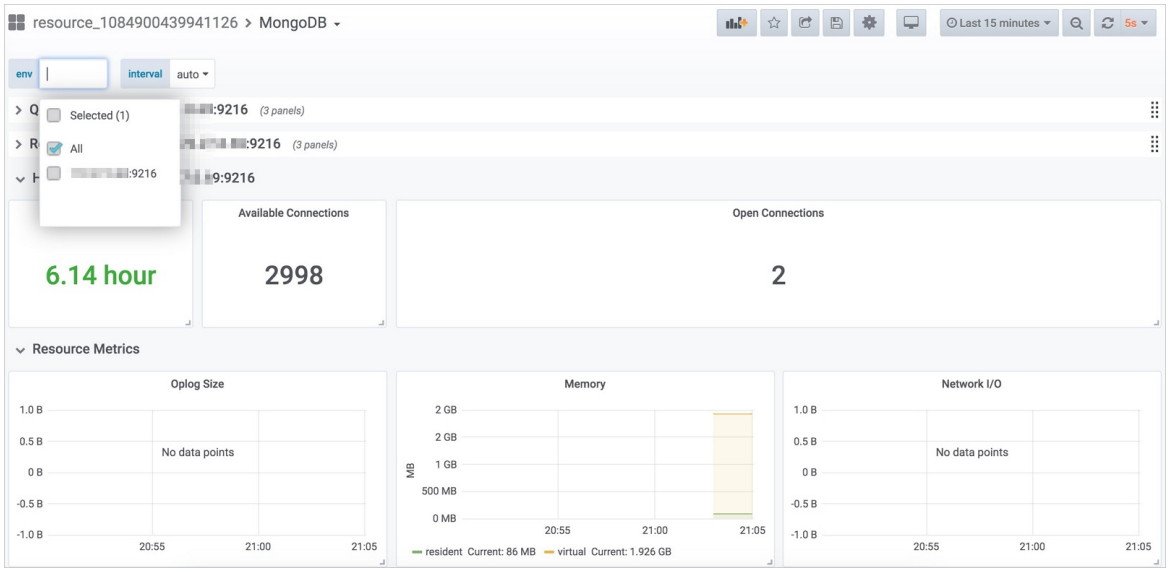
添加MongoDB类型的Exporter

1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在MongoDb Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.5. 使用ARMS Prometheus监控PostgreSQL

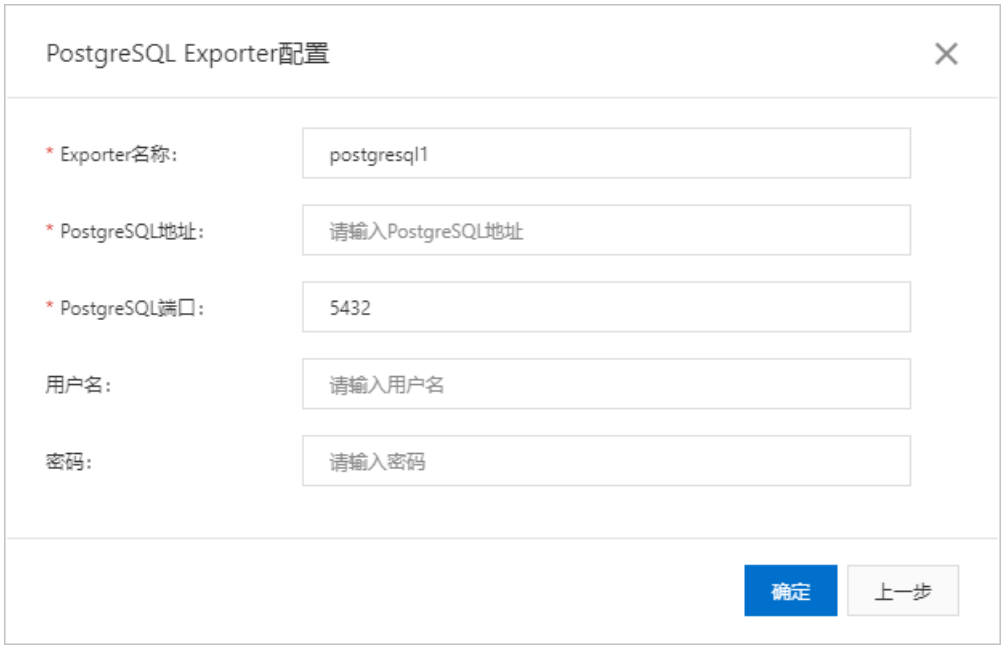
ARMS Prometheus监控提供一键安装配置PostgreSQL类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

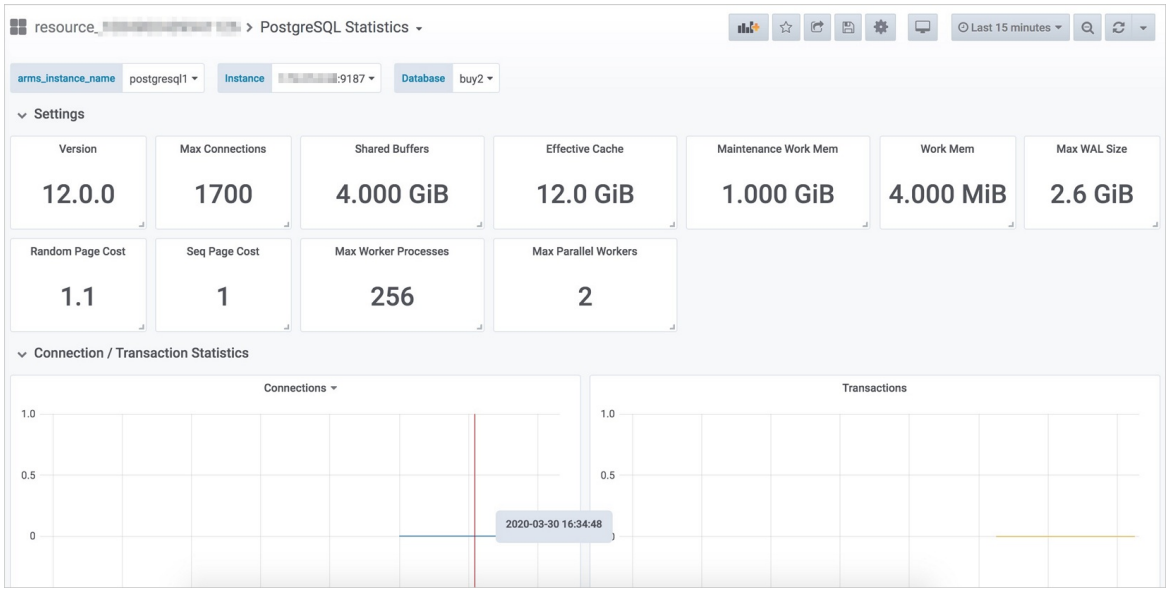
添加PostgreSQL类型的Exporter

1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在PostgreSQL Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.6. 使用ARMS Prometheus监控Kafka

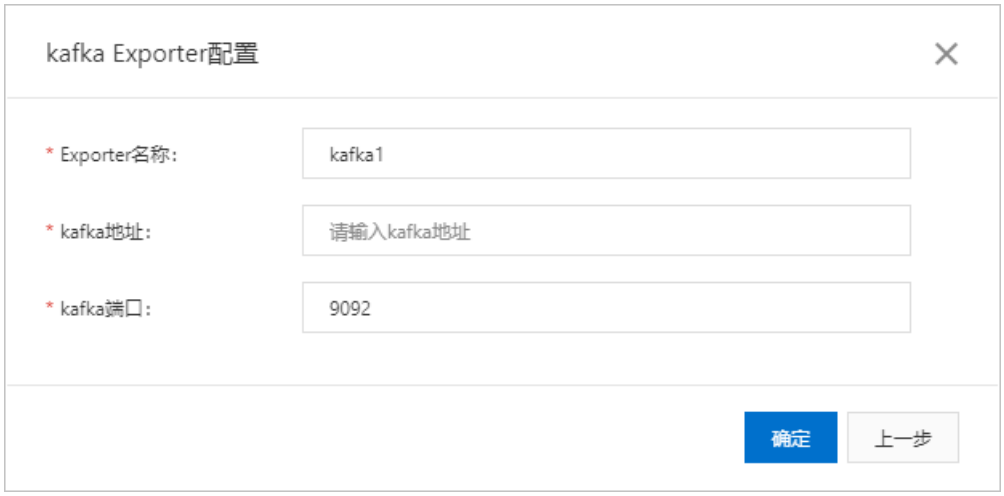
ARMS Prometheus监控提供一键安装配置Kafka类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

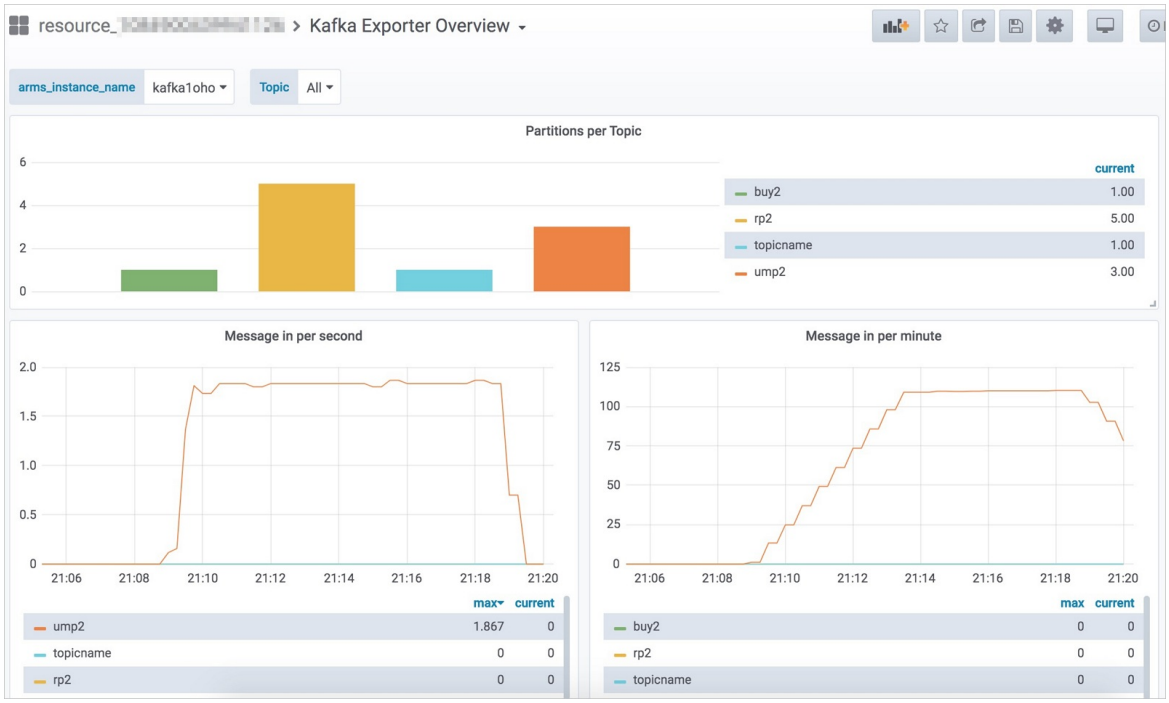
添加Kafka类型的Exporter

1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在kafka Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。

- 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.7. 使用ARMS Prometheus监控RabbitMQ

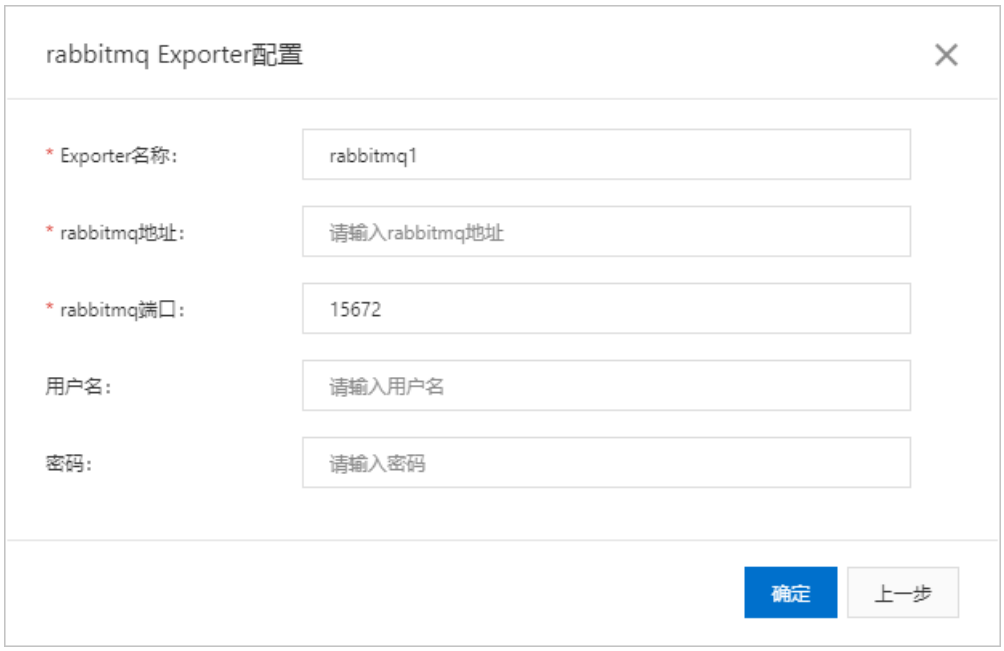
ARMS Prometheus监控提供一键安装配置RabbitMQ类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

添加RabbitMQ类型的Exporter

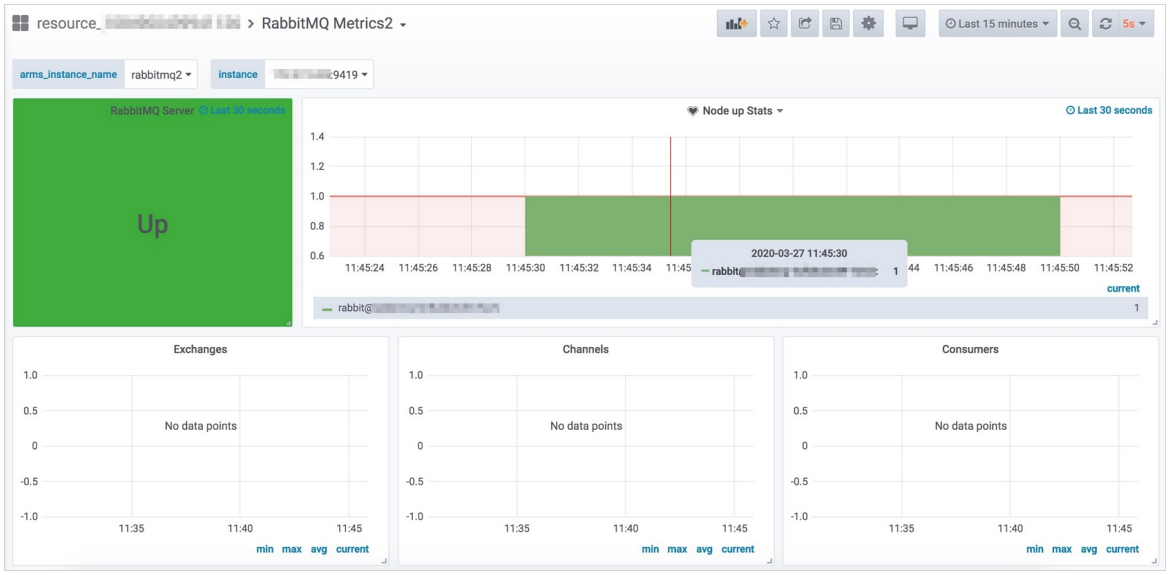
1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在rabbitmq Exporter配置对话框输入各项参数，并单击确定。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter类型及数字后缀组成。

参数	描述
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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.8. 使用ARMS Prometheus监控Nginx（旧版）

ARMS Prometheus监控提供一键安装配置Nginx类型Exporter的功能，并提供开箱即用的专属监控大盘。本文介绍旧版Nginx类型Exporter的安装配置详情。

背景信息

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

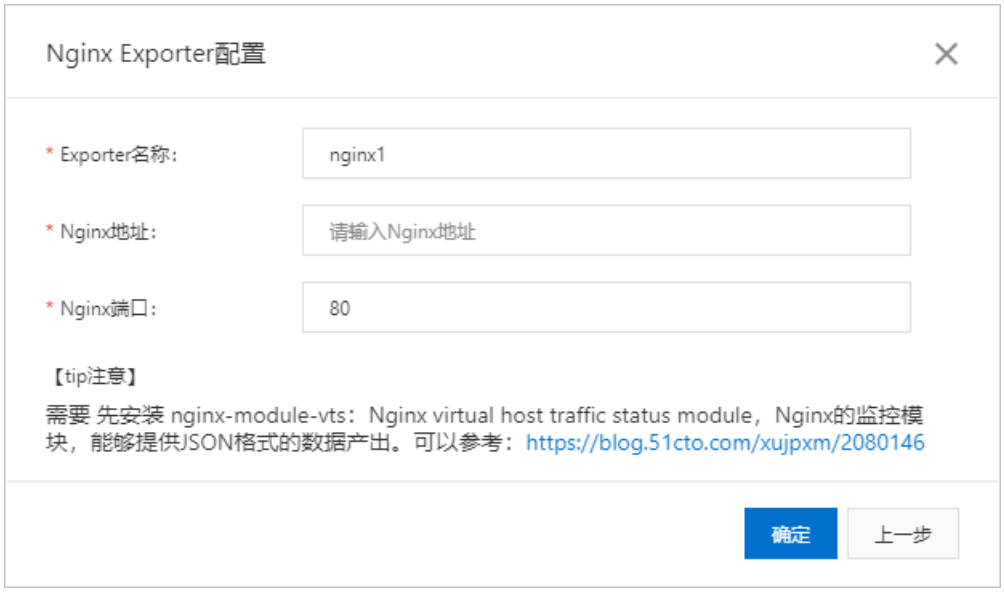
指标	类型	描述
nginx_server_requests	Server	Server请求数
nginx_server_bytes	Server	Server字节数
nginx_server_cache	Server	Server缓存
nginx_filter_requests	Filter	Filter请求数
nginx_filter_bytes	Filter	Filter字节数
nginx__filter_responseMsec	Filter	Filter响应时间
nginx_upstream_requests	Upstreams	上行请求数
nginx_upstream_bytes	Upstreams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

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击**Prometheus监控**，并在**Prometheus监控**页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击**Exporter接入**。

添加Nginx类型的Exporter

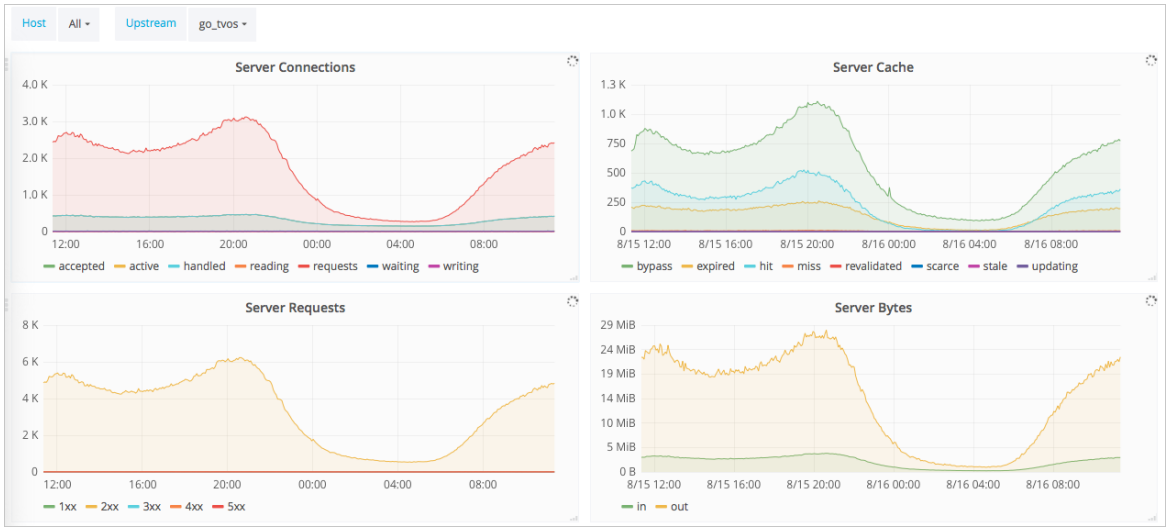
1. 在**Exporter接入**页面，单击右上角的**添加Exporter**。
2. 在**Exporter列表**对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在**Nginx Exporter配置**对话框输入各项参数，并单击**确定**。



参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

4. **在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。**



5. **（可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：**

- 单击操作列的删除，可删除已添加的Exporter。
- 单击操作列的日志，可查看Exporter的运行日志。
- 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 使用ARMS Prometheus监控Nginx（新版）
- 其他类型的Exporter接入

4.1.9. 使用ARMS Prometheus监控Nginx（新版）

ARMS Prometheus监控提供一键安装配置Nginx类型Exporter的功能，并提供开箱即用的专属监控大盘。本文介绍新版Nginx类型Exporter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Exporter。

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。

```
[root@ ~]# nginx -V 2>&1 | grep -o with-http_stub_status_module
with-http_stub_status_module
[root@ ~]#
```

- 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。

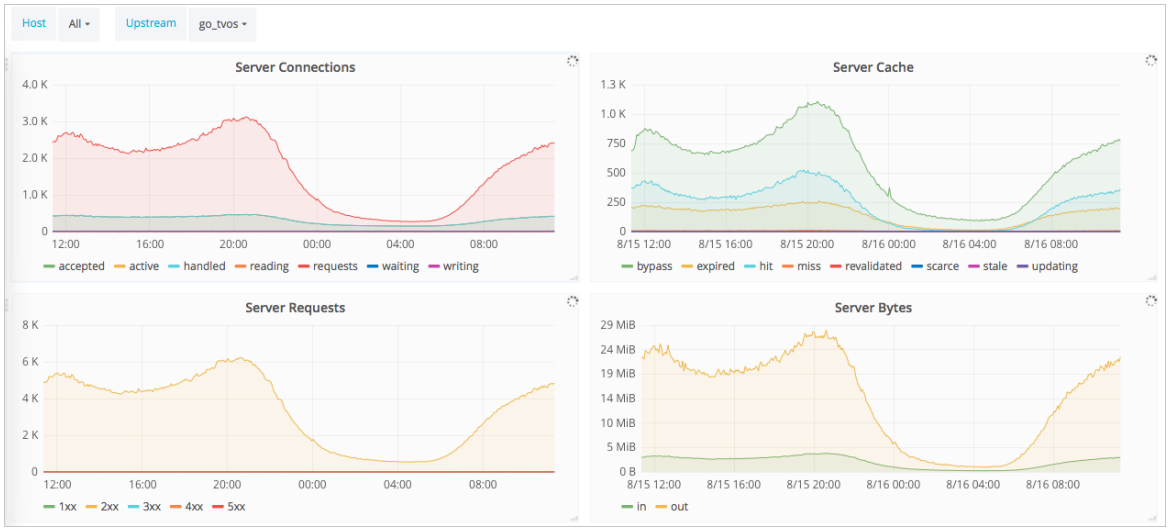
```
[root@ ~]# curl 127.0.0.1/nginx_status
Active connections: 1
server accepts handled requests
 177 177 3956
Reading: 0 Writing: 1 Waiting: 0
[root@ ~]#
```

步骤二：安装新版Nginx类型的Exporter

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。
4. 在Exporter接入页面，单击右上角的添加Exporter。
5. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
6. 在Nginx Exporter配置对话框输入各项参数，并单击确定。

参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

7. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



8. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 使用ARMS Prometheus监控Nginx（旧版）
- 其他类型的Exporter接入

4.1.10. 使用ARMS Prometheus监控Zookeeper

ARMS Prometheus监控提供一键安装配置Zookeeper类型Exporter的功能，并提供开箱即用的专属监控大盘。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，并在Prometheus监控页面顶部选择所需地域，然后单击目标K8s集群名称。
3. 在左侧导航栏单击Exporter接入。

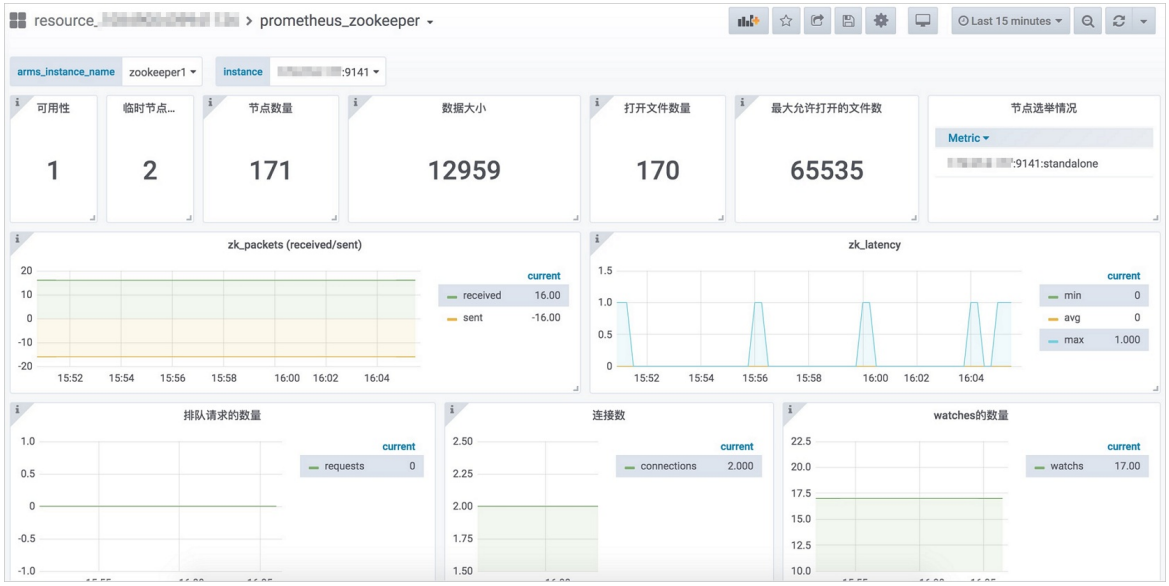
添加Zookeeper类型的Exporter

1. 在Exporter接入页面，单击右上角的添加Exporter。
2. 在Exporter列表对话框右上角的搜索框中，输入要添加的Exporter名称，并在筛选结果中单击目标Exporter图标。
3. 在Zookeeper Exporter配置对话框输入各项参数，并单击确定。

参数	描述
Exporter名称	Exporter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。 默认名称由Exporter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。

4. 在Exporter接入页面，单击面板列的大盘链接，查看Prometheus监控大盘。



5. （可选）在Exporter接入页面，可对已添加的Exporter执行以下操作：
   - 单击操作列的删除，可删除已添加的Exporter。
   - 单击操作列的日志，可查看Exporter的运行日志。
   - 单击操作列的详情，可查看Exporter的详情，包括Exporter的环境变量和描述信息。

相关文档

- 其他类型的Exporter接入

4.1.11. 其他类型的Exporter接入

ARMS 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Exporter的功能，并提供开箱即用的专属监控大盘。

添加其他类型的Exporter

ARMS将陆续支持其他各种类型的Exporter的一键接入功能，如您所需的Exporter还未支持，请先使用手动方式安装Exporter、配置服务发现并创建大盘，详情请参见手动接入MySQL监控。

Prometheus Exporter列表请参见Exporters and integrations。

相关文档

- 客户端接入概述

4.2. 客户端接入

4.2.1. 客户端接入概述

本教程系列介绍如何使用 ARMS Prometheus 监控您的 JVM、MySQL、Go、Redis 等应用或组件，并以 Grafana 大盘展示监控数据。

4.2.2. 通过ARMS Prometheus监控JVM

通过在应用中埋点来暴露JVM数据，使用ARMS Prometheus监控采集JVM数据，借助ARMS Prometheus Grafana大盘来展示JVM数据，并创建报警，即可实现利用ARMS Prometheus监控JVM的目的。本文档以安装在阿里云容器服务K8s集群上的应用为例，介绍如何通过ARMS Prometheus监控JVM。

Prometheus监控 JVM监控

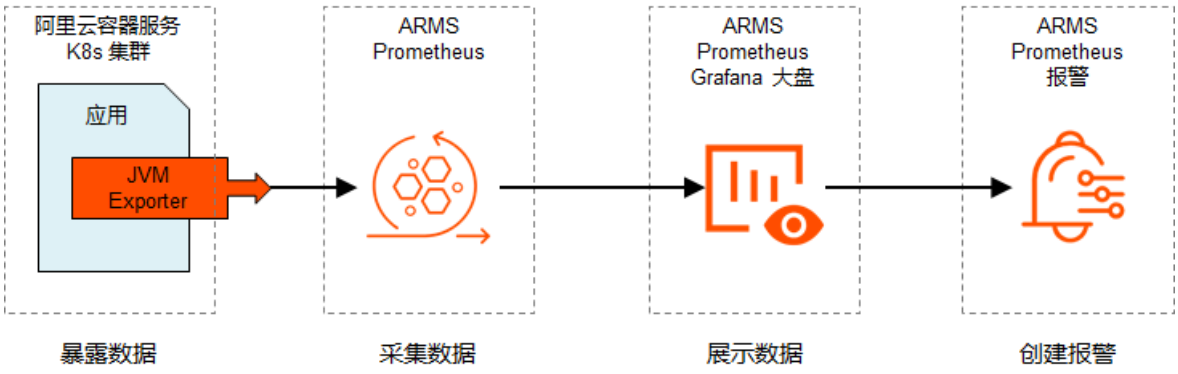
前提条件

您已完成以下操作：

- 下载Demo工程
- 快速创建Kubernetes集群
- 使用私有镜像仓库创建应用

操作流程

本教程的操作流程如图所示。



步骤一：通过埋点暴露JVM数据

您需要在应用中使用JVM Exporter暴露JVM数据。

具体步骤如下：

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. 在可以执行初始化的位置添加初始化JVM Exporter的方法。

   例如Demo工程的*/src/\main/java/com/monitise/prometheus_demo/DemoController.java*文件。

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

3. 在*/src/main/resources/application.properties*文件中配置用于Prometheus监控的端口（Port）和路径（Path）。

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

4. 在*/src/main/java/com/monitise/prometheus_demo/PrometheusDemoApplication.java*文件中打开HTTP端口。

```
@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);
}
}
```

步骤二：将应用部署至阿里云容器服务K8s集群

您需要将该应用部署至容器服务K8s集群，以便ARMS Prometheus监控采集JVM数据。

具体步骤如下：

1. 逐行运行Demo工程中的*buildDockerImage.sh*中的以下命令。

```
mvn clean install -DskipTests
docker build -t <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> . --no-cache
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>
sudo docker push <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
```

例如：

```
mvn clean install -DskipTests
docker build -t promethues-demo:v0 . --no-cache
sudo docker tag promethues-demo:v0 registry.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0
sudo docker push registry.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0
```

此步骤构建了名为promethues-demo的Docker镜像，并将镜像推送至阿里云容器镜像服务ACR。

2. 登录容器服务Kubernetes版控制台。
3. 在左侧导航栏选择**集群 > 集群**，在**集群列表**页面上的目标集群右侧**操作**列单击**控制台**。

4. 在左侧导航栏选择**工作负载 > 部署**，在页面右上角单击**创建**，并在**使用文本创建**页签的文本框中填写以下内容。

> **说明** 以下配置文件中的prometheus.io/port和prometheus.io/path的值分别为**步骤一**中暴露的ARMS Prometheus监控端口和路径。

```
apiVersion: extensions/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.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0
ports:
- containerPort: 8080
name: tomcat-normal
- containerPort: 8081
name: tomcat-monitor
```

此步骤将步骤中创建的名称为promethues-demo的Docker镜像部署至容器服务K8s集群中。

5. 在左侧导航栏选择**服务发现与负载均衡 > 服务**，在页面右上角单击**创建**，并在**使用文本创建**区域填写以下内容。

```
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
```

步骤三：配置ARMS Prometheus监控以采集JVM数据

您需要在ARMS控制台配置ARMS Prometheus监控以采集JVM数据。

具体步骤如下：

1. 登录ARMS控制台。
2. 在左侧导航栏中单击**Prometheus监控**。
3. 在**Prometheus监控**页面顶部选择容器服务K8s集群所在的地域，并在目标集群右侧的**操作**列单击**安装**。
   ARMS Prometheus Agent安装完毕后，请继续执行下一步操作。
4. 在目标集群右侧的**操作**列单击**设置**。
5. 在**配置详情**页签上单击**添加ServiceMonitor**，在**新增ServiceMonitor**对话框中填写以下内容。

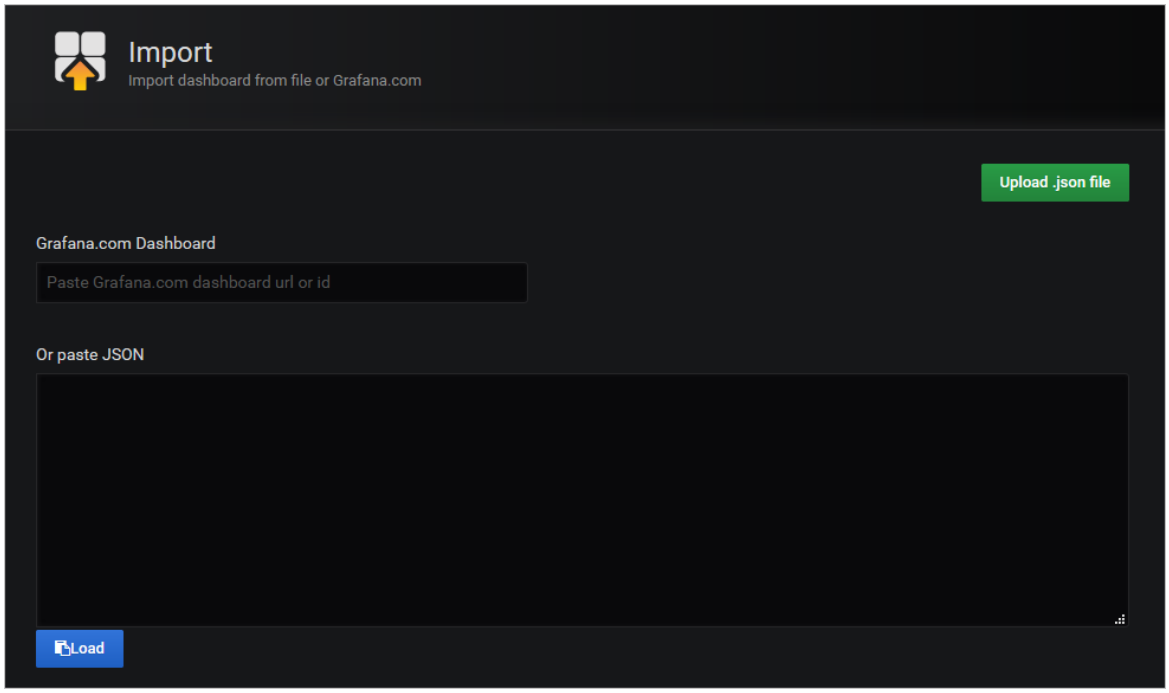
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
# 填写一个唯一名称。
name: tomcat-demo
# 填写目标命名空间。
namespace: default
spec:
endpoints:
- interval: 30s
# 填写Prometheus Exporter对应的Port的Name字段的值。
port: tomcat-monitor
# 填写Prometheus Exporter对应的Path的值。
path: /prometheus-metrics
namespaceSelector: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

步骤四：通过Grafana大盘展示JVM数据

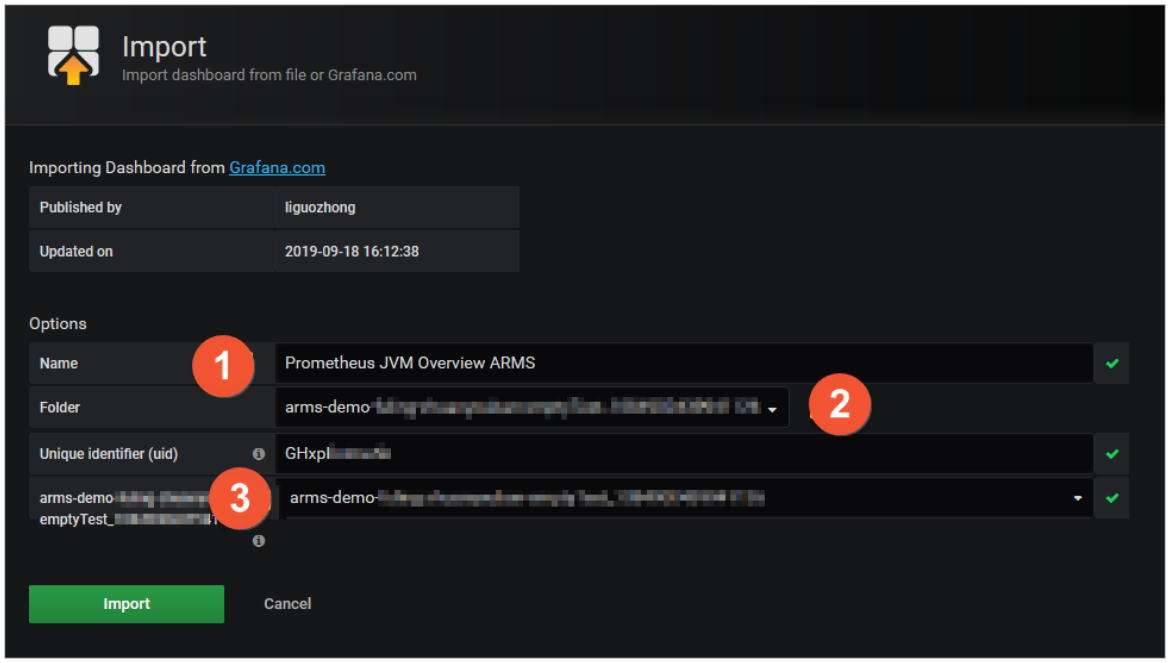
您需要在ARMS控制台导入Grafana大盘模板并指定Prometheus数据源所在的容器服务K8s集群。

具体步骤如下：

1. 打开ARMS Prometheus Grafana大盘概览页。
2. 在左侧导航栏中选择**+ > Import**，并在**Grafana.com Dashboard**文本框输入*10877*，然后单击**Load**。



3. 在Import页面输入以下信息，然后单击Import。

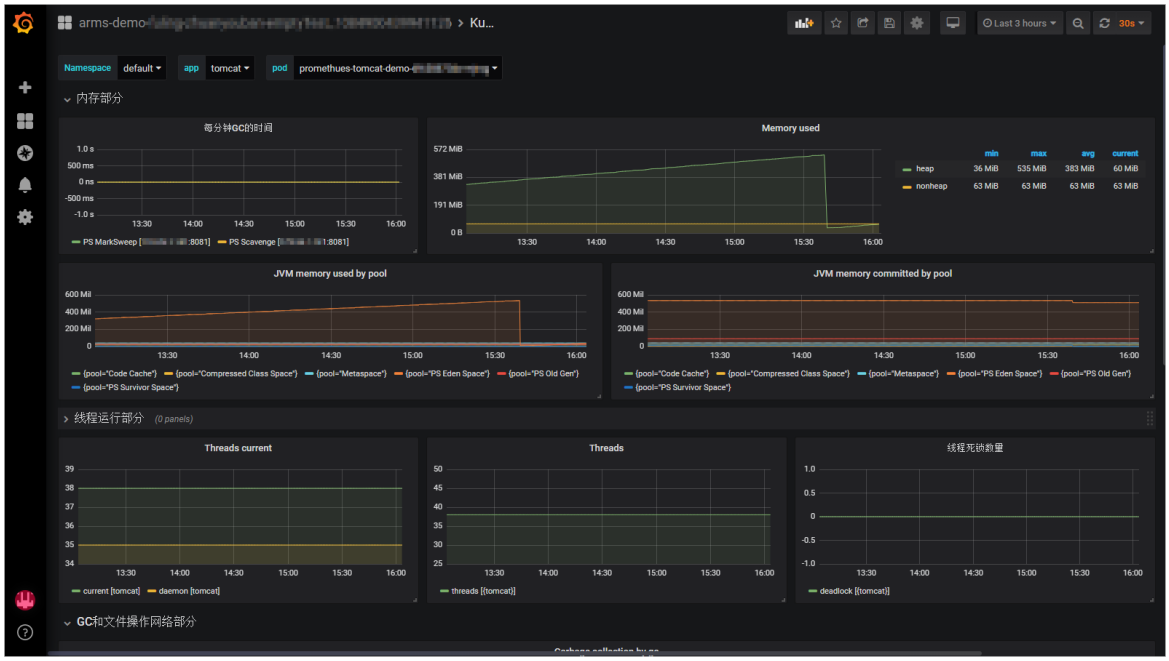


i. 在Name文本框中输入自定义的大盘名称。

ii. 在Folder列表中选择您的阿里云容器服务K8s集群。

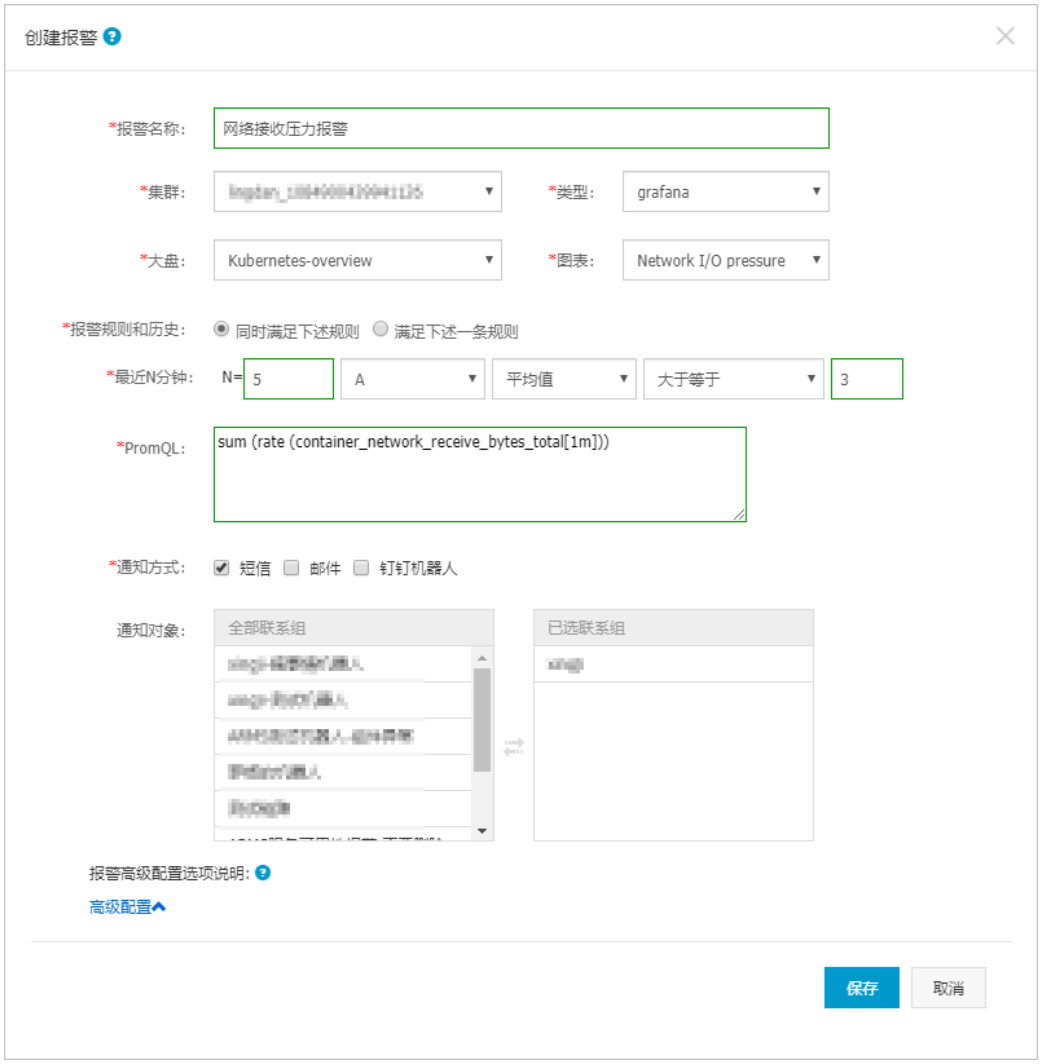
iii. 在Select a Prometheus data source下拉框中选择您的阿里云容器服务K8s集群。

配置完毕后的ARMS Prometheus Grafana JVM大盘如图所示。



步骤五：创建Prometheus监控报警

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节：
   - 在ARMS Prometheus Grafana大盘的New DashBoard页面，单击左侧的 图标，跳转至ARMS Prometheus报警规则和历史页面，在右上角单击创建报警 > Prometheus。
   - 在控制台左侧导航栏中选择报警管理 > 报警策略管理，进入报警规则和历史页面，在右上角单击创建报警 > Prometheus。
2. 在创建报警对话框中输入所有必填信息，完成后单击保存。



在创建报警对话框中配置如下参数：

i. 填写报警名称，例如：网络接收压力报警。
ii. 选择要创建报警的Prometheus监控对应的集群。
iii. 选择类型为grafana。
iv. 选择要监控的具体大盘和图表。

v. 设置报警规则。

a. 选中**同时满足下述规则**。

b. 编辑报警规则，例如：N=5时网络接收数据字节（MB）的平均值大于等于3时则报警。

> 说明 一个Grafana图表中可能有A、B、C等多条曲线数据，您可根据您的需求设置监控其中的一条线。

c. 在**PromQL输入框**中编辑或重新输入PromQL语句。

> 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]))` 。

vi. 选中通知方式，例如：选中短信。

vii. 设置通知对象。在**全部联系组**中单击联系人分组的名称，该联系人分组出现在**已选联系组**中，即设置成功。

相关操作

ARMS Prometheus Grafana JVM大盘配置完毕后，您可以查看Prometheus监控指标和进一步自定义大盘，详情请参见以下文档：

相关文档

- 查看Prometheus监控指标
- 通过ARMS Prometheus自定义Grafana大盘
- 创建报警

4.2.3. 通过ARMS Prometheus监控MySQL

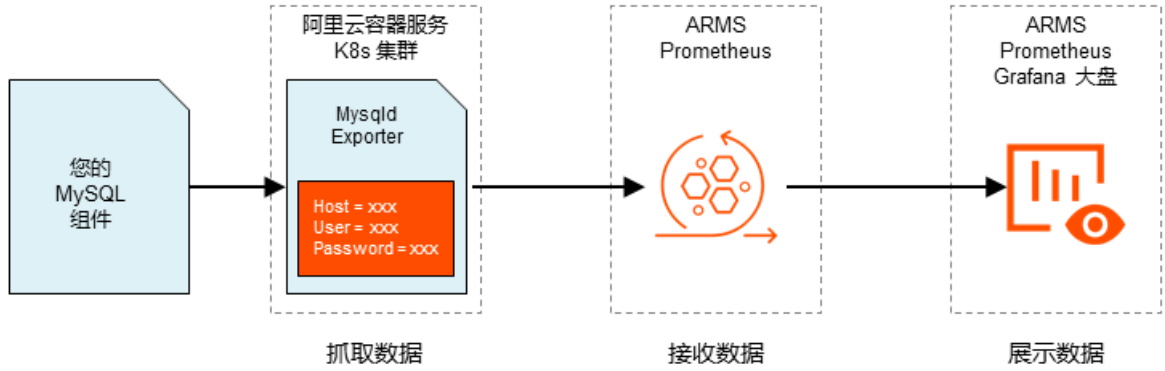
使用ARMS Prometheus监控抓取MySQL数据，并借助ARMS Prometheus Grafana大盘来展示MySQL数据，即可实现通过ARMS Prometheus监控MySQL的目的。

前提条件

- 快速创建Kubernetes托管版集群
- 使用私有镜像仓库创建应用

背景信息

本教程的操作流程如图所示。



步骤一：通过外部应用抓取MySQL数据

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。

1. 登录容器服务Kubernetes版控制台。
2. 容器服务Kubernetes版控制台分为新版与旧版，您可以根据需要自行选择：
   - 新版：在左侧导航栏单击**集群**，在**集群列表**页面上的目标集群右侧**操作**列选择**更多 > Dashboard**。

   - 旧版：在左侧导航栏选择**集群 > 集群**，在**集群列表**页面的目标集群右侧**操作**列选择**更多 > Dashboard**。

3. 在左侧导航栏单击**工作负载**，在页面右上角单击**创建**，并在**使用文本创建**页签填写以下内容。

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

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: mysqld-exporter
spec:
replicas: 1
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
```

此步骤将mysqld-exporter应用部署至容器服务K8s集群中。

4. 在左侧导航栏选择**服务发现与负载均衡 > 服务**，在页面右上角单击**创建**，并在**使用文本创建**页签上填写以下内容。

```
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
```

步骤二：配置ARMS Prometheus监控以接收MySQL数据

接下来需要在ARMS控制台配置ARMS Prometheus监控以接收外部应用所抓取的MySQL数据。

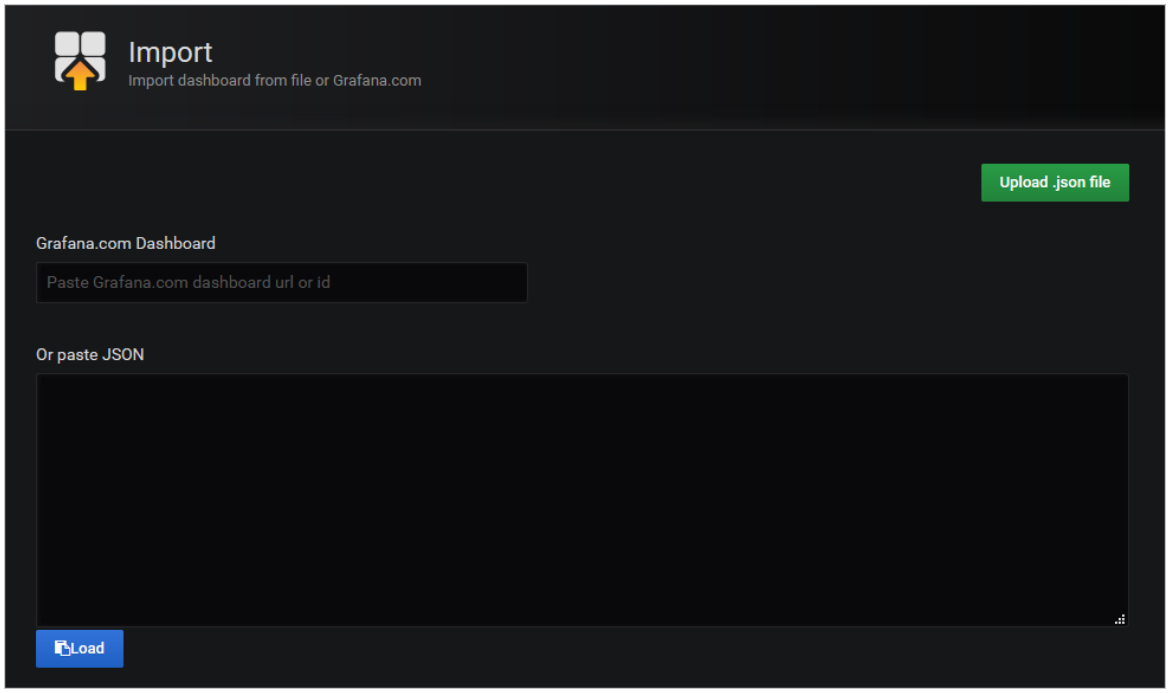
1. 登录ARMS控制台。
2. 在左侧导航栏中单击**Prometheus监控**。
3. 在**Prometheus监控**页面顶部选择容器服务K8s集群所在的地域，并在目标集群右侧的**操作**列单击**安装**。
4. ARMS Prometheus Agent安装完毕后，在目标集群右侧的**操作**列单击**设置**。
5. 在**服务发现**页签单击**添加ServiceMonitor**，在**添加ServiceMonitor**对话框中填写以下内容。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
labels:
app: mysqld-exporter
release: p
# 填写一个唯一名称name: mysqld-exporter
# 填写目标命名空间namespace: default
spec:
endpoints:
- interval: 30s
# Mysqld Grafana模版ID为7362
# 填写service.yaml中Prometheus Exporter对应的Port的Name字段的值port: mysqld-exporter
#填写Prometheus Exporter代码中暴露的地址path: /metrics
namespaceSelector:
any: true
# Nginx Demo的命名空间selector:
matchLabels:
app: mysqld-exporter
```

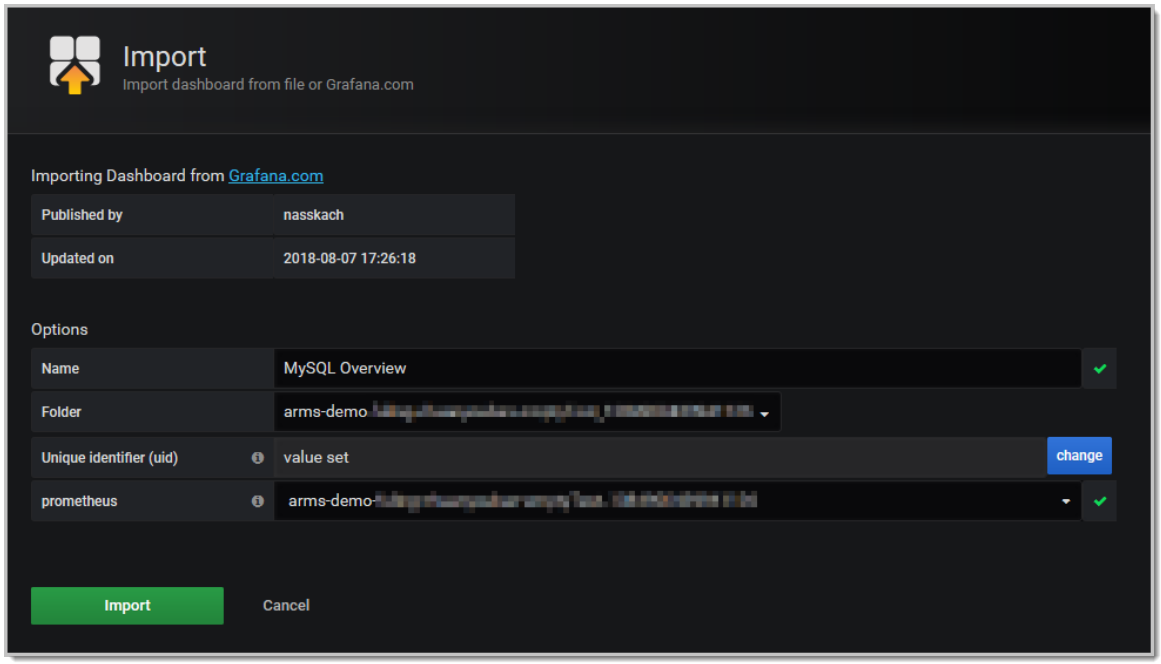
步骤三：通过Grafana大盘展示MySQL数据

最后需要在ARMS控制台导入Grafana大盘模板并指定Prometheus数据源所在的容器服务K8s集群。

1. 打开ARMS Prometheus Grafana大盘概览页。
2. 在左侧导航栏中选择+ > Import，并在Grafana.com Dashboard文本框输入*7362*，然后单击Load。



3. 在Import页面输入以下信息，然后单击Import。

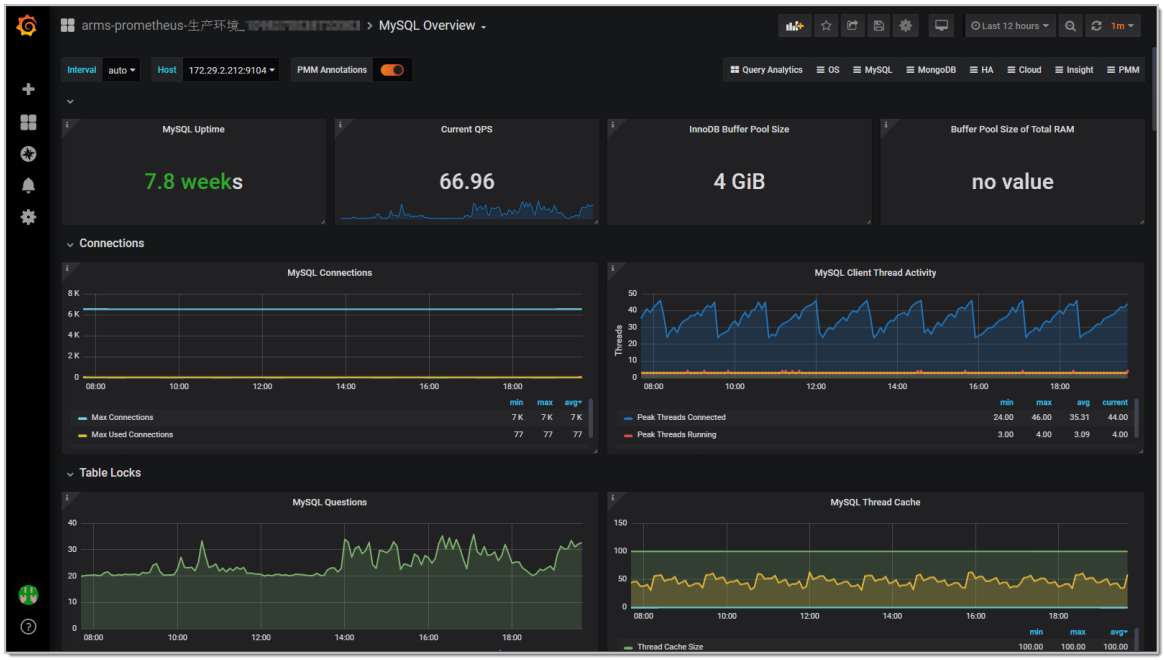


i. 在Name文本框中输入自定义的大盘名称。

ii. 在Folder下拉框中选择您的阿里云容器服务K8s集群。

iii. 在prometheus下拉框中选择您的阿里云容器服务K8s集群。

配置完毕后的ARMS Prometheus Grafana MySQL大盘如图所示。

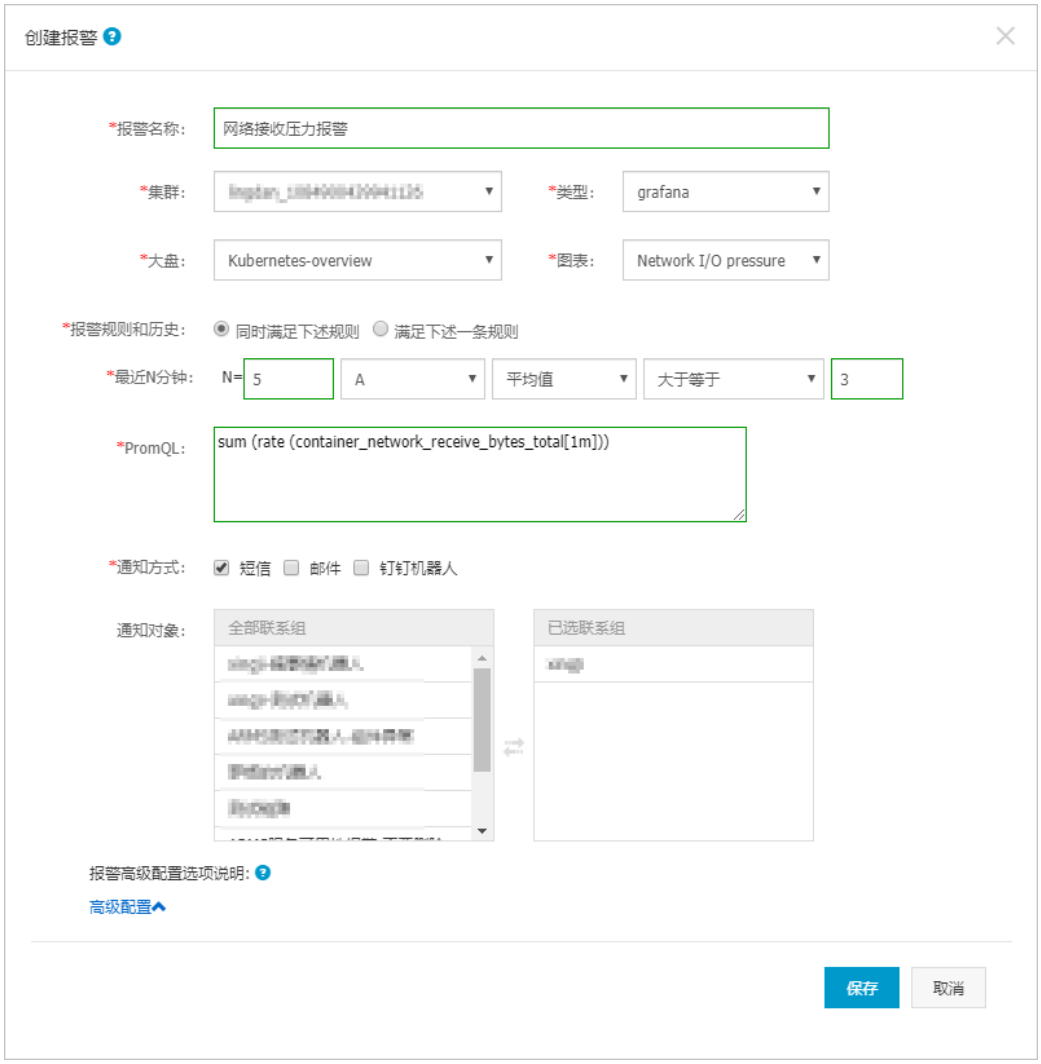


步骤四：创建Prometheus监控报警

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节：
   - 在ARMS Prometheus Grafana大盘的New DashBoard页面，单击左侧的 图标，跳转至ARMS Prometheus报警规则和历史页面，在右上角单击创建报警 > Prometheus。

- 在控制台左侧导航栏中选择**报警管理 > 报警策略管理**，进入**报警规则和历史**页面，在右上角单击**创建报警 > Prometheus**。

2. 在**创建报警**对话框中输入所有必填信息，完成后单击**保存**。



在创建报警对话框中配置如下参数：

i. 填写报警名称，例如：网络接收压力报警。

ii. 选择要创建报警的Prometheus监控对应的集群。

iii. 选择类型为grafana。

iv. 选择要监控的具体大盘和图表。

v. 设置报警规则。

a. 选中同时满足下述规则。

b. 编辑报警规则，例如：N=5时网络接收数据字节（MB）的平均值大于等于3时则报警。

> 说明 一个Grafana图表中可能有A、B、C等多条曲线数据，您可根据您的需求设置监控其中的一条线。

c. 在PromQL输入框中编辑或重新输入PromQL语句。

> 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])) 。

vi. 选中通知方式，例如：选中短信。

vii. 设置通知对象。在**全部联系组**中单击联系人分组的名称，该联系人分组出现在**已选联系组**中，即设置成功。

后续步骤

ARMS Prometheus Grafana MySQL大盘配置完毕后，您可以查看Prometheus监控指标和进一步自定义大盘，详见相关文档。

相关文档

- 查看Prometheus监控指标
- 通过ARMS Prometheus自定义Grafana大盘
- 创建报警

4.2.4. 通过ARMS Prometheus监控Go应用

您可以使用ARMS Prometheus监控部署在阿里云K8s集群中的GO应用的运行情况，在监控大盘上查看多种监控数据，还可配置监控报警。本教程介绍如何通过在Go应用中埋点来暴露应用的数据，使用ARMS Prometheus抓取数据，并借助ARMS Prometheus Grafana大盘来展示数据，最终实现通过ARMS Prometheus监控Go应用的目的。

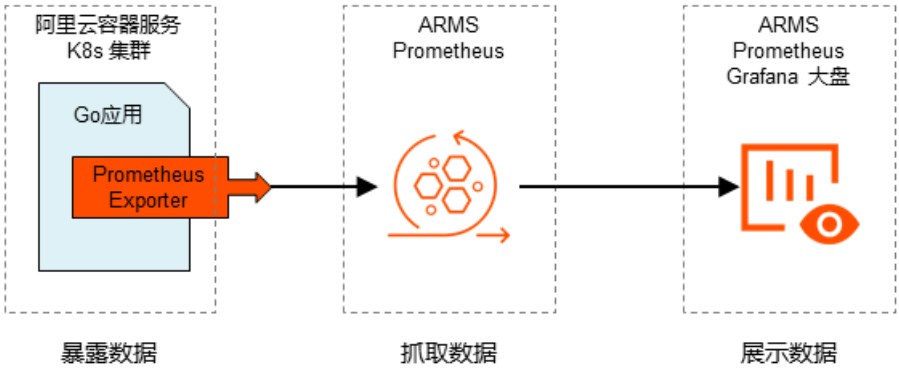
前提条件

在开始本教程前，确保您已经完成了以下操作：

- 下载Demo工程
- 快速创建Kubernetes托管版集群
- 使用私有镜像仓库创建应用

背景信息

本教程的操作流程如图所示。



步骤一：通过埋点暴露Go应用的数据

首先需要在Go应用中使用Prometheus Exporter暴露应用数据。

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：将应用部署至阿里云容器服务K8s集群

其次需要将应用部署至容器服务K8s集群，以便ARMS Prometheus监控抓取应用的数据。

1. 逐行运行*buildDockerImage.sh*中的以下命令。

```
mvn clean install -DskipTests
docker build -t <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> . --no-cache
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>
sudo docker push <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
```

例如：

```
mvn clean install -DskipTests
docker build -t promethues-arms-aliyun-go-demo:v0.1 . --no-cache
sudo docker tag promethues-arms-aliyun-go-demo:v0.1 registry.cn-hangzhou.aliyuncs.com/fuling
/prometheus-arms-aliyun-go-demo-amd64:dev-v0.1
sudo docker push registry.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0.1
```

此步骤构建了名为promethues-arms-aliyun-go-demo的Docker镜像，并将镜像推送至阿里云Docker Registry。

2. 登录容器服务Kubernetes版控制台。
3. 在左侧导航栏选择**集群 > 集群**，在**集群列表**页面上的目标集群右侧**操作**列单击**控制台**。

4. 在左侧导航栏单击**工作负载**，在页面右上角单击**创建**，并在**使用文本创建**区域填写以下内容。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: promethues-arms-aliyun-go-demo
spec:
replicas: 2
template:
metadata:
labels:
app: promethues-arms-aliyun-go-demo
spec:
containers:
- name: promethues-arms-aliyun-go-demo
imagePullPolicy: Always
image: registry.cn-hangzhou.aliyuncs.com/fuling/prometheus-arms-aliyun-go-demo-amd64:dev-v
0.1
ports:
- containerPort: 8077
name: arms-go-demo
```

此步骤将步骤中创建的Docker镜像部署至容器服务K8s集群中。

5. 在左侧导航栏选择**服务发现与负载均衡 > 服务**，在页面右上角单击**创建**，并在**使用文本创建**区域填写以下内容。

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

步骤三：配置ARMS Prometheus监控以抓取Go应用的数据

接下来需要在ARMS控制台配置ARMS Prometheus监控以抓取Go应用的数据。

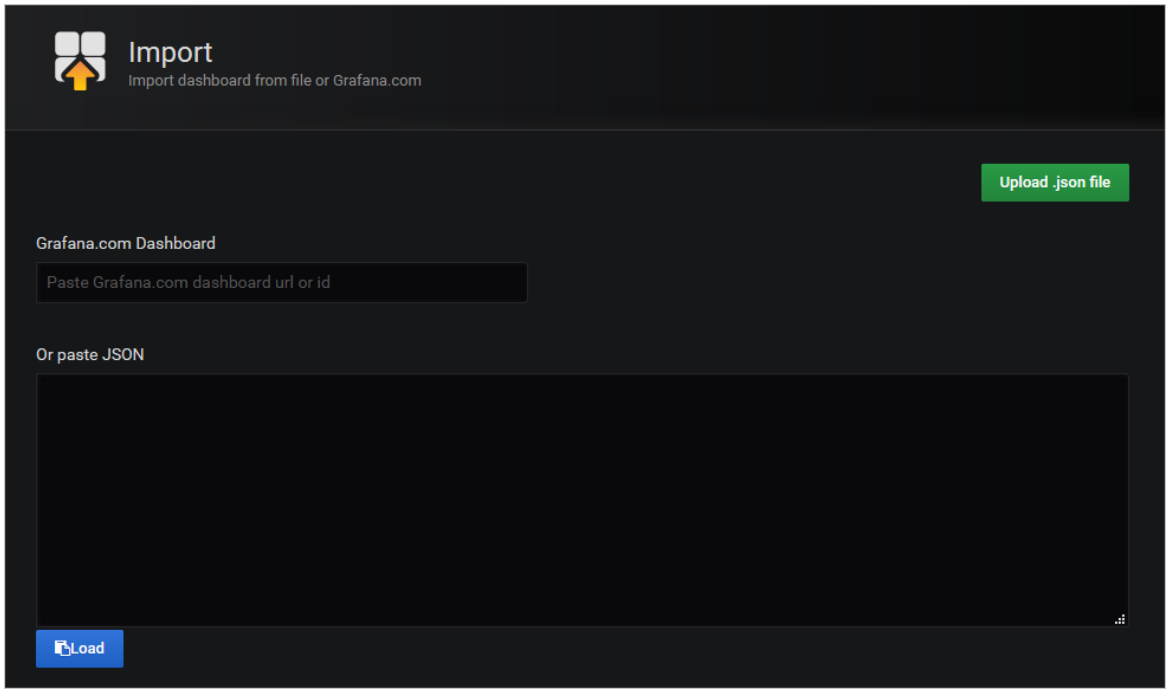
1. 登录ARMS控制台。
2. 在左侧导航栏中单击**Prometheus监控**。
3. 在**Prometheus监控**页面顶部选择容器服务K8s集群所在的地域，并在目标集群右侧的**操作**列单击**安装**。
4. ARMS Prometheus Agent安装完毕后，在目标集群右侧的**操作**列单击**设置**。
5. 在**服务发现**页签上单击**添加ServiceMonitor**，在**添加ServiceMonitor**对话框中填写以下内容，然后单击**确定**。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
# 填写一个唯一名称name: promethues-arms-aliyun-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
# Nginx Demo的命名空间selector:
matchLabels:
app: promethues-arms-aliyun-go-demo
```

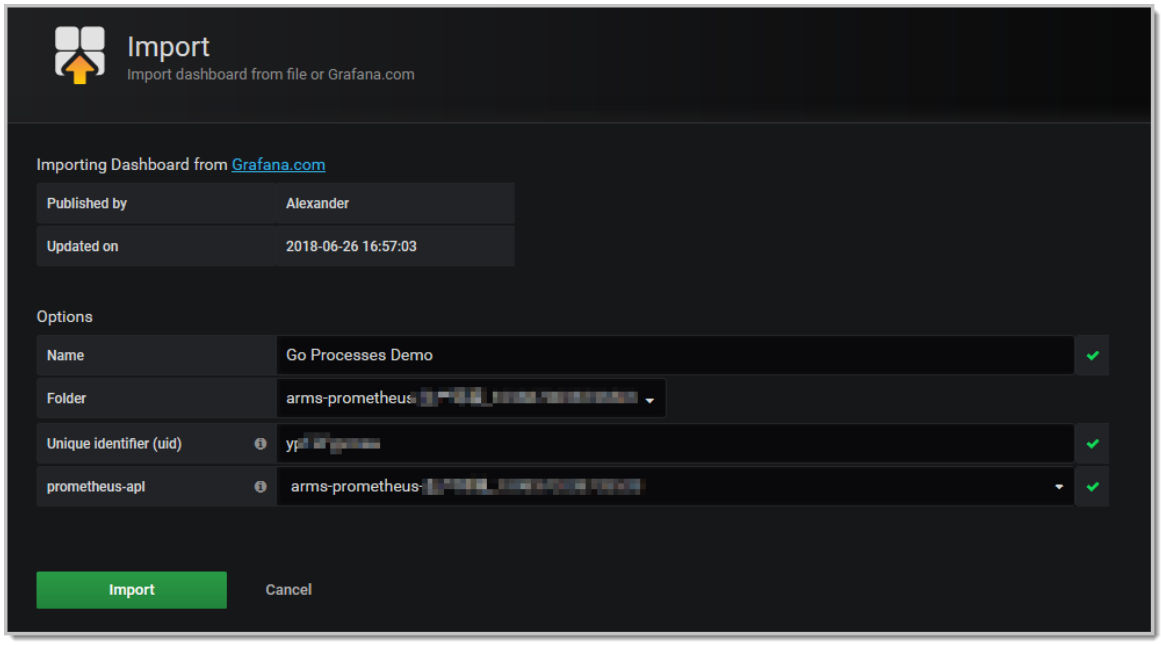
步骤四：通过Grafana大盘展示Go应用的数据

最后需要在ARMS控制台导入Grafana大盘模板并指定Prometheus数据源所在的容器服务K8s集群。

1. 打开ARMS Prometheus Grafana大盘概览页。
2. 在左侧导航栏中选择+ > **Import**，并在**Grafana.com Dashboard**文本框输入*6671*，然后单击**Load**。



3. 在Import页面输入以下信息，然后单击Import。

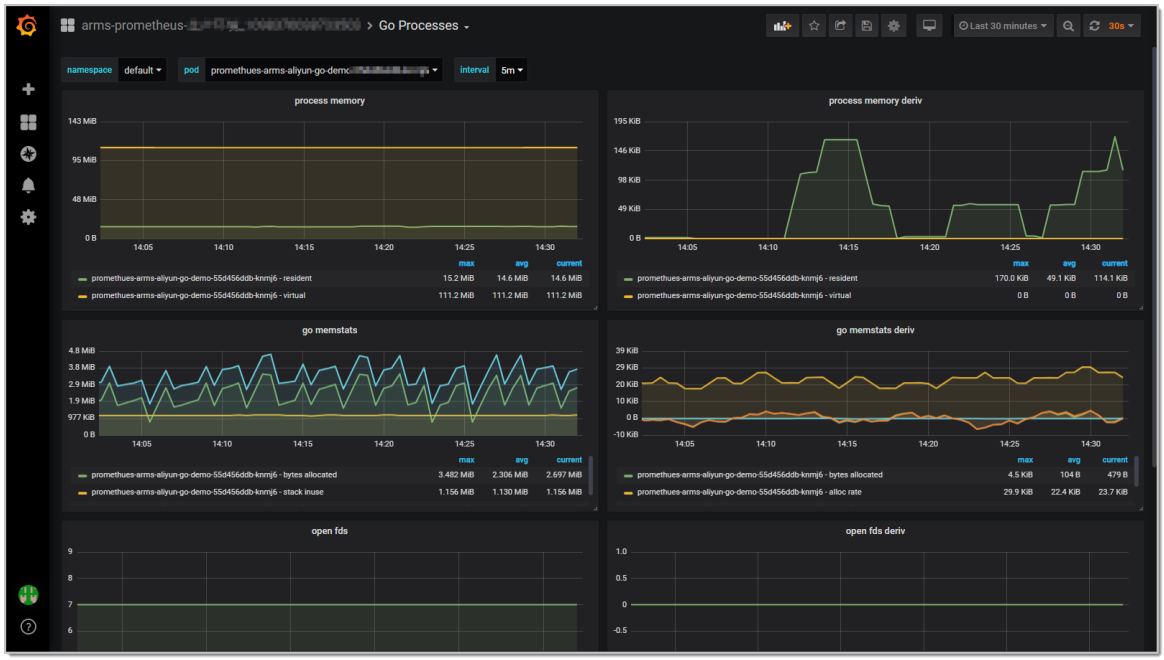


i. 在Name文本框中输入自定义的大盘名称。

ii. 从Folder列表中，选择您的阿里云容器服务K8s集群。

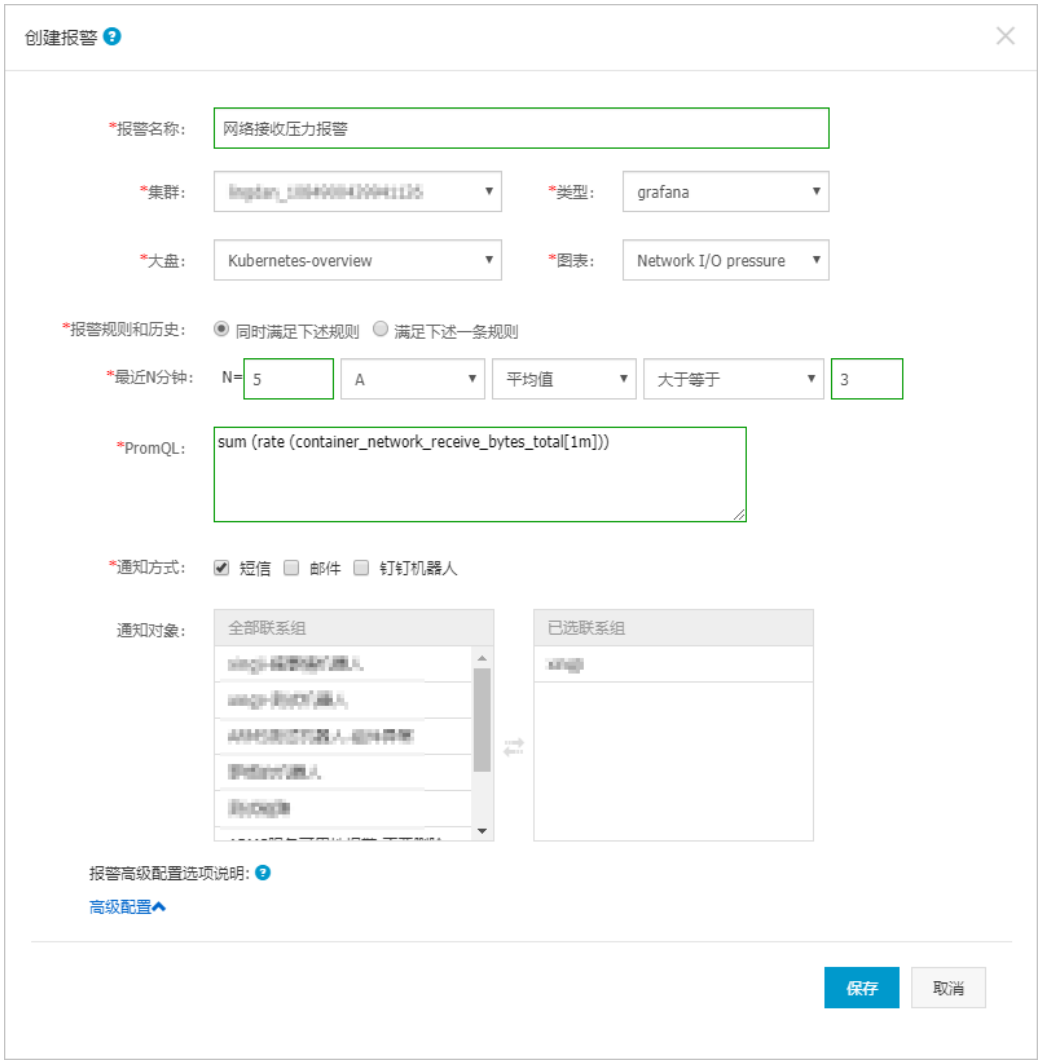
iii. 在prometheus-apl下拉框中选择您的阿里云容器服务K8s集群。

配置完毕后的ARMS Prometheus Grafana Go大盘如图所示。



步骤五：创建Prometheus监控报警

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节：
   - 在ARMS Prometheus Grafana大盘的New DashBoard页面，单击左侧的 图标，跳转至ARMS Prometheus报警规则和历史页面，在右上角单击创建报警 > Prometheus。
   - 在控制台左侧导航栏中选择报警管理 > 报警策略管理，进入报警规则和历史页面，在右上角单击创建报警 > Prometheus。
2. 在创建报警对话框中输入所有必填信息，完成后单击保存。



在创建报警对话框中配置如下参数：

i. 填写报警名称，例如：网络接收压力报警。
ii. 选择要创建报警的Prometheus监控对应的集群。
iii. 选择类型为grafana。
iv. 选择要监控的具体大盘和图表。

v. 设置报警规则。

a. 选中**同时满足下述规则**。

b. 编辑报警规则，例如：N=5时网络接收数据字节（MB）的平均值大于等于3时则报警。

> 说明 一个Grafana图表中可能有A、B、C等多条曲线数据，您可根据您的需求设置监控其中的一条线。

c. 在**PromQL输入框**中编辑或重新输入PromQL语句。

> 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]))` 。

vi. 选中通知方式，例如：选中短信。

vii. 设置通知对象。在**全部联系组**中单击联系人分组的名称，该联系人分组出现在**已选联系组**中，即设置成功。

后续步骤

ARMS Prometheus Grafana Go大盘配置完毕后，您可以查看Prometheus监控指标和进一步自定义大盘，详见以下文档。

查看Prometheus监控指标

通过ARMS Prometheus自定义Grafana大盘

相关文档

- 什么是Prometheus监控？

4.2.5. 通过 ARMS Prometheus 监控 Redis

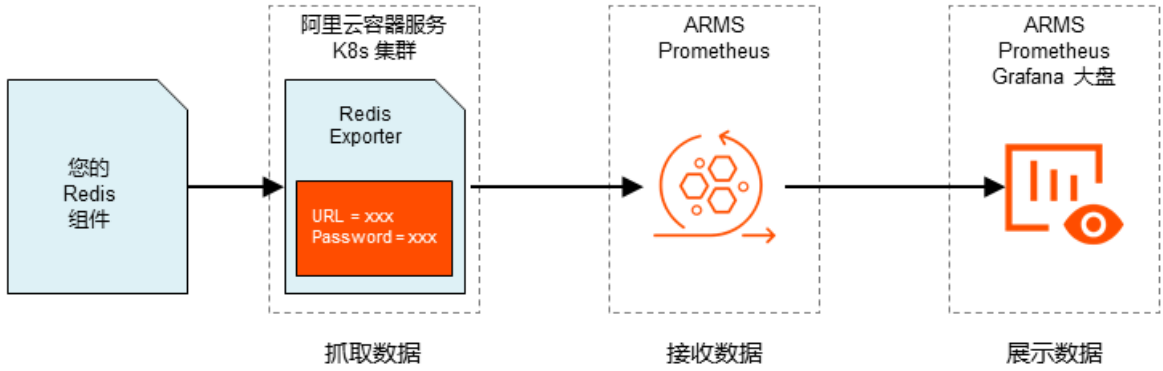
使用 ARMS Prometheus 监控抓取 Redis 数据，并借助 ARMS Prometheus Grafana 大盘来展示 Redis 数据，即可实现通过 ARMS Prometheus 监控 Redis 的目的。

前提条件

- 快速创建Kubernetes托管版集群
- 使用私有镜像仓库创建应用

背景信息

本教程的操作流程如图所示。



步骤一：通过外部应用抓取 Redis 数据

将 redis-exporter 镜像部署至容器服务 K8s 集群，以便抓取 Redis 数据。

1. 登录容器服务Kubernetes版控制台。
2. 在左侧导航栏选择**集群 > 集群**，在**集群列表**页面上的目标集群右侧**操作**列单击**控制台**。

3. 在左侧导航栏选择**工作负载 > 部署**，在页面右上角单击**创建**，并在**使用文本创建**页签上填写以下内容。

> **说明** 请将 *<地址>*和*<密码>*替换为 Redis 的对应值。您也可以使用 ARMS 提供的示例值体验：
>
> - Redis 地址：*redis://r-bp167pgqkqh7h25coypd.redis.rds.aliyuncs.com:6379*
> - Redis 密码：`Arms1234`

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: redis-exporter
spec:
replicas: 1
template:
metadata:
labels:
app: redis-exporter
spec:
containers:
- name: redis-exporter
imagePullPolicy: Always
env:
- name: REDIS_ADDR
value: "<地址>"
- name: REDIS_PASSWORD
value: "<密码>"
- name: REDIS_EXPORTER_DEBUG
value: "1"
image: oliver006/redis_exporter
ports:
- containerPort: 9121
name: redis-exporter
```

此步骤将 redis-exporter 应用部署至容器服务 K8s 集群中。

4. 在左侧导航栏选择**服务发现与负载均衡 > 服务**，在页面右上角单击**创建**，并在**使用文本创建**页签上填写以下内容。

```
apiVersion: v1
kind: Service
metadata:
labels:
app: redis-exporter
name: redis-exporter
spec:
ports:
- name: redis-exporter
port: 9121
protocol: TCP
targetPort: 9121
type: NodePort
selector:
app: redis-exporter
```

步骤二：配置 ARMS Prometheus 监控以接收 Redis 数据

接下来需要在 ARMS 控制台配置 ARMS Prometheus 监控以接收外部应用所抓取的 Redis 数据。

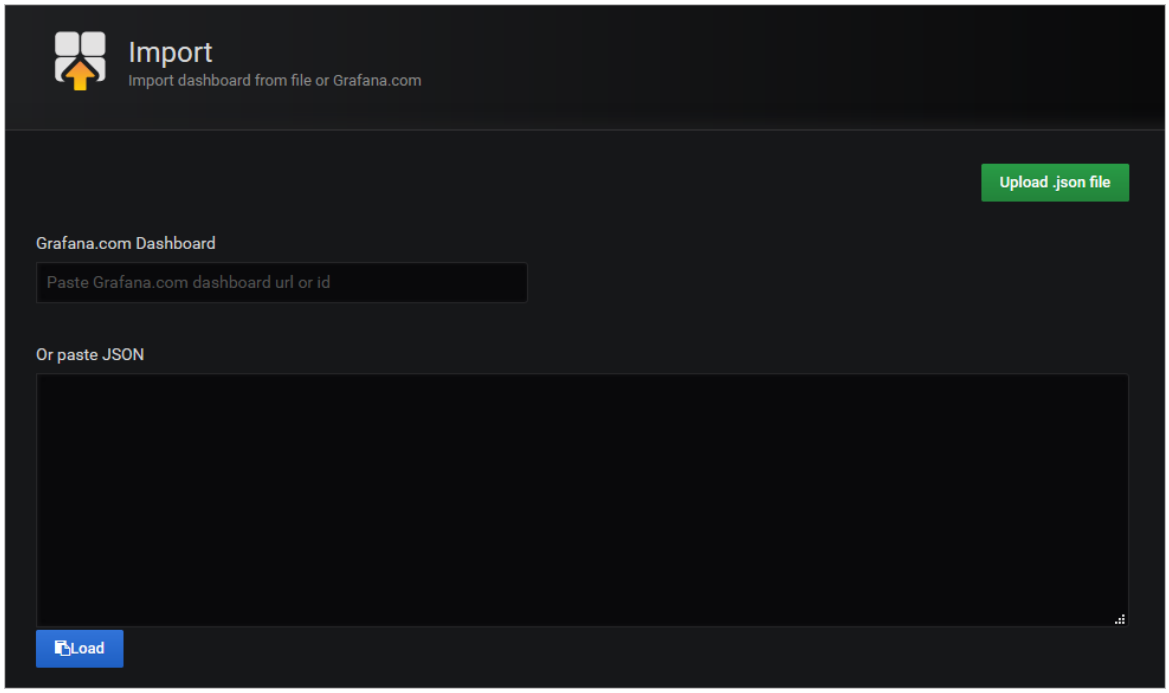
1. 登录ARMS控制台。
2. 在左侧导航栏中单击 **Prometheus监控**。
3. 在 **Prometheus监控**页面顶部选择容器服务 K8s 集群所在的地域，并在目标集群右侧的**操作**列单击**安装**。
4. ARMS Prometheus Agent 安装完毕后，在目标集群右侧的**操作**列单击**设置**。
5. 在**配置详情**页签上单击**添加ServiceMonitor**，在**新增ServiceMonitor** 对话框中填写以下内容。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
# 填写一个唯一名称
name: redis-exporter
# 填写目标命名空间
namespace: default
spec:
endpoints:
- interval: 30s
# Redis Grafana 模版 ID 为 763
# 填写 service.yaml 中 prometheus exporter 对应的 Port 的 Name 字段的值
port: redis-exporter
# 填写 Prometheus Exporter 代码中暴露的地址
path: /metrics
namespaceSelector:
any: true
# Nginx Demo 的命名空间
selector:
matchLabels:
# 填写 service.yaml 的 Label 字段以定位目标 service.yaml
app: redis-exporter
```

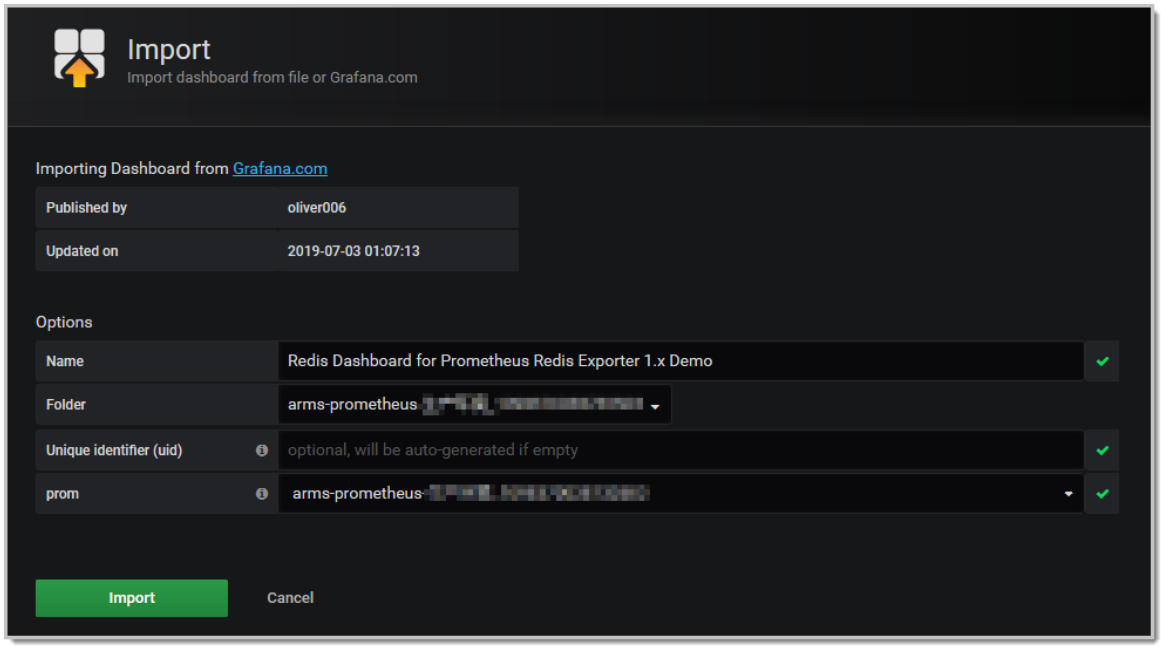
步骤三：通过 Grafana 大盘展示 Redis 数据

最后需要在 ARMS 控制台导入 Grafana 大盘模板并指定 Prometheus 数据源所在的容器服务 K8s 集群。

1. 打开 ARMS Prometheus Grafana 大盘概览页。
2. 在左侧导航栏中选择 + > **Import**，并在 **Grafana.com Dashboard** 文本框输入 *763*，然后单击 **Load**。



3. 在 Import 页面输入以下信息，然后单击 Import。

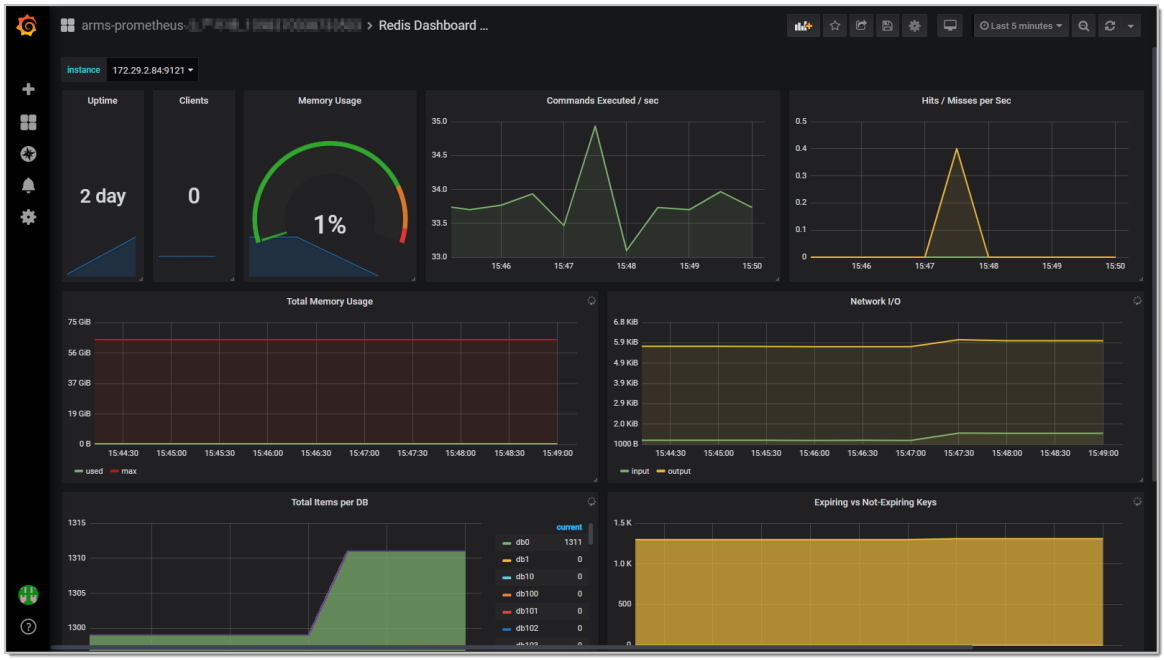


i. 在 Name 文本框中输入自定义的大盘名称。

ii. 在 Folder 下拉框中选择您的阿里云容器服务 K8s 集群。

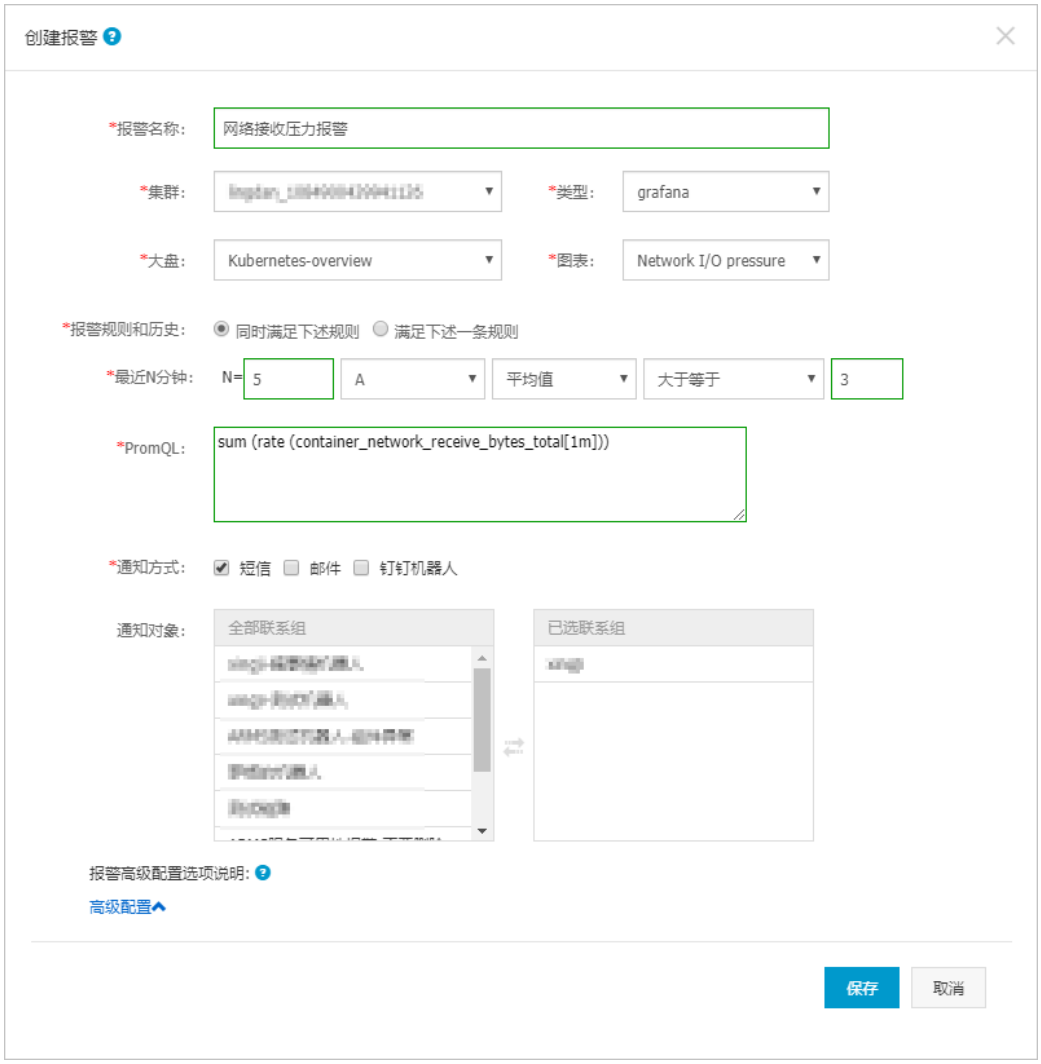
iii. 在 prom 下拉框中选择您的阿里云容器服务 K8s 集群。

配置完毕后的 ARMS Prometheus Grafana Redis 大盘如图所示。



步骤四：创建 Prometheus 监控报警

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节：
   - 在ARMS Prometheus Grafana大盘的New DashBoard页面，单击左侧的 图标，跳转至ARMS Prometheus报警规则和历史页面，在右上角单击创建报警 > Prometheus。
   - 在控制台左侧导航栏中选择报警管理 > 报警策略管理，进入报警规则和历史页面，在右上角单击创建报警 > Prometheus。
2. 在创建报警对话框中输入所有必填信息，完成后单击保存。



在创建报警对话框中配置如下参数：

i. 填写报警名称，例如：网络接收压力报警。
ii. 选择要创建报警的Prometheus监控对应的集群。
iii. 选择类型为grafana。
iv. 选择要监控的具体大盘和图表。

v. 设置报警规则。

a. 选中**同时满足下述规则**。

b. 编辑报警规则，例如：N=5时网络接收数据字节（MB）的平均值大于等于3时则报警。

> 说明 一个Grafana图表中可能有A、B、C等多条曲线数据，您可根据您的需求设置监控其中的一条线。

c. 在**PromQL输入框**中编辑或重新输入PromQL语句。

> 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]))` 。

vi. 选中通知方式，例如：选中短信。

vii. 设置通知对象。在**全部联系组**中单击联系人分组的名称，该联系人分组出现在**已选联系组**中，即设置成功。

后续步骤

ARMS Prometheus Grafana Redis 大盘配置完毕后，您可以查看 Prometheus 监控指标和进一步自定义大盘，详见相关文档。

相关文档

- 查看Prometheus监控指标
- 通过ARMS Prometheus自定义Grafana大盘
- 创建报警

4.2.6. 通过ARMS Prometheus自定义Grafana大盘

ARMS Prometheus监控可以通过ARMS Prometheus 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。本文主要介绍自定义Grafana大盘来展示监控数据的方式。

前提条件

快速创建Kubernetes托管版集群

步骤一：将应用部署至阿里云容器服务K8s集群

首先需要将应用部署至容器服务K8s集群，以便ARMS Prometheus监控抓取JVM数据。

1. 逐行运行*buildDockerImage.sh*中的以下命令。

```
mvn clean install -DskipTests
docker build -t <本地临时Docker镜像名称>:<本地临时Docker镜像版本号> . --no-cache
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>
sudo docker push <Registry域名>/<命名空间>/<镜像名称>:<镜像版本号>
```

例如：

```
mvn clean install -DskipTests
docker build -t promethues-demo:v0 . --no-cache
sudo docker tag promethues-demo:v0 registry.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0
sudo docker push registry.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0
```

此步骤构建了名为promethues-demo的Docker镜像，并将镜像推送至阿里云Docker Registry。

2. 登录容器服务Kubernetes版控制台。
3. 在左侧导航栏选择**集群 > 集群**，在**集群列表**页面上的目标集群右侧**操作**列单击**控制台**。

4. 在左侧导航栏选择**工作负载 > 部署**，在页面右上角单击**创建**，并在**使用文本创建**页签上填写以下示例内容。

> **说明** 以下配置文件中的prometheus.io/port和prometheus.io/path的值分别为应用中暴露的Prometheus监控端口和路径。

```
apiVersion: extensions/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.cn-hangzhou.aliyuncs.com/fuling/promethues-demo:v0
ports:
- containerPort: 8080
name: tomcat-normal
- containerPort: 8081
name: tomcat-monitor
```

此步骤将promethues-demo这个Docker镜像部署至容器服务K8s集群中。

5. 在左侧导航栏选择**服务发现与负载均衡 > 服务**，在页面右上角单击**创建**，并在**使用文本创建**页签上填写以下内容。

```
apiVersion: v1
kind: Service
metadata:
labels:
name: 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
```

步骤二：为应用安装ARMS Prometheus Agent

1. 登录ARMS控制台。
2. 在左侧导航栏中单击**Prometheus监控**。
3. 在**Prometheus监控**页面顶部选择容器服务K8s集群所在的地域，并在目标集群右侧的**操作**列单击**安装**。

> **说明** 如果安装失败，请多次尝试安装直至成功。如果多次尝试后依然失败，请联系我们的钉钉账号 `23148410` 进行解决。

步骤三：为应用配置Prometheus监控采集规则

ARMS Prometheus Agent安装完成后，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，那么需要为应用配置Prometheus监控采集规则。

1. 在目标集群右侧的**操作**列单击**设置**。
2. 为应用配置Prometheus监控采集规则分为以下两种情况。
   - 如果您需要监控部署在K8s集群内的应用的业务数据，例如订单信息，可以在**服务发现**页签上单击**添加ServiceMonitor**，并在**新增ServiceMonitor**对话框中参考以下示例内容进行填写。

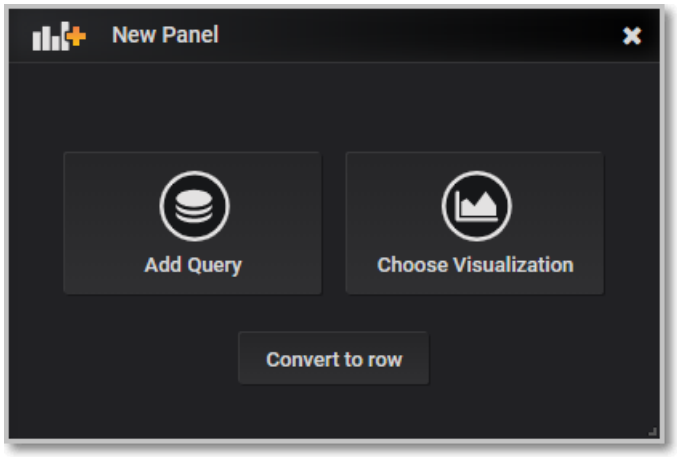
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
# 填写一个唯一名称
name: tomcat-demo
# 填写目标命名空间
namespace: default
spec:
endpoints:
- interval: 30s
# 填写Prometheus Exporter对应的Port的Name字段的值
port: tomcat-monitor
# 填写Prometheus Exporter对应的Path的值
path: /prometheus-metrics
namespaceSelector: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

- 如果您需要监控部署在K8s集群之外的业务数据，如Redis连接数时，可以在**配置详情**页签上单击**编辑prometheus.yaml**配置原生的prometheus.yaml文件，并在**编辑prometheus.yaml**对话框中参考以下示例内容进行填写。

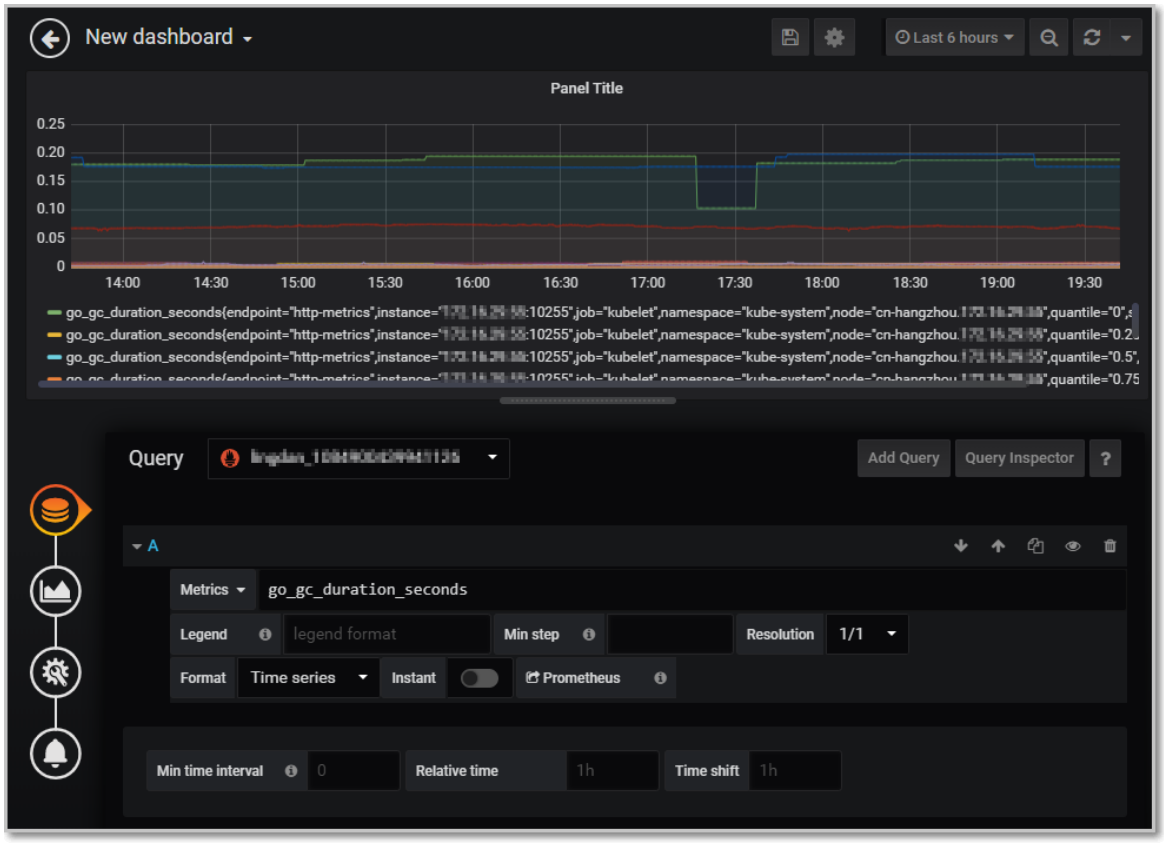
```
global:
scrape_interval: 15s
evaluation_interval: 15s
scrape_configs:
- job_name: 'prometheus'
static_configs:
- targets: ['localhost:9090']
```

步骤四：自定义创建Grafana大盘

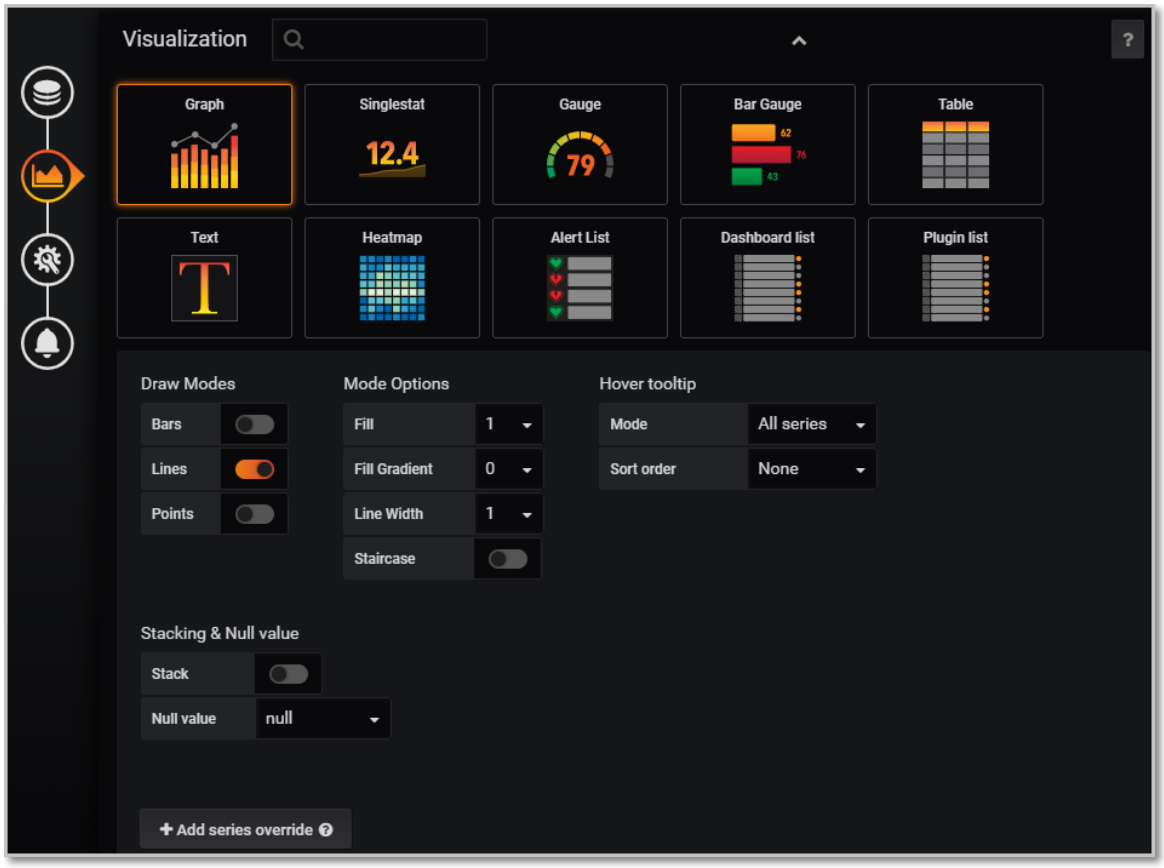
1. 打开ARMS Prometheus Grafana大盘概览页。
2. 在左侧导航栏中选择+ > **Dashboard**，并在**New Panel**区域框中单击**Add Query**。



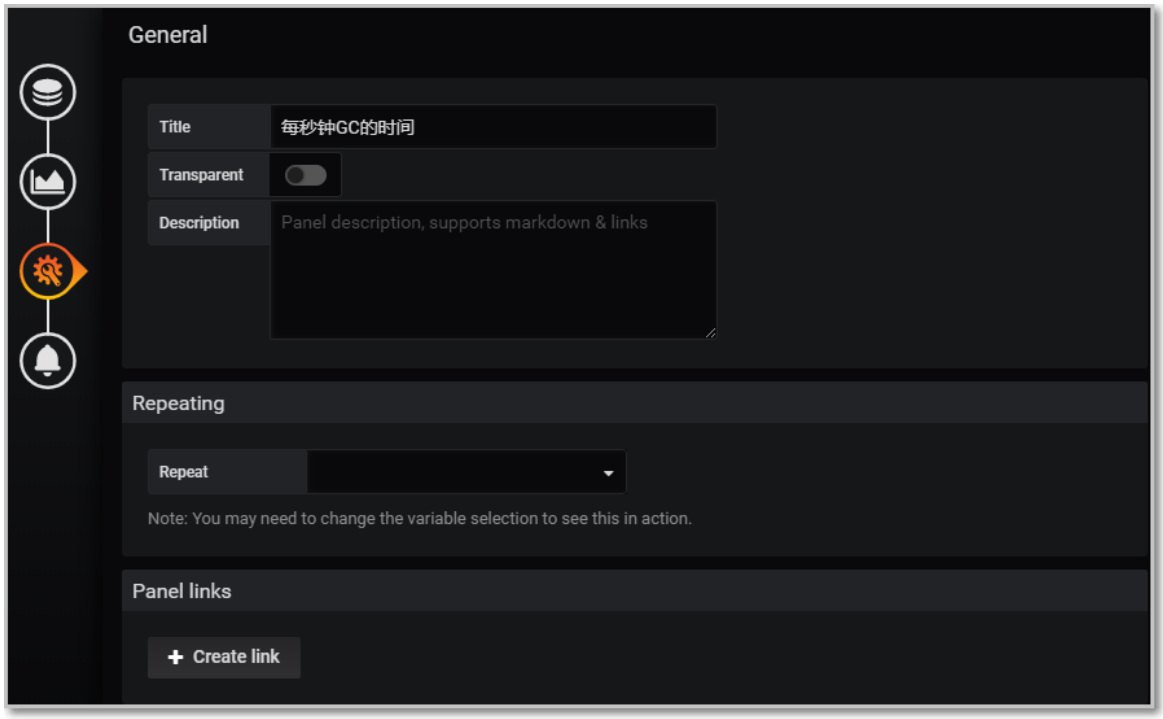
3. 在Query右侧的下拉列表中选择集群。在A折叠面板的Metrics下拉列表中选择监控指标，例如：go_gc_duration_seconds。



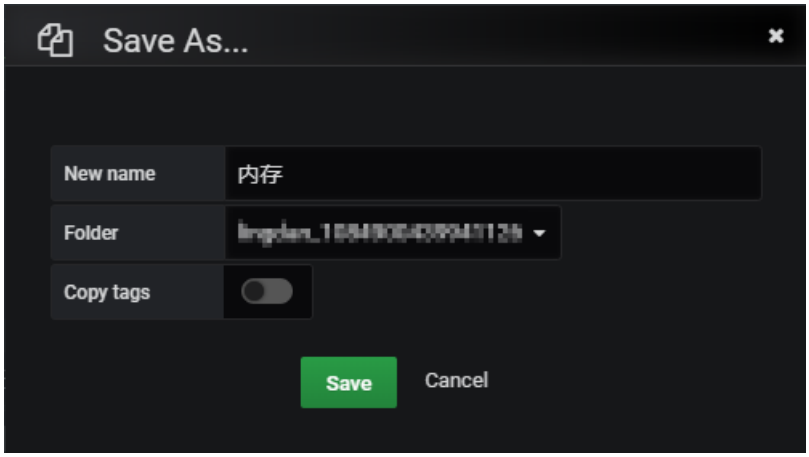
4. 单击页面左侧的图表图标，选择大盘的可视化类型，如图形、表格、热点图等，并根据您的需求配置其他参数。



5. 单击页面左侧的设置图标，填写图表名称。



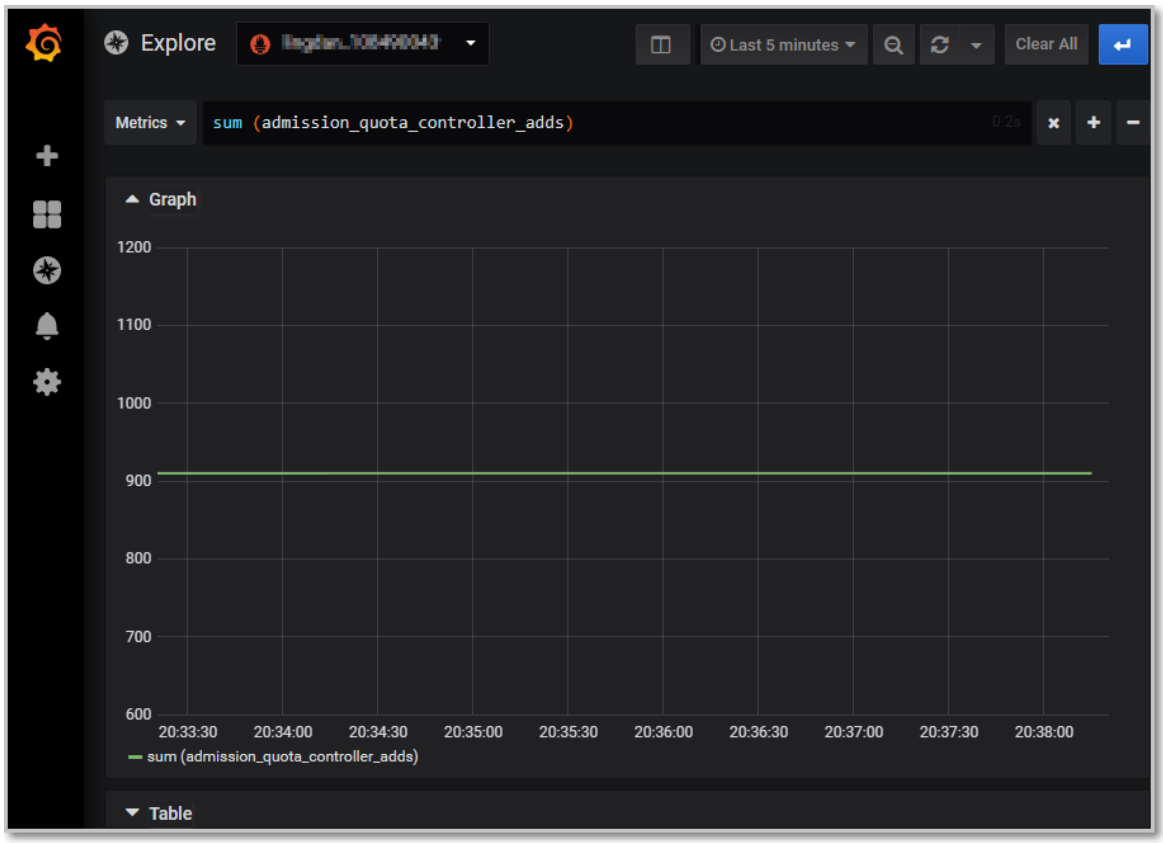
6. 单击页面左侧的铃铛图标，在Alert区域单击Create Alert配置告警。后续告警配置页面会跳转至ARMS Prometheus监控告警配置页面，详情请参见创建Prometheus监控报警。
7. 单击右上角的保存图标，在Save As...对话框中输入大盘名称，并选择集群，然后单击Save保存大盘和图表。您可根据需要自行创建多个大盘和图表。



步骤五：进行数据调试监控复杂指标

如果需要监控涉及复杂运算的指标，您需要在ARMS Prometheus监控中进行数据调试，从而得到相应的PromQL语句。

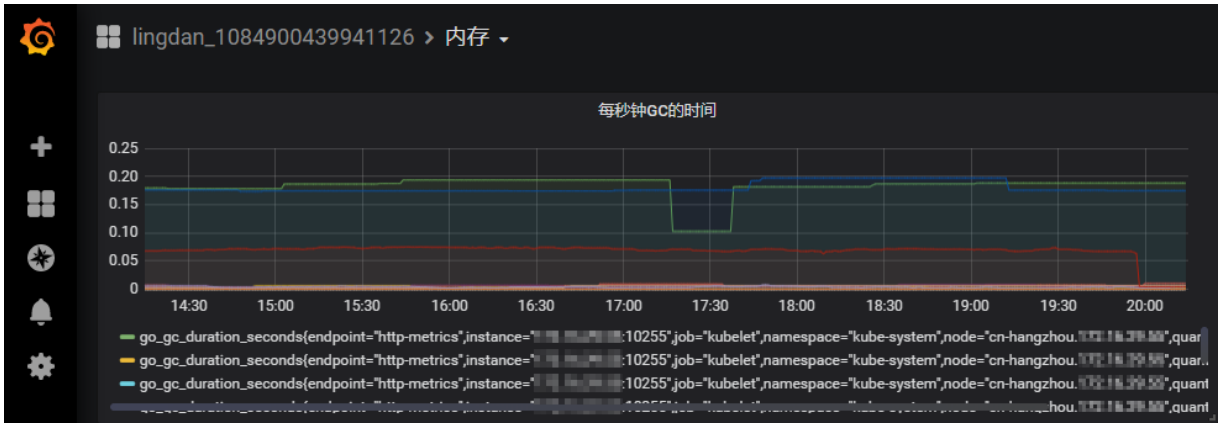
1. 在**Prometheus监控**页面，单击目标集群右侧的**操作**列的**设置**。
2. 单击**数据调试**，跳转至ARMS Prometheus Grafana的**Explore**页面，您可以在**Metrics**输入框中输入PromQL语句进行调试。



3. 调试成功后，您可以参考上述步骤继续添加大盘或图表，详情请参见步骤四：自定义创建Grafana大盘。

执行结果

配置完毕后的ARMS Prometheus Grafana大盘如图所示。



相关文档

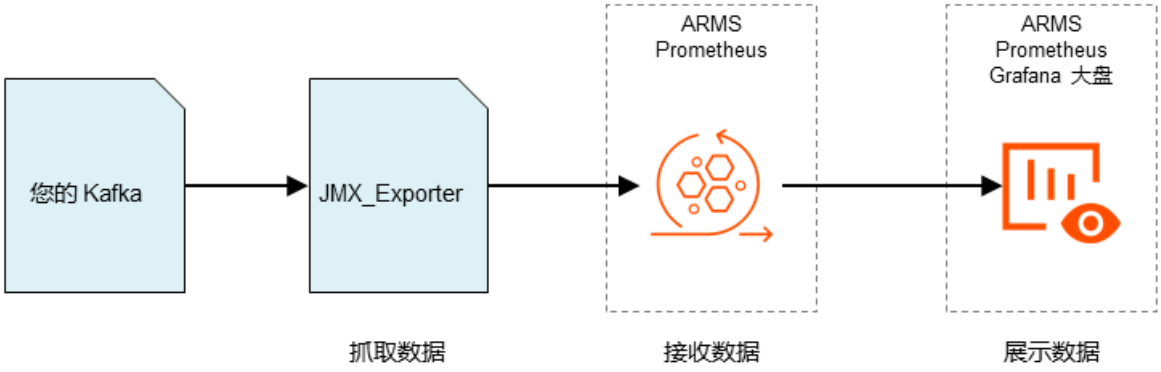
- 查看Prometheus监控指标
- 通过ARMS Prometheus监控JVM

4.2.7. 通过 ARMS Prometheus 监控 Kafka 应用

通过在 Kafka 应用中埋点来暴露应用的数据，使用 ARMS Prometheus 监控抓取数据，并借助 ARMS Prometheus Grafana 大盘来展示数据，即可实现通过 ARMS Prometheus 监控 Kafka 应用的目的。

背景信息

本教程的操作流程如图所示。



步骤一：启动 JMX 服务

在 Kafka 应用中启用 JMX 服务以获取资源信息。

1. 修改 */opt/kafka/kafka_2.11-0.8.2.1/bin/kafka-server-start.sh*，在第一行添加 `export JMX_PORT="9999"` 。
2. 重启 Kafka。

步骤二：启动 jmx_exporter

启动 jmx_exporter，让 JMX 信息可以直接通过 HTTP 方式访问，以便 ARMS Prometheus 监控抓取。

1. 下载 kafka.yml 到 */opt/exporter_kafka/* 目录下。
2. 修改下载的 */opt/exporter_kafka/kafka-0.8.2.yml* 文件。

   在第一行添加 `hostPort: localhost:9999` ，将 JMX 服务运行的端口暴露给 jmx_exporter。
3. 下载 jmx_exporter 的可执行文件到 */opt/exporter_kafka/* 目录下。
4. 启动 jmx_exporter。

   ```
   java -Dcom.sun.management.jmxremote.ssl=false -
   Dcom.sun.management.jmxremote.authenticate=false -
   Dcom.sun.management.jmxremote.port=9998 -cp
   /opt/exporter_kafka/jmx_prometheus_httpserver-0.12.0-jar-with-dependencies.jar
   io.prometheus.jmx.WebServer 9997 /opt/exporter_kafka/kafka-0-8-2.yml &
   ```

   至此应用部分配置完成。可通过运行以下命令验证 jmx_exporter 是否可以正常工作。

   ```
   curl http://<jmx_exporter所在机器的IP>:9997/metrics
   ```

   Kafka 各指标的含义请参见 Monitoring Kafka。

步骤三：配置 ARMS Prometheus 监控以抓取 Kafka 应用的数据

在 ARMS 控制台配置 ARMS Prometheus 监控以抓取 Kafka 应用的数据。

1. 登录ARMS控制台。
2. 在左侧导航栏中单击 **Prometheus监控**。
3. 在 **Prometheus监控**页面顶部选择容器服务 K8s 集群所在的地域，然后单击目标集群的名称。
4. 在接下来的页面单击**配置详情**页签，然后单击**编辑prometheus.yaml**。

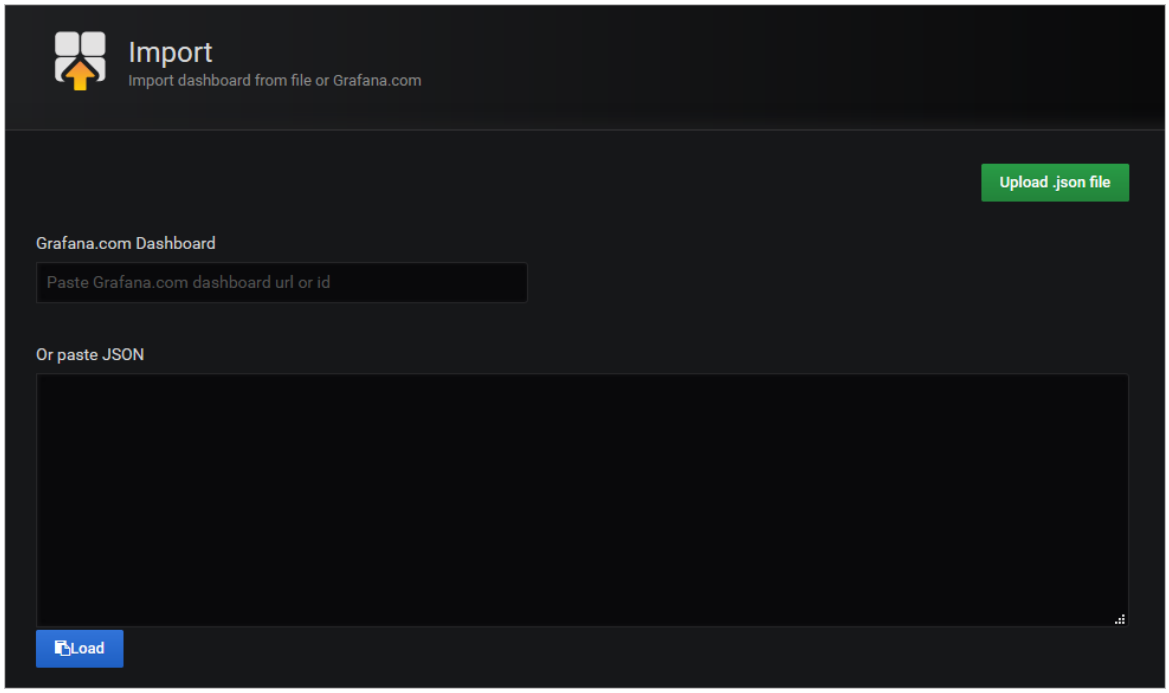
5. 粘贴以下代码。

```
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 every 1 minute.
scrape_configs:
- job_name: 'kafka'
static_configs:
- targets: ['121.40.124.46:9997']
```

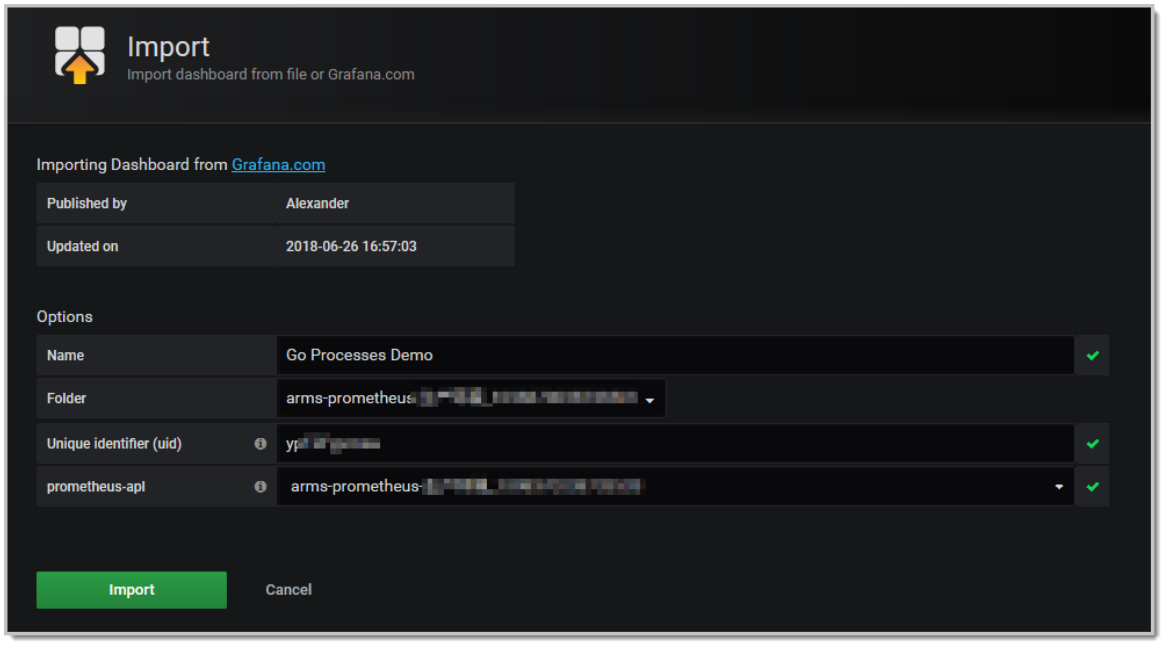
步骤四：通过 Grafana 大盘展示 Kafka 应用的数据

在 ARMS 控制台导入 Grafana 大盘模板并指定 Prometheus 数据源所在的容器服务 K8s 集群。

1. 打开 ARMS Prometheus Grafana 大盘概览页。
2. 在左侧导航栏中选择 + > **Import**，并在 **Grafana.com Dashboard** 文本框输入 *10973*，然后单击 **Load**。



3. 在 Import 页面输入以下信息，然后单击 Import。

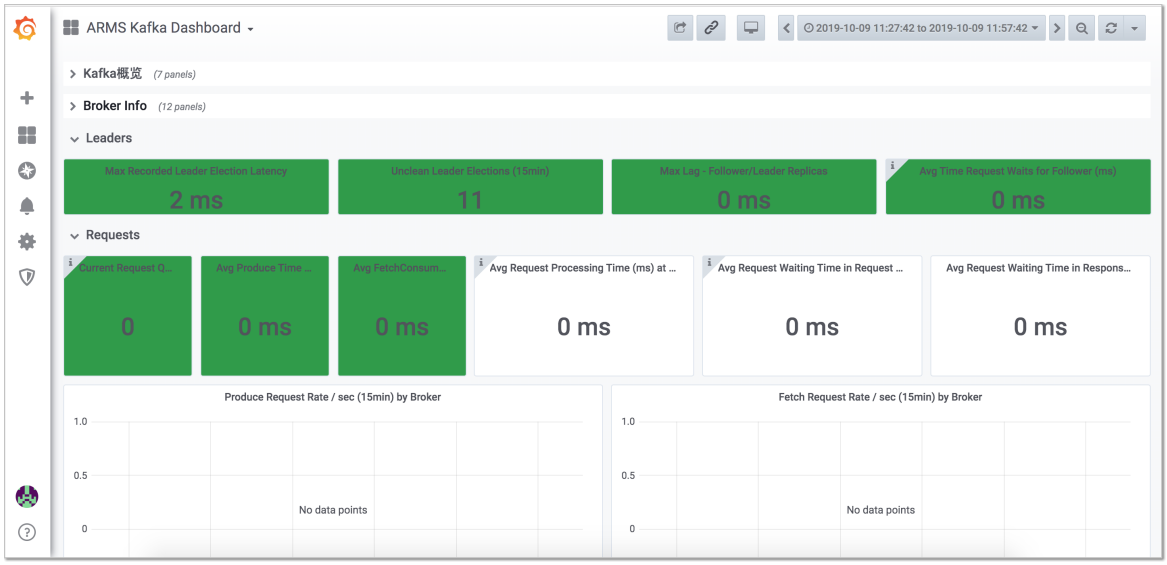


i. 在 Name 文本框中输入自定义的大盘名称。

ii. 在 Folder 下拉框中选择您的阿里云容器服务 K8s 集群。

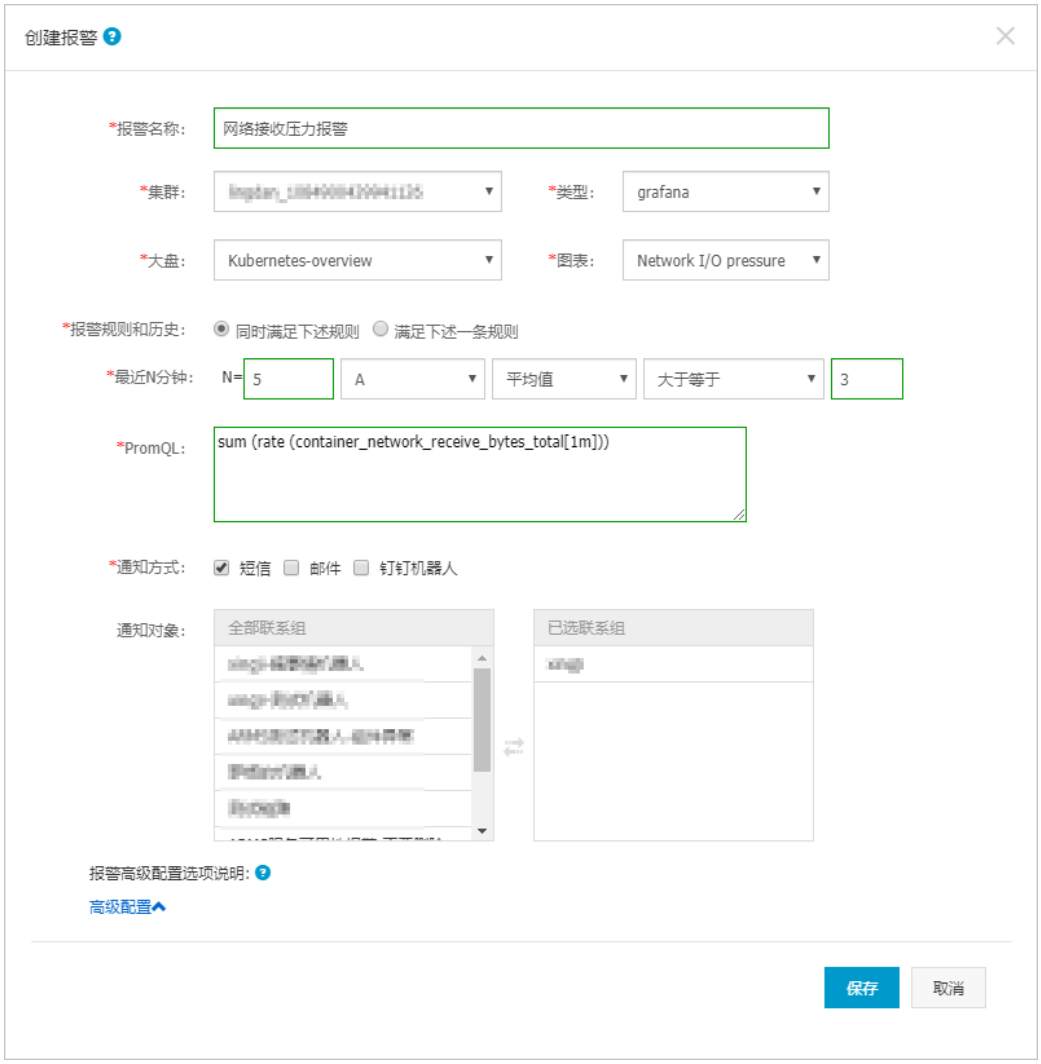
iii. 在 prometheus-apl 下拉框中选择您的阿里云容器服务 K8s 集群。

配置完毕后的 ARMS Prometheus Grafana Kafka 大盘如图所示。



步骤五：创建 Prometheus 监控报警

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节：
   - 在ARMS Prometheus Grafana大盘的New DashBoard页面，单击左侧的 图标，跳转至ARMS Prometheus报警规则和历史页面，在右上角单击创建报警 > Prometheus。
   - 在控制台左侧导航栏中选择报警管理 > 报警策略管理，进入报警规则和历史页面，在右上角单击创建报警 > Prometheus。
2. 在创建报警对话框中输入所有必填信息，完成后单击保存。



在创建报警对话框中配置如下参数：

i. 填写报警名称，例如：网络接收压力报警。
ii. 选择要创建报警的Prometheus监控对应的集群。
iii. 选择类型为grafana。
iv. 选择要监控的具体大盘和图表。

v. 设置报警规则。

a. 选中**同时满足下述规则**。

b. 编辑报警规则，例如：N=5时网络接收数据字节（MB）的平均值大于等于3时则报警。

> 说明 一个Grafana图表中可能有A、B、C等多条曲线数据，您可根据您的需求设置监控其中的一条线。

c. 在**PromQL输入框**中编辑或重新输入PromQL语句。

> 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]))` 。

vi. 选中通知方式，例如：选中短信。

vii. 设置通知对象。在**全部联系组**中单击联系人分组的名称，该联系人分组出现在**已选联系组**中，即设置成功。

后续步骤

ARMS Prometheus Grafana Kafka 大盘配置完毕后，您可以查看 Prometheus 监控指标和进一步自定义大盘，详见以下文档。

查看Prometheus监控指标

通过ARMS Prometheus自定义Grafana大盘

创建报警

相关文档

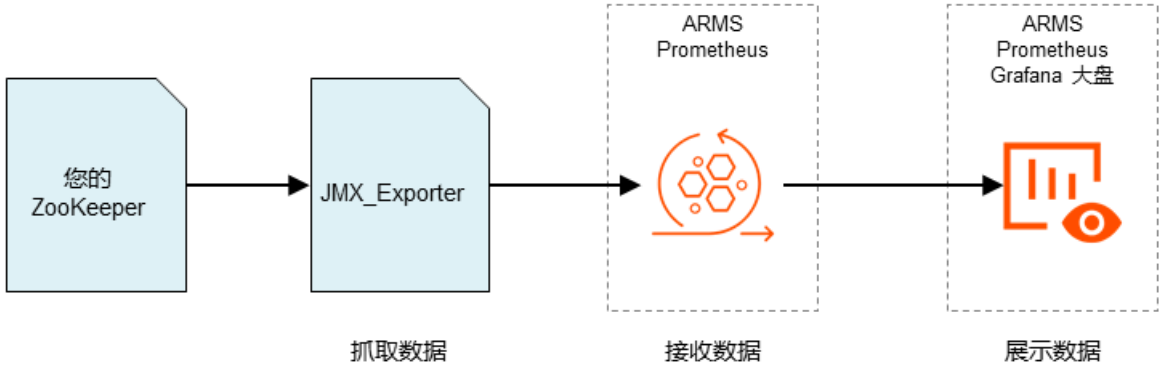
- 什么是Prometheus监控？

4.2.8. 通过 ARMS Prometheus 监控 ZooKeeper

通过在 ZooKeeper 中埋点来暴露数据，使用 ARMS Prometheus 监控抓取数据，并借助 ARMS Prometheus Grafana 大盘来展示数据，即可实现通过 ARMS Prometheus 监控 ZooKeeper 的目的。

背景信息

本教程的操作流程如图所示。



步骤一：启动 JMX 服务

首先需要在 ZooKeeper 中启用 JMX 服务以获取资源信息。

1. 修改*/opt/zk/zookeeper-3.4.10/bin/zkServer.sh*，在第44行添加 `JMXPORT=8999` 。具体添加位置如下所示。

```
if [ "x$JMXLOCALONLY" = "x" ]
then
JMXLOCALONLY=false
fi

JMXPORT=8999 ## 添加在此处第 44 行

if [ "x$JMXDISABLE" = "x" ] || [ "$JMXDISABLE" = 'false' ]
then
echo "ZooKeeper JMX enabled by default $JMXPORT ..." >&2
if [ "x$JMXPORT" = "x" ]
then
```

2. 重启 ZooKeeper。

```
/opt/zk/zookeeper-3.4.10/bin/zkServer.sh start /opt/zk/zookeeper-3.4.10/conf/zoo_sample.cfg &
```

步骤二：在 ZooKeeper 中启动 jmx_exporter

其次需要启动 jmx_exporter，让 JMX 信息可以直接通过HTTP方式访问，以便 ARMS Prometheus 监控抓取。

1. 下载 zookeeper.yaml 到*/opt/exporter_zookeeper/*目录下。
2. 修改下载的 */opt/exporter_zookeeper/zookeeper.yaml*文件，在第一行添加 `hostPort: localhost:8999` ，将 JMX 服务运行的端口暴露给 jmx_exporter。
3. 下载 jmx_exporter 的可执行文件到 *opt/exporter_zookeeper/*目录下。
4. 启动 jmx_exporter。

```
java -Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=8998 -cp /opt/exporter_zookeeper/jmx_prometheus_https
erver-0.12.0-jar-with-dependencies.jar
io.prometheus.jmx.WebServer 8997 /opt/exporter_zookeeper/zookeeper_exporter.yaml &
```

至此应用部分配置完成。可通过运行如下命令验证 jmx_exporter 是否可以正常工作。

```
curl http://<jmx_exporter 所在机器的IP>:9997/metrics
```

步骤三：配置 ARMS Prometheus 监控以抓取 ZooKeeper 的数据

接下来需要在 ARMS 控制台配置 ARMS Prometheus 监控以抓取 ZooKeeper 的数据。

1. 登录ARMS控制台。
2. 在左侧导航栏中单击 **Prometheus监控**。
3. 在 **Prometheus监控**页面顶部选择容器服务 K8s 集群所在的地域，然后单击目标集群的名称。
4. 在接下来的页面单击**配置详情**页签，然后单击**编辑prometheus.yaml**。

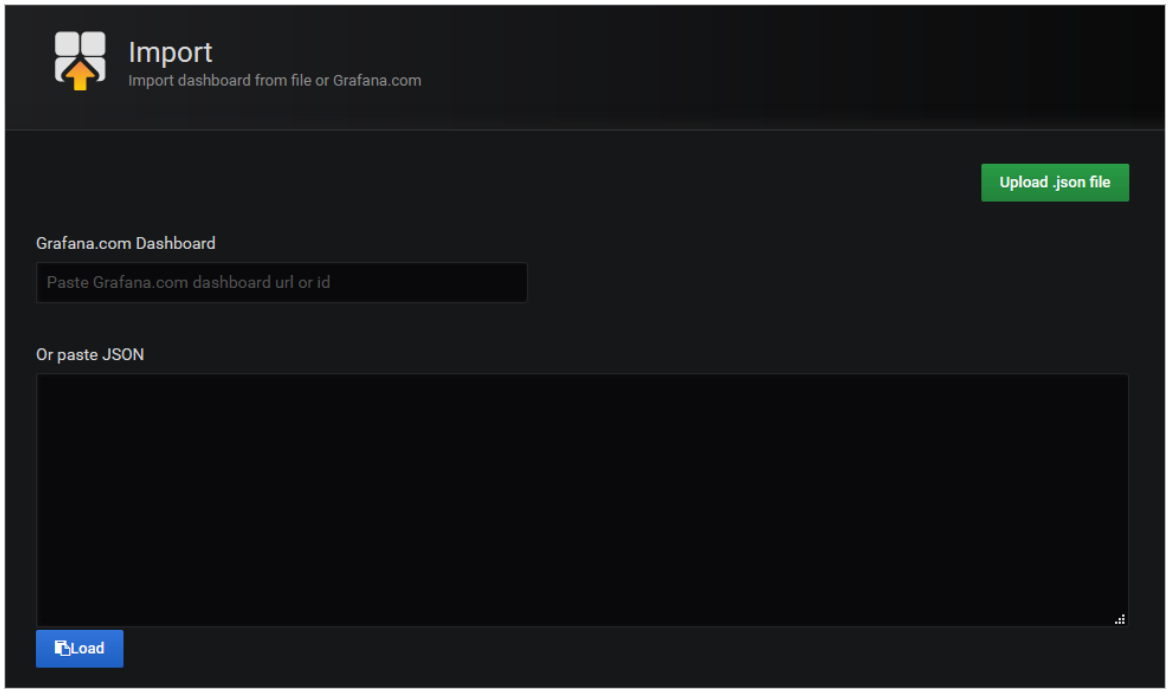
5. 粘贴以下代码。

```
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 every 1 minute.
scrape_configs:
 - job_name: 'zookeeper'
static_configs:
- targets: ['121.40.124.46:8997']
```

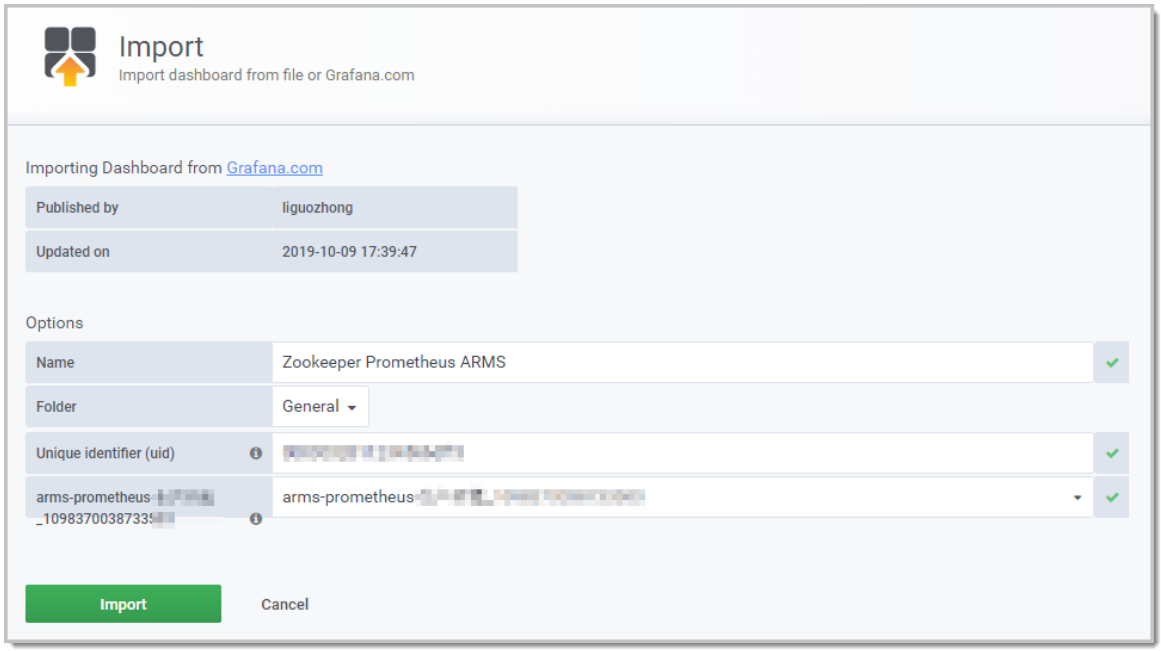
步骤四：通过 Grafana 大盘展示 ZooKeeper 的数据

最后需要在 ARMS 控制台导入 Grafana 大盘模板并指定 Prometheus 数据源所在的容器服务 K8s 集群。

1. 打开 ARMS Prometheus Grafana 大盘概览页。
2. 在左侧导航栏中选择 + > **Import**，并在 **Grafana.com Dashboard** 文本框输入 *10981*，然后单击 **Load**。



3. 在 Import 页面输入以下信息，然后单击 Import。

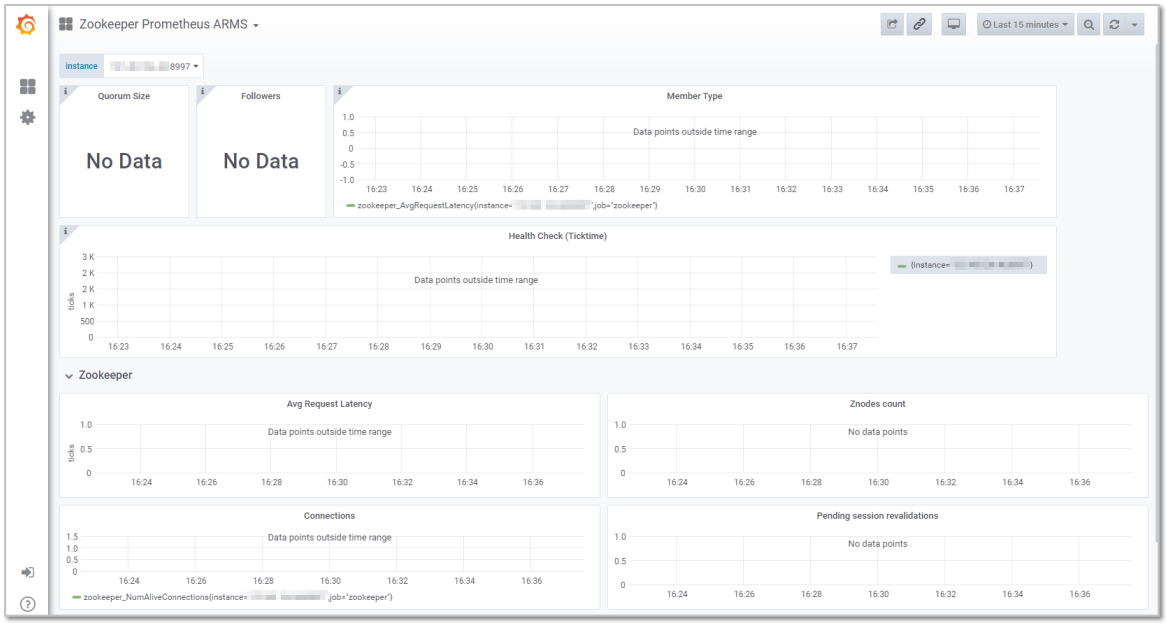


i. 在 Name 文本框中输入自定义的大盘名称。

ii. 在 Folder 下拉框中选择您的阿里云容器服务 K8s 集群。

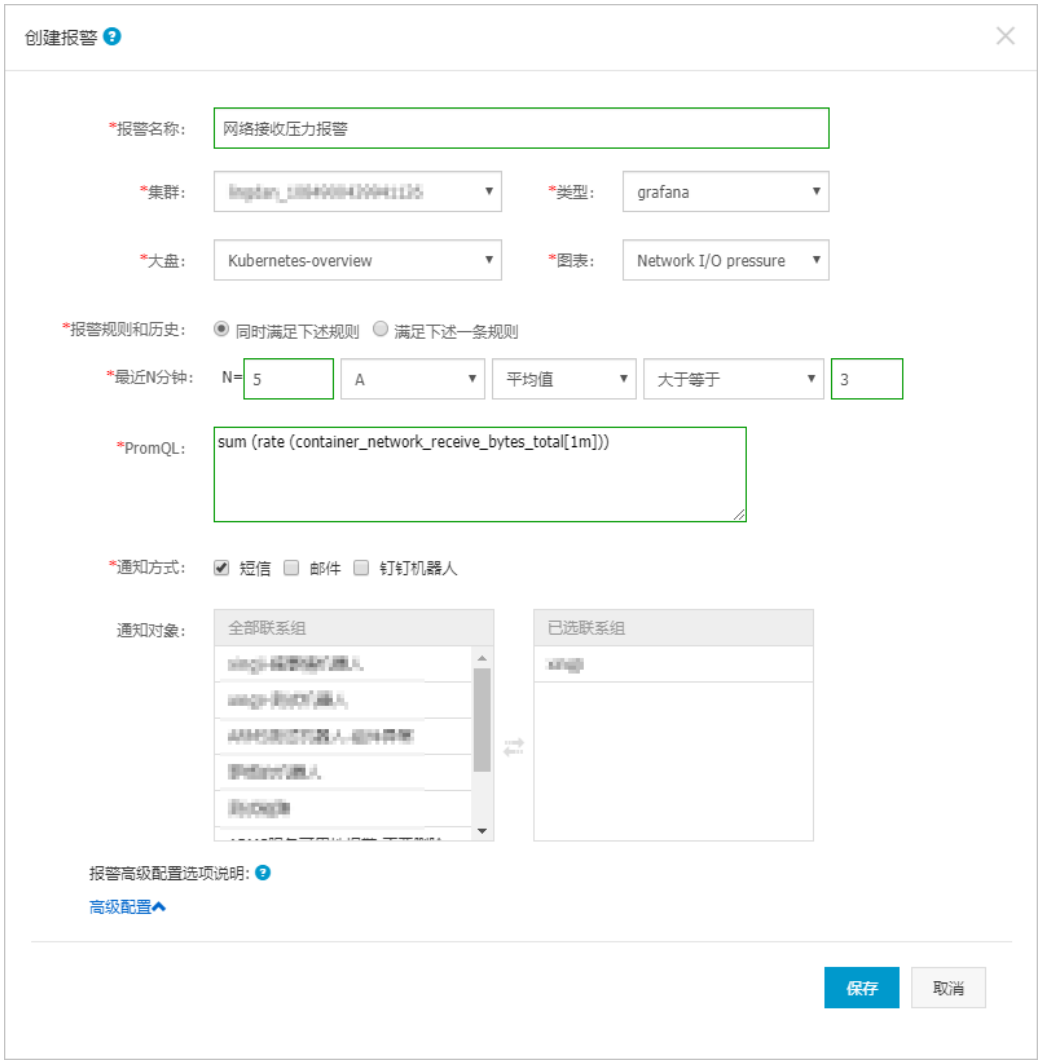
iii. 在最下方的下拉框中选择您的阿里云容器服务 K8s 集群。

配置完毕后的 ARMS Prometheus Grafana ZooKeeper 大盘如图所示。



步骤五：创建 Prometheus 监控报警

1. 报警创建提供两个入口，您可根据需要自行选择其中一个入口进入创建报警环节：
   - 在ARMS Prometheus Grafana大盘的New DashBoard页面，单击左侧的 图标，跳转至ARMS Prometheus报警规则和历史页面，在右上角单击创建报警 > Prometheus。
   - 在控制台左侧导航栏中选择报警管理 > 报警策略管理，进入报警规则和历史页面，在右上角单击创建报警 > Prometheus。
2. 在创建报警对话框中输入所有必填信息，完成后单击保存。



在创建报警对话框中配置如下参数：

i. 填写报警名称，例如：网络接收压力报警。
ii. 选择要创建报警的Prometheus监控对应的集群。
iii. 选择类型为grafana。
iv. 选择要监控的具体大盘和图表。

v. 设置报警规则。

a. 选中**同时满足下述规则**。

b. 编辑报警规则，例如：N=5时网络接收数据字节（MB）的平均值大于等于3时则报警。

> 说明 一个Grafana图表中可能有A、B、C等多条曲线数据，您可根据您的需求设置监控其中的一条线。

c. 在**PromQL输入框**中编辑或重新输入PromQL语句。

> 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]))` 。

vi. 选中通知方式，例如：选中短信。

vii. 设置通知对象。在**全部联系组**中单击联系人分组的名称，该联系人分组出现在**已选联系组**中，即设置成功。

后续步骤

ARMS Prometheus Grafana ZooKeeper 大盘配置完毕后，您可以查看 Prometheus 监控指标和进一步自定义大盘，详见相关文档。

相关文档

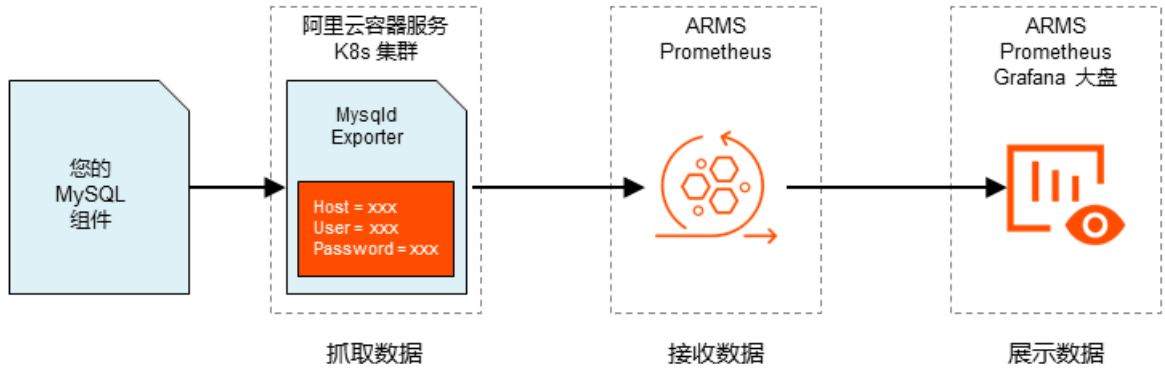
- 查看Prometheus监控指标
- 通过ARMS Prometheus自定义Grafana大盘
- 创建报警

4.3. 云服务接入

ARMS Prometheus监控的三方组件监控功能提供一键监控MySQL和Redis组件的能力，仅需配置相关认证信息，即可生成Grafana监控大盘展示监控数据。三方组件监控功能可监控的MySQL和Redis组件目前仅支持部署在阿里云上的云数据库RDS版和云数据库Redis版。

背景信息

手动通过ARMS Prometheus监控MySQL和Redis组件时，需要抓取数据、接收数据和展示数据三个操作步骤。例如监控MySQL组件的流程，如下图所示。



借助ARMS Prometheus监控的三方组件监控功能，您无需关心以上手动监控这些组件的监控流程，仅需提供阿里云账号的AccessKeyId、AccessKeySecret以及云数据库RDS版和云数据库Redis版的登录密钥，即可批量、自动化地监控这些组件。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击**Prometheus监控**，并在**Prometheus监控**页面顶部选择地域，然后单击需要查看的K8s集群名称。
3. 在左侧导航栏单击**三方组件监控**。

监控三方组件

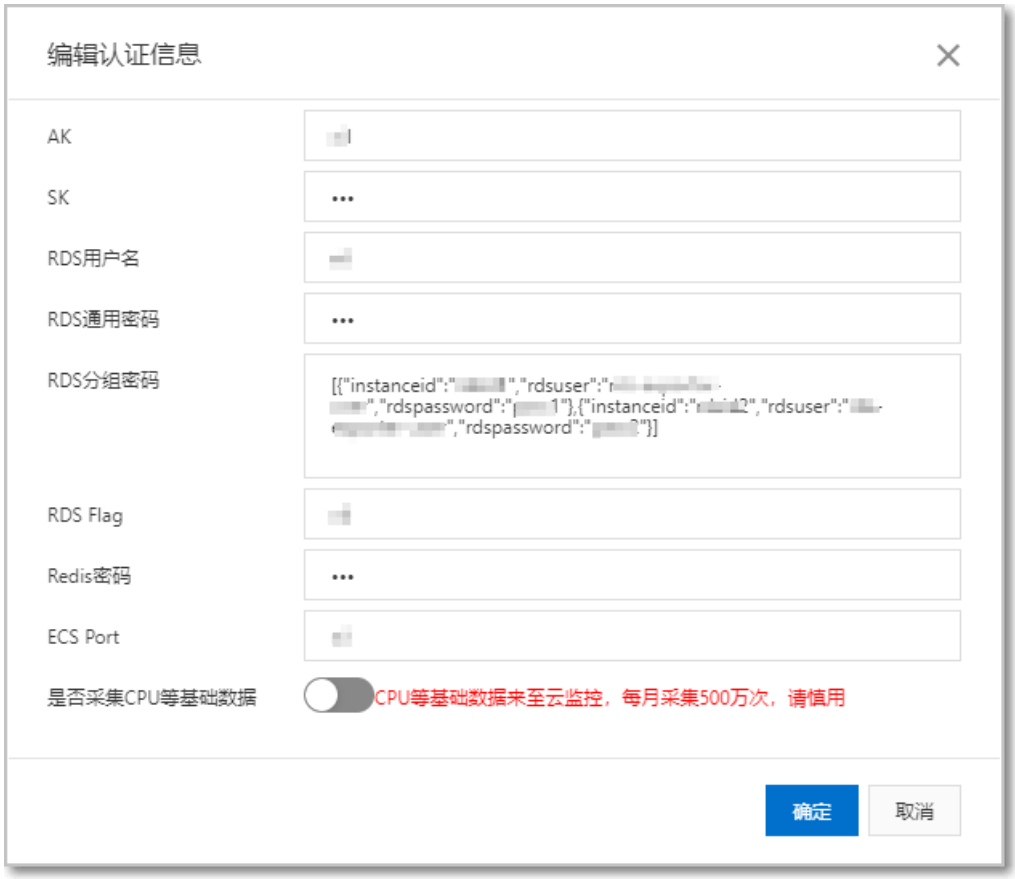
1. 在**三方组件监控**页面的右上角单击**编辑认证信息**。
2. 在**编辑认证信息**对话框中输入认证信息，并单击**确定**。

参数	描述
AK、SK	阿里云账号的AccessKeyId和AccessKeySecret。
RDS用户名	云数据库RDS版实例的用户名。
RDS通用密码	如果仅有一个云数据库RDS版实例，或者有多个实例并且使用同一个通用密码时，填写此通用密码。

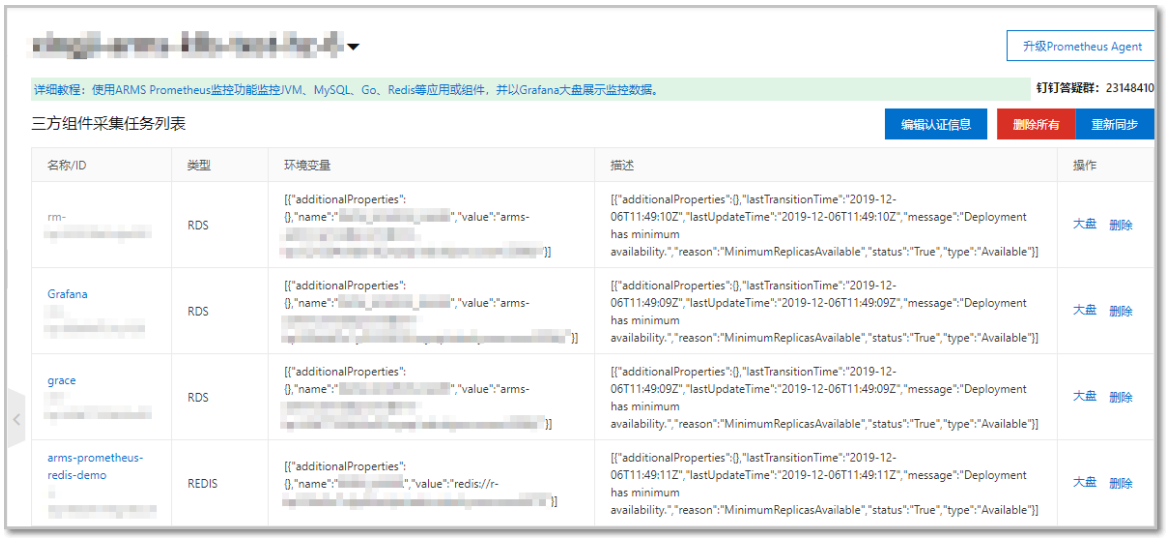
参数	描述

RDS分组密码	如果有多个云数据库RDS版实例，并且使用分组密码，则按照以下示例填写分组密码。
	说明
	◦ 请将<rdsid>、<rds-exporter-user>、<password>替换为具体的各个实例ID、实例名称以及实例密码。
	◦ 同时填写RDS通用密码和分组密码时，分组密码会覆盖通用密码。
	[
	{
	"instanceid":"<rdsid1>",
	"rdsuser":"<rds-exporter-user1>",
	"rdspassword":"<password1>"
	},
	{
	"instanceid":"<rdsid2>",
	"rdsuser":"<rds-exporter-user2>",
	"rdspassword":"<password2>"
	}
	]

RDS Flag	采集RDS数据的选择器，请参见MySQL Server Exporter。
Redis密码	云数据库Redis版实例的密码。
ECS Port	阿里云云服务器ECS用于暴露Prometheus监控数据的端口，例如：9090。填写此端口，即可在ECS采集Prometheus监控数据。
是否采集CPU等基础数据	如果需要监控云数据库RDS版或云数据库Redis版的CPU、内存等基础数据，可以开启此功能。 注意 以上的CPU、内存等基础数据来自阿里云的云监控，每月采集500万次。Prometheus监控正式商用之后，开启此功能会影响到您的费用，请谨慎使用。

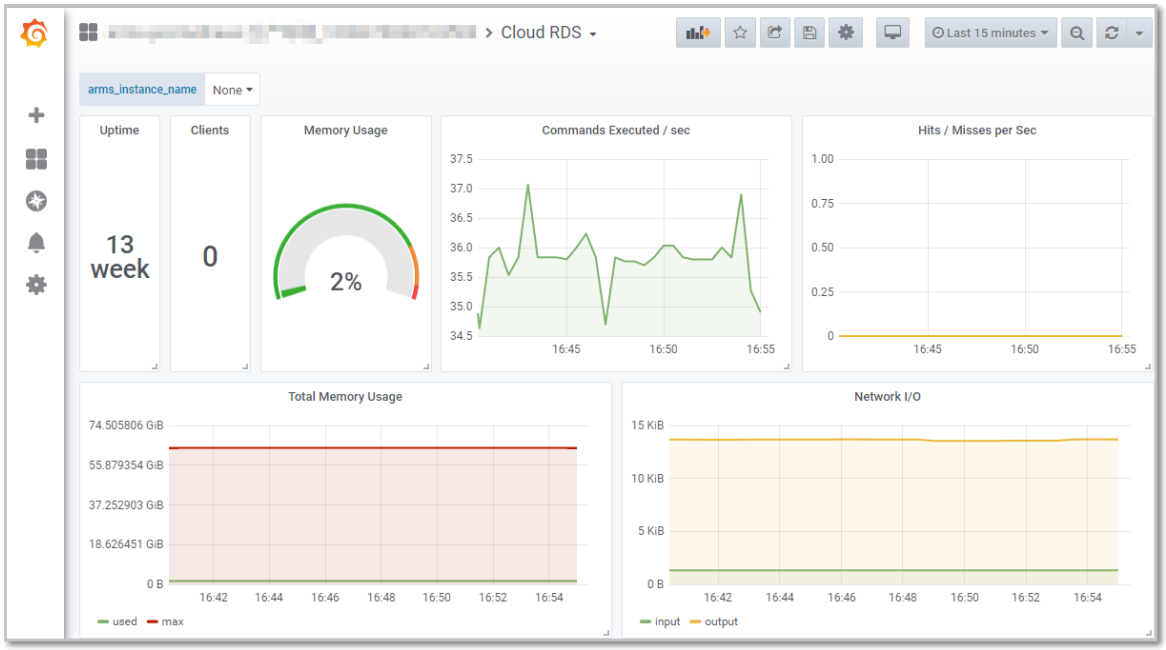


认证信息填写完成后，Prometheus监控会自动获取并监控集群下的云数据库RDS版和云数据库Redis版实例，生成Grafana监控大盘展示监控数据，并显示在三方组件采集任务列表中。



3. 单击需要查看的三方组件采集任务操作列的大盘，即可查看其对应的Grafana监控大盘。

> **说明** 三方组件采集任务的组件名称对应云数据库RDS版和云数据库Redis版的实例名称，单击组件名称，也可查看对应的Grafana监控大盘。



4. （可选）当集群下的云数据库RDS版或云数据库Redis版的实例有新增、修改或删除时，单击右上角的**重新同步**，可将其重新同步至三方组件采集任务列表中。

> **说明**
>
> - 三方组件监控任务列表每10分钟自动同步一次。
> - 每次编辑完认证信息后，三方组件监控任务列表会自动同步。

5.控制台功能

5.1. 查看Prometheus监控指标

ARMS Prometheus 监控提供的预置监控仪表板包括 K8s 集群概览、K8s 部署、Pod 和主机详情，您可以通过这些仪表板查看丰富的 Prometheus 监控指标，并按需更改仪表板数据的时间区间、刷新频率等属性。

前提条件

开始使用Prometheus监控

打开监控仪表板

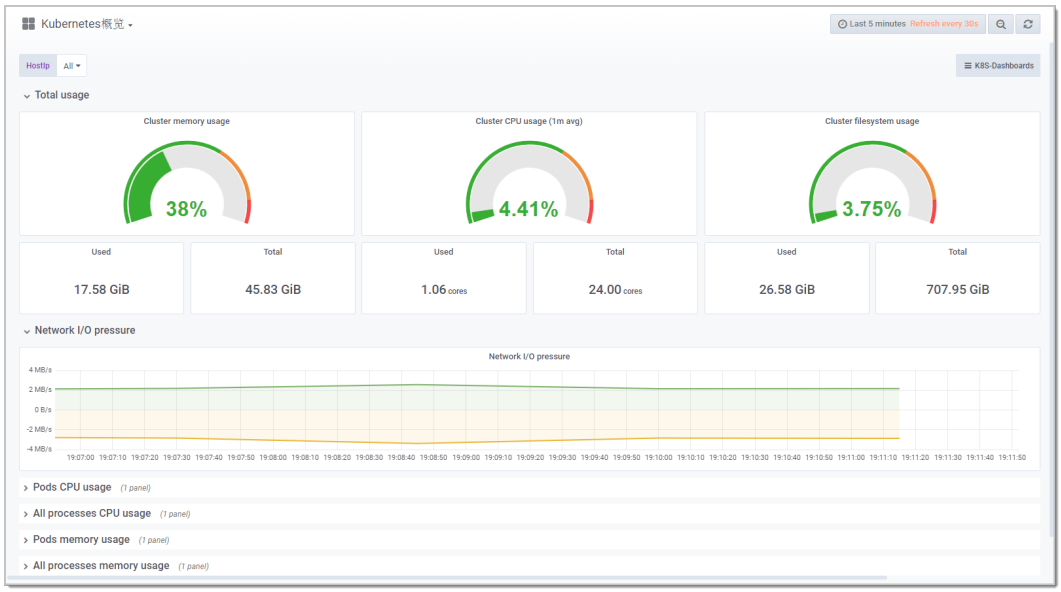
安装 Prometheus 监控插件后，即可在 **Prometheus 监控**页面打开预置的监控仪表板。

1. 登录 ARMS 控制台。
2. 在左侧导航栏中单击 **Prometheus 监控**。
   Prometheus 监控页面会列出您的阿里云账号下在阿里云容器服务 Kubernetes 版中的全部 Kubernetes 集群。
3. 在 **Prometheus 监控**页面上，单击**已安装插件**列中的链接，即可在浏览器新窗口中打开对应的监控仪表板。

查看监控指标

您可以通过以下预置监控仪表板查看监控指标。

- K8s 集群概览仪表板

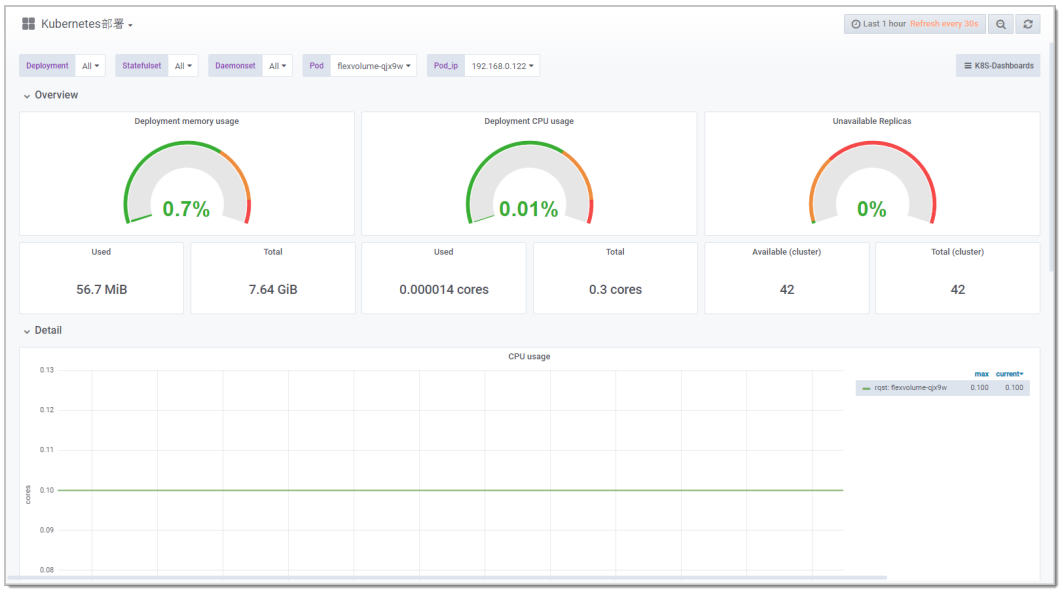


该仪表板展示的监控指标主要包括：

- 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。

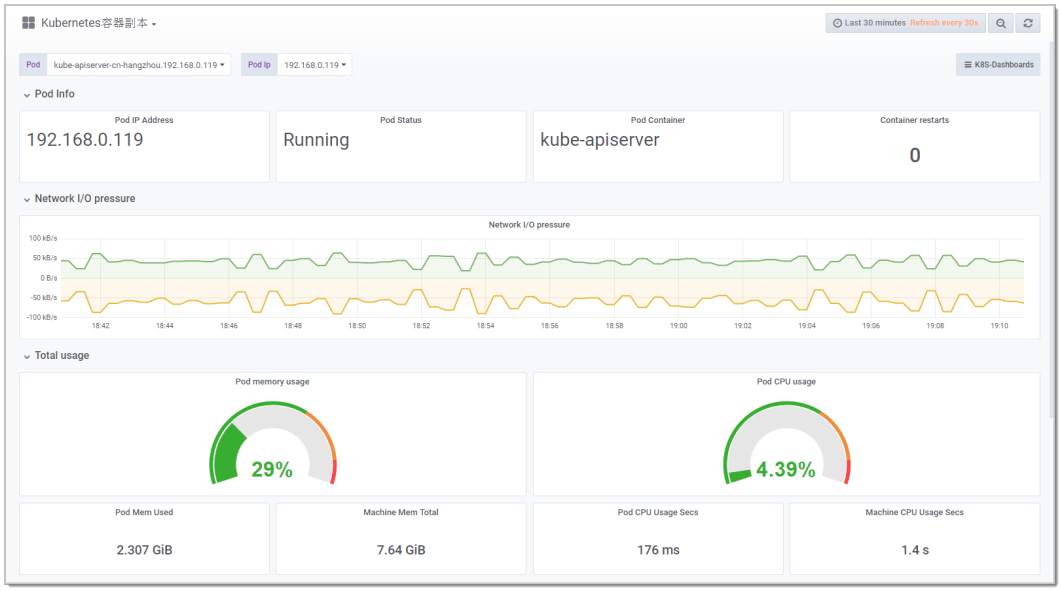
- K8s 部署仪表板



该仪表板展示的监控指标主要包括：

- 概览：部署 CPU 使用率、部署内存使用率、不可用副本数。
- 详情：CPU 使用率、内存使用率、全部进程网络 I/O。

- Pod 仪表板

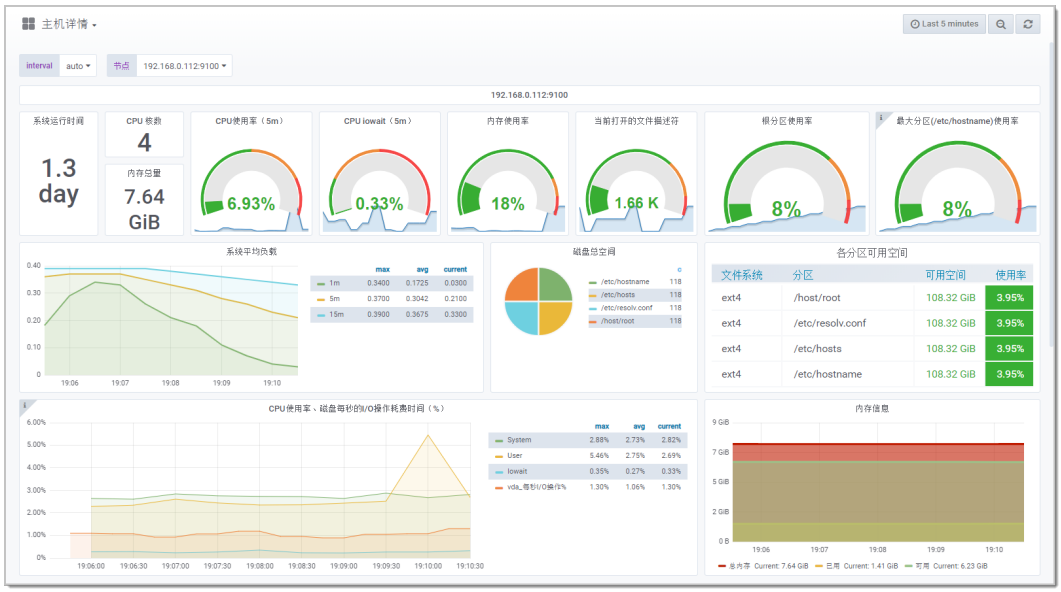


该仪表板展示的监控指标主要包括：

- 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。

- 主机详情仪表板

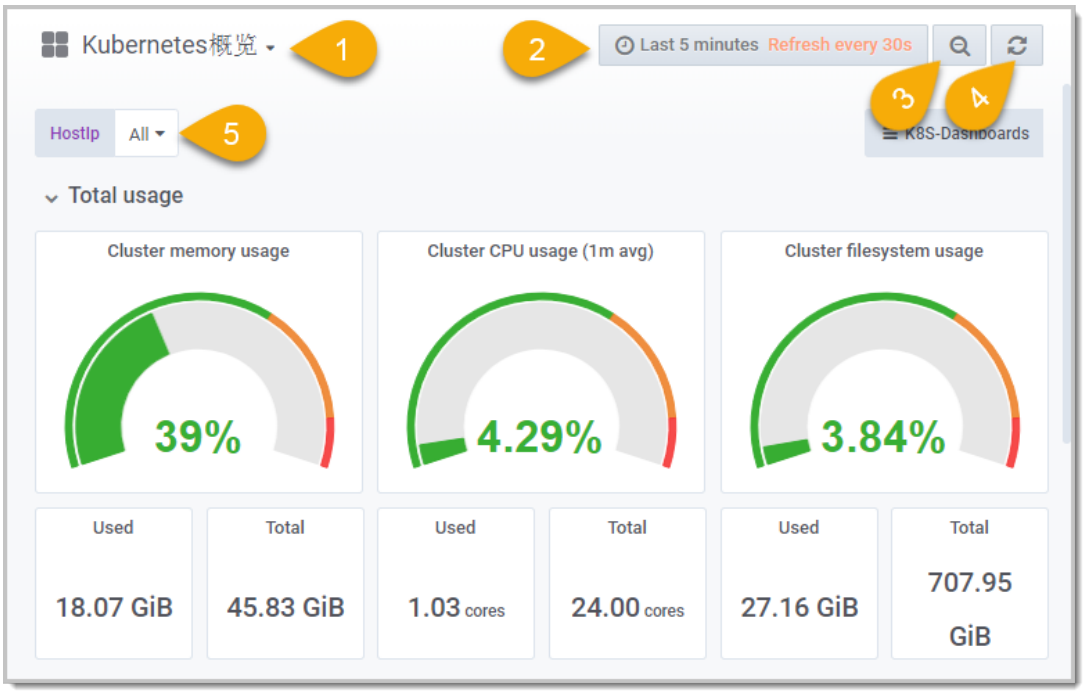


该仪表板展示的监控指标主要包括：

- 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。

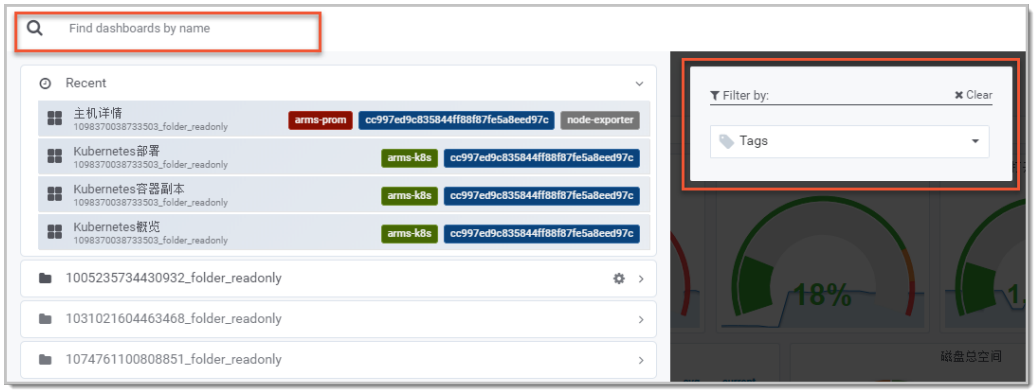
仪表板常用操作

对仪表板的常用操作如下所示。



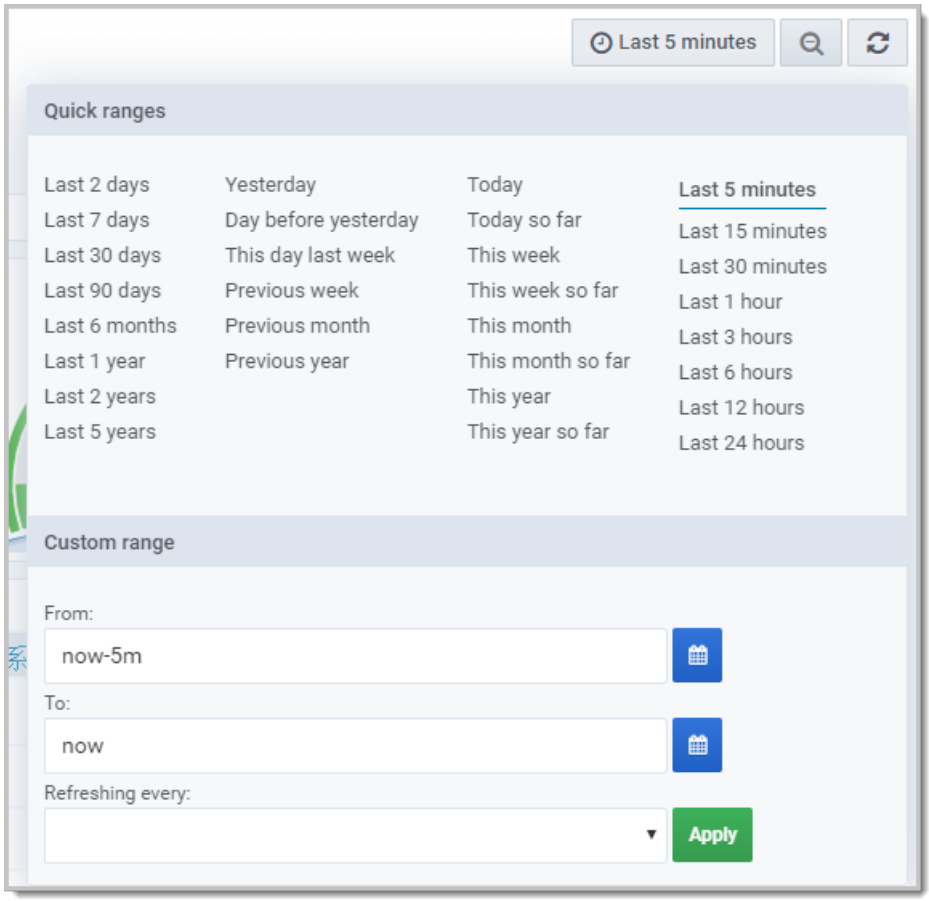
1. 切换仪表板

   该下拉菜单显示当前查看的仪表板名称，并可用于切换至下拉菜单中的其他仪表板。您可以通过在顶部搜索栏中输入名称来查找仪表板，或者使用 Filter（筛选条件）在下拉菜单中筛选出带有指定 Tag（标签）的仪表板。



2. 设置时间区间和刷新频率

   单击此图标后，您可以在浮层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。此外，您可以在浮层中设置仪表板的刷新频率。



3. 扩大时间区间

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，则单击一次该过大按钮后，时间区间的前面和后面将会各延长 5 分钟。

4. 手动刷新

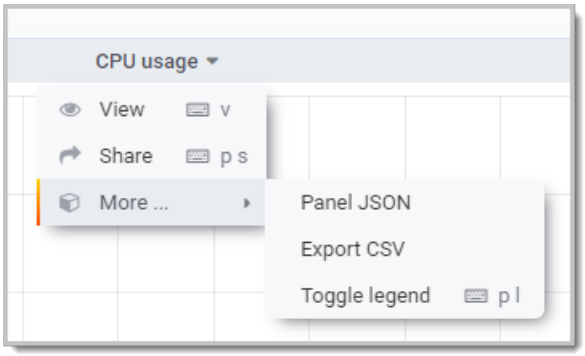
   单击此按钮将会刷新当前仪表板中所有面板的监控数据。

5. 筛选监控数据

   选择此下拉菜单中的选项即可筛选当前仪表板显示的监控数据。

仪表板面板常用操作

单击面板顶部的下拉菜单后，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。
- 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。
- 获得当前面板的 JSON 代码：选择 **More > Panel JSON**，然后在 **JSON** 对话框中拷贝 JSON 代码。
- 将当前面板的数据导出为 CSV 文件：选择 **More > Export CSV**，然后在 **Export CSV** 对话框中设置导出格式并导出。
- 打开或关闭图例：选择 **More > Toggle legend**，或依次按下 P 和 L 即可切换图例的可见性。

相关文档

- 什么是Prometheus监控？
- 开始使用Prometheus监控

5.2. 健康检查

健康检查功能可以检查 ARMS Prometheus 监控是否安装成功。如果您发现 ARMS Prometheus 监控无法监控到数据，那么可以根据健康检查结果排查原因。

前提条件

您已完成使用 ARMS Prometheus 监控您的应用或组件的操作步骤，请参见客户端接入概述。

功能入口

1. 登录ARMS控制台。
2. 在左侧导航栏单击 Prometheus 监控，并在 Prometheus 监控页面顶部选择地域，然后单击需要查看的 K8s 集群名称。
3. 在左侧导航栏单击健康检查。

查看健康检查结果

在健康检测页面，您可以查看健康检查结果。健康检查结果主要是 ARMS Prometheus 监控各个阶段的运行数据，包括：

1. Grafana 创建情况。
2. API 请求情况。

   > 说明 自定义创建的 Grafana 大盘和除 K8s 集群外的自建集群通过 API URL 获取数据源。

3. 容器服务 K8s 集群运行时状态采集情况。
4. ARMS Prometheus Agent 采集条数及详情。
5. 采集指标对应的采集任务（Job）详情。此项内容可以查看哪些采集任务（Job）是免费或者收费。
6. 采集指标的数量统计情况。
7. 获取 Promethues Metric 的种类数量，以及最近 1 分钟内的排序结果。

添加废弃指标

如果您想要解决单指标爆炸问题，或者不想监控某些指标，那么可以将其添加为废弃指标。

1. 在健康检测页面，单击右上角的编辑废弃指标。
2. 在编辑废弃指标对话框，添加需要废弃的指标名称 metricName，多个 metricName 通过换行隔开。

相关文档

- 客户端接入概述

5.3. 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，详情请参见开始使用Prometheus监控。

操作步骤

1. 登录ARMS控制台。
2. 在左侧导航栏中单击Prometheus监控。
3. 在目标集群右侧的操作列单击设置。
4. 为应用配置Prometheus监控采集规则分为以下两种情况。
   - 如果您需要监控部署在K8s集群内的应用的业务数据，例如订单信息，可以在服务发现页签上单击添加ServiceMonitor，并在新增ServiceMonitor对话框中参考以下示例内容进行填写。

```
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: /prometheus-metrics
namespaceSelector:
any: true
selector:
matchLabels:
#填写service.yaml的label字段，用来定位目标service.yaml
app: tomcat
```

   - 如果您需要监控部署在K8s集群之外的业务数据，如Redis连接数时，可以在配置详情页签上单击编辑prometheus.yaml配置原生的prometheus.yaml文件，并在编辑prometheus.yaml对话框中参考以下示例内容进行填写。

```
global:
scrape_interval: 15s
evaluation_interval: 15s
scrape_configs:
- job_name: 'prometheus'
static_configs:
- targets: ['localhost:9090']
```

6.产品对接

6.1. 将ARMS 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ARMS Prometheus监控数据，可以利用ARMS Prometheus监控提供的专用API接口轻松实现此目的。本文介绍了将ARMS Prometheus监控数据接入本地Grafana的实现方法。

Prometheus Grafana 本地

前提条件

- 您已成功安装ARMS Prometheus监控Agent，详情请参见开始使用Prometheus监控。
- 您已在本地成功安装Grafana软件，详情请参见Grafana。

快速入门

步骤一：获取ARMS Prometheus监控提供的专用API接口

该专用API接口是连接ARMS Prometheus和本地Grafana的纽带。请按照以下步骤获取该接口：

1. 登录ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，在顶部菜单栏选择目标地域，并在Prometheus监控页面表格的操作列中单击目标K8s集群的设置。
3. 在设置页面顶部单击Agent设置。
4. 在Agent设置页签上，复制步骤2：API接口地址后的地址。

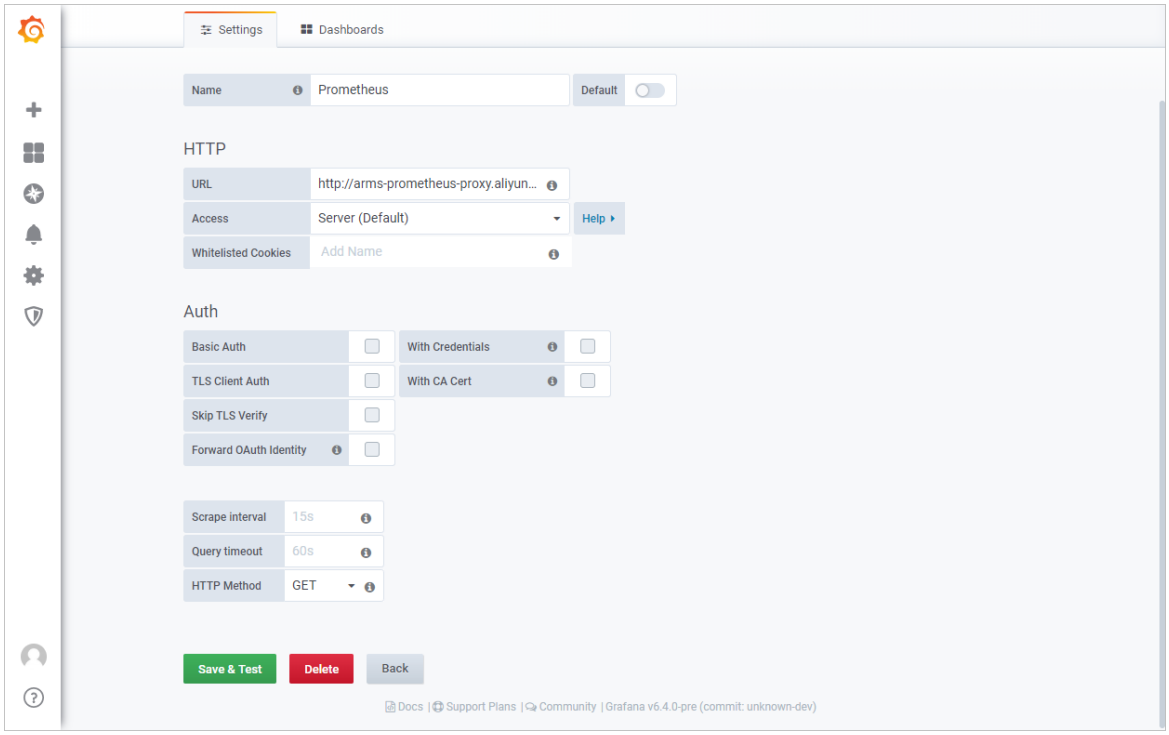
步骤二：在本地Grafana添加数据源

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。

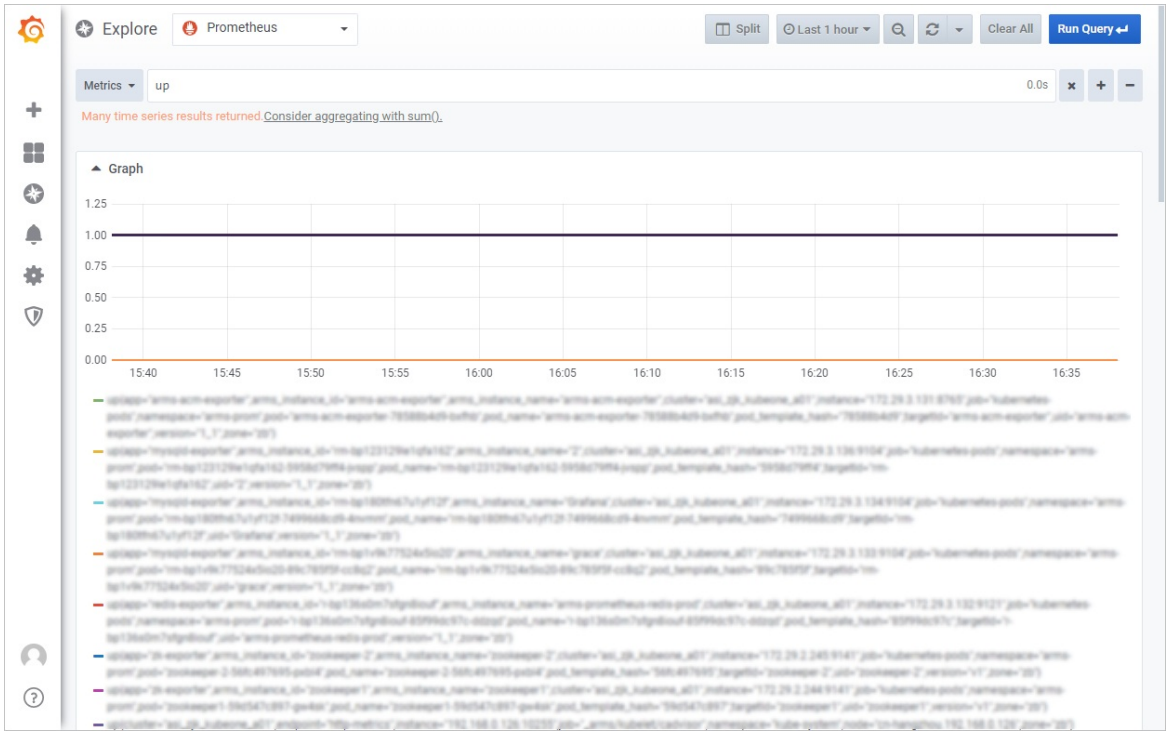
3. 在Data Sources页签上单击Add data source。
4. 在Add data source页面上单击Prometheus。
5. 在Settings页签的Name字段输入自定义的名称，在URL字段粘贴步骤一获得的API接口地址，并单击页签底部的Save & Test。



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. 在左侧导航栏中单击Explore。
3. 在Explore页面顶部的下拉框中选择步骤二中添加的数据源，在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是否正确，以及数据源是否有Prometheus监控数据。



7.参考信息

7.1. 基础指标说明

ARMS 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 1 月 6 日开始收费。

ARMS Prometheus 监控支持的基础指标涉及的采集任务（Job）如下表所示。

任务类型（Job）	任务名称（Job Name）
ARMS Prometheus	_arms-prom/node-exporter/0
	node-exporter
kubelet 信息	_arms/kubelet/metric
	_arms-prom/kube-apiserver/metric
	_arms-prom/kubelet/0
	_arms-prom/kubelet/1
Pod CPU	_arms/kubelet/cadvisor
	_arms-prom/kube-apiserver/cadvisor
API Server	_arms-prom-kube-apiserver
	apiserver
	_arms-prom/kube-apiserver/0
K8s 静态 YAML	_kube-state-metrics
Ingress	arms-ack-ingress
	ingress
CoreDNS	arms-ack-coredns
	coredns

您还可以在 Prometheus 监控健康检查页面查看基础指标涉及的采集任务（Job）。

1. 登录ARMS控制台。
2. 在左侧导航栏单击 Prometheus 监控，并在 Prometheus 监控页面顶部选择地域，然后单击需要查看的 K8s 集群名称。
3. 在左侧导航栏单击健康检查。
4. 在健康检查结果页面的健康检查结果第 4 步查看目前启动的采集任务（Job），根据是否免费，可以判断是否是基础指标涉及的采集任务（Job）。

7.2. 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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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	状态集更新未推出。
KubeDaemonSetRolloutStuck	kube_daemonset_status_number_ready{job="kube-state-metrics"} / kube_daemonset_status_desired_number_scheduled{job="kube-state-metrics"} < 1.00	15	DaemonSet推出回退。
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。
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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
KubeAPIErrorBudgetBurn	sum(apiserver_request:burnrate1h) > (14.40 * 0.01000) and sum(apiserver_request:burnrate5m) > (14.40 * 0.01000)	2	API错误过多。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
KubeNodeNotReady	kube_node_status_condition{job="kube-state-metrics",condition="Ready",status="true"} == 0	15	Node未准备好。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。
TargetDown	100 * (count(up == 0) BY (job, namespace, service) / count(up) BY (job, namespace, service)) > 10	10	目标下线。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
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。

报警名称	表达式	采集数据时间（分钟）	报警触发条件
InnoDB Plugin is enabled	mysql_global_variables_ignore_builtin_innodb == 1	1	插件启用。
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。
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。

8.Prometheus监控常见问题

本章节汇总了使用ARMS Prometheus监控的常见问题。

Prometheus监控 常见问题 集成

Grafana、Istio和HPA等第三方系统如何集成ARMS Prometheus监控？

Grafana、Istio和HPA等第三方系统集成ARMS Prometheus监控时，需要获取ARMS 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ARMS控制台。
2. 在左侧导航栏单击Prometheus监控，在顶部菜单栏选择目标地域，并在Prometheus监控页面表格的操作列中单击目标K8s集群的设置。
3. 在设置页面顶部单击Agent设置。
4. 在Agent设置页签上，复制步骤2：API接口地址后的地址。

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ARMS Prometheus监控。完整的Grafana集成ARMS Prometheus监控的操作请参见将ARMS Prometheus监控数据接入本地Grafana。

相关文档

- 将ARMS Prometheus监控数据接入本地Grafana