ALIBABA CLOUD

阿里云

日志服务

时序存储

文档版本：20220511

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.数据接入

1.1. 采集主机监控数据

日志服务Logtail支持采集主机CPU、内存、负载、磁盘、网络等监控数据。本文介绍如何通过日志服务控制台创建Logtail采集配置来采集主机监控数据。

前提条件

已在服务器上安装Logtail（Linux Logtail 0.16.40及以上版本）。更多信息，请参见安装Logtail（Linux系统）。

使用限制

- 不支持Windows版本。
- 不支持采集GPU、硬件状态等监控数据。

操作步骤

1. 登录日志服务控制台。
2. 在**接入数据**区域，单击**主机监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。
6. 在**数据源设置**页签中，配置**配置名称**和**插件配置**。

   inputs为Logtail采集配置，必选项，请根据您的数据源配置。

   > **说明** 一个inputs中只允许配置一个类型的数据源。

```
{
    "inputs": [
        {
            "detail": {
                "IntervalMs": 30000
            },
            "type": "metric_system_v2"
        }
    ]
}
```

参数	类型	是否必选	参数说明
type	string	是	数据源类型，固定为metric_system_v2。
IntervalMs	int	是	每次请求的间隔，单位：ms。不能低于5000，建议设置为30000。

7. 单击**下一步**，完成配置。

指标说明

主机CPU、内存、负载、磁盘、网络等指标说明如下：

- CPU相关指标

指标名	说明	单位	示例
cpu_count	CPU核数	个	2.0
cpu_util	CPU使用率，计算方式为排除idle、wait、steal后的占比	百分号（%）	7.68
cpu_guest_util	客户时间（guest time）占比	百分号（%）	0.0
cpu_guestnice_util	Nice进程客户时间（nice guest time）占比	百分号（%）	0.0
cpu_irq_util	硬中断处理时间（Hard Irq time）占比	百分号（%）	0.0
cpu_nice_util	Nice时间（Nice time）占比	百分号（%）	0.0
cpu_softirq_util	软中断处理时间（Soft Irq time）占比	百分号（%）	0.06
cpu_steal_util	等待宿主机CPU时间（Steal time）占比	百分号（%）	0.0
cpu_sys_util	内核态（System time）占比	百分号（%）	2.77
cpu_user_util	用户态（User time）占比	百分号（%）	4.84
cpu_wait_util	等待IO（Waiting time）占比	百分号（%）	0.11

- 内存相关指标

指标名	说明	单位	示例
mem_util	内存使用率	百分号（%）	51.03
mem_cache	已申请但未使用的内存	byte	3566386668.0
mem_free	未使用的内存	byte	177350084.0
mem_available	可用内存	byte	3699885553.0
mem_used	已使用内存	byte	4041510463.0
mem_swap_util	swap内存使用率	百分号（%）	0.0
mem_total	内存总量	byte	7919128576.0

- 磁盘相关指标

指标名	说明	单位	示例
disk_rbps	硬盘每秒读取流量	byte/s	8376.81
disk_wbps	硬盘每秒写入流量	byte/s	247633.58
disk_riops	硬盘每秒读取次数	次/s	0.22
disk_wiops	硬盘每秒写入次数	次/s	43.39
disk_rlatency	平均读延迟	ms	2.83
disk_wlatency	平均写延迟	ms	2.15
disk_util	IO使用率	百分号（%）	0.27
disk_space_usage	磁盘使用百分比	百分号（%）	9.12
disk_inode_usage	inode使用率	百分号（%）	1.18
disk_space_used	磁盘已使用容量	byte	11068512238.59
disk_space_total	磁盘总量	byte	126692061184.0
disk_inode_total	inode总量	byte	7864320.0
disk_inode_used	inode已使用容量	byte	93054.78

- NET相关指标

指标名	说明	单位	示例
net_drop_util	丢弃的数据包占总数据包的比值	百分号（%）	0.0

指标名	说明	单位	示例
net_err_util	报错数据包占总数据包的比值	百分号（%）	0.0
net_in	网络接收速率	byte/s	8440.91
net_in_pkt	每秒接收的数据包	个/s	40.83
net_out	网络发送速率	byte/s	12446.53
net_out_pkt	每秒发送的数据包	个/s	39.95

- TCP相关指标

指标名	说明	单位	示例
protocol_tcp_established	已建立连接数	个	205.0
protocol_tcp_insegs	接收的所有报文数	个	4654.0
protocol_tcp_outsegs	发送的报文数	个	4870.0
protocol_tcp_retran_segs	重传报文数	个	0.0
protocol_tcp_retran_util	重传报文占总发送报文数量的比值	百分号（%）	0.0

- system相关指标

指标名	说明	单位	示例
system_boot_time	系统启动时间	s	1578461935.0
system_load1	系统平均负载，1分钟平均值	不涉及	0.58
system_load5	系统平均负载，5分钟平均值	不涉及	0.68
system_load15	系统平均负载，15分钟平均值	不涉及	0.60

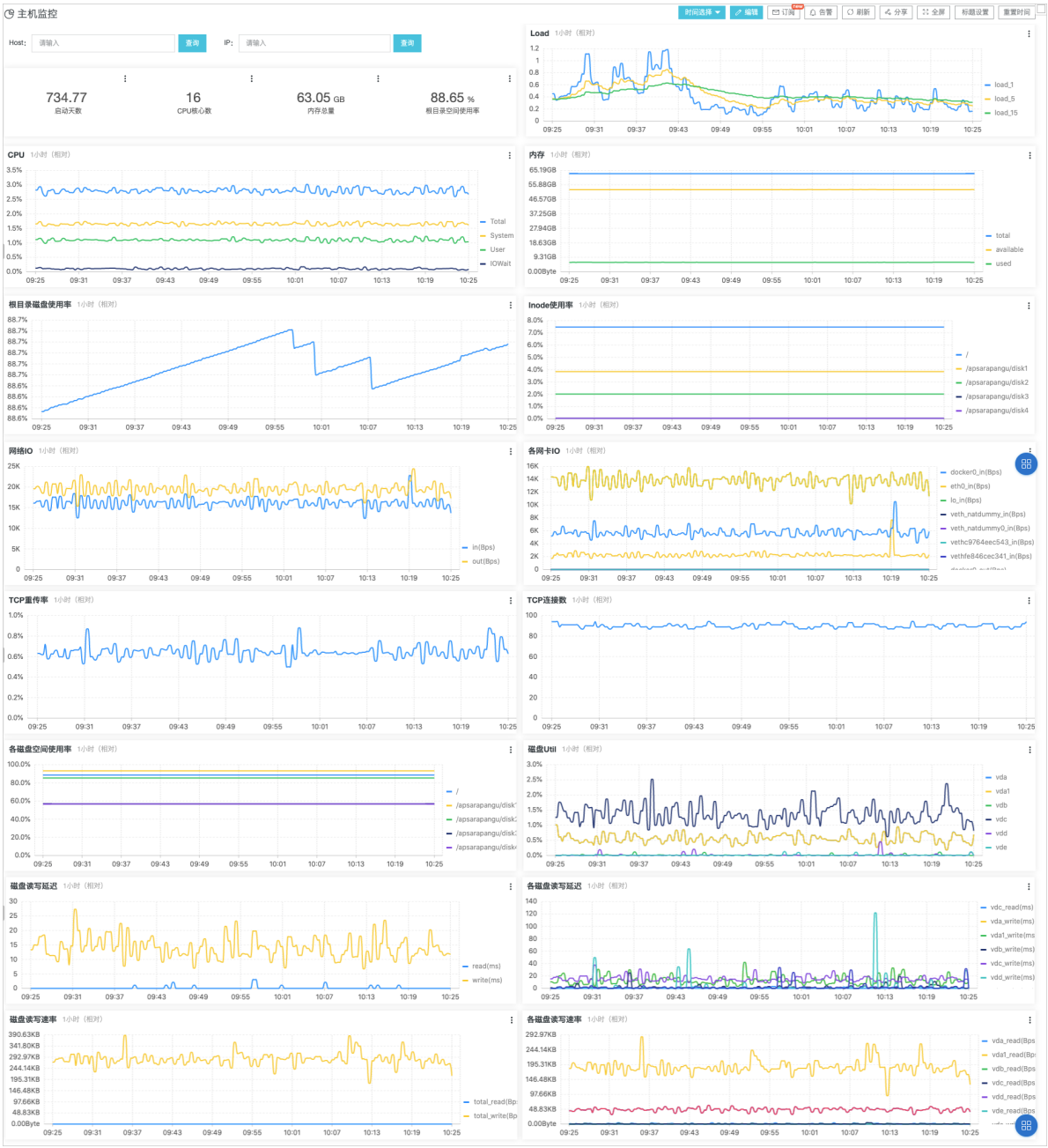
后续步骤

- 查询分析

  采集到数据后，您可以在MetricStore查询分析页面进行查询分析操作。更多信息，请参见查询和分析时序数据。

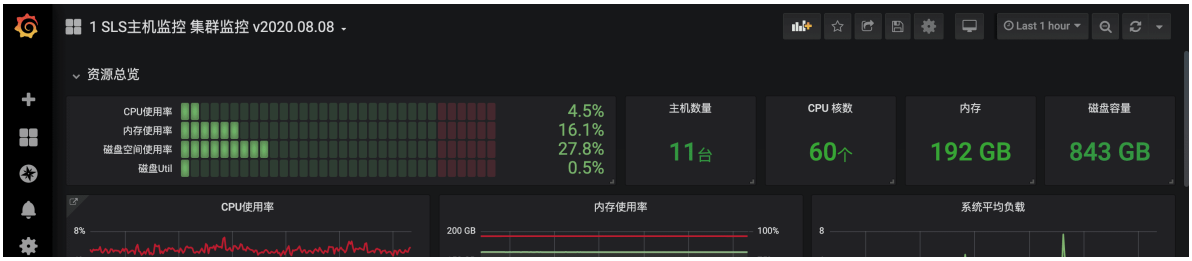
- 日志服务可视化

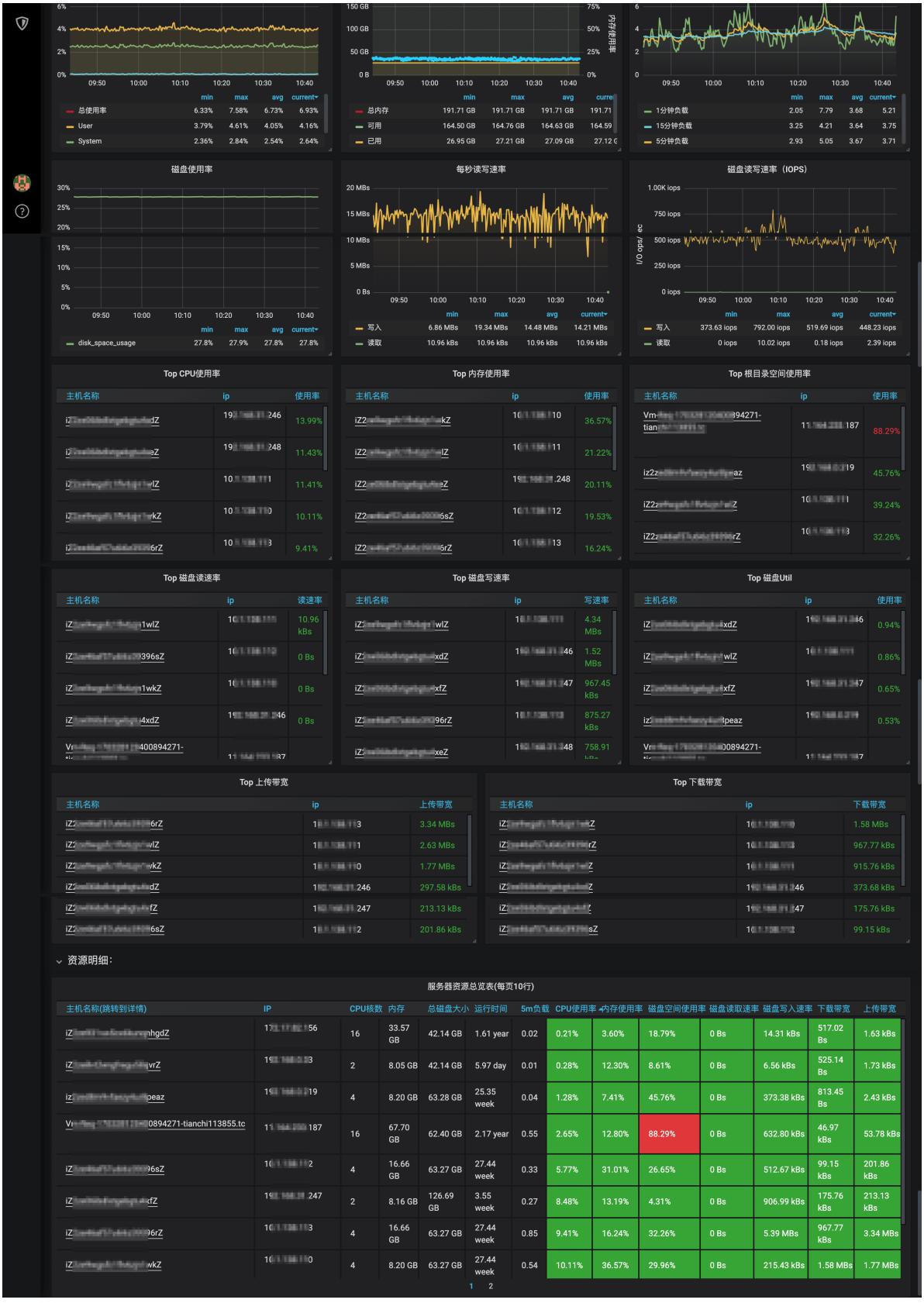
日志服务自动在对应Project中生成主机监控仪表盘，您可以直接使用该仪表盘查看查询分析结果，及进行告警等相关操作。



- Grafana可视化

日志服务为主机监控数据提供Grafana模板，您可以使用Grafana仪表盘展示查询分析结果。更多信息，请参见使用Prometheus采集Kubernetes监控数据。Grafana模板详情请参见《1 SLS主机监控单机指标v2020.08.08》。





1.2. 采集Open-Falcon数据

Open-Falcon是一款企业级、高可用、可扩展的开源监控解决方案，用于监控服务器的状态，例如磁盘空间、端口存活、网络流量等。本文介绍如何通过Logtail和Transfer将Open-Falcon数据上传至日志服务。

前提条件

已在服务器上安装Logtail（Linux Logtail 0.16.44及以上版本），详情请参见安装Logtail（Linux系统）。

出于性能和可靠性考虑，推荐将Logtail和Open-Falcon的transfer模块安装在相同机器上。

使用限制

您所使用的Open-Falcon版本需包含Influxdb support功能。

步骤1：创建Logtail采集配置

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**时序存储 > 时序库**页签中，单击目标MetricStore下面**数据接入 > logtail配置**右侧的加号。
4. 在**接入数据**页面中，单击**自定义数据插件**。
5. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
6. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

7. 在**数据源设置**页签中，配置**配置名称**和**插件配置**。

   inputs为Logtail插件采集配置，必选项，请根据您的数据源配置。

   > **说明** 一个inputs中只允许配置一个类型的数据源。

```
{
    "inputs": [
        {
            "detail": {
                "Format": "influx",
                "Address": ":127.0.0.1:8476"
            },
            "type": "service_http_server"
        }
    ],
    "global": {
        "AlwaysOnline": true,
        "DelayStopSec": 500
    }
}
```

参数	类型	是否必选	参数说明
type	string	是	数据源类型，固定为service_http_server。
Format	string	是	数据类型，固定为influx。
Address	string	是	监听地址与端口，格式为**ip:port**。

8. 单击**下一步**，完成配置。

步骤2：修改Open-Falcon配置

1. 登录Open-Falcon所在服务器。
2. 添加transfer配置。
   i. 打开配置文件。

      配置文件默认为*cfg.json*。

   ii. 将如下脚本添加到配置文件中。

      address中配置的IP地址和端口号要与您在步骤7中配置的Address中的IP地址和端口号一致，详情参数说明请参见Transfer。

```
    "influxdb": {
        "enabled": true,
        "batch": 200,
        "retry": 3,
        "maxConns": 32,
        "precision": "s",
        "address": "http://127.0.0.1:8478",
        "timeout": 5000
    }
```

后续步骤

配置完成后，日志服务将Open-Falcon数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

1.3. 采集ping和tcping数据

本文介绍通过Logtail采集ping和tcping数据到日志服务Metricstore的操作步骤。

前提条件

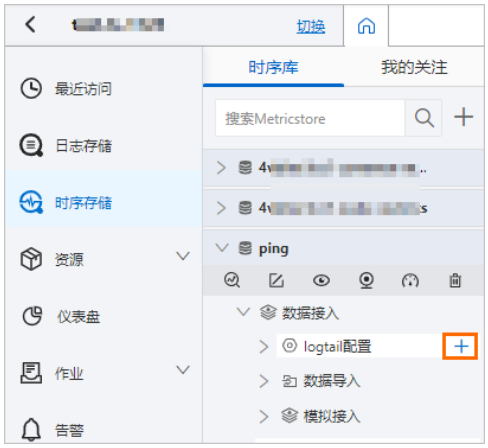
已创建Project和MetricStore。具体操作，请参见创建Project和创建MetricStore。

使用限制

只有Linux Logtail 1.0.31及以上版本的Logtail支持采集ping和tcping数据。如果您已在服务器上安装旧版本的Logtail，需先升级。具体操作，请参见在线升级Logtail。

操作步骤

1. 登录日志服务控制台。
2. 在**Project列表**区域，单击目标Project。
3. 在**时序存储 > 时序库**页签中，单击目标MetricStore下面**数据接入 > logtail配置**右侧的加号。



4. 在**快速数据接入**面板中，单击**自定义数据插件**。
5. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
6. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

7. 在**数据源设置**页签中，配置**配置名称**和**插件配置**。

   inputs为Logtail插件采集配置，必选项，请根据您的数据源配置。

> **说明** 一个inputs中只允许配置一个类型的数据源。

```
{
    "inputs": [
        {
            "detail": {
                "tcp": [
                    {
                        "port": 80,
                        "src": "192.XX.XX.103",
                        "count": 3,
                        "target": "www.aliyun.com"
                    }
                ],
                "interval_seconds": 60,
                "icmp": [
                    {
                        "src": "192.XX.XX.103",
                        "count": 3,
                        "target": "www.aliyun.com"
                    }
                ]
            },
            "type": "metric_input_netping"
        }
    ]
}
```

参数	类型	是否必选	参数说明

参数	类型	是否必选	参数说明
tcp	array	是	采集TCP ping数据。详细参数说明如下，参数值需根据实际值替换。 ◦ port：端口号。 ◦ src：执行ping命令的服务器的IP地址。即由src字段决定在机器组的哪台机器中执行ping命令。 ◦ count：限制执行一次ping命令发送的数据包数量。建议使用默认值3，取值范围为(0,10)。 ◦ target：目标主机名或IP地址。即由target字段决定ping的目标主机名或IP地址。 支持添加多个IP地址，示例如下： `"tcp": [ { "port": 80, "src": "192.XX.XX.103", "count": 3, "target": "www.aliyun.com" }, { "port": 80, "src": "192.XX.XX.104", "count": 3, "target": "www.aliyun.com" } ]`

参数	类型	是否必选	参数说明
icmp	array	是	采集ICMP ping数据。详细参数说明如下，参数值需根据实际值替换。 ○ src：执行ping命令的服务器的IP地址。即由src字段决定在机器组的哪台机器中执行ping命令。 ○ count：限制执行一次ping命令发送的数据包数量。建议使用默认值3，取值范围为(0,10)。 ○ target：目标的主机名或IP地址。即由target字段决定ping的目标主机名或IP地址。 支持添加多个IP地址，示例如下： `"icmp": [ { "src": "192.XX.XX.103", "count": 3, "target": "www.aliyun.com" }, { "src": "192.XX.XX.104", "count": 3, "target": "www.aliyun.com" } ]`
interval_seconds	int	是	执行ping命令的时间间隔，单位：秒。 ○ 默认值：60。 ○ 取值范围：[10, 86400)
type	string	是	数据源类型，固定为metric_input_netping。

8. 单击**下一步**，完成配置。

后续步骤

采集ping数据后，您可以在Metricstore中进行查询分析。具体操作，请参见查询和分析时序数据。

相关指标说明如下表所示。

分类	指标名	说明
ICMP ping	ping_failed	单次执行icmp ping命令，发送失败的数据包数量。
	ping_rtt_avg_ms	单次执行icmp ping命令的平均响应时间，单位：毫秒。
	ping_rtt_max_ms	单次执行icmp ping命令的最大响应时间，单位：毫秒。
	ping_rtt_min_ms	单次执行icmp ping命令的最小响应时间，单位：毫秒。
	ping_rtt_stddev_ms	单次执行icmp ping命令的标准差时间，单位：毫秒。
	ping_rtt_total_ms	单次执行icmp ping命令的总响应时间，单位：毫秒。
	ping_succcess	单次执行icmp ping命令，发送成功的数据包数量。
	ping_total	单次执行icmp ping命令，发送的数据包总数。
TCP ping	tcping_failed	单次执行tcp ping命令，发送失败的数据包数量。
	tcping_rtt_avg_ms	单次执行tcp ping命令的平均响应时间，单位：毫秒。
	tcping_rtt_max_ms	单次执行tcp ping命令的最大响应时间，单位：毫秒。
	tcping_rtt_min_ms	单次执行tcp ping命令的最小响应时间，单位：毫秒。
	tcpping_rtt_stddev_ms	单次执行tcp ping命令的标准差时间，单位：毫秒。
	tcping_rtt_total_ms	单次执行tcp ping命令的总响应时间，单位：毫秒。

分类	指标名	说明
	tcping_succcess	单次执行tcp ping命令，发送成功的数据包数量。
	tcping_total	单次执行tcp ping命令，发送的数据包总数。

1.4. 接入Prometheus监控数据

1.4.1. 通过Remote Write协议接入Prometheus监控数据

Prometheus是一款面向云原生的监控软件，支持众多软件、系统的数据采集与监控。本文介绍如何将Prometheus监控数据采集到日志服务，从而使用日志服务对数据进行分析与监控。

前提条件

- 已创建MetricStore。具体操作，请参见创建MetricStore。
- 已安装Prometheus。具体操作，请参见GETTING STARTED。
- 已在Prometheus上配置数据采集规则。具体操作，请参见scrape_config。

操作步骤

日志服务支持Prometheus的Remote Write协议，只需要在Prometheus中启动Remote Write功能即可采集数据到日志服务，相关操作如下所示。

1. 登录Prometheus所在服务器。
2. 打开配置文件，并根据实际情况替换如下参数。具体操作，请参见remote_write。

```
url: https://sls-prometheus-test.cn-beijing.log.aliyuncs.com/prometheus/sls-prometheus-test/prometheus-raw/api/v1/write
basic_auth:
  username: access-key-id
  password: access-key-secret
queue_config:
  batch_send_deadline: 20s
  capacity: 20480
  max_backoff: 5s
  max_samples_per_send: 2048
  min_backoff: 100ms
  min_shards: 100
```

参数	说明

参数	说明
url	日志服务MetricStore的URL，格式为https://*{project}*.*{sls-endpoint}*/prometheus/*{project}*/*{metricstore}*/api/v1/write。其中： ○ *{sls-endpoint}*：服务入口。更多信息，请参见服务入口。 ○ *{project}*：您已创建的Project。 ○ *{metricstore}*：您已创建的MetricStore。 **注意** ○ 如果您是在阿里云内网，请优先使用内网域名。 ○ 为保证传输安全性，请务必使用https。
basic_auth	鉴权信息，以Remote Write协议写入数据到日志服务需要BasicAuth鉴权。其中： ○ username为您的阿里云账号AccessKey ID。 ○ password为您的阿里云AccessKey Secret。 建议您使用只具备日志服务Project写入权限的RAM用户AccessKey。更多信息，请参见授予指定Project写入权限。

queue_config	queue_config用于设置写入的缓存、重试等策略。
	为避免过多无效网络请求，建议min_backoff不低于100ms，max_backoff不低于5s。
	如果Prometheus数据量较大，可修改queue_config配置，建议修改为：
	`batch_send_deadline: 20s`
	`capacity: 20480`
	`max_backoff: 5s`
	`max_samples_per_send: 2048`
	`min_backoff: 100ms`
	`min_shards: 100`

3. 验证是否已上传数据到日志服务。

   配置好Prometheus后，您可通过预览方式查看数据是否已上传到日志服务。

   i. 登录日志服务控制台。

   ii. 在Project列表区域，单击目标Project。

iii. 在**时序存储 > 时序库**页签中，选择目标MetricStore右侧的**图标 > 消费预览**。

在**消费预览**页面，如果有数据，则表示配置成功。

后续步骤

采集到Prometheus监控数据后，您可以进行如下操作：

- 使用日志服务查询和分析Prometheus监控数据。具体操作，请参见查询和分析时序数据。
- 使用Grafana可视化展示Prometheus监控数据。具体操作，请参见时序数据对接Grafana。

1.4.2. 通过Logtail插件接入Prometheus监控数据

日志服务Logtail插件支持采集Prometheus格式的各类指标数据，例如Node Exporter、Kafka Exporter及应用所涉及的Prometheus指标等。本文介绍如何通过日志服务控制台创建Logtail采集配置来采集Prometheus监控数据。

前提条件

已创建MetricStore。具体操作，请参见创建MetricStore。

操作步骤

> **注意** 一个Logtail插件上只能同时存在一个Prometheus的Logtail配置。如果同时存在多个，则随机生效一个。

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**抓取Prometheus格式指标**。
3. 在**选择日志空间**向导中，选择目标Project和MetricStore，单击**下一步**。
4. 创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）。

a. 在**ECS机器**页签中，通过手动选择实例方式选择目标ECS实例，单击**立即执行**。

更多信息，请参见安装Logtail（ECS实例）。

> **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Linux Logtail 0.16.66及以上版本。更多信息，请参见安装Logtail（Linux系统）。

b. 安装完成后，单击**确认安装完毕**。

c. 在**创建机器组**页面，输入**名称**，单击**下一步**。

日志服务支持创建IP地址机器组和用户自定义标识机器组，详细参数说明请参见创建IP地址机器组和创建用户自定义标识机器组。

5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**向导中，设置**配置名称**和**插件配置**，单击**下一步**。

插件配置包括inputs和processors。日志服务已提供inputs模板，包括global和scrape_configs两个节点。

- inputs为Logtail采集配置，必选项，请根据您的数据源配置。

> **注意**
> - Prometheus格式指标的抓取配置和Prometheus本身的抓取配置规则一致，只支持global和scrape_configs两个节点的配置。更多信息，请参见Prometheus抓取配置规则。
> - 一个inputs中只允许配置一个类型的数据源。

- processors为Logtail处理配置，可选项。

如果您上报的数据需要添加Logtail所在主机的IP、HostName或其他自定义字段，您可以打开**开启高级编辑模式**开关，添加processors配置，此处需使用追加字段插件。例如：

```
{
  "processors":[
    {
      "type":"processor_appender",
      "detail": {
        "Key": "__labels__",
        "Value": "|host#$#{{__host__}}|ip#$#{{__ip__}}",
        "SortLabels": true
      }
    }
  ]
}
```

更多信息，请参见追加字段。

后续步骤

- 查询和分析

采集到数据后，您可以在MetricStore查询和分析页面进行查询和分析操作。具体操作，请参见查询和分析时序数据。

- 日志服务可视化

  日志服务自动在对应Project中生成主机监控仪表盘，您可以直接使用该仪表盘查看查询分析结果，及进行告警等相关操作。具体操作，请参见仪表盘。

- Grafana可视化

  日志服务的时序数据支持直接对接Grafana进行可视化。具体操作，请参见时序数据对接Grafana。

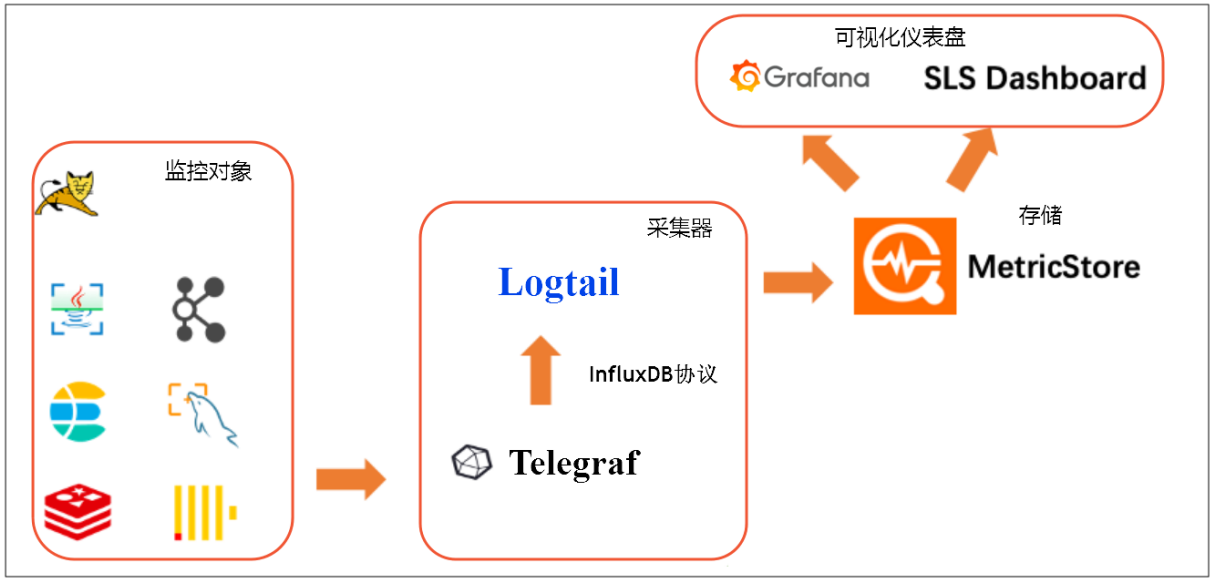
1.5. 接入Telegraf数据

1.5.1. 概述

Telegraf是InfluxData开发的数据采集器，支持众多的输入输出插件（例如MySQL、Redis、Elasticsearch等），在开源领域被广泛使用。本文介绍Telegraf的工作机制、安装步骤及采集方式。

工作机制

日志服务基于Telegraf开发的监控模板，支持将Telegraf采集的监控数据（MySQL监控数据、Redis监控数据、Elasticsearch监控数据等）通过InfluxDB协议写入Logtail，Logtail再将监控数据上传到日志服务MetricStore中。针对常用插件，日志服务提供界面化配置，同时为您自动创建对应的仪表盘。Telegraf整体工作机制如下图所示。



安装Telegraf

- 如果Logtail版本>=0.16.50，服务器首次获取到Telegraf配置时会自动安装Telegraf。
- 如果0.16.48=<Logtail版本<0.16.50，您需要先更新Logtail到最新版本。如果不更新，则需要手动安装Telegraf，步骤如下：
  i. 确认您要安装Telegraf的机器所在的地域并选择网络，详情请参见选择网络。
  ii. 下载logtail.sh安装脚本，详情请参见安装Logtail（Linux系统）。

  仅下载脚本即可，无需执行安装。如果您无法确定网络环境，可使用公网下载。

```
wget http://logtail-release-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/linux64/logtail.sh -O logtail.sh;
chmod 755 logtail.sh
```

iii. 安装Telegraf。

```
sudo ./logtail.sh install-agent ${region} telegraf
```

${region}为您机器所在地域，例如**sudo ./logtail.sh install-agent cn-hangzhou-internet telegraf**。

iv. 确认安装结果。

执行如下命令查看*telegraf*目录，如果*telegraf*目录下存在*telegraf*、*telegrafd*等文件，则说明安装成功。

```
ls /etc/ilogtail/telegraf
```

采集模式

使用Telegraf采集监控数据时，您可以选择如下两种采集方式：

- 本机采集

  使用Telegraf采集本机的监控指标，您机器组中的机器即为您采集的目标机器，配置页面中的服务器地址可填写为127.0.0.1。如果无特殊需求，推荐使用该模式。

- 远程采集

  您可以单独使用一台机器安装Telegraf，并使用该机器远程采集其他机器上的监控指标，此时配置页面中的服务器地址需填写为对应的机器IP地址或服务域名作为采集目标。使用远程采集时，机器组中只能有一台机器，否则会产生重复数据。适用场景如下：

  - 当您的采集目标是云服务时，您无法部署Logtail和Telegraf，则可使用远程采集。
  - 当您不希望在正在运行某些服务的机器上部署额外采集器时，可使用远程采集实现无侵入式监控。

1.5.2. 接入Elasticsearch监控数据

您可使用Telegraf采集Elasticsearch监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建Elasticsearch可视化监控方案。本文介绍如何通过日志服务来完成Elasticsearch监控数据的采集和可视化。

前提条件

- 已在服务器上安装Logtail（Linux Logtail 0.16.48及以上版本）。更多信息，请参见安装Logtail（Linux系统）。
- Telegraf所在的服务器可通过内网连接Elasticsearch服务器。

操作步骤

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**Elasticsearch监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。

4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
集群名称	Elasticsearch集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名称*的标签。 **说明** 请确保该集群名称唯一，否则可能出现数据冲突。
服务器列表	单击**+**，添加Elasticsearch服务器信息，相关配置项如下所示： - 地址：Elasticsearch服务器的连接地址。 - 端口号：默认为9200，无需修改。 您可以根据业务需求，添加多台Elasticsearch服务器信息。
索引名称	单击**+**，添加Elasticsearch索引名称，用于监控Elasticsearch索引指标。配置为_all，表示采集所有索引的指标数据。 您可以根据业务需求，添加多个索引名称。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击**+**，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  配置完成后，日志服务自动在对应Project中生成名为*Elasticsearch监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.3. 接入MySQL监控数据

您可使用Telegraf采集MySQL监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建MySQL可视化监控方案。本文介绍如何通过日志服务来完成MySQL监控数据的采集和可视化。

前提条件

- 已在服务器上安装Logtail（Linux Logtail 0.16.48及以上版本）。更多信息，请参见安装Logtail（Linux系统）。
- Telegraf所在的服务器可通过内网连接MySQL服务器。

使用限制

仅支持MySQL 5.5及以上版本。

操作步骤

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**MySQL监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
集群名称	MySQL集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名*的标签。 **说明** 请确保该集群名称唯一，否则可能出现数据冲突。
服务器列表	单击+，添加MySQL服务器信息，相关配置项如下所示： ○ **账户**：MySQL服务器的用户名。 **说明** 建议创建独立用户用于监控MySQL，并只授予监控相关权限。 ○ **密码**：MySQL服务器的用户密码。 ○ **地址**：MySQL服务器的连接地址，可以为服务器IP地址、主机名、域名。 ○ **端口**：MySQL服务器的连接端口号，默认为3306。 您可以根据业务需求，添加多台MySQL服务器信息。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击+，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  配置完成后，日志服务自动在对应Project中生成名为*MySQL监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.4. 接入Redis监控数据

您可使用Telegraf采集Redis监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建Redis可视化监控方案。本文介绍如何通过日志服务来完成Redis监控数据的采集和可视化。

前提条件

- 已在服务器上安装Logtail（Linux Logtail 0.16.48及以上版本）。更多信息，请参见安装Logtail（Linux系统）。
- Telegraf所在的服务器可通过内网连接Redis服务器。

操作步骤

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**Redis监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。
6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
集群名称	Redis集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名*的标签。 **说明** 请确保该集群名称唯一，否则可能出现数据冲突。

参数名称	说明
服务器列表	单击+，添加Redis服务器信息，相关配置项如下所示： ○ **地址**：Redis服务器的连接地址。 ○ **端口**：Redis服务器的连接端口号。 您可以根据业务需求，添加多台Redis服务器信息。
是否需要密码	当Redis服务器设置了鉴权时，此处需要输入对应的Redis服务器密码。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击+，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  配置完成后，日志服务自动在对应Project中生成名为*Redis监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.5. 接入Kafka监控数据

您可使用Telegraf采集Kafka监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建Kafka可视化监控方案。本文介绍如何通过日志服务来完成Kafka监控数据的采集和可视化。

前提条件

- 已在服务器上安装Linux Logtail 0.16.48或以上版本。更多信息，请参见安装Logtail（Linux系统）。
- 已在服务器上安装Java 1.6或以上版本 。

步骤1：创建Logtail采集配置

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**Kafka监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。

- 如果您已有可用的机器组，请单击**使用现有机器组**。
- 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
  a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

     如果已在ECS上安装Logtail，请单击**确认安装完毕**。

     > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

  b. 安装完成后，单击**确认安装完毕**。
  c. 创建机器组。

     如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。

5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
集群名称	Kafka集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名称*的标签。 **说明** 请确保该集群名称唯一，否则可能出现数据冲突。
服务器列表	单击**+**，添加Kafka服务器信息。 - **地址**：Kafka服务器的连接地址。 - **端口**：Kafka服务器的端口号，您可自定义配置，但要与步骤2：配置JavaAgent中配置的端口号一致。 您可以根据业务需求，添加多台Kafka服务器信息。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击**+**，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

步骤2：配置JavaAgent

完成Logtail采集配置后，您还需要将JMX协议转换为HTTP协议。日志服务支持使用Jolokia将JMX协议转换为HTTP协议。您可以按照Jolokia官方文档下载及加载Jolokia，也可以使用日志服务Logtail自带的Jolokia JavaAgent。Logtail自带的Jolokia JavaAgent位于 `/etc/ilogtail/telegraf/javaagent/jolokia-jvm.jar` 中。

您需要在kafka所在机器上设置 `KAFKA_JVM_PERFORMANCE_OPTS` 环境变量，例如 `export KAFKA_JVM_PERFORMANCE_OPTS=-javaagent:/etc/ilogtail/telegraf/javaagent/jolokia-jvm.jar=port=7777` ，其中7777为服务器的端口号，与步骤1：创建Logtail采集配置中配置的端口号保持一致。

> **说明** 默认Jolokia JavaAgent只在127.0.0.1上监听，即只允许本机请求。如果您的Logtail和被监控的应用不在相同的机器上，您可以在添加的脚本中补充host=字段，使其可监听其他IP地址。如果设置为host=0.0.0.0，则表示监听所有IP地址。相关命令如下所示：
>
> ```
> -javaagent:/tmp/jolokia-jvm.jar=port=7777,host=0.0.0.0
> ```

设置完成后，需重启应用。如果您暂时无法重启应用，可使用如下命令将Jolokia JavaAgent连接到指定的Java进程，实现实时生效。其中进程PID请根据实际值替换。

> **说明** 该操作仅用于测试，请确保按照上述操作完成配置，否则重启后将失效。

```
java -jar /etc/ilogtail/telegraf/javaagent/jolokia-jvm.jar --port 7777 start 进程PID
```

如果返回如下信息则表示连接成功。

```
Jolokia is already attached to PID 752
http://127.0.0.1:7777/jolokia/
```

连接成功后，您可以访问该URL，验证连接是否正常。

```
curl http://127.0.0.1:7777/jolokia/
# 返回参考
{"request":{"type":"version"},"value":{"agent":"1.6.2","protocol":"7.2","config":{"listenFo
rHttpService":"true","maxCollectionSize":"0","authIgnoreCerts":"false","agentId":"30.43.124
.186-752-5b091b5d-jvm","debug":"false","agentType":"jvm","policyLocation":"classpath:\/jolo
kia-access.xml","agentContext":"\/jolokia","serializeException":"false","mimeType":"text\/p
lain","maxDepth":"15","authMode":"basic","authMatch":"any","discoveryEnabled":"true","strea
ming":"true","canonicalNaming":"true","historyMaxEntries":"10","allowErrorDetails":"true","
allowDnsReverseLookup":"true","realm":"jolokia","includeStackTrace":"true","maxObjects":"0"
,"useRestrictorService":"false","debugMaxEntries":"100"},"info":{"product":"tomcat","vendor
":"Apache","version":"8.5.57"}},"timestamp":1602663330,"status":200}
```

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  配置完成后，日志服务自动在对应Project中生成名为*kafka监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.6. 接入Clickhouse监控数据

您可使用Telegraf采集Clickhouse监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建Clickhouse可视化监控方案。本文介绍如何通过日志服务来完成Clickhouse监控数据的采集和可视化。

前提条件

- 已在服务器上安装Logtail（Linux Logtail 0.16.48及以上版本）。更多信息，请参见安装Logtail（Linux系统）。
- Telegraf所在的服务器可通过内网连接Clickhouse服务器。

操作步骤

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**Clickhouse监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。
6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。

参数名称	说明
集群名称	Clickhouse集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名称*的标签。 **说明** 请确保该集群名称唯一，否则可能出现数据冲突。
用户名	Clickhouse服务器的用户名。
密码	Clickhouse服务器的用户密码。
服务器列表	单击+，添加Clickhouse服务器信息。 ○ **地址**：Clickhouse服务器的连接地址。 ○ **端口**：默认为8123，无需修改。 您可以根据业务需求，添加多台Clickhouse服务器信息。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击+，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  配置完成后，日志服务自动在对应Project中生成名为*Clickhouse监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.7. 接入Java应用或Tomcat的监控数据

您可使用Telegraf采集Java应用或Tomcat的监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建Java应用或Tomcat可视化监控方案。本文以Java应用监控数据为例，介绍如何通过日志服务来完成Java应用数据的采集和可视化。

前提条件

- 已在服务器上安装Linux Logtail 0.16.48或以上版本。更多信息，请参见安装Logtail（Linux系统）。
- 已在服务器上安装Java 1.6或以上版本。

步骤1：创建Logtail采集配置

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**Java应用监控**。

   此处以接入Java应用监控数据为例，如果您要接入Tomcat监控数据，请选择**Tomcat监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。

     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。
6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
应用名称	输入您的应用名称。配置该参数后，日志服务会为您的数据添加*cluster=应用名*的标签。 **说明** 请确保该应用名称唯一，否则将出现数据冲突。
服务器列表	单击**+**，添加服务器信息。 - **地址**：服务器的连接地址。 - **端口**：服务器的端口号，您可自定义配置，但要与步骤2：配置JavaAgent中配置的端口号一致。 您可以根据业务需求，添加多台服务器信息。

参数名称	说明
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击**+**，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

步骤2：配置JavaAgent

完成Logtail采集配置后，您还需要将JMX协议转换为HTTP协议。日志服务支持使用Jolokia将JMX协议转换为HTTP协议。您可以按照Jolokia官方文档下载及加载Jolokia，也可以使用日志服务Logtail自带的Jolokia JavaAgent。Logtail自带的Jolokia JavaAgent位于*/etc/ilogtail/telegraf/javaagent/jolokia-jvm.jar*中。

- 如果是普通Java应用，需在Java启动参数中添加 `-javaagent:/etc/ilogtail/telegraf/javaagent/jolokia-jvm.jar=port=7777` 。
- 如果是Tomcat，需设置 `JAVA_OPTS` 环境变量，例如 `export JAVA_OPTS="-javaagent:/etc/ilogtail/telegraf/jolokia-jvm.jar=port=7777"` ，其中7777为服务器端口号，要与步骤1：创建Logtail采集配置中配置的端口号一致。

> **说明** 默认Jolokia JavaAgent只在127.0.0.1上监听，即只允许本机请求。如果您的Logtail和被监控的应用不在相同的机器上，您可以在添加的脚本中补充host=字段，使其可监听其他IP地址。如果设置为**host=0.0.0.0**，则表示监听所有IP地址。相关命令如下所示：
>
> ```
> -javaagent:/tmp/jolokia-jvm.jar=port=7777,host=0.0.0.0
> ```

设置完成后，需重启应用。如果您暂时无法重启应用，可使用如下命令将Jolokia JavaAgent连接到指定的Java进程，实现实时生效。其中进程PID请根据实际值替换。

> **说明** 该操作仅用于测试，请确保按照上述操作完成配置，否则重启后将失效。

```
java -jar /etc/ilogtail/telegraf/javaagent/jolokia-jvm.jar --port 7777 start 进程PID
```

如果返回如下信息则表示连接成功。

```
Jolokia is already attached to PID 752
http://127.0.0.1:7777/jolokia/
```

连接成功后，您可以访问该URL，验证连接是否正常。

```
curl http://127.0.0.1:7777/jolokia/
# 返回参考
{"request":{"type":"version"},"value":{"agent":"1.6.2","protocol":"7.2","config":{"listenFo
rHttpService":"true","maxCollectionSize":"0","authIgnoreCerts":"false","agentId":"30.43.124
.186-752-5b091b5d-jvm","debug":"false","agentType":"jvm","policyLocation":"classpath:\/jolo
kia-access.xml","agentContext":"\/jolokia","serializeException":"false","mimeType":"text\/p
lain","maxDepth":"15","authMode":"basic","authMatch":"any","discoveryEnabled":"true","strea
ming":"true","canonicalNaming":"true","historyMaxEntries":"10","allowErrorDetails":"true","
allowDnsReverseLookup":"true","realm":"jolokia","includeStackTrace":"true","maxObjects":"0"
,"useRestrictorService":"false","debugMaxEntries":"100"},"info":{"product":"tomcat","vendor
":"Apache","version":"8.5.57"}},"timestamp":1602663330,"status":200}
```

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化
  - 完成Java应用监控相关配置后，日志服务自动在对应Project中生成名为*Java应用监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。
  - 完成Tomcat监控相关配置后，日志服务自动在对应Project中生成名为*Tomcat监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.8. 接入Nginx监控数据

Nginx中的自建状态页可用于监控Nginx状态。您可使用Telegraf采集Nginx监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建Nginx可视化监控方案。本文介绍如何通过日志服务来完成Nginx监控数据的采集和可视化。

前提条件

已在服务器上安装Logtail（Linux Logtail 0.16.50及以上版本）。更多信息，请参见安装Logtail（Linux系统）。

步骤1：配置Nginx Status模块

1. 执行以下命令确认Nginx已具备Status功能。

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

   如果回显信息为**with-http_stub_status_module**，表示支持Status功能。

2. 配置Nginx Status模块。

   在Nginx配置文件（默认为/etc/nginx/nginx.conf）中配置Status模块，配置示例如下所示。更多信

息，请参见Nginx Status。

```
location /private/nginx_status {
  stub_status on;
  access_log   off;
  allow 192.0.2.1;
  deny all;
}
```

- /private/nginx_status表示Nginx Status模块的URI，请根据实际情况替换。
- allow 192.0.2.1表示只允许IP地址为192.0.2.1的服务器访问Nginx Status模块，请根据实际情况替换。

3. 执行如下命令验证安装Logtail的服务器具备Nginx Status模块访问权限。

```
$curl http://192.0.2.1/private/nginx_status
```

如果回显信息如下所示，则表示已完成Nginx Status模块配置。

```
Active connections: 1
server accepts handled requests
2507455 2507455 2512972
Reading: 0 Writing: 1 Waiting: 0
```

步骤2：接入数据

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**Nginx监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**页签中，配置如下参数，然后单击**下一步**。

参数名称	说明
配置名称	Logtail配置名称。
集群名称	集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名*的标签。 **说明** 请确保该集群名称唯一，否则可能出现数据冲突。
服务器列表	单击+，添加Nginx服务器信息，相关配置项如下所示： ○ **地址**：Nginx服务器的连接地址。 ○ **端口**：Nginx服务器的连接端口号。 ○ **Path**：Nginx Status模块的URI，例如*/private/nginx_status*。如何配置Nginx Status模块，请参见步骤1：配置Nginx Status模块。 您可以根据业务需求，添加多台Nginx服务器信息。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击+，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作。更多信息，请参见查询和分析时序数据。

- 可视化

  配置完成后，日志服务自动在对应Project中生成名为*Nginx监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.9. 接入NVIDIA GPU监控数据

您可使用Telegraf采集NVIDIA GPU监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建NVIDIA GPU可视化监控方案。本文介绍如何通过日志服务来完成NVIDIA GPU监控数据的采集和可视化。

前提条件

已在Linux服务器上安装Logtail（Linux Logtail 0.16.50及以上版本）。更多信息，请参见安装Logtail（Linux系统）。

步骤1：安装NVIDIA GPU驱动

日志服务使用nvidia-smi命令采集GPU信息，该命令包含在GPU驱动程序中，因此需要先安装驱动程序。具体操作，请参见在GPU计算型实例中安装GPU驱动（Linux）。如果您使用阿里云ECS的GPU实例，则GPU实例中已默认安装驱动，可跳过此步骤。

步骤2：创建Logtail采集配置

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**NVIDIA GPU监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。

     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

   > **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。
6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
集群名	集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名*的标签。 **说明** 请确保该集群名称唯一，否则将出现数据冲突。
nvidia-smi路径	安装nvidia-smi命令的路径。默认不需要填写。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击**+**，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  完成NVIDIA GPU监控相关配置后，日志服务自动在对应Project中生成名为*NVIDIA_GPU监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.5.10. 接入MongoDB监控数据

您可使用Telegraf采集MongoDB监控数据，再通过日志服务Logtail将Telegraf数据上传到MetricStore中，搭建MongoDB可视化监控方案。本文介绍如何通过日志服务来完成MongoDB监控数据的采集和可视化。

前提条件

已在Linux服务器上安装Logtail（Linux Logtail 0.16.50及以上版本）。更多信息，请参见安装Logtail（Linux系统）。

操作步骤

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**MongoDB监控**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore。更多信息，请参见创建Project和创建MetricStore。
4. 在**创建机器组**页签中，创建机器组。
   - 如果您已有可用的机器组，请单击**使用现有机器组**。
   - 如果您还没有可用的机器组，请执行以下操作（以ECS为例）：
     a. 选择ECS实例安装Logtail。更多信息，请参见安装Logtail（ECS实例）。

        如果已在ECS上安装Logtail，请单击**确认安装完毕**。

        > **说明** 如果是自建集群、其他云厂商服务器，需要手动安装Logtail。更多信息，请参见安装Logtail（Linux系统）。

     b. 安装完成后，单击**确认安装完毕**。
     c. 创建机器组。

        如何创建机器组，请参见创建IP地址机器组或创建用户自定义标识机器组。
5. 选中目标机器组，将该机器组从**源机器组**移动到**应用机器组**，单击**下一步**。

> **注意** 如果创建机器组后立刻应用，可能因为连接未生效，导致心跳为FAIL，您可单击**自动重试**。如果还未解决，请参见Logtail机器组无心跳进行排查。

6. 在**数据源设置**页签中，配置如下参数。

参数名称	说明
配置名称	Logtail配置名称。
集群名	集群名称。配置该参数后，日志服务会为您的数据添加*cluster=集群名*的标签。 **说明** 请确保该集群名称唯一，否则将出现数据冲突。
服务器列表	单击+，添加MongoDB服务器信息，相关配置项如下所示： ○ **地址**：MongoDB服务器的连接地址，可以为服务器IP地址、主机名、域名。 ○ **端口**：MongoDB服务器的连接端口号，默认为3717。 ○ **账户**：MongoDB服务器的用户名。 **说明** 建议创建独立用户用于监控MongoDB，并只授予监控相关权限。 ○ **密码**：MongoDB服务器的用户密码。 您可以根据业务需求，添加多台MongoDB服务器信息。
自定义标签	一个MetricStore下可创建多个Logtail配置，您可以使用**自定义标签**为通过该Logtail配置采集到的数据添加标签。 单击+，添加自定义标签，支持添加多个标签。添加的标签将加入到每一条数据中。

常见问题

如何查看Telegraf采集是否正常？

您可以在服务器上查看*/etc/ilogtail/telegraf/telegraf.log*文件中记录的日志进行判断，还可以将该日志采集到日志服务中进行查询。

后续步骤

- 查询分析

  配置完成后，Telegraf将采集到的监控数据通过Logtail上传到日志服务MetricStore中。您可以在MetricStore查询分析页面进行查询分析操作，详情请参见查询和分析时序数据。

- 可视化

  完成MongoDB监控相关配置后，日志服务自动在对应Project中生成名为*MongoDB监控_集群名称*的仪表盘，您可以直接使用该仪表盘，还可以进行告警设置等操作。

1.6. 导入云监控数据

您可以通过数据导入方式将阿里云云监控数据导入到日志服务，实现时序数据的查询分析、数据加工等操作。

前提条件

已创建MetricStore，详情请参见创建MetricStore。

导入数据

1. 登录日志服务控制台。
2. 在**接入数据**区域，选择**云监控数据**。
3. 在**选择日志空间**页签中，选择目标Project和MetricStore，单击**下一步**。

   您也可以单击**立即创建**，重新创建Project和MetricStore，详情请参见创建Project和创建MetricStore。
4. 设置导入配置，单击**下一步**。

参数	说明
配置名称	配置的名称。
是否全部导入	选择**是**，则导入所有监控项中的数据，监控项详情请参见预设监控项参考。
Namespace	当**是否全部导入**选择为否时，可在**Namespace**下拉列表中手动选择监控项。
AccessKeyId	您的阿里云账号的AccessKey ID。建议使用RAM用户，该RAM用户需具备云监控读权限（AliyunCloudMonitorReadOnlyAccess）。
AccessKeySecret	您的阿里云账号的AccessKey Secret。建议使用RAM用户，该RAM用户需具备云监控读权限（AliyunCloudMonitorReadOnlyAccess）。
导入间隔	数据导入间隔，单位：分钟。

查看导入配置

创建导入配置成功后，您可以在控制台中查看已创建的导入配置及生成的统计报表。

1. 在**Project列表**区域，单击目标Project。
2. 在**时序存储 > 时序库**页签中，单击目标Metricstore下的**数据接入 > 数据导入**。
3. 单击目标配置名称。
4. 在**导入配置概览**页面，查看导入配置的基本信息和统计报表。

相关操作

在**导入配置概览**页面，您还可以进行如下操作：

- 修改配置

  单击**修改配置**，修改导入配置的相关配置，具体配置请参见设置导入配置。
- 删除配置

  单击**删除配置**，删除该导入配置。

> 注意 删除后不可恢复，请谨慎操作。

1.7. 通过SDK写入时序数据

日志服务支持通过SDK写入时序数据，本文列举了Java、Golang和Python语言的SDK demo。

> 说明
> - 使用SDK写入时序数据时，需遵循时序数据格式。
> - 尽可能使用Producer Library发送数据，目前支持ProducerLibrary的语言有Java、Golang和C。使用其他语言时，可以将多条数据放到同一个LogGroup中发送，尽可能减少网络请求次数。
> - __time_nano__字段支持秒、毫秒、微秒和纳秒。

Java SDK示例

更多信息，请参见Aliyun LOG Java Producer。

```
import com.aliyun.openservices.aliyun.log.producer.Callback;
import com.aliyun.openservices.aliyun.log.producer.LogProducer;
import com.aliyun.openservices.aliyun.log.producer.Producer;
import com.aliyun.openservices.aliyun.log.producer.ProducerConfig;
import com.aliyun.openservices.aliyun.log.producer.ProjectConfig;
import com.aliyun.openservices.aliyun.log.producer.Result;
import com.aliyun.openservices.aliyun.log.producer.errors.ProducerException;
import com.aliyun.openservices.log.common.LogItem;
import java.util.*;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.atomic.AtomicInteger;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class Main {
  private static final Logger LOGGER = LoggerFactory.getLogger(Main.class);
  private static final Random random = new Random();
  public static void main(String[] args) throws InterruptedException {
    final String project = "";
    final String logStore = "";
    final String endpoint = "https://cn-hangzhou.log.aliyuncs.com";
    final String accessKeyId = "";
    final String accessKeySecret = "";
    int sendThreadCount = 8;
    final int times = 10;
    LOGGER.info(
        "project={}, logStore={}, endpoint={}, sendThreadCount={}, times={}",
        project,
        logStore,
        endpoint,
        sendThreadCount,
        times);
    ExecutorService executorService = Executors.newFixedThreadPool(sendThreadCount);
```

```
    ProducerConfig producerConfig = new ProducerConfig();
    producerConfig.setBatchSizeThresholdInBytes(3 * 1024 * 1024);
    producerConfig.setBatchCountThreshold(40960);
    final Producer producer = new LogProducer(producerConfig);
    producer.putProjectConfig(new ProjectConfig(project, endpoint, accessKeyId, accessKeySe
cret));
    final AtomicInteger successCount = new AtomicInteger(0);
    final CountDownLatch latch = new CountDownLatch(sendThreadCount);
    LOGGER.info("Test started.");
    long t1 = System.currentTimeMillis();
    Map labels = new HashMap<>();
    labels.put("test_k", "test_v");
    for (int i = 0; i < sendThreadCount; ++i) {
      executorService.submit(
          new Runnable() {
            @Override
            public void run() {
              try {
                for (int i = 0; i < times; ++i) {
                  int r = random.nextInt(times);
                  producer.send(
                      project,
                      logStore,
                      generateTopic(r),
                      generateSource(r),
                      buildLogItem("test_metric", labels, i),
                      new Callback() {
                        @Override
                        public void onCompletion(Result result) {
                          if (result.isSuccessful()) {
                            successCount.incrementAndGet();
                          }
                        }
                      });
                }
              } catch (Exception e) {
                LOGGER.error("Failed to send log, e=", e);
              } finally {
                latch.countDown();
              }
            }
          });
    }
    latch.await();
    while (true) {
      if (successCount.get() == sendThreadCount * times) {
        break;
      }
      Thread.sleep(100);
    }
    long t2 = System.currentTimeMillis();
    LOGGER.info("Test end.");
    LOGGER.info("======Summary======");
    LOGGER.info("Total count " + sendThreadCount * times + ".");
```

```
    long timeCost = t2 - t1;
    LOGGER.info("Time cost " + timeCost + " millis");
    try {
      producer.close();
    } catch (ProducerException e) {
      LOGGER.error("Failed to close producer, e=", e);
    }
    executorService.shutdown();
  }
  private static String generateTopic(int r) {
    return "topic-" + r % 5;
  }
  private static String generateSource(int r) {
    return "source-" + r % 10;
  }
  /**
   * @param metricName: the metric name, eg: http_requests_count
   * @param labels: labels map, eg: {'idc': 'idc1', 'ip': '1.2.3.4', 'hostname': 'appserver
1'}
   * @param value: double value, eg: 1.234
   * @return LogItem
   */
  public static LogItem buildLogItem(String metricName, Map labels, double value) {
    String labelsKey = "__labels__";
    String timeKey = "__time_nano__";
    String valueKey = "__value__";
    String nameKey = "__name__";
    LogItem logItem = new LogItem();
    int timeInSec = (int)(System.currentTimeMillis() / 1000);
    logItem.SetTime(timeInSec);
    logItem.PushBack(timeKey, String.valueOf(timeInSec)+"000000");
    logItem.PushBack(nameKey, metricName);
    logItem.PushBack(valueKey, String.valueOf(value));
    // 按照字典序对labels排序，如果您的labels已排序，请忽略此步骤。
    TreeMap sortedLabels = new TreeMap<>(labels);
    StringBuilder labelsBuilder = new StringBuilder();
    Iterator> it = sortedLabels.entrySet().iterator();
    while (it.hasNext()) {
      Map.Entry entry = it.next();
      labelsBuilder.append(entry.getKey());
      labelsBuilder.append("#$#");
      labelsBuilder.append(entry.getValue());
      if (it.hasNext()) {
        labelsBuilder.append("|");
      }
    }
    logItem.PushBack(labelsKey, labelsBuilder.toString());
    return logItem;
  }
}
```

Python SDK示例

更多信息，请参见Aliyun Log Python SDK。

```
# encoding: utf-8
import time
from aliyun.log import *
def build_log_item(metric_name, labels, value):
    """
    build log item
    :param metric_name: the metric name, eg: http_requests_count
    :param labels: dict labels, eg: {'idc': 'idc1', 'ip': '1.2.3.4', 'hostname': 'appserver
1'}
    :param value: double value, eg: 1.234
    :return: LogItem
    """
    sorted_labels = sorted(labels.items())
    log_item = LogItem()
    now = int(time.time())
    log_item.set_time(now)
    contents = [('__time_nano__', str(now)), ('__name__', metric_name), ('__value__', str(v
alue))]
    labels_str = ''
    for i, kv in enumerate(sorted_labels):
        labels_str += kv[0]
        labels_str += '#$#'
        labels_str += kv[1]
        if i < len(sorted_labels) - 1:
            labels_str += '|'
    contents.append(('__labels__', labels_str))
    log_item.set_contents(contents)
    return log_item
def main():
    endpoint = 'cn-hangzhou.log.aliyuncs.com'
    accessKeyId = ''
    accessKey = ''
    project = ''
    logstore = ''
    client = LogClient(endpoint, accessKeyId, accessKey)
    item1 = build_log_item('test1', {'k1': 'v1'}, 1)
    item2 = build_log_item('test2', {'k2': 'v2'}, 2)
    item3 = build_log_item('test3', {'k3': 'v3'}, 3)
    items = [item1, item2, item3]
    res = client.put_logs(PutLogsRequest(project=project, logstore=logstore, logitems=items
))
    res.log_print()
if __name__ == '__main__':
    main()
```

GO SDK示例

更多信息，请参见Aliyun Log Golang Producer。

```
package main
import (
   "fmt"
```

```
    sls "github.com/aliyun/aliyun-log-go-sdk"
    "github.com/aliyun/aliyun-log-go-sdk/producer"
    "github.com/golang/protobuf/proto"
    "os"
    "os/signal"
    "sort"
    "strconv"
    "sync"
    "time"
)
func buildLogItem(metricName string, labels map[string]string, value float64) *sls.Log {
    now := uint32(time.Now().Unix())
    log := &sls.Log{Time: proto.Uint32(now)}
    var contents []*sls.LogContent
    contents = append(contents, &sls.LogContent{
        Key:   proto.String("__time_nano__"),
        Value: proto.String(strconv.FormatInt(int64(now), 10)),
    })
    contents = append(contents, &sls.LogContent{
        Key:   proto.String("__name__"),
        Value: proto.String(metricName),
    })
    contents = append(contents, &sls.LogContent{
        Key:   proto.String("__value__"),
        Value: proto.String(strconv.FormatFloat(value, 'f', 6, 64)),
    })
    keys := make([]string, 0, len(labels))
    for k := range labels {
        keys = append(keys, k)
    }
    sort.Strings(keys)
    labelsStr := ""
    for i, k := range keys {
        labelsStr += k
        labelsStr += "#$#"
        labelsStr += labels[k]
        if i < len(keys) - 1 {
            labelsStr += "|"
        }
    }
    contents = append(contents, &sls.LogContent{Key: proto.String("__labels__"), Value: prot
o.String(labelsStr)})
    log.Contents = contents
    return log
}
func main() {
    project := ""
    logstore := ""
    producerConfig := producer.GetDefaultProducerConfig()
    producerConfig.Endpoint = "https://cn-hangzhou.log.aliyuncs.com"
    producerConfig.AccessKeyID = ""
    producerConfig.AccessKeySecret = ""
    producerInstance := producer.InitProducer(producerConfig)
    ch := make(chan os.Signal)
```

```
    signal.Notify(ch)
    producerInstance.Start()
    var m sync.WaitGroup
    for i := 0; i < 10; i++ {
        m.Add(1)
        go func() {
            defer m.Done()
            for i := 0; i < 1000; i++ {
                // GenerateLog  is producer's function for generating SLS format logs
                // GenerateLog has low performance, and native Log interface is the best choice
for high performance.
                log := buildLogItem("test_metric", map[string]string{"test_k":"test_v"}, float6
4(i))
                err := producerInstance.SendLog(project, logstore, "topic", "127.0.0.1", log)
                if err != nil {
                    fmt.Println(err)
                }
            }
        }()
    }
    m.Wait()
    fmt.Println("Send completion")
    if _, ok := <-ch; ok {
        fmt.Println("Get the shutdown signal and start to shut down")
        producerInstance.Close(60000)
    }
}
```

2.查询与分析

2.1. 时序数据查询分析简介

本文介绍时序数据的查询分析语法及使用限制。

日志服务提供如下两种时序数据查询分析方式：

- SQL查询分析：使用SQL语法，根据时序数据的编码方式进行查询分析。
- SQL + PromQL查询分析：使用PromQL（Prometheus的查询语言）简化对时序数据的查询分析，并使用SQL语法进行嵌套查询。其中PromQL语法请参见Prometheus官方文档。

SQL查询分析

SQL查询分析语句示例如下所示：

- 查询分析全部数据

```
*| SELECT * FROM "my_metric_store.prom" WHERE __name__ != ''
```

- 查询__labels__, 'domain'值为*www.example.com*的数据，并对__value__字段进行求和计算。

```
*| SELECT sum(__value__) FROM "my_metric_store.prom" WHERE element_at(__labels__, 'domain
')='www.example.com'
```

- 查询__labels__, 'domain'值为*www.example.com*的数据，并对__value__字段进行求和计算以及对数据按小时聚合。

```
*| SELECT sum(__value__),date_trunc('hour', __time_nano__/1000000) as t
FROM "my_metric_store.prom"
WHERE element_at(__labels__, 'domain')='www.example.com'
GROUP BY t
ORDER BY t DESC
```

SQL查询分析语句相关说明如下所示：

- 时序数据的SQL查询分析语法与日志查询分析语法一致。更多信息，请参见分析语法。但在时序数据的SQL查询分析语法中，FROM的表名只能为{metrics_store_name}.prom，其中*{metrics_store_name}*为您已创建的MetricStore名称。

  > **说明** 请保留表名两端的双引号。

- __labels__可使用element_at()函数获取其中某个Key的值，例如**element_at(__labels__, 'key')**。
- 表结构请参见编码方式。

SQL+PromQL查询分析

通过SQL+PromQL查询分析方式，不仅可以使用PromQL语法，还可以使用日志服务提供的机器学习语法、安全检测函数等高级功能。

> 注意
> - 当使用SQL+PromQL查询分析时序数据时，您FROM的表名固定为metrics。
> - PromQL函数的详细接口特性和说明，请参见Prometheus官方文档。

日志服务提供5个PromQL函数，其中promql_query、promql_labels、promql_label_values和promql_series函数只能在MetricStore的查询和分析页面中执行。详细说明如下表所示。

函数名	说明	样例
promql_query(string)	即时查询分析，查询分析离结束时间最近的数据。该函数对应Prometheus的/query API，参数为query=<string>。	*| SELECT promql_query('up') FROM metrics
promql_query_range(string, string)	查询分析一定时间范围内的数据。对应Prometheus的/query_range API，参数为query=<string>、step=<duration>。	*| SELECT promql_query_range('up', '5m') FROM metrics
promql_labels()	返回所有的Label Key。	*| SELECT promql_labels() FROM metrics
promql_label_values(string)	返回某个Label的值。	*| SELECT promql_label_values('__name__') FROM metrics
promql_series(string)	返回匹配的时间序列。	*| SELECT promql_series('up') FROM metrics

PromQL函数相当于UDTF，即返回一个表。

- promql_query(string)、promql_query_range(string, string)函数返回的表的结构如下表所示。

字段名	字段类型	说明
metric	varchar	时序名称。如果查询分析中使用了Group By语法，那么该值可能为空。
labels	map<varchar, varchar>	Lables信息，Map类型。
time	bigint	时间。
value	double	某个时间点对应的值。

- promql_labels()、promql_label_values(string)函数返回的表的结构如下表所示。

字段	字段类型	说明
label	varchar	Label Key

- promql_series(string)函数返回的表的结构如下表所示。

字段	字段类型	说明
series	map<varchar, varchar>	时间序列

使用限制

- MetricStore仅支持Prometheus查询分析API（例如/query API、/query_range API等），其它API如/admin API、/alerts API、/rules API等均不支持。
- 当使用SQL+PromQL查询分析时，最多返回11000个时间点。
- 当使用SQL+PromQL查询分析时，您的Metric name和Label命名应符合命名规范。更多信息，请参见时序标识。

2.2. 查询和分析时序数据

本文介绍如何在日志服务MetricStore中查询和分析时序数据以及设置时序图图例名称等相关操作。

前提条件

已采集到时序数据。

操作步骤

> **说明** 在日志服务MetricStore中的时序查询页面中，只支持SQL+PromQL查询语法。如果您需要使用标准SQL语法，请单击**查询页面**跳转到查询页面。

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**时序存储 > 时序库**页签中，单击目标MetricStore。
4. 在页面右上角，单击**15分钟（相对）**，设置查询和分析的时间范围。

   您可以选择相对时间、整点时间和自定义时间范围。

   > **说明** 查询和分析结果相对于指定的时间范围来说，有1min以内的误差。

5. 在**查询配置**页签中，执行查询和分析操作。

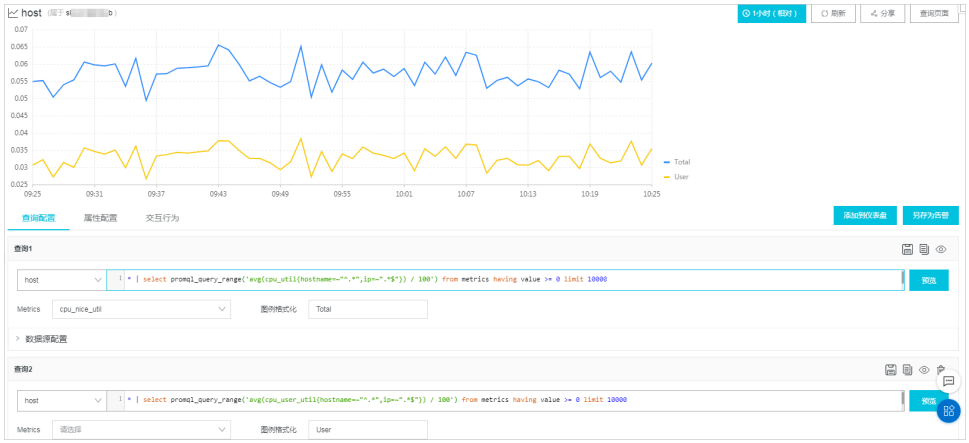
   您可以通过如下方式输入查询和分析语句：

   - 直接输入查询和分析语句，单击**预览**。

     您还可以单击**添加查询分析**或图标，新增一个查询和分析窗口，然后输入查询和分析语句，单击**预览**。

     日志服务支持在时序图中叠加显示多个查询结果。
   - 在**Metrics**下拉列表中，选择对应的监控项，自动生成查询和分析语句，单击**预览**。

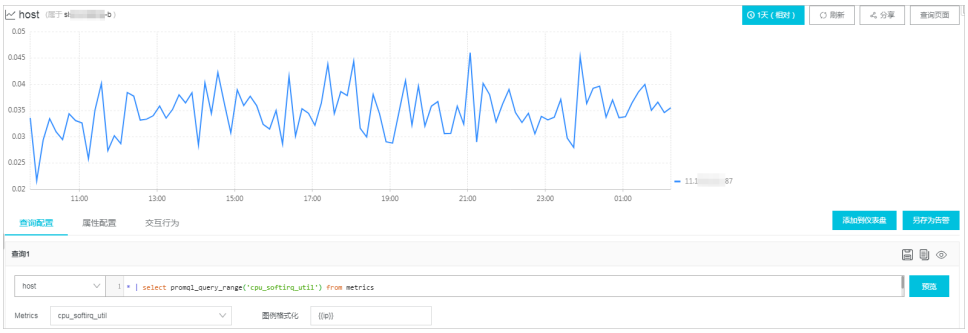
     您还可以修改自动生成的查询和分析语句。



设置图例格式化

在**查询配置**页签中，执行查询和分析操作后，您可以设置时序图图例的显示名称。

默认由Metric name和Labels拼接组成时序图图例名称，日志服务支持使用魔法变量引用Labels中指定的值，格式为{{值名称}}。例如Labels为{ip="192.0.2.1"}，则将**图例格式化**设置为{{ip}}后，时序图图例显示为192.0.2.1。

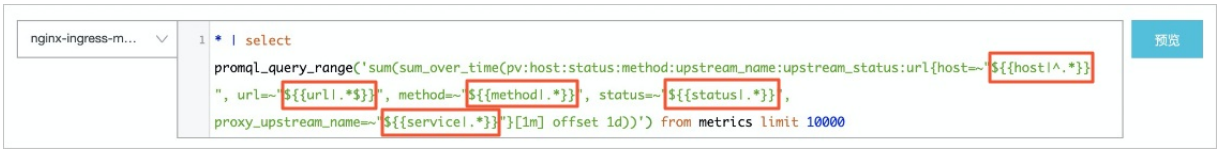


设置占位符变量

日志服务支持在查询和分析语句中设置占位符变量，当存在图表的下钻行为是跳转到这个图表所在的仪表盘，那么当变量名一致的情况下，会将单击触发下钻的数据替换为此处设置的占位符变量，重新执行分析。占位符变量格式为 `${{变量名|默认值}}` ，例如 `host=~"^.*"` 可设置为 `host=~"${{host|^.*}}"` 。

示例：在如下查询和分析语句中，将host、url、method、status、proxy_upstream_name字段的值设置为占位符变量，设置结果如下图所示。

```
* | select promql_query_range('sum(sum_over_time(pv:host:status:method:upstream_name:upstream_status:url{host=~"^.*", url=~".*$", method=~".*", status=~".*", proxy_upstream_name=~".*"}[1m]))') from metrics limit 10000
```



相关操作

操作	说明
保存快速查询	在**查询配置**页签中，单击  图标，将某一查询和分析语句保存为快速查询语句。 具体操作，请参见快速查询。
复制查询和分析	在**查询配置**页签中，单击  图标，复制当前的查询和分析，生成一个新增的查询和分析窗口。
查看原始数据	在**查询配置**页签中，单击  图标，查看原始的时序数据。
配置时序图属性	在**属性配置**页签中，配置时序图属性。具体操作，请参见时序图。
下钻分析	在**交互行为**页签中，配置下钻分析。具体操作，请参见交互事件。
添加到仪表盘	单击**添加到仪表盘**，可将查询和分析结果添加到仪表盘中。
另存为告警	单击**另存为告警**，可为查询和分析结果设置告警。具体操作，请参见设置告警。
刷新数据	通过手动或自动两种方式刷新MetricStore。 • 单击**刷新 > 仅一次**，表示立即刷新一次MetricStore。 • 单击**刷新 > 自动刷新**，表示按照指定的时间间隔自动刷新MetricStore。 时间间隔可设置为15秒、60秒、5分钟或15分钟。
分享	单击**分享**，可复制当前MetricStore查询和分析页面的链接，您可以将该链接发送给有查看该MetricStore权限的其他用户。其他用户看到的MetricStore查询和分析页面会保留您的一系列设置。
跳转到查询页面	单击**查询页面**，跳转到查询页面。

3.可视化

3.1. 时序图

本文介绍时序图操作步骤。

简介

时序图是专门为MetricStore定制的可视化图表，支持Prometheus查询数据结果的可视化，并支持多个查询结果同时显示。

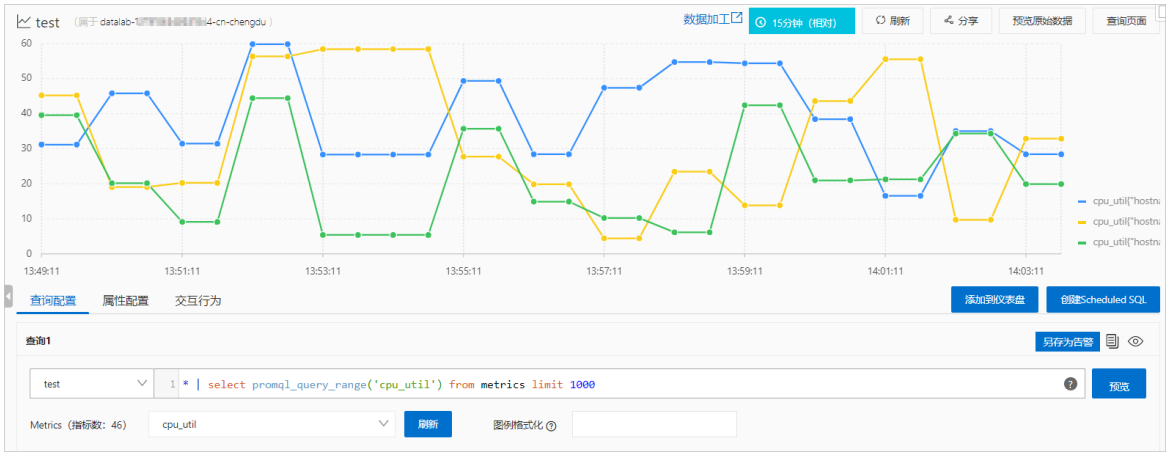
操作步骤

1. 登录日志服务控制台。
2. 在Project列表区域，单击目标Project。
3. 在**时序存储 > 时序库**页签中，单击目标MetricStore。
4. 在**属性配置**页签中，配置图表属性。

参数	说明
自动补点	开启**自动补点**开关后，如果时序数据中存在数据缺失问题，日志服务会自动补齐缺失数据。
Y轴最小值	设置Y轴最小值。
Y轴最大值	设置Y轴最大值。
左Y轴格式化	将左Y轴数据按照指定格式进行显示。
X轴刻度密度	设置X轴刻度密度，取值范围为3~30。
线形	时序图显示格式，可以为直线或曲线。
显示点	开启**显示点**开关后，在线上显示具体的数据点。
边距	坐标轴距离图表边界距离，包括上边距、下边距、右边距和左边距。

5. 在**查询配置**页签中，执行查询分析操作。

   您可以单击页面右上角的**预览原始数据**，查看已采集到的时序数据，例如 `__labels__:hostname#$#hostname1|ip#$#192.0.2.0 __time_nano__:1644309671000000000 __value__:52.71 __name__:cpu_util` 。日志服务将根据该时序数据以及您所选择的指标，绘制时序图。例如您要查询不同主机的CPU使用率使用情况，则可以选择cpu_util指标，日志服务将展示不同主机CPU使用率的时序图。



3.2. 时序数据对接Grafana

日志服务MetricStore提供了兼容Prometheus的查询接口，您可以直接通过Prometheus数据源方式对接到Grafana进行可视化演示。本文介绍配置Prometheus监控数据为Grafana数据源的操作步骤。

前提条件

- 已安装Grafana。详情请参见安装Grafana。
- 已接入时序数据。详情请参见通过Remote Write协议接入Prometheus监控数据。

对接Grafana

1. 登录Grafana。
2. 在左侧导航栏，单击**Configuration > Data Sources**。
3. 在**Data Sources**页签，单击**Add data source**。
4. 选择**Prometheus**，单击**Select**。
5. 在**Settings**页签，请您参考如下说明配置数据源。

参数	说明
Name	请您自定义一个数据源的名称，例如Prometheus-01。
HTTP	○ **URL**：日志服务MetricStore的URL，格式为https://{project}.{sls-enpoint}/prometheus/{project}/{metricstore}。其中*{sls-enpoint}*为Project所在地域的Endpoint，详情请参见服务入口，*{project}*和*{metricstore}*为您已创建的日志服务的Project和Metricstore，请根据实际值替换。例如：https://sls-prometheus-test.cn-hangzhou.log.aliyuncs.com/prometheus/sls-prometheus-test/prometheus。 **说明** 为保证传输安全性，请务必设置为`https`。 ○ **Whitelisted Cookies**：添加访问白名单，可选。
Auth	打开**Basic auth**开关。

参数	说明
Basic Auth Details	○ User为阿里云账号AccessKeyID。 ○ Password为阿里云账号AccessKeySecret。 建议您使用仅具备指定Project只读权限的RAM用户账号，详情请参见指定Project只读授权策略。

6. 单击Save & Test。

导入日志服务Grafana模板

您可以在Grafana模板市场中查找日志服务提供的可视化模板并一键导入到您的Grafana中，进行可视化展示。

1. 复制Grafana模板ID。
   i. 登录Grafana模板市场。
   ii. 单击您要导入的模板。
   iii. 在页面右侧，单击Copy ID to Clipboard。
2. 登录Grafana。
3. 在左侧导航栏中，单击Create > Import。
4. 在Grafana.com Dashboard文本框中输入您在步骤1中复制的Grafana模板ID。

   配置完成后，单击空白处，即可进入配置页面，配置数据源。
5. 配置数据源。

   此处需配置为您在对接Grafana中添加的数据源。不同仪表盘对应的数据源参数不同，可能为telegraf、host等。
6. 单击Import。

Prometheus查询API

日志服务提供了兼容Prometheus的查询API，可直接配置日志服务作为Grafana的Prometheus数据源，同时也支持用各类Prometheus API直接访问。支持的API如下：

API名称	示例
Instant queries	`GET /api/v1/query` `POST /api/v1/query`
Range queries	`GET /api/v1/query_range` `POST /api/v1/query_range`
Getting label names	`GET /api/v1/labels` `POST /api/v1/labels`

API名称	示例
Querying label values	`GET /api/v1/label/<label_name>/values`
Finding series by label matchers	`GET /api/v1/series` `POST /api/v1/series`

4.最佳实践

4.1. 使用Prometheus采集Kubernetes监控数据

本文介绍如何在Kubernetes上部署Prometheus，将监控数据采集到日志服务MetricStore中，并将日志服务MetricStore对接到Grafana实现监控数据可视化展示。

前提条件

- 已拥有Kubernetes集群，集群版本在1.10以上。
- 已创建MetricStore。更多信息，请参见创建MetricStore。
- 已安装Grafana。更多信息，参见安装Grafana。

背景信息

Prometheus作为面向云原生的监控软件，对Kubernetes提供了友好的支持。在Kubernetes中，几乎所有的组件都提供了Prometheus的指标接口，因此Prometheus基本成为Kubernetes监控的事实标准。

Grafana是一个开源的度量分析与可视化套件，兼容所有的Prometheus仪表盘模板。日志服务支持Grafana访问时序数据，您可直接将日志服务MetricStore作为Grafana的Prometheus数据源进行接入，实现时序数据可视化展示。

在自建Kubernetes上安装Prometheus

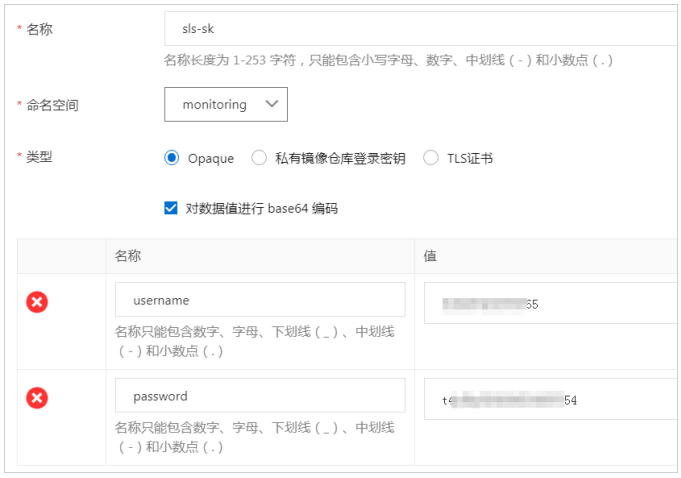
如果您使用自建Kubernetes，推荐以注册集群的方式接入到阿里云。更多信息，请参见注册集群概述。注册好后按照阿里云Kubernetes安装方式安装Prometheus。更多信息，请参见阿里云Kubernetes安装方式。如果您不使用注册集群方式，可通过Helm安装包安装Prometheus，安装前需先创建保密字典并调整默认配置。

在阿里云Kubernetes上安装Prometheus

如果您使用阿里云Kubernetes，可直接在应用目录中安装并配置Prometheus将数据存储到日志服务。

1. 登录容器服务管理控制台。
2. 在**集群**页面，单击目标集群。
3. 创建命名空间。
   i. 在左侧导航栏中，单击**命名空间**。
   ii. 单击**创建**。
   iii. 配置名称为monitoring，并单击**确定**。
4. 创建保密字典。
   i. 在左侧导航栏中，选择**配置管理 > 保密字典**。
   ii. 单击**创建**。

iii. 配置如下参数，单击**确定**。



参数	说明
名称	配置为**sls-ak**。
命名空间	选中您在步骤2中创建的命名空间，即**monitoring**。
类型	选中**Opaque**，并添加如下两个键值对： ■ **名称**为**username**，**值**为您的RAM用户的AccessKey ID。 ■ **名称**为**password**，**值**为您的RAM用户的AccessKey Secret。 建议您使用只具备日志服务Project写入权限的RAM用户的AccessKey。更多信息，请参见授予指定Project写入权限。

5. 创建PrometheusOperator。
   i. 在左侧导航中，选择**市场 > 应用目录**。
   ii. 单击**ack-prometheus-operator**。
   iii. 在**参数**页签下，修改其中的配置项。
   - 调整prometheusSpec下的retention，建议修改为1d或12h。
   - 替换其中的remoteWrite配置。更多信息，请参见RemoteWrite配置。

     remoteWrite配置中的url为日志服务Metricstore的URL，请根据实际值替换。格式为https://{project}.{sls-enpoint}/prometheus/{project}/{metricstore}/api/v1/write。其中*{sls-enpoint}*为日志服务的Endpoint。更多信息，请参见服务入口，*{project}*和*{metricstore}*为您已创建的日志服务的Project和Metricstore。

     > **说明** 如果您是在阿里云内网，请优先使用内网域名。

- 如果Prometheus数据量较大，可修改queue_config配置，建议修改为：

```
batchSendDeadline: 30s
capacity: 204800
maxBackoff: 5s
maxSamplesPerSend: 4096
minBackoff: 100ms
minShards: 100
maxShards: 2048
```

> **注意** 为减少上报的数据量，建议在remoteWrite中增加writeRelabelConfigs去除不重要的指标，默认配置中已为您去除Kubernetes监控中不常用的指标。

```
      remoteWrite:
      - basicAuth:
          username:
            name: sls-ak
            key: username
          password:
            name: sls-ak
            key: password
        queueConfig:
          batchSendDeadline: 30s
          capacity: 204800
          maxBackoff: 5s
          maxSamplesPerSend: 4096
          minBackoff: 100ms
          minShards: 100
          maxShards: 2048
        writeRelabelConfigs:
        - action: drop
          regex: APIServiceOpenAPIAggregationControllerQueue1_adds|APIServiceOpenAP
IAggregationControllerQueue1_depth|APIServiceOpenAPIAggregationControllerQueue1_que
ue_latency|APIServiceOpenAPIAggregationControllerQueue1_queue_latency_count|APIServ
iceOpenAPIAggregationControllerQueue1_queue_latency_sum|APIServiceOpenAPIAggregatio
nControllerQueue1_retries|APIServiceOpenAPIAggregationControllerQueue1_work_duratio
n|APIServiceOpenAPIAggregationControllerQueue1_work_duration_count|APIServiceOpenAP
IAggregationControllerQueue1_work_duration_sum|APIServiceRegistrationController_add
s|APIServiceRegistrationController_depth|APIServiceRegistrationController_queue_lat
ency|APIServiceRegistrationController_queue_latency_count|APIServiceRegistrationCon
troller_queue_latency_sum|APIServiceRegistrationController_retries|APIServiceRegist
rationController_work_duration|APIServiceRegistrationController_work_duration_count
|APIServiceRegistrationController_work_duration_sum|AvailableConditionController_ad
ds|AvailableConditionController_depth|AvailableConditionController_queue_latency|Av
ailableConditionController_queue_latency_count|AvailableConditionController_queue_l
atency_sum|AvailableConditionController_retries|AvailableConditionController_work_d
uration|AvailableConditionController_work_duration_count|AvailableConditionControll
er_work_duration_sum|DiscoveryController_adds|DiscoveryController_depth|DiscoveryCo
ntroller_queue_latency|DiscoveryController_queue_latency_count|DiscoveryController_
queue_latency_sum|DiscoveryController_retries|DiscoveryController_work_duration|Dis
coveryController_work_duration_count|DiscoveryController_work_duration_sum|admissio
n_quota_controller_adds|admission_quota_controller_depth|admission_quota_controller
_queue_latency|admission_quota_controller_queue_latency_count|admission_quota_contr
```

```
oller_queue_latency_sum|admission_quota_controller_work_duration|admission_quota_co
ntroller_work_duration_count|admission_quota_controller_work_duration_sum|alertmana
ger_alerts|alertmanager_alerts_invalid_total|alertmanager_alerts_received_total|ale
rtmanager_build_info|alertmanager_cluster_failed_peers|alertmanager_cluster_health_
score|alertmanager_cluster_members|alertmanager_cluster_messages_pruned_total|alert
manager_cluster_messages_queued|alertmanager_cluster_messages_received_size_total|a
lertmanager_cluster_messages_received_total|alertmanager_cluster_messages_sent_size
_total|alertmanager_cluster_messages_sent_total|alertmanager_cluster_peers_joined_t
otal|alertmanager_cluster_peers_left_total|alertmanager_cluster_peers_update_total|
alertmanager_cluster_reconnections_failed_total|alertmanager_cluster_reconnections_
total|alertmanager_cluster_refresh_join_failed_total|alertmanager_cluster_refresh_j
oin_total|alertmanager_config_hash|alertmanager_config_last_reload_success_timestam
p_seconds|alertmanager_config_last_reload_successful|alertmanager_http_request_dura
tion_seconds_bucket|alertmanager_http_request_duration_seconds_count|alertmanager_h
ttp_request_duration_seconds_sum|alertmanager_http_response_size_bytes_bucket|alert
manager_http_response_size_bytes_count|alertmanager_http_response_size_bytes_sum|al
ertmanager_nflog_gc_duration_seconds|alertmanager_nflog_gc_duration_seconds_count|a
lertmanager_nflog_gc_duration_seconds_sum|alertmanager_nflog_gossip_messages_propag
ated_total|alertmanager_nflog_queries_total|alertmanager_nflog_query_duration_secon
ds_bucket|alertmanager_nflog_query_duration_seconds_count|alertmanager_nflog_query_
duration_seconds_sum|alertmanager_nflog_query_errors_total|alertmanager_nflog_snaps
hot_duration_seconds|alertmanager_nflog_snapshot_duration_seconds_count|alertmanage
r_nflog_snapshot_duration_seconds_sum|alertmanager_nflog_snapshot_size_bytes|alertm
anager_notification_latency_seconds_bucket|alertmanager_notification_latency_second
s_count|alertmanager_notification_latency_seconds_sum|alertmanager_notifications_fa
iled_total|alertmanager_notifications_total|alertmanager_oversize_gossip_message_du
ration_seconds_bucket|alertmanager_oversize_gossip_message_duration_seconds_count|a
lertmanager_oversize_gossip_message_duration_seconds_sum|alertmanager_oversized_gos
sip_message_dropped_total|alertmanager_oversized_gossip_message_failure_total|alert
manager_oversized_gossip_message_sent_total|alertmanager_peer_position|alertmanager
_silences|alertmanager_silences_gc_duration_seconds|alertmanager_silences_gc_durati
on_seconds_count|alertmanager_silences_gc_duration_seconds_sum|alertmanager_silence
s_gossip_messages_propagated_total|alertmanager_silences_queries_total|alertmanager
_silences_query_duration_seconds_bucket|alertmanager_silences_query_duration_second
s_count|alertmanager_silences_query_duration_seconds_sum|alertmanager_silences_quer
y_errors_total|alertmanager_silences_snapshot_duration_seconds|alertmanager_silence
s_snapshot_duration_seconds_count|alertmanager_silences_snapshot_duration_seconds_s
um|alertmanager_silences_snapshot_size_bytes|apiserver_admission_controller_admissi
on_latencies_seconds_bucket|apiserver_admission_controller_admission_latencies_seco
nds_count|apiserver_admission_controller_admission_latencies_seconds_sum|apiserver_
admission_step_admission_latencies_seconds_bucket|apiserver_admission_step_admissio
n_latencies_seconds_count|apiserver_admission_step_admission_latencies_seconds_sum|
apiserver_admission_step_admission_latencies_seconds_summary|apiserver_admission_st
ep_admission_latencies_seconds_summary_count|apiserver_admission_step_admission_lat
encies_seconds_summary_sum|apiserver_admission_webhook_admission_latencies_seconds_
bucket|apiserver_admission_webhook_admission_latencies_seconds_count|apiserver_admi
ssion_webhook_admission_latencies_seconds_sum|apiserver_audit_event_total|apiserver
_audit_level_total|apiserver_client_certificate_expiration_seconds_bucket|apiserver
_client_certificate_expiration_seconds_count|apiserver_client_certificate_expiratio
n_seconds_sum|apiserver_current_inflight_requests|apiserver_registered_watchers|api
server_request_latencies_bucket|apiserver_request_latencies_count|apiserver_request
_latencies_sum|apiserver_request_latencies_summary_count|apiserver_request_latencie
s_summary_sum|apiserver_response_sizes_bucket|apiserver_response_sizes_count|apiser
ver_response_sizes_sum|apiserver_storage_data_key_generation_failures_total|apiserv
```

```
er_storage_data_key_generation_latencies_microseconds_bucket|apiserver_storage_data
_key_generation_latencies_microseconds_count|apiserver_storage_data_key_generation_
latencies_microseconds_sum|apiserver_storage_envelope_transformation_cache_misses_t
otal|authenticated_user_requests|autoregister_adds|autoregister_depth|autoregister_
queue_latency|autoregister_queue_latency_count|autoregister_queue_latency_sum|autor
egister_retries|autoregister_work_duration|autoregister_work_duration_count|autoreg
ister_work_duration_sum|cadvisor_version_info|container_cpu_cfs_periods_total|conta
iner_cpu_load_average_10s|container_cpu_system_seconds_total|container_cpu_user_sec
onds_total|container_fs_inodes_free|container_fs_inodes_total|container_fs_io_curre
nt|container_fs_io_time_seconds_total|container_fs_io_time_weighted_seconds_total|c
ontainer_fs_reads_merged_total|container_fs_reads_total|container_fs_sector_reads_t
otal|container_fs_sector_writes_total|container_fs_writes_merged_total|container_fs
_writes_total|container_last_seen|container_memory_cache|container_memory_failcnt|c
ontainer_memory_failures_total|container_memory_mapped_file|container_memory_max_us
age_bytes|container_memory_swap|container_network_receive_errors_total|container_ne
twork_receive_packets_dropped_total|container_network_receive_packets_total|contain
er_network_transmit_errors_total|container_network_transmit_packets_dropped_total|c
ontainer_scrape_error|container_spec_cpu_period|container_spec_cpu_shares|container
_spec_memory_reservation_limit_bytes|container_spec_memory_swap_limit_bytes|contain
er_start_time_seconds|container_tasks_state|coredns_autopath_success_count_total|co
redns_forward_request_count_total|coredns_forward_request_duration_seconds_bucket|c
oredns_forward_request_duration_seconds_count|coredns_forward_request_duration_seco
nds_sum|coredns_forward_response_rcode_count_total|coredns_forward_sockets_open|cor
edns_plugin_enabled|coredns_proxy_request_count_total|coredns_proxy_request_duratio
n_seconds_bucket|coredns_proxy_request_duration_seconds_count|coredns_proxy_request
_duration_seconds_sum|crdEstablishing_adds|crdEstablishing_depth|crdEstablishing_qu
eue_latency_count|crdEstablishing_queue_latency_sum|crdEstablishing_retries|crdEsta
blishing_work_duration_count|crdEstablishing_work_duration_sum|etcd_helper_cache_en
try_count|etcd_helper_cache_hit_count|etcd_helper_cache_miss_count|etcd_object_coun
ts|etcd_request_cache_add_latencies_summary_count|etcd_request_cache_add_latencies_
summary_sum|etcd_request_cache_get_latencies_summary_count|etcd_request_cache_get_l
atencies_summary_sum|get_token_count|get_token_fail_count|go_memstats_heap_released
_bytes_total|grafana_alerting_active_alerts|grafana_alerting_execution_time_millise
conds_count|grafana_alerting_execution_time_milliseconds_sum|grafana_api_admin_user
_created_total|grafana_api_dashboard_get_milliseconds_count|grafana_api_dashboard_g
et_milliseconds_sum|grafana_api_dashboard_save_milliseconds_count|grafana_api_dashb
oard_save_milliseconds_sum|grafana_api_dashboard_search_milliseconds_count|grafana_
api_dashboard_search_milliseconds_sum|grafana_api_dashboard_snapshot_create_total|g
rafana_api_dashboard_snapshot_external_total|grafana_api_dashboard_snapshot_get_tot
al|grafana_api_dataproxy_request_all_milliseconds_count|grafana_api_dataproxy_reque
st_all_milliseconds_sum|grafana_api_login_oauth_total|grafana_api_login_post_total|
grafana_api_models_dashboard_insert_total|grafana_api_org_create_total|grafana_api_
response_status_total|grafana_api_user_signup_completed_total|grafana_api_user_sign
up_invite_total|grafana_api_user_signup_started_total|grafana_aws_cloudwatch_get_me
tric_data_total|grafana_aws_cloudwatch_get_metric_statistics_total|grafana_aws_clou
dwatch_list_metrics_total|grafana_build_info|grafana_db_datasource_query_by_id_tota
l|grafana_info|grafana_instance_start_total|grafana_page_response_status_total|graf
ana_proxy_response_status_total|grafana_stat_active_users|grafana_stat_total_orgs|g
rafana_stat_total_playlists|grafana_stat_total_users|grafana_stat_totals_dashboard|
grpc_client_handled_total|grpc_client_msg_received_total|grpc_client_msg_sent_total
|grpc_client_started_total|http_request_duration_microseconds|http_request_duration
_microseconds_count|http_request_duration_microseconds_sum|http_request_duration_mi
lliseconds_count|http_request_duration_milliseconds_sum|http_request_size_bytes|htt
p_request_size_bytes_count|http_request_size_bytes_sum|http_request_total|http_requ
```

ests_total|http_response_size_bytes|http_response_size_bytes_count|http_response_size_bytes_sum|kube_configmap_created|kube_configmap_info|kube_configmap_metadata_resource_version|kube_daemonset_labels|kube_daemonset_metadata_generation|kube_daemonset_status_current_number_scheduled|kube_daemonset_status_number_available|kube_daemonset_status_number_misscheduled|kube_daemonset_status_number_unavailable|kube_daemonset_updated_number_scheduled|kube_deployment_labels|kube_deployment_spec_paused|kube_deployment_spec_strategy_rollingupdate_max_surge|kube_endpoint_address_available|kube_endpoint_address_not_ready|kube_endpoint_created|kube_endpoint_info|kube_endpoint_labels|kube_ingress_created|kube_ingress_info|kube_ingress_labels|kube_ingress_metadata_resource_version|kube_job_complete|kube_job_created|kube_job_info|kube_job_labels|kube_job_owner|kube_job_spec_completions|kube_job_spec_parallelism|kube_job_status_active|kube_job_status_completion_time|kube_job_status_failed|kube_job_status_start_time|kube_job_status_succeeded|kube_namespace_annotations|kube_namespace_created|kube_namespace_labels|kube_namespace_status_phase|kube_node_created|kube_node_status_allocatable|kube_node_status_capacity|kube_node_status_capacity_pods|kube_pod_completion_time|kube_pod_container_resource_requests|kube_pod_container_status_last_terminated_reason|kube_pod_container_status_ready|kube_pod_container_status_terminated_reason|kube_pod_container_status_waiting_reason|kube_pod_created|kube_pod_start_time|kube_pod_status_ready|kube_pod_status_scheduled|kube_pod_status_scheduled_time|kube_replicaset_created|kube_replicaset_labels|kube_replicaset_metadata_generation|kube_replicaset_owner|kube_replicaset_spec_replicas|kube_replicaset_status_fully_labeled_replicas|kube_replicaset_status_observed_generation|kube_replicaset_status_ready_replicas|kube_replicaset_status_replicas|kube_secret_created|kube_secret_info|kube_secret_labels|kube_secret_metadata_resource_version|kube_secret_type|kube_service_created|kube_service_labels|kube_service_spec_type|kube_service_status_load_balancer_ingress|kube_statefulset_created|kube_statefulset_labels|kube_statefulset_metadata_generation|kube_statefulset_replicas|kube_statefulset_status_current_revision|kube_statefulset_status_observed_generation|kube_statefulset_status_replicas|kube_statefulset_status_replicas_current|kube_statefulset_status_replicas_ready|kube_statefulset_status_replicas_updated|kube_statefulset_status_update_revision|kubelet_certificate_manager_client_expiration_seconds|kubelet_cgroup_manager_latency_microseconds|kubelet_cgroup_manager_latency_microseconds_count|kubelet_cgroup_manager_latency_microseconds_sum|kubelet_containers_per_pod_count|kubelet_containers_per_pod_count_count|kubelet_containers_per_pod_count_sum|kubelet_docker_operations|kubelet_docker_operations_errors|kubelet_docker_operations_latency_microseconds|kubelet_docker_operations_latency_microseconds_count|kubelet_docker_operations_latency_microseconds_sum|kubelet_docker_operations_timeout|kubelet_eviction_stats_age_microseconds_count|kubelet_eviction_stats_age_microseconds_sum|kubelet_network_plugin_operations_latency_microseconds|kubelet_network_plugin_operations_latency_microseconds_count|kubelet_network_plugin_operations_latency_microseconds_sum|kubelet_node_config_error|kubelet_pleg_relist_interval_microseconds|kubelet_pleg_relist_interval_microseconds_count|kubelet_pleg_relist_interval_microseconds_sum|kubelet_pleg_relist_latency_microseconds|kubelet_pleg_relist_latency_microseconds_count|kubelet_pleg_relist_latency_microseconds_sum|kubelet_pod_start_latency_microseconds|kubelet_pod_start_latency_microseconds_count|kubelet_pod_start_latency_microseconds_sum|kubelet_pod_worker_latency_microseconds|kubelet_pod_worker_latency_microseconds_count|kubelet_pod_worker_latency_microseconds_sum|kubelet_pod_worker_start_latency_microseconds|kubelet_pod_worker_start_latency_microseconds_count|kubelet_pod_worker_start_latency_microseconds_sum|kubelet_running_container_count|kubelet_running_pod_count|kubelet_runtime_operations|kubelet_runtime_operations_errors|kubelet_runtime_operations_latency_microseconds|kubelet_runtime_operations_latency_microseconds_count|kubelet_runtime_operations_latency_microseconds_sum|kubernetes_build_info|net_conntrack_dialer_conn_attempted_total|net_conntrack_dialer_conn_closed_total|net_conntrack_dialer_conn_established_total|net_conntrack_dialer_conn_failed_total|net_conntrack_listener_conn_accepted_total|net_conntrack_listener_conn_closed_total|nginx_ingress_contro

```
ller_leader_election_status|nginx_ingress_controller_ssl_expire_time_seconds|promet
heus_api_remote_read_queries|prometheus_build_info|prometheus_config_last_reload_su
ccess_timestamp_seconds|prometheus_config_last_reload_successful|prometheus_engine_
queries|prometheus_engine_queries_concurrent_max|prometheus_engine_query_duration_s
econds|prometheus_engine_query_duration_seconds_count|prometheus_engine_query_durat
ion_seconds_sum|prometheus_http_request_duration_seconds_bucket|prometheus_http_req
uest_duration_seconds_count|prometheus_http_request_duration_seconds_sum|prometheus
_http_response_size_bytes_bucket|prometheus_http_response_size_bytes_count|promethe
us_http_response_size_bytes_sum|prometheus_notifications_alertmanagers_discovered|p
rometheus_notifications_dropped_total|prometheus_notifications_errors_total|prometh
eus_notifications_latency_seconds|prometheus_notifications_latency_seconds_count|pr
ometheus_notifications_latency_seconds_sum|prometheus_notifications_queue_capacity|
prometheus_notifications_queue_length|prometheus_notifications_sent_total|prometheu
s_operator_node_address_lookup_errors_total|prometheus_operator_reconcile_errors_to
tal|prometheus_operator_spec_replicas|prometheus_operator_triggered_total|prometheu
s_remote_storage_dropped_samples_total|prometheus_remote_storage_enqueue_retries_to
tal|prometheus_remote_storage_failed_samples_total|prometheus_remote_storage_highes
t_timestamp_in_seconds|prometheus_remote_storage_pending_samples|prometheus_remote_
storage_queue_highest_sent_timestamp_seconds|prometheus_remote_storage_remote_read_
queries|prometheus_remote_storage_retried_samples_total|prometheus_remote_storage_s
amples_in_total|prometheus_remote_storage_sent_batch_duration_seconds_bucket|promet
heus_remote_storage_sent_batch_duration_seconds_count|prometheus_remote_storage_sen
t_batch_duration_seconds_sum|prometheus_remote_storage_shard_capacity|prometheus_re
mote_storage_shards|prometheus_remote_storage_succeeded_samples_total|prometheus_ru
le_evaluation_duration_seconds|prometheus_rule_evaluation_duration_seconds_count|pr
ometheus_rule_evaluation_duration_seconds_sum|prometheus_rule_evaluation_failures_t
otal|prometheus_rule_evaluations_total|prometheus_rule_group_duration_seconds|prome
theus_rule_group_duration_seconds_count|prometheus_rule_group_duration_seconds_sum|
prometheus_rule_group_interval_seconds|prometheus_rule_group_iterations_missed_tota
l|prometheus_rule_group_iterations_total|prometheus_rule_group_last_duration_second
s|prometheus_rule_group_last_evaluation_timestamp_seconds|prometheus_rule_group_rul
es|prometheus_sd_consul_rpc_duration_seconds_count|prometheus_sd_consul_rpc_duratio
n_seconds_sum|prometheus_sd_consul_rpc_failures_total|prometheus_sd_discovered_targ
ets|prometheus_sd_dns_lookup_failures_total|prometheus_sd_dns_lookups_total|prometh
eus_sd_file_read_errors_total|prometheus_sd_file_scan_duration_seconds_count|promet
heus_sd_file_scan_duration_seconds_sum|prometheus_sd_kubernetes_cache_last_resource
_version|prometheus_sd_kubernetes_cache_list_duration_seconds_count|prometheus_sd_k
ubernetes_cache_list_duration_seconds_sum|prometheus_sd_kubernetes_cache_list_items
_count|prometheus_sd_kubernetes_cache_list_items_sum|prometheus_sd_kubernetes_cache
_list_total|prometheus_sd_kubernetes_cache_short_watches_total|prometheus_sd_kubern
etes_cache_watch_duration_seconds_count|prometheus_sd_kubernetes_cache_watch_durati
on_seconds_sum|prometheus_sd_kubernetes_cache_watch_events_count|prometheus_sd_kube
rnetes_cache_watch_events_sum|prometheus_sd_kubernetes_cache_watches_total|promethe
us_sd_kubernetes_events_total|prometheus_sd_kubernetes_http_request_duration_second
s_count|prometheus_sd_kubernetes_http_request_duration_seconds_sum|prometheus_sd_ku
bernetes_http_request_total|prometheus_sd_kubernetes_workqueue_depth|prometheus_sd_
kubernetes_workqueue_items_total|prometheus_sd_kubernetes_workqueue_latency_seconds
_count|prometheus_sd_kubernetes_workqueue_latency_seconds_sum|prometheus_sd_kuberne
tes_workqueue_longest_running_processor_seconds|prometheus_sd_kubernetes_workqueue_
unfinished_work_seconds|prometheus_sd_kubernetes_workqueue_work_duration_seconds_co
unt|prometheus_sd_kubernetes_workqueue_work_duration_seconds_sum|prometheus_sd_rece
ived_updates_total|prometheus_sd_updates_total|prometheus_target_interval_length_se
conds|prometheus_target_interval_length_seconds_count|prometheus_target_interval_le
ngth_seconds_sum|prometheus_target_scrape_pool_reloads_failed_total|prometheus_targ
```

```
et_scrape_pool_reloads_total|prometheus_target_scrape_pool_sync_total|prometheus_ta
rget_scrape_pools_failed_total|prometheus_target_scrape_pools_total|prometheus_targ
et_scrapes_cache_flush_forced_total|prometheus_target_scrapes_exceeded_sample_limit
_total|prometheus_target_scrapes_sample_duplicate_timestamp_total|prometheus_target
_scrapes_sample_out_of_bounds_total|prometheus_target_scrapes_sample_out_of_order_t
otal|prometheus_target_sync_length_seconds|prometheus_target_sync_length_seconds_co
unt|prometheus_target_sync_length_seconds_sum|prometheus_template_text_expansion_fa
ilures_total|prometheus_template_text_expansions_total|prometheus_treecache_watcher
_goroutines|prometheus_treecache_zookeeper_failures_total|prometheus_tsdb_blocks_lo
aded|prometheus_tsdb_checkpoint_creations_failed_total|prometheus_tsdb_checkpoint_c
reations_total|prometheus_tsdb_checkpoint_deletions_failed_total|prometheus_tsdb_ch
eckpoint_deletions_total|prometheus_tsdb_compaction_chunk_range_seconds_bucket|prom
etheus_tsdb_compaction_chunk_range_seconds_count|prometheus_tsdb_compaction_chunk_r
ange_seconds_sum|prometheus_tsdb_compaction_chunk_samples_bucket|prometheus_tsdb_co
mpaction_chunk_samples_count|prometheus_tsdb_compaction_chunk_samples_sum|prometheu
s_tsdb_compaction_chunk_size_bytes_bucket|prometheus_tsdb_compaction_chunk_size_byt
es_count|prometheus_tsdb_compaction_chunk_size_bytes_sum|prometheus_tsdb_compaction
_duration_seconds_bucket|prometheus_tsdb_compaction_duration_seconds_count|promethe
us_tsdb_compaction_duration_seconds_sum|prometheus_tsdb_compaction_populating_block
|prometheus_tsdb_compactions_failed_total|prometheus_tsdb_compactions_total|prometh
eus_tsdb_compactions_triggered_total|prometheus_tsdb_head_active_appenders|promethe
us_tsdb_head_chunks|prometheus_tsdb_head_chunks_created_total|prometheus_tsdb_head_
chunks_removed_total|prometheus_tsdb_head_gc_duration_seconds_count|prometheus_tsdb
_head_gc_duration_seconds_sum|prometheus_tsdb_head_max_time|prometheus_tsdb_head_ma
x_time_seconds|prometheus_tsdb_head_min_time|prometheus_tsdb_head_min_time_seconds|
prometheus_tsdb_head_samples_appended_total|prometheus_tsdb_head_series|prometheus_
tsdb_head_series_created_total|prometheus_tsdb_head_series_not_found_total|promethe
us_tsdb_head_series_removed_total|prometheus_tsdb_head_truncations_failed_total|pro
metheus_tsdb_head_truncations_total|prometheus_tsdb_lowest_timestamp|prometheus_tsd
b_lowest_timestamp_seconds|prometheus_tsdb_reloads_failures_total|prometheus_tsdb_r
eloads_total|prometheus_tsdb_size_retentions_total|prometheus_tsdb_storage_blocks_b
ytes|prometheus_tsdb_symbol_table_size_bytes|prometheus_tsdb_time_retentions_total|
prometheus_tsdb_tombstone_cleanup_seconds_bucket|prometheus_tsdb_tombstone_cleanup_
seconds_count|prometheus_tsdb_tombstone_cleanup_seconds_sum|prometheus_tsdb_vertica
l_compactions_total|prometheus_tsdb_wal_completed_pages_total|prometheus_tsdb_wal_c
orruptions_total|prometheus_tsdb_wal_fsync_duration_seconds_count|prometheus_tsdb_w
al_fsync_duration_seconds_sum|prometheus_tsdb_wal_page_flushes_total|prometheus_tsd
b_wal_truncate_duration_seconds_count|prometheus_tsdb_wal_truncate_duration_seconds
_sum|prometheus_tsdb_wal_truncations_failed_total|prometheus_tsdb_wal_truncations_t
otal|prometheus_wal_watcher_current_segment|prometheus_wal_watcher_record_decode_fa
ilures_total|prometheus_wal_watcher_records_read_total|prometheus_wal_watcher_sampl
es_sent_pre_tailing_total|reflector_items_per_list_count|reflector_items_per_list_s
um|reflector_items_per_watch|reflector_items_per_watch_count|reflector_items_per_wa
tch_sum|reflector_last_resource_version|reflector_list_duration_seconds_count|refle
ctor_list_duration_seconds_sum|reflector_lists_total|reflector_short_watches_total|
reflector_watch_duration_seconds|reflector_watch_duration_seconds_count|reflector_w
atch_duration_seconds_sum|reflector_watches_total|rest_client_request_latency_secon
ds_bucket|rest_client_request_latency_seconds_count|rest_client_request_latency_sec
onds_sum|ssh_tunnel_open_count|ssh_tunnel_open_fail_count|storage_operation_duratio
n_seconds_bucket|storage_operation_duration_seconds_count|storage_operation_duratio
n_seconds_sum|storage_operation_errors_total|volume_manager_total_volumesapiserver_
current_inflight_requests
          sourceLabels:
          - __name__
```

```
        ### url格式为https://{project}.{sls-enpoint}/prometheus/{project}/{metricsto
re}/api/v1/write
        ### {sls-enpoint}为日志服务的Endpoint。更多信息，请参见服务入口。
        ### {project}和{metricstore}替换为您已创建的日志服务的Project和Metricstore。
        url: https://sls-prometheus-test.cn-beijing.log.aliyuncs.com/prometheus/sls
-prometheus-test/prometheus-raw/api/v1/write
```

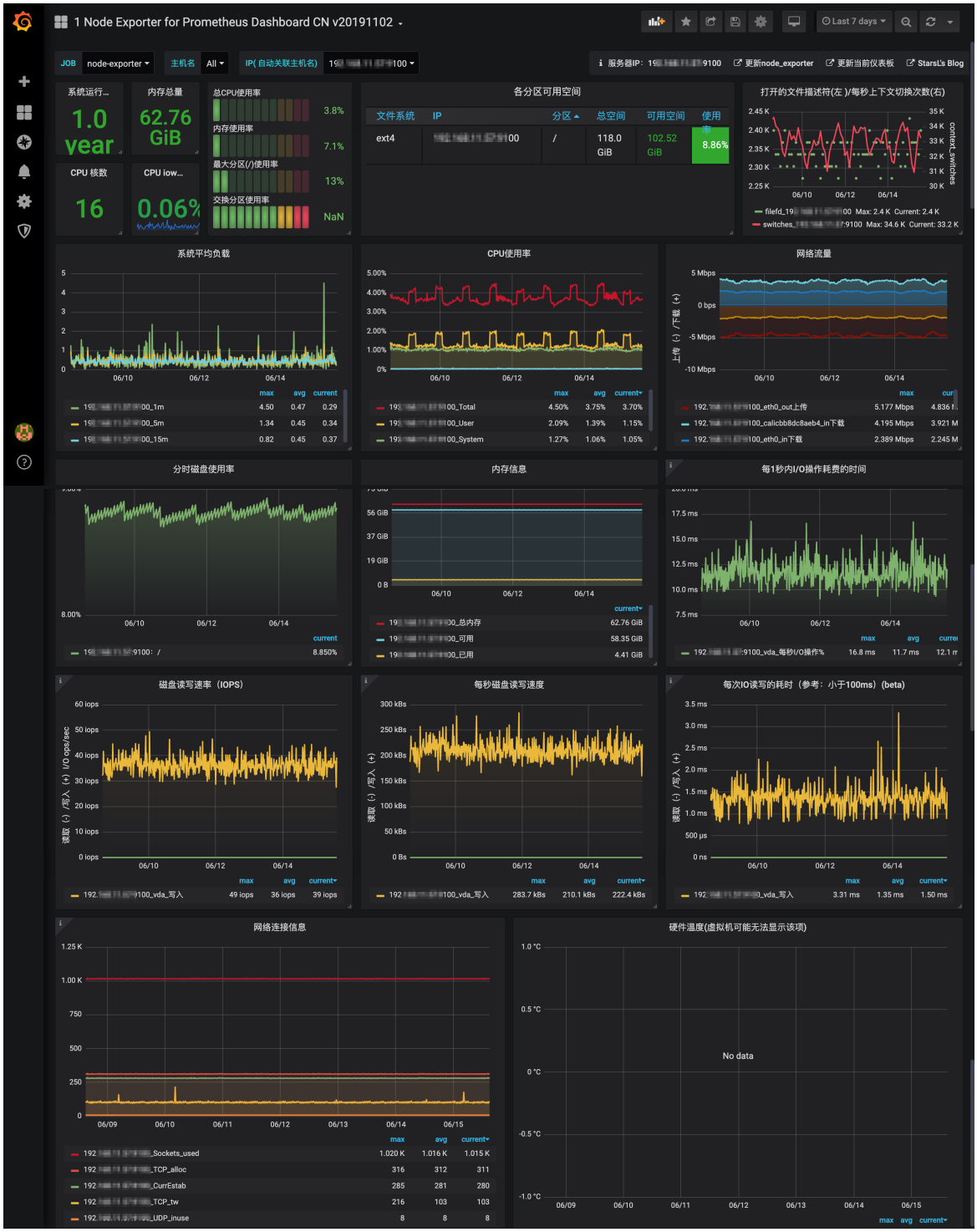
使用Grafana访问Prometheus数据

1. 登录Grafana。
2. 在左侧导航栏，单击  > Data Sources。
3. 在Data Sources页签，单击Add data source。
4. 单击Prometheus区域中的Select。
5. 在Settings页签中，配置如下参数。

参数	说明
Name	配置数据源名称，例如Prometheus-01。
HTTP	○ URL：日志服务MetricStore的URL，格式为https://{project}.{sls-enpoint}/prometheus/{project}/{metricstore}。其中*{sls-enpoint}*为日志服务的Endpoint。更多信息，请参见服务入口，*{project}*和*{metricstore}*为您已创建的日志服务的Project和Metricstore，请根据实际值替换。例如https://sls-prometheus-test.cn-hangzhou.log.aliyuncs.com/prometheus/sls-prometheus-test/prometheus。 注意 ■ 如果您是在阿里云内网，请优先使用内网域名。 ■ 为保证传输安全性，请务必使用https。 ○ Whitelisted Cookies：添加访问白名单，可选。
Auth	只需打开Basic auth开关。
Basic Auth Details	○ User为阿里云账号的AccessKey ID。 ○ Password为阿里云账号的AccessKey Secret。 建议您使用仅具备日志服务Project只读权限的RAM用户的AccessKey。更多信息，请参见授予指定Project只读权限。

6. 单击Save&Test。

   配置完成后，您可以在Grafana上查看数据仪表盘。


1 Node Exporter for Prometheus Dashboard CN v20191102
JOB
node-exporter
主机名
All
系统运行...
1.0 year
内存总量
62.76 GiB
CPU 核数
16
CPU iow...
0.06%
总CPU使用率
3.8%
内存使用率
7.1%
最大分区(/)使用率
13%
交换分区使用率
NaN
各分区可用空间
文件系统
IP
分区
总空间
可用空间
使用率
ext4
/
118.0 GiB
102.52 GiB
8.86%
打开的文件描述符(左)/每秒上下文切换次数(右)
系统平均负载
CPU使用率
网络流量
分时磁盘使用率
内存信息
每1秒内I/O操作耗费的时间
磁盘读写速率（IOPS）
每秒磁盘读写速度
每次IO读写的耗时（参考：小于100ms）(beta)
网络连接信息
硬件温度(虚拟机可能无法显示该项)
No data