Prometheus 监控服务

接入指南

腾讯云

文档目录

接入指南
抓取配置说明

最近更新时间：2024-12-05 16:07:34

概述

Prometheus 主要通过 Pull 的方式来抓取目标服务暴露出来的监控接口，因此需要配置对应的抓取任务来请求监控数据并写入到 Prometheus 提供的存储中，目前 Prometheus 服务提供了如下几个任务的配置：

- 原生 Job 配置：提供 Prometheus 原生抓取 Job 的配置。
- 云服务器服务发现配置：提供腾讯云服务器实例的服务发现配置。
- Pod Monitor：在 K8S 生态下，基于 Prometheus Operator 来抓取 Pod 上对应的监控数据。
- Service Monitor：在 K8S 生态下，基于 Prometheus Operator 来抓取 Service 对应 Endpoints 上的监控数据。

> **说明**
> `[]` 中的配置项为可选。

原生 Job 配置

相应配置项说明如下：

```
# 抓取任务名称，同时会在对应抓取的指标中加了一个 label(job=job_name)
job_name: <job_name>

# 抓取任务时间间隔
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# 抓取请求超时时间
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# 抓取任务请求 URI 路径
[ metrics_path: <path> | default = /metrics ]

# 解决当抓取的 label 与后端 Prometheus 添加 label 冲突时的处理。
# true: 保留抓取到的 label，忽略与后端 Prometheus 冲突的 label;
# false: 对冲突的 label，把抓取的 label 前加上 exported_<original-label>，添加后端 Prometheus 增加的
label;
[ honor_labels: <boolean> | default = false ]

# 是否使用抓取到 target 上产生的时间。
# true: 如果 target 中有时间，使用 target 上的时间;
# false: 直接忽略 target 上的时间;
[ honor_timestamps: <boolean> | default = true ]

# 抓取协议: http 或者 https
[ scheme: <scheme> | default = http ]

# 抓取请求对应 URL 参数
params:
  [ <string>: [<string>, ...] ]

# 通过 basic auth 设置抓取请求头中 `Authorization` 的值，password/password_file 互斥，优先取 password_file
里面的值。
```

```
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# 通过 bearer token 设置抓取请求头中 `Authorization` bearer_token/bearer_token_file 互斥，优先取
bearer_token 里面的值。
[ bearer_token: <secret> ]

# 通过 bearer token 设置抓取请求头中 `Authorization` bearer_token/bearer_token_file 互斥，优先取
bearer_token 里面的值。
[ bearer_token_file: <filename> ]

# 抓取连接是否通过 TLS 安全通道，配置对应的 TLS 参数
tls_config:
  [ <tls_config> ]

# 通过代理服务来抓取 target 上的指标，填写对应的代理服务地址。
[ proxy_url: <string> ]

# 通过静态配置来指定 target，详见下面的说明。
static_configs:
  [ - <static_config> ... ]

# HTTP 服务发现配置。原生服务发现配置可参考 Prometheus 官方文档
http_sd_configs:
  [ - <http_sd_config> ... ]

# 在抓取数据之后，把 target 上对应的 label 通过 relabel 的机制进行改写，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
relabel_configs:
  [ - <relabel_config> ... ]

# 数据抓取之后，通过 relabel 机制进行改写 label 对应的值，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
metric_relabel_configs:
  [ - <relabel_config> ... ]

# 一次抓取数据点限制，0：不作限制，默认为 0
[ sample_limit: <int> | default = 0 ]

# 一次抓取 Target 限制，0：不作限制，默认为 0
[ target_limit: <int> | default = 0 ]
```

static_config 配置

相应配置项说明如下：

```
# 指定对应 target host 的值，如ip:port。
targets:
  [ - '<host>' ]

# 在所有 target 上加上对应的 label，类似全局 label 的概念。
labels:
  [ <labelname>: <labelvalue> ... ]
```

示例：

```
job_name: prometheus
scrape_interval: 30s
static_configs:
- targets:
  - 127.0.0.1:9090
```

云服务器服务发现配置

> **注意：**
> 云服务器服务发现配置 cvm_sd_configs 不是 Prometheus 原生配置，不支持在集成中心或集成容器服务的抓取任务中使用。目前支持在集成中心的 CVM 云服务器集成中配置使用。以下简称 CVM 服务发现。

CVM 服务发现利用腾讯云 API 自动获取 CVM 实例列表，默认使用 CVM 的私网 IP。服务发现产生以下元标签，这些标签可以在 relabel 配置中使用。

标签	说明
__meta_cvm_instance_id	实例 ID
__meta_cvm_instance_name	实例名
__meta_cvm_instance_state	实例状态
__meta_cvm_instance_type	实例机型
__meta_cvm_OS	实例操作系统
__meta_cvm_private_ip	私网 IP
__meta_cvm_public_ip	公网 IP
__meta_cvm_vpc_id	网络 ID
__meta_cvm_subnet_id	子网 ID
__meta_cvm_tag_<tagkey>	实例标签值
__meta_cvm_region	实例所在区域
__meta_cvm_zone	实例的可用区

cvm_sd_configs 配置说明

```
# 腾讯云的地域，地域列表见文档
https://cloud.tencent.com/document/api/213/15692#.E5.9C.B0.E5.9F.9F.E5.88.97.E8.A1.A8。
region: <string>

# 自定义 endpoint。
[ endpoint: <string> ]

# 访问腾讯云 API 的凭证信息。如果不设置，取环境变量 TENCENT_CLOUD_SECRET_ID 和 TENCENT_CLOUD_SECRET_KEY 的
值。
# 如使用集成中心的 CVM 抓取任务进行配置，则无需填写。
[ secret_id: <string> ]
[ secret_key: <secret> ]

# CVM 列表的刷新周期。
```

```
[ refresh_interval: <duration> | default = 60s ]

# 抓取 metrics 的端口。
ports:
  - [ <int> | default = 80 ]

# CVM 列表的过滤规则。支持的过滤条件见文档
https://cloud.tencent.com/document/api/213/15728#2.-.E8.BE.93.E5.85.A5.E5.8F.82.E6.95.B0。
filters:
  [ - name: <string>
      values: <string>, [...] ]
```

说明

使用集成中心的 CVM 云服务器配置 cvm_sd_configs 时，该集成自动使用服务预设角色授权确保安全性，无需您手动填写如下参数：secret_id、secret_key、endpoint。

CVM 云服务器集成配置示例

```
job_name: demo-monitor
cvm_sd_configs:
- region: ap-guangzhou
  ports:
  - 8080
  filters:
  - name: tag:service
    values:
    - demo
relabel_configs:
- source_labels: [__meta_cvm_instance_state]
  regex: RUNNING
  action: keep
- regex: __meta_cvm_tag_(.*)
  replacement: $1
  action: labelmap
- source_labels: [__meta_cvm_region]
  target_label: region
  action: replace
```

Pod Monitor

相应配置项说明如下：

```
# Prometheus Operator CRD 版本
apiVersion: monitoring.coreos.com/v1
# 对应 K8S 的资源类型，这里面 Pod Monitor
kind: PodMonitor
# 对应 K8S 的 Metadata，这里只用关心 name，如果没有指定 jobLabel，对应抓取指标 label 中 job 的值为
<namespace>/<name>
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus  # namespace不固定，除kube-system下的任意namespace都可以
  labels:
      prom_id: prom-xxx
```

```
# 描述抓取目标 Pod 的选取及抓取任务的配置
spec:
  # 填写对应 Pod 的 label，pod monitor 会取对应的值作为 job label 的值。
  # 如果查看的是 Pod Yaml，取 pod.metadata.labels 中的值。
  # 如果查看的是 Deployment/Daemonset/Statefulset，取 spec.template.metadata.labels。
  [ jobLabel: string ]
  # 把对应 Pod 上的 Label 添加到 Target 的 Label 中
  [ podTargetLabels: []string ]
  # 一次抓取数据点限制，0：不作限制，默认为 0
  [ sampleLimit: uint64 ]
  # 一次抓取 Target 限制，0：不作限制，默认为 0
  [ targetLimit: uint64 ]
  # 配置需要抓取暴露的 Prometheus HTTP 接口，可以配置多个 Endpoint
  podMetricsEndpoints:
  [ - <endpoint_config> ... ] # 详见下面 endpoint 说明
  # 选择要监控 Pod 所在的 namespace，不填为选取所有 namespace
  [ namespaceSelector: ]
    # 是否选取所有 namespace
    [ any: bool ]
    # 需要选取 namespace 列表
    [ matchNames: []string ]
  # 填写要监控 Pod 的 Label 值，以定位目标 Pod  [K8S metav1.LabelSelector]
(https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.24/#labelselector-v1-meta)
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

示例：

```
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: redis-exporter # 填写一个唯一名称
    namespace: cm-prometheus  # namespace不固定，除kube-system下的任意namespace都可以
    labels:
        prom_id: prom-xxx # 配置您的实例 ID
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port  # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 Redis 实例 ID
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: ip
        replacement: '1.x.x.x' # 调整成对应的 Redis 实例 IP
```

```
    namespaceSelector:    # 选择要监控pod所在的namespace
      matchNames:
      - redis-test
    selector:     # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: redis-exporter
```

Service Monitor

相应配置项说明如下：

```
# Prometheus Operator CRD 版本
apiVersion: monitoring.coreos.com/v1
# 对应 K8S 的资源类型，这里面 Service Monitor
kind: ServiceMonitor
# 对应 K8S 的 Metadata，这里只用关心 name，如果没有指定 jobLabel，对应抓取指标 label 中 job 的值为 Service 的
名称。
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus   # namespace不固定，除kube-system下的任意namespace
  labels:
      prom_id: prom-xxx   # 配置您的实例 ID
# 描述抓取目标 Pod 的选取及抓取任务的配置
spec:
  # 填写对应 Pod 的 label(metadata/labels)，service monitor 会取对应的值作为 job label 的值
  [ jobLabel: string ]
  # 把对应 service 上的 Label 添加到 Target 的 Label 中
  [ targetLabels: []string ]
  # 把对应 Pod 上的 Label 添加到 Target 的 Label 中
  [ podTargetLabels: []string ]
  # 一次抓取数据点限制，0：不作限制，默认为 0
  [ sampleLimit: uint64 ]
  # 一次抓取 Target 限制，0：不作限制，默认为 0
  [ targetLimit: uint64 ]
  # 配置需要抓取暴露的 Prometheus HTTP 接口，可以配置多个 Endpoint
  endpoints:
  [ - <endpoint_config> ... ] # 详见下面 endpoint 说明
  # 选择要监控 Pod 所在的 namespace，不填为选取所有 namespace
  [ namespaceSelector: ]
    # 是否选取所有 namespace
    [ any: bool ]
    # 需要选取 namespace 列表
    [ matchNames: []string ]
  # 填写要监控 Pod 的 Label 值，以定位目标 Pod  [K8S metav1.LabelSelector](https://v1-
17.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#labelselector-v1-meta)
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

示例：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: go-demo     # 填写一个唯一名称
  namespace: cm-prometheus  # namespace不固定，除 kube-system 下的任意namespace
  labels:
      prom_id: prom-xxx    # 配置您的实例 ID
spec:
  endpoints:
  - interval: 30s
    # 填写service yaml中Prometheus Exporter对应的Port的Name
    port: 8080-8080-tcp
    # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    path: /metrics
    relabelings:
    # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 app，
    # 我们通过 relabel 的 replace 动作把它替换成了 application
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # 选择要监控service所在的namespace
  namespaceSelector:
    matchNames:
    - golang-demo
  # 填写要监控service的Label值，以定位目标service
  selector:
    matchLabels:
      app: golang-app-demo
```

endpoint_config 配置

相应配置项说明如下：

```
# 对应 port 的名称，这里需要注意不是对应的端口，默认：80，对应的取值如下：
# ServiceMonitor：对应 Service>spec/ports/name；
# PodMonitor：说明如下：
#   如果查看的是 Pod Yaml，取 pod.spec.containers.ports.name 中的值。
#   如果查看的是 Deployment/Daemonset/Statefulset，取 spec.template.spec.containers.ports.name。
[ port: string | default = 80]
# 抓取任务请求 URI 路径
[ path: string | default = /metrics ]
# 抓取协议：http 或者 https
[ scheme: string | default = http]
# 抓取请求对应 URL 参数
[ params: map[string][]string]
# 抓取任务间隔的时间
[ interval: string | default = 30s ]
# 抓取任务超时
[ scrapeTimeout: string | default = 30s]
# 抓取连接是否通过 TLS 安全通道，配置对应的 TLS 参数
[ tlsConfig: TLSConfig ]
# 通过对应的文件读取 bearer token 对应的值，放到抓取任务的 header 中
[ bearerTokenFile: string ]
# 通过对应的 K8S secret key 读取对应的 bearer token，注意 secret namespace 需要和
PodMonitor/ServiceMonitor 相同
```

```
[ bearerTokenSecret: string ]
# 解决当抓取的 label 与后端 Prometheus 添加 label 冲突时的处理。
# true: 保留抓取到的 label，忽略与后端 Prometheus 冲突的 label；
# false: 对冲突的 label，把抓取的 label 前加上 exported_<original-label>，添加后端 Prometheus 增加的
label；
[ honorLabels: bool | default = false ]
# 是否使用抓取到 target 上产生的时间。
# true: 如果 target 中有时间，使用 target 上的时间；
# false: 直接忽略 target 上的时间；
[ honorTimestamps: bool | default = true ]
# basic auth 的认证信息，username/password 填写对应 K8S secret key 的值，注意 secret namespace 需要和
PodMonitor/ServiceMonitor 相同。
[ basicAuth: BasicAuth ]
# 通过代理服务来抓取 target 上的指标，填写对应的代理服务地址。
[ proxyUrl: string ]
# 在抓取数据之前，把 target 上对应的 label 通过 relabel 的机制进行改写，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
relabelings:
[ - <relabel_config> ...]
# 数据抓取完成之后写入之前，通过 relabel 机制进行改写 label 对应的值，按顺序执行多个 relabel 规则。
# relabel_config 详见下面说明。
metricRelabelings:
[ - <relabel_config> ...]
```

relabel_config/relabelings 配置

相应配置项说明如下：

```
# 从原始 labels 中取哪些 label 的值进行 relabel，取出来的值通过 separator 中的定义进行字符拼接。
# 如果是 PodMonitor/ServiceMonitor 对应的配置项为 sourceLabels 。
[ source_labels: '[' <labelname> [, ...] ']' ]
# 定义需要 relabel 的 label 值拼接的字符，默认为 ';'。
[ separator: <string> | default = ; ]

# action 为 replace/hashmod 时，通过 target_label 来指定对应 label name。
# 如果是 PodMonitor/ServiceMonitor 对应的配置项为 targetLabel 。
[ target_label: <labelname> ]

# 需要对 source labels 对应值进行正则匹配的表达式。
[ regex: <regex> | default = (.*) ]

# action 为 hashmod 时用到，根据 source label 对应值 md5 取模值。
[ modulus: <int> ]

# action 为 replace 的时候，通过 replacement 来定义当 regex 匹配之后需要替换的表达式，可以结合 regex 正规则表达
式替换。
[ replacement: <string> | default = $1 ]

# 基于 regex 匹配到的值进行相关的操作，对应的 action 如下，默认为 replace:
# replace: 如果 regex 匹配到，通过 replacement 中定义的值替换相应的值，并通过 target_label 设置并添加相应的
label
# keep: 如果 regex 没有匹配到，丢弃
# drop: 如果 regex 匹配到，丢弃
# hashmod: 通过 modulus 指定的值把 source label 对应的 md5 值取模，添加一个新的 label，label name 通过
target_label 指定
```

```
# labelmap: 如果 regex 匹配到，使用 replacement 替换对应的 label name
# labeldrop: 如果 regex 匹配到，删除对应的 label
# labelkeep: 如果 regex 没有匹配到，删除对应的 label
[ action: <relabel_action> | default = replace ]
```

自定义监控

最近更新时间：2024-12-05 16:07:34

操作场景

您可以通过 Prometheus 监控服务自定义上报指标监控数据，对应用或者服务内部的一些状态进行监控，如请求处理数，下单数等，也可以对一些核心逻辑的处理耗时进行监控，如请求外部服务的耗时情况等。
本文以 Go 这个语言为例，介绍如何通过 Prometheus 监控服务进行业务自定义指标上报，可视化及告警。

支持开发语言

Prometheus 开源社区官方 SDK：

- Go
- Java or Scala
- Python
- Ruby

其它第3方开发语言 SDK：

- Bash
- C
- C++
- Common Lisp
- Dart
- Elixir
- Erlang
- Haskell
- Lua for Nginx
- Lua for Tarantool
- .NET / C#
- Node.js
- Perl
- PHP
- R
- Rust

更多信息请参见此处

数据模型

Prometheus 具有多维分析的能力，数据模型有如下几部分组成。
`Metric Name(指标名称)` + `Labels(标签)` + `Timestamp(时间戳)` + `Value/Sample(监控值/样品)`

- Metric Name(指标名称)：监控对象的含义（例如，http_request_total – 表示当前系统接收到的 HTTP 请求总量）。
- 标签(label)：表示当前样本的特征维度，是一个 K/V 结构，通过这些维度 Prometheus 可以对样本数据进行过滤，聚合等。
- 时间戳(timestamp)：一个精确到毫秒的时间戳。
- 样本值(value)：一个 float64 的浮点型数据表示当前样本的值。

`Metric Name(指标名称)/Labels(标签)` 只能由 ASCII 字符、数字、下划线以及冒号组成，并必须符合正则表达式[a-zA-Z_:][a-zA-Z0-9_:]*。

- 更多 Data Model 说明
- Metric/Label 命名最佳实践

如何监控埋点

Prometheus 根据监控的不同场景提供了 `Counter` / `Gauge` / `Histogram` / `Summary` 四种指标类型，每种指标类型说明可参见下文。更多说明请参见 Prometheus 官网 METRIC TYPES 。
Prometheus 社区提供了多种开发语言的 SDK，每种语言的使用方法基本上类似，主要是开发语言语法上的区别，下面主要以 Go 作为例子如何上报自定义监控指标数据。

Counter

计数类型，数据是单调递增的指标，服务重启之后会重置。可以用 Counter 来监控请求数/异常数/用户登录数/订单数等。
如何通过 `Counter` 来监控订单数：

```
package order

import (
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
)

// 定义需要监控 Counter 类型对象
var (
    opsProcessed = promauto.NewCounterVec(prometheus.CounterOpts{
        Name: "order_service_processed_orders_total",
        Help: "The total number of processed orders",
    }, []string{"status"}) // 处理状态
)

// 订单处理
func makeOrder() {
    opsProcessed.WithLabelValues("success").Inc() // 成功状态
    // opsProcessed.WithLabelValues("fail").Inc() // 失败状态

    // 下单的业务逻辑
}
```

例如，通过 rate() 函数获取订单的增长率：

```
rate(order_service_processed_orders_total[5m])
```

Gauge

当前值，监控打点的时候可对其做加减。可以用 Gauge 来监控当前内存使用率 /CPU 使用率/当前线程数/队列个数等。
如何通过 `Gauge` 来监控订单队列大小：

```
package order

import (
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
)

// 定义需要监控 Gauge 类型对象
var (
    queueSize = promauto.NewGaugeVec(prometheus.GaugeOpts{
        Name: "order_service_order_queue_size",
        Help: "The size of order queue",
    }, []string{"type"})
)
```

```
type OrderQueue struct {
    queue chan string
}

func newOrderQueue() *OrderQueue  {
    return &OrderQueue{
        queue: make(chan string,100),
    }
}

// 产生订单消息
func (q *OrderQueue)produceOrder() {
    // 产生订单消息

    // 队列个数加1
    queueSize.WithLabelValues("make_order").Inc() // 下单队列
    // queueSize.WithLabelValues("cancel_order").Inc() // 取消订单队列
}

// 消费订单消息
func (q *OrderQueue)consumeOrder() {
    // 消费订单消息

    // 队列个数减1
    queueSize.WithLabelValues("make_order").Dec()
}
```

通过 Gauge 指标，直接查看订单每种类型队列的当前大小：

```
order_service_order_queue_size
```

Histogram

直方图，Prometheus 会根据配置的 `Bucket` 来计算样本的分布情况，后期可以再加工，一般多用于耗时的监控，通过 Histogram 可以计算出P99/P95/P50等耗时，同时也可以监控处理的个数，如果用上 Histogram 就不需要再用 Counter 统计个数。可以用 Histogram 来监控接口响应时间/数据库访问耗时等。
Histogram 和 Summary 的使用方式类似，可以直接参考 Summary 的使用方式。

Summary

摘要，和 Histogram 有一点类似，也是计算样本的分布情况，区别是 Summary 会在客户端计算出分布情况(P99/P95/Sum/Count)，因此也会更占客户端资源，后期不可再聚合计算处理，同样可以用 Summary 来监控接口响应时间/数据库访问耗时等。
如何通过 `Summary` 来监控订单处理耗时：

```
package order

import (
    "net/http"
    "time"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

// 定义需要监控 Summary 类型对象
```

```
var (
    opsProcessCost = promauto.NewSummaryVec(prometheus.SummaryOpts{
        Name: "order_service_process_order_duration",
        Help: "The order process duration",
    }, []string{"status"})
)

func makeOrder() {
    start := time.Now().UnixNano()
    // 下单逻辑处理结束，记录处理耗时
    defer opsProcessCost.WithLabelValues("success").Observe((float64)(time.Now().UnixNano() -
start))

    // 下单的业务逻辑
    time.Sleep(time.Second) // 模拟处理耗时
}
```

通过 Summary 指标，直接查看下单处理平均耗时：

```
order_service_processed_order_duration_sum / order_service_processed_order_duration_count
```

暴露 Prometheus 指标

通过 `promhttp.Handler()` 把监控埋点数据暴露到 HTTP 服务上。

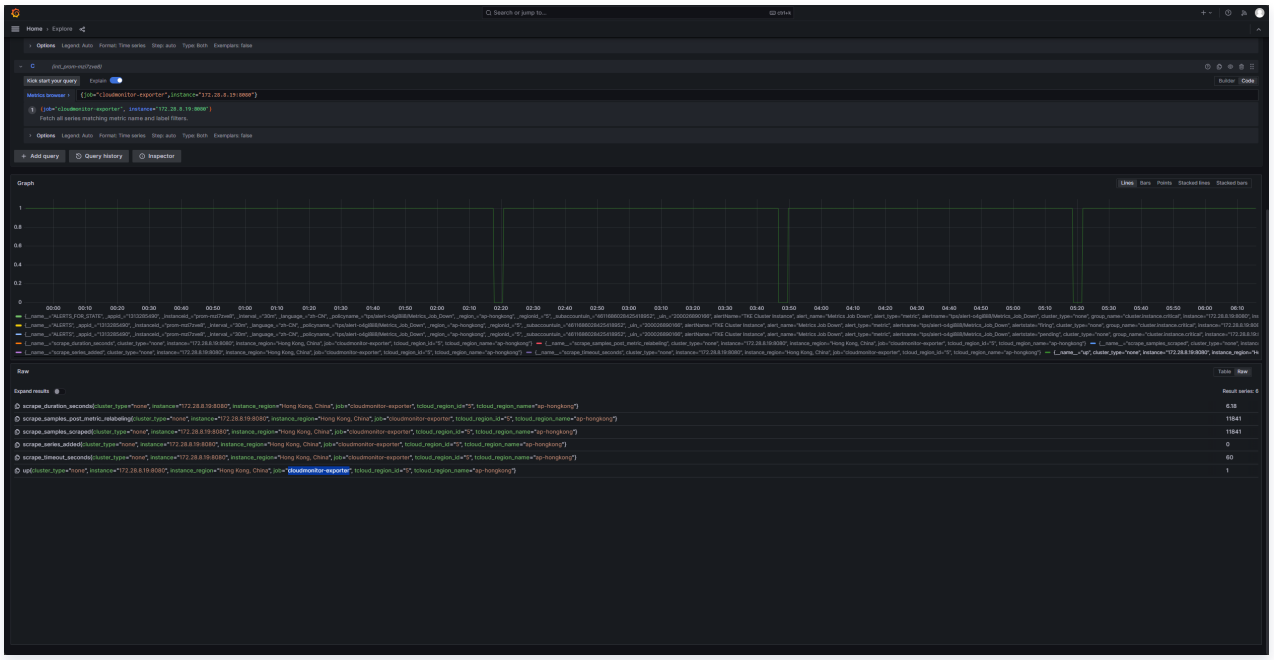
```
package main

import (
    "net/http"

    "github.com/prometheus/client_golang/prometheus/promhttp"
)

func main() {
        // 业务代码

        // 把 Prometheus 指标暴露在 HTTP 服务上
        http.Handle("/metrics", promhttp.Handler())

        // 业务代码
}
```

采集数据

完成相关业务自定义监控埋点之后，应用发布，即可通过 Prometheus 来抓取监控指标数据。详情请参见 Golang 接入 。

查看监控数据和告警

- 打开 Prometheus 监控服务自带的 Grafana，通过 `Explore` 来查看监控指标数据，如下图，也可以 自定义 Grafana 监控大盘 。



- 通过 `Prometheus` 和 腾讯云可观测平台 `告警管理` 的能力可以对自定义监控指标进行实时告警，详情请参见 告警介绍及使用 。

EMR 接入
Flink 接入

最近更新时间：2025-03-07 10:12:53

操作场景

在使用 Flink 过程中需要对 Flink 任务运行状态进行监控，以便了解 Flink 任务是否正常运行，排查 Flink 故障等。Prometheus 监控服务对 push gateway 做了集成，支持 Flink 写入 metrics，并提供了开箱即用的 Grafana 监控大盘。

前提条件

- 购买的腾讯云弹性 MapReduce（以下简称 EMR）产品包含 Flink 组件，并在实例上跑 Flink 任务。
- 使用与 EMR 相同的地域及私有网络 VPC 购买腾讯云 Prometheus 监控实例。

操作步骤

产品接入

获取 PushGateway 访问地址

1. 登录 腾讯云可观测平台。
2. 进入 Prometheus 实例，根据 Pushgateway 接入 创建 Pushgateway。
3. 获取 PushGateway 地址。

修改 Flink 配置

1. 进入 弹性 MapReduce，选择并进入对应的“实例”，选择**集群服务**页面。
2. 找到 **Flink** 配置项，选择**配置管理** > **flink-conf.yaml**，进入配置管理页面。
3. 在页面单击**编辑配置** > **新增配置项**，依次添加以下配置，更多信息请参见 官方文档。

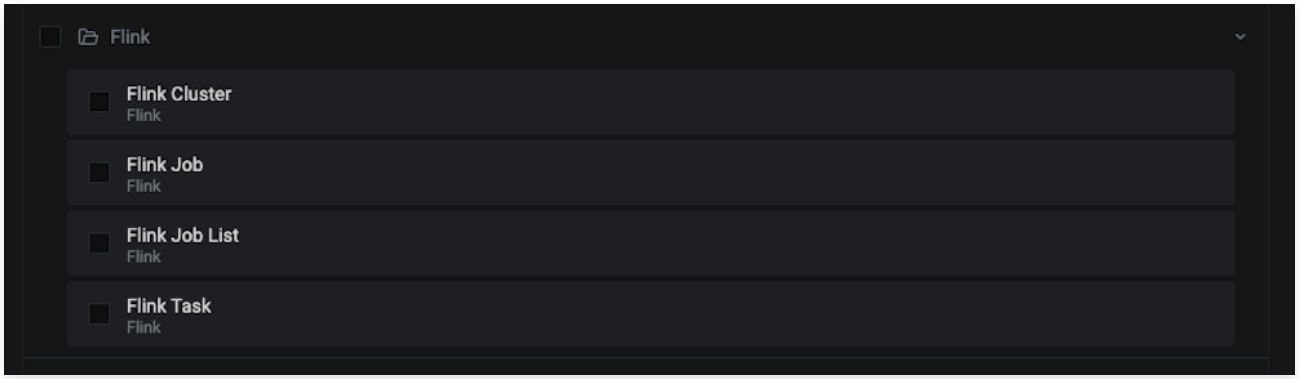
配置名	默认	类型	描述	建议
metrics.reporter.promgateway.class	无	字符串	实现 metrics 导出到 push gateway 的 Java 类名	org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporterFactory
metrics.reporter.promgateway.jobName	无	字符串	push 任务名	指定方便理解的字符串
metrics.reporter.promgateway.randomJobNameSuffix	true	布尔	是否在任务名后添加随机字符串	需设置为 true，如果不添加，Flink 任务间 metrics 会相互覆盖
metrics.reporter.promgateway.groupingKey	无	字符串	添加到每个 metrics 的全局 label，格式为 k1=v1;k2=v2	添加 EMR 实例 ID 方便区分不同实例的数据，例如 instance_id=emr-xxx
metrics.reporter.promgateway.interval	无	时间	推送 metrics 的时间间隔，例如30秒	建议设置在1分钟左右
metrics.reporter.promgateway.hostUrl	无	字符串	push gateway 的服务地址	控制台上 prometheus 实例的服务地址
metrics.reporter.promgateway.deleteOnShutdown	true	布尔	Flink 任务执行完后是否删除 push gateway 上对应的 metrics	设置为 true

配置示例如下：

```
metrics.reporter.promgateway.class:
org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporter
metrics.reporter.promgateway.jobName: my-job
metrics.reporter.promgateway.randomJobNameSuffix: true
metrics.reporter.promgateway.interval: 60 SECONDS
metrics.reporter.promgateway.groupingKey: instance_id=emr-xxxx
metrics.reporter.promgateway.host: pushgateway 的 ip
metrics.reporter.promgateway.port: 8080
```

查看监控

1. 登录 Prometheus 监控，选择并进入对应 Prometheus 实例。
2. 选择**数据采集** > **集成中心**，在**集成中心**中找到 **Flink** 监控，在 **Dashboard** 页面单击**安装/升级**即可开启 Flink 监控大盘。
3. 进入 Grafana，单击展开 Flink 监控面板。

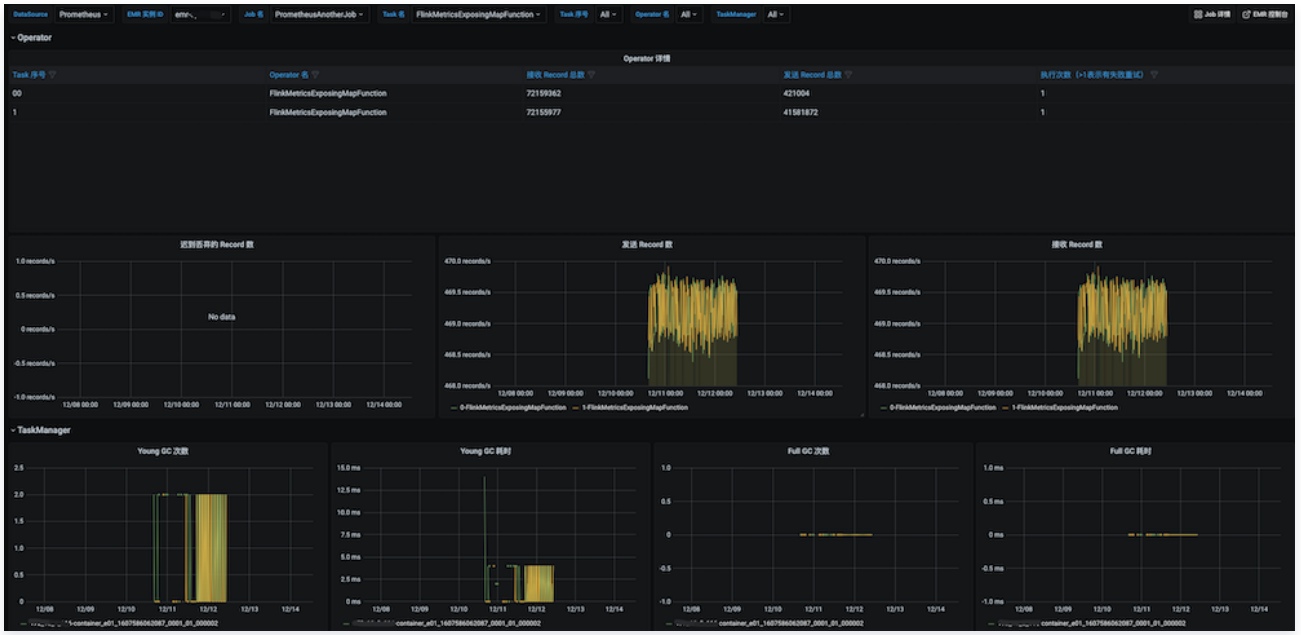


4. 单击 **Flink Job List** 查看监控。

5. 单击表格中的 **Job 名**或 **Job ID 列值**，查看 Job 监控详情。

6. 单击右上角的 **Flink 集群**，查看 Flink 集群监控。

7. 单击表格中的 **Task 名列值**，查看 Task 监控详情。



告警接入

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**，选择对应 Prometheus 实例进入管理页面。
3. 选择**告警管理** > **告警策略**，单击**新建告警策略**可以添加相应的告警策略，详情请参见 新建告警策略 。

Prometheus 采集 EMR 组件

最近更新时间：2025-03-21 17:40:02

操作场景

在使用 腾讯云弹性 MapReduce（以下简称 EMR）产品过程中，需要将 EMR 监控指标上报到 Prometheus 监控服务，可参考下文指引。下文将为您介绍如何快速采集 EMR 监控指标。

采集 EMR on CVM 实例

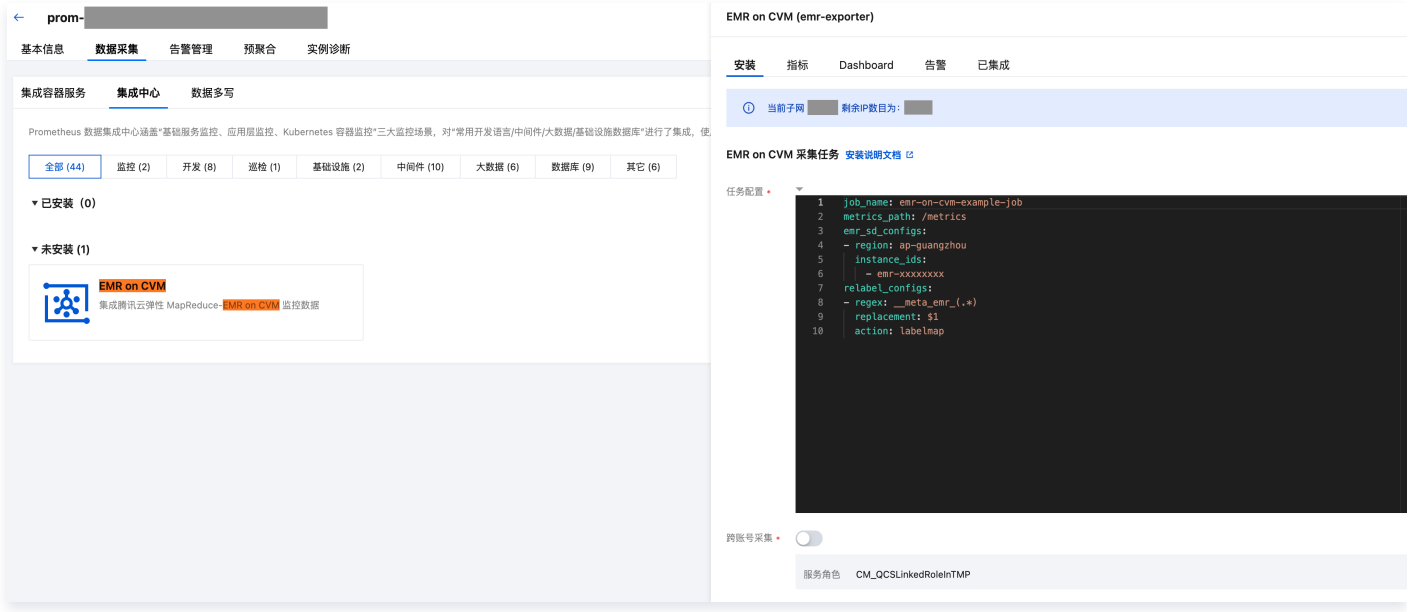
前提条件

使用与 EMR 相同的地域及私有网络 VPC，购买腾讯云 Prometheus 监控实例 。可查看 Prometheus 监控服务支持的地域 。

> **说明：**
> - Prometheus 监控服务 通过 EMR 节点内网地址采集监控指标，跨 VPC 采集需要先打通网络。
> - 在网络相通的前提下，支持 跨账号采集 。

操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控服务**。
3. 在 Prometheus 实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，选择**数据采集 > 集成中心**。
5. 在集成中心找到并单击 **EMR on CVM**，即会弹出一个安装窗口，确认信息后单击**保存**即可。



6. 进入 EMR 控制台 ，单击集群的 ID/名称，在**实例信息**页面，获取 EMR 集群所在地域、EMR 实例 ID。

7. 填写任务配置（Yaml 格式）。按下图红框依次填写任务名、EMR 集群所在地域、EMR 实例 ID。

> **说明：**
> - 地域填写格式可以参见 开服地域 的地域说明，例如：`ap-guangzhou`。
> - 实例 ID 支持填写多个。
> - relabel_configs 配置参见 抓取配置说明。

EMR on CVM 采集任务 安装说明文档

任务配置 *

```
job_name: emr-on-cvm-example-job
metrics_path: /metrics
emr_sd_configs:
- region: ap-guangzhou
  instance_ids:
    - emr-xxxxxxxx
relabel_configs:
- regex: __meta_emr_(.*)
  replacement: $1
  action: labelmap
```

采集 EMR on TKE 实例

前提条件

Prometheus 实例关联 EMR on TKE 实例所在容器集群，请参见 关联集群。

操作步骤

1. 登录 腾讯云可观测平台。
2. 在左侧菜单栏中选择 **Prometheus 监控服务**。
3. 在 Prometheus 实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，选择**数据采集 > 集成中心**。
5. 在集成中心找到并单击 **EMR on TKE**，即会弹出一个安装窗口。

6. 选择 EMR on TKE 实例所在容器集群，会自动查找容器集群中的 EMR on TKE 实例，选择需要采集的实例移动到右侧框，单击**保存**即可。

采集 EMR Serverless 实例

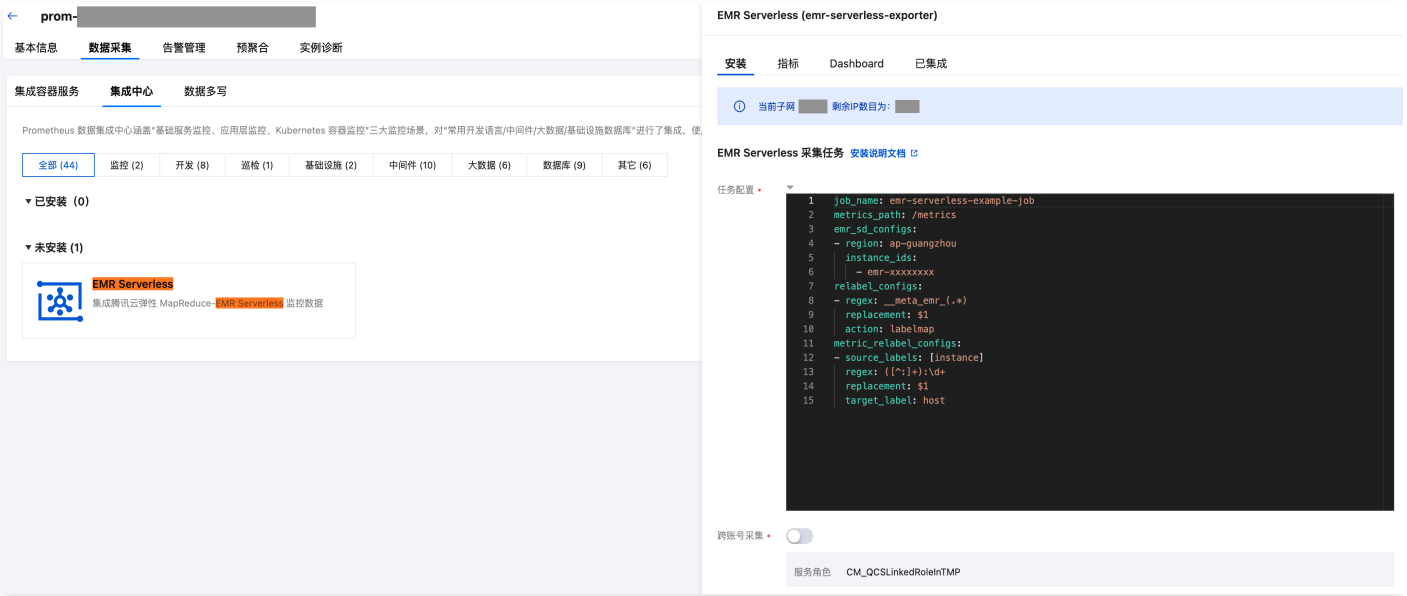
前提条件

使用与 EMR 相同的地域及私有网络 VPC，购买腾讯云 Prometheus 监控实例 。可查看 Prometheus 监控服务支持的地域 。

> **说明：**
> - Prometheus 监控服务 通过 EMR 节点内网地址采集监控指标，跨 VPC 采集需要先打通网络。
> - 在网络相通的前提下，支持 跨账号采集 。

操作步骤

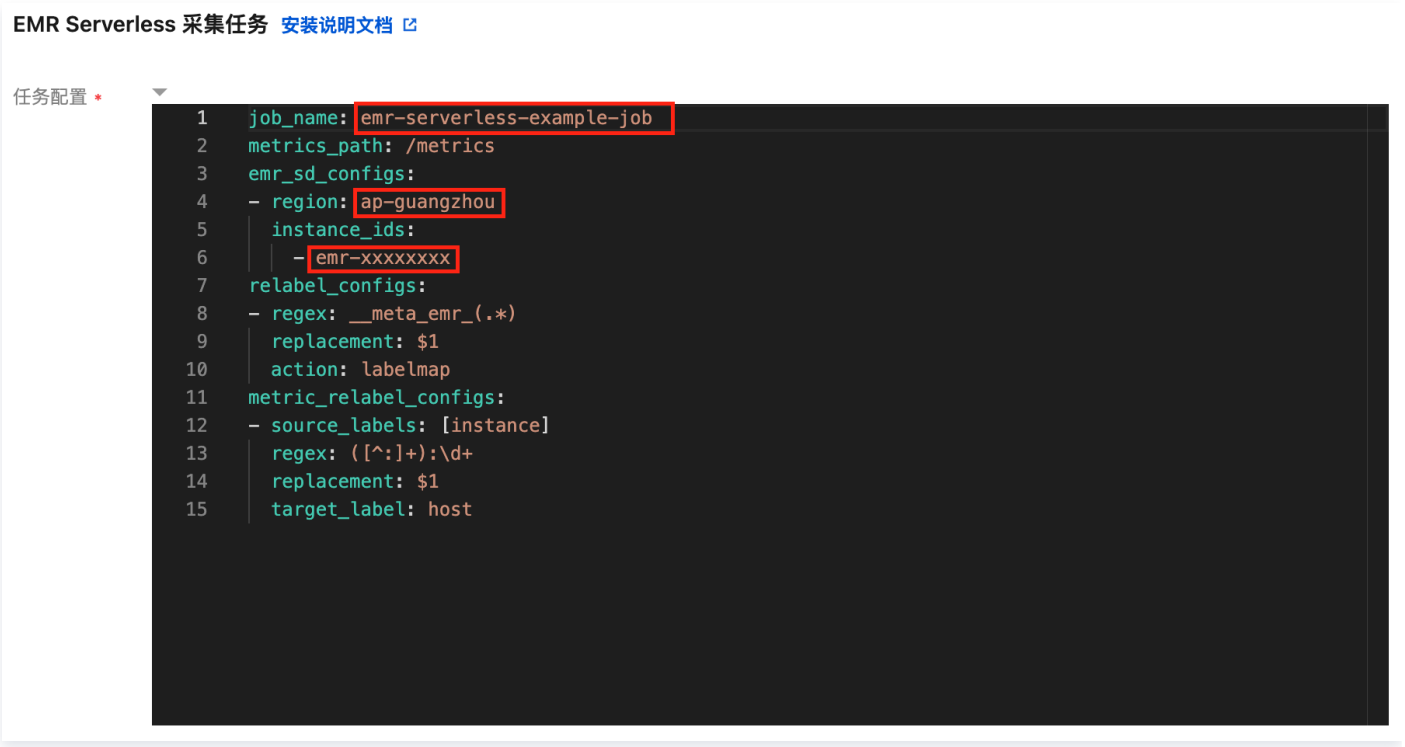
1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控服务**。
3. 在 Prometheus 实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，选择**数据采集 > 集成中心**。
5. 在集成中心找到并单击 **EMR Serverless**，即会弹出一个安装窗口，确认信息后单击**保存**即可。



6. 进入 EMR 控制台，单击集群的 ID/名称，在**实例信息**页面，获取 EMR 集群所在地域和 EMR 实例 ID。

7. 填写任务配置（Yaml 格式）。按下图红框依次填写任务名、EMR 集群所在地域、EMR 实例 ID。

> **说明：**
> - 地域填写格式可以参见 开服地域 的地域说明，例如：`ap-guangzhou`。
> - 实例 ID 支持填写多个。
> - relabel_configs 配置参见 抓取配置说明。



支持指标

Prometheus 监控服务支持所有 EMR 指标，详细指标列表请参见 EMR 集群监控指标 。

如何正确判断服务异常

- EMR on CVM 的 service_status 指标在服务异常或者用户手动停止时都会显示0，无法正确区分异常服务。因为 EMR 无法直接提供服务异常的指标，只能判断用户是否手动停止。Prometheus 监控服务根据服务是否手动停止的信息，新增 emr_additional_service_status 指标，与 service_status 指标组合可用于区分异常服务，值为 0 表示服务异常：

  ```
  (service_status{} * on(instance_id, host, type) group_left() (emr_additional_service_status{} == 1))
  ```
- EMR on TKE、EMR Serverless 不存在上述情况，可直接使用 service_status 指标，值为 0 表示服务异常。

Java 应用接入
Spring Boot 接入

最近更新时间：2025-02-25 11:00:22

操作场景

在使用 Spring Boot 作为开发框架时，需要监控应用的状态，例如 JVM/Spring MVC 等。Prometheus 监控服务基于 Spring Actuator 机制采集 JVM 等数据，结合配套提供的 Grafana Dashboard 可以方便地监控 Spring Boot 应用的状态。
本文档以在容器服务上部署 Spring Boot 应用为例，介绍如何通过 Prometheus 监控服务监控其状态。

前提条件

- 创建 腾讯云容器服务-托管版集群：在腾讯云容器服务中创建 Kubernetes 集群。
- 使用 容器镜像服务 管理应用镜像。
- 应用基于 Spring Boot 框架进行开发。

以下主要以 Maven 为例，说明整个集成。

操作步骤

⚠ 注意：

- Spring Boot 已提供 actuator 组件来对应用进行监控，简化了开发的使用成本，所以这里直接使用 actuator 为 Spring Boot 应用进行监控埋点，基于 Spring Boot 2.0及以上的版本，低版本会有配置上的差别需要注意。
- 若您使用 spring boot 1.5接入，接入时和2.0会有一定区别，需要注意如下几点：
  - 访问 `prometheus metrics` 的地址和2.0不一样，1.5默认的是 `/prometheus` ，即 `http://localhost:8080/prometheus` 。
  - 若报401错误则表示没有权限(Whitelabel Error Page)，1.5默认对 `management` 接口加了安全控制，需要修改 `management.security.enabled=false` 。
  - 若项目中用 `bootstrap.yml` 来配置参数，在 `bootstrap.yml` 中修改 `management` 不起作用，需要在 `application.yml` 中修改，原因： spring boot 启动加载顺序有关。
  - `metric common tag` 不能通过 `yml` 来添加，可以通过代码加一个 `bean` 的方式实现，如下：

```
@Component
public class MetricCommonTag {
    @Autowired
    protected MeterRegistry meterRegistry;

    @PostConstruct
    public void initialize() {
        this.meterRegistry.config()
                .commonTags("region", "gz");  // key->value
    }
}
```

修改应用的依赖及配置

步骤1：修改 pom 依赖

项目中已经引用 `spring-boot-starter-web` 的基础上，在 `pom.xml` 文件中添加 `actuator/prometheus` Maven 依赖项。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

步骤2：修改配置

编辑 `resources` 目录下的 `application.yml` 文件，修改 `actuator` 相关的配置来暴露 Prometheus 协议的指标数据。

```
server:
  tomcat:
    mbeanregistry:
      enabled: true
management:
  endpoints:
    web:
      exposure:
        include: prometheus  # 打开 Prometheus 的 Web 访问 Path
  metrics:
    # 在 Prometheus 中添加特别的 Labels
    # 必须加上对应的应用名，因为需要以应用的维度来查看对应的监控
    tags:
      application: spring-mvc-demo
    # 下面选项建议打开，以监控 http 请求的 P99/P95 等，具体的时间分布可以根据实际情况设置
    distribution:
      sla:
        http:
          server:
            requests: 1ms,5ms,10ms,50ms,100ms,200ms,500ms,1s,5s
      percentiles-histogram:
        http:
          server:
            requests: true # 开启 http server 的请求监控
```

步骤3：本地验证

在项目当前目录下，运行 `mvn spring-boot:run` 之后，可以通过 `http://localhost:8080/actuator/prometheus` 访问到 Prometheus 协议的指标数据，说明相关的依赖配置已经正确。

> **说明：**
> 例子中配置默认配置，对应的端口和路径以实际项目为准。

将应用发布到腾讯云容器服务上

步骤1：本地配置 Docker 镜像环境

如果本地之前未配置过 Docker 镜像环境，可以参考 Docker 镜像操作快速入门 进行配置，如果已经配置可以直接执行下一步。

步骤2：打包及上传镜像

1. 在项目根目录下添加 `Dockerfile` ，您可以参考如下示例进行添加，在实际项目中需要修改 `Dockerfile` 。

```
FROM openjdk:21-jdk
WORKDIR /spring-mvc-demo
ADD target/spring-mvc-demo-*.jar /spring-mvc-demo/spring-mvc-demo.jar
```

```
CMD ["java","-jar","spring-boot-demo.jar"]
```

2. 打包镜像，在项目根目录下运行如下命令，在实际项目中需要替换对应的 `namespace`、`ImageName`、**镜像版本号**。

```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
```

例如：

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

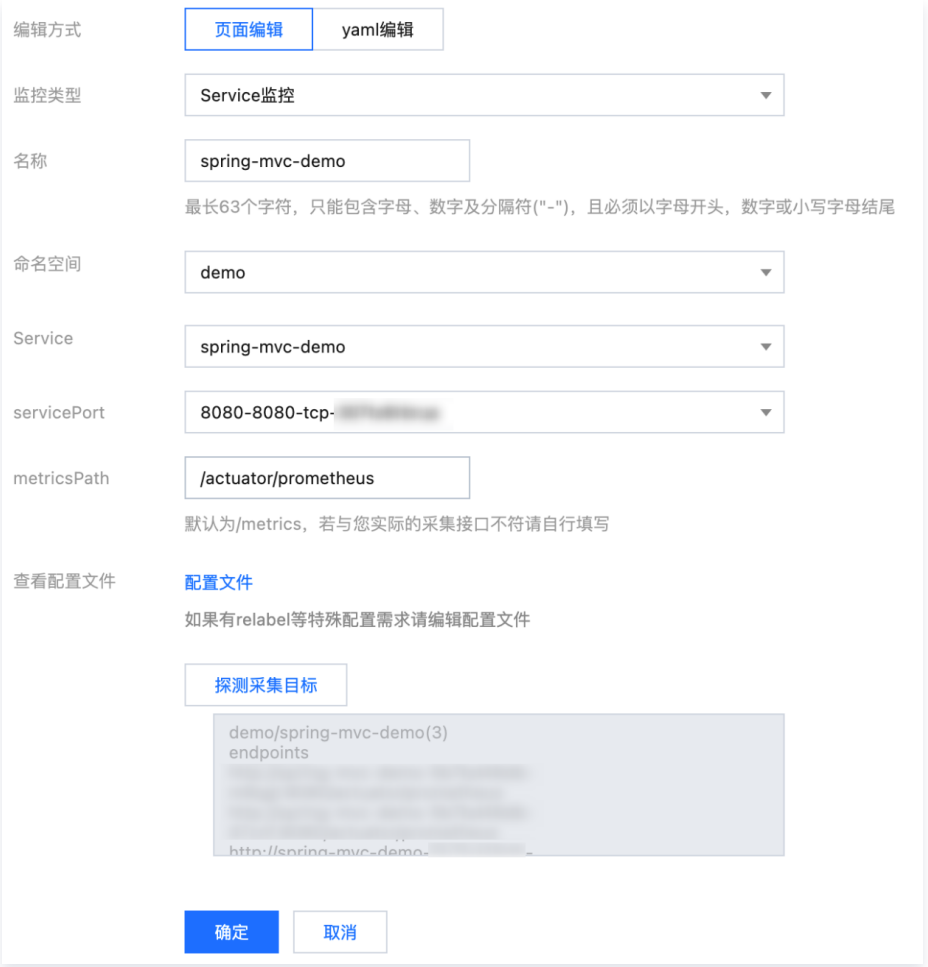
步骤3：应用部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中选择**集群**，进入集群列表选择需要部署的容器集群。
3. 选择**工作负载** > **Deployment**，在 **Deployment** 管理页面，选择对应的命名空间来进行部署服务，这里选择通过控制台的方式创建，同时打开 Service 访问方式，您也可以选择通过命令行的方式创建。


名称
spring-mvc-demo
最长63个字符，只能包含小写字母、数字及分隔符("-")，且必须以小写字母开头，数字或小写字母结尾
描述
请输入描述信息，不超过1000个字符
命名空间
demo
Labels
新增
标签键名称不超过63个字符,仅支持英文、数字、'/'、'-',且不允许以('/')开头。支持使用前缀，更多说明查看详情
标签键值只能包含字母、数字及分隔符("-"、"_"、".")，且必须以字母、数字开头和结尾
数据卷（选填）
添加数据卷
为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷、文件存储NFS、配置文件、PVC，还需挂载到容器的指定路径中。使用指引
实例内容器
spring-mv...
添加容器
名称
spring-mvc-demo
最长63个字符，只能包含小写字母、数字及分隔符("-")，且不能以分隔符开头或结尾
镜像
prometheus-demo.tencentcloudcr.com/prometheus-demo/spring-mvc-der
选择镜像
镜像版本（Tag）
latest
选择镜像版本
镜像拉取策略
Always
IfNotPresent
Never
总是从远程拉取该镜像
环境变量
新增变量
变量名为空时，在变量名称中粘贴一行或多行key=value或key: value的键值对可以实现快速批量输入
CPU/内存限制
CPU限制
request 0.25 - limit 0.5 核
内存限制
request 256 - limit 1024 MiB
Request用于预分配资源,当集群中的节点没有request所要求的资源数量时,容器会创建失败。
Limit用于设置容器使用资源的最大上限,避免异常情况下节点资源消耗过多。
GPU 资源
卡数: 0 个
配置该工作负载使用的最少GPU资源，请确保集群内已有足够的GPU资源
容器端口
添加容器端口
显示高级设置
实例数量
手动调节
自动调节
直接设定实例数量
实例数量 1 个
镜像访问凭证
prom-demo
添加镜像访问凭证
请指定镜像访问凭证以拉取私有镜像，或参考为服务账号添加 ImagePullSecrets 实现免密拉取； 如无合适的访问凭证，请新建访问凭证
节点调度策略
不使用调度策略
自定义调度规则
可根据调度规则，将Pod调度到符合预期的Label的节点中。设置工作负载的调度规则指引
如果集群开启了注册节点能力，请注意不要将nginx-ingress调度到注册节点上，具体参考节点调度策略
容忍调度
不使用容忍调度
使用容忍调度
显示高级设置

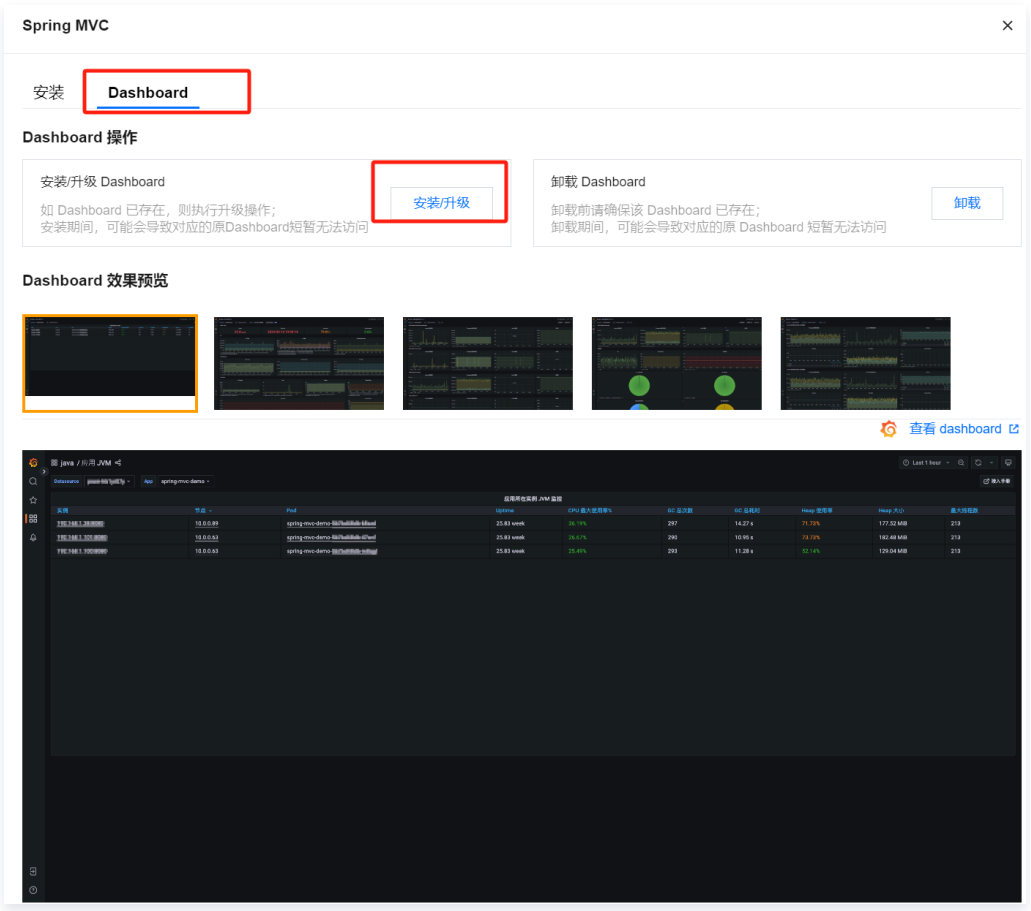
步骤4：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，进入到容器服务集成管理页面，选择服务部署所在的容器集群（该集群已经与 Prometheus 实例进行了关联）。
3. 在操作列单击**数据采集配置**，然后单击**新建自定义监控**，添加 ServiceMonitor 采集配置，具体如下：



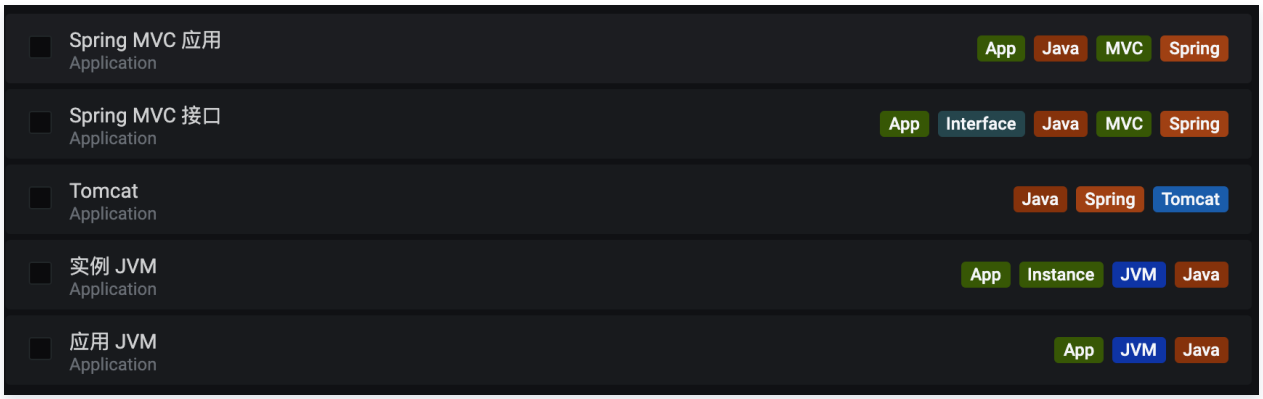
步骤5：安装 Dashboard

1. 在**集成中心**下找到并单击 **Spring MVC**。
2. 选择 **Dashboard**，单击 **Dashboard 操作**下的**安装/升级**。



步骤6：查看监控

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 打开 Prometheus 实例对应的 Grafana 地址，在 `Dashboards/Manage/Application` 下查看应用相关的监控大屏。
   - **Spring MVC 应用**：监控 MVC 的状态，例如请求耗时/请求量/成功率/异常分布等。
   - **Spring MVC 接口**：接口级监控，可以对应多个接口，方便定位是哪个接口出问题。
   - **Tomcat**：Tomcat 内部状态的监控大屏，例如线程使用情况等。
   - **应用 JVM**：从应用角度出发，查看该应用下所有实例是否有问题，当发现某个实例有问题时可以下钻到对应的实例监控。
   - **实例 JVM**：单实例 JVM 详细的监控数据。


Spring MVC 应用
Application
App Java MVC Spring
Spring MVC 接口
Application
App Interface Java MVC Spring
Tomcat
Application
Java Spring Tomcat
实例 JVM
Application
App Instance JVM Java
应用 JVM
Application
App JVM Java


RED Method(Overview)
Latency(请求耗时)
Throughput(每秒请求数)
Error(失败数)
成功率
应用层
2XX
3XX/4XX/5XX
日志级别
Top 10(请求量)
/hello/{name} 13.71 K
/actuator/prometheus 480
/**/favicon.ico 0
接口异常类型分布
NumberFormatException 344
接口异常实例分布
344
接口异常分布
/hello/{name} 344

JVM 接入

最近更新时间：2025-02-28 15:22:53

操作场景

在使用 Java 作为开发语言时，需要监控 JVM 的性能。Prometheus 监控服务通过采集应用暴露出来的 JVM 监控数据，并提供了开箱即用的 Grafana 监控大盘。
本文介绍了通过 client_java 或 jmx_exporter 两种方式输出 JVM 指标，用 Prometheus 监控服务监控其状态。

> **说明：**
> 若已使用 Spring Boot 作为开发框架，请参见 Spring Boot 接入 。

前提条件

- 创建腾讯云容器服务 托管版集群 。
- 使用 容器镜像服务 管理应用镜像。

指标埋点

client_java

client_java 是 Prometheus 官方提供的采集 SDK，提供简洁的 API 自定义指标埋点，还有开箱即用的 JVM 指标，是开发者接入 Prometheus 监控服务的首选方式。

修改应用的依赖及配置

1. 修改 pom 依赖。在 `pom.xml` 文件中添加相关的 Maven 依赖项，1.x 版本做了重构和老版本已经不兼容，优先选择最新版本，示例如下：

```
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>prometheus-metrics-core</artifactId>
    <version>1.3.3</version>
</dependency>
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>prometheus-metrics-instrumentation-jvm</artifactId>
    <version>1.3.3</version>
</dependency>
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>prometheus-metrics-exporter-httpserver</artifactId>
    <version>1.3.3</version>
</dependency>
```

2. 修改代码。初始化并注册 Metrics，示例如下：

```
package org.example;

import io.prometheus.metrics.core.metrics.Counter;
import io.prometheus.metrics.exporter.httpserver.HTTPServer;
import io.prometheus.metrics.instrumentation.jvm.JvmMetrics;

import java.io.IOException;

public class Main {
```

```
    public static void main(String[] args) throws InterruptedException, IOException {
        JvmMetrics.builder().register(); // 初始化并注册 JVM 指标
         // 初始化并注册业务自定义指标
        Counter counter = Counter.builder()
                .name("my_count_total")
                .help("example counter")
                .labelNames("status")
                .register();
        counter.labelValues("ok").inc();
        counter.labelValues("ok").inc();
        counter.labelValues("error").inc();
        // 启动 metrics http server
        HTTPServer server = HTTPServer.builder()
                .port(9400)
                .buildAndStart();
        System.out.println("HTTPServer listening on port http://localhost:" + server.getPort() +
"/metrics")

        Thread.currentThread().join();
    }

}
```

3. 本地验证。本地启动之后，可以通过 `http://localhost:9400/metrics` 访问到 Prometheus 协议的指标数据。

```
http://localhost:9400/metrics
```

将应用发布到腾讯云容器服务上

1. 本地配置 Docker 镜像环境。如果本地之前未配置过 Docker 镜像环境，可以参见容器镜像服务 Docker 镜像操作快速入门 进行配置。若已配置请执行下一步。
2. 打包及上传镜像。

   2.1 在项目根目录下添加 `Dockerfile`，请根据实际项目进行修改。示例如下：

   ```
   FROM openjdk:8-jdk
   WORKDIR /java-demo
   ADD target/java-demo-*.jar /java-demo/java-demo.jar
   CMD ["java","-jar","java-demo.jar"]
   ```

   2.2 打包镜像，在项目根目录下运行如下命令，需要替换对应的 [namespace]、[ImageName] 和 [镜像版本号]。

   ```
   mvn clean package
   docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
   docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
   ```

jmx_exporter

jmx_exporter 是 Prometheus 官方 exporter，把 JVM 原生 MBeans 数据转换为 Prometheus 格式的指标并通过 HTTP 服务暴露出来。jmx_exporter 以 Java Agent 无代码侵入方式运行，但是只能暴露已经注册到 MBeans 上的指标，无法做业务自定义埋点。对于绝大部分的开发者，client_java 是最常用的接入手段。

准备 jmx_exporter 资源

1. 下载 jar 包。在项目 发布页 下载最新版本的 Java Agent Jar 包，这里以1.0.1为例。

```
wget
https://repo.maven.apache.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/1.0.1/jmx_prometh
eus_javaagent-1.0.1.jar
```

2. 准备配置文件。此处使用最少可用配置，各个配置项的详细说明请参见 文档 。

```
rules:
- pattern: ".*"
```

3. 本地验证。jmx_exporter 启动格式是 `-javaagent:<jmx_exporter jar 路径>=< metric 暴露端口>:< jmx_exporter 配置文件路径>` 。

```
java -javaagent:./jmx_prometheus_javaagent-1.0.1.jar=9400:./jmx.yml -jar demo.jar
```

4. 正常启动后，访问 `http://localhost:9400/metrics` 会返回 jvm 开头的指标。

```
 http://localhost:9400/metrics
```

将应用发布到腾讯云容器服务上

1. 本地配置 Docker 镜像环境。如果本地之前未配置过 Docker 镜像环境，请参见容器镜像服务 Docker 镜像操作快速入门 进行配置。若已配置请执行下一步。
2. 打包及上传镜像。

   2.1 在项目根目录下添加 `Dockerfile` ，请根据实际项目进行修改。示例如下：

```
FROM openjdk:8-jdk
WORKDIR /java-demo
# 添加下载的 jmx_exporter jar 包
ADD ./jmx_prometheus_javaagent-1.0.1.jar /java-demo/jmx_prometheus_javaagent-1.0.1.jar
# 添加准备的 jmx_exporter 配置文件
ADD ./jmx.yml  /java-demo/jmx.yml
ADD ./target/application.jar /java-demo/java-demo.jar
# 通过 java agent 注入 jmx_exporter
CMD ["java", "-javaagent:/java-demo/jmx_prometheus_javaagent-1.0.1.jar=9400:/java-
demo/jmx.yml", "-jar", "java-demo.jar"]
```

   2.2 打包镜像，在项目根目录下运行如下命令，需要替换对应的 [namespace]、[ImageName] 和 [镜像版本号]。

```
 docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
 docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[镜像版本号]
```

应用部署

1. 登录 容器服务控制台 ，在左侧菜单栏中集群页面，选择需要部署的容器集群。
2. 通过**工作负载** > **Deployment** 进入 **Deployment** 管理页面，选择对应的**命名空间**进行部署服务，通过 YAML 来创建对应的 Deployment，YAML 配置如下。

> **说明：**
> 如需通过控制台创建，请参见 Spring Boot 接入 。

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
    labels:
      k8s-app: java-demo
    name: java-demo
    namespace: java-demo
spec:
    replicas: 1
    selector:
      matchLabels:
        k8s-app: java-demo
    template:
      metadata:
        labels:
          k8s-app: java-demo
      spec:
       containers:
       - image: ccr.ccs.tencentyun.com/prometheus-demo/java-demo
         imagePullPolicy: Always
         name: java-demo
         ports:
          - containerPort: 9400
            name: metric-port
```

添加采集任务

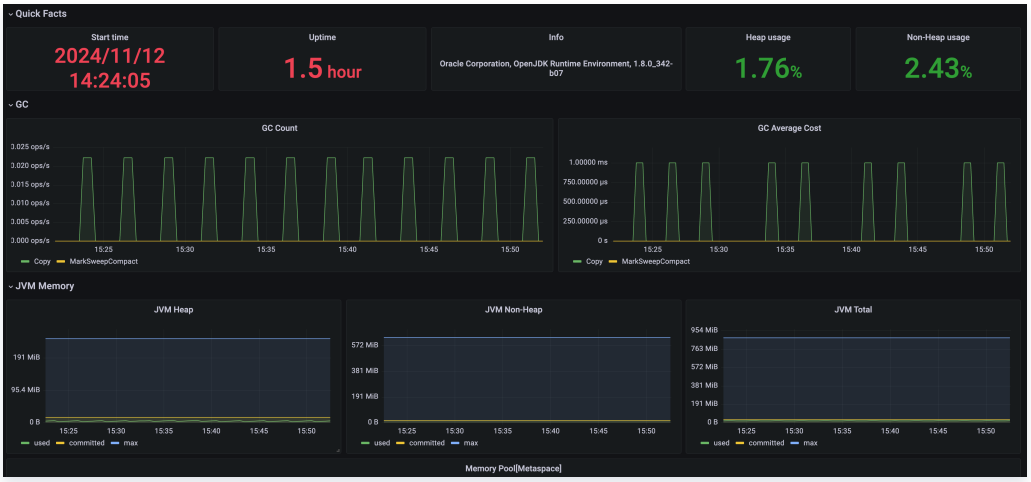
1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中选择 **Prometheus 监控**，选择对应 Prometheus 实例进入管理页面。
3. 选择**数据采集 > 集成容器服务**，进入到容器服务集成管理页面。
4. 选择**数据采集配置 > 新增自定义监控 > yaml 编辑 > PodMonitors** 新增抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: java-demo
  namespace: java-demo
spec:
  namespaceSelector:
    matchNames:
    - java-demo
  podMetricsEndpoints:
  - interval: 15s
    path: /metrics
    port: metric-port
    relabelings:
    - action: replace
      sourceLabels:
      - __meta_kubernetes_pod_label_k8s_app
      targetLabel: application
  selector:
    matchLabels:
      k8s-app: java-demo
```

查看监控

1. 进入对应 Prometheus 实例，在**数据采集** > **集成中心**中找到 **JVM 监控**，安装对应的 Grafana Dashboard 即可开启 JVM 监控大盘。
2. 打开 Prometheus 实例对应的 Grafana 地址，在 `Dashboards` 下查看应用相关的监控大屏。
   - **应用 JVM**：从应用角度出发，查看该应用下所有实例是否存在异常，当发现某个实例有异常时，可以下钻到对应的实例监控。

   - **实例 JVM**：单实例 JVM 详细的监控数据。



Golang 应用接入

最近更新时间：2025-03-18 15:06:02

Prometheus 提供了 官方版 Golang 库 用于采集并暴露监控数据，本文为您介绍如何使用官方版 Golang 库来暴露 Golang runtime 相关的数据，以及其它一些基本简单的示例，并使用 Prometheus监控服务来采集指标展示数据等。

添加指标

> **说明：**
> Golang Client API 相关的文档请参见 GoDoc 。

安装

通过 `go get` 命令来安装相关依赖，示例如下。

```
go get github.com/prometheus/client_golang/prometheus
go get github.com/prometheus/client_golang/prometheus/promauto
go get github.com/prometheus/client_golang/prometheus/promhttp
```

开始（运行时指标）

1. 准备一个 HTTP 服务，路径通常使用 `/metrics` 。可以直接使用 `prometheus/promhttp` 里提供的 `Handler` 函数。
   如下是一个简单的示例应用，通过 `http://localhost:2112/metrics` 暴露 Golang 应用的一些默认指标数据（包括运行时指标、进程相关指标以及构建相关的指标）。

   ```
   package main

   import (
           "net/http"

           "github.com/prometheus/client_golang/prometheus/promhttp"
   )

   func main() {
           http.Handle("/metrics", promhttp.Handler())
           http.ListenAndServe(":2112", nil)
   }
   ```

2. 执行以下命令启动应用。

   ```
   go run main.go
   ```

3. 执行以下命令，访问基础内置指标数据。

   ```
   curl http://localhost:2112/metrics
   ```

应用层面指标

1. 上述示例仅暴露了一些基础的内置指标。应用层面的指标还需要额外添加（后续我们将提供一些 SDK 方便接入）。如下示例暴露了一个名为 `myapp_processed_ops_total` 的 计数类型 指标，用于对目前已经完成的操作进行计数。如下每两秒操作一次，同时计数器加1。

   ```
   package main
   ```

```
import (
        "net/http"
        "time"

        "github.com/prometheus/client_golang/prometheus"
        "github.com/prometheus/client_golang/prometheus/promauto"
        "github.com/prometheus/client_golang/prometheus/promhttp"
)

func recordMetrics() {
        go func() {
                for {
                        opsProcessed.Inc()
                        time.Sleep(2 * time.Second)
                }
        }()
}

var (
        opsProcessed = promauto.NewCounter(prometheus.CounterOpts{
                Name: "myapp_processed_ops_total",
                Help: "The total number of processed events",
        })
)

func main() {
        recordMetrics()

        http.Handle("/metrics", promhttp.Handler())
        http.ListenAndServe(":2112", nil)
}
```

2. 执行以下命令启动应用。

```
go run main.go
```

3. 执行以下命令，访问暴露的指标。

```
curl http://localhost:2112/metrics
```

从输出结果我们可以看到 `myapp_processed_ops_total` 计数器相关的信息，包括帮助文档、类型信息、指标名和当前值，如下所示。

```
# HELP myapp_processed_ops_total The total number of processed events
# TYPE myapp_processed_ops_total counter
myapp_processed_ops_total 666
```

使用 Prometheus 监控服务

步骤1：打包并部署应用

1. Golang 应用一般可以使用如下形式的 Dockerfile（按需修改）。

```
FROM golang:alpine AS builder
RUN apk add --no-cache ca-certificates \
```

```
        make \
        git
COPY . /go-build
RUN cd /go-build && \
        export GO111MODULE=on && \
        export GOPROXY=https://goproxy.io && \
        go build -o 'golang-exe' path/to/main/

FROM alpine
RUN apk add --no-cache tzdata
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs
COPY --from=builder /go-build/golang-exe /usr/bin/golang-exe
ENV TZ Asia/Shanghai
CMD ["golang-exe"]
```

2. 镜像可以使用 腾讯云的镜像仓库，或者使用其它公有或者自有镜像仓库。
3. 需要根据应用类型定义一个 Kubernetes 的资源，这里我们使用 Deployment，示例如下。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: golang-app-demo
  labels:
    app: golang-app-demo
spec:
  replicas: 3
  selector:
    matchLabels:
      app: golang-app-demo
  template:
    metadata:
      labels:
        app: golang-app-demo
    spec:
      containers:
      - name: golang-exe-demo:v1
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

4. 同时需要 Kubernetes Service 做服务发现和负载均衡。

```
apiVersion: v1
kind: Service
metadata:
  name: golang-app-demo
spec:
  selector:
    app: golang-app-demo
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

> ⚠ 注意：

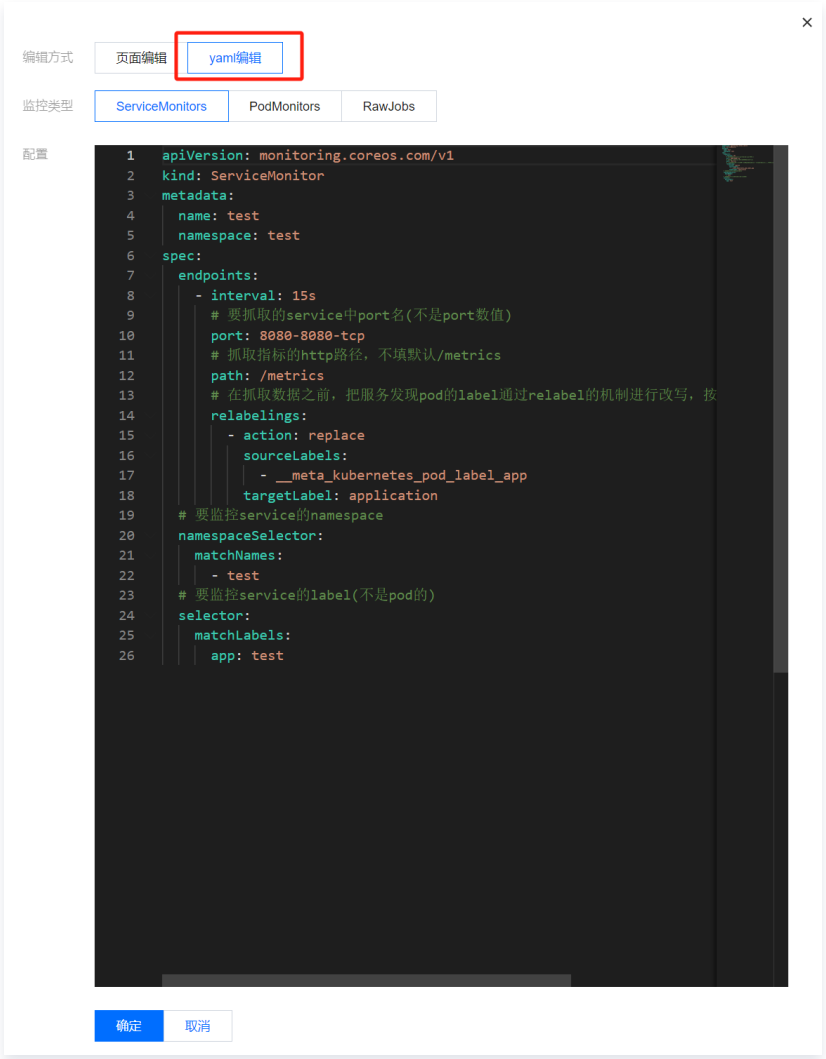
> 必须添加一个 Label 来标明目前的应用，Label 名不一定为 `app`，但是必须有类似含义的 Label 存在，其它名字的 Label 我们可以在后面添加数据采集任务的时候做 relabel 来达成目的。

步骤2：添加数据采集任务

当服务运行起来之后，需要进行如下操作让腾讯云 Prometheus 监控服务发现并采集监控指标：

1. 登录 腾讯云可观测平台 Prometheus 控制台，单击新建的**实例 ID/名称**。
2. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：

3. 进入数据采集配置页面后，单击**新建自定义监控**，在弹出的页面中选择 **yaml 编辑**，如下图所示：



4. 通过此方式添加 `Service Monitor`，目前支持基于 `Labels` 发现对应的目标实例地址，因此可以对一些服务添加特定的 `K8S Labels`，可以使 `Labels` 下的服务都会被 Prometheus 服务自动识别出来，不需要再为每个服务添加采集任务，以上面的例子配置信息如下：

> **说明：**
> `port` 的取值为 `service yaml` 配置文件里的 `spec/ports/name` 对应的值。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: go-demo     # 填写一个唯一名称
  namespace: cm-prometheus  # namespace固定，不要修改
spec:
  endpoints:
  - interval: 30s
    # 填写service yaml中Prometheus Exporter对应的Port的Name
    port: 2112
    # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    path: /metrics
    relabelings:
    # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 app，
    # 我们通过 relabel 的 replace 动作把它替换成了 application
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # 选择要监控service所在的namespace
  namespaceSelector:
    matchNames:
    - golang-demo
    # 填写要监控service的Label值，以定位目标service
  selector:
    matchLabels:
      app: golang-app-demo
```

> ⚠ **注意：**
> 示例中名称为 application 的 Label 必须配置，否则无法使用我们提供一些其它的开箱即用的集成功能。更多高阶用法请参见 ServiceMonitor 或 PodMonitor 。

步骤3：查看监控

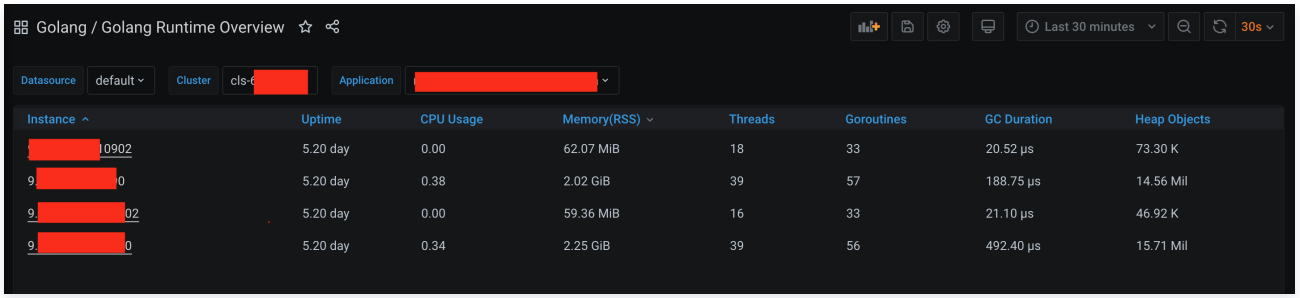
前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 在 Prometheus 实例 列表，找到对应的 Prometheus 实例，单击实例 ID 右侧图标，打开 Grafana，输入您的账号密码，即可进行 Grafana 可视化大屏操作区。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到并单击 **Golang 监控**，选择 **Dashboard > Dashboard 操作 > Dashboard 安装/升级**，单击**安装/升级**，安装对应的 Grafana Dashboard。

3. 进入 Grafana，单击 图表，展开监控面板，单击对应的监控图表名称即可查看监控数据。



总结

本文通过两个示例展示了如何将 Golang 相关的指标暴露给 Prometheus 监控服务，以及如何使用 Grafana 可视化的图表查看监控数据。文档只使用了计数类型 Counter 的指标，对于其它场景可能还需要 Gauge、Histogram 以及 Summary 类型的指标，请参见 指标类型 。
对于其它应用场景，我们会集成更多框架提供更多开箱即用的指标监控、可视化面板以及告警模板。

Exporters 接入
ElasticSearch Exporter 接入

最近更新时间：2025-02-18 09:23:02

操作场景

在使用 ElasticSearch 过程中需要对 ElasticSearch 运行状态进行监控，例如集群及索引状态等，Prometheus 监控服务提供了基于 Exporter 的方式来监控 ElasticSearch 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 ElasticSearch Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 ElasticSearch 是腾讯云 ElasticSearch Service，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

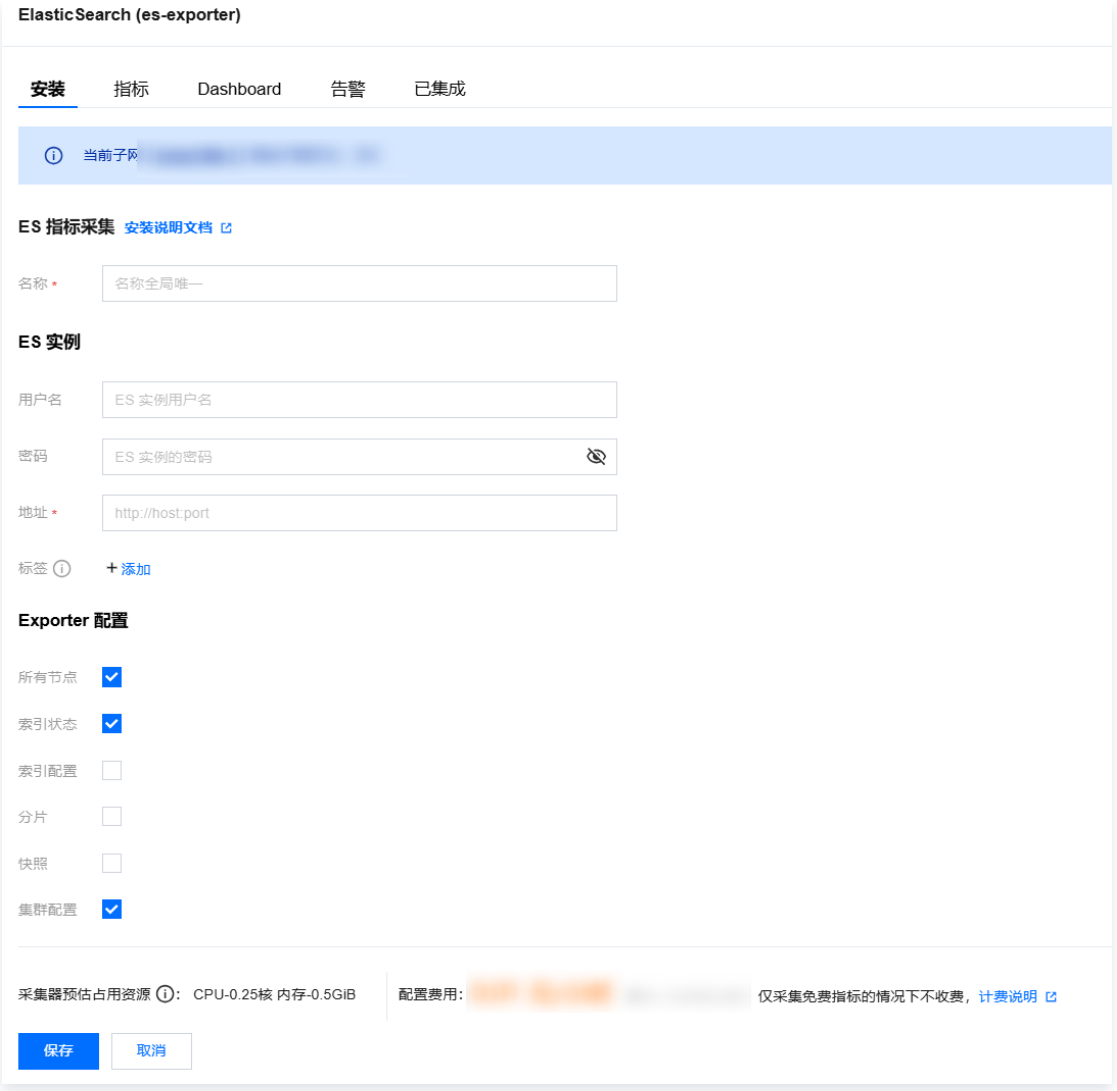
方式一：一键安装（推荐）

前提条件

Prometheus 实例所在私有网络 VPC 与 ElasticSearch 网络相通。

操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集** > **集成中心**。
4. 在**集成中心**找到并单击 **ElasticSearch**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
用户名	ElasticSearch 的用户名称。
密码	ElasticSearch 的密码。
地址	ElasticSearch 的连接地址。
标签	给指标添加自定义 Label。
Exporter 配置	• 所有节点：勾选表示查询集群中所有节点的统计信息；不勾选表示仅查询连接的节点的统计信息。 • 索引状态：勾选表示查询集群中所有索引的统计信息。 • 索引配置：勾选表示查询集群中所有索引配置的统计信息。 • 分片：勾选表示查询集群中所有索引的统计信息，包括分片级别的统计信息（相当于勾选了索引状态）。 • 快照：勾选表示查询集群快照的统计信息。 • 集群配置：勾选表示查询集群配置的统计信息。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

前提条件

在您执行操作前，请确认已满足以下条件：

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台，选择并进入对应的 Prometheus 实例，在**数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。详情可参见指引 关联集群。

操作步骤

步骤1：部署 ElasticSearch Exporter

1. 登录 容器服务控制台。
2. 在左侧菜单栏中选择**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 使用 Secret 管理 ElasticSearch 连接串。

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

4.1 选择**工作负载 > Deployment**，进入 Deployment 页面。

4.2 在页面右上角单击 **YAML 创建**，创建 YAML 配置，配置说明如下：

使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 ElasticSearch Exporter 的时候直接使用 Secret Key，需要调整对应的 URI，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: es-secret-test
  namespace: es-demo
type: Opaque
stringData:
  esURI: you-guess  #对应 ElasticSearch 的 URI
```

5. 部署 ElasticSearch Exporter。

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: es-exporter # 根据业务需要调整成对应的名称
  name: es-exporter # 根据业务需要调整成对应的名称
  namespace: es-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: es-exporter # 根据业务需要调整成对应的名称
```

```
template:
  metadata:
    labels:
      k8s-app: es-exporter # 根据业务需要调整成对应的名称
  spec:
    containers:
    - env:
        - name: ES_URI
          valueFrom:
            secretKeyRef:
              name: es-secret-test # 对应上一步中的 Secret 的名称
              key: esURI # 对应上一步中的 Secret Key
        - name: ES_ALL
          value: "true"
      image: ccr.ccs.tencentyun.com/rig-agent/es-exporter:1.1.0
      imagePullPolicy: IfNotPresent
      name: es-exporter
      ports:
      - containerPort: 9114
        name: metric-port
      securityContext:
        privileged: false
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```

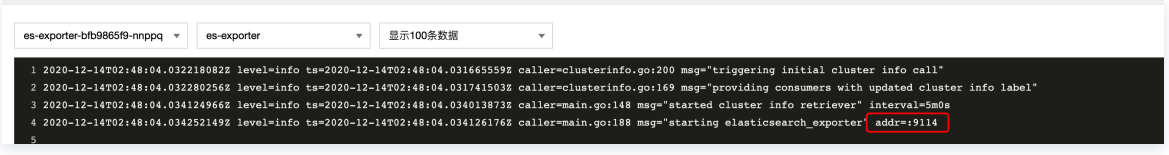
说明

上述示例通过 `ES_ALL` 采集了所有 ElasticSearch 的监控项，可以通过对应的参数进行调整，Exporter 更多详细的参数请参见 elasticsearch_exporter 。

6. 验证。

6.1 在 **Deployment** 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

6.2 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：



6.3 单击 **Pod 管理**页签进入 Pod 页面。

6.4 在右侧的操作项下单击**远程登录**登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 ElasticSearch 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
curl localhost:9114/metrics
```

执行结果如下图所示：

```
# HELP elasticsearch_breakers_estimated_size_bytes Estimated size in
# TYPE elasticsearch_breakers_estimated_size_bytes gauge
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",clus
2.0102643e+07
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",clus
1.9926654e+07
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",clus
1.9685163e+07
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clust
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clust
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clust
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_reques
0
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_reques
1167
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_reques
1167
elasticsearch_breakers_estimated_size_bytes{breaker="parent",cluster=
2.0102643e+07
```

步骤2：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 在**数据采集 > 集成容器服务**页面选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: es-exporter      # 填写一个唯一名称
  namespace: cm-prometheus    # 按量实例：集群的 namespace；包年包月实例(已停止售卖)：namespace 固定，不要
修改
spec:
  namespaceSelector:
    matchNames:
      - es-demo
  podMetricsEndpoints:
  - interval: 30s
    path: /metrics
    port: metric-port
  selector:
    matchLabels:
      k8s-app: es-exporter
```

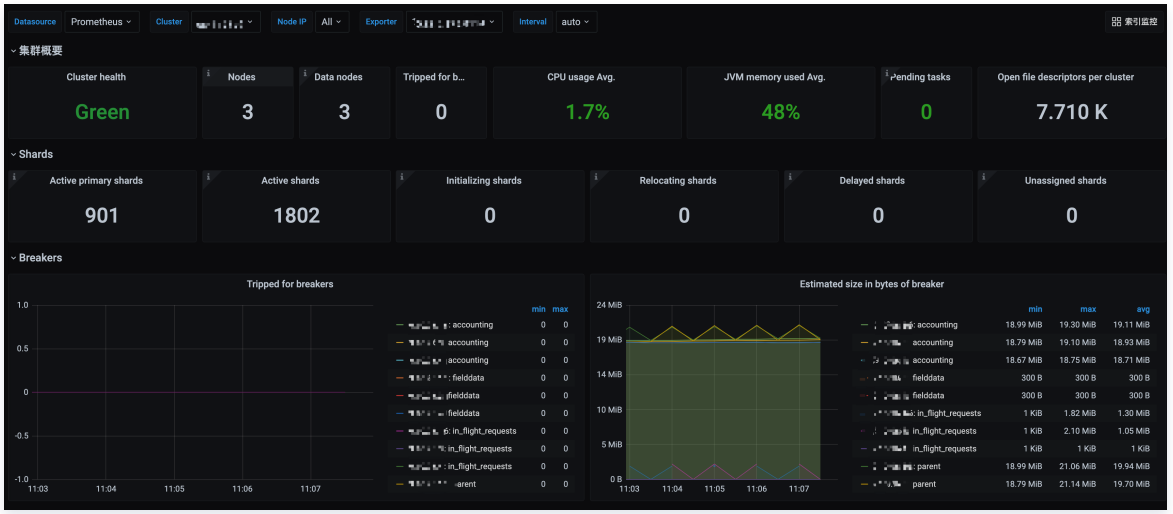
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，进入集成中心页面。找到 ElasticSearch 监控，选择 **Dashboard > Dashboard 操作 > Dashboard 安装/升级**，单击**安装/升级**，安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中单击 Grafana 图标即可自动打开 Kafka 监控大盘，查看实例相关的监控数据，如下图所示：



配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**告警管理** > **告警配置**，单击**新建告警策略**添加相应的告警策略，详情请参见 新建告警策略 。

Kafka Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

在使用 Kafka 过程中需要对 Kafka 运行状态进行监控，例如集群状态、消息消费情况是否有积压等， Prometheus 监控服务提供基于 Exporter 的方式来监控 Kafka 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 Kafka Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 Kafka 是腾讯云 消息队列 CKafka 版，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

方式一：一键安装(推荐)

前提条件

- Prometheus 实例所在私有网络 VPC 与 Kafka 网络相通。
- 需在 Kafka 放通 Prometheus IPv4 地址的读权限，详细步骤可参见 配置 ACL 策略 。

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集**，再点击**集成中心**。
4. 在集成中心找到并单击 **Kafka**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
地址	填写 Kafka Broker 的连接地址。
Kafka 版本	选填，部分特定版本必填，例如 0.10.2.0。
标签	给指标添加自定义 Label。
topic 过滤正则	选填，不填默认采集全部的 topic。填写后只会采集符合正则的 topic。

group 过滤正则	选填，不填默认采集全部的 group。填写后只会采集符合正则的 group。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台 > 选择**对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。详情可参见 关联集群。
- 需在 Kafka 放通 Prometheus IPv4 地址的读权限，同上文 一键安装 的前提条件。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
5. 在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: kafka-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
  name: kafak-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
  namespace: kafka-demo # 集群的 namespace，exporter 会部署在该 namespace 下
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: kafka-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
  template:
    metadata:
      labels:
        k8s-app: kafka-exporter # 根据业务需要调整成对应的名称，建议加上 Kafka 实例的信息，如 ckafka-2vrgx9fd-kafka-exporter
    spec:
      containers:
      - args:
        - --kafka.server=x.x.x.x:9092 # 对应 Kafka 实例的地址信息
        image: ccr.ccs.tencentyun.com/rig-agent/kafka-exporter:v1.3.0
        imagePullPolicy: IfNotPresent
        name: kafka-exporter
        ports:
        - containerPort: 9121
          name: metric-port  # 这个名称在配置抓取任务的时候需要
        securityContext:
```

```
        privileged: false
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```

说明：
Exporter 详细参数请参见 kafka_exporter 。

步骤二：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 进入**集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 自定义监控 > 新建自定义监控 > YAML编辑**，来添加采集配置。
   监控类型选择 `PodMonitors` ，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: kafka-exporter   # 填写一个唯一名称
  namespace: cm-prometheus   # 按量实例：集群的 namespace；包年包月实例(已停止售卖)：namespace 固定，不
要修改
spec:
  podMetricsEndpoints:
  - interval: 30s # 采集间隔
    port: metric-port # 填写步骤一 yaml 中的 spec.template.spec.containers[0].ports[0].name
    path: /metrics # Exporter 的指标采集路径，默认填 /metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'ckafka-xxxxxx' # 调整成对应的 Kafka 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 Kafka 实例 IP
  namespaceSelector:
    matchNames:
    - kafka-demo # 填写 Exporter 所在的 namespace
  selector:  # 填写要监控 pod 的 Label 值，以定位目标 pod
    matchLabels:
      k8s-app: kafka-exporter
```

说明：

> 由于 `Exporter` 和 `Kafka` 部署在不同的服务器上，因此建议通过 Prometheus Relabel 机制将 Kafka 实例的信息放到监控指标中，以便定位问题。

查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到 Kafka 监控，选择 **Dashboard 操作 > Dashboard 安装/升级** 来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可自动打开 Kafka 监控大盘，查看实例相关的监控数据，如下图所示：

配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**告警管理**，可以添加相应的告警策略，详情请参见 新建告警策略 。

Rabbitmq Exporter 接入

最近更新时间：2025-02-08 18:12:23

操作场景

RabbitMQ Exporter 是一个用于监控 RabbitMQ 消息队列监控系统的 Prometheus 插件，用于收集和暴露 RabbitMQ 的性能指标。通过 RabbitMQ Exporter，用户可以实时了解消息队列的运行状况，包括消息的传输速率、队列的大小、连接数等指标信息。腾讯云可观测平台 Prometheus 提供了与 RabbitMQ Exporter 集成及开箱即用的 Grafana 监控大盘。
从 RabbitMQ 3.7.0版本开始，RabbitMQ 内置了 Prometheus 插件，RabbitMQ 可以直接作为 Prometheus 采集目标，用于 RabbitMQ 的性能监控，而 RabbitMQ Exporter 导出指标可以作为内置采集的补充数据。

接入方式：RabbitMQ 内置导出

前提条件

RabbitMQ 版本不低于3.7.0。

启用 RabbitMQ 的 Prometheus 插件

开源 RabbitMQ 开启

开启方式：

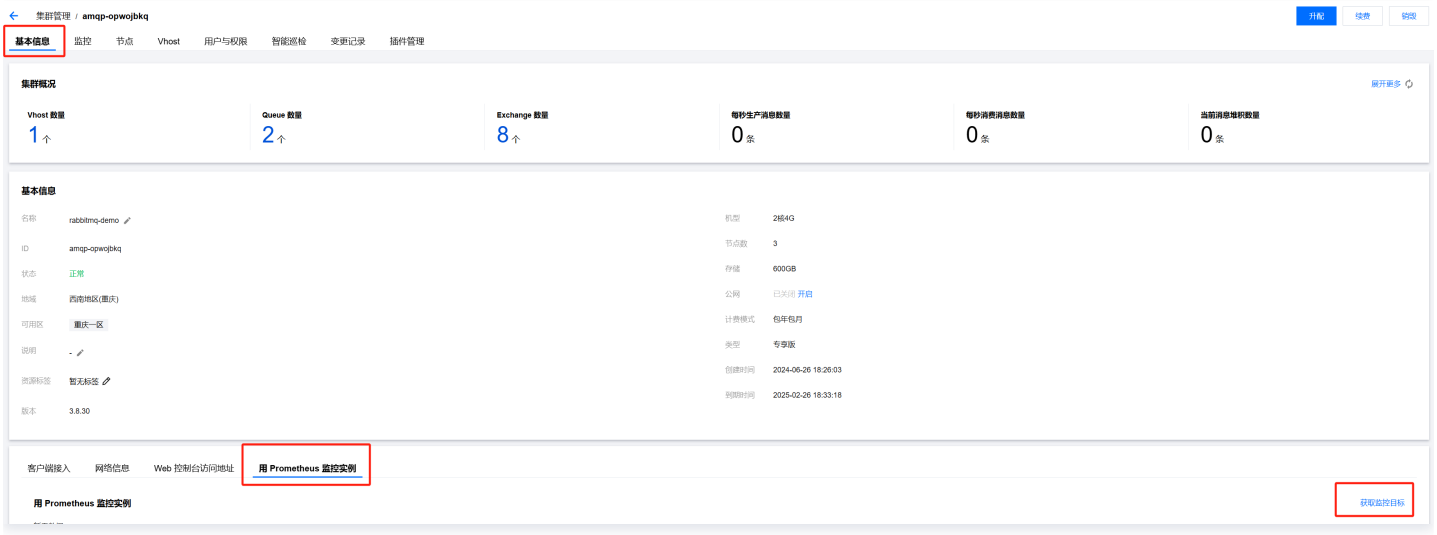
```
rabbitmq-plugins enable rabbitmq_prometheus
```

查看方式：

```
curl localhost:15692/metrics  # 默认开放为15692端口
```

腾讯云消息队列 RabbitMQ 版开启

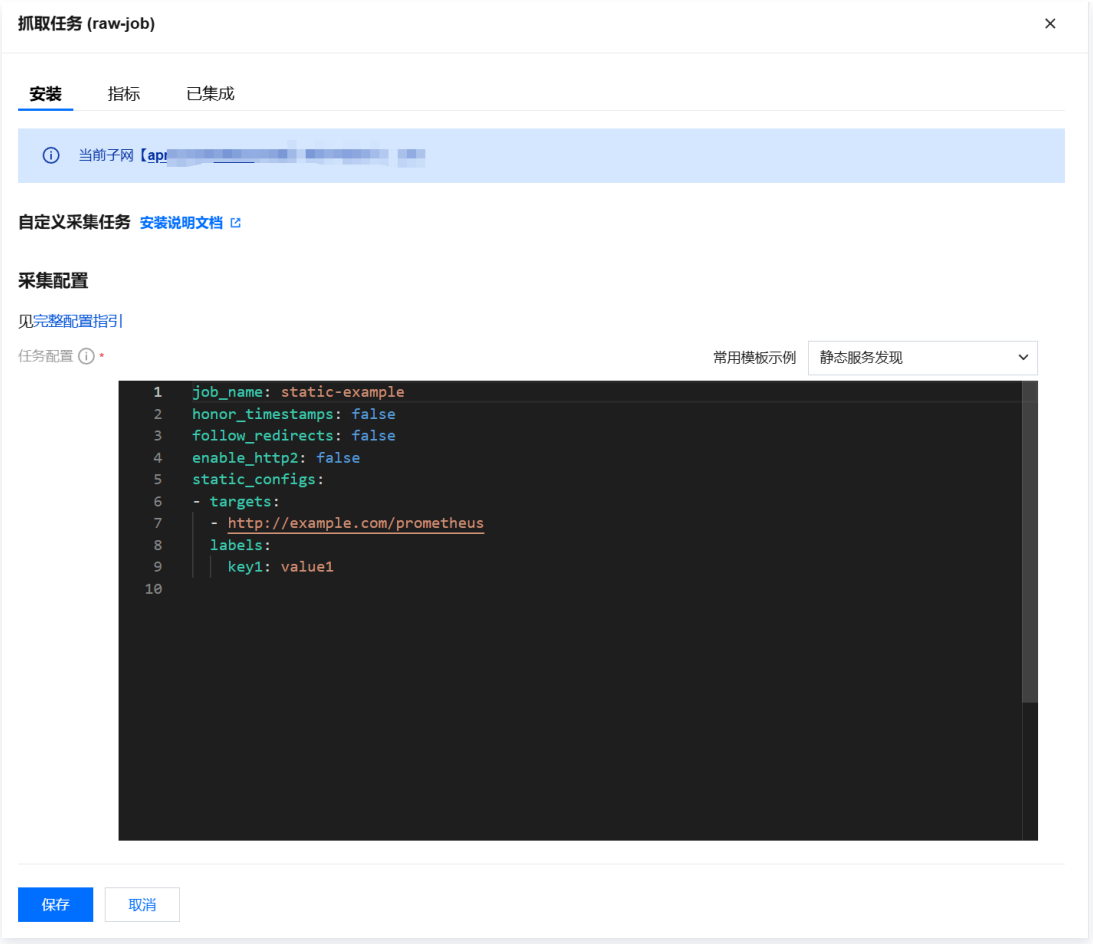
在 腾讯云消息队列 RabbitMQ 版 中找到自己的集群，单击进入集群，在**基本信息 > 用 Prometheus 监控实例**下单击**获取监控目标**，即可得到拉取监控数据地址。



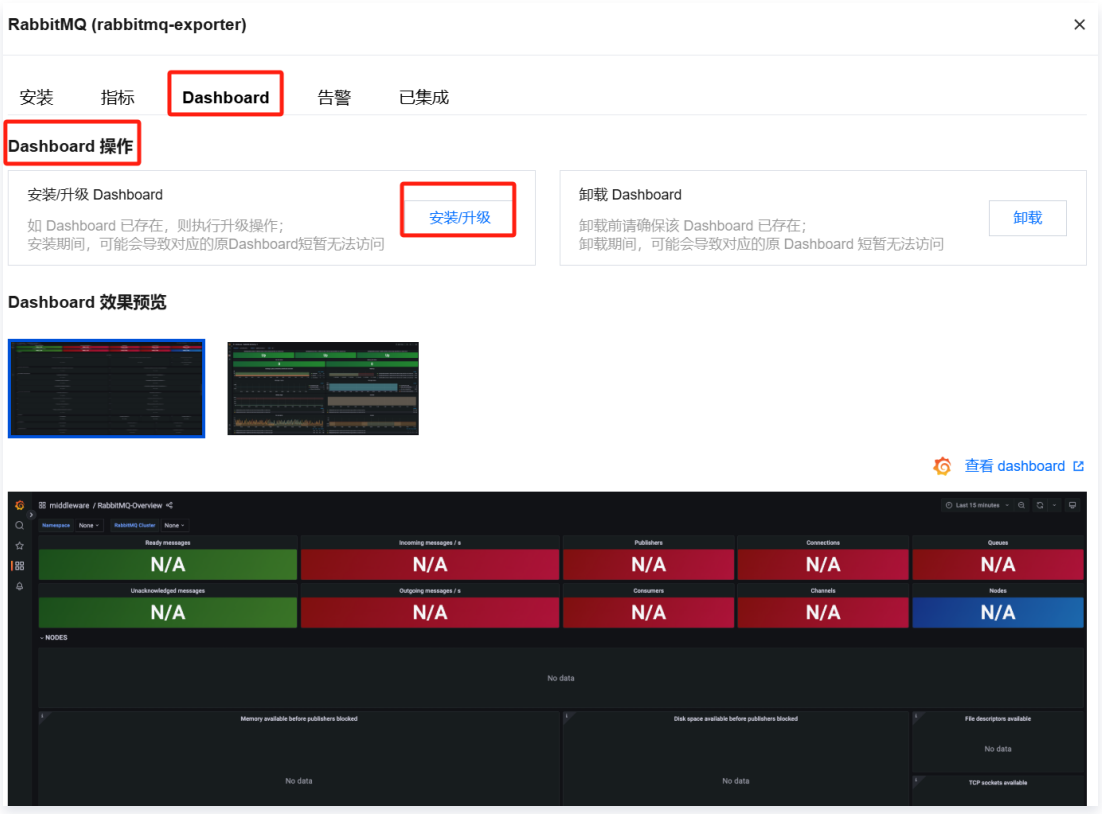
指标采集

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。

4. 在集成中心找到并单击**抓取任务**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



5. 在集成中心找到并单击 **RabbitMQ**，选择 **Dashboard > Dashboard 操作**，然后单击**安装/升级**。

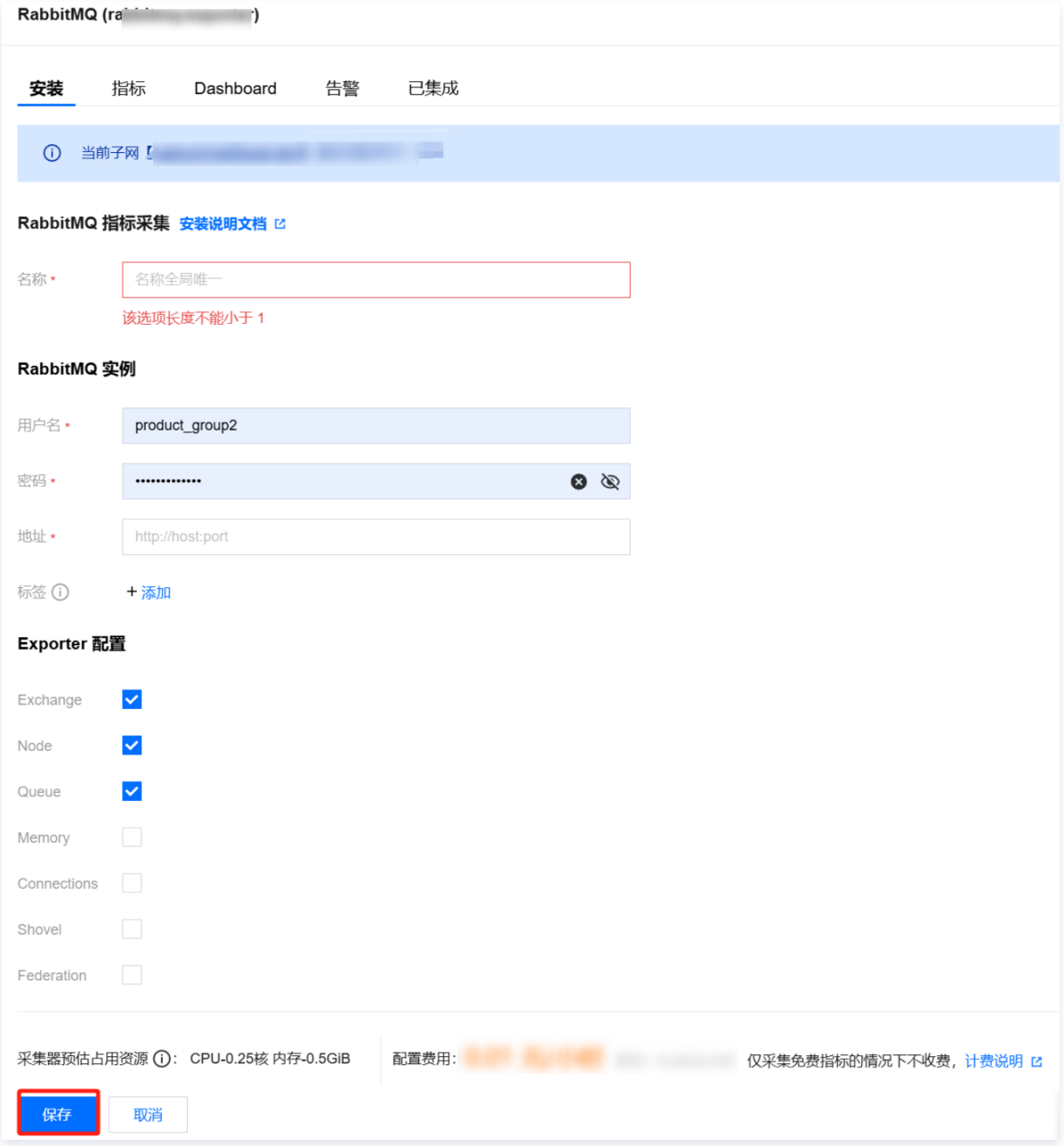


接入方式：开源 Exporter 导出

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **RabbitMQ**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。

用户名	RabbitMQ 用户名称。
密码	RabbitMQ 用户密码。
地址	RabbitMQ web 访问地址。
标签	给指标添加自定义 Label。
Exporter 配置	需要采集的信息选项，包括 Exchange、Node、Queue、Memory、Connections、Shovel、Federation。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间。
- 在 Prometheus 监控服务控制台，选择对应的 Prometheus 实例，选择**数据采集** > **集成容器服务**，然后找到对应容器集群完成关联集群操作。可参见指引 关联集群。

操作步骤

步骤1：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 RabbitMQ Exporter > 验证 步骤完成 Exporter 部署。

步骤2：部署 RabbitMQ Exporter

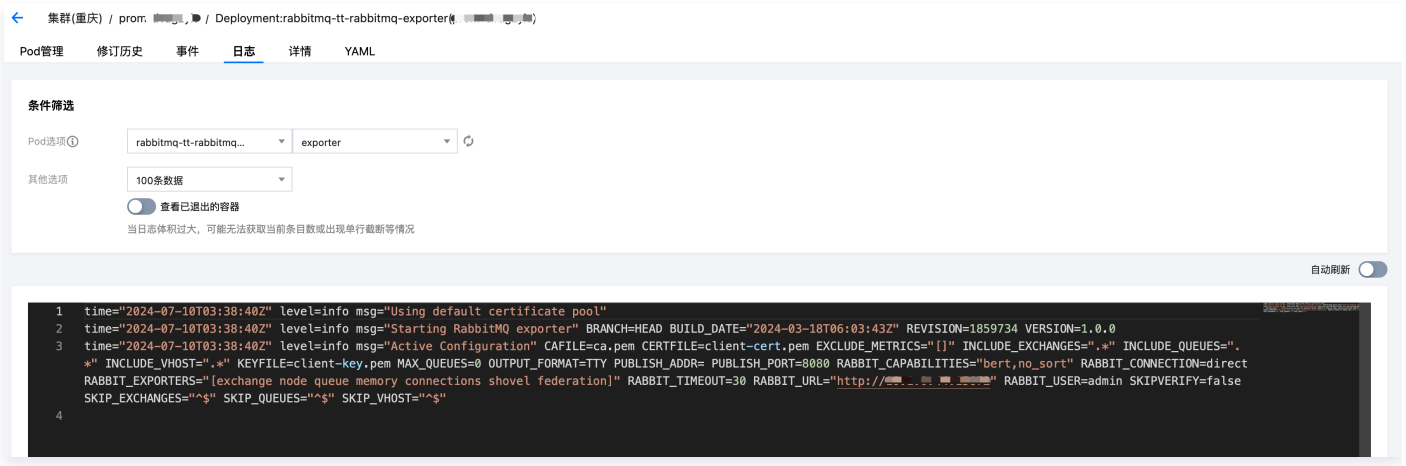
在 Deployment 管理页面，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
  name: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
  namespace: rabbitmq-demo # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
  template:
    metadata:
      labels:
        k8s-app: rabbitmq-exporter  # 根据业务需要调整成对应的名称，建议加上 RabbitMQ 实例的信息
    spec:
      containers:
      - env:
        - name: RABBIT_URL
          value: http://10.1.2.3:15672  # 根据业务需要调整成对应的地址
        - name: RABBIT_USER
```

```
        value: test     # 根据业务进行调整
      - name: RABBIT_PASSWORD
        value: abc123   # 根据业务进行调整
      - name: RABBIT_EXPORTERS
        value: exchange,node,queue,memory,connections,shovel,federation   # 根据业务进行调整
      - name: PUBLISH_PORT
        value: "8080"
      image: ccr.ccs.tencentyun.com/rig-agent/common-image:rabbitmq-exporter-1.0.0
      imagePullPolicy: IfNotPresent
      name: rabbitmq-exporter
      ports:
      - containerPort: 8080
        name: metric-port
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```

步骤3：验证

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



3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 RabbitMQ 指标：

```
curl localhost:8080/metrics
```

执行结果如下图所示：

```
rabbitmq_queue_messages_published_total{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_ram Total number of messages which are resident in ram.
# TYPE rabbitmq_queue_messages_ram gauge
rabbitmq_queue_messages_ram{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_ram{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_ready Number of messages ready to be delivered to clients.
# TYPE rabbitmq_queue_messages_ready gauge
rabbitmq_queue_messages_ready{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_ready{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_ready_global Number of messages ready to be delivered to clients.
# TYPE rabbitmq_queue_messages_ready_global gauge
rabbitmq_queue_messages_ready_global{cluster="amqp-opwojbkq"} 0
# HELP rabbitmq_queue_messages_ready_ram Number of messages from messages_ready which are resident in ram.
# TYPE rabbitmq_queue_messages_ready_ram gauge
rabbitmq_queue_messages_ready_ram{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_ready_ram{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_redelivered_total Count of subset of messages in deliver_get which had the redelivered flag set.
# TYPE rabbitmq_queue_messages_redelivered_total counter
rabbitmq_queue_messages_redelivered_total{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_redelivered_total{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_returned_total Count of messages returned to publisher as unroutable.
# TYPE rabbitmq_queue_messages_returned_total counter
rabbitmq_queue_messages_returned_total{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_returned_total{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_unacknowledged Number of messages delivered to clients but not yet acknowledged.
# TYPE rabbitmq_queue_messages_unacknowledged gauge
rabbitmq_queue_messages_unacknowledged{cluster="amqp-opwojbkq",durable="false",policy="",queue="aliveness-test",self="0",vhost="/"} 0
rabbitmq_queue_messages_unacknowledged{cluster="amqp-opwojbkq",durable="true",policy="",queue="tdmq_event_handle",self="1",vhost="/"} 0
# HELP rabbitmq_queue_messages_unacknowledged_global Number of messages delivered to clients but not yet acknowledged.
# TYPE rabbitmq_queue_messages_unacknowledged_global gauge
rabbitmq_queue_messages_unacknowledged_global{cluster="amqp-opwojbkq"} 0
# HELP rabbitmq_queue_messages_unacknowledged_ram Number of messages from messages_unacknowledged which are resident in ram.
```

步骤4：添加采集任务

1. 登录 Prometheus 监控控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: rabbitmq-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port     # 填写 pod yaml 中 Prometheus Exporter 对应的 Port 的 Name
    path: /metrics  # 填写 Prometheus Exporter 对应的 Path 的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 RabbitMQ 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 RabbitMQ 实例 IP
  namespaceSelector:    # 选择要监控 pod 所在的 namespace
    matchNames:
    - rabbitmq-demo
  selector:  # 填写要监控 pod 的 Label 值，以定位目标 pod
    matchLabels:
      k8s-app: rabbitmq-exporter
```

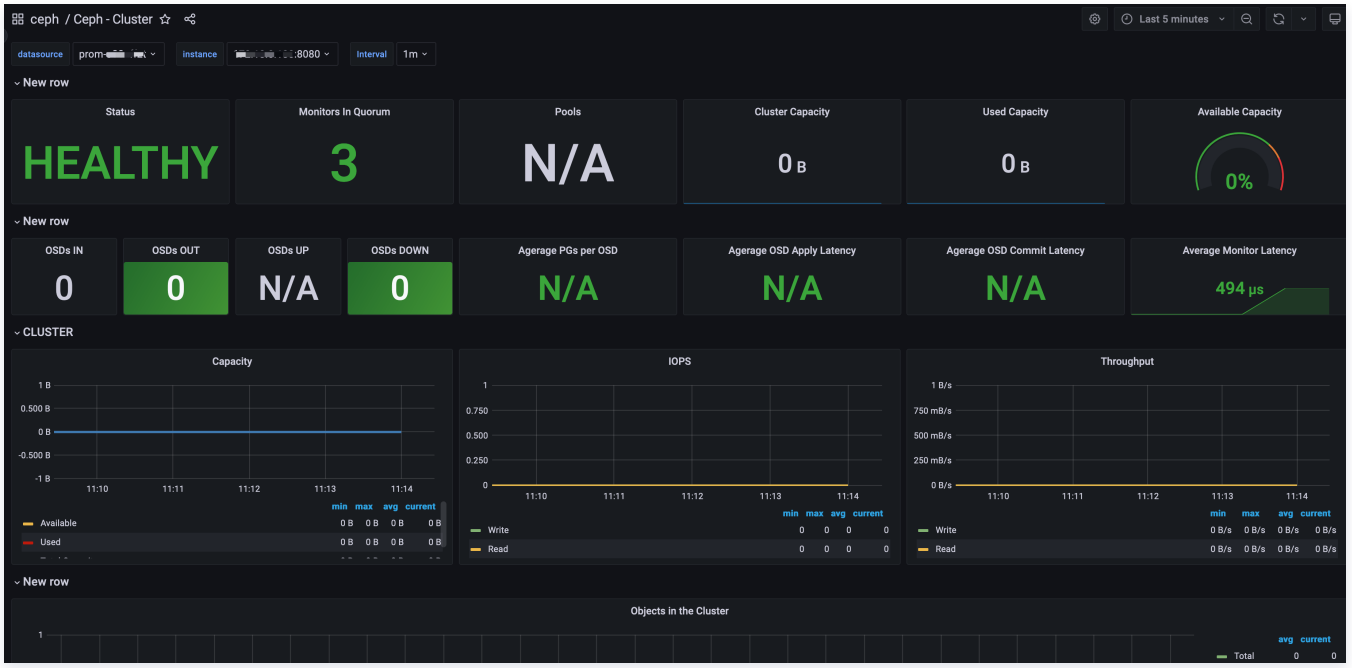
查看监控

前提条件

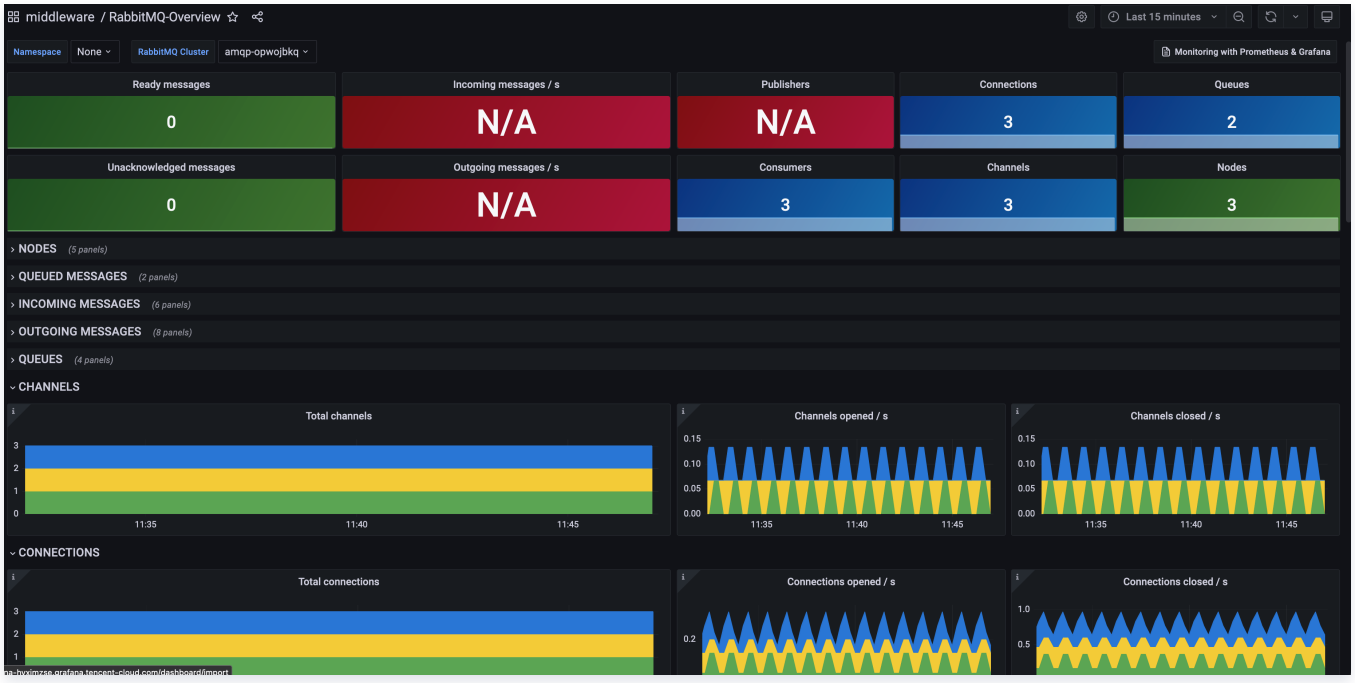
Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 Grafana 地址，打开并登录，然后在 middleware 文件夹中找到 RabbitMQ Monitoring 监控面板，查看实例相关监控数据，如下图所示：



3. RabbitMQ 内置监控可在 middleware 文件夹下的 RabbitMQ-Overview 面板中查看。



配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：RabbitMQ Exporter 环境变量配置

名称	默认	描述
RABBIT_URL	http://127.0.0.1:15672	RabbitMQ 管理插件 URL。
RABBIT_USER	guest	RabbitMQ 管理插件的用户名。
RABBIT_PASSWORD	guest	RabbitMQ 管理插件的密码。
RABBIT_CONNECTION	direct	RabbitMQ 连接方式，direct 或者 loadbalancer。
RABBIT_USER_FILE	-	带有用户名的文件位置（对 docker secrets 有用）。
RABBIT_PASSWORD_FILE	-	带有密码的文件的位置（对 docker secrets 有用）。
PUBLISH_PORT	9419	Exporter 导出指标端口，默认9419。
PUBLISH_ADDR	""	Exporter 导出指标监听地址，默认为""。
OUTPUT_FORMAT	TTY	日志输出格式，支持TTY和JSON。
LOG_LEVEL	info	日志级别。可能的值：“debug”、“info”、“warning”、“error”、“fatal”或“panic”，默认 info。
CAFILE	ca.pem	访问管理插件的根证书路径。如果使用自签名证书则需要。如果文件不存在将被忽略。
CERTFILE	client-cert.pem	用于验证导出器真实性的客户端证书路径。如果文件不存在，将被忽略。
KEYFILE	client-key.pem	与证书一起使用以验证导出者真实性的私钥路径。如果文件不存在，将被忽略。
SKIPVERIFY	false	true/0 将忽略管理插件的证书错误。
SKIP_VHOST	^$	正则表达式，匹配的虚拟主机名不会被导出。首先执行 INCLUDE_VHOST，然后执行 SKIP_VHOST。适用于队列和交换器。
INCLUDE_VHOST	.*	正则表达式虚拟主机过滤器。仅导出匹配的虚拟主机。适用于队列和交换机。
INCLUDE_QUEUES	.*	正则表达式队列过滤器。仅导出匹配的名称。
SKIP_QUEUES	^$	正则表达式，匹配的队列名称不会被导出（对于短暂的 rpc 队列很有用）。首先执行 INCLUDE，然后执行 SKIP。
INCLUDE_EXCHANGES	.*	正则表达式交换过滤器。（仅导出匹配的虚拟主机中的交换）。
SKIP_EXCHANGES	^$	正则表达式，匹配的交易所名称不会被导出。首先执行 INCLUDE，然后执行 SKIP。
RABBIT_CAPABILITIES	bert,no_sort	目标 RabbitMQ 服务器支持的扩展抓取功能的逗号分隔列表。
RABBIT_EXPORTERS	exchange,node,queue	已启用模块列表。可能的模块：连接、铲子、联合、交换、节点、队列、内存。
RABBIT_TIMEOUT	30	从管理插件检索数据的超时时间（秒）。

MAX_QUEUES	0	删除指标之前的最大队列数（如果设置为0则禁用）。
EXCLUDE_METRICS	-	要从导出中排除的指标名称。以逗号分隔。

Fluentd/FluentBit 接入

最近更新时间：2025-02-26 16:29:52

Fluentd 是一个开源的数据收集器，旨在简化日志管理和分析的过程。它提供了一个统一的日志层，能够收集、转换并将数据发送到多种目的地。
Fluentd 的设计哲学是提供简单、可靠和灵活的日志管理解决方案，使得它能够轻松地与各种数据源和最终存储系统集成。
Fluentd 的核心特点包括其插件架构，这使得它能够通过安装不同的插件来支持广泛的输入/输出源，包括文件、日志、数据库和 HTTP 源等。此外，Fluentd 还支持数据的实时处理，包括过滤、修改和丰富数据，以满足特定的日志管理需求。
使用 Fluentd 可以帮助组织减少日志管理的复杂性，提高数据处理的效率，便于日志数据的分析和监控。无论是在云环境还是在本地部署，Fluentd 都能提供灵活的日志管理解决方案，帮助企业更好地理解和利用他们的数据。
Fluentd 和 Fluent Bit 都可以用作聚合器或转发器，它们可以相互补充或用作独立解决方案。近年来，出于性能和兼容性原因，云提供商从 Fluentd 转向 Fluent Bit。Fluent Bit 现在被认为是下一代解决方案。
本文将为您介绍各类部署方式部署的 Fluentd 组件如何暴露 Prometheus 指标数据，并通过腾讯云可观测平台-Prometheus监控服务来采集指标并展示数据。Prometheus 监控服务支持提供预设的 Fluentd 监控面板，以方便您使用。

Fluentd

Fluentd 部署方式

Fluentd 在云环境下部署有很多方式，本文将介绍基于以下四种部署方式进行监控接入：

- 使用 Fluentd 镜像自定义部署
- 使用 Fluentd Daemonset 镜像进行部署
- 使用 Fluentd Operator 进行管理 Fluentd
- 使用 Logging Operator 部署管理 Fluentd

Fluentd 镜像自定义部署

部署流程

1. 获取或者自制安装符合业务使用插件的 Fluentd 镜像。
2. 创建 Fluentd 配置，进行输入、过滤、输出等插件的配置。
3. 使用 Fluentd 配置和镜像创建 Fluentd 负载。
4. 查看 Fluentd 运行情况。

指标导出

方式一：Prometheus 插件导出（推荐）

1. Fluentd 官方镜像是没有 Prometheus 插件的，需要安装，您可以使用以下 Dockerfile，来输出包含 Prometheus 插件 的 Fluentd 镜像。

```
FROM fluent/fluentd:latest

RUN fluent-gem install fluent-plugin-prometheus
## 对于 td-agent：
## RUN td-agent-gem install fluent-plugin-prometheus
```

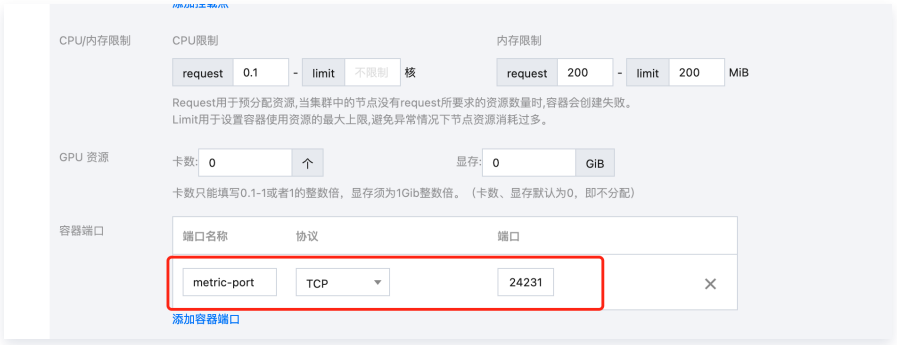
若您已经有 DockerFile，则在其中加入安装 fluent-plugin-prometheus 插件命令即可。

```
RUN fluent-gem install fluent-plugin-prometheus
## 对于 td-agent：
## RUN td-agent-gem install fluent-plugin-prometheus
```

2. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluentd 配置，在其中添加 prometheus 相关输入插件，导出 prometheus 指标。

```
# expose metrics in prometheus format

<source>
  @type prometheus
  bind 0.0.0.0
  port 24231
  metrics_path /metrics
</source>

<source>
  @type prometheus_output_monitor
  interval 10
  @id in_prometheus_output_monitor
</source>
```

3. 在**工作负载**中找到您的 Fluentd 负载，更新镜像为第一步中输出的镜像，然后再给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。



方式二：Exporter 导出

1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluentd 配置，在其中添加 monitor-agent 相关输入插件，导出监控数据。

```
<source>
    @type monitor_agent
    bind 0.0.0.0
    port 24220
</source>
```

2. 部署 fluentd-exporter 导出 prometheus 指标。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/instance: fluentd-demo # 根据业务需要调整成对应的名称，建议加上 Fluentd 实例的信息
    app.kubernetes.io/name: fluentd-exporter
  name: fluentd-demo-fluentd-exporter  # 根据业务需要调整成对应的名称，建议加上 Fluentd 实例的信息，
  namespace: cm-prometheus  # 根据业务调整命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/instance: fluentd-test
      app.kubernetes.io/name: fluentd-exporter
```

```
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app.kubernetes.io/instance: fluentd-demo # 根据业务需要调整成对应的名称，建议加上 fluentd 实例的
信息
        app.kubernetes.io/job: fluentd-demo
        app.kubernetes.io/name: fluentd-exporter
    spec:
      containers:
      - args:
        - --telemetry.address=:8080  ## 指标开放到 8080端口
        - --scrape_uri=http://172.0.0.1:24220/api/plugins.json  ## 地址使用 Fluentd pod 实例 IP
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:fluentd_exporter-latest
        imagePullPolicy: IfNotPresent
        name: fluentd-exporter
        ports:
        - containerPort: 8080
          name: metric-port  # 这个名称在配置抓取任务的时候需要
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

Fluentd Daemonset 镜像部署

部署流程

根据实际业务使用相应的 fluentd-k8s-daemonset，下面以输出到 elasticsearch 的 fluentd-daemonset-elasticsearch 为例，介绍其部署流程：

1. 在 fluentd-kubernetes-daemonset 项目的 配置生成模板目录 下找到配置文件的 erb 文件，根据业务进行配置的调整，然后生成各配置文件。或者使用 Docker 镜像目录 中已经生成出来的配置，根据业务调整得到最终配置。
2. 使用 fluentd-daemonset-elasticsearch.yaml（根据实际业务调整 host、端口、鉴权等信息）部署输出到 elasticsearch 的 fluentd。
3. 查看 fluentd 运行情况。

指标导出

Fluentd Daemonset 自带 Prometheus 插件，同时默认配置已进行 Prometheus 指标导出。

```
root@fluentd-94ttz:/fluentd/etc# cat fluent.conf

# AUTOMATICALLY GENERATED
# DO NOT EDIT THIS FILE DIRECTLY, USE /templates/conf/fluent.conf.erb

@include "#{ENV['FLUENTD_SYSTEMD_CONF'] || 'systemd'}.conf"
@include "#{ENV['FLUENTD_PROMETHEUS_CONF'] || 'prometheus'}.conf"
@include kubernetes.conf
@include conf.d/*.conf

<match **>
   @type elasticsearch
   @id out_es
   @log_level info
```

prometheus.conf 默认导出配置。

```
# AUTOMATICALLY GENERATED
# DO NOT EDIT THIS FILE DIRECTLY, USE /templates/conf/prometheus.conf.erb

# Prometheus metric exposed on 0.0.0.0:24231/metrics
<source>
  @type prometheus
  @id in_prometheus
  bind "#{ENV['FLUENTD_PROMETHEUS_BIND'] || '0.0.0.0'}"
  port "#{ENV['FLUENTD_PROMETHEUS_PORT'] || '24231'}"
  metrics_path "#{ENV['FLUENTD_PROMETHEUS_PATH'] || '/metrics'}"
</source>

<source>
  @type prometheus_output_monitor
  @id in_prometheus_output_monitor
</source>
```

Fluentd Operator 部署管理

部署流程

1. 使用 Fluentd Operator 管理部署 Fluentd 之前需要先部署 Fluent Operator。部署方式有两种：一种为 yaml 部署，另一种为 helm 部署，详情请参见 开始安装 。
2. 创建 ClusterInput、ClusterFilter、ClusterOutput CR 进行输入、输出、过滤配置。
3. 创建 ClusterFluentdConfig CR，将上述组件关联进来。
4. 挂载 ClusterFluentdConfig 创建 Fluentd CR，Operator 会根据 CR 创建出 Fluentd。
5. 查看 Fluentd 运行情况。

Fluentd 指标导出

由于目前 Fluentd Operator 只支持 Fluent Bit Prometheus 指标的导出，Fluentd 暂不支持，故而 Fluentd 需使用 Fluentd 镜像自定义部署指标导出 。

Logging Operator 部署管理

部署流程

1. 使用 Logging Operator 管理部署 Fluentd 之前需要先部署 Logging Operator，详情请参见 示例 。
2. 创建 FluentbitAgent CR，Operator 会根据该 CR 进行 Fluentd 部署。

3. 创建 Flow、Output CR，进行 Fluentd 配置。
4. 查看 Fluentd 运行情况。

指标导出

Logging Operator 与 Prometheus 有良好的适配性，在 Logging CR 添加如下配置即可导出指标。

```
spec:
  fluentdSpec:
    metrics:
      serviceMonitor: true
  fluentbitSpec:
    metrics:
      serviceMonitor: true
```

导出之后配置 ServiceMonitor 即可完成指标采集。

Fluent Bit

Fluent Bit 官方镜像本身就支持导出 Prometheus 指标能力，只需要按需开放即可。

镜像部署指标导出

方法一：HTTP Server 导出

1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluent Bit 配置，在其中添加如下配置：

```
[SERVICE]
    HTTP_Server  On     # 若存在 HTTP_Server，但状态为Off，需改为On
    HTTP_Listen  0.0.0.0
    HTTP_PORT    2020   # 端口可自行调整

[INPUT]
    Name cpu

[OUTPUT]
    Name  stdout
    Match *
```

2. Prometheus 指标导出验证。

```
curl -s http://172.0.0.1:2020/api/v2/metrics/prometheus
```

指标输出：

```
# TYPE fluentbit_input_storage_chunks_down gauge
fluentbit_input_storage_chunks_down{name="fluentbit_metrics.1"} 0
# HELP fluentbit_input_storage_chunks_busy Total number of chunks in a busy state.
# TYPE fluentbit_input_storage_chunks_busy gauge
fluentbit_input_storage_chunks_busy{name="fluentbit_metrics.1"} 0
# HELP fluentbit_input_storage_chunks_busy_bytes Total number of bytes used by chunks in a busy state.
# TYPE fluentbit_input_storage_chunks_busy_bytes gauge
fluentbit_input_storage_chunks_busy_bytes{name="fluentbit_metrics.1"} 0
# HELP fluentbit_output_upstream_total_connections Total Connection count.
# TYPE fluentbit_output_upstream_total_connections gauge
fluentbit_output_upstream_total_connections{name="stdout.0"} 0
# HELP fluentbit_output_upstream_busy_connections Busy Connection count.
# TYPE fluentbit_output_upstream_busy_connections gauge
fluentbit_output_upstream_busy_connections{name="stdout.0"} 0
# HELP fluentbit_output_chunk_available_capacity_percent Available chunk capacity (percent)
# TYPE fluentbit_output_chunk_available_capacity_percent gauge
fluentbit_output_chunk_available_capacity_percent{name="stdout.0"} 100
# HELP fluentbit_output_upstream_total_connections Total Connection count.
# TYPE fluentbit_output_upstream_total_connections gauge
fluentbit_output_upstream_total_connections{name="prometheus_exporter.1"} 0
# HELP fluentbit_output_upstream_busy_connections Busy Connection count.
# TYPE fluentbit_output_upstream_busy_connections gauge
```

3. 在**工作负载**中找到您的 Fluent Bit 负载，给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。

方法二：Fluent Bit Metrics 导出

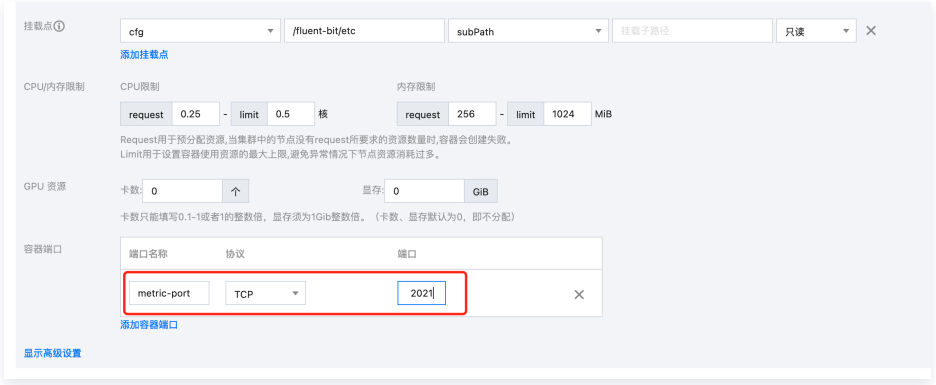
1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，在**配置管理**中找到您的 Fluent Bit 配置，在其中添加如下配置：

```
[SERVICE]
    flush           1
    log_level       info

[INPUT]
    name            fluentbit_metrics
    tag             internal_metrics
    scrape_interval 2

[OUTPUT]
    name            prometheus_exporter
    match           internal_metrics
    host            0.0.0.0
    port            2021
```

2. Prometheus 指标导出验证。

```
curl -s http://172.0.0.:2021/metrics
```

3. 在**工作负载**中找到您的 Fluent Bit 负载，给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。



Fluentd Operator 部署指标导出

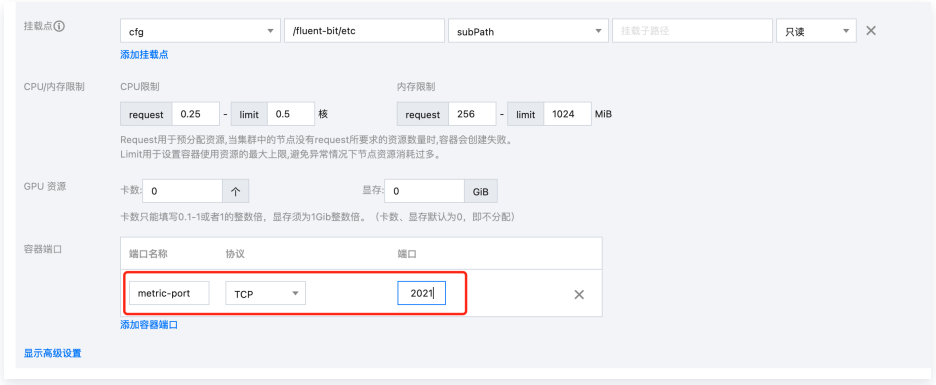
1. 登录 腾讯云容器服务，单击集群名称/ID 进入您的集群，添加如下 ClusterInput CR：

```
apiVersion: fluentbit.fluent.io/v1alpha2
kind: ClusterInput
metadata:
  name: fluentbit_metrics
  labels:
    fluentbit.fluent.io/enabled: "true"
    fluentbit.fluent.io/mode: "fluentbit"
spec:
  tag: internal_metrics
  scrape_interval: 2
  scrape_on_start: true
```

添加如下 ClusterOutput CR：

```
apiVersion: fluentbit.fluent.io/v1alpha2
kind: ClusterOutput
metadata:
  name: prometheus-exporter
  labels:
    fluentbit.fluent.io/enabled: "true"
    fluentbit.fluent.io/mode: "fluentbit"
spec:
  match: internal_metrics
  metricsExporter:
    host: "0.0.0.0"
    port: 2021
    addLabels:
      app: "fluentbit"
```

2. 将上述 CR Selector 到 ClusterFluentdConfig CR 中。
3. 在**工作负载**中找到您的 Fluent Bit 负载，给 Pod 添加容器端口，该容器端口名称会在 PodMonitor 采集中使用。



Logging Operator 部署指标导出

Logging Operator 与 Prometheus 有良好的适配性，在 Logging CR 添加如下配置即可导出指标。

```
spec:
  fluentdSpec:
    metrics:
      serviceMonitor: true
  fluentbitSpec:
    metrics:
      serviceMonitor: true
```

导出之后配置 ServiceMonitor 即可完成指标采集。

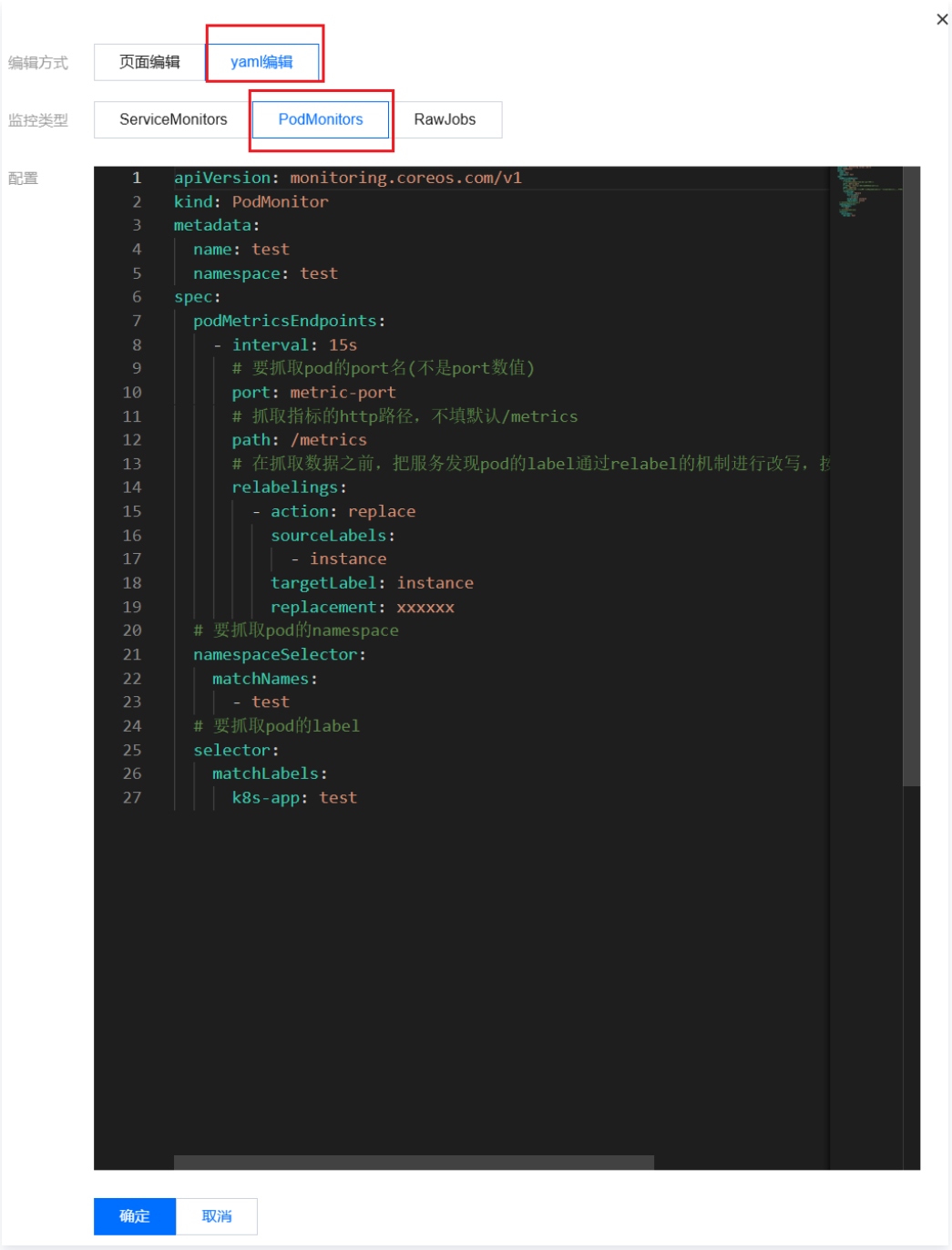
接入 Prometheus 监控

Podmonitor 采集（推荐）

在 Fluentd 存在动态扩缩容场景下，为了能够动态服务发现，需要使用 Podmonitor 来进行指标采集：

1. 登录 腾讯云可观测平台 > Prometheus 控制台，单击新建的**实例 ID/名称**。
2. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：

3. 进入数据采集配置页面后，单击**新建自定义监控**，在弹出的页面中选择 **yaml编辑**，监控类型选择 **PodMonitors**，如下图所示：



4. 通过此方式添加 `Pod Monitor`，目前支持基于 `Labels` 发现对应的目标实例地址，因此可以对一些负载添加特定的 `K8S Labels`，可以使 `Labels` 下负载的 Pod 都会被 Prometheus 服务自动识别出来，不需要再为每个服务添加采集任务，以上面的例子配置信息如下：

> **说明：**
> `port` 的取值为 `pod yaml` 配置文件里的 `spec/ports/name` 对应的值。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: fluentd-demo     # 填写一个唯一名称
  namespace: cm-prometheus   # 根据业务调整命名空间
spec:
  podMetricsEndpoints:
  - interval: 15s
```

```
    # 要抓取pod的port名（不是port数值）
    port: metric-port
    # 抓取指标的http路径，不填默认/metrics，Fluent Bit 方式一为 /api/v2/metrics/prometheus
    path: /metrics
    # 在抓取数据之前，把服务发现pod的label通过relabel的机制进行改写，按顺序执行多个relabel规则
    # 根据业务实际使用进行配置，也可以不进行relabel配置
    relabelings:
      - action: replace
        sourceLabels:
          - instance
        targetLabel: instance
        replacement: xxxxxx
  # 选择要监控pod所在的namespace
  namespaceSelector:
    matchNames:
    - fluentd-demo
  # 填写要监控pod的Label值，以定位目标pod
  selector:
    matchLabels:
      app: fluentd-demo
```

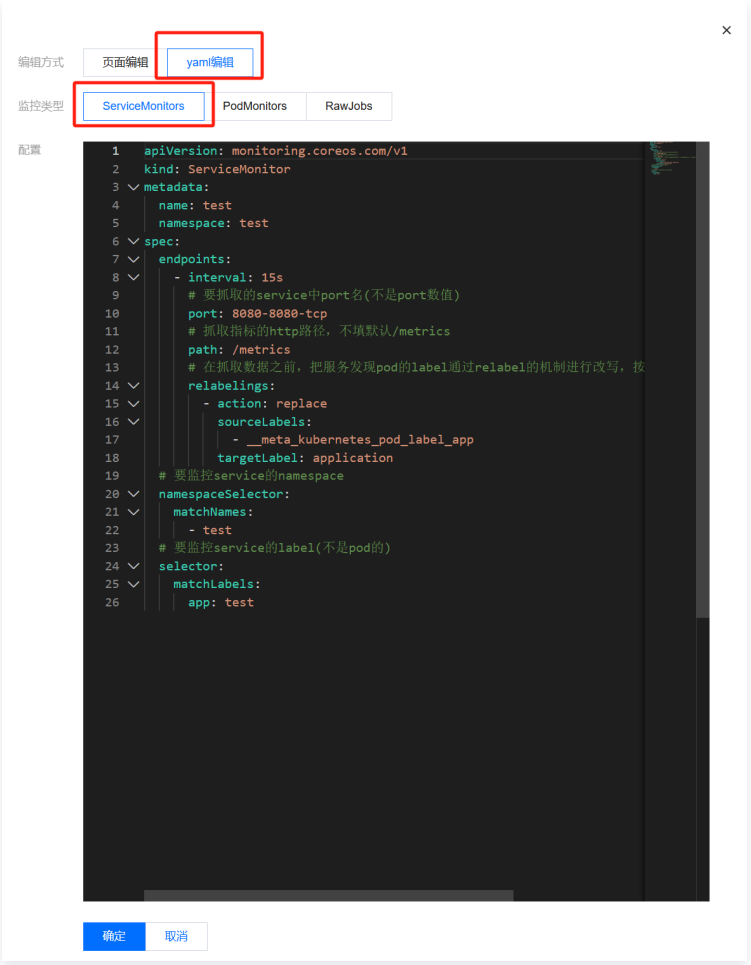
> **注意：**
> 示例中名称为 application 的 Label 必须配置，否则无法使用我们提供一些其它的开箱即用的集成功能。更多高阶用法请参见 ServiceMonitor 或 PodMonitor 。

ServiceMonitor 采集（Logging Operator 使用）

Logging Operator 部署 Fluent Bit 场景中，指标导出是使用 ServiceMonitor 的方式，需要使用 ServiceMonitor 进行指标采集：

1. 登录 腾讯云可观测平台 > Prometheus 控制台，单击新建的**实例 ID/名称**。
2. 在**数据采集**页面，选择**集成容器服务**，单击集群右侧的**数据采集配置**，如下图所示：

3. 进入数据采集配置页面后，单击**新建自定义监控**，在弹出的页面中选择 **yaml 编辑**，监控类型选择 **ServiceMonitors**，如下图所示：



4. 通过此方式添加 `Service Monitor`，目前支持基于 `Labels` 发现对应的目标实例地址，因此可以对一些服务添加特定的 `K8S Labels`，可以使 `Labels` 下的服务都会被 Prometheus 服务自动识别出来，不需要再为每个服务添加采集任务，以上面的例子配置信息如下：

> **说明：**
> `port` 的取值为 `service yaml` 配置文件里的 `spec/ports/name` 对应的值。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: fluent-bit-demo     # 填写一个唯一名称
  namespace: cm-prometheus  # 根据业务调整命名空间
spec:
  endpoints:
  - interval: 30s
    # 填写service yaml中Prometheus Exporter对应的Port的Name
    port: metrics
    # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    path: /metrics
    relabelings:
    # ** 必须要有一个 label 为 application，这里假设 k8s 有一个 label 为 app，
    # 我们通过 relabel 的 replace 动作把它替换成了 application
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # 选择要监控service所在的namespace
  namespaceSelector:
    matchNames:
```

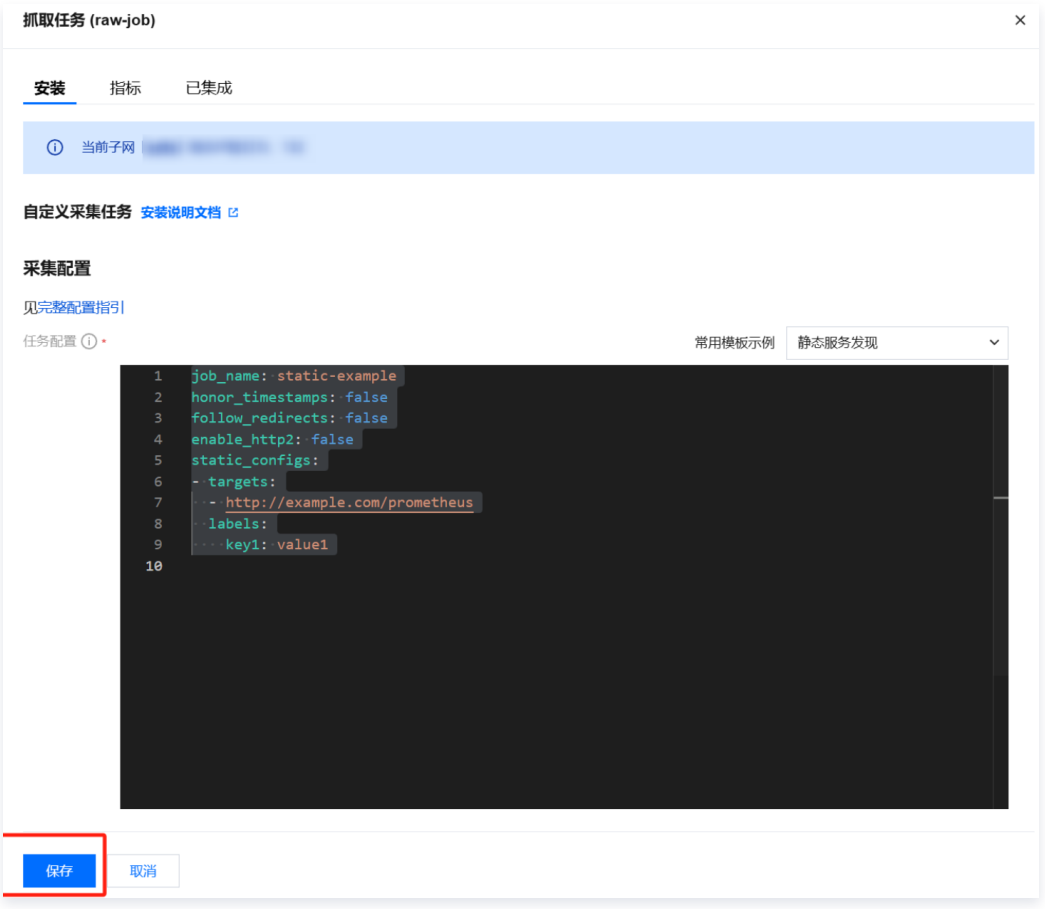
```
    - fluent-bit-demo
    # 填写要监控service的Label值，以定位目标service
  selector:
    matchLabels:
      app: fluent-bit-app-demo
```

> **注意：**
> 示例中名称为 application 的 Label 必须配置，否则无法使用我们提供一些其它的开箱即用的集成功能。更多高阶用法请参见 ServiceMonitor 或 PodMonitor 。

Rawjob 采集

Rawjob 采集需要配置 Fluentd 服务 pod ip，仅能在组件稳定为固定的 pod 时使用，动态扩缩容的情况下需要使用 Podmonitor 或者 Servicemonitor 来保证动态的服务发现。Rawjob 采集配置方式：

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击**抓取任务**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



查看监控

1. 在 Prometheus 实例 列表，找到对应的 Prometheus 实例，单击实例ID 右侧图标，打开您的专属 Grafana，输入您的账号密码，即可进行 Grafana 可视化大屏操作区。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到并单击 **Fluentd 监控**，选择 **Dashboard > Dashboard 操作 > Dashboard 安装/升级**，单击**安装/升级**安装对应的 Grafana Dashboard。

3. 进入 Grafana，单击  图表，展开监控面板，单击对应的监控图表名称即可查看监控数据。

MongoDB Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

在使用 MongoDB 过程中需要对 MongoDB 运行状态进行监控，以便了解 MongoDB 服务是否运行正常，排查 MongoDB 故障问题原因，Prometheus 监控服务提供了基于 Exporter 的方式来监控 MongoDB 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署 Exporter 以及实现 MongoDB Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 MongoDB 是腾讯云 云数据库 MongoDB 版，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **MongoDB**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。

	• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
用户名	MongoDB 的用户名称。
密码	MongoDB 的密码。
地址	MongoDB 的连接地址。
标签	给指标添加自定义 Label。
Exporter 配置	• database：启用数据库指标的收集。 • collection：启用集合指标的收集。 • topmetrics：启用数据库表头指标信息的收集。 • indexusage：启用索引使用统计信息的收集。 • connpoolstats：收集 MongoDB 连接池统计信息。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间。
- 在 **Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 MongoDB 连接串 > 部署 MongoDB Exporter > 验证 步骤完成 Exporter 部署。

使用 Secret 管理 MongoDB 连接串

1. 在左侧菜单中选择**工作负载 > Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 MongoDB Exporter 的时候直接使用 Secret Key，需要调整对应的 URI，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
    name: mongodb-secret-test
    namespace: mongodb-test
type: Opaque
stringData:
    datasource: "mongodb://{user}:{passwd}@{host1}:{port1},{host2}:{port2},{host3}:{port3}/admin"
# 对应连接URI
```

部署 MongoDB Exporter

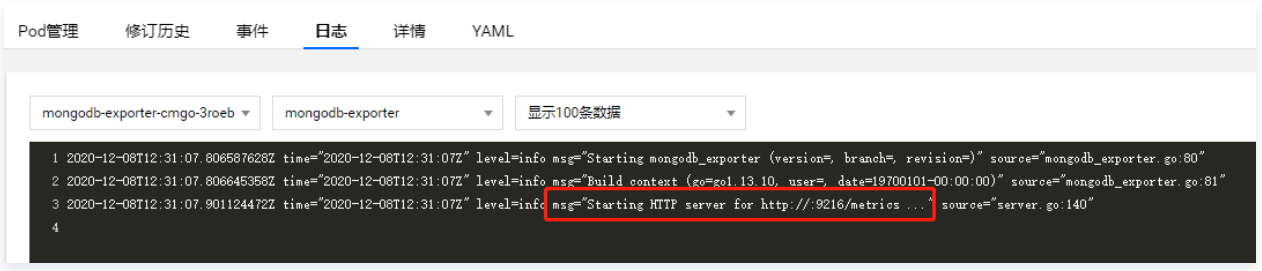
在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
  name: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
  namespace: mongodb-test
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
  template:
    metadata:
      labels:
        k8s-app: mongodb-exporter # 根据业务需要调整成对应的名称，建议加上 MongoDB 实例的信息
    spec:
      containers:
        - args:
            - --collect.database       # 启用采集 Database metrics
            - --collect.collection     # 启用采集 Collection metrics
            - --collect.topmetrics     # 启用采集 table top metrics
            - --collect.indexusage     # 启用采集 per index usage stats
            - --collect.connpoolstats  # 启动采集 MongoDB connpoolstats
          env:
            - name: MONGODB_URI
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret-test
                  key: datasource
          image: ccr.ccs.tencentyun.com/rig-agent/mongodb-exporter:0.10.0
          imagePullPolicy: IfNotPresent
          name: mongodb-exporter
          ports:
            - containerPort: 9216
              name: metric-port  # 这个名称在配置抓取任务的时候需要
          securityContext:
            privileged: false
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
        - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: { }
      terminationGracePeriodSeconds: 30
```

说明：
Exporter 详细参数请参见 mongodb_exporter 。

验证

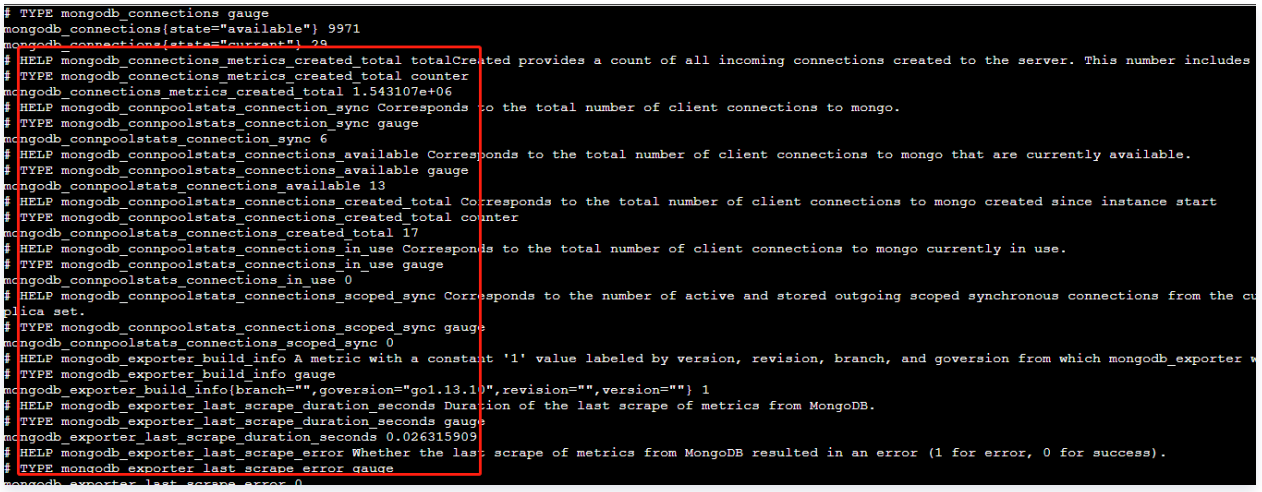
1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：



3. 单击 **Pod 管理**页签，进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，登录 Pod，在命令行中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 MongoDB 指标，若发现未能得到对应的数据，请检查一下连接 URI 是否正确，具体如下：

```
wget 127.0.0.1:9216/metrics
cat metrics
```

命令执行结果如下图所示：



步骤二：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 通过**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: mongodb-exporter # 填写一个唯一名称
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不
要修改
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
```

```
      regex: (.*)
      targetLabel: instance
      replacement: 'cmgo-xxxxxxxx' # 调整成对应的 MongoDB 实例 ID
  namespaceSelector:  # 选择要监控pod所在的namespace
    matchNames:
    - mongodb-test
  selector: # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: mongodb-exporter
```

说明：
由于 `Exporter` 和 `MongoDB` 部署在不同的服务器上，因此建议通过 Prometheus Relabel 机制将 MongoDB 实例的信息放到监控指标中，以便定位问题。

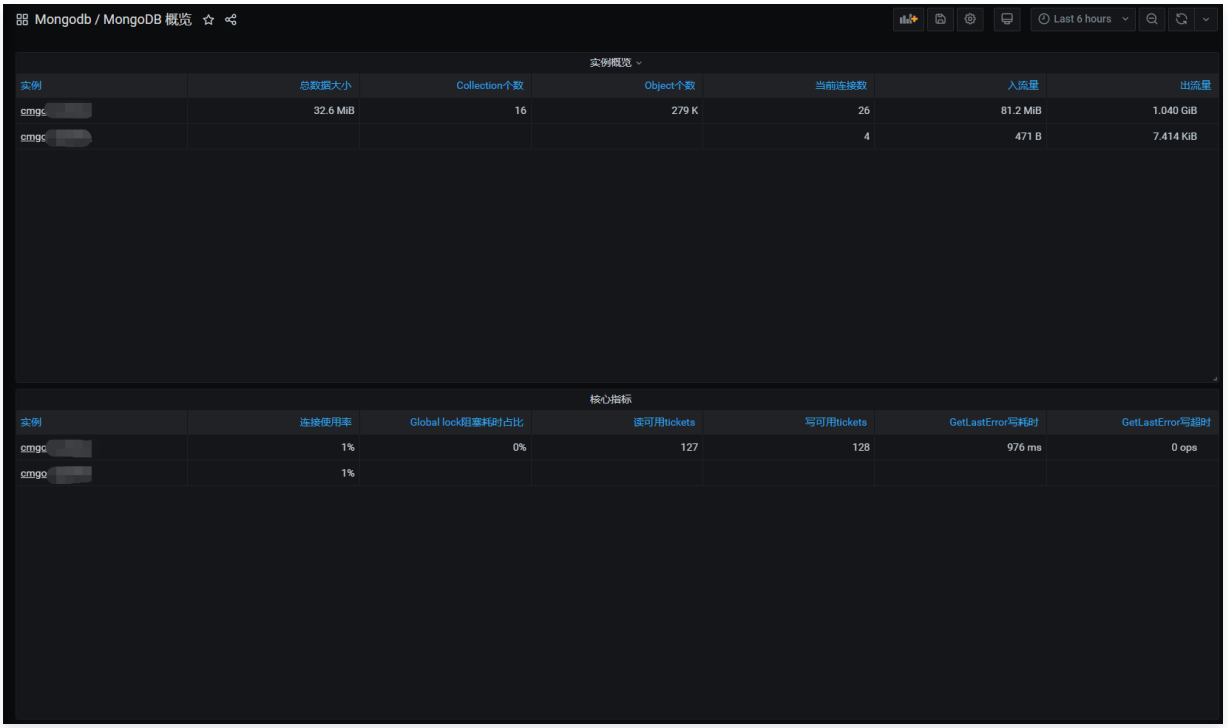
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，进入集成中心页面。找到 `MongoDB` 监控，安装对应的 Grafana Dashboard 即可开启 `MongoDB` 监控大盘，查看实例相关的监控数据，如下图所示：
   - **MongoDB 概览**：以实例的维度查看实例状态，例如文档个数、连接使用率、读写耗时等，可单击实例跳转到该实例详情。



- **MongoDB 详情**：可以查看某个实例的详细状态，例如元数据概览、核心指标、命令操作、请求流量、读写 Top 等。

> **说明：**
>
> 每个图表可以单击左侧的 进行查看说明。

配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击告警策略，可以添加相应的告警策略，详情请参见 新建告警策略 。

常见问题

客户端报错：client checkout connect timeout，该如何处理？

可能是连接池使用率达到100%，导致创建连接失败。可以通过 Grafana 大盘 **MongoDB 详情/核心指标/连接使用率**指标排查。

写入不断超时，该如何处理?

需检查 Cache 使用率是否过高、Transactions 可用个数是否为0，可以通过 Grafana 大盘 **MongoDB 详情/核心指标/ WiredTiger Transactions 可用个数| WiredTiger Cache 使用率| GetLastError 写耗时| GetLastError 写超时**指标排查。

PostgreSQL Exporter 接入

最近更新时间：2025-04-24 19:00:43

操作场景

在使用 PostgreSQL 过程中都需要对 PostgreSQL 运行状态进行监控，以便了解 PostgreSQL 服务是否运行正常，排查 PostgreSQL 故障问题原因， Prometheus 监控服务提供了基于 Exporter 的方式来监控 PostgreSQL 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文介绍如何部署Exporter 以及实现 PostgreSQL Exporter 告警接入等操作。

> **说明：**
> 如果需要监控的 PostgreSQL 是腾讯云 云数据库 PostgreSQL，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

方式一：一键安装（推荐）

操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在**集成中心**找到并单击 **PostgreSQL**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。

用户名	PostgreSQL 的用户名称。
密码	PostgreSQL 的密码。
地址	PostgreSQL 的连接地址。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台，选择对应的 Prometheus 实例，选择**数据采集** > **集成容器服务**，然后找到对应容器集群完成关联集群操作。可参见指引 关联集群。

操作步骤

步骤1：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中选择**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 PostgreSQL 密码 > 部署 PostgreSQL Exporter > 获取指标 步骤完成 Exporter 部署。

使用 Secret 管理 PostgreSQL 密码

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 PostgreSQL Exporter 的时候直接使用 Secret Key，需要调整对应的 `password`，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
    name: postgres-test
type: Opaque
stringData:
    username: postgres
    password: you-guess #对应 PostgreSQL 密码
```

部署 PostgreSQL Exporter

在 Deployment 管理页面，单击**新建**，选择对应的**命名空间**来进行部署服务。可以通过控制台的方式创建，如下以 YAML 的方式部署 Exporter，YAML 配置示例如下（**复制下面的内容，可根据实际业务调整相应的参数**）。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test # 根据业务需要调整成对应的名称，建议加上 PG 实例的信息
  namespace: postgres-test # 根据业务需要调整成对应的名称，建议加上 PG 实例的信息
  labels:
    app: postgres
```

```
    app.kubernetes.io/name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
      app.kubernetes.io/name: postgresql
  template:
    metadata:
      labels:
        app: postgres
        app.kubernetes.io/name: postgresql
    spec:
      containers:
      - name: postgres-exporter
        image: ccr.ccs.tencentyun.com/rig-agent/postgres-exporter:v0.8.0
        args:
          - "--web.listen-address=:9187" # export 开启的端口
          - "--log.level=debug" # 日志级别
        env:
          - name: DATA_SOURCE_USER
            valueFrom:
              secretKeyRef:
                name: postgres-test # 对应上一步中的 Secret 的名称
                key: username  # 对应上一步中的 Secret Key
          - name: DATA_SOURCE_PASS
            valueFrom:
              secretKeyRef:
                name: postgres-test # 对应上一步中的 Secret 的名称
                key: password  # 对应上一步中的 Secret Key
          - name: DATA_SOURCE_URI
            value: "x.x.x.x:5432/postgres?sslmode=disable"  # 对应的连接信息
        ports:
        - name: http-metrics
          containerPort: 9187
```

说明：
上述示例将 Secret 中的用户名密码传给了环境变量 `DATA_SOURCE_USER` 和 `DATA_SOURCE_PASS`，使用户无法查看到明文的用户名密码。您还可以用 `DATA_SOURCE_USER_FILE` / `DATA_SOURCE_PASS_FILE` 从文件读取用户名密码，或使用 `DATA_SOURCE_NAME` 将用户名密码也放在连接串里，例如 `postgresql://login:password@hostname:port/dbname`。

参数说明

`DATA_SOURCE_URI` / `DATA_SOURCE_NAME` 连接串 query 部分（`?` 之后）支持的参数如下（最新的以 GoDoc 为准）。

参数	参数说明
sslmode	是否使用 SSL，支持的值如下： • - disable：不使用 SSL。 • - require：总是使用（跳过验证）。 • - verify-ca：总是使用（检查服务端提供的证书是不是由一个可信的 CA 签发）。 • - verify-full：总是使用（检查服务端提供的证书是不是由一个可信的 CA 签发，并且检查 hostname 是不是被证书所匹配）。
fallback_application_n	一个备选的 application_name。

ame	
connect_timeout	最大连接等待时间，单位秒。0值等于无限大。
sslcert	证书文件路径。文件数据格式必须是 PEM。
sslkey	私钥文件路径。文件数据格式必须是 PEM。
sslrootcert	root 证书文件路径。文件数据格式必须是 PEM。

另外 Exporter 支持其他参数，如下说明（详情请参见 README ）。

参数	参数说明	环境变量
--web.listen-address	监听地址，默认9487。	PG_EXPORTER_WEB_LISTEN_ADDRESS
--web.telemetry-path	暴露指标的路径，默认 /metrics。	PG_EXPORTER_WEB_TELEMETRY_PATH
--extend.query-path	指定一个包含自定义查询语句的 YAML 文件，参见 queries.yaml 。	PG_EXPORTER_EXTEND_QUERY_PATH
--disable-default-metrics	只使用通过 queries.yaml 提供的指标。	PG_EXPORTER_DISABLE_DEFAULT_METRICS
--disable-settings-metrics	不抓取 pg_settings 相关的指标。	PG_EXPORTER_DISABLE_SETTINGS_METRICS
--auto-discover-databases	是否自动发现 Postgres 实例上的数据库。	PG_EXPORTER_AUTO_DISCOVER_DATABASES
--dumpmaps	打印内部的指标信息，除了 debug 不要使用，方便排查自定义 queries 相关的问题。	-
--constantLabels	自定义标签，通过 key=value 的形式提供，多个标签对使用 , 分隔。	PG_EXPORTER_CONSTANT_LABELS
--exclude-databases	需要排除的数据库，仅在 --auto-discover-databases 开启的情况下有效	PG_EXPORTER_EXCLUDE_DATABASES
--log.level	日志级别 debug/info/warn/error/fatal。	PG_EXPORTER_LOG_LEVEL

获取指标

通过 `curl http://exporter:9187/metrics` 无法获取 `Postgres` 实例运行时间。我们可以通过自定义一个 `queries.yaml` 来获取该指标：

1. 创建一个包含 `queries.yaml` 的 ConfigMap 。
2. 将 ConfigMap 作为 Volume 挂载到 Exporter 某个目录下面。
3. 通过 `--extend.query-path` 来使用 ConfigMap，将上述的 Secret 以及 Deployment 进行汇总，汇总后的 YAML 如下所示。

```
# 注意：以下 document 创建一个名为 postgres-test 的 Namespace，仅作参考
apiVersion: v1
kind: Namespace
metadata:
  name: postgres-test

# 以下 document 创建一个包含用户名密码的 Secret
---
apiVersion: v1
kind: Secret
metadata:
```

```
  name: postgres-test-secret
  namespace: postgres-test
type: Opaque
stringData:
  username: postgres
  password: you-guess

# 以下 document 创建一个包含自定义指标的 queries.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-test-configmap
  namespace: postgres-test
data:
  queries.yaml: |
    pg_postmaster:
      query: "SELECT pg_postmaster_start_time as start_time_seconds from
pg_postmaster_start_time()"
      master: true
      metrics:
        - start_time_seconds:
            usage: "GAUGE"
            description: "Time at which postmaster started"

# 以下 document 挂载了 Secret 和 ConfigMap ，定义了部署 Exporter 相关的镜像等参数
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test
  namespace: postgres-test
  labels:
    app: postgres
    app.kubernetes.io/name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
      app.kubernetes.io/name: postgresql
  template:
    metadata:
      labels:
        app: postgres
        app.kubernetes.io/name: postgresql
    spec:
      containers:
        - name: postgres-exporter
          image: wrouesnel/postgres_exporter:latest
          args:
            - "--web.listen-address=:9187"
            - "--extend.query-path=/etc/config/queries.yaml"
            - "--log.level=debug"
          env:
            - name: DATA_SOURCE_USER
              valueFrom:
                secretKeyRef:
```

```
                name: postgres-test-secret
                key: username
          - name: DATA_SOURCE_PASS
            valueFrom:
              secretKeyRef:
                name: postgres-test-secret
                key: password
          - name: DATA_SOURCE_URI
            value: "x.x.x.x:5432/postgres?sslmode=disable"
        ports:
          - name: http-metrics
            containerPort: 9187
        volumeMounts:
          - name: config-volume
            mountPath: /etc/config
    volumes:
      - name: config-volume
        configMap:
          name: postgres-test-configmap
```

4. 执行 `curl http://exporter:9187/metrics` 命令后，即可通过自定义的 `queries.yaml` 查询到 Postgres 实例启动时间指标。示例如下：

```
# HELP pg_postmaster_start_time_seconds Time at which postmaster started
# TYPE pg_postmaster_start_time_seconds gauge
pg_postmaster_start_time_seconds{server="x.x.x.x:5432"} 1.605061592e+09
```

步骤2：添加采集任务

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

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: postgres-exporter  # 填写一个唯一名称
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，
不要修改
  spec:
    namespaceSelector:
      matchNames:
      - postgres-test
    podMetricsEndpoints:
    - interval: 30s
      path: /metrics
      port: http-metrics # 前面 Exporter 那个 Container 的端口名
      relabelings:
      - action: labeldrop
        regex: __meta_kubernetes_pod_label_(pod_|statefulset_|deployment_|controller_)(.+)
      - action: replace
        regex: (.*)
        replacement: postgres-xxxxxx
        sourceLabels:
```

```
      - instance
        targetLabel: instance
    selector:
      matchLabels:
        app: postgres
```

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

查看监控

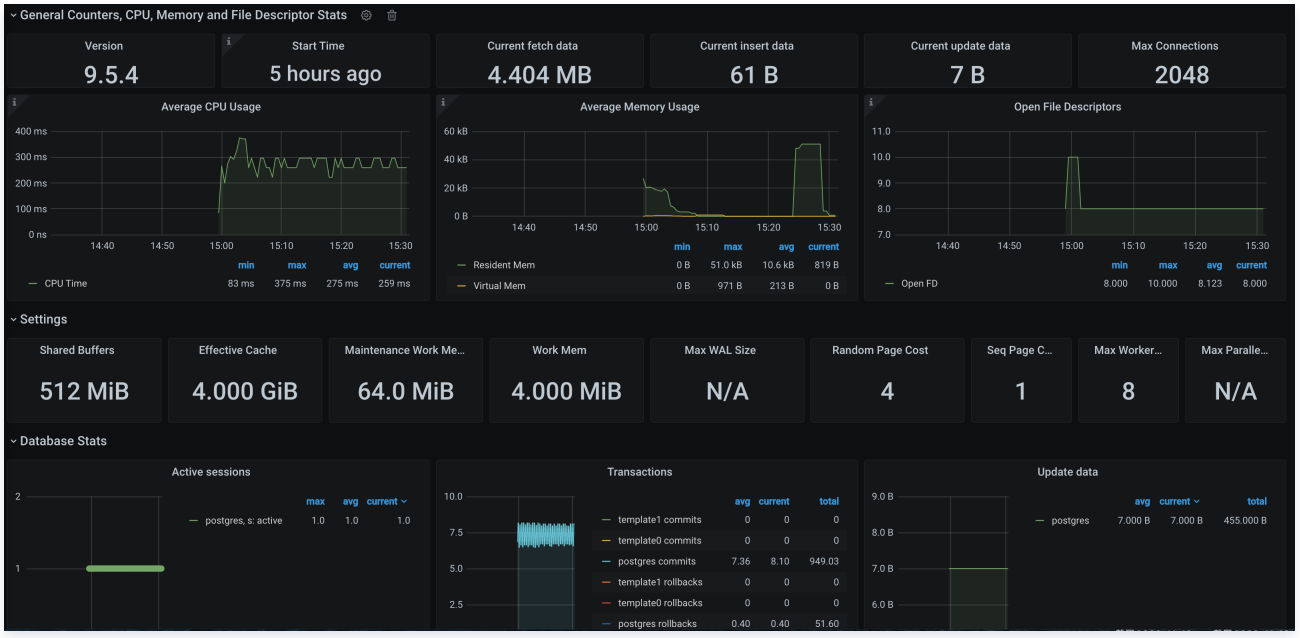
> **说明：**
> 需要使用上述 获取指标 配置来获取 Postgres 实例的启动时间。

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到 PostgreSQL 监控，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 单击 实例 ID 右侧图标，打开您的专属 Grafana，输入您的账号密码，即可进行 Grafana 可视化大屏操作区。
4. 进入 Grafana，单击图标，展开监控面板，单击对应的监控图标名称即可查看监控数据。



配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 在**告警管理 > 告警策略**页面，可以添加相应的告警策略，详情请参见 新建告警策略 。

> **说明：**

后续 Prometheus 监控服务将提供更多 PostgreSQL 相关的告警模板。

Nginx Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

Nginx 通过 stub_status 页面暴露了部分监控指标。Nginx Prometheus Exporter 会采集单个 Nginx 实例指标，并将其转化为 Prometheus 可用的监控数据， 最终通过 HTTP 协议暴露给 Prometheus 服务进行采集。我们可以通过 Exporter 上报重点关注的监控指标，用于异常报警和大盘展示。

前提条件

开启 NGINX stub_status 功能

> **说明：**
> 1. 下述例子为 Nginx 部署在容器服务中，其他部署方式登录及配置修改方式进行对应调整即可。
> 2. 容器服务相关操作可参见 容器服务 相关文档。

因为 Nginx Prometheus Exporter 是通过 Nginx 的 stub_status 模块对其进行监控，所以需要确保 Nginx 服务打开了 stub_status 模块，具体步骤如下：

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**，找到业务 Nginx 服务所在集群，进入集群，找到业务 Nginx 服务。
3. 登录到业务 Nginx 服务，执行以下命令检查 Nginx 是否已经开启了该模块：

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

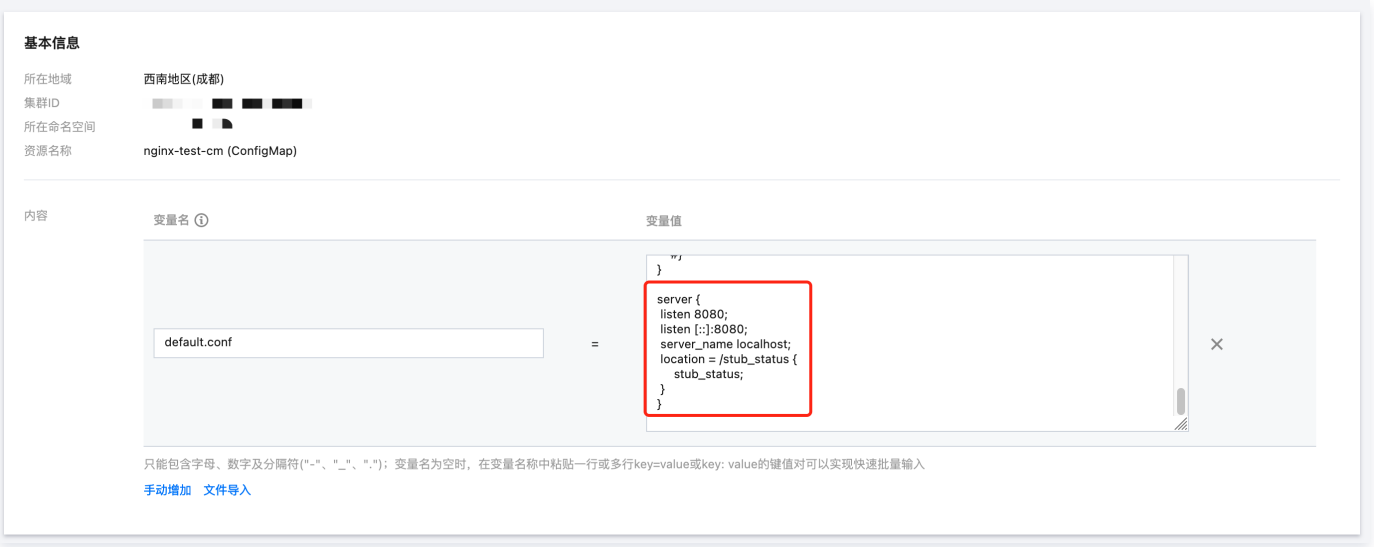
   - 如果在终端中输出 `with-http_stub_status_module` ，则说明 Nginx 已启用 stub_status 模块。
   - 如果未输出任何结果，则可以使用 `--with-http_stub_status_module` 参数从源码重新配置编译一个 Nginx。示例如下：

```
./configure \
… \      ## 之前编译nginx所需的命令
--with-http_stub_status_module
make
sudo make install
```

4. 若未添加 Nginx 服务相关 ConfigMap，可登录到业务 Nginx 服务，将配置目录（官方镜像为 `/etc/nginx/conf.d` ）下的 default.conf 配置信息进行拷贝，创建 ConfigMap，将配置信息添加到该 ConfigMap 中，ConfigMap 相关操作指引见 ConfigMap 管理 。
5. 确认 stub_status 模块启用之后，在 ConfigMap 的default.conf 中添加如下配置。示例如下：

```
server {
 listen 8080;  # 根据业务情况进行调整
 listen [::]:8080; # 根据业务情况进行调整
 server_name localhost; # 根据业务情况进行调整
 location = /stub_status { # 具体路径可根据业务情况进行调整
     stub_status;
 }
}
```

ConfigMap 中配置示例如下：



6. 配置修改完成之后，找到业务 Nginx 服务，点击**更多 > 重新部署**，完成配置的重新加载。非容器服务环境重加载命令：

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

7. 完成上述步骤之后，登录业务 Nginx 服务，执行如下命令，即可查看 Nginx 上次启动后工作状态的统计结果。

```
curl http://localhost:8080/stub_status  ## 根据配置文件中配置内容进行相应调整
```

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

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **Nginx**，找到后单击它即会弹出一个安装窗口。
5. 在弹出窗口的安装页面，填写指标采集名称、地址、路径等信息，并单击**保存**。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
地址	Nginx 服务的连接地址。
路径	Nginx 服务的服务状态路径，在配置中指定
用户名	Nginx 服务 HTTP 验证用的用户名称。
密码	Nginx 服务 HTTP 验证用的密码。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 **Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 Nginx Exporter > 验证 步骤完成 Exporter 部署。

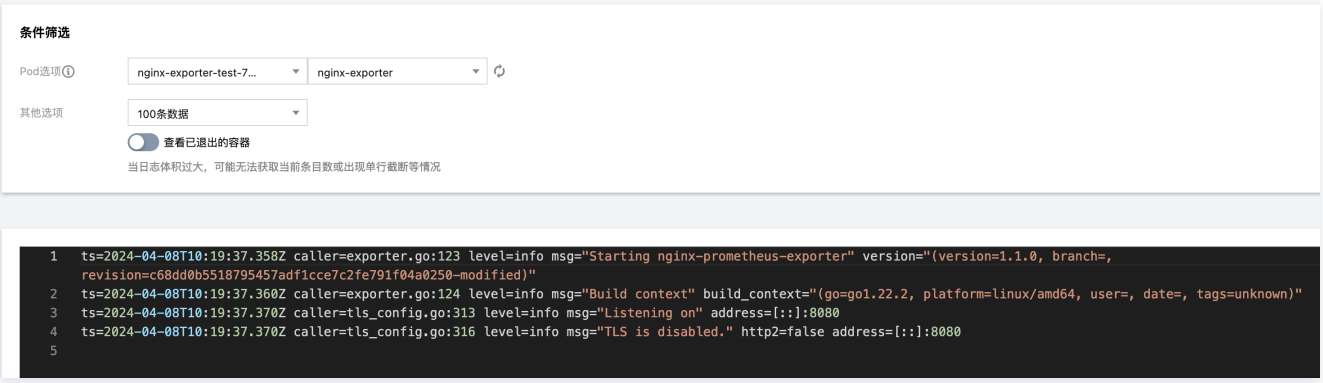
步骤二：部署 Nginx Exporter

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
  name: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
  namespace: nginx-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
  template:
    metadata:
      labels:
        k8s-app: nginx-exporter  # 根据业务需要调整成对应的名称，建议加上 Nginx 实例的信息
    spec:
      containers:
      - args:
        - --web.listen-address=:8080
        - --nginx.scrape-uri=http://127.0.0.1:8080/stub_status  # 根据业务需要调整成 Nginx 实例对应地
址
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:nginx-exporter-v1.1.0
        name: nginx-exporter
        ports:
        - containerPort: 9113
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

步骤三：验证

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



3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 Nginx 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:8080/metrics
```

执行结果如下图所示：

```
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 2
# HELP nginx_connections_handled Handled client connections
# TYPE nginx_connections_handled counter
nginx_connections_handled 128
# HELP nginx_connections_reading Connections where NGINX is reading the request header
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
# HELP nginx_connections_waiting Idle client connections
# TYPE nginx_connections_waiting gauge
nginx_connections_waiting 1
# HELP nginx_connections_writing Connections where NGINX is writing the response back to the client
# TYPE nginx_connections_writing gauge
nginx_connections_writing 1
# HELP nginx_exporter_build_info A metric with a constant '1' value labeled by version, revision, branch, goversion from which nginx_exporter was built, and the goos and goarch for the build.
# TYPE nginx_exporter_build_info gauge
nginx_exporter_build_info{branch="",goarch="amd64",goos="linux",goversion="go1.22.2",revision="c68dd0b5518795457adf1cce7c2fe791f04a0250-modified",tags="unknown",version="1.1.0"} 1
# HELP nginx_http_requests_total Total http requests
# TYPE nginx_http_requests_total counter
nginx_http_requests_total 29564
# HELP nginx_up Status of the last metric scrape
# TYPE nginx_up gauge
nginx_up 1
```

步骤四：添加采集任务

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: nginx-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
```

```
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Nginx 实例信息
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - nginx-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: nginx-exporter
```

查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 middleware 文件夹中找到 nginx 实例监控面板，查看实例相关监控数据，如下图所示：

配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：Nginx Exporter 采集参数说明

全局配置参数

名称	描述
web.telemetry-path	指标暴露路径，默认 `/metrics` 。
nginx.scrape-uri	nginx 指标抓取 url，默认 `http://127.0.0.1:8080/stub_status` 。

[no-]nginx.plus	是否启用 NGINX Plus，默认是启用的。
[no-]nginx.ssl-verify	是否验证 ssl 证书。
nginx.ssl-ca-cert	ssl 证书路径。
nginx.ssl-client-cert	ssl 证书路径。
nginx.ssl-client-key	ssl 证书路径。
nginx.timeout	nginx 指标抓取超时。
prometheus.const-label	将在每个指标中使用的标签。格式为 label=value，允许重复多次。
[no-]web.systemd-socket	使用 systemd 套接字监听器代替端口监听器（仅限 Linux）。
web.listen-address	监听地址，默认：9113。
web.config.file	配置文件的路径，可以启用 TLS 或身份验证（实验性参数）。
log.level	日志级别，默认 info。
log.format	日志消息的输出格式，取值范围：[logfmt，json]，默认 logfmt。
version	打印版本信息。

Redis Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

在使用数据库 Redis 过程中需要对 Redis 运行状态进行监控，以便了解 Redis 服务是否运行正常，排查 Redis 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Redis 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控 Redis。

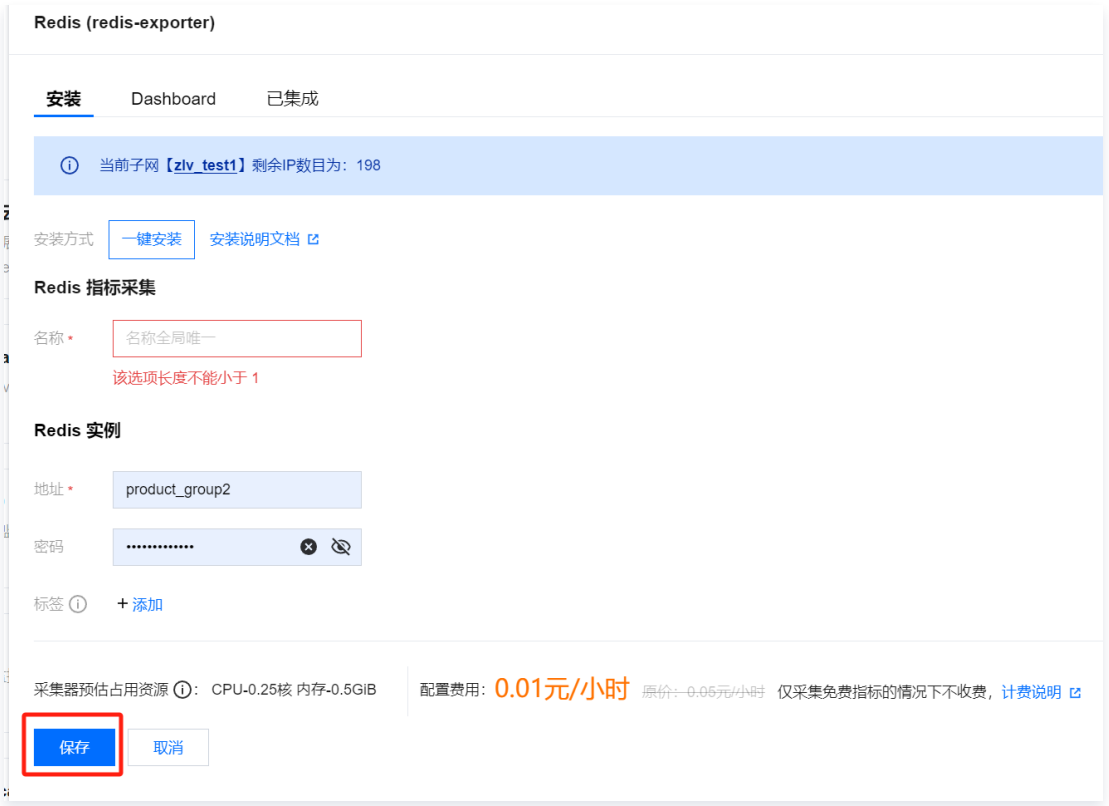
> **说明：**
> 如果需要监控的 Redis 是腾讯云 云数据库 Redis，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Redis**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。

地址	Redis 的连接地址。
密码	Redis 的密码。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 进行统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络 VPC 下，创建 腾讯云容器服务，并为集群创建 命名空间 。
- 在 **Prometheus 监控服务控制台** > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 Redis 密码 > 部署 Redis Exporter > 验证 步骤完成 Exporter 部署。

使用 Secret 管理 Redis 密码

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理密码并对密码进行加密处理，在启动 Redis Exporter 的时候直接使用 Secret Key，需要调整对应的 `password`，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
    name: redis-secret-test
    namespace: redis-test
type: Opaque
stringData:
    password: you-guess  #对应 Redis 密码
```

部署 Redis Exporter

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

> **说明：**
> 更多 Exporter 详细参数介绍请参见 redis_exporter 。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
```

```
    k8s-app: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-redis-
exporter
  name: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-redis-
exporter
  namespace: redis-test # 选择一个适合的 namespace 来部署 exporter，如果没有需要新建一个 namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-
redis-exporter
  template:
    metadata:
      labels:
        k8s-app: redis-exporter # 根据业务需要调整成对应的名称，建议加上 Redis 实例的信息，如crs-66e112fp-
redis-exporter
    spec:
      containers:
      - env:
        - name: REDIS_ADDR
          value: ip:port # 对应 Redis 的 ip:port
        - name: REDIS_PASSWORD
          valueFrom:
            secretKeyRef:
              name: redis-secret-test
              key: password
        image: ccr.ccs.tencentyun.com/rig-agent/redis-exporter:v1.32.0
        imagePullPolicy: IfNotPresent
        name: redis-exporter
        ports:
        - containerPort: 9121
          name: metric-port  # 这个名称在配置抓取任务的时候需要
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

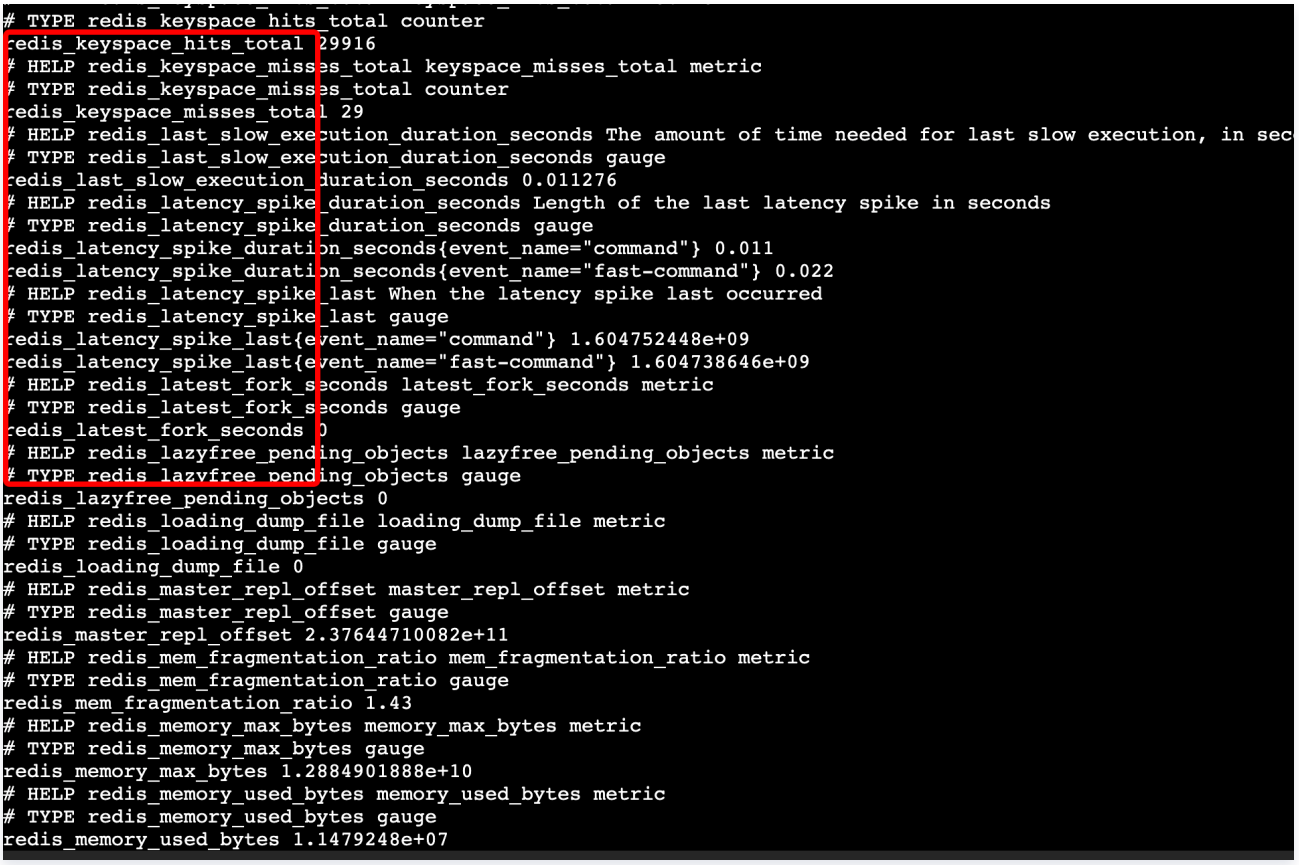
验证

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

3. 单击 **Pod 管理**页签，进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录** Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 Redis 指标。如发现未能得到对应的数据，请检查一下 `REDIS_ADDR` 和 `REDIS_PASSWORD` 是否正确。示例如下：

```
curl http://localhost:9121/metrics
```

命令执行结果如下图所示：



步骤二：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port  # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
```

```
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Redis 实例 ID
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # 调整成对应的 Redis 实例 IP
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - redis-test
  selector:     # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: redis-exporter
```

说明：
由于 `Exporter` 和 `Redis` 部署在不同的服务器上，因此建议通过 Prometheus Relabel 机制将 Redis 实例的信息放到监控指标中，以方便定位问题。

查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

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

配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**告警管理 > 告警策略**，可以添加相应的告警策略，详情请参见 新建告警策略 。

Aerospike Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

Aerospike Exporter 是一个用于 Aerospike 数据库的 Prometheus 指标导出工具，允许用户监视和收集 Aerospike 数据库的性能指标和统计信息。它可以帮助用户实时监控 Aerospike 集群的健康状况、性能表现和负载情况，有助于进行故障排除、性能优化和规划容量。通过将这些指标导出到 Prometheus，用户可以利用 Prometheus 的强大功能进行数据可视化、报警和分析。腾讯云可观测平台 Prometheus 提供了 Aerospike Exporter 集成及开箱即用的 Grafana 监控大盘。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Aerospike**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
域名	Aerospike 数据库域名。

地址	Aerospike 数据库端口。
用户名	Aerospike 数据库用户名称。
密码	Aerospike 数据库密码。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 **Prometheus 监控服务控制台** > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 Exporter 配置 > 部署 Aerospike Exporter > 验证 步骤完成 Exporter 部署。

步骤二：部署 Exporter 配置

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: aerospike-secret-test   # 根据业务需要调整成相应名称
  namespace: aerospike-demo    # 根据业务需要调整到相应命名空间
type: Opaque
stringData:
  ape.toml: |-
      [Agent]
          # metrics server timeout in seconds
          timeout = 30

          # support system statistics also
          refresh_system_stats = true

          # prometheus binding port
          bind = ":8080"  # 暴露指标端口

      [Aerospike]
          db_host = "127.0.0.1"    # 根据业务需要调整成对应的 IP 或域名
          db_port = 3000    # 根据业务需要调整成对应的端口
          user = "admin"  # 根据业务需要调整成对应的用户名
          password = "admin"  # 根据业务需要调整成对应的密码
```

```
        # timeout for sending commands to the server node in seconds
        timeout = 30
  gauge_stats_list.toml: |-
      # This file represents a list of metrics which are treated as Gauges while exporting to
Prometheus or some other Observability tool.
      # to know more about these stats, please visit https://docs.aerospike.com

      #
      # SETS: below section define all Sets stats which are treated as Gauges
      #
      sets_gauge_stats = [

        "device_data_bytes",
        "index_populating",
        "memory_data_bytes",
        "objects",
        "sindexes",
        "tombstones",
        "truncate_lut",

      # 7.0 changes
        "data_used_bytes",
        "truncating",
      ]

      #
      # XDR: below section define all XDR stats which are treated as Gauges
      #
      xdr_gauge_stats = [
        "compression_ratio",
        "in_progress",
        "in_queue",
        "lag",
        "lap_us",
        "latency_ms",
        "nodes",
        "recoveries_pending",
        "throughput",
        "uncompressed_pct",
      ]

      #
      # Sindex: below section define all Sindex stats which are treated as Gauges
      #
      sindex_gauge_stats = [
        "entries_per_bval",
        "entries_per_rec",
        "entries",
        "histogram",                    # removed in server6.0
        "ibtr_memory_used",             # removed in server6.0
        "keys", # removed in server6.0
        "load_pct",
        "load_time",
        "loadtime",                     # removed in server6.0
        "memory_used",                  # deprecated in server6.3 version and replaced by used_bytes
        "nbtr_memory_used",             # removed in server6.0
        "query_basic_avg_rec_count",    # removed in server6.0
```

```
        "used_bytes",                    # added in server6.3 represents memory used by data (aka
memory_used)
      ]

      #
      # Node: below section define all Node stats which are treated as Gauges
      #

      node_gauge_stats = [
        "batch_index_proto_compression_ratio",
        "batch_index_proto_uncompressed_pct",
        "batch_index_queue",
        "batch_index_unused_buffers",
        "client_connections",
        "cluster_clock_skew_ms",
        "cluster_clock_skew_stop_writes_sec",
        "cluster_integrity",
        "cluster_is_member",
        "cluster_max_compatibility_id",
        "cluster_min_compatibility_id",
        "cluster_size",
        "fabric_bulk_recv_rate",
        "fabric_bulk_send_rate",
        "fabric_connections",
        "fabric_ctrl_recv_rate",
        "fabric_ctrl_send_rate",
        "fabric_meta_recv_rate",
        "fabric_meta_send_rate",
        "fabric_rw_recv_rate",
        "fabric_rw_send_rate",
        "failed_best_practices",
        "heap_active_kbytes",
        "heap_allocated_kbytes",
        "heap_efficiency_pct",
        "heap_mapped_kbytes",
        "heap_site_count",
        "heartbeat_connections",
        "info_queue",
        "migrate_partitions_remaining",
        "objects",
        "process_cpu_pct",
        "proxy_in_progress",
        "queries_active",
        "rw_in_progress",
        "scans_active",
        "sindex_gc_list_creation_time",
        "sindex_gc_list_deletion_time",
        "system_free_mem_pct",
        "system_kernel_cpu_pct",
        "system_total_cpu_pct",
        "system_user_cpu_pct",
        "threads_detached",
        "threads_joinable",
        "threads_pool_active",
        "threads_pool_total",
        "time_since_rebalance",
        "tombstones",
        "tree_gc_queue",
```

```
    "tsvc_queue",
    #
    # 4.x XDR stats
    "dlog_free_pct",
    "dlog_used_objects",
    "xdr_active_failed_node_sessions",
    "xdr_active_link_down_sessions",
    "xdr_global_lastshiptime",
    "xdr_read_active_avg_pct",
    "xdr_read_idle_avg_pct",
    "xdr_read_latency_avg",
    "xdr_read_reqq_used_pct",
    "xdr_read_reqq_used",
    "xdr_read_respq_used",
    "xdr_read_txnq_used_pct",
    "xdr_read_txnq_used",
    "xdr_ship_compression_avg_pct",
    "xdr_ship_inflight_objects",
    "xdr_ship_latency_avg",
    "xdr_ship_outstanding_objects",
    "xdr_throughput",
    "xdr_timelag",
  ]

  #
  # Namespace: below section define all Namespace stats which are treated as Gauges
  #
  namespace_gauge_stats =[
    "appeals_rx_active",
    "appeals_tx_active",
    "appeals_tx_remaining",
    "available_bin_names",
    "cache_read_pct",
    "clock_skew_stop_writes",
    "dead_partitions",
    "defrag_q",
    "device_available_pct",
    "device_compression_ratio",
    "device_free_pct",
    "device_total_bytes",
    "device_used_bytes",
    "effective_is_quiesced",
    "effective_prefer_uniform_balance",
    "effective_replication_factor",
    "evict_ttl",
    "hwm_breached",
    "index_flash_alloc_bytes",
    "index_flash_alloc_pct",
    "index_flash_used_bytes",
    "index_flash_used_pct",
    "index_pmem_used_bytes",
    "index_pmem_used_pct",
    "master_objects",
    "master_tombstones",
    "memory_free_pct",
    "memory_used_bytes",
    "memory_used_data_bytes",
    "memory_used_index_bytes",
```

```
        "memory_used_set_index_bytes",
        "memory_used_sindex_bytes",
        "migrate_rx_instances",
        "migrate_rx_partitions_active",
        "migrate_rx_partitions_initial",
        "migrate_rx_partitions_remaining",
        "migrate_signals_active",
        "migrate_signals_remaining",
        "migrate_tx_instances",
        "migrate_tx_partitions_active",
        "migrate_tx_partitions_imbalance",
        "migrate_tx_partitions_initial",
        "migrate_tx_partitions_lead_remaining",
        "migrate_tx_partitions_remaining",
        "n_nodes_quiesced",
        "non_expirable_objects",
        "non_replica_objects",
        "non_replica_tombstones",
        "ns_cluster_size",
        "nsup_cycle_deleted_pct",
        "nsup_cycle_duration",
        "nsup_cycle_sleep_pct",
        "objects",
        "pending_quiesce",
        "pmem_available_pct",
        "pmem_compression_ratio",
        "pmem_free_pct",
        "pmem_total_bytes",
        "pmem_used_bytes",
        "prole_objects",
        "prole_tombstones",
        "query_aggr_avg_rec_count",
        "query_basic_avg_rec_count",
        "query_proto_compression_ratio",
        "query_proto_uncompressed_pct",
        "record_proto_compression_ratio",
        "record_proto_uncompressed_pct",
        "scan_proto_compression_ratio",
        "scan_proto_uncompressed_pct",
        "shadow_write_q",
        "stop_writes",
        "storage-engine.device.defrag_q",
        "storage-engine.device.free_wblocks",
        "storage-engine.device.shadow_write_q",
        "storage-engine.device.used_bytes",
        "storage-engine.device.write_q",
        "storage-engine.device.age",
        "storage-engine.file.defrag_q",
        "storage-engine.file.free_wblocks",
        "storage-engine.file.shadow_write_q",
        "storage-engine.file.used_bytes",
        "storage-engine.file.write_q",
        "storage-engine.file.age",
        "storage-engine.stripe.defrag_q",
        "storage-engine.stripe.free_wblocks",
        "storage-engine.stripe.shadow_write_q",
        "storage-engine.stripe.used_bytes",
        "storage-engine.stripe.write_q",
```

```
        "storage-engine.stripe.age",
        "storage-engine.stripe.backing_write_q",
        "migrate_fresh_partitions",
        "tombstones",
        "truncate_lut",
        "unavailable_partitions",
        "unreplicated_records",
        "write_q",
        "xdr_bin_cemeteries",
        "xdr_tombstones",

      # added in 7.0
        "data_avail_pct",
        "data_compression_ratio",
        "data_total_bytes",
        "data_used_bytes",
        "data_used_pct",
        "index_mounts_used_pct",
        "index_used_bytes",
        "indexes_memory_used_pct",
        "set_index_used_bytes",
        "sindex_mounts_used_pct",
        "sindex_used_bytes",
        "truncating",

      ]

      # System Info Gauge metrics list
      #
      system_info_gauge_stats = [
        "",
      ]
```

步骤三：部署 Aerospike Exporter

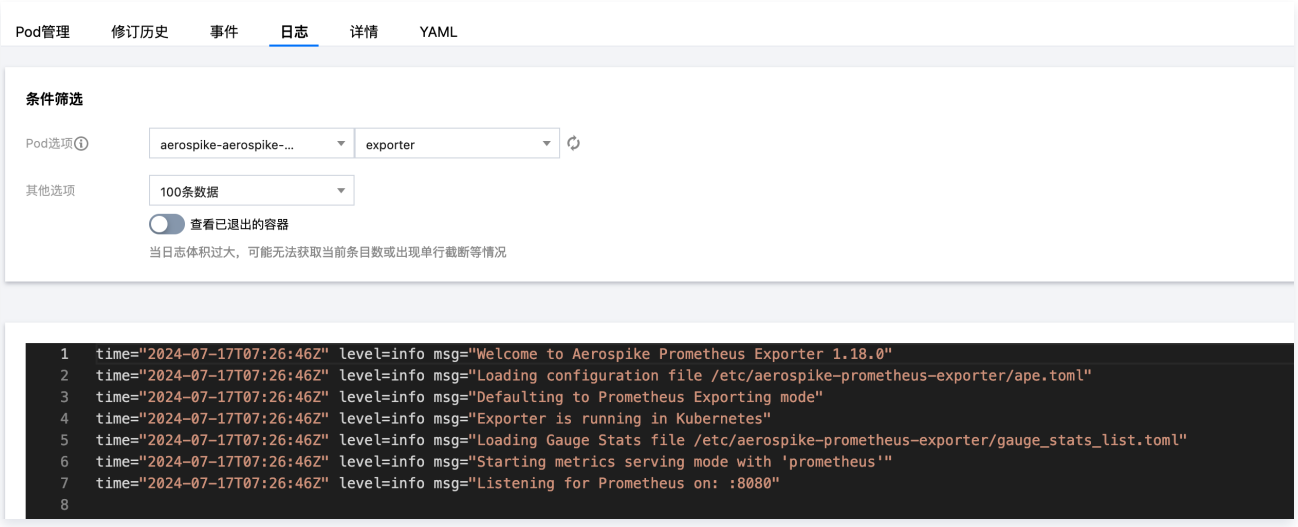
1. 在左侧菜单中选择**工作负载 > Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
  name: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
  namespace: aerospike-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
  template:
    metadata:
      labels:
        k8s-app: aerospike-exporter  # 根据业务需要调整成对应的名称，建议加上 Aerospike 实例的信息
    spec:
```

```
      volumes:
      - name: sec
        secret:
          defaultMode: 420
          secretName: aerospike-secret-test # 对应 步骤二 配置名称
      containers:
      - name: aerospike-exporter
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:aerospike-exporter-1.18.0
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 8080  # 对应 步骤二 配置中的指标导出端口
            name: metrics
        livenessProbe:
          tcpSocket:
            port: metrics
        readinessProbe:
          tcpSocket:
            port: metrics
        volumeMounts:
          - mountPath: /etc/aerospike-prometheus-exporter
            name: sec
            readOnly: true
```

步骤四：验证

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，无报错信息输出即可，如下图所示：



3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 Aerospike 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:8080/metrics
```

执行结果如下图所示：

```
/ # wget -qO- http://localhost:8080/metrics
# HELP aerospike_namespace_allow_ttl_without_nsup allow ttl without nsup
# TYPE aerospike_namespace_allow_ttl_without_nsup gauge
aerospike_namespace_allow_ttl_without_nsup{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_appeals_records_exonerated appeals records exonerated
# TYPE aerospike_namespace_appeals_records_exonerated counter
aerospike_namespace_appeals_records_exonerated{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_appeals_rx_active appeals rx active
# TYPE aerospike_namespace_appeals_rx_active gauge
aerospike_namespace_appeals_rx_active{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_appeals_tx_active appeals tx active
# TYPE aerospike_namespace_appeals_tx_active gauge
aerospike_namespace_appeals_tx_active{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_appeals_tx_remaining appeals tx remaining
# TYPE aerospike_namespace_appeals_tx_remaining gauge
aerospike_namespace_appeals_tx_remaining{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_auto_revive auto revive
# TYPE aerospike_namespace_auto_revive gauge
aerospike_namespace_auto_revive{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_auto_revived_partitions auto revived partitions
# TYPE aerospike_namespace_auto_revived_partitions counter
aerospike_namespace_auto_revived_partitions{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 0
# HELP aerospike_namespace_background_query_max_rps background query max rps
# TYPE aerospike_namespace_background_query_max_rps gauge
aerospike_namespace_background_query_max_rps{cluster_name="docker",ns="test",service="aerospike-aerospike-exporter-df4db9758-94d7c",storage_engine="device"} 10000
```

步骤五：添加采集任务

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: aerospike-exporter  # 填写一个唯一名称
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例(已停止售卖)：namespace 固定，不
要修改
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 Aerospike 实例 ID
    namespaceSelector:    # 选择要监控 aerospike exporter pod 所在的 namespace
      matchNames:
      - aerospike-demo
    selector:  # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: aerospike-exporter
```

查看监控

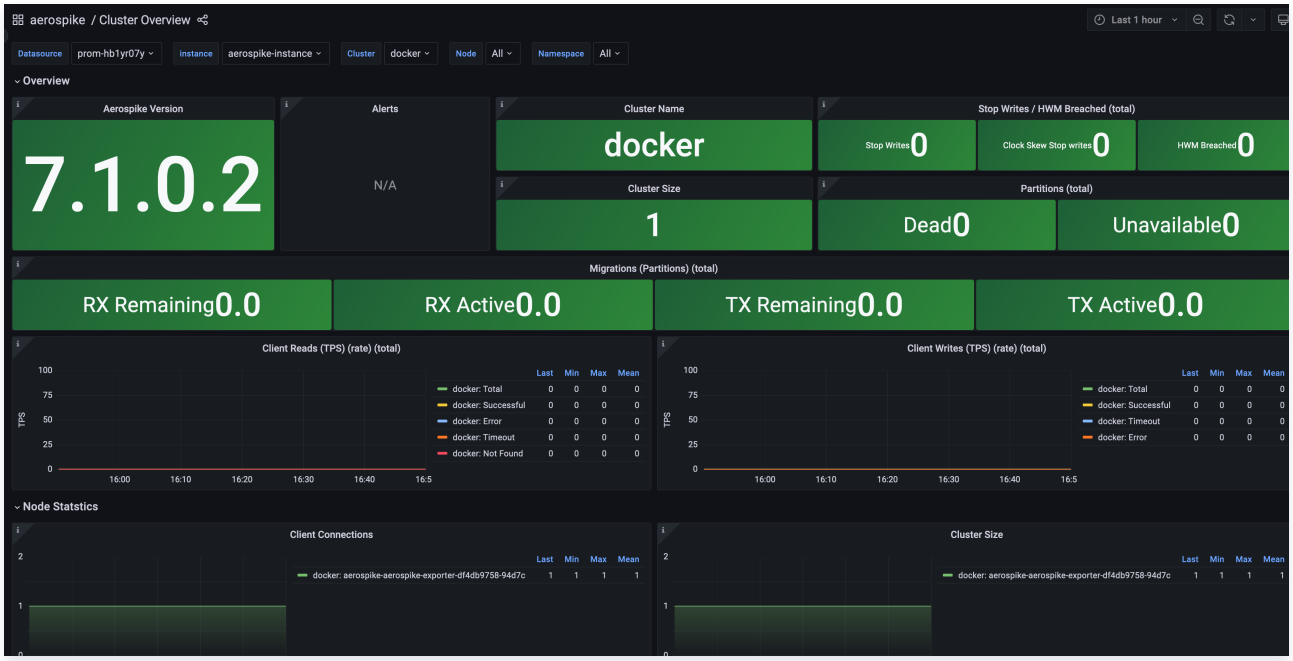
前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。

2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 aerospike 文件夹中找到 Aerospike 实例相关监控面板，查看实例相关监控数据，如下图所示：



配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：Aerospike Exporter 配置文件主要配置项

Agent 配置项

名称	描述
bind	指标导出端口，默认":9145"
cert_file	签名用证书文件
key_file	签名用证书文件
root_ca	签名用证书文件
basic_auth_username	http auth 验证用户名
basic_auth_password	http auth 验证密码
timeout	指标拉取超时
labels	自定义标签值
refresh_system_stats	支持系统数据统计

Aerospike 配置项

名称	描述
db_host	Aerospike 数据库域名或 IP
db_port	Aerospike 数据库服务端口
auth_mode	Aerospike 校验模式，默认 internal，取值有 "external"，"internal"，"pki"，""

user	Aerospike 数据库用户名
password	Aerospike 数据库密码
timeout	Aerospike 数据库连接超时

MySQL Exporter 接入

最近更新时间：2025-02-19 19:47:12

操作场景

MySQL Exporter 是社区专门为采集 MySQL/MariaDB 数据库监控指标而设计开发，通过 Exporter 上报核心的数据库指标，用于异常报警和监控大盘展示，腾讯云可观测平台 Prometheus 提供了与 MySQL Exporter 集成及开箱即用的 Grafana 监控大盘。
目前，Exporter 支持高于5.6版本的 MySQL 和高于10.1版本的 MariaDB。在 MySQL/MariaDB 低于5.6版本时，部分监控指标可能无法被采集。

> **说明：**
> 如果需要监控的 MySQL 是腾讯云 云数据库 MySQL，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

方式一：一键安装（推荐）

操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **MySQL**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明

名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
用户名	MySQL 的用户名称。
密码	MySQL 的密码。
地址	MySQL 的连接地址。
标签	给指标添加自定义 Label。
Exporter 配置	参数描述可能存在偏差，具体可参考 官方文档 。 • auto_increment.columns：从 information_schema 采集 auto_increment 列和最大值。 • binlog_size：采集所有已注册的 binlog 文件的当前大小。 • engine_innodb_status：从 SHOW ENGINE INNODB STATUS 采集。 • engine_tokudb_status：从 SHOW ENGINE TOKUDB STATUS 采集。 • global_status：从 SHOW GLOBAL STATUS 采集，默认为 true。 • global_variables：从 SHOW GLOBAL VARIABLES 采集，默认为 true。 • info_schema.clientstats：如果使用 userstat=1 运行，则设置为 true 以采集客户端统计信息。 • info_schema.innodb_metrics：从 information_schema.innodb_metrics 采集指标。 • info_schema.innodb_tablespaces：从 information_schema.innodb_sys_tablespaces 采集指标。 • info_schema.innodb_cmp：从 information_schema.innodb_cmp 采集 InnoDB 压缩表指标，默认为 true。 • info_schema.innodb_cmpmem：从 information_schema.innodb_cmpmem 采集 InnoDB 缓冲池压缩指标(默认为 true)。 • info_schema.processlist：从 information_schema.processlist 采集线程状态计数。 • info_schema.query_response_time：如果 query_response_time_stats 为 ON，则采集查询响应时间分布(默认为 true)。 • info_schema.tables：从 information_schema.tables 采集指标。 • info_schema.tables.databases：用于收集表统计信息的数据库列表，逗号分隔，默认是 '*' 表示所有数据库。 • info_schema.tablestats：如果使用 userstat=1 运行，则设置为 true 以采集表统计信息。 • info_schema.schemastats：如果使用 userstat=1 运行，则设置为 true 以采集架构统计信息。 • info_schema.userstats：如果使用 userstat=1 运行，则设置为 true 以采集用户统计信息。 • perf_schema.eventsstatements：从 performance_schema.events_statements_summary_by_digest 采集指标。 • perf_schema.eventsstatements.digest_text_limit：标准化语句文本的最大长度，默认值为120。 • perf_schema.eventsstatements.limit：按响应时间限制事件语句摘要的数量，默认值为250。 • perf_schema.eventsstatements.timelimit：限制 'last_seen' 事件语句的最大时间，单位为秒，默认值为86400。 • perf_schema.eventsstatementssum：从 performance_schema.events_statements_summary_by_digest 汇总采集指标。 • perf_schema.eventswaits：从 performance_schema.events_waits_summary_global_by_event_name 采集指标。 • perf_schema.file_events：从 performance_schema.file_summary_by_event_name 采集指标。 • perf_schema.file_instances：从 performance_schema.file_summary_by_instance采集指标。 • perf_schema.indexiowaits：从 performance_schema.table_io_waits_summary_by_index_usage 采集指标。 • perf_schema.tableiowaits：从 performance_schema.table_io_waits_summary_by_table 采集指标。 • perf_schema.tablelocks：从 performance_schema.table_lock_waits_summary_by_table 采集指标。 • perf_schema.replication_group_member_stats：从 performance_schema.replication_group_member_stats 采集指标。 • perf_schema.replication_applier_status_by_worker：从 performance_schema.replication_applier_status_by_worker 采集指标。 • slave_status：从 SHOW SLAVE STATUS 采集，默认为 true。 • slave_hosts：从 SHOW SLAVE HOSTS 采集。

- heartbeat：从 heartbeat 采集。
- heartbeat.database：收集 heartbeat 数据的数据库，默认值为 heartbeat。
- heartbeat.table：收集 heartbeat 数据的表，默认值为 heartbeat。

方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

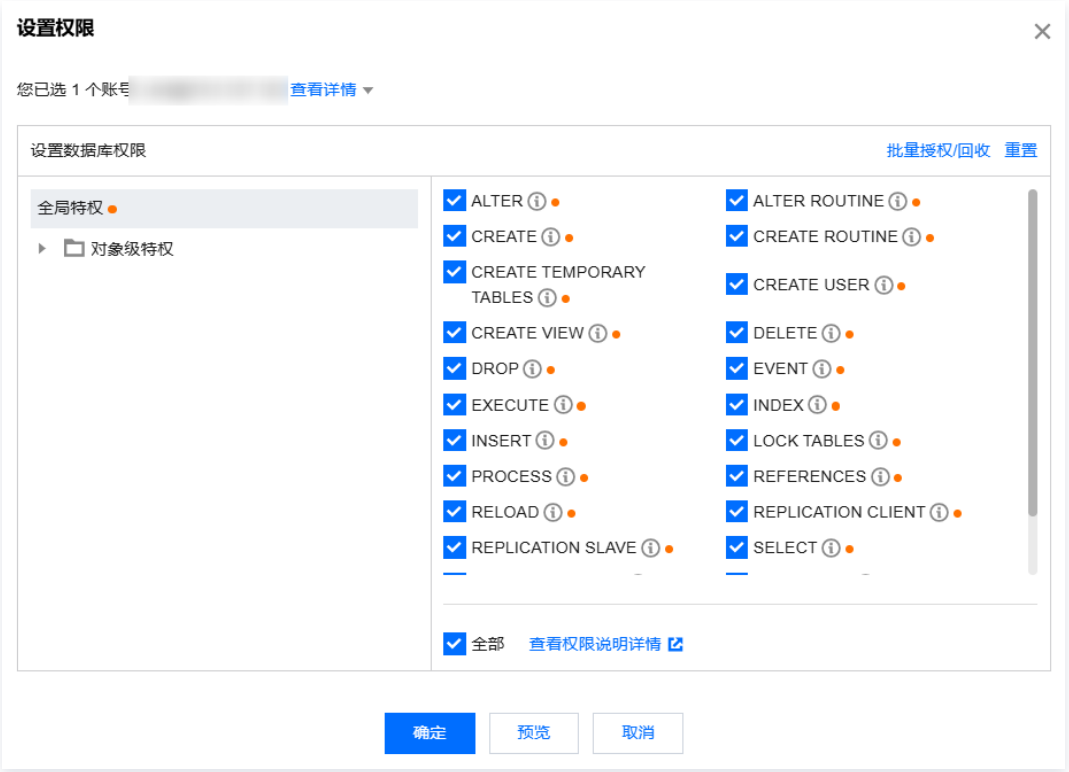
- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 Prometheus 监控服务控制台，选择并进入对应的 Prometheus 实例，在**数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

操作步骤

步骤1：数据库授权

因为 MySQL Exporter 是通过查询数据库中状态数据来对其进行监控，所以需要为对应的数据库实例进行授权。账号和密码需根据实际情况而定，授权步骤如下：

1. 登录 云数据库 MySQL 控制台。
2. 在实例列表页面单击需要授权的数据库名称，进入数据库详情页。
3. 选择**数据库管理 > 账号管理**，进入账号管理页面，根据业务实际需要创建监控建立的账号。
4. 单击账号右侧操作项下的**修改权限**，修改对应权限。示例如下图所示：



您也可以在您的云服务器中通过执行以下命令进行授权。

```
CREATE USER 'exporter'@'ip' IDENTIFIED BY 'XXXXXXXX' WITH MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter'@'ip';
```

> **说明**

> 建议为该用户设置最大连接数限制，以避免因监控数据抓取对数据库带来影响。但并非所有的数据库版本中都可以生效，例如 MariaDB 10.1 版本不支持最大连接数设置，则无法生效。详情请参见 MariaDB 说明 。

步骤2：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏选择**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 使用 Secret 管理 MySQL 连接串。
   4.1 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
   4.2 在页面右上角单击 **YAML 创建**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理连接串，并对连接串进行加密处理，在启动 MySQL Exporter 的时候直接使用 Secret Key，需要调整对应的**连接串**，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret-test
  namespace: mysql-demo
type: Opaque
stringData:
  datasource: "user:password@tcp(ip:port)/"  #对应 MySQL 连接串信息
```

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

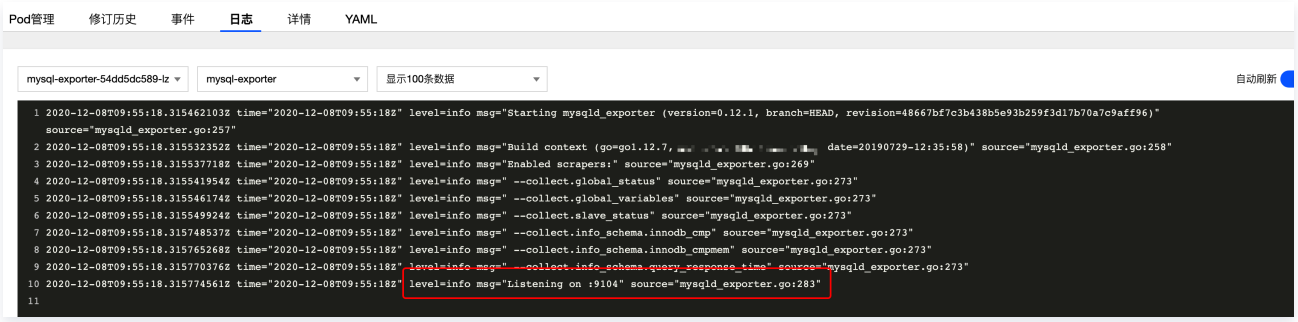
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
  name: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
  namespace: mysql-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
  template:
    metadata:
      labels:
        k8s-app: mysql-exporter  # 根据业务需要调整成对应的名称，建议加上 MySQL 实例的信息
    spec:
      containers:
      - env:
        - name: DATA_SOURCE_NAME
          valueFrom:
            secretKeyRef:
              name: mysql-secret-test # 对应上一步中的 Secret 的名称
              key: datasource # 对应上一步中的 Secret Key
        image: ccr.ccs.tencentyun.com/rig-agent/mysqld-exporter:v0.12.1
        imagePullPolicy: IfNotPresent
        name: mysql-exporter
```

```
      ports:
      - containerPort: 9104
        name: metric-port
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```

6. 验证。

6.1 在 **Deployment** 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

6.2 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：

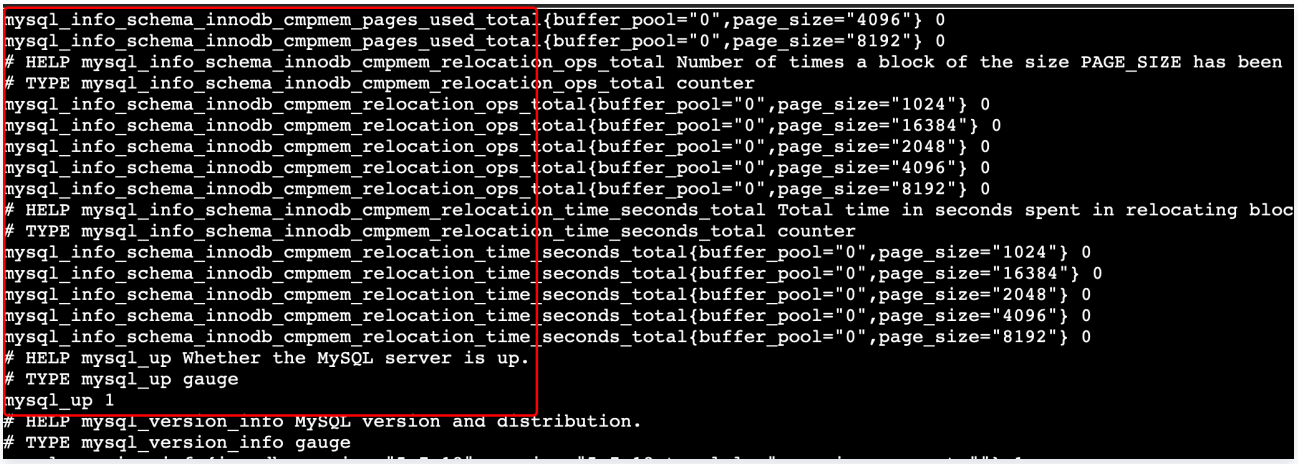


6.3 单击 **Pod 管理**页签进入 Pod 页面。

6.4 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令，可以正常得到对应的 MySQL 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -O- localhost:9104/metrics
```

执行结果如下图所示：



步骤2：添加采集任务

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集** > **集成容器服务**，选择已经关联的集群，通过**数据采集配置** > **新建自定义监控** > **YAML 编辑**来添加采集配置。
3. 通过服务发现添加 **PodMonitors** 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: mysql-exporter  # 填写一个唯一名称
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不
要修改
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 MySQL 实例 ID
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: ip
        replacement: '1.x.x.x' # 调整成对应的 MySQL 实例 IP
    namespaceSelector:    # 选择要监控pod所在的namespace
      matchNames:
      - mysql-demo
    selector:  # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: mysql-exporter
```

查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，进入集成中心页面，找到并单击 **MySQL**，选择 **Dashboard > Dashboard 操作**下的**安装/升级 Dashboard**，单击**安装/升级**安装对应的 Grafana Dashboard。
3. 选择**已集成**，在已集成列表中单击 Grafana 图标即可自动打开 MySQL 监控大盘，查看实例相关的监控数据，如下图所示：

配置告警

腾讯云 Prometheus 托管服务内置了部分 MySQL 数据库的报警策略模板，可根据业务实际的情况调整对应的阈值来添加告警策略。详情请参见 新建告警策略 。

附录：MySQL Exporter 采集参数说明

MySQL Exporter 使用各种 `Collector` 来控制采集数据的启停，具体参数如下：

名称	MySQL 版本	描述
collect.auto_increment.columns	5.1	在 information_schema 中采集 auto_increment 和最大值。
collect.binlog_size	5.1	采集所有注册的 binlog 文件大小。
collect.engine_innodb_status	5.1	从 SHOW ENGINE INNODB STATUS 中采集状态数据。
collect.engine_tokudb_status	5.6	从 SHOW ENGINE TOKUDB STATUS 中采集状态数据。
collect.global_status	5.1	从 SHOW GLOBAL STATUS（默认开启）中采集状态数据。
collect.global_variables	5.1	从 SHOW GLOBAL VARIABLES（默认开启）中采集状态数据。
collect.info_schema.clientstats	5.5	如果设置了 userstat=1，设置成 true 来开启用户端数据采集。
collect.info_schema.innodb_metrics	5.6	从 information_schema.innodb_metrics 中采集监控数据。
collect.info_schema.innodb_tablespaces	5.7	从 information_schema.innodb_sys_tablespaces 中采集监控数据。
collect.info_schema.innodb_cmp	5.5	从 information_schema.innodb_cmp 中采集 InnoDB 压缩表的监控数据。
collect.info_schema.innodb_cmpmem	5.5	从 information_schema.innodb_cmpmem 中采集 InnoDB buffer pool compression 的监控数据。
collect.info_schema.processlist	5.1	从 information_schema.processlist 中采集线程状态计数的监控数据。
collect.info_schema.processlist.min_time	5.1	线程可以被统计所维持的状态的最小时间。（默认：0）
collect.info_schema.query_response_time	5.5	如果 query_response_time_stats 被设置成 ON，采集查询相应时间的分布。

collect.info_schema.replica_host	5.6	从 information_schema.replica_host_status 中采集状态数据。
collect.info_schema.tables	5.1	从 information_schema.tables 中采集状态数据。
collect.info_schema.tables.databases	5.1	设置需要采集表状态的数据库, 或者设置成 '`*`' 来采集所有的。
collect.info_schema.tablestats	5.1	如果设置了 userstat=1，设置成 true 来采集表统计数据。
collect.info_schema.schemastats	5.1	如果设置了 userstat=1，设置成 true 来采集 schema 统计数据。
collect.info_schema.userstats	5.1	如果设置了 userstat=1，设置成 true 来采集用户统计数据。
collect.perf_schema.eventsstatements	5.6	从 performance_schema.events_statements_summary_by_digest 中采集监控数据。
collect.perf_schema.eventsstatements.digest_text_limit	5.6	设置正常文本语句的最大长度。(默认：120)
collect.perf_schema.eventsstatements.limit	5.6	事件语句的限制数量。（默认：250）
collect.perf_schema.eventsstatements.timelimit	5.6	限制事件语句 'last_seen' 可以保持多久，单位为秒。(默认：86400)
collect.perf_schema.eventsstatementssum	5.7	从 performance_schema.events_statements_summary_by_digest summed 中采集监控数据。
collect.perf_schema.eventswaits	5.5	从 performance_schema.events_waits_summary_global_by_event_name 中采集监控数据。
collect.perf_schema.file_events	5.6	从 performance_schema.file_summary_by_event_name 中采集监控数据。
collect.perf_schema.file_instances	5.5	从 performance_schema.file_summary_by_instance 中采集监控数据。
collect.perf_schema.indexiowaits	5.6	从 performance_schema.table_io_waits_summary_by_index_usage 中采集监控数据。
collect.perf_schema.tableiowaits	5.6	从 performance_schema.table_io_waits_summary_by_table 中采集监控数据。
collect.perf_schema.tablelocks	5.6	从 performance_schema.table_lock_waits_summary_by_table 中采集监控数据。
collect.perf_schema.replication_group_members	5.7	从 performance_schema.replication_group_members 中采集监控数据。
collect.perf_schema.replication_group_member_stats	5.7	从 from performance_schema.replication_group_member_stats 中采集监控数据。
collect.perf_schema.replication_applier_status_by_worker	5.7	从 performance_schema.replication_applier_status_by_worker 中采集监控数据。
collect.slave_status	5.1	从 SHOW SLAVE STATUS（默认开启）中采集监控数据。

collect.slave_hosts	5.1	从 SHOW SLAVE HOSTS 中采集监控数据。
collect.heartbeat	5.1	从 heartbeat 中采集监控数据。
collect.heartbeat.database	5.1	数据库心跳检测的数据源。（默认：heartbeat）
collect.heartbeat.table	5.1	表心跳检测的数据源。（默认：heartbeat）
collect.heartbeat.utc	5.1	对当前的数据库服务器使用 UTC 时间戳（`pt-heartbeat` is called with `--utc`）。（默认：false）

全局配置参数

名称	描述
config.my-cnf	用来读取数据库认证信息的配置文件 `.my.cnf` 位置。（默认：`~/.my.cnf`）
log.level	日志级别。（默认：info）
exporter.lock_wait_timeout	为链接设置 lock_wait_timeout（单位：秒）以避免对元数据的锁时间太长。（默认：2）
exporter.log_slow_filter	添加 log_slow_filter 以避免抓取的慢查询被记录。 **说明：** 不支持 Oracle MySQL。
web.listen-address	web 端口监听地址。
web.telemetry-path	metrics 接口路径。
version	打印版本信息。

heartbeat 心跳检测

如果开启 `collect.heartbeat`，mysqld_exporter 会通过心跳检测机制抓取复制延迟数据。

SQL Server Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

Microsoft SQL Server Exporter 是社区专门为采集 Microsoft SQL Server（简称：MSSQL）数据库监控指标而设计开发，通过 MSSQL Exporter 上报性能和健康状况等的指标数据，用于监控大盘展示和异常报警。腾讯云可观测平台 Prometheus 提供了与 MSSQL Exporter 集成及开箱即用的 Grafana 监控大盘。

> **说明：**
> 如果需要监控的 MSSQL 是腾讯云 云数据库 SQL Server，推荐使用集成中心 云监控集成，支持一键采集云产品指标。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **MSSQL**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
用户名	MSSQL 的用户名称。
密码	MSSQL 的密码。
域名	MSSQL 的服务域名。
地址	MSSQL 的服务端口。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 **Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 MSSQL Exporter > 验证 步骤完成 Exporter 部署。

步骤二：部署 MSSQL Exporter

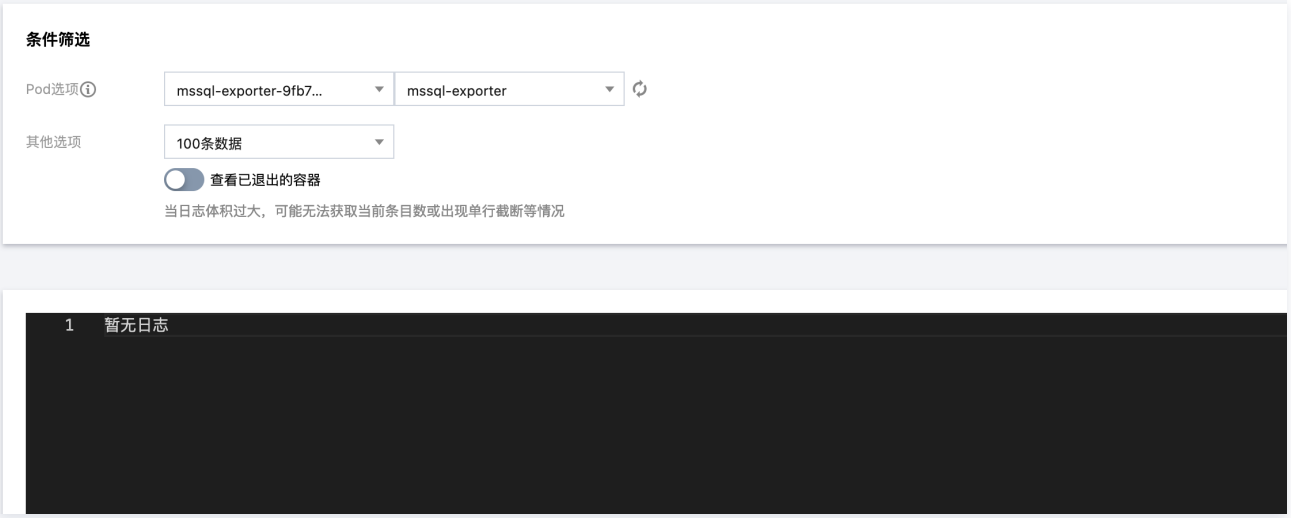
1. 在左侧菜单中选择**工作负载 > Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
  name: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
  namespace: mssql-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
  template:
    metadata:
      labels:
        k8s-app: mssql-exporter  # 根据业务需要调整成对应的名称，建议加上 MSSQL 实例的信息
```

```
    spec:
      containers:
      - env:
        - name: SERVER
          value: "127.0.0.1"  # 根据业务需要调整成对应的 MSSQL 的域名
        - name: PORT
          value: "1433"  # 根据业务需要调整成对应的 MSSQL 的端口
        - name: USERNAME
          value: user  # 根据业务需要调整成对应的 MSSQL 的用户名
        - name: PASSWORD
          value: "123456"  # 根据业务需要调整成对应的 MSSQL 的密码
        - name: EXPOSE
          value: "4000"  # 暴露指标端口，根据业务需要调整成对应的端口
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:mssql-exporter-v1.3.0
        imagePullPolicy: IfNotPresent
        name: mssql-exporter
        ports:
        - containerPort: 4000  # 开放上述环境变量 EXPOSE 所对应的端口
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

步骤三：验证

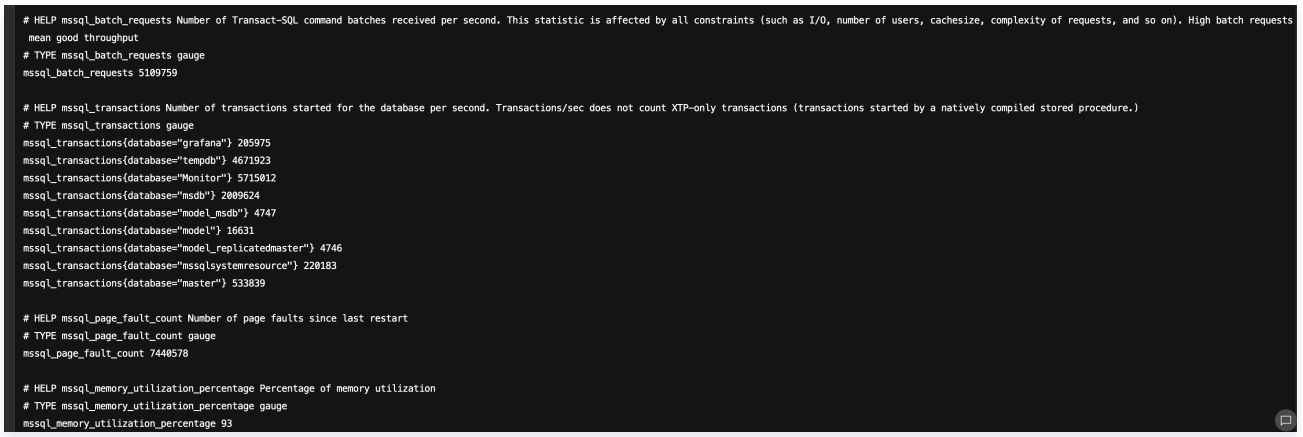
1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，无报错信息输出即可，如下图所示：



3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 MSSQL 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:4000/metrics
```

执行结果如下图所示：



```
# HELP mssql_batch_requests Number of Transact-SQL command batches received per second. This statistic is affected by all constraints (such as I/O, number of users, cachesize, complexity of requests, and so on). High batch requests
 mean good throughput
# TYPE mssql_batch_requests gauge
mssql_batch_requests 5109759

# HELP mssql_transactions Number of transactions started for the database per second. Transactions/sec does not count XTP-only transactions (transactions started by a natively compiled stored procedure.)
# TYPE mssql_transactions gauge
mssql_transactions{database="grafana"} 205975
mssql_transactions{database="tempdb"} 4671923
mssql_transactions{database="Monitor"} 5715012
mssql_transactions{database="msdb"} 2009624
mssql_transactions{database="model_msdb"} 4747
mssql_transactions{database="model"} 16631
mssql_transactions{database="model_replicatedmaster"} 4746
mssql_transactions{database="mssqlsystemresource"} 220183
mssql_transactions{database="master"} 533839

# HELP mssql_page_fault_count Number of page faults since last restart
# TYPE mssql_page_fault_count gauge
mssql_page_fault_count 7440578

# HELP mssql_memory_utilization_percentage Percentage of memory utilization
# TYPE mssql_memory_utilization_percentage gauge
mssql_memory_utilization_percentage 93
```

步骤四：添加采集任务

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: mssql-exporter  # 填写一个唯一名称
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不
要修改
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 MSSQL 实例 ID
    namespaceSelector:    # 选择要监控 mssql exporter pod所在的namespace
      matchNames:
      - mssql-demo
    selector:  # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: mssql-exporter
```

查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。

2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 database 文件夹中找到 mssql 实例监控面板，查看实例相关监控数据，如下图所示：

配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：MSSQL Exporter 环境变量配置

名称	描述
SERVER	必需，MSSQL 服务 IP 或 域名
PORT	MSSQL 服务端口，默认1433
USERNAME	必需，MSSQL 服务用户名
PASSWORD	必需，MSSQL 服务密码
ENCRYPT	强制加密设置，默认为true
TRUST_SERVER_CERTIFICATE	是否信任服务器的证书设置，默认为true
DEBUG	逗号分割的启用日志列表，可选 currently supports app 和 metrics

Oracle DB Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

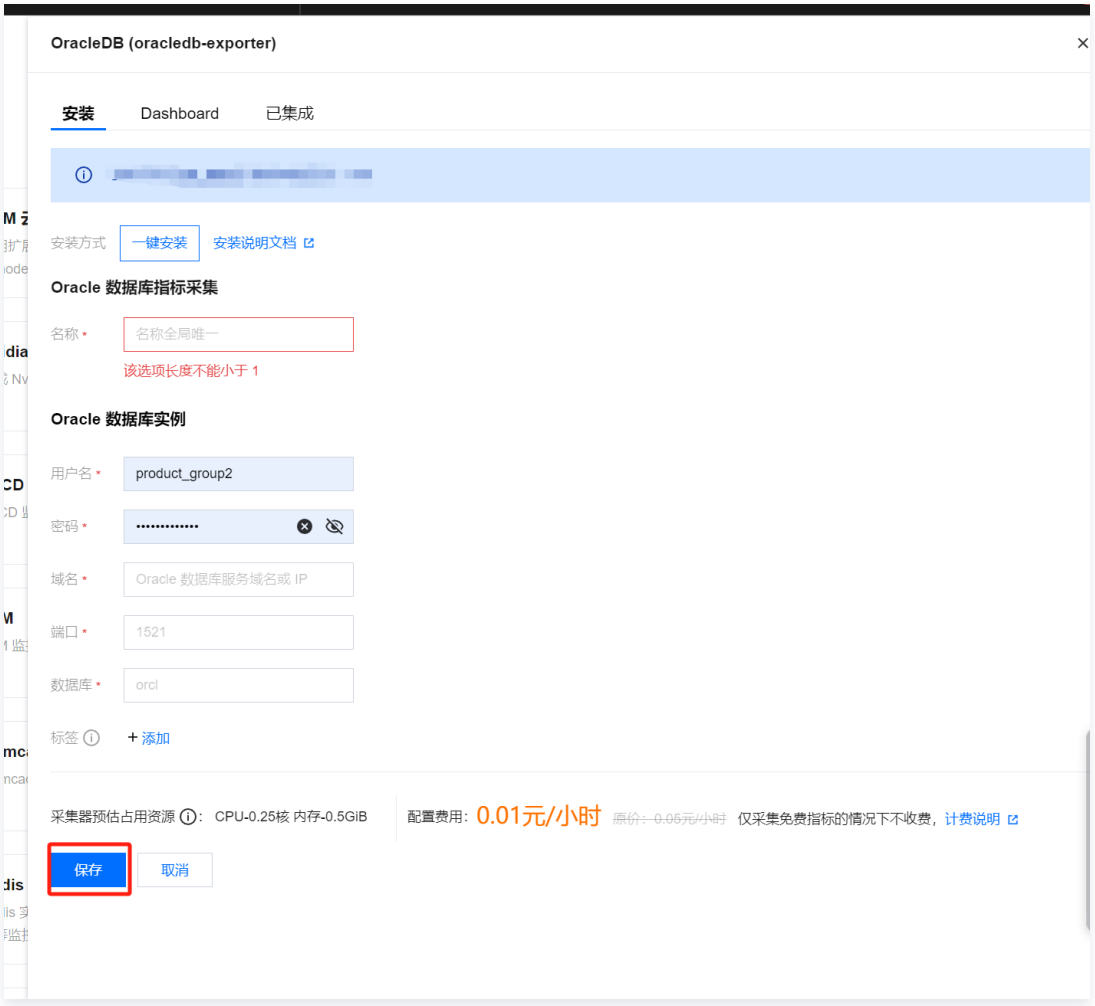
OracleDB Exporter 用于从 Oracle 数据库中抓取指标，并将其通过 Prometheus 指标的方式向外暴露的开源组件，通过该 Exporter 上报的性能、负载及健康状况等指标数据，用于监控大盘展示和异常报警。腾讯云可观测平台 Prometheus 提供了与 OracleDB Exporter 集成及开箱即用的 Grafana 监控大盘。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **OracleDB**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。

	• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
用户名	OracleDB 的用户名称。
密码	OracleDB 的密码。
域名	OracleDB 的服务域名。
端口	OracleDB 的服务端口。
数据库	OracleDB 的数据库名称。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间。
- 在 Prometheus 监控控制台 > **选择对应的 Prometheus 实例** > **数据采集** > **集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 使用 Secret 管理 OracleDB 连接串 > 部署 OracleDB Exporter > 验证 步骤完成 Exporter 部署。

步骤二：使用 Secret 管理 OracleDB 连接串

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，配置说明如下：
   使用 Kubernetes 的 Secret 来管理连接串，并对连接串进行加密处理，在启动 OracleDB Exporter 的时候直接使用 Secret Key，需要调整对应的**连接串**，YAML 配置示例如下：

```
apiVersion: v1
kind: Secret
metadata:
  name: oracledb-secret-test  # 根据业务需要调整成对应的名称
  namespace: oracledb-demo  # 根据业务需要调整成对应的命名空间
type: Opaque
stringData:
  datasource: "oracle://test:123456/127.0.0.1:1521/ORCLPDB1"  # 对应 OracleDB 连接串信息
  # test为用户名，123456为用户密码，127.0.0.1为数据库IP或者域名，1521为数据库端口，ORCLPDB1为数据库名称
```

步骤三：部署 OracleDB Exporter

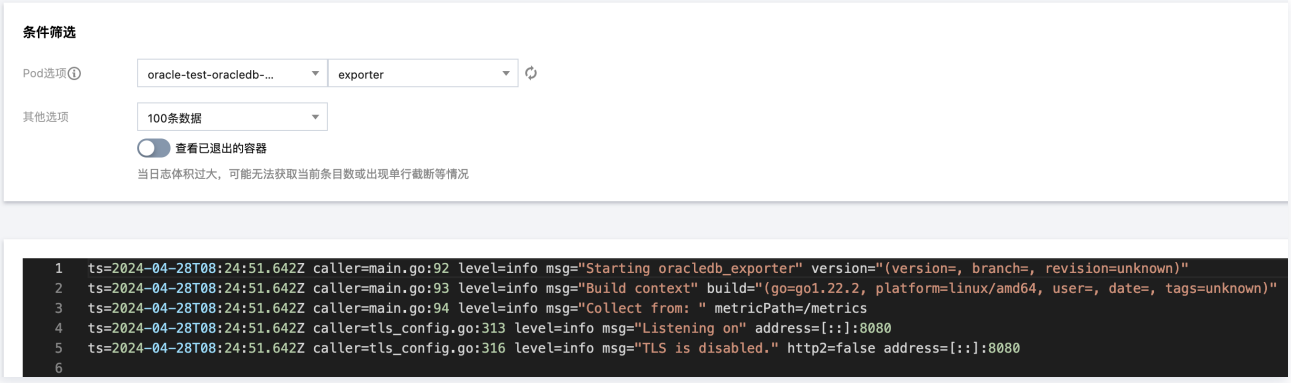
1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 管理页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  labels:
    k8s-app: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
  name: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
  namespace: oracledb-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
  template:
    metadata:
      labels:
        k8s-app: oracledb-exporter  # 根据业务需要调整成对应的名称，建议加上 OracleDB 实例的信息
    spec:
      containers:
      - args:
        - --web.listen-address=:8080
        envFrom:
        - secretRef:
            name: oracledb-secret-test  # 上一步创建出来的Secret名称
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:oracledb-exporter-v0.6.0-alpine
        imagePullPolicy: IfNotPresent
        name: oracledb-exporter
        ports:
        - containerPort: 8080
          name: metrics
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

步骤四：验证

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。
2. 单击**日志**页签，无报错信息输出即可，如下图所示：



3. 单击 **Pod 管理**页签进入 Pod 页面。

4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 wget 命令对应 Exporter 暴露的地址，可以正常得到对应的 OracleDB 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
wget -qO- http://localhost:8080/metrics
```

执行结果如下图所示：

```
# HELP oracledb_tablespace_bytes Generic counter metric of tablespaces bytes in Oracle.
# TYPE oracledb_tablespace_bytes gauge
oracledb_tablespace_bytes{tablespace="SYSAUX",type="PERMANENT"} 5.0397184e+08
oracledb_tablespace_bytes{tablespace="SYSTEM",type="PERMANENT"} 2.56049152e+08
oracledb_tablespace_bytes{tablespace="TEMP",type="TEMPORARY"} 0
oracledb_tablespace_bytes{tablespace="USERS",type="PERMANENT"} 1.048576e+06
# HELP oracledb_tablespace_free Generic counter metric of tablespaces free bytes in Oracle.
# TYPE oracledb_tablespace_free gauge
oracledb_tablespace_free{tablespace="SYSAUX",type="PERMANENT"} 9.05609216e+09
oracledb_tablespace_free{tablespace="SYSTEM",type="PERMANENT"} 9.031385088e+09
oracledb_tablespace_free{tablespace="TEMP",type="TEMPORARY"} 9.046269952e+09
oracledb_tablespace_free{tablespace="USERS",type="PERMANENT"} 9.029484544e+09
# HELP oracledb_tablespace_max_bytes Generic counter metric of tablespaces max bytes in Oracle.
# TYPE oracledb_tablespace_max_bytes gauge
oracledb_tablespace_max_bytes{tablespace="SYSAUX",type="PERMANENT"} 9.560064e+09
oracledb_tablespace_max_bytes{tablespace="SYSTEM",type="PERMANENT"} 9.28743424e+09
oracledb_tablespace_max_bytes{tablespace="TEMP",type="TEMPORARY"} 9.046269952e+09
oracledb_tablespace_max_bytes{tablespace="USERS",type="PERMANENT"} 9.03053312e+09
# HELP oracledb_tablespace_used_percent Gauge metric showing as a percentage of how much of the tablespace has been used.
# TYPE oracledb_tablespace_used_percent gauge
oracledb_tablespace_used_percent{tablespace="SYSAUX",type="PERMANENT"} 5.271636675235647
oracledb_tablespace_used_percent{tablespace="SYSTEM",type="PERMANENT"} 2.756941749285538
oracledb_tablespace_used_percent{tablespace="TEMP",type="TEMPORARY"} 0
oracledb_tablespace_used_percent{tablespace="USERS",type="PERMANENT"} 0.011611451794332161
```

步骤五：添加采集任务

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```
  apiVersion: monitoring.coreos.com/v1
  kind: PodMonitor
  metadata:
    name: oracledb-exporter  # 填写一个唯一名称
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例(已停止售卖)：namespace 固定，不
要修改
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 OracleDB 实例 ID
    namespaceSelector:    # 选择要监控 oracledb exporter pod所在的namespace
      matchNames:
      - oracledb-demo
    selector:  # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: oracledb-exporter
```

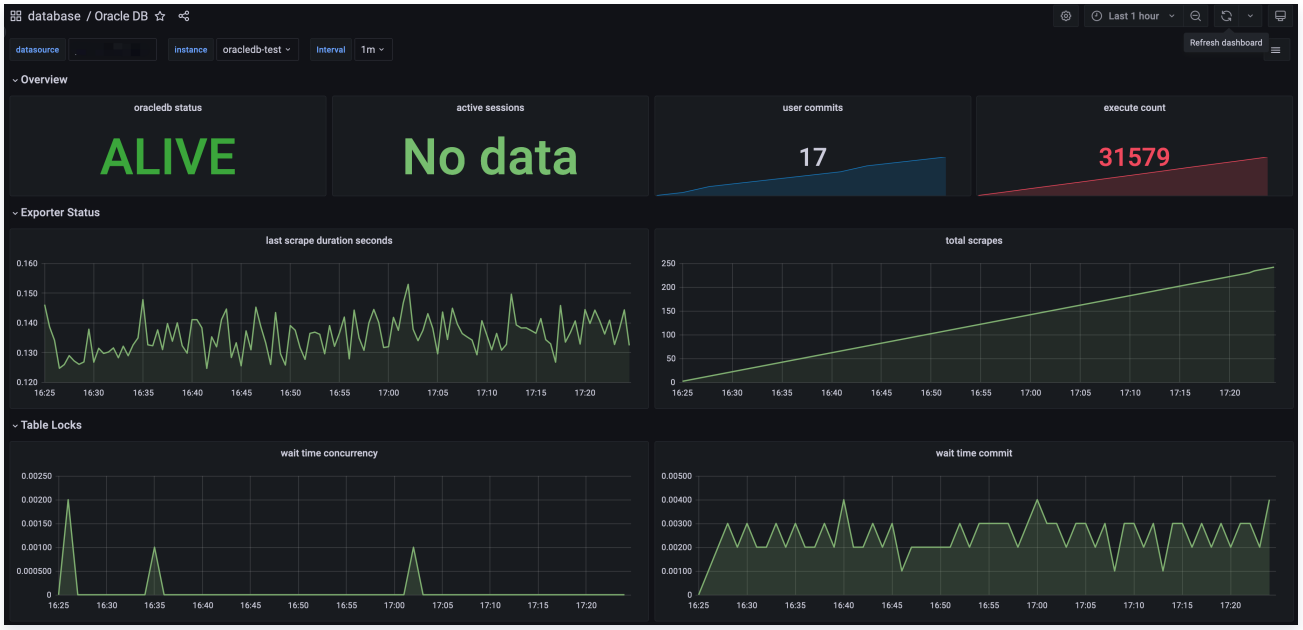
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 database 文件夹中找到 Oracle DB 实例监控面板，查看实例相关监控数据，如下图所示：



配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：OracleDB Exporter 环境变量配置

名称	描述
web.telemetry-path	指标暴露路径，默认 `/metrics` 。
web.systemd-socket	使用 systemd 套接字监听器代替端口监听器（仅限 Linux）。
web.listen-address	监听地址，默认：9161。
web.config.file	配置文件的路径，可以启用 TLS 或身份验证。
log.level	日志级别，可选值列表[debug, info, warn, error, fatal]。
log.format	日志消息的输出格式，示例 logger:syslog?appname=bob&local=7 或 logger:stdout?json=true，默认 stderr。
custom.metrics	自定义指标配置路径。
default.metrics	默认指标配置路径。
database.maxIdleConns	最大空闲连接数，默认0。

database.maxOpenConns	最大打开连接数，默认0。
database.dsn	数据库 dsn 串。
database.dsnFile	读取 dsn 串的文件。
query.timeout	采集查询超时，默认5s。
scrape.interval	抓取间隔设置。

Consul Exporter 接入

最近更新时间：2024-10-24 16:23:33

操作场景

在使用 Consul 过程中需要对 Consul 运行状态进行监控，以便了解 Consul 服务是否运行正常，排查 Consul 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Consul 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务 Consul。

操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Consul**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

名称	描述
名称	每个集成需要一个唯一名称
地址	要采集的 Consul 实例的地址和端口
标签	增加具有业务含义的标签，会自动添加到 Prometheus 的 Label 中

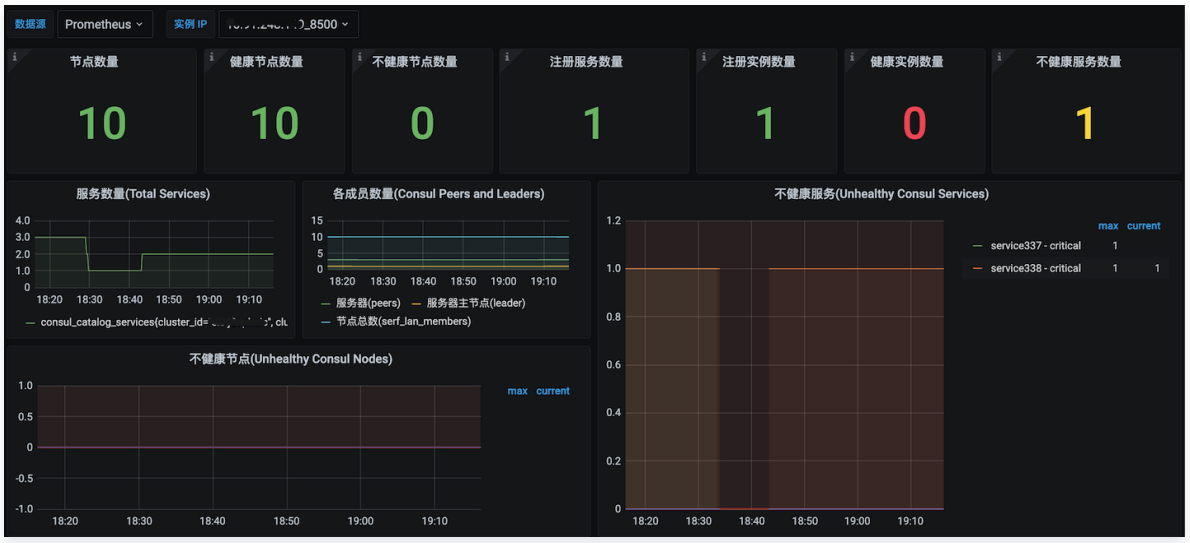
查看监控

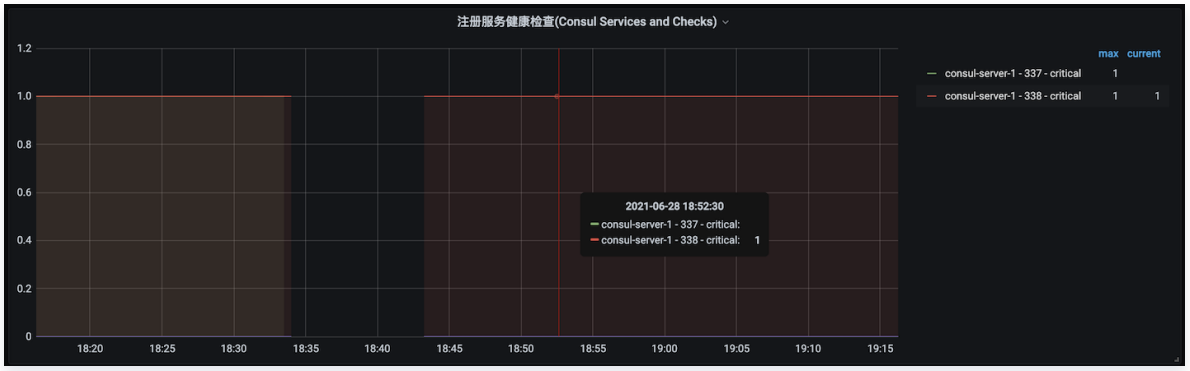
前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，在集成中心页面找到 Consul 监控，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可通过监控大盘清晰看到如下监控状态：
   - Consul 集群节点状态
   - Consul 上注册服务的状态


数据源
Prometheus
实例 IP
节点数量
10
健康节点数量
10
不健康节点数量
0
注册服务数量
1
注册实例数量
1
健康实例数量
0
不健康服务数量
1
服务数量(Total Services)
consul_catalog_services{cluster_id=
各成员数量(Consul Peers and Leaders)
服务器(peers)
服务器主节点(leader)
节点总数(serf_lan_members)
不健康服务(Unhealthy Consul Services)
max current
service337 - critical 1
service338 - critical 1 1
不健康节点(Unhealthy Consul Nodes)
max current


注册服务健康检查(Consul Services and Checks)
max current
consul-server-1 - 337 - critical 1
consul-server-1 - 338 - critical 1 1
2021-06-28 18:52:30
consul-server-1 - 337 - critical:
consul-server-1 - 338 - critical: 1

Ingress NGINX Controller Exporter 接入

最近更新时间：2024-10-24 16:23:33

操作场景

在使用 Ingress NGINX Controller 过程中需要对 Ingress NGINX Controller 运行状态进行监控，以便了解 Ingress NGINX Controller 服务是否运行正常，排查 Ingress NGINX Controller 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Ingress NGINX Controller 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务 Ingress NGINX Controller。

操作步骤

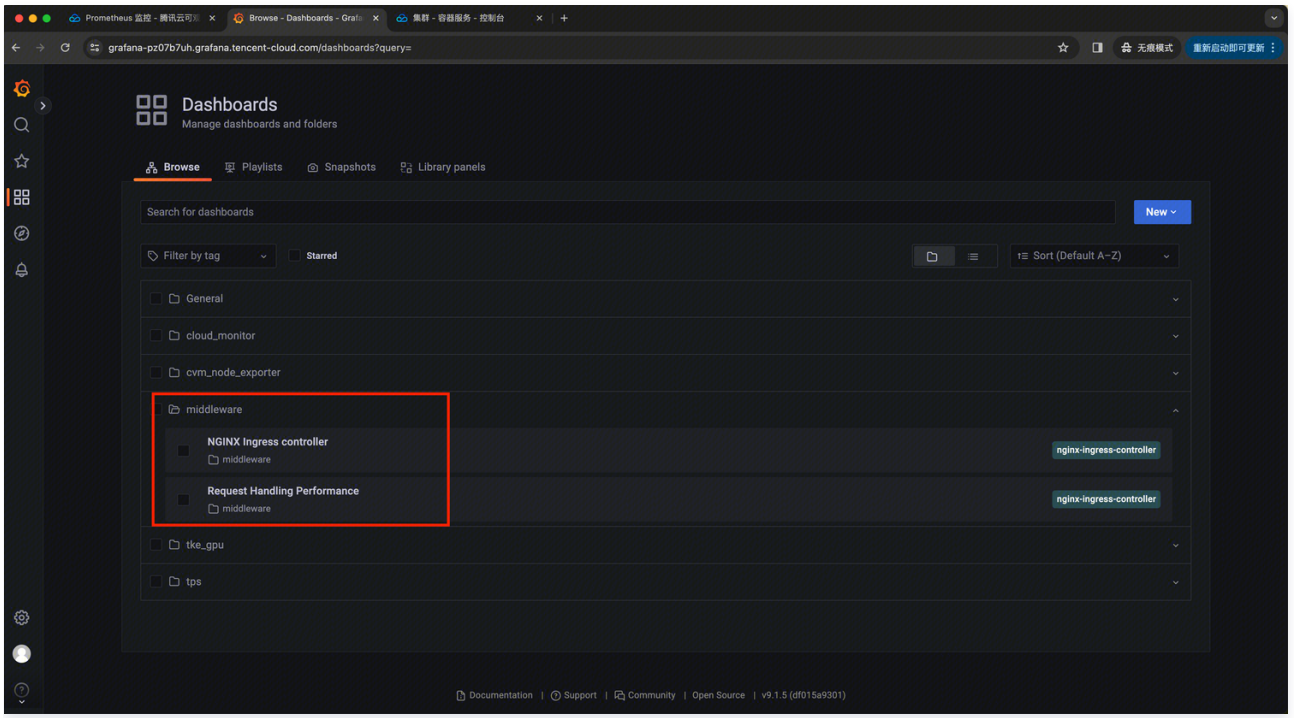
1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **Ingress NGINX Controller**，即会弹出一个安装窗口，在安装页面填写集成名称，选取待监控的 nginx-ingress controller 所在的集群以及它的实例名，然后单击**保存**。

配置说明

名称	描述
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
集群	选取待监控的 nginx-ingress controller 所在的集群
实例	选取待监控的 nginx-ingress controller 所在的实例名

查看监控

待部署成功后（1分钟之内），在 Prometheus 相关联的 Grafana 里，找到 nginx-ingress 相关面板，即可观察上述 nginx-ingress controller 的 dashboards。


Dashboards
Manage dashboards and folders
Browse
Playlists
Snapshots
Library panels
Search for dashboards
New
Filter by tag
Starred
Sort (Default A–Z)
General
cloud_monitor
cvm_node_exporter
middleware
NGINX Ingress controller
middleware
nginx-ingress-controller
Request Handling Performance
middleware
nginx-ingress-controller
tke_gpu
tps
Documentation
Support
Community
Open Source
v9.1.5 (df015a9301)


middleware / NGINX Ingress controller
Controller Request Volume
0.000554 ops/s
Controller Connections
3.47
Controller Success Rate (non-4/5xx responses)
0%
Config Reloads
0
Last Config Failed
N/A
Ingress Request Volume
Ingress Success Rate (non-4/5xx responses)
Network I/O pressure
Average Memory Usage
Average CPU Usage
Ingress Percentile Response Times and Transfer Rates
Ingress Percentile Response Times (Ingress Namespaces)
Ingress Request Latency Heatmap (Ingress Namespaces)


grafana-pz07b7uh.grafana.tencent-cloud.com/d/c4282b4ce0a34f2fb0b1086c9ccb13bd/request-handling-performance?orgid=1
middleware / Request Handling Performance
Request Latency Percentiles
Upstream Response Latency Percentiles
Request Rate by Method and Path
Median Upstream Response Time by Method and Path
Response Error Rate by Method and Path
Upstream Response Time by Method and Path
Response Error Rate by Method and Path
Average Response Size by Method and Path

TKE GPU Exporter 接入

最近更新时间：2025-06-09 18:16:52

操作场景

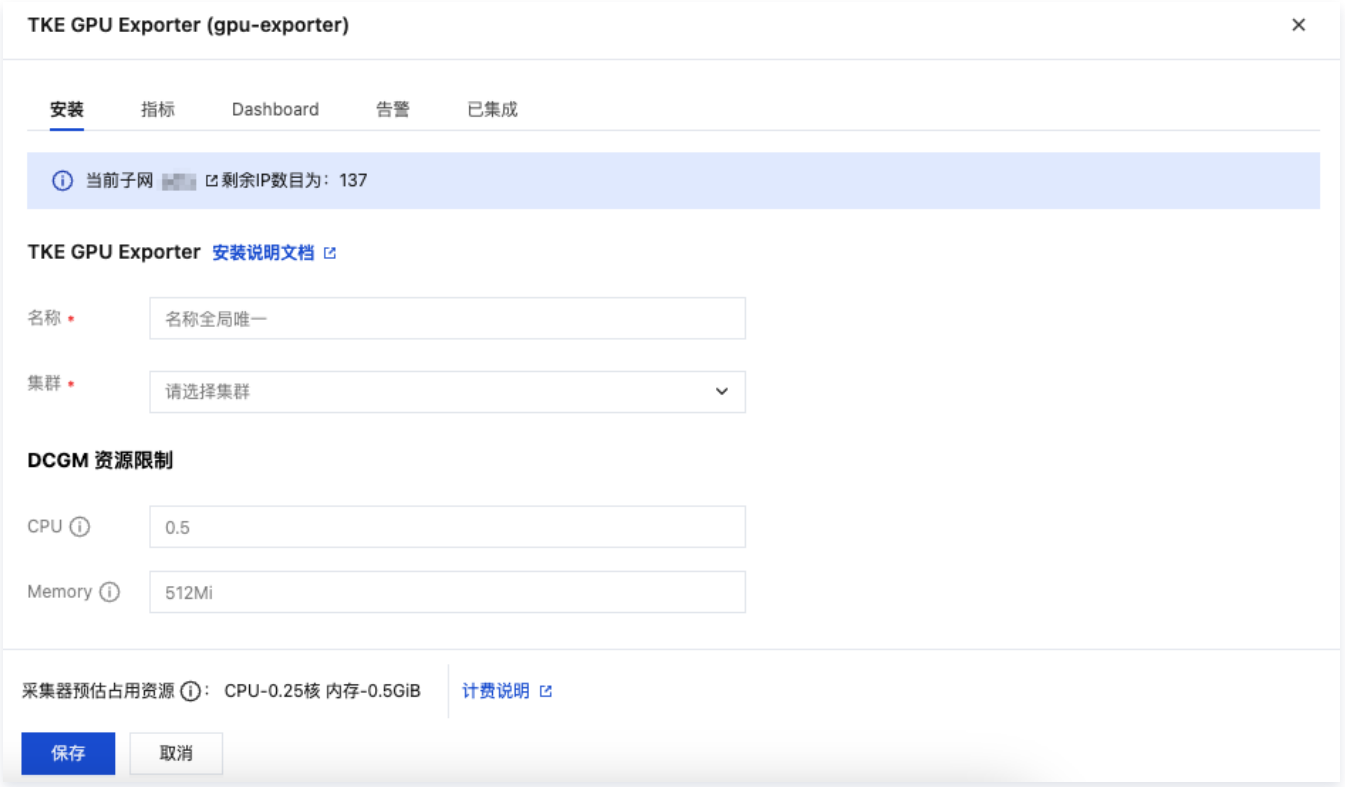
在使用 TKE GPU 资源过程中需要对资源使用状态进行监控，以便了解 GPU 服务是否运行正常，排查 GPU 资源故障。Prometheus 监控服务提供基于 Exporter 的方式来监控容器化 GPU 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务部署 TKE GPU Exporter 以及实现 TKE GPU Exporter 告警接入等操作。

前提条件

Prometheus 实例已成功关联待监控的 TKE 集群，步骤参考 集成容器服务 。

操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击 **TKE GPU Exporter**，即会弹出一个安装窗口，在安装页面填写集成名称，选取待监控的 GPU 所在的集群，然后单击**保存**。



配置说明

名称	描述
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
集群	选取待监控的 GPU 所在的集群。
CPU	DCGM Exporter 工作负载的 CPU 限制，以 CPU 核心数为单位，需符合 Kubernetes 资源限制格式，具体支持的格式如下：

	• 小数或整数，例如0.5表示0.5个 CPU 核心。 • 毫核值，整数后加单位m，例如500m表示500毫核（等同于0.5核心）。
Memory	DCGM Exporter 工作负载的内存限制，以字节为单位，需符合 Kubernetes 资源限制格式，具体支持的格式如下： • 纯整数，例如1048576表示1048576字节。 • 整数后加十进制单位，支持 P、T、G、M、k，例如100k表示100,000字节。 • 整数后加二进制单位，支持Pi、Ti、Gi、Mi、Ki，例如256Mi表示约268,435,456字节。

查看监控

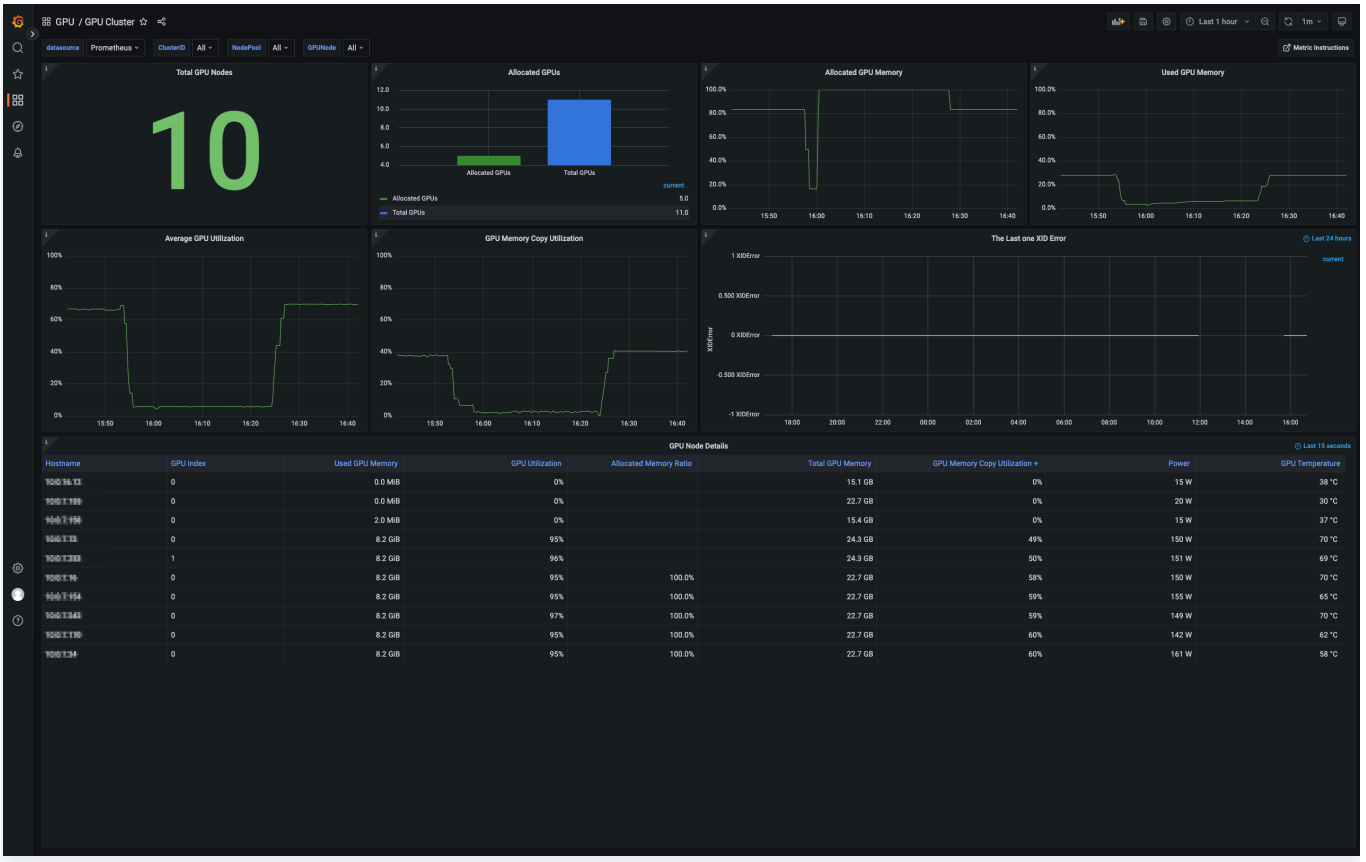
前提条件

Prometheus 实例已绑定 Grafana 实例。

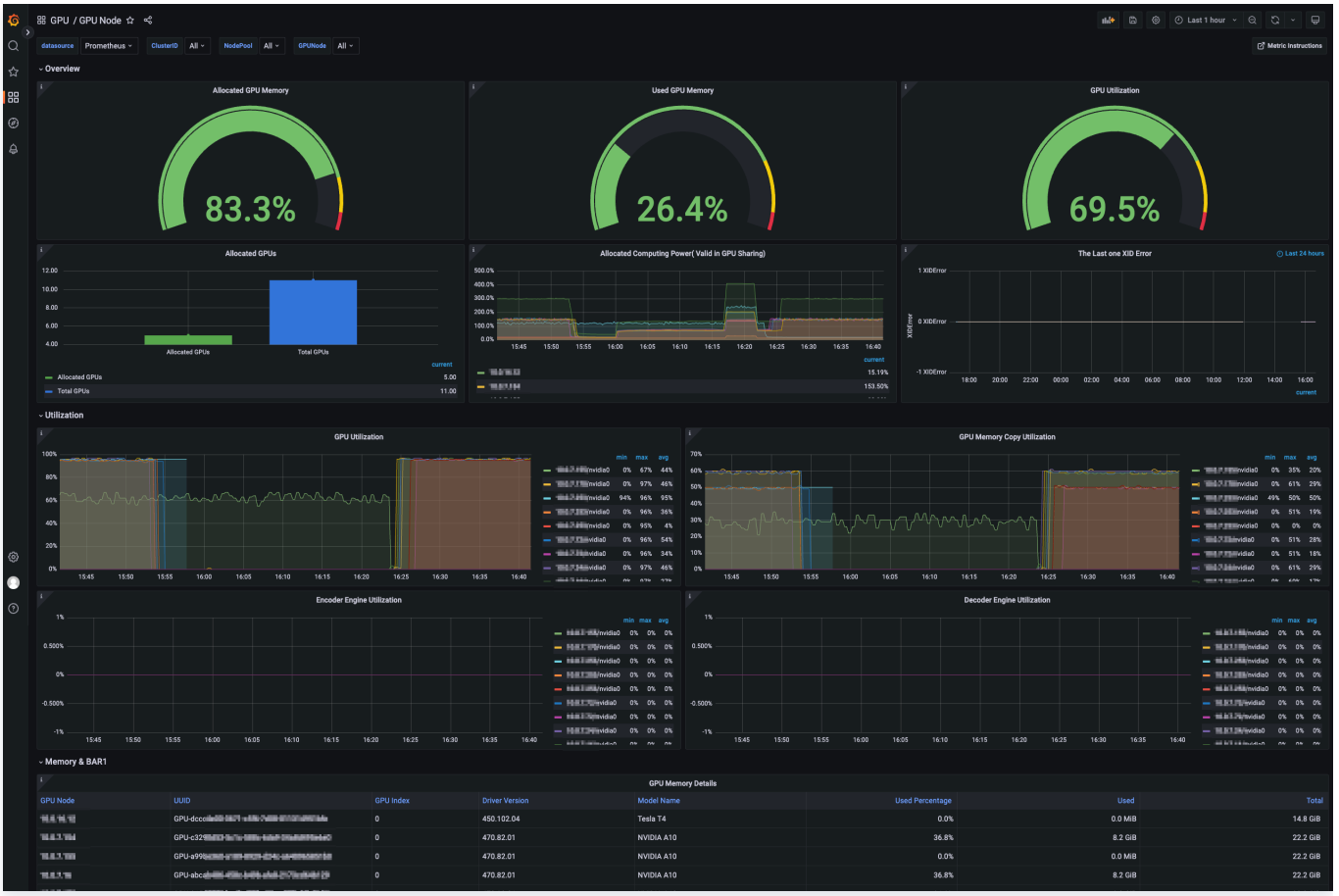
操作步骤

登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。

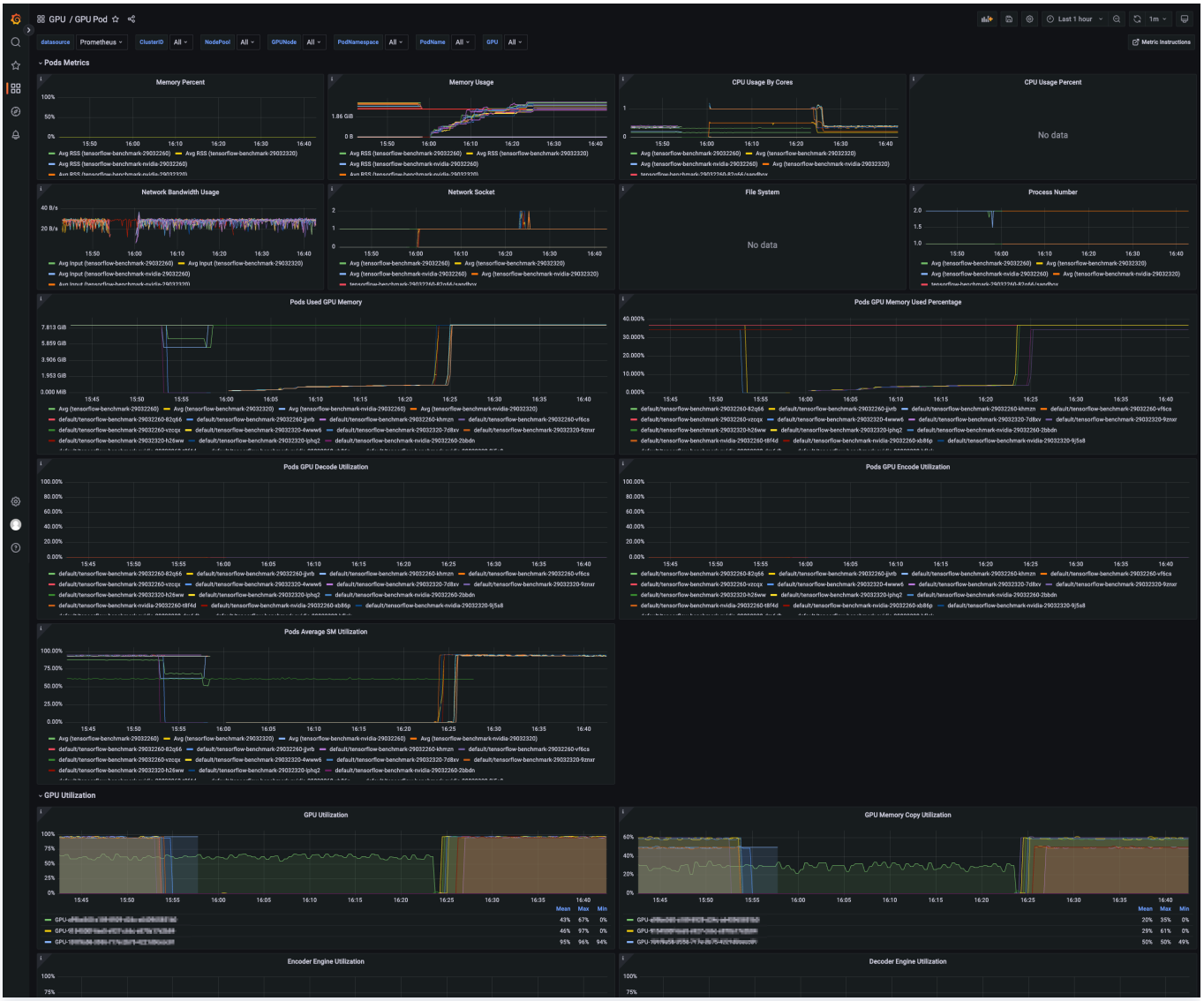
1. 选择**数据采集** > **集成中心**，进入集成中心页面。找到 **TKE GPU Exporter** 集成，选择 **Dashboard** > **Dashboard 操作** > **安装/升级 Dashboard**，单击**安装/升级**，安装对应的 Grafana Dashboard。
2. 选择查看已集成，在已集成列表中单击 Grafana 图标即可自动进入 GPU 监控大盘文件夹，查看实例相关的监控数据，如下图所示：
   - GPU Cluster：以集群的维度查看 GPU 节点状态，例如节点个数、GPU 使用率、GPU 内存、GPU 内存使用率等。



   - GPU Node：以节点池的维度查看 GPU 节点状态，例如节点状态概览、GPU 卡信息、内存使用率、温度与能耗信息等。



- GPU Pod：以 pod 的维度查看 GPU 节点状态，例如使用 GPU 节点的 pod CPU 内存利用率、网络带宽与文件系统监控、GPU 使用率、GPU 内存使用率等。



配置告警

腾讯云 Prometheus 托管服务支持告警配置，在集成中心页面找到 TKE GPU Exporter 集成，选择**告警 > 告警模板**，单击**展开规则列表**，可查看 TKE GPU 预设告警策略。

单击配置，选择通知模板并保存即可将预设 GPU 告警策略添加至当前 Prometheus 实例。同时也可根据业务实际的情况来添加自定义告警策略。详情请参见 新建告警策略 。


配置告警
策略名称
TKE GPU Exporter
收敛时间
5分钟
告警收敛时间对alertmanager渠道不生效。在收敛时间周期内若多次满足告警条件，仅会发送一次通知，若设置为1小时，则1小时内该策略被触发后仅会发送1次告警通知。
告警渠道
腾讯云
Webhook
自建alertmanager
告警通知
选择模板
新建
已选择 0 个通知模板，还可以选择 3 个
通知模板名称
包含操作
操作
当前通知模板列表为空，您可以选择相应的通知模板
请选择告警通知模板
保存
取消

Memcached Exporter 接入

最近更新时间：2024-11-04 09:39:02

操作场景

在使用 Memcached 过程中需要对 Memcached 运行状态进行监控，以便了解 Memcached 服务是否运行正常，排查 Memcached 故障等。Prometheus 监控服务提供基于 Exporter 的方式来监控 Memcached 运行状态，并提供了开箱即用的 Grafana 监控大盘。本文为您介绍如何使用 Prometheus 监控服务 Memcached。

操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心选择 **Memcached** 单击安装进行集成。

配置说明

名称	描述
名称	每个集成需要一个唯一名称
地址	要采集的 Memcached 实例的地址和端口
标签	增加具有业务含义的标签，会自动添加到 Prometheus 的 Label 中

查看监控

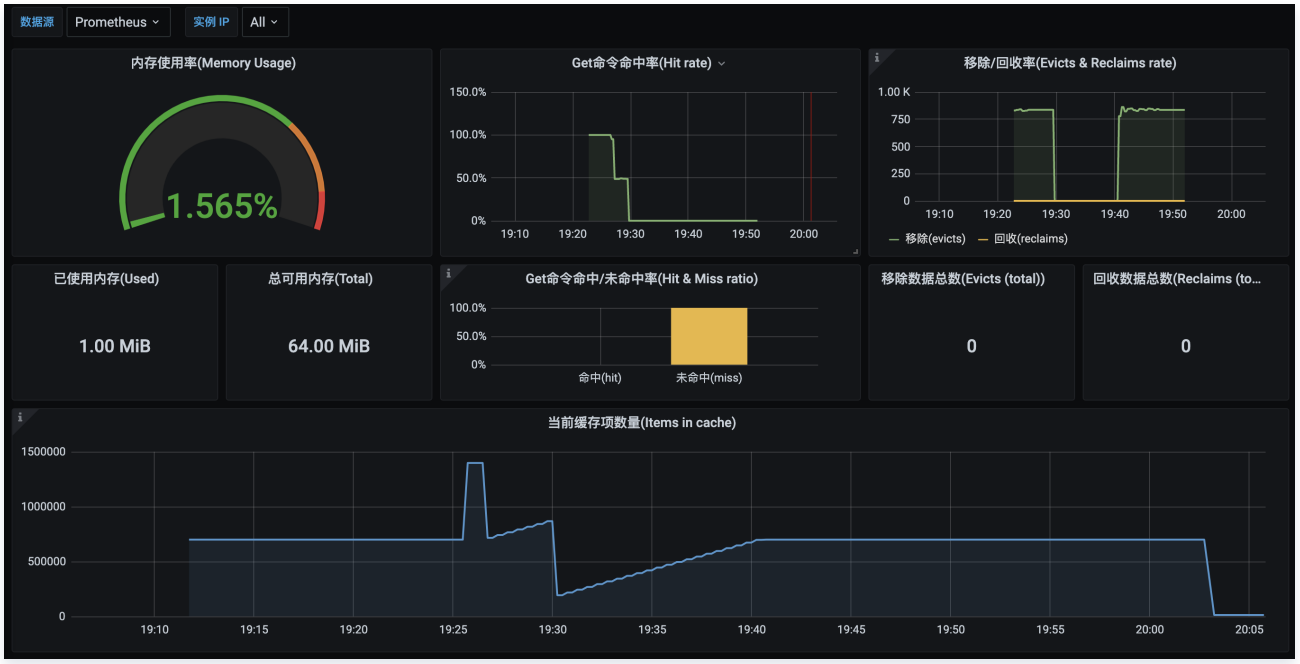
前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，在集成中心页面找到 Memcached 监控，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可通过监控大盘清晰看到如下监控状态：
   - 内存使用率，同时显示已使用的内存和内存总量。
   - 当前的 Get 命令的命中率，同时显示服务运行期间 Get 命令命中和未命中的比例。

- Memcached 移除旧数据和回收过期数据的速率，同时显示服务运行期间移除和回收的数据总数。
- Memcached 存储的数据总量。
- 从网络中读取和写入的字节数。
- 当前打开的连接数。
- 服务运行期间 Get 命令和 Set 命令的比例。
- 当前各命令的产生速率。



Apache Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

Apache Exporter 是一种用于收集和公开 Apache HTTP 服务器指标的工具，Exporter 上报的核心指标，用于异常报警和监控大盘展示。腾讯云可观测平台 Prometheus 提供了 Apache Exporter 集成与开箱即用的 Grafana 监控大盘。

> **说明：**
> 为了保证 Exporter 能够采集数据，需要确保 Apache 服务已打开 服务状态 相关内容。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **Apache**，找到后单击它即会弹出一个安装窗口。
5. 在弹出窗口的安装页面，填写指标采集名称、地址、路径等信息，并单击**保存**。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。

	• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
地址	Apache HTTP 服务的连接地址。
路径	Apache HTTP 服务的服务状态路径，默认为 `/server-status`
用户名	Apache HTTP 服务的用户名称。
密码	Apache HTTP 服务的密码。
标签	给指标添加自定义 Label。

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。
- 在 **Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务**中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

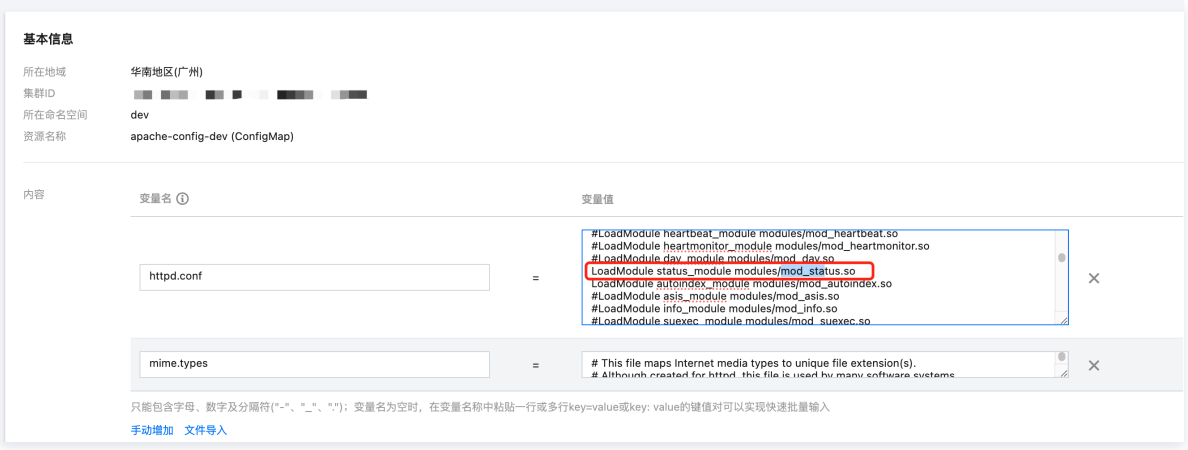
操作步骤

步骤一：Apache 服务开启 mod_status 模块

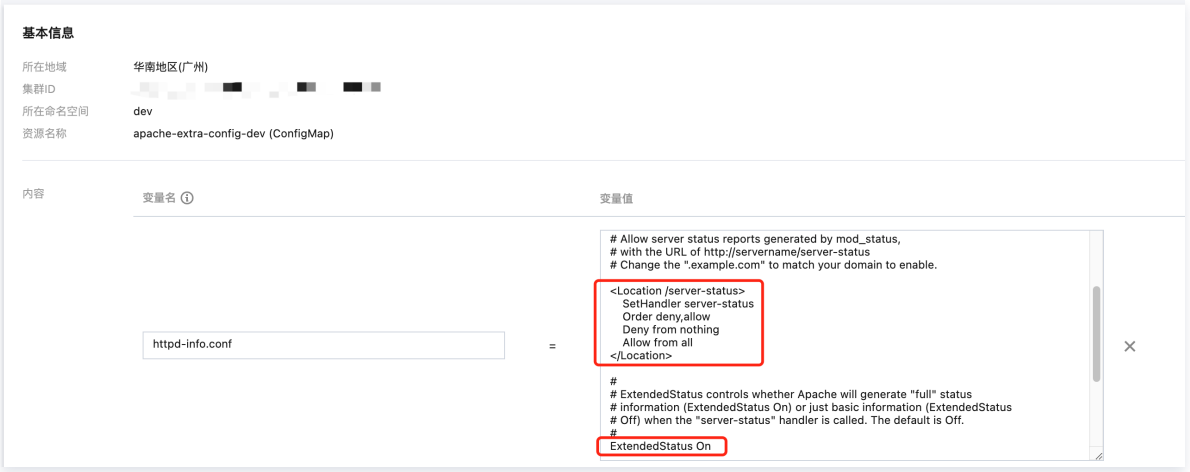
> **说明：**
> 容器服务相关操作可参见 容器服务 相关文档。

因为 Apache Exporter 是通过 Apache 的 mod_status 模块对其进行监控，所以需要确保 Apache 服务打开了 mod_status 模块，具体步骤如下：

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**，找到业务 Apache 服务所在集群，进入集群，找到业务 Apache 服务。
3. 若未添加 Apache 服务相关 ConfigMap，登录到业务 Apache 服务，将配置目录下的 httpd.conf、mime.types、extra/httpd-info.conf 等配置信息进行拷贝，创建 ConfigMap，将配置信息添加到该 ConfigMap 中，ConfigMap 相关操作指引见 ConfigMap 管理 。
4. 在 httpd.conf 中将这一行 LoadModule status_module modules/mod_status.so 注释去掉（即去掉最前面的#）。若服务存在 extra 相关配置，则在 httpd.conf 中放开相应配置，方法为去掉相关配置的注释。示例如下图所示：



5. 修改 httpd-info.conf 为业务需要的规则，并放开 ExtendedStatus，若不存在 extra 相关配置，则在 httpd.conf 中修改即可。示例如下图所示：



6. 验证mod_status模块是否开放成功，访问服务的 /server-status 相关接口，若正常返回数据即开放成功。示例如下图所示：

```
# curl  /server-status?auto
ServerVersion: Apache/2.4.53 (Unix) OpenSSL/1.1.1o
ServerMPM: event
Server Built: May 24 2022 19:22:43
CurrentTime: Friday, 29-Mar-2024 03:27:56
RestartTime: Thursday, 28-Mar-2024 07:16:19
ParentServerConfigGeneration: 1
ParentServerMPMGeneration: 0
ServerUptimeSeconds: 72696
ServerUptime: 20 hours 11 minutes 36 seconds
Load1: 6.14
Load5: 3.92
Load15: 3.31
Total Accesses: 4891
Total kBytes: 7363
Total Duration: 6526
```

步骤二：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 部署 Apache Exporter > 验证 步骤完成 Exporter 部署。

步骤三：部署 Apache Exporter

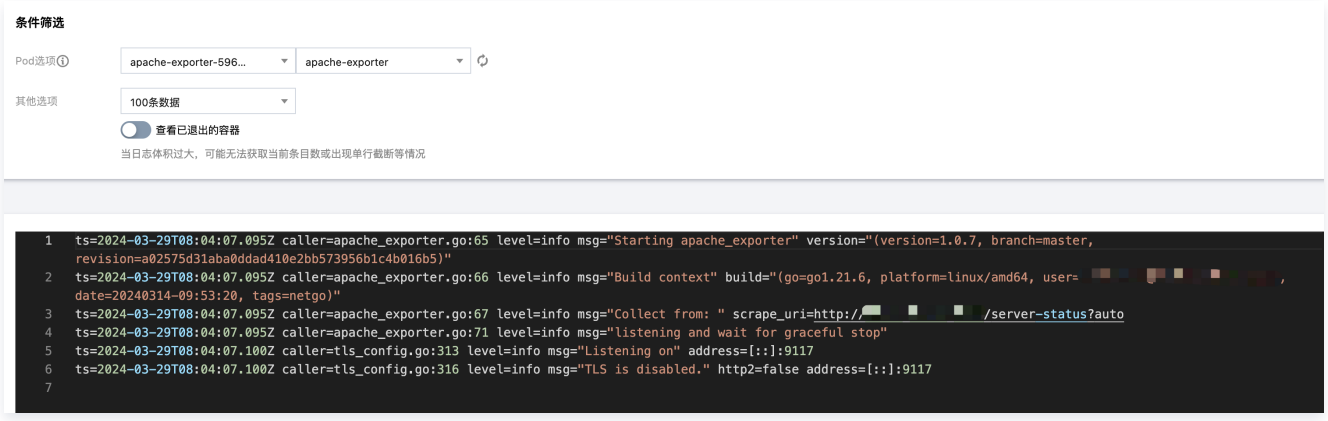
1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
  name: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
  namespace: apache-demo  # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      k8s-app: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
  template:
    metadata:
      labels:
        k8s-app: apache-exporter  # 根据业务需要调整成对应的名称，建议加上 Apache 实例的信息
    spec:
      containers:
      - args:
        - --web.listen-address=:9117
        - --scrape_uri=http://192.1.1.2:8080/server-status?auto # 根据业务需要调整成 Apache 实例对应地址
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:apache-exporter-v1.0.7
        name: apache-exporter
        ports:
        - containerPort: 9117
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

步骤四：验证

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



3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 Apache 指标。如发现未能得到对应的数据，请检查**连接串**是否正确，具体如下：

```
curl localhost:9117/metrics
```

执行结果如下图所示：

```
# HELP apache_accesses_total Current total apache accesses (*)
# TYPE apache_accesses_total counter
apache_accesses_total 6031
# HELP apache_connections Apache connection statuses
# TYPE apache_connections gauge
apache_connections{state="closing"} 1
apache_connections{state="keepalive"} 0
apache_connections{state="total"} 1
apache_connections{state="writing"} 0
# HELP apache_cpu_time_ms_total Apache CPU time
# TYPE apache_cpu_time_ms_total counter
apache_cpu_time_ms_total{type="system"} 3670
apache_cpu_time_ms_total{type="user"} 4019.9999999999995
# HELP apache_cpuload The current percentage CPU used by each worker and in total by all workers combined (*)
# TYPE apache_cpuload gauge
apache_cpuload 0.00858278
# HELP apache_duration_ms_total Total duration of all registered requests in ms
# TYPE apache_duration_ms_total counter
apache_duration_ms_total 8188
# HELP apache_exporter_build_info A metric with a constant '1' value labeled by version, revision, branch, goversion from which apache_exporter was built, and the goos and goarch for the build.
# TYPE apache_exporter_build_info gauge
apache_exporter_build_info{branch="master",goarch="amd64",goos="linux",goversion="go1.21.6",revision="a02575d31aba0ddad410e2bb573956b1c4b016b5",tags="netgo",version="1.0.7"} 1
# HELP apache_generation Apache restart generation
# TYPE apache_generation gauge
apache_generation{type="config"} 1
apache_generation{type="mpm"} 0
# HELP apache_info Apache version information
# TYPE apache_info gauge
apache_info{mpm="event",version="Apache/2.4.53 (Unix) OpenSSL/1.1.1o"} 1
```

步骤五：添加采集任务

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: apache-exporter  # 填写一个唯一名称
  namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不要修改
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # 填写pod yaml中Prometheus Exporter对应的Port的Name
    path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
    relabelings:
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # 调整成对应的 Apache 实例信息
  namespaceSelector:    # 选择要监控pod所在的namespace
    matchNames:
    - apache-demo
  selector:  # 填写要监控pod的Label值，以定位目标pod
    matchLabels:
      k8s-app: apache-exporter
```

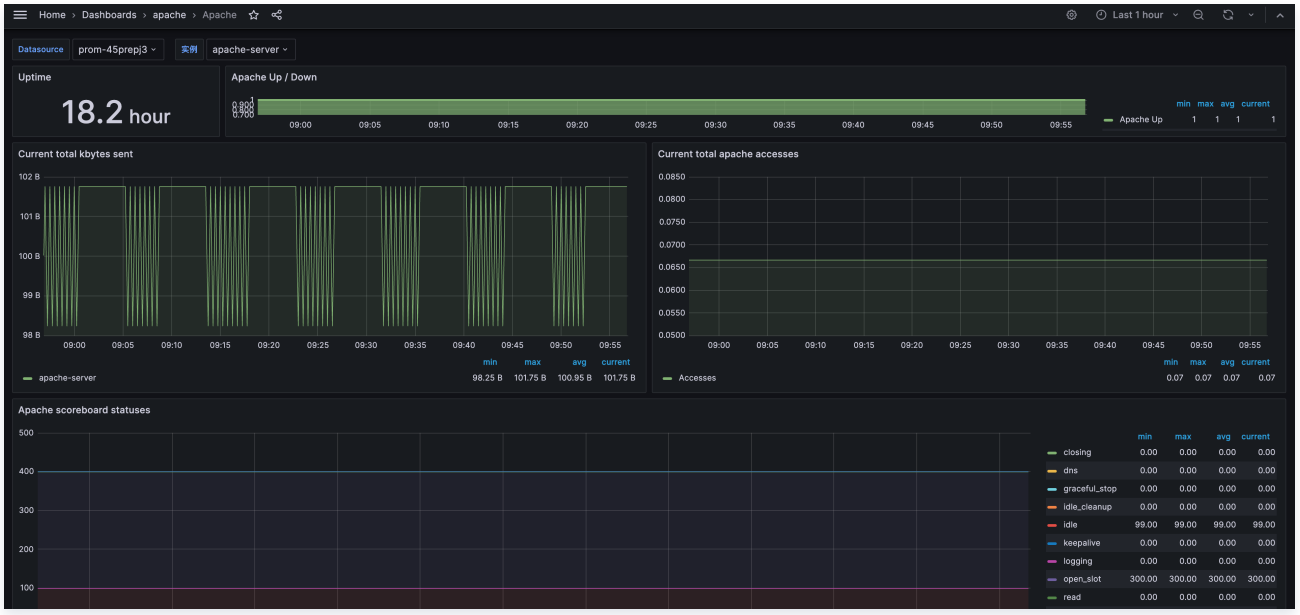
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

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



配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：Apache Exporter 采集参数说明

全局配置参数

名称	描述
telemetry.endpoint	指标暴露路径，默认 `/metrics` 。
scrape_uri	apache 服务状态页面 url，默认 `http://localhost/server-status/?auto` 。
host_override	覆盖 HTTP 主机标头，空字符串为没有覆盖。
[no-]insecure	如果使用 https，则忽略服务器证书。
custom_headers	将自定义标头添加到 Exporter。
[no-]web.systemd-socket	使用 systemd 套接字监听器代替端口监听器（仅限 Linux）。
web.listen-address	监听地址，默认：9117。
web.config.file	配置文件的路径，可以启用 TLS 或身份验证（实验性参数）。
log.level	日志级别，默认 info。
log.format	日志消息的输出格式，取值范围：[logfmt，json]，默认 logfmt。
version	打印版本信息。

Ceph Exporter 接入

最近更新时间：2024-12-09 18:42:52

操作场景

Ceph Exporter 是一个用于 Prometheus 监控系统的插件，用于收集和暴露 Ceph 分布式存储集群的性能指标。它允许用户监视 Ceph 集群的健康状况、性能指标和状态信息，帮助管理员及时发现问题并进行相应的调整和优化。腾讯云可观测平台 Prometheus 提供了与 Ceph Exporter 集成及开箱即用的 Grafana 监控大盘。
Ceph Luminous 12.2.1 之前的版本需要使用该 Exporter 导出指标；Ceph Luminous 12.2.1 的 mgr 中自带了 Prometheus 插件，内置了 Prometheus ceph exporter，可以使用 Ceph mgr 内置的 exporter 作为 Prometheus 的采集目标，故而 Ceph Luminous 12.2.1 及之后的版本更建议使用自带的指标导出。

接入方式：Mgr 原生导出

前提

Ceph 版本不低于 Luminous 12.2.1。

启用 Ceph 的 Prometheus 插件

查看是否已经开启：

```
ceph mgr module ls
```

若输出列表中包含 "prometheus" 则说明已开启 prometheus 插件，若不存在则需要开启该插件。
开启 prometheus 插件：

```
ceph mgr module enable prometheus
```

验证：

```
ceph mgr module ls
```

输出列表中包含 "prometheus"，说明已开启 prometheus 插件。然后登录 mgr pod 验证拉取指标：

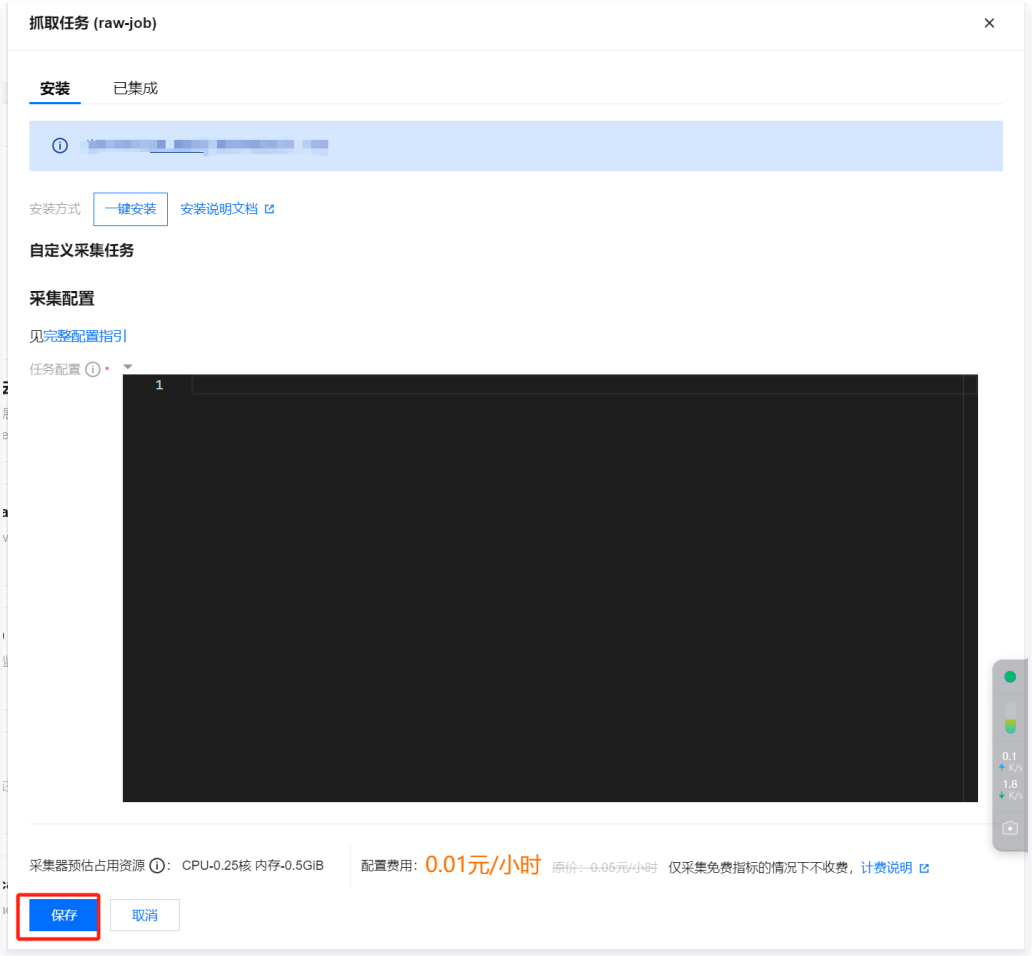
```
curl 127.0.0.1:9283/metrics ## 默认为9283端口，按业务实际端口进行调整
```

即可看到拉取的指标信息：

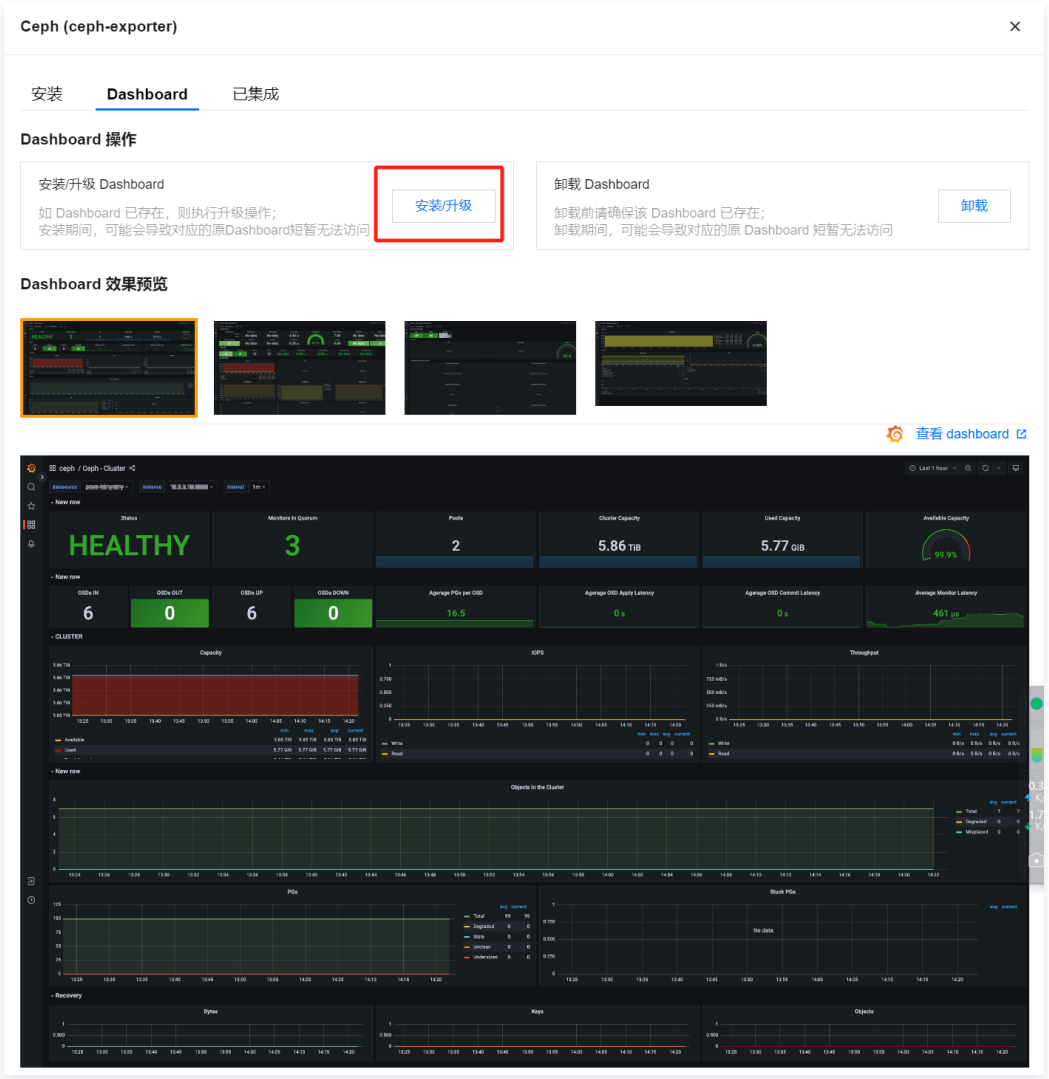
```
[root@roo[redacted]eph]# curl 127.0.0.1:9283/metrics

# HELP ceph_health_status Cluster health status
# TYPE ceph_health_status untyped
ceph_health_status 1.0
# HELP ceph_mon_quorum_status Monitors in quorum
# TYPE ceph_mon_quorum_status gauge
ceph_mon_quorum_status{ceph_daemon="mon.a"} 1.0
ceph_mon_quorum_status{ceph_daemon="mon.b"} 1.0
ceph_mon_quorum_status{ceph_daemon="mon.c"} 1.0
# HELP ceph_fs_metadata FS Metadata
# TYPE ceph_fs_metadata untyped
# HELP ceph_mds_metadata MDS Metadata
# TYPE ceph_mds_metadata untyped
# HELP ceph_mon_metadata MON Metadata
# TYPE ceph_mon_metadata untyped
ceph_mon_metadata{ceph_daemon="mon.a",hostname="172.[redacted]",public_addr="172.[redacted]",rank="0",ceph_version="ceph version 17.2.3 (dff484dfc9e19a9819f375586300b3b79d80034d) quincy (stable)"} 1.0
ceph_mon_metadata{ceph_daemon="mon.b",hostname="172.[redacted]",public_addr="172.[redacted]",rank="1",ceph_version="ceph version 17.2.3 (dff484dfc9e19a9819f375586300b3b79d80034d) quincy (stable)"} 1.0
ceph_mon_metadata{ceph_daemon="mon.c",hostname="172.[redacted]",public_addr="172.[redacted]",rank="2",ceph_version="ceph version 17.2.3 (dff484dfc9e19a9819f375586300b3b79d80034d) quincy (stable)"} 1.0
```

指标采集

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心找到并单击**抓取任务**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。



5. 在集成中心找到并单击 ceph，即会弹出一个窗口，选择 Dashboard > 安装/升级。



接入方式：开源 Exporter 导出

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **Ceph**，即会弹出一个安装窗口，在安装页面填写指标采集名称和地址等信息，并单击**保存**即可。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
用户名	Ceph 的用户名称。
秘钥	上述用户名对应的秘钥。
集群	Ceph 的集群名称。
标签	给指标添加自定义 Label。
ceph 配置	连接 Ceph 所需的 ceph.conf 配置文件

方式二：自定义安装

> **说明：**
> 为了方便安装管理 Exporter，推荐使用腾讯云 容器服务 来统一管理。

前提条件

- 在 Prometheus 实例对应地域及私有网络（VPC）下，创建腾讯云容器服务 Kubernetes 集群，并为集群创建 命名空间 。

- 在 Prometheus 监控服务控制台 > 选择对应的 Prometheus 实例 > 数据采集 > 集成容器服务中找到对应容器集群完成关联集群操作。可参见指引 关联集群 。

操作步骤

步骤一：Exporter 部署

1. 登录 容器服务控制台 。
2. 在左侧菜单栏中单击**集群**。
3. 单击需要获取集群访问凭证的集群 ID/名称，进入该集群的管理页面。
4. 执行以下 创建配置及鉴权用 Secret > 部署 Ceph Exporter > 验证 步骤完成 Exporter 部署。

步骤二：创建配置及鉴权用 Secret

1. 在左侧菜单中选择**工作负载** > **Deployment**，进入 Deployment 页面。
2. Base64编码生成秘钥环与 ceph.conf。

```
ceph.conf:
[global]
mon_host = 172.18.0.2:6789,172.18.0.3:6789,172.18.0.4:6789
## 该换行不可省略，是必需内容

keyring:
[client.admin]
key = AJKs8dsdfJkEEHxASDanasiKasfdJLYE5G3UAw==
## 该换行不可省略，是必需内容
```

Base64编码后：

```
ceph.conf:
W2dsb2JhbF0KbW9uX2hvc3QgPSAxNzIuMTguMC4yOjY3ODksMTcyLjE4LjAuMzo2Nzg5LDE3Mi4xOC4wLjQ6Njc4OQo=
keyring:
W2NsaWVudC5hZG1pbl0Ka2V5ID0gQUpLczhkc2RmSmtFRUh4QVNEYW5hc2lLYXNmZEpMWUU1RzNVQXc9PQo=
```

3. 在页面右上角单击 **YAML 创建资源**，创建 YAML 配置：

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret-test  # 根据业务需要调整成对应的名称
  namespace: ceph-demo  # 根据业务需要调整成对应的命名空间
type: Opaque
data:
  ## ceph.client.user.keyring 中的 user 为鉴权用的用户名，请根据实际业务进行替换，例如
ceph.client.admin.keyring
  ceph.client.user.keyring:
W2NsaWVudC5hZG1pbl0Ka2V5ID0gQUpLczhkc2RmSmtFRUh4QVNEYW5hc2lLYXNmZEpMWUU1RzNVQXc9PQo=
  ceph.conf:
W2dsb2JhbF0KbW9uX2hvc3QgPSAxNzIuMTguMC4yOjY3ODksMTcyLjE4LjAuMzo2Nzg5LDE3Mi4xOC4wLjQ6Njc4OQo=
```

步骤三：部署 Ceph Exporter

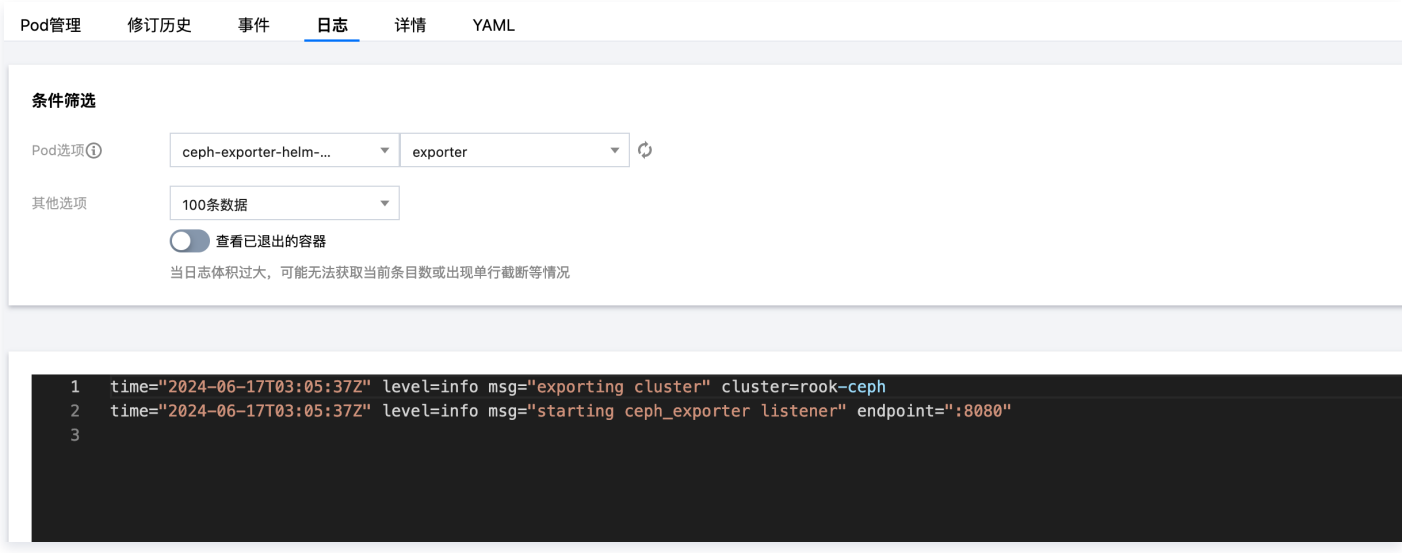
在 Deployment 管理页面，选择对应的命名空间来进行部署服务，可以通过控制台的方式创建。如下以 YAML 的方式部署 Exporter， 配置示例如下：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 Ceph 实例的信息
  name: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 Ceph 实例的信息
  namespace: ceph-demo # 根据业务需要调整成对应的命名空间
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 Ceph 实例的信息
  template:
    metadata:
      labels:
        k8s-app: ceph-exporter  # 根据业务需要调整成对应的名称，建议加上 ceph 实例的信息
    spec:
      containers:
      - env:
        - name: CEPH_CLUSTER
          value: ceph  # 根据业务需要调整成相应的集群名称
        - name: CEPH_USER
          value: admin # 根据业务需要调整成相应的用户名
        - name: TELEMETRY_ADDR
          value: :8080
        image: ccr.ccs.tencentyun.com/rig-agent/common-image:ceph-exporter-v4.2.3
        imagePullPolicy: IfNotPresent
        name: ceph-exporter
        ports:
        - containerPort: 8080
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /etc/ceph
          name: conf
          readOnly: true
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      volumes:
      - name: conf
        secret:
          defaultMode: 420
          secretName: ceph-secret-test  # 上面步骤中创建的 Secret 的名称
```

步骤四：验证

1. 在 Deployment 页面单击上述步骤创建的 Deployment，进入 Deployment 管理页面。

2. 单击**日志**页签，可以查看到 Exporter 成功启动并暴露对应的访问地址，如下图所示：



3. 单击 **Pod 管理**页签进入 Pod 页面。
4. 在右侧的操作项下单击**远程登录**，即可登录 Pod，在命令行窗口中执行以下 curl 命令对应 Exporter 暴露的地址，可以正常得到对应的 Ceph 指标。如发现未能得到对应的数据，请检查**秘钥**和 **ceph.conf** 配置是否正确，具体如下：

```
curl localhost:8080/metrics
```

执行结果如下图所示：

```
# HELP ceph_down_pgs No. of PGs in the cluster in down state
# TYPE ceph_down_pgs gauge
ceph_down_pgs{cluster="rook-ceph"} 0
# HELP ceph_features Counts of current client features, parsed from `ceph features`
# TYPE ceph_features gauge
ceph_features{cluster="rook-ceph",daemon="client",features="0x3f01cfbf7ffdffff",release="luminous"} 6
ceph_features{cluster="rook-ceph",daemon="mgr",features="0x3f01cfbf7ffdffff",release="luminous"} 2
ceph_features{cluster="rook-ceph",daemon="mon",features="0x3f01cfbf7ffdffff",release="luminous"} 3
# HELP ceph_forced_backfill_pgs No. of PGs in the cluster with forced_backfill state
# TYPE ceph_forced_backfill_pgs gauge
ceph_forced_backfill_pgs{cluster="rook-ceph"} 0
# HELP ceph_forced_recovery_pgs No. of PGs in the cluster with forced_recovery state
# TYPE ceph_forced_recovery_pgs gauge
ceph_forced_recovery_pgs{cluster="rook-ceph"} 0
# HELP ceph_health_status Health status of Cluster, can vary only between 3 states (err:2, warn:1, ok:0)
# TYPE ceph_health_status gauge
ceph_health_status{cluster="rook-ceph"} 1
# HELP ceph_health_status_interp Health status of Cluster, can vary only between 4 states (err:3, critical_warn:2, soft_warn:1, ok:0)
# TYPE ceph_health_status_interp gauge
ceph_health_status_interp{cluster="rook-ceph"} 1
# HELP ceph_incomplete_pgs No. of PGs in the cluster in incomplete state
# TYPE ceph_incomplete_pgs gauge
ceph_incomplete_pgs{cluster="rook-ceph"} 0
```

步骤五：添加采集任务

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成容器服务**，选择已经关联的集群，通过**数据采集配置 > 新建自定义监控 > YAML 编辑**来添加采集配置。
3. 通过服务发现添加 `PodMonitors` 来定义 Prometheus 抓取任务，YAML 配置示例如下：

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

```
    namespace: cm-prometheus  # 按量实例：集群的 namespace；包年包月实例（已停止售卖）：namespace 固定，不
要修改
  spec:
    podMetricsEndpoints:
    - interval: 30s
      port: metric-port     # 填写pod yaml中Prometheus Exporter对应的Port的Name
      path: /metrics  # 填写Prometheus Exporter对应的Path的值，不填默认/metrics
      relabelings:
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'crs-xxxxxx' # 调整成对应的 Ceph 实例 ID
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: ip
        replacement: '1.x.x.x' # 调整成对应的 Ceph 实例 IP
    namespaceSelector:    # 选择要监控pod所在的namespace
      matchNames:
      - ceph-demo
    selector:  # 填写要监控pod的Label值，以定位目标pod
      matchLabels:
        k8s-app: ceph-exporter
```

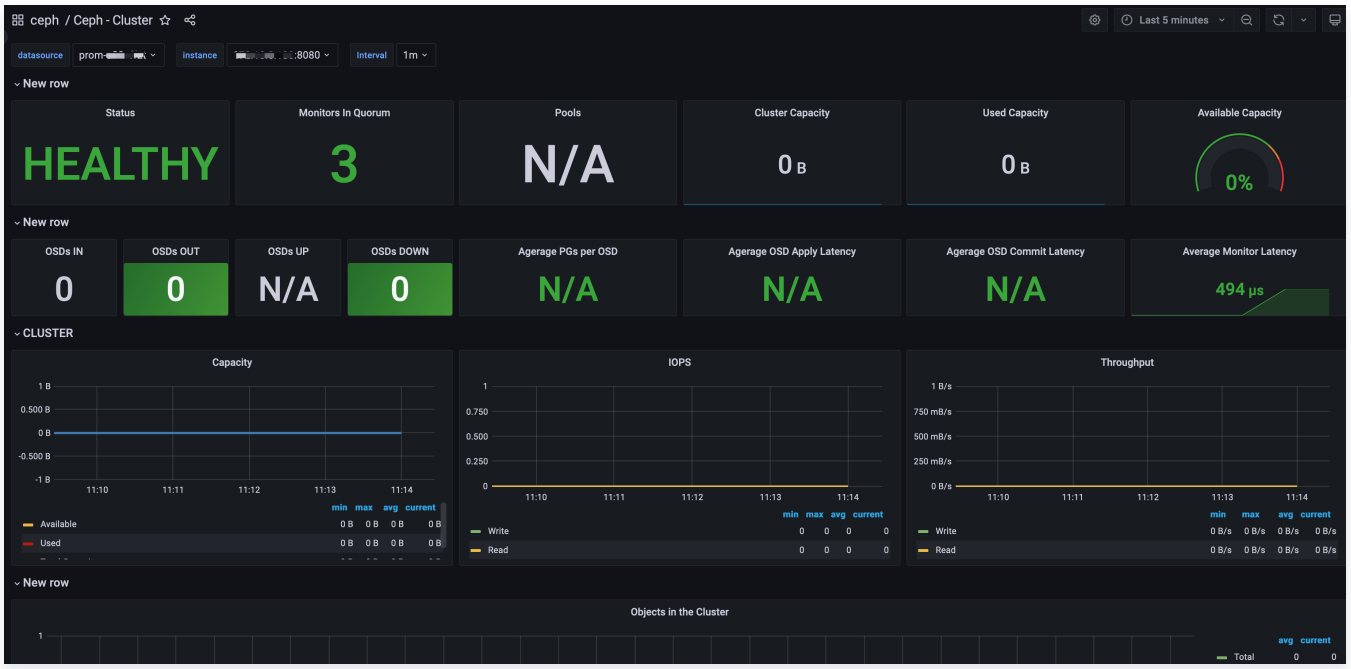
查看监控

前提条件

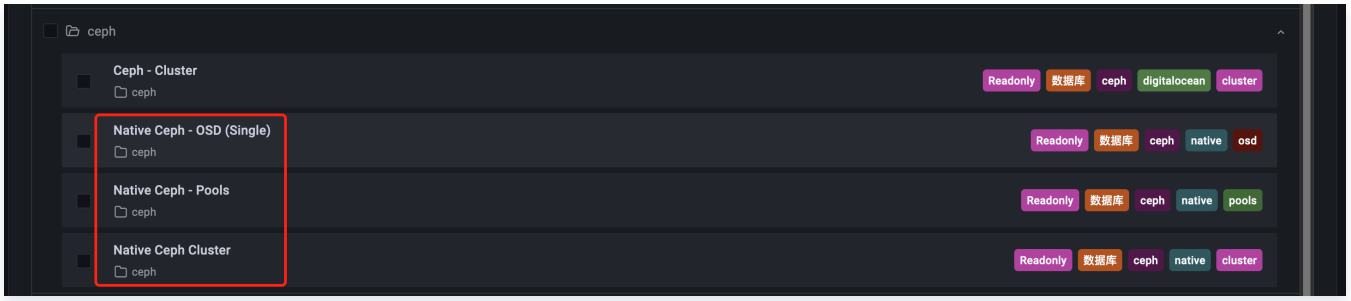
Prometheus 实例已绑定 Grafana 实例。

操作步骤

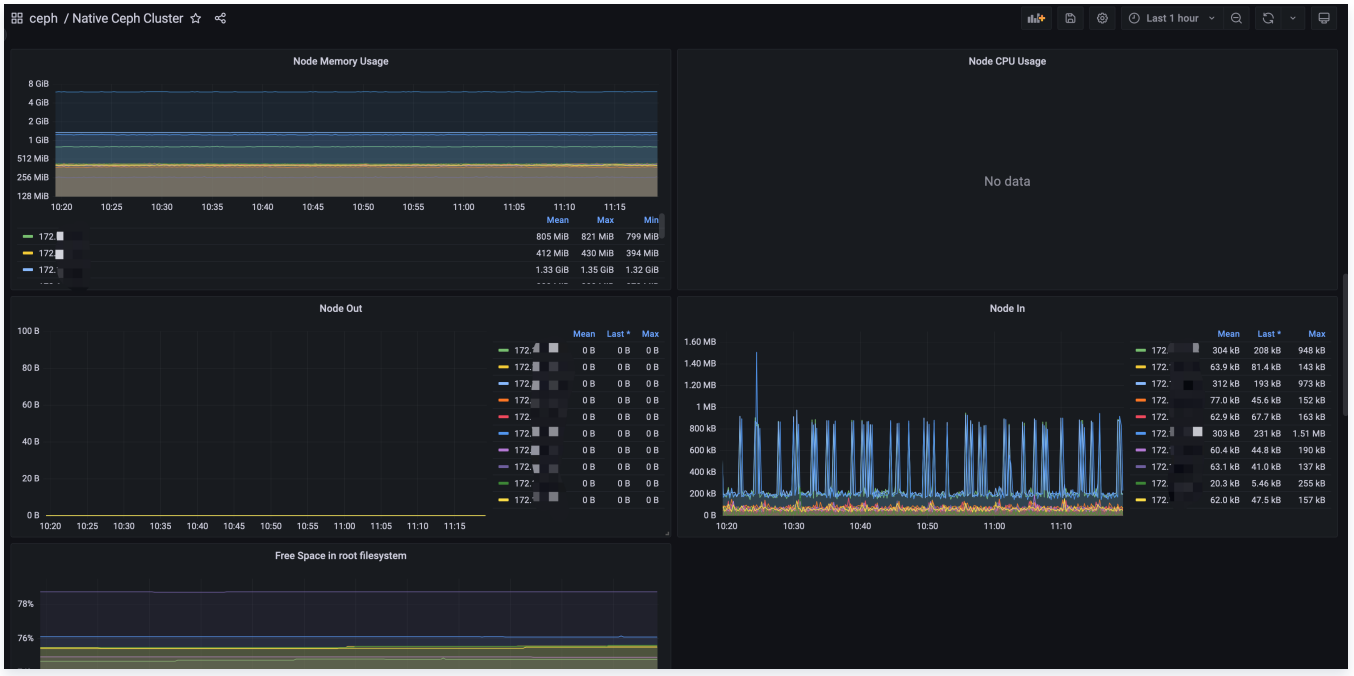
1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 Grafana 地址，打开并登录，然后在 ceph 文件夹中找到 Ceph - Cluster 监控面板，查看实例相关监控数据，如下图所示：



3. mgr 原生监控可在如下三个面板中查看：



4. Native Ceph Cluster 面板展示：



配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

附录：Ceph Exporter 环境变量配置

名称	描述
TELEMETRY_ADDR	暴露指标地址，默认 *:9128。
TELEMETRY_PATH	暴露指标路径，默认 /metrics。
EXPORTER_CONFIG	exporter 配置路径，默认 `/etc/ceph/exporter.yml`，存在该配置时，CEPH_CLUSTER、CEPH_CONFIG、CEPH_USER 等信息都将去其中获取。
RGW_MODE	启用从 RGW 收集统计数据，取值如下： • 0：禁用（默认值） • 1：启用 • 2：后台
CEPH_CLUSTER	Ceph 集群名称，默认 ceph。
CEPH_CONFIG	ceph.conf 配置位置，默认 `/etc/ceph/ceph.conf`。
CEPH_USER	Ceph 连接集群用户名，默认 admin。
CEPH_RADOS_OP_TIMEOUT	Ceph osd 与 mon 操作超时时长，默认30s。
LOG_LEVEL	日志等级，取值范围：[trace, debug, info, warn, error, fatal, panic]，默认 info。
TLS_CERT_FILE_PATH	tls 验证文件路径
TLS_KEY_FILE_PATH	tls 验证文件路径

其他 Exporter 接入

最近更新时间：2024-10-21 19:29:32

操作场景

Prometheus 监控服务目前已经提供了常用基础组件的集成方式，以及开箱即用的监控大屏，由于兼容原生 Prometheus，所以您也可以安装社区其他的 Exporter。

操作方式

如果您所使用的基础组件还没有提供相应的集成方式，可以参考如下方式进行集成，以及自定义监控大屏来满足相应的监控需求。

- 开源社区 Exporter 列表
- MySQL 的集成方式

健康巡检

最近更新时间：2024-11-08 11:01:22

操作场景

通过定期探测对应服务的连通性，来检测其服务的健康情况，帮助您实时的掌握服务的健康状况，及时发现异常，来提升服务的 SLA。

操作步骤

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 在实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，单击**数据采集 > 集成中心**。
5. 在集成中心选择**健康巡检** ，单击它即会弹出一个安装窗口。

探测说明

参数	说明
名称	每个探测任务需要一个唯一名称，对应 Grafana 监控大屏中的探测分组
探测方式	目前支持如下几种探测方式： • http_get

	• http_post • tcp • ssh • ping
探测目标	被探测服务对应地址
标签	增加具有业务含义的标签，会自动添加到 Prometheus 的 Label 中

查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 单击**数据采集 > 集成中心**，在集成中心页面找到**健康巡检**，选择 **Dashboard 操作 > Dashboard 安装/升级**来安装对应的 Grafana Dashboard。
3. 选择**查看已集成**，在已集成列表中点击 Grafana 图标即可通过监控大盘清晰看到如下监控状态：
   - 服务访问的延时，是否健康。
   - 服务访问各阶段处理的延时。
   - 如果是 HTTPS 可以监控证书的过期时间。
   - 以及各种探测类型的状态。

云监控

最近更新时间：2025-06-17 15:50:53

操作场景

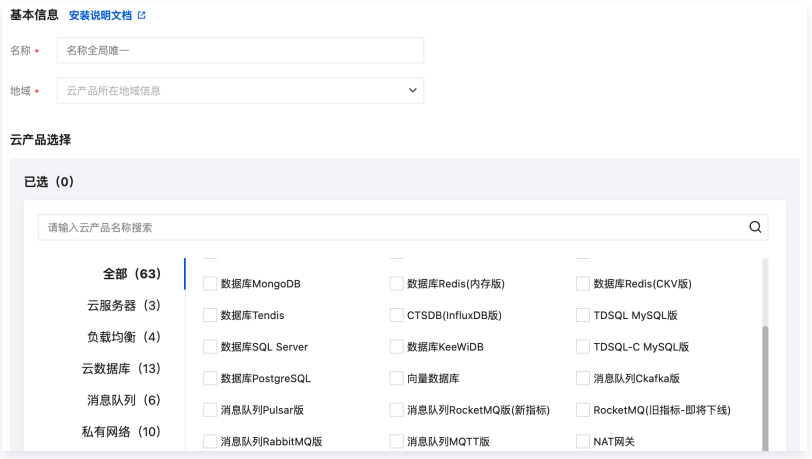
Prometheus 监控服务-云监控模块集成腾讯云产品基础监控数据，通过 Prometheus 监控进行统一采集、存储和可视化。

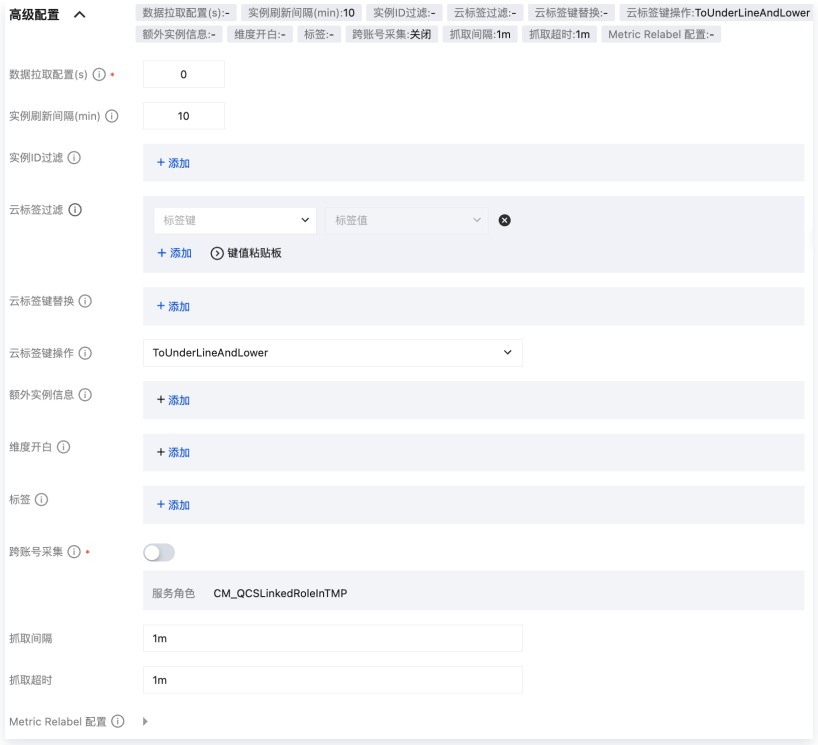
说明：

- 数据采集间隔：默认1分钟。目前不支持更小的采集间隔。
- 监控数据粒度：1分钟。如果指标不支持1分钟粒度，则选择5分钟粒度。
- 集成的监控数据包含云产品的标签数据（部分云产品不支持），标签键必须符合正则表达式 `[a-Za-Z_][a-Za-Z0-9_]*`，否则会被过滤。
- 不支持多地域。如果云产品分布在多个地域，需要安装多个集成。

操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心单击**云监控**，默认进入安装页面。定义集成名称、选择对应的云产品和进行 Exporter 配置。





配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
地域	必填，云产品所在地域。如果云产品不区分地域，则填写任意地域。
云产品选择	勾选想要采集的云产品。
数据拉取配置	单位秒。若设置为0，将忽略原始数据的时间戳；若设置大于0，将上报原始数据的时间戳，由于云产品监控数据上报到基础监控存在一定的延迟，该延迟将会体现在最新的数据上。 拉取数据范围：(当前时间 - 数据采集延迟 - 固定的时间间隔, 当前时间 - 数据采集延迟)。
实例刷新间隔	单位分钟，最小值为10。每隔一个实例刷新间隔，集成会重新拉取云产品实例信息。如果修改了实例名、云标签或者增删实例，会在一个实例刷新间隔内更新监控数据。
实例 ID 过滤	选填。不填默认采集主账号下所有实例的数据。键值对形式填写，键是集成定义的云产品 唯一 ID，值是逗号分隔的云产品实例 ID。填写键值对的云产品，只会采集填写的实例。
云标签过滤	选填。键值对形式填写，一个标签键可以对应多个标签值，以 | 分割。不同的标签键取交集，同一标签键下的多个标签值取并集。对于支持云标签过滤的产品，如果同时配置了实例 ID 过滤，该产品的云标签过滤将不会生效。
云标签键替换	选填。将不合法的云产品标签键替换为合法值，例如将中文名转换成自定义的英文名。
云标签键操作	集成默认将标签键的大写字母转换成下划线+小写字母。支持对云产品标签键的转换操作： • ToUnderLineAndLower：默认操作。 • ToLower：表示全转成小写字母。 • NoOperation：表示不做转换。
额外实例信息	添加额外的实例信息。Project 表示添加项目 ID 和项目名称标签，只有部分云产品支持。
维度开白	选填。部分云产品的维度存在指标名称相同、功能需要开白等问题，默认不采集，可通过该配置开启采集。 • lb_public:listener：负载均衡（公网）-监听器维度。 • lb_public:target：负载均衡（公网）-后端服务器维度。

	• lb_public:domain：负载均衡（公网）-转发规则域名维度。 • lb_private:listener：负载均衡（内网）-监听器维度。 • lb_private:target：负载均衡（内网）-后端服务器维度。 • lb_private:domain：负载均衡（内网）-转发规则域名维度。 • apigw_cloudnative:node：云原生 API 网关-节点维度。 • scf_v2:version：云函数-版本维度。 • vbc:qosid：云联网-调度队列维度。 • ces:node：Elasticsearch-节点维度。
标签	选填。可以给集成采集到的指标添加额外的自定义标签。
跨账号采集	开启后可填写 **跨账号采集** 配置。 • 本账号角色：自定义角色，用于获取本账号临时密钥。 • 目标账号角色：自定义角色，用于获取目标账号临时密钥。 • 目标账号 uin：目标账号的主账号 ID。
抓取间隔	最小1分钟。因为监控数据粒度是1分钟，低于1分钟的抓取间隔没有意义。填写示例：60s、1m30s、5m、1h30m。
抓取超时	抓取超时必须小于等于抓取间隔。填写方式同抓取间隔。
Metric Relabel 配置	选填。兼容 Prometheus 原生的 metric_relabel_configs 和 Prometheus Operator 原生的 metricRelabelings 配置。不可混用。

Metric Relabel 配置示例

下面是常用的 metric_relabel_configs 示例：

```
metric_relabel_configs:
- action: labeldrop  # 去掉名为 labelA 的 label。regex是正则表达式，多个正则表达式用 | 分隔
  regex: labelA
- regex: ins-(.*) # 新增一个名为 id 的 label，其值通过名为 instance_id 的 label 的值经过正则处理后得到。例如 instance_id="ins-a"，新得到的 id="a"
  replacement: $1
  source_labels:
  - instance_id
  target_label: id
- target_label: region # 新增一个 region="ap-guangzhou" 的 label
  replacement: ap-guangzhou
- action: drop # 去掉名为 metricA 或 metricB 的指标
  source_labels:
  - __name__
  regex: metricA|metricB
```

支持的云产品

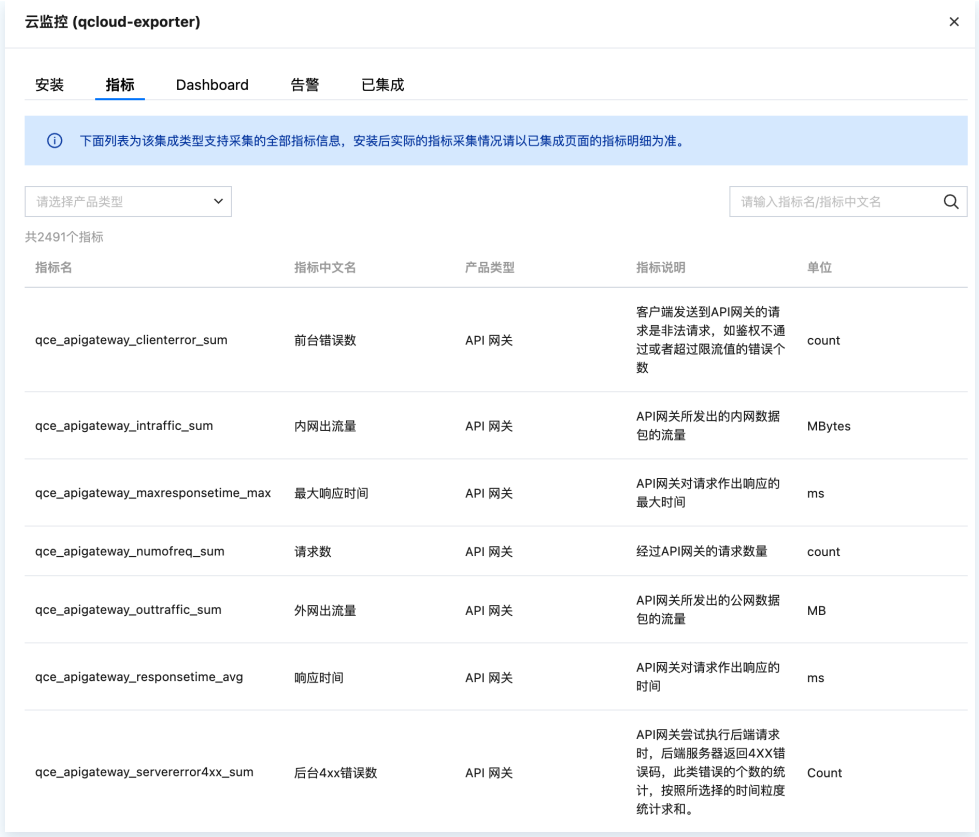
云产品/指标文档	是否支持采集云标签	唯一 ID	补充说明
云服务器	是	cvm	仅支持实例维度指标。
云服务器（内网）	是	sdn_vm	-
云硬盘	是	cbs	-
负载均衡（公网）	是	lb_public	默认采集实例维度指标，如需监听器、转发规则域名或后端服务器维度指标，请自行添加维度开白配置。不同维度的指标名相同，可以通过 monitor_view 标签来区分维度： • 实例维度：instance； • 监听器维度：listener；

			• 后端服务器维度：target； • 转发规则域名维度：domain。
负载均衡（内网）	是	lb_private	默认采集实例维度指标，如需监听器或转发规则域名维度指标，请自行添加维度开白配置。不同维度的指标名相同，可以通过 monitor_view 标签来区分维度： • 实例维度：instance； • 监听器维度：listener； • 后端服务器维度：target； • 转发规则域名维度：domain。
负载均衡（四层独占集群）	是	tgw_set	该产品是负载均衡开白产品，指标随时可能变化，不提供对外指标文档。
负载均衡（七层独占集群）	是	tgw_set_l7	该产品是负载均衡开白产品，指标随时可能变化，不提供对外指标文档。
云数据库 MongoDB	是	cmongo	-
云数据库 MySQL（CDB）	是	cdb	-
云数据库 Redis（CKV 版）	是	redis	-
云数据库 Redis（内存版）	是	redis_mem	支持实例维度和节点维度指标。
云数据库 Tendis	是	tendis	-
CTSDB（InfluxDB 版）	是	xstor	该产品是 CTSDB 开白产品，指标随时可能变化，不提供对外指标文档。
云数据库 MariaDB	是	mariadb	仅支持实例维度指标。
云数据库 PostgreSQL	是	postgres	-
TDSQL MySQL 版	是	tdmysql	仅支持实例维度指标。
TDSQL-C MySQL 版	是	cynosdb_mysql	仅支持实例维度指标。
云数据库 SQL Server	是	sqlserver	仅支持实例维度指标。
云数据库 KeeWiDB	是	keewidb	-
向量数据库	是	vecdb	-
NAT 网关	是	nat_gateway	-
NAT 实例监控丢包率	是	vpc_gw_detect	NAT 网关指标。
消息队列 Ckafka 版	是	ckafka	不支持 broker_ip 维度指标。
消息队列 Pulsar 版	是	tdmq_pulsar	-
消息队列 RocketMQ 版	是	rocketmq	指标文档和唯一 ID 对应新指标，使用旧指标的用户建议尽快切换。
消息队列 RabbitMQ 版	是	rabbitmq	-
消息队列 MQTT 版	是	mqtt	-
弹性公网 IP	是	lb	-
VPN 网关	是	vpngw	-

VPN 通道	是	vpnx	-
网络探测	不支持标签	vpc_net_detect	-
私有网络-跨可用区流量	是	sdn_az	指标不支持1分钟粒度，默认拉取5分钟粒度数据。
私有网络-私有连接	是	private_link	-
CDN（国内域名）	是	cdn	不区分地域。仅支持域名维度指标。
CDN（国外域名）	是	ov_cdn	不区分地域。
COS	是	cos	存储相关指标延迟过高（2小时左右），不会保留数据的原始时间戳。存储相关指标不支持1分钟粒度，默认拉取5分钟粒度数据。
专线接入-物理专线	是	dc	不区分地域。
专线接入-专用通道	是	dcx	不区分地域。
专线接入-专线网关	是	dcg	同私有网络/网络连接/专线网关。
轻量应用服务器	是	lighthouse	-
云原生 API 网关	是	apigw_cloudnative	默认采集实例维度和公网负载均衡维度指标，如需节点维度指标，请自行添加维度开白配置。实例维度与节点维度的指标名相同，可以通过 monitor_view 标签来区分维度： • 实例维度：gateway； • 公网负载均衡维度：loadbalancer； • 节点维度：node。
Nacos	是	nacos	-
Zookeeper	是	zookeeper	-
Elasticsearch	是	ces	默认采集实例维度指标，如需节点维度指标，请自行添加维度开白配置。可以通过 monitor_view 标签来区分维度： • 实例维度：instance； • 节点维度：node。
流计算 Oceanus	是	tstream	需要在 流计算控制台 > **成员管理**中将服务角色 CM_QCSLinkedRoleInTMP 设为访客角色。
数据湖计算 DLC	是	dlc	支持内部存储桶、Spark 引擎、Presto 引擎、网关 ID、Spark 作业、引擎维度指标。可以通过 monitor_view 标签来区分维度： • 内部存储桶：bucket • Spark 引擎：spark_engine • Presto 引擎：presto_engine • 网关 ID：gateway • Spark 作业：spark_job • 引擎（包括 Spark 和 Presto 引擎）：engine
腾讯云数据仓库 TCHouse-C	是	cdwch	-
腾讯云数据仓库 TCHouse-D	是	cdwdrs	-
数据传输服务	是	dts	不支持 Kafka 相关维度指标。

云联网	是	vbc	如需调度队列维度指标，请自行添加维度开白配置。
全球应用加速	是	gaap	-
EdgeOne（七层）	是	edgeone_l7	仅支持【子域名-站点】维度指标。
Web 应用防火墙	是	waf	-
文件存储	是	cfs	目前未采集元数据、快照相关指标。
数据加速器 GooseFSx	是	goosefsx	-
共享带宽包	是	bwp	-
云函数	是	scf_v2	默认采集别名维度指标，如需版本维度指标，请自行添加维度开白配置。别名维度与版本维度的指标名相同，可以通过 monitor_view 标签来区分维度： • 别名维度：alias。 • 版本维度：version。
云点播（VOD）	是	vod	不区分地域。目前 FluxHitRate、RequestsHitRate、BackOriginBandwidth、BackOriginFlux 四个指标未对外，无法采集。
日志服务（CLS）-日志主题	是	cls	-
数据万象	是	ci	-
API 网关	是	apigateway	仅支持 API 维度指标。
TI-ONE（任务式建模）	是	ti_traintask	-
TI-ONE（Notebook）	是	ti_notebook	-
TI-ONE（在线服务）	是	ti_model	-

说明：
为了区分不同云产品的指标，云监控集成对云产品指标名（指标文档中的指标英文名）做了转换。指标页中提供了云监控集成支持采集的指标信息，方便用户直接查看和使用。



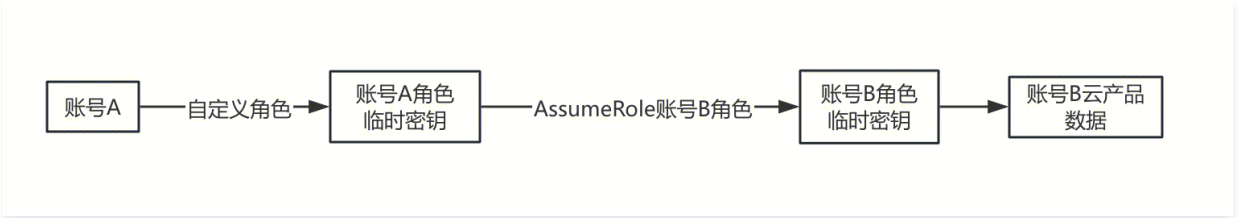
跨账号采集

> **注意：**
> 不支持跨站采集（国内站账号与国际站账号不能互相采集）。

场景：账号 A 跨账号采集 账号 B 的监控数据。
配置填写：

- 在账号 A 下的 Prometheus 监控服务实例中创建云监控集成。
- 开启**跨账号采集**。
- **本账号角色**选择账号 A 创建的自定义角色。
- **目标账号角色**填入账号 B 创建的自定义角色。
- **目标账号 uin** 填入账号 B 的主账号 ID。

简要流程图



自定义角色

账号 A 创建自定义角色

1. 在 策略 页面，通过策略语法创建 自定义策略，添加 sts:AssumeRole 权限，该权限用于扮演账号 B 的角色。策略语法如下：

```
{
 "version": "2.0",
 "statement": [
 {
     "effect": "allow",
     "action": ["sts:AssumeRole"],
     "resource": ["*"]
 }
 ]
}
```

> ⚠ **注意：**
> 如果需要限制权限，例如只能扮演账号 B 的自定义角色，可以将 resource 修改为 "qcs::cam::uin/[账号 B 主账号 ID]:roleName/[账号 B 自定义角色]"。

2. 在 角色 列表页面，单击**新建角色**。
3. 在弹出的选择角色载体窗口，选择**腾讯云产品服务**，进入角色信息填写页面。
4. 勾选**云服务器（cvm）**作为角色载体，使用案例选择**云服务器**，单击**下一步**。
5. 在策略列表内，勾选第1步创建的策略为角色配置策略，单击**下一步**。
6. 标记角色的标签键和标签值，可不填，单击**下一步**。
7. 输入您的角色名称，单击**完成**后即完成自定义角色创建。

账号 B 创建自定义角色

1. 在角色列表页面，单击**新建角色**。
2. 在弹出的选择角色载体信息窗口，选择**腾讯云账户**作为角色载体，进入角色信息填写页面。
3. 在输入角色载体信息页面，**云账号类型**选择**其他主账号**，**账号 ID** 填写账号 A 主账号 ID，其它可不填，单击**下一步**。
4. 在策略列表内，勾选预设策略 **ReadOnlyAccess** 为角色配置策略，单击**下一步**。
5. 标记角色的标签键和标签值，可不填，单击**下一步**。
6. 输入您的角色名称，单击**完成**后即完成自定义角色创建。

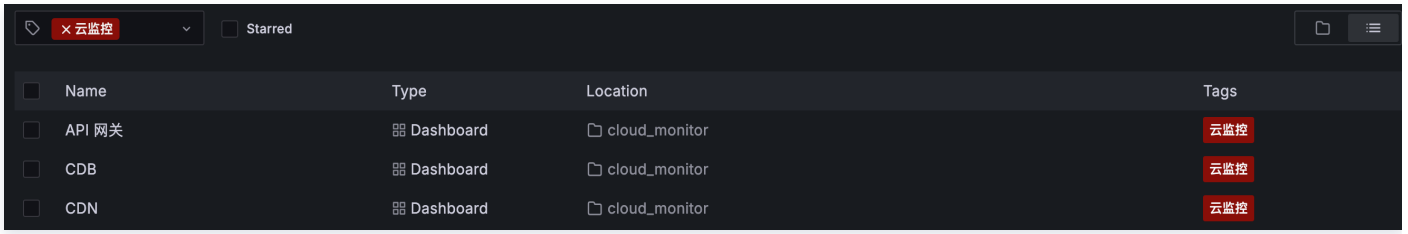
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，在集成中心页面，找到并单击**云监控**，选择 **Dashboard > Dashboard 操作**下的**安装/升级 Dashboard**，单击**安装/升级**安装对应的 Grafana Dashboard。
3. 选择**已集成**，在已集成列表中单击 Grafana 图标即可自动打开云监控集成大盘列表，选择对应云产品大盘，查看实例相关的监控数据，如下图所示：



常见问题

“数据拉取配置”该怎么配置?

- 若配置为0，Prometheus 会使用当前时间戳，覆盖数据的原始时间戳。

  使用场景：保证数据时间戳的实时性，以最大限度保证 Prometheus 能及时发出告警。
- 若配置为某个大于0的值 x：
  - 只要是大于0的值，Prometheus 就会保留数据的原始时间戳。

    使用场景：与云产品控制台监控页的时间戳保持一致。
  - 延迟拉取数据的时间窗口（延迟量等于 x）。

    问题背景：为了兼容云产品监控数据上报链路的时延，Prometheus 默认以 `(now-固定时延，now)` 的时间范围拉取数据。

    使用场景：若个别产品上报链路时延过大，此处需设置 x，使得拉取数据的时间范围变为：`(now-固定时延-x，now-x)`，以保证在这个延迟的窗口内，能更大限度地拉取到数据。

数据偶尔会产生1-2分钟的断点?

- **数据拉取配置**为0：一般不会产生断点。产生断点时请 提交工单 。
- **数据拉取配置**非0：查看集成日志，如果没有明显报错，则可能是延迟波动导致的断点。此时可观察指标正常时的延迟，将**数据拉取配置**重新设置为该延迟大小，例如3分钟的延迟，就设置为180，如果断点情况未能改善，请 提交工单 。

> ⚠ **注意：**
> - 重新设置**数据拉取配置**会影响当前集成中的所有云产品，如果只是个别云产品数据有断点，建议单独新建一个集成。
> - **延迟波动**：部分云产品监控数据，其延迟并不是稳定的。当延迟突然变低时，意味着一分钟内落盘了多个数据点，而云监控集成每一分钟只会采集最新的数据点，这就会导致断点。

Targets 显示有问题?

- 无采集对象：刚创建的集成需要等待几分钟才能展示正确的 targets。
- (1/2)down：集成采用滚动更新，在新 pod 成功运行之前会继续采集旧 pod，期间就会显示两个 targets。

某个云产品没采集到指标?

1. 在**已集成**下，查看如下信息：
   - 查看**实例信息**是否含有该云产品，没有则说明未勾选该云产品。
   - 确定 Targets 是 up 状态。
   - 查看**指标明细**中是否有该云产品指标，若有则等待一分钟后再查询。

2. 确定所选地域下有该云产品实例。
3. 查看是否配置了**实例 ID 过滤**或**云标签过滤**，确定对应配置能获取到该云产品实例。
4. 查看是否配置了 Metric Relabel 配置，确定对应配置没有过滤该云产品指标。

如何重启集成/更新集成版本

1. 在**已集成**下，单击需要操作的集成名称，进入集成编辑页。

2. 直接单击**保存**

更新动态

云监控集成每次保存，都会将集成更新为最新版本。下面是集成每次版本更新的主要时间节点与内容，可以用来评估集成更新的影响。

时间	更新内容
2025年6月	修复消息队列 RocketMQ 版在特殊情况下不采集部分 topic 监控数据的问题。
2025年5月	Elasticsearch 支持节点维度指标，新增标签 monitor_view；TI-ONE 新增 gpu、gpu_type、app_id、sub_uin、sub_uin_name 标签。 注意：gpu 表示 GPU 卡资源，单位为1单位的 gpu_type；gpu_type 表示 GPU 类型；存量 Elasticsearch 和 TI-ONE 实例更新后会产生新时间线。
2025年4月	支持数据湖计算 DLC、数据加速器 GooseFSx；负载均衡（内网）支持后端服务器维度、清理重复标签 vpcid、loadbalancerid、loadbalancerport；云服务器新增标签 gpu_type，仅影响 GPU 云服务器。 注意：gpu_type 表示 GPU 类型，存量 GPU 云服务器实例更新后会产生新时间线；存量负载均衡（内网）实例更新后会产生新时间线。
2025年3月	支持 NAT 实例监控丢包率；云服务器支持 HCCPNV6e、HCCPNV5b、HCCSA4机型的 vRDMA 指标；云联网支持网络实例维度。
2025年2月	云数据库 Tendis 新增标签 node_zone_id。 注意：node_zone_id 表示节点可用区 ID，存量实例更新后会产生新时间线。
2025年1月	支持数据万象、消息队列 MQTT 版；文件存储支持 Turbo 文件系统指标。 注意：文件系统新增标签 protocol，表示文件系统协议，存量实例更新后会产生新时间线。
2024年12月	云联网支持服务等级维度和调度队列维度。
2024年11月	支持流计算 Oceanus、腾讯云数据仓库 TCHouse-C。
2024年10月	云数据库 MongoDB 支持 mongos 节点维度。
2024年9月	支持向量数据库；云产品全量支持采集云标签和云标签过滤；支持跨账号采集；负载均衡支持转发规则域名维度。
2024年8月	支持私有网络-跨可用区流量、私有网络-私有连接；COS 支持5分钟粒度的存储相关指标；TDSQL-C MySQL 版支持集群 ID 过滤。
2024年7月	支持消息队列 Pulsar 版；云函数支持版本维度。 注意：云函数新增固定标签 monitor_view，存量实例更新后会产生新时间线。
2024年6月	支持负载均衡（七层独占集群）、腾讯云数据仓库 TCHouse-D、云服务器（内网）；负载均衡（内网）支持监听器维度；云原生 API 网关支持公网负载均衡维度。 注意：负载均衡（内网）新增固定标签 monitor_view，存量实例更新后会产生新时间线。
2024年5月	支持云数据库 KeeWiDB；负载均衡（公网）支持后端服务器维度。
2024年3月	支持云原生 API 网关、消息队列 RabbitMQ 版、EdgeOne（七层）；支持配置云标签键替换和云标签键转换规则。
2023年12月	支持消息队列 RocketMQ 版 v5、CTSDB（InfluxDB 版）、TI-ONE、云数据库 Tendis、弹性公网 Ipv6。
2023年11月	支持网络探测；负载均衡（公网）支持监听器维度、云数据库 MySQL 支持代理节点维度。
2023年9月	支持日志服务、API 网关、负载均衡（四层独占集群）。
2023年8月	支持 CDN（国外域名）、云点播、云函数（别名）、消息队列 RocketMQ 版。

2023年7月	支持采集域名型负载均衡（公网）；支持采集 TDSQL-C MySQL 版云标签。
2023年6月	支持共享带宽包。云数据库 Redis（内存版）Redis 节点维度添加节点类型标签。
2023年5月	支持采集消息队列 CKafka 版云标签。
2023年4月	支持文件存储。
2023年3月	支持云硬盘。
2023年2月	支持负载均衡（内网）、Web 应用防火墙。
2022年11月	支持全球应用加速、云联网、数据传输服务。
2022年9月	支持专线网关；支持采集云产品标签。
2022年7月	支持 Zookeeper、Nacos、COS、CDN（国内域名）。
2022年6月	云监控集成上线。

非腾讯云主机监控

最近更新时间：2024-10-15 16:04:51

背景

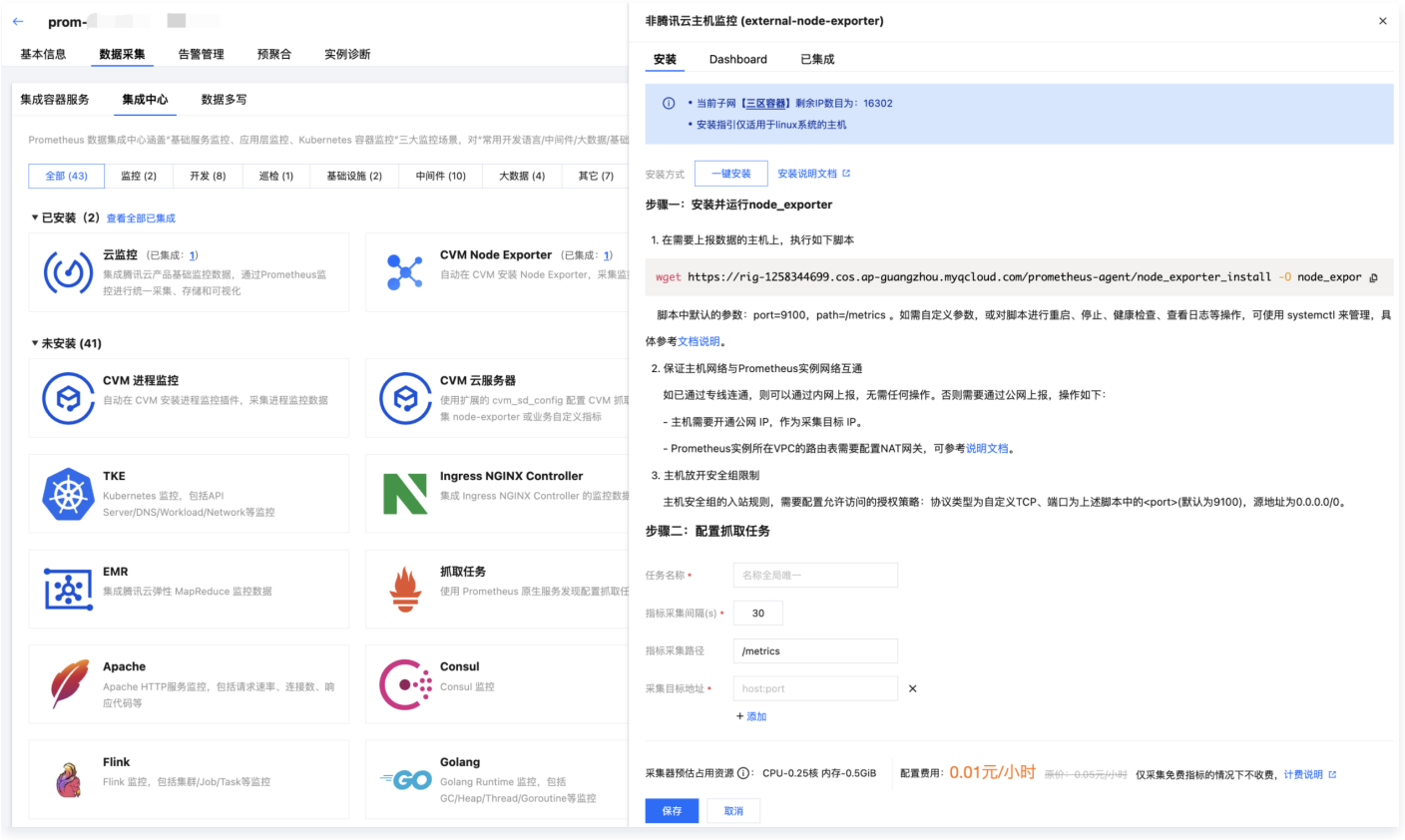
本文主要引导用户如何快速采集非腾讯云主机的监控数据，降低用户配置成本。

接入方式

方式一：一键安装(推荐)

操作步骤

1. 登录 Prometheus 控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，单击**数据采集 > 集成中心**。
4. 在集成中心找到并单击**非腾讯云主机监控**，即会弹出一个安装窗口。



步骤一：安装并运行 node_exporter

1. **在需要上报数据的主机上执行以下脚本：**

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter_install -O
node_exporter_install && chmod +x
node_exporter_install && ./node_exporter_install
```

执行脚本会自动触发以下动作：下载 node exporter、运行 node exporter、检查数据上报、完成（数据成功暴露在9100端口）。
脚本执行结果示例如下：


```
Start downloading Node Exporter v1.3.1.linux-amd64...
--2024-09-23 16:23:56--  https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter
Resolving rig-1258344699.cos.ap-guangzhou.myqcloud.com (rig-1258344699.cos.ap-guangzhou.myqcloud.com)... 159.75.57.35, 159.75.57.69
Connecting to rig-1258344699.cos.ap-guangzhou.myqcloud.com (rig-1258344699.cos.ap-guangzhou.myqcloud.com)|159.75.57.35|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18228926 (17M) [application/octet-stream]
Saving to: '/usr/local/bin/node_exporter'

100%[==========================================================================================================================================================>] 18,228,926  29.3MB/s   in 0.6s

2024-09-23 16:23:57 (29.3 MB/s) - '/usr/local/bin/node_exporter' saved [18228926/18228926]

Download Completed.


Starting Node Exporter...
Node Exporter v1.3.1.linux-amd64 Started.
● node_exporter.service - Node Exporter
   Loaded: loaded (/usr/lib/systemd/system/node_exporter.service; disabled; vendor preset: disabled)
   Active: active (running) since Mon 2024-09-23 16:23:57 CST; 3s ago
 Main PID: 8208 (node_exporter)
   CGroup: /system.slice/node_exporter.service
           └─8208 /usr/local/bin/node_exporter --web.listen-address=:9101

Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=thermal_zone
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=time
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=timex
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=udp_queues
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=uname
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=vmstat
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=xfs
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:115 level=info collector=zfs
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=node_exporter.go:199 level=info msg="Listening on" address=:9101
Sep 23 16:23:57 VM-0-11-tencentos node_exporter[8208]: ts=2024-09-23T08:23:57.454Z caller=tls_config.go:195 level=info msg="TLS is disabled." http2=false


Check the data exposed by node_exporter:
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0

promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
The node-exporter data has been successfully exposed on port 9101. Please go to the Prometheus Monitoring Console of Tencent Cloud Observable Platform to create a collection configuration.
[root@VM-0-11-tencentos ~]#
```

> **说明：**
> 脚本中默认的参数：port=9100，path=/metrics ，如需自定义参数或对脚本进行重启、停止、健康检查、查看日志等操作，可使用 systemctl 来管理。

自定义参数：

- 修改 port，执行脚本语句替换为：

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter_install -O node_exporter_install && chmod +x node_exporter_install && ./node_exporter_install --web.listen-address=":9100"
```

- 修改 path，执行脚本语句替换为：

```
 wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter_install -O node_exporter_install && chmod +x node_exporter_install && ./node_exporter_install --web.telemetry-path="/metrics"
```

> **说明：**
> 更多自定义参数的配置指引可参考 文档说明 。

常用的脚本管理操作：

- 重启：

```
systemctl restart node_exporter
```

- 停止：

```
systemctl stop node_exporter
```

- 状态检查：

```
systemctl status node_exporter
```

- 日志查看：

```
journalctl -u node_exporter
```

2. 保证主机网络与 Prometheus 实例内网互通

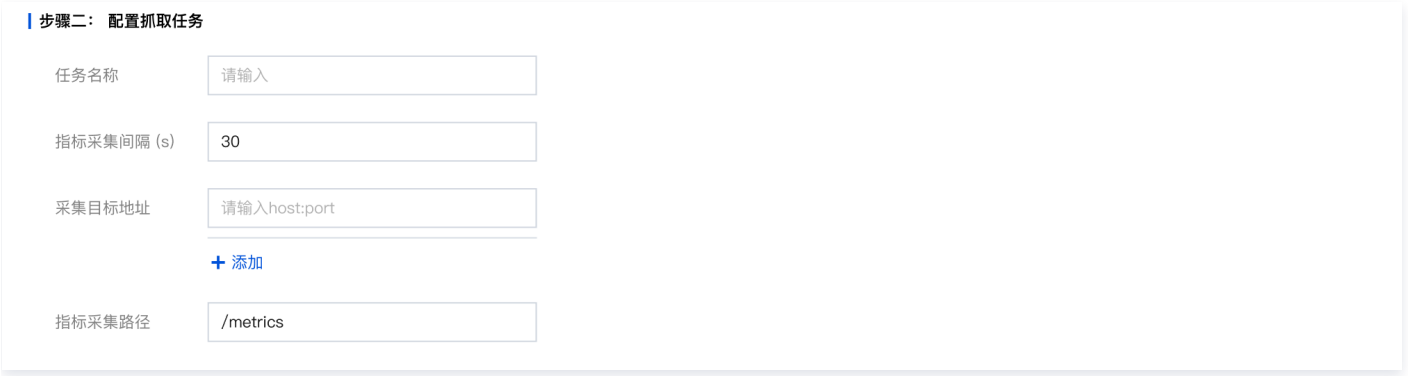
如已通过专线连通，则可以通过内网上报，无需任何操作。否则需要通过公网上报，操作如下：

- 主机需要开通公网 IP，作为采集目标 IP。
- Prometheus 实例所在 VPC 的路由表需要配置 NAT 网关，可参考 TKE Serverless 集群如何放通外网 。

3. 主动放开安全组限制

主机安全组的入站规则，需要配置允许访问的授权策略：协议类型为自定义 TCP、端口为上述脚本中的<port>，源地址为0.0.0.0/0。

步骤二：配置抓取任务



参数	说明
任务名称	集成名称，命名规范如下： • 名称具有唯一性。 • 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
指标采集间隔(s)	输入指标采集间隔，单位s。
采集目标地址	输入采集目标地址，格式：host:port，支持添加多个。
指标采集路径	输入指标采集路径，默认为/metrics。

方式二：自定义安装

上述步骤一中脚本安装的方式，还可以替换为自定义安装，参考下述指引。

1. 下载安装 node_exporter：

在需要上报数据的主机上，下载并安装 node_exporter，您可以点击进入 Prometheus 开源官网下载地址 node_exporter ，也可以直接执行下列命令：

```
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/prometheus-agent/node_exporter -O
node_exporter
```

目录为当前文件夹：

```
[root@VM-0-46-tencentos ~]# ls
node_exporter
[root@VM-0-46-tencentos ~]#
```

2. 运行 node_exporter 采集基础监控数据：

- 赋予权限，执行 node_exporter并查看日志。

```
chmod +x node_exporter && nohup ./node_exporter &
cat nohup.out
```

如下图所示即为执行成功：

```
[root@VM-0-13-tencentos ~]# chmod +x node_exporter && nohup ./node_exporter &
[2] 11678
[root@VM-0-13-tencentos ~]# cat nohup.out
ts=2024-09-27T07:42:28.513Z caller=node_exporter.go:182 level=info msg="Starting node_exporter" version="(version=1.3.1, branch=HEAD, revision=a2321e7b940ddcff26873612bccdf7cd4c42b6b6)"
ts=2024-09-27T07:42:28.513Z caller=node_exporter.go:183 level=info msg="Build context" build_context="(go=go1.17.3, user=root@243aafa5525c, date=20211205-11:09:49)"
ts=2024-09-27T07:42:28.513Z caller=node_exporter.go:185 level=warn msg="Node Exporter is running as root user. This exporter is designed to run as unprivileged user, root is not required."
ts=2024-09-27T07:42:28.513Z caller=filesystem_common.go:111 level=info collector=filesystem msg="Parsed flag --collector.filesystem.mount-points-exclude" flag="^/(dev|proc|run/credentials/.+|sys|var/lib/docker/.+)($|/)
ts=2024-09-27T07:42:28.513Z caller=filesystem_common.go:113 level=info collector=filesystem msg="Parsed flag --collector.filesystem.fs-types-exclude" flag="^(autofs|binfmt_misc|bpf|cgroup2?|configfs|debugfs|devpts|devtmpfs|fusectl|hugetlbfs|iso9660|mqueue|nsfs|overlay|proc|procfs|pstore|rpc_pipefs|securityfs|selinuxfs|squashfs|sysfs|tracefs)$
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:108 level=info msg="Enabled collectors"
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=arp
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=bcache
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=bonding
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=btrfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=conntrack
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=cpu
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=cpufreq
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=diskstats
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=dmi
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=edac
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=entropy
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=fibrechannel
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=filefd
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=filesystem
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=hwmon
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=infiniband
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=ipvs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=loadavg
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=mdadm
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=meminfo
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=netclass
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=netdev
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=netstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=nfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=nfsd
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=nvme
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=os
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=powersupplyclass
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=pressure
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=rapl
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=schedstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=sockstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=softnet
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=stat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=tapestats
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=textfile
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=thermal_zone
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=time
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=timex
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=udp_queues
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=uname
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=vmstat
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=xfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:115 level=info collector=zfs
ts=2024-09-27T07:42:28.514Z caller=node_exporter.go:199 level=info msg="Listening on" address=:9100
ts=2024-09-27T07:42:28.514Z caller=tls_config.go:195 level=info msg="TLS is disabled." http2=false
nohup: ignoring input and appending output to 'nohup.out'
[root@VM-0-13-tencentos ~]#
```

- 可通过下列命令，查看暴露在9100端口的监控数据：

```
curl 127.0.0.1:9100/metrics
```

如下图为执行命令后看到的暴露出来的指标监控数据：

```
[root@VM-0-13-tencentos ~]# curl 127.0.0.1:9100/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.38724e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.38724e+06
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.445375e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 753
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 4.19036e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
```

完成上述操作后，需在页面中配置抓取任务，参考方式一中的 配置描述 。

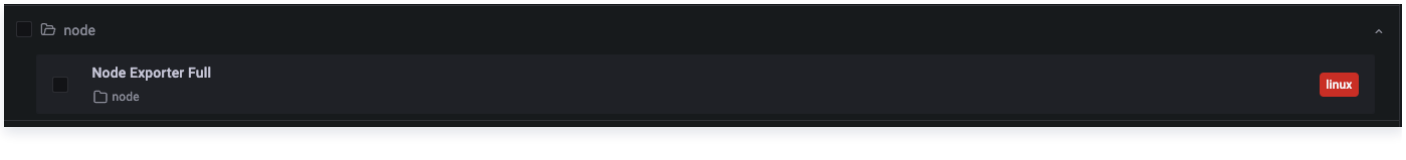
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**数据采集 > 集成中心**，在集成中心页面找到非腾讯云主机监控卡片并点击弹出集成页面，选择 **Dashboard > Dashboard 安装/升级** 来安装对应的 Grafana Dashboard。
3. 打开 Prometheus 实例关联的Grafana实例地址，在 `Dashboards` 页面查看相关的监控大盘。



配置告警

1. 登录 Prometheus 监控服务控制台，选择对应 Prometheus 实例进入管理页面。
2. 选择**告警管理**，可以添加相应的告警策略，详情请参见 新建告警策略 。

通过 Remote Read 读取云托管 Prometheus 实例数据

最近更新时间：2024-10-24 16:23:33

操作场景

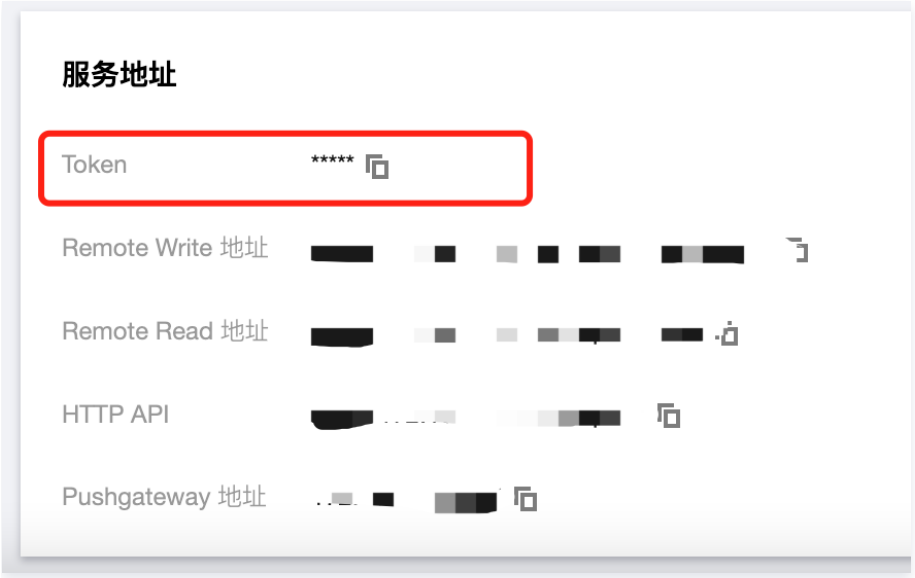
Prometheus 提供了 Remote read 接口，该接口支持将一系列 Prometheus 协议的数据源组织为单一数据源查询。本文介绍如何使用自建 Prometheus 通过 Remote read 读取云托管 Prometheus 实例的数据。

Remote Read 配置

推荐配置 `prometheus.yml` 如下：

```
remote_read:
  - url: 'http://prom_ip:prom_port/api/v1/read'
    read_recent: true
    basic_auth:
      username: app_id
      password: token
```

推荐使用 Basic Auth 方式访问云托管 Prometheus 实例，username 为账号 AppID ，password 为 Prometheus 控制台 > 基本信息 > 服务地址中获取的 Token。



注意事项

- **配置 Remote read 的 Prometheus 需谨慎配置 `global:external_labels` **：
  external_labels 会被附加在 Remote read 的查询条件中，不正确的 label 可能导致查询不到需要的数据。
  `filter_external_labels: false` 配置项可以避免将 external_labels 加入查询条件（v2.34 版本以上支持）。
- **避免出现相同的 series**：
  对于完全相同的两个 series，Prometheus 会在查询合并时在每个时间点在随机一个 series 取值组成新的 series 作为查询结果，这会导致查询结果不准确。
  在 Prometheus 的设计理念中不存在多副本冗余存储的情况，所以不会对这种场景提供支持。

Remote read 完整配置项

> **说明：**
> `[]` 中的配置项为可选项（本文展示 Prometheus:v2.40 版本配置，低版本可能缺少部分配置项，详见 prometheus官方文档 ）

```
# remote read 目标 prometheus 实例的 api 地址
url: <string>

# 标识一个唯一的 remote read 配置名称
[ name: <string> ]

# 查询 promql 中必须包含以下 label 过滤条件才会进行 remote read 查询
required_matchers:
  [ <labelname>: <labelvalue> ... ]

# remote read 查询超时时间
[ remote_timeout: <duration> | default = 1m ]

# 自定义 remote read 请求中附带的 headers，无法覆盖 prometheus 原本添加的 headers
headers:
  [ <string>: <string> ... ]

# 在本地有完整数据存储的时间范围是否进行 remote read 查询
[ read_recent: <boolean> | default = false ]

# 为每个 remote read 请求添加 Authorization header，password password_file 二选一
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# 自定义 Authorization header 配置
authorization:
  # 认证类型
  [ type: <string> | default: Bearer ]
  # 认证密钥，credentials credentials_file 二选一
  [ credentials: <secret> ]
  # 密钥从文件中获取
  [ credentials_file: <filename> ]

# OAuth2.0认证，不能与 basic_auth authorization 同时使用
oauth2:
  [ <oauth2> ]

# TLS 配置
tls_config:
  [ <tls_config> ]

# 代理 URL
[ proxy_url: <string> ]

# 查询请求是否接受3XX 跳转
[ follow_redirects: <boolean> | default = true ]

# 是否启用 HTTP2
[ enable_http2: <bool> | default: true ]

# 是否在 remote read 时附加 external_labels
[ filter_external_labels: <boolean> | default = true ]
```

Agent 自助接入

最近更新时间：2024-10-24 16:23:33

使用场景

采集自建 IDC 上的服务，需要部署 Agent 并管理采集配置，上报监控数据到云上 Prometheus 监控服务。对于云上服务推荐使用 集成中心，集成中心会托管 Agent，提供多种中间件和抓取任务的自动化集成。

获取 Prometheus 实例访问配置

1. 前往 Prometheus 监控控制台，选择对应的实例 ID/名称，在**基本信息 > 服务地址**页面，获取 Remote Write 地址和 Token。

2. 在 账号信息 页面获取 APPID。

确认所在网络环境和云上实例联通

根据获取的 RemoteWrite 地址，执行如下命令，如果网络联通返回信息会包含 `401 Unauthorized` 。

```
curl -v -X POST  ${RemoteWriteURL}
```

安装并启动 vmagent

vmagent 占用较少的资源且兼容 Prometheus 采集配置和 Remote Write 协议而得到广泛使用。本文只介绍 vmagent 常用启动选项，通过 Systemd 或 Docker 来管理 vmagent，更多详细信息可以参考 官方文档。

常用启动选项

- -promscrape.noStaleMarkers：如果采集目标消失，默认会生成所有关联指标的 stale marker 并写到远程存储。设置该选项禁止该行为，能减少内存的占用。
- -loggerTimezone：日志中时间的时区，例如 `Asia/Shanghai, Europe/Berlin` 或者 `Local` (默认是 "UTC")。
- -remoteWrite.tmpDataPath：采集后待写出数据临时存储的文件路径。
- -remoteWrite.url：数据写向远程存储的 URL。
- -remoteWrite.basicAuth.username：远程存储 -remoteWrite.url 对应的 basic auth 用户名。
- -remoteWrite.basicAuth.password：远程存储 -remoteWrite.url 对应的 basic auth 密码。
- -promscrape.config：采集配置的路径，可以是文件路径或者 HTTP URL，更多详情 参考文档 。
- -promscrape.configCheckInterval：检查 -promscrape.config 配置变化的时间间隔，关于配置更新可以 参考文档 。

通过 Docker 管理

1. 在 vmagent 发布页面 选择镜像版本，推荐使用 latest。
2. 替换脚本中的 Prometheus 实例信息，启动 vmagent。

```
mkdir /etc/prometheus
touch /etc/prometheus/scrape-config.yaml
docker run -d --name vmagent --restart always --net host -v /etc/prometheus:/etc/prometheus
victoriametrics/vmagent:latest \
-promscrape.noStaleMarkers \
-loggerTimezone=Local \
-remoteWrite.url="${RemoteWriteURL}" \
-remoteWrite.basicAuth.username="${APPID}" \
-remoteWrite.basicAuth.password='${Token}' \
-remoteWrite.tmpDataPath=/var/lib/vmagent  \
-promscrape.config=/etc/prometheus/scrape-config.yaml \
-promscrape.configCheckInterval=5s
```

3. 查看 vmagent 日志

```
docker ps
docker logs vmagent
```

如果正常启动，执行如下命令会返回 `OK` 。

```
curl localhost:8429/health
```

通过 Systemd 管理

1. 在 vmagent 发布页面，根据操作系统和 CPU 架构，下载对应 vmutils-* 压缩包并解压。
2. 替换脚本中的 Prometheus 实例访问信息，启动 vmagent。

```
mkdir /etc/prometheus
touch /etc/prometheus/scrape-config.yaml
cat >/usr/lib/systemd/system/vmagent.service <<EOF
[Unit]
Description=VictoriaMetrics Agent
After=network.target

[Service]
LimitNOFILE=10240
ExecStart=/usr/bin/vmagent \
-promscrape.noStaleMarkers \
-loggerTimezone=Local    \
-remoteWrite.url="${RemoteWriteURL}" \
-remoteWrite.basicAuth.username="${APPID}" \
-remoteWrite.basicAuth.password="${Token}" \
-remoteWrite.tmpDataPath=/var/lib/vmagent \
-promscrape.config=/etc/prometheus/scrape-config.yaml \
-promscrape.configCheckInterval=5s
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
```

```
systemctl enable vmagent
systemctl start vmagent
sleep 3
systemctl status vmagent
```

3. 查看日志

```
journalctl -u vmagent
```

如果正常启动，执行如下命令会返回 `OK` 。

```
curl localhost:8429/health
```

管理配置

修改配置文件

编辑采集配置文件 `/etc/prometheus/scrape-config.yaml` 添加/更新/删除采集任务，关于 Prometheus 采集任务配置可以参考 官方文档。

```
global:
   scrape_interval: 30s
scrape_configs:
   - job_name: agent-monitor
     static_configs:
        - targets:
             - localhost:8429
```

修改配置后，要过选项 `-promscrape.configCheckInterval` 设置时间才会生效。

查看监控目标信息

执行如下命令，查看采集目标，可以查看配置是否生效并符合预期。

```
curl localhost:8429/api/v1/targets
```

Pushgateway 接入

最近更新时间：2024-12-16 15:51:32

使用场景

PushGateway 是 Prometheus 生态中的一个重要一员，它允许任何客户端向其 Push 符合规范的自定义监控指标，再结合 Prometheus 统一收集监控。Prometheus Pushgateway 用于接收短期任务的指标数据，这些任务通过服务发现的监控系统无法直接监控。Pushgateway 允许临时性的工作（例如批处理作业）将指标推送到一个中央位置，而无需直接暴露其指标。这些数据可以被 Prometheus 服务器拉取和进行持久化存储。

> **注意：**
> Pushgateway 并不能将 Prometheus 转换为基于推送的监控系统，仅用作特殊场景中的指标暂存组件。同时 Pushgateway 不对暂存的指标做持久化保证，重启 Pushgateway 会导致暂存的指标丢失。实际场景请参见 Prometheus 官方文档 示例判断是否应该使用 Pushgateway。

一键安装

1. 登录 腾讯云可观测平台 。
2. 在左侧菜单栏中单击 **Prometheus 监控**。
3. 在实例列表中，选择对应的 Prometheus 实例。
4. 进入实例详情页，单击**数据采集 > 集成中心**。
5. 在集成中心搜索 **Pushgateway**，找到后单击它即会弹出一个安装窗口。
6. 在弹出窗口的安装页面，根据提示填写相关信息，并单击**保存**。

配置说明

参数	说明
名称	集成名称，命名规范如下： • 名称具有唯一性。

	• 名称需要符合下面的正则：'^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$'。
采集超时	Pushgateway 采集超时，时间格式，不能大于采集间隔。
采集间隔	Pushgateway 采集间隔，时间格式。
CPU/核	Pushgateway CPU 核数限制，不能大于64。
内存/Gi	Pushgateway 内存限制，配置时需带上单位Gi，不能大于512Gi。

7. 在已集成列表中获取 Pushgateway 地址信息。

数据推送

基本概念

- 分组： 以 job 和上报时指定的 label 确定的一组指标。
- 分组路径：分组指标上报请求在 pushgateway 中的对应请求路径，格式为：`/metrics/job/<JOB_NAME>{/<LABEL_NAME>/<LABEL_VALUE>}` 。

开发准备

golang

```
go get github.com/prometheus/client_golang/prometheus
```

python

```
pip install prometheus-client
```

java

```
        <dependency>
            <groupId>io.prometheus</groupId>
            <artifactId>prometheus-metrics-core</artifactId>
            <version>1.3.3</version> <!-- 推荐使用最新版本参考：
https://mvnrepository.com/artifact/io.prometheus/prometheus-metrics-core -->
        </dependency>
        <dependency>
            <groupId>io.prometheus</groupId>
            <artifactId>prometheus-metrics-instrumentation-jvm</artifactId>
            <version>1.3.3</version>
        </dependency>
        <dependency>
            <groupId>io.prometheus</groupId>
```

```
            <artifactId>prometheus-metrics-exporter-pushgateway</artifactId>
            <version>1.3.3</version>
        </dependency>
```

其他语言和更详细信息请参见 官方文档 。

PUT 请求

PUT 请求用来向 pushgateway 上报一个完整的分组，pushgateway 会使用 PUT 请求体中的指标更新整个分组（请求体中不存在的指标会被删除）。
请求体为空的 PUT 请求会使 pushgateway 删除整个分组。

golang

```golang
package main

import (
    "log"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/push"

    // 配合后续代码debug使用
    // "github.com/prometheus/common/expfmt"
)

// 定义并注册指标
var (
    serverRequestTotal = prometheus.NewCounterVec(prometheus.CounterOpts{
        Name: "http_server_request_total",
        Help: "total count of http server requests",
    }, []string{"path"})

    metricsRegistry = prometheus.NewRegistry()
)

func init() {
    metricsRegistry.MustRegister(serverRequestTotal)
}

func main() {
    url := "YOUR-PUSHGATEWAY-ADDRESS"

    serverRequestTotal.WithLabelValues("api/v1/list").Inc()

    if err := push.New(url, "tcloud_test_job").
        Collector(metricsRegistry).
        // 自定义分组label
        Grouping("provider", "tcloud").
        Grouping("product", "tmp").
        // 发送text/plain格式请求，便于debug
        // Format(expfmt.NewFormat(expfmt.TypeTextPlain)).
        // 发送PUT请求
        Push(); err != nil {
        log.Fatalf("Could not push completion time to Pushgateway: %v", err)
    }
```

```
}
```

python

```
from prometheus_client import CollectorRegistry, Counter, push_to_gateway, pushadd_to_gateway,
delete_from_gateway

registry = CollectorRegistry()
counter = Counter('http_server_request_total', 'total count of http server request', ['path'],
registry=registry)
address = "YOUR-PUSHGATEWAY-ADDRESS"
job = "tcloud_test_job"
grouping_key = {"product": "tmp", "provider": "tcloud"}

if __name__ == "__main__":
    counter.labels(path='api/v1/list').inc()
    push_to_gateway(address, job=job, grouping_key=grouping_key, registry=registry)
```

java

```
package org.example;

import java.io.IOException;

import io.prometheus.metrics.core.metrics.Counter;
import io.prometheus.metrics.exporter.pushgateway.PushGateway;
import io.prometheus.metrics.instrumentation.jvm.JvmMetrics;
import io.prometheus.metrics.model.registry.PrometheusRegistry;


public class Main {
    private static PrometheusRegistry registry = new PrometheusRegistry();
    private static String address = "YOUR-PUSHGATEWAY-ADDRESS";
    private static String job = "tcloud_test_job";
    private static PushGateway pushGateway =
PushGateway.builder().address(address).groupingKey("product", "tmp").groupingKey("provider",
"tcloud").job(job).registry(registry).build();
    private static Counter counter =
Counter.builder().name("http_request_total").labelNames("path").help("total count of http server
request").register(registry);


    public static void main(String[] args) throws IOException {
        JvmMetrics.builder().register(registry);
        counter.labelValues("api/v1/list").inc();
        pushGateway.push();
    }
}
```

shell

```
cat <<EOF | curl -X PUT --data-binary @-
http://*.*.*.*:8080/metrics/job/some_job/some_group_key1/value1/some_group_key2/value2
# TYPE some_metric counter
```

```
some_metric{label="val1"} 42
# TYPE another_metric gauge
# HELP another_metric Just an example.
another_metric 2398.283
EOF
```

POST 请求

POST 请求用来向 pushgateway 上报分组中需要修改的部分指标，pushgateway 仅会更新 POST 请求体中存在的指标，其他指标不变。
请求体为空的 POST 请求仅会更新 pushgateway 中记录的上报时间。

golang

```
// 整体逻辑同PUT请求示例
err := push.New(url, "tcloud_test_job").
       Collector(metricsRegistry).
       // 自定义分组label
       Grouping("provider", "tcloud").
       Grouping("product", "tmp").
       // 发送POST请求
       Add()
```

python

```
if __name__ == "__main__":
    counter.labels(path='api/v1/list').inc()
    pushadd_to_gateway(address, job=job, grouping_key=grouping_key, registry=registry)
```

java

```
public static void main(String[] args) throws IOException {
    counter.labelValues("api/v1/list").inc();
    pushGateway.pushAdd();
}
```

shell

```
cat <<EOF | curl -X POST --data-binary @-
http://*.*.*.*:8080/metrics/job/some_job/some_group_key1/value1/some_group_key2/value2
# TYPE some_metric counter
some_metric{label="val1"} 42
# TYPE another_metric gauge
# HELP another_metric Just an example.
another_metric 2398.283
EOF
```

DELETE 请求

DELETE 请求用来删除 pushgateway 中的整个分组。

golang

```
// 整体逻辑同PUT请求示例
err := push.New(url, "tcloud_test_job").
       // 自定义分组label
       Grouping("provider", "tcloud").
       Grouping("product", "tmp").
       // 发送DELETE请求
       Delete()
```

python

```
if __name__ == "__main__":
    delete_from_gateway(address, job=job, grouping_key=grouping_key)
```

java

```
public static void main(String[] args) throws IOException {
    pushGateway.delete();
}
```

shell

```
curl -X DELETE
http://*.*.*.*:8080/metrics/job/some_job/some_group_key1/value1/some_group_key2/value2
```

管理 API

pushgateway 集成支持原生 pushgateway 的管理 API。可使用以下命令管理 pushgateway 自带的 API：

```
PUT /api/v1/admin/wipe
```

> ⚠ **注意：**
> wipe API 会清空 pushgateway 中的全部上报指标。

接入建议

因为 pushgateway 自身实现和设计逻辑，仅建议短期任务或特殊场景（存在网络隔离，仅支持主动推送指标等）使用 pushgateway 接入。可以根据实际需要参考以下建议：

- **每个上报端使用单独分组**
  多个上报端同时上报指标到同一个 pushgateway 时，建议附带单独的分组 label，例如 `instance="instanceXX", instance="ip:port"` 等，防止在 pushgateway 中指标相互覆盖。
- **（持续上报场景）上报结束后删除 pushgateway 中对应指标**
  在持续使用 pushgateway 上报的场景中，上报端离线前应主动删除上报的分组，防止最后一次上报的数据一直保留在 pushgateway 中被反复采集。

Docker 接入

最近更新时间：2024-10-24 16:23:33

操作场景

Docker Daemon 是 Docker 的核心组件之一，它是一个持续运行的后台进程，负责管理 Docker 容器的创建、运行和监控。目前其支持了 Prometheus 指标导出能力，可以通过配置将其开启。腾讯云可观测平台 Prometheus 监控服务通过抓取任务采集 Docker 暴露的 Daemon 监控数据，并提供了开箱即用的 Grafana 监控大盘。

操作步骤

前提条件

docker 版本不低于 17.04.0。

开启 Prometheus 监控

配置 docker daemon

要将 Docker Daemon 配置为 Prometheus 目标，需要在 daemon.json 配置文件中添加 metrics-address 配置项，同时由于该功能在某些版本中属于实验性功能，可能还需要添加 experimental 配置项，以下为各系统中 daemon.json 配置默认位置，若配置不存在，则需要创建该配置：

- Linux：`/etc/docker/daemon.json`
- Windows 服务器：`C:\ProgramData\docker\config\daemon.json`
- Docker Desktop：打开 Docker Desktop 设置并选择 Docker Engine 来编辑文件。

如果文件为空，将下面内容放入即可：

```
{
  "metrics-addr" : "127.0.0.1:9323",  ## 集群节点上的docker导出指标时，127.0.0.1替换为节点ip，端口按实际业务调整
  "experimental" : true
}
```

如果文件非空，添加字段 metrics-address 和 experimental，并确保文件仍然是有效的 JSON 格式。
保存文件或配置，重启 Docker：

```
systemctl restart docker
或者
systemctl restart dockerd
```

> ⚠ **注意：**
> 除了最后一行，每一行都需要用逗号结尾。

验证

登录集群或者 docker 容器中 pod 验证拉取指标：

```
curl 127.0.0.1:9323/metrics ## 使用上述配置中的地址端口
```

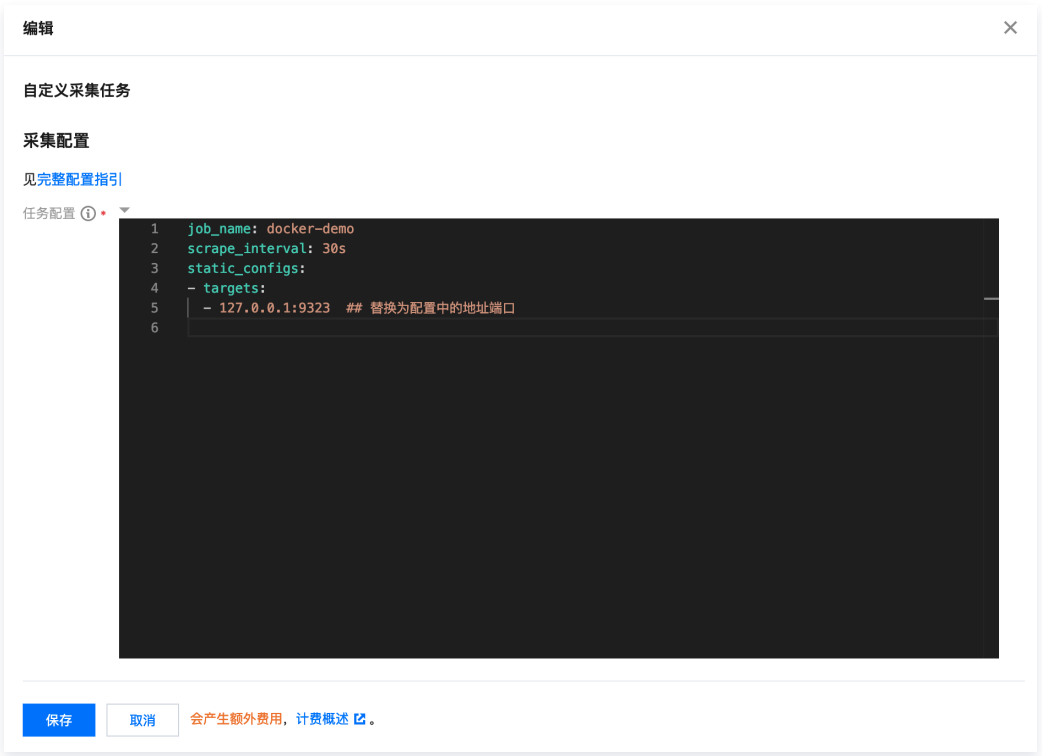
即可看到拉取的指标信息：

```
# HELP engine_daemon_engine_cpus_cpus The number of cpus that the host system of the engine has
# TYPE engine_daemon_engine_cpus_cpus gauge
engine_daemon_engine_cpus_cpus 2
# HELP engine_daemon_engine_info The information related to the engine and the OS it is running on
# TYPE engine_daemon_engine_info gauge
engine_daemon_engine_info{architecture="x86_64",commit="562656c364",daemon_id="FPJ5:VKVG:WAMF:LKPE:FDRQ:345D:HJXM:KQXA:YXKO:ZNQU:LE5N:APQS",graphdriver="overlay2",kernel="5.4.119-1
e="linux",version="v19.3.9-tke.1"} 1
# HELP engine_daemon_engine_memory_bytes The number of bytes of memory that the host system of the engine has
# TYPE engine_daemon_engine_memory_bytes gauge
engine_daemon_engine_memory_bytes 1.8052096e+09
# HELP engine_daemon_events_subscribers_total The number of current subscribers to events
# TYPE engine_daemon_events_subscribers_total gauge
engine_daemon_events_subscribers_total 0
# HELP engine_daemon_events_total The number of events logged
# TYPE engine_daemon_events_total counter
engine_daemon_events_total 28
# HELP engine_daemon_health_checks_failed_total The total number of failed health checks
# TYPE engine_daemon_health_checks_failed_total counter
engine_daemon_health_checks_failed_total 0
# HELP engine_daemon_health_checks_total The total number of health checks
# TYPE engine_daemon_health_checks_total counter
engine_daemon_health_checks_total 0
# HELP engine_daemon_network_actions_seconds The number of seconds it takes to process each network action
# TYPE engine_daemon_network_actions_seconds histogram
engine_daemon_network_actions_seconds_bucket{action="allocate",le="0.005"} 0
engine_daemon_network_actions_seconds_bucket{action="allocate",le="0.01"} 0
engine_daemon_network_actions_seconds_bucket{action="allocate",le="0.025"} 2
```

指标采集

1. 登录 Prometheus 监控服务控制台 。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索**抓取任务**，单击**一键安装**。

5. 在新建页，填写指标采集名称和地址等信息，并单击**保存**。



6. 在集成中心搜索 docker，单击 **Dashboard 安装/升级**。

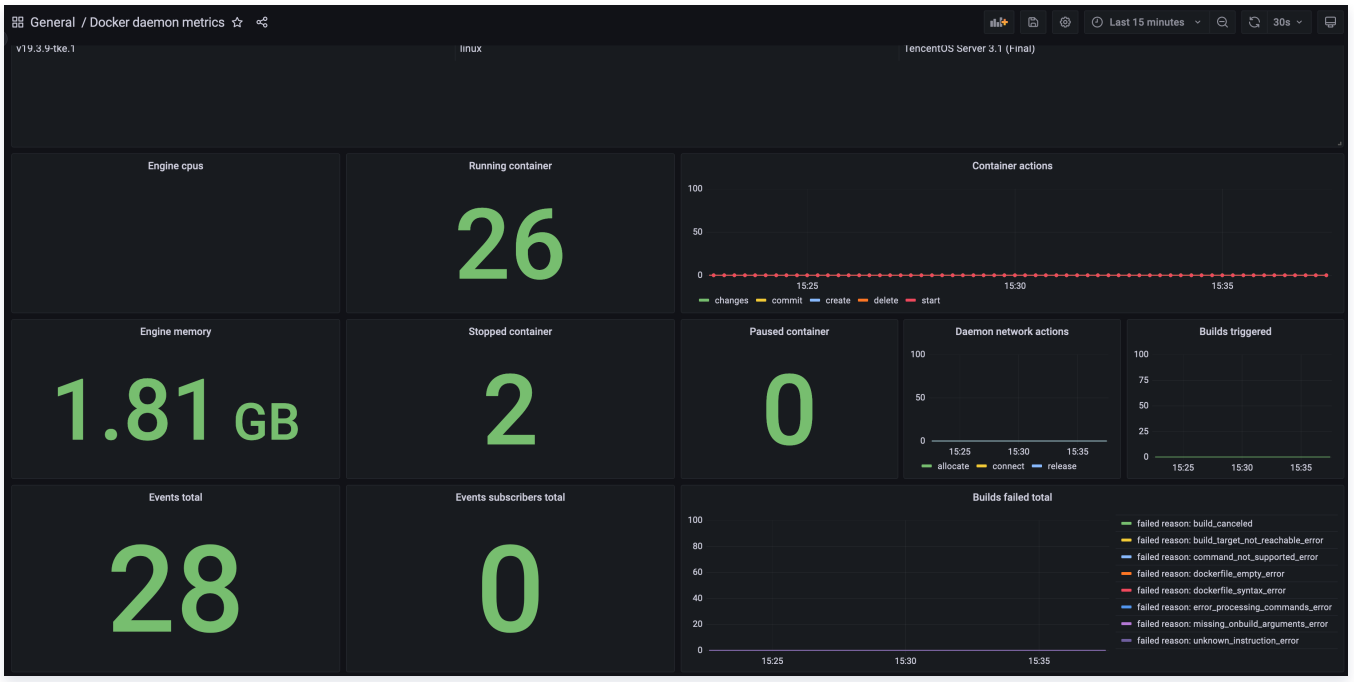
查看监控

前提条件

Prometheus 实例已绑定 Grafana 实例。

操作步骤

1. 登录 腾讯云可观测平台 Prometheus 控制台，选择对应 Prometheus 实例进入管理页面。
2. 在实例**基本信息**页面，找到绑定的 grafana 地址，打开并登录，然后在 others 文件夹中找到 **Docker daemon metrics** 监控面板，查看实例相关监控数据，如下图所示：



配置告警

腾讯云 Prometheus 托管服务支持告警配置，可根据业务实际的情况来添加告警策略。详情请参见 新建告警策略 。

TKE 集群内安装组件说明

最近更新时间：2025-01-06 10:26:52

本文介绍 Prometheus 监控服务在 集成容器服务 过程中，在用户 TKE 集群内安装的各个组件的功能，使用权限和占用资源。

proxy-agent

组件介绍

由于 TKE 集群有独立的网络环境，proxy-agent 部署在集群内为集群外的采集组件提供访问代理。外部采集组件一方面通过 proxy-agent 服务发现集群内的资源，另一方面通过 proxy-agent 抓取指标并写到 Prometheus 实例的时序存储中。

部署在集群内的资源对象

Namespace	kubernetes 对象名称	类型	资源量	说明
<Prometheus 实例 ID>	proxy-agent	Deployment	0.25C256Mi*2	采集代理
	<Prometheus 实例 ID>	ServiceAccount	-	权限载体
		ClusterRole	-	采集权限相关
		Role	-	外部集群额外管理权限
	<Prometheus 实例 ID>-crb	ClusterRoleBinding	-	采集权限相关
	<Prometheus 实例 ID>-rb	RoleBinding	-	外部集群额外管理权限

组件权限说明

权限场景

功能	涉及对象	涉及操作权限
采集配置管理	scrapeconfigs,servicemonitors,podmonitors,probes,configmaps,secrets,namespaces	get/list/watch
服务发现	services,endpoints,nodes,pods,ingresses	get/list/watch
部分系统组件指标抓取	nodes/metrics,nodes/proxy,pods/proxy	get/list/watch
带 RBAC 鉴权的指标抓取	/metrics,/metrics/cadvisor	get

外部 Kubernetes 集群额外权限场景

功能	涉及对象	涉及操作权限
采集配置管理	scrapeconfigs,servicemonitors,podmonitors, probes	*(all)
管理采集专用 namespace	<Prometheus 实例 ID>/*	*(all)

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: prom-instance
rules:
  - apiGroups:
      - monitoring.coreos.com
    resources:
      - scrapeconfigs
      - servicemonitors
      - podmonitors
      - probes
      - prometheuses
      - prometheusrules
    verbs:
      - get
      - list
      - watch
    # 外部 Kubernetes 集群使用
    # - *
  - apiGroups:
      - ""
    resources:
      - namespaces
      - configmaps
      - secrets
      - nodes
      - services
      - endpoints
      - pods
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups: [ "" ]
    resources:
      - nodes/metrics
      - nodes/proxy
      - pods/proxy
    verbs:
      - get
      - list
      - watch
  - nonResourceURLs: [ "/metrics", "/metrics/cadvisor" ]
    verbs:
      - get
---
# 外部 Kubernetes 集群使用
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: prom-instance
  namespace: prom-instance
```

```
rules:
  - apiGroups: [ "", "extensions", "apps" ]
    resources: [ "*" ]
    verbs: [ "*" ]
```

tke-kube-state-metrics

组件介绍

tke-kube-state-metrics 使用开源组件 kube-state-metrics，监听集群的 API server，生成集群内各种对象的状态指标。

部署在集群内的资源对象

Namespace	kubernetes 对象名称	类型	资源量	说明
kube-system	tke-kube-state-metrics	Statefulset	0.5C512Mi	采集程序
		ServiceAccount	-	权限载体
		ClusterRole	-	采集权限相关
		ClusterRoleBinding	-	采集权限相关
		Service	-	采集程序对应服务，供服务发现使用
		Role	-	分片采集权限相关
		RoleBinding	-	分片采集权限相关
	kube-state-metrics	ServiceMonitor	-	采集配置

组件权限说明

权限场景

功能	涉及对象	涉及操作权限
监听集群内各种资源的状态	绝大部分 Kubernetes 资源	list/watch
获取采集 Pod 所在分片序号	statefulsets,pods	get

权限定义

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tke-kube-state-metrics
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - secrets
      - nodes
      - pods
      - services
      - serviceaccounts
      - resourcequotas
      - replicationcontrollers
```

```
      - limitranges
      - persistentvolumeclaims
      - persistentvolumes
      - namespaces
      - endpoints
    verbs:
      - list
      - watch
  - apiGroups:
      - apps
    resources:
      - statefulsets
      - daemonsets
      - deployments
      - replicasets
    verbs:
      - list
      - watch
  - apiGroups:
      - batch
    resources:
      - cronjobs
      - jobs
    verbs:
      - list
      - watch
  - apiGroups:
      - autoscaling
    resources:
      - horizontalpodautoscalers
    verbs:
      - list
      - watch
  - apiGroups:
      - authentication.k8s.io
    resources:
      - tokenreviews
    verbs:
      - create
  - apiGroups:
      - authorization.k8s.io
    resources:
      - subjectaccessreviews
    verbs:
      - create
  - apiGroups:
      - policy
    resources:
      - poddisruptionbudgets
    verbs:
      - list
      - watch
  - apiGroups:
      - certificates.k8s.io
    resources:
      - certificatesigningrequests
    verbs:
      - list
```

```
      - watch
  - apiGroups:
      - storage.k8s.io
    resources:
      - storageclasses
      - volumeattachments
    verbs:
      - list
      - watch
  - apiGroups:
      - admissionregistration.k8s.io
    resources:
      - mutatingwebhookconfigurations
      - validatingwebhookconfigurations
    verbs:
      - list
      - watch
  - apiGroups:
      - networking.k8s.io
    resources:
      - networkpolicies
      - ingresses
    verbs:
      - list
      - watch
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs:
      - list
      - watch
  - apiGroups:
      - rbac.authorization.k8s.io
    resources:
      - clusterrolebindings
      - clusterroles
      - rolebindings
      - roles
    verbs:
      - list
      - watch
---
kind: Role
metadata:
  name: tke-kube-state-metrics
  namespace: kube-system
rules:
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - get
  - apiGroups:
      - apps
    resourceNames:
      - tke-kube-state-metrics
```

```
    resources:
      - statefulsets
    verbs:
      - get
```

tke-node-exporter

组件介绍

tke-node-exporter 使用开源项目 node_exporter，部署在集群内的每个 Node 上，用来采集硬件和类Unix操作系统指标。

部署在集群内的资源

Namespace	kubernetes 对象名称	类型	资源量	说明
kube-system	tke-node-exporter	DaemonSet	0.1C180Mi*node数量	采集程序
		Service	-	采集程序对应服务，供服务发现使用
	node-exporter	ServiceMonitor	-	采集配置

组件权限说明

该组件不使用任何集群权限。

安全组开放说明

最近更新时间：2024-10-24 16:23:33

概述

本文介绍 Prometheus 监控服务 在 集成容器服务 过程中，托管集群和用户集群安全组需要开放的端口说明。同时介绍绑定出现安全组相关问题时的解决方式。

托管集群

托管集群安全组由 Prometheus 监控服务创建，一般情况下不需要修改。

安全组

规则	协议端口	策略
入站规则	TCP:9093,9090,10901,10902,9990,3000,8080,8008	允许
入站规则	TCP:8100-8200	允许
出站规则	ALL	允许

端口说明

端口	作用	备注
TCP:8008	proxy-server 监听 proxy-agent 连接端口	-
TCP:8080	集群内部接口调用端口	-
TCP:3000	grafana 代理端口	-
TCP:9990	cm-notify 同步端口	即将下线
TCP:10901,10902	thanos sidecar 监听地址	-
TCP:9090	配置 reload 端口，采集数据查询接口	-
TCP:9093	告警用端口	-
TCP:8100-8200	proxy-server 监听采集用端口	由于采集端口范围为100个，所以关联集群最多不能超过100个

查看方式

登录 Prometheus 监控，选择实例的 ID/名称 > **实例诊断**，采集诊断选择**集成中心**，在**采集架构图**中就可以看到托管集群安全组，点击安全组可通过超链接跳转至安全组界面，即可查看托管集群安全组。

用户集群

用户集群安全组在用户创建节点时指定，若不指定则为默认安全组。

安全组

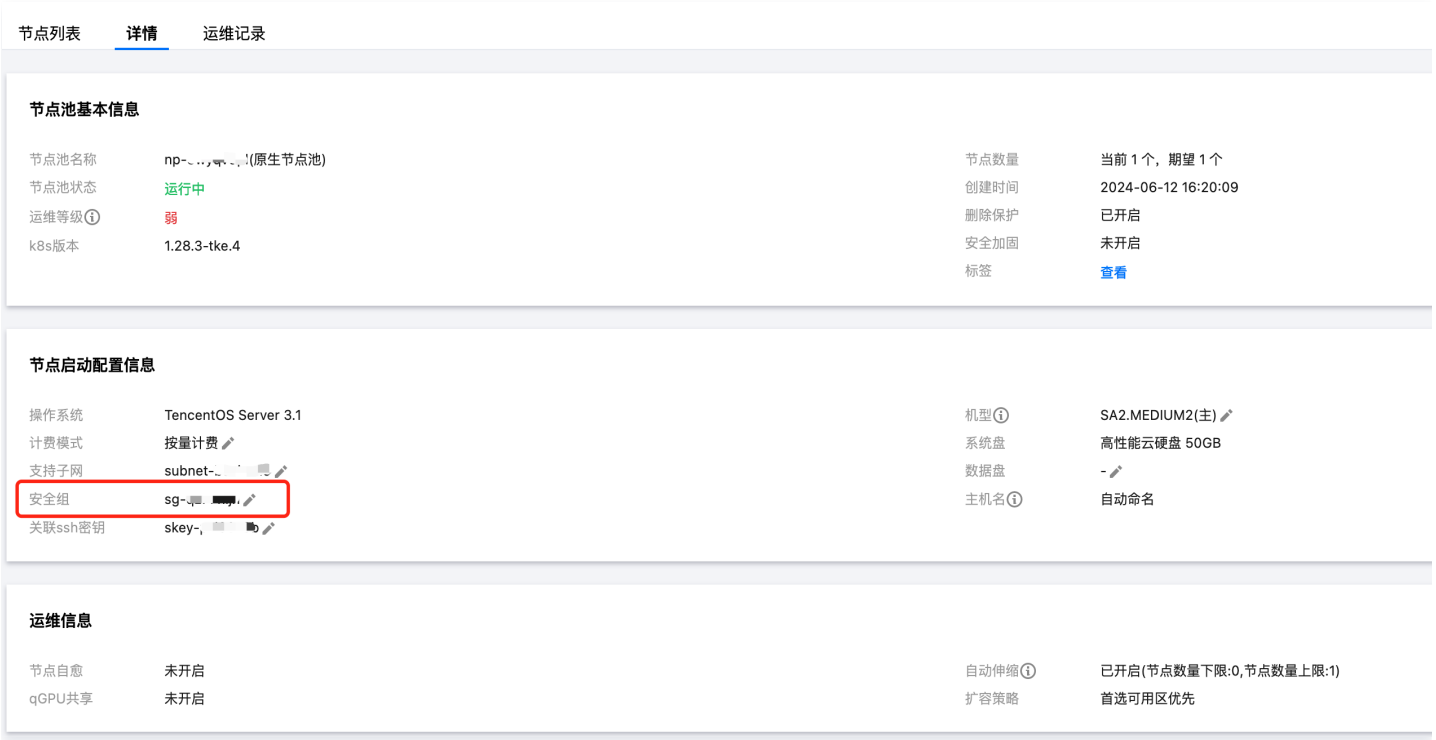
规则	集群类型	协议端口	策略	说明
出站规则	-	TCP:8008	允许	保证 proxy-agent 与 proxy-server 能够建立连接
入站规则	标准集群	-	-	标准集群无需开放端口
入站规则	独立集群	TCP:9092,8180,443,10249,9100,60002,10252,10257,10259,10251等	允许	独立集群需额外开放 master 节点相关端口，保证 proxy-agent 能够拉取 master 节点相关监控数据

查看方式

登录 Prometheus 监控，选择实例的 ID/名称 > **数据采集**，点击集群 ID/名称即可跳转到该集群容器服务界面。

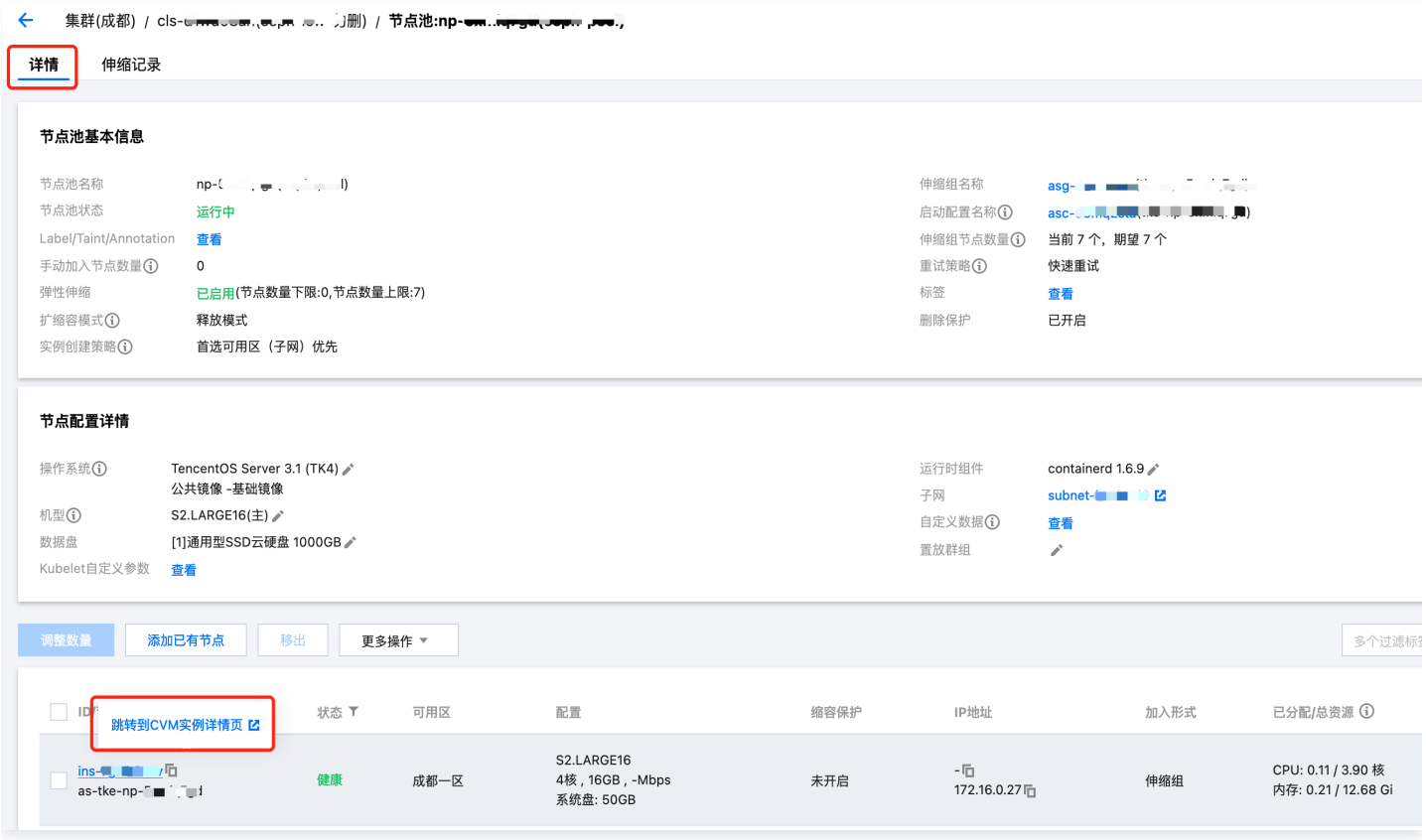
原生节点

点击**节点管理** > **Worker 节点** > **节点池**，点击节点池 id，在**详情**页即可看到安全组，在**腾讯云安全组**按安全组 id 进行搜索，查看具体规则即可。



普通节点

点击节点管理 > Worker 节点 > 节点池，点击节点池 id，在详情页将鼠标放在节点 ID 上点击跳转到 CVM 实例详情页：



跳转至实例详情页面后，点击安全组，即可查看具体安全组信息：

超级节点

点击**节点管理** > **Worker 节点** > **节点池**，点击节点池 id，在**节点池基本信息**中即可查看安全组：

相关问题

问题描述

绑定状态异常，“安装 tmp-agent CR”步骤显示“context deadline exceeded”：

安装 proxy-server 服务	完成	2024-06-12 11:04:52	2024-06-12 11:04:52	N/A
用户集群安装 proxy-agent	完成	2024-06-12 11:04:52	2024-06-12 11:04:57	N/A
安装基础采集配置	完成	2024-06-12 11:04:55	2024-06-12 11:05:00	N/A
安装 tmp-agent CR	异常	2024-06-12 11:04:57		{Reason:get resourceInformer failed,Object:TMPAgent/prom-[redacted]/tke-cls-[redacted],Message:failed to sync cache for ServiceMonitor informer}: context deadline exceeded

用户集群内安装组件说明

确定 刷新

问题处理

vpc 是否相同，或者打通

1. 点击用户集群链接，打开关联集群，查看集群节点网络（即 vpcid）：

2. 在 Prometheus 实例的**基本信息**页面，点击**所属网络**，打开后即可查看集群网络：

3. 对比 vpcid，若不一致，查看 vpc 是否通过云联网打通。若云联网没有打通，则需要关联云联网，打通两边的 vpc，或者在关联集群时勾选**创建公网 CLB**。若云联网已打通，仍不能关联成功，需查看云联网是否达到带宽上限，若已达到则需要调大云联网带宽上限。

- 关联云联网：

- 勾选创建公网 CLB：

- 升级云联网带宽。进入 私有网络 > 云联网 页面，选择实例 ID > **带宽管理** > **升级带宽**：

安全组是否放通

1. 查看用户集群安全组，查看方式见 **用户集群安全组查看方式**，查看规则是否与要求一致；
2. 若用户集群为独立集群，查看 Master&Etcd 安全组信息，点击**节点管理 > Master&Etcd > 节点池**，点击节点池 id，并将鼠标放在节点 ID 上，点击**跳转到CVM实例详情页**，然后在 CVM 的**安全组**页面即可查看具体安全组信息：

检查安全组规则是否符合要求。

批量安装 Node Exporter

最近更新时间：2024-11-04 09:39:02

Prometheus 监控服务支持使用 CVM Node Exporter 采集云服务器（Cloud Virtual Machine，CVM）实例上的指标。目前已支持用户在控制台批量选择 CVM 实例并安装 Node Exporter。您可以参考下文进行操作。

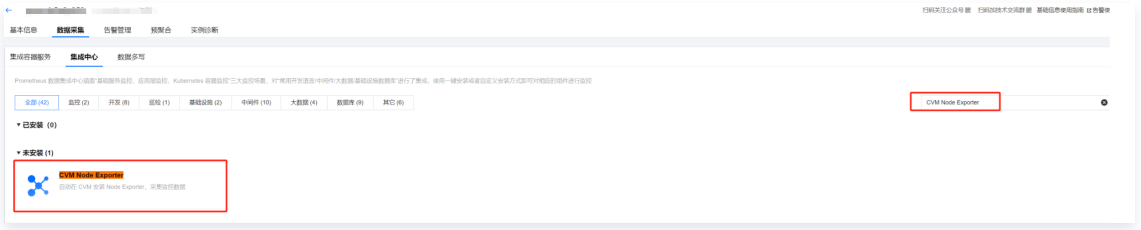
前提条件

CVM 已 安装腾讯云自动化助手（TencentCloud Automation Tools，TAT）。

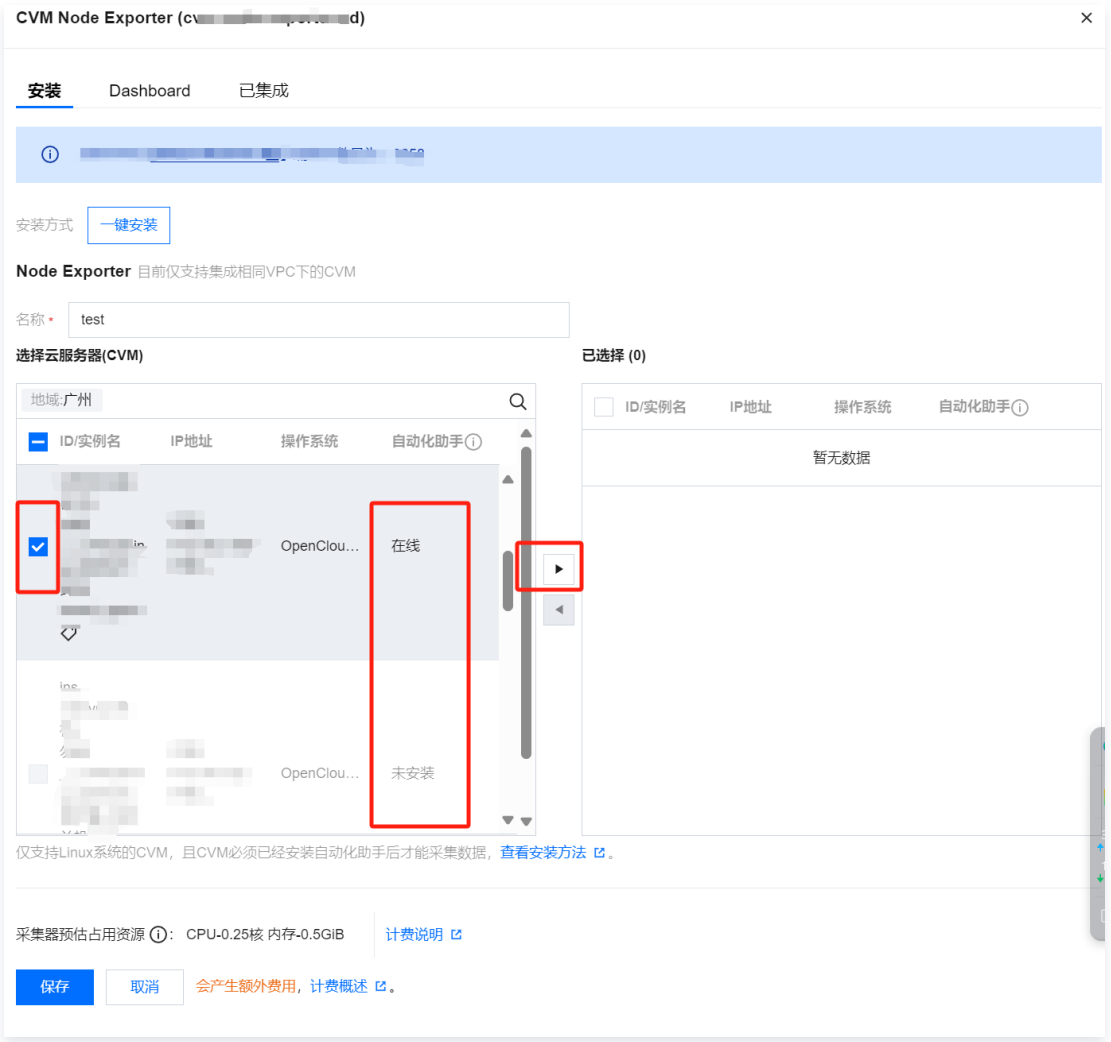
操作步骤

> **说明**
> 目前仅支持集成相同 VPC 下的 CVM。

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择对应的 Prometheus 实例。
3. 进入实例详情页，单击**数据采集 > 集成中心**。
4. 在集成中心搜索 **CVM Node Exporter**，找到后单击它即会弹出一个安装窗口。



5. 进入 Node Exporter 安装页面，勾选已安装腾讯云自动化助手的 CVM 实例，并单击右侧 按钮，完成后单击**保存**。
   您可以在实例列表的**自动化助手**列中可查看自动化助手安装状态。



6. 保存成功后，等待安装。如下图，若运行状态为已部署，则安装成功。

CVM 进程监控

最近更新时间：2025-01-15 14:09:12

Prometheus 监控服务支持使用 CVM Node Exporter 采集云服务器（Cloud Virtual Machine，CVM）实例上的指标。目前已支持用户在控制台批量选择 CVM 实例并安装 Node Exporter。您可以参考下文进行操作。

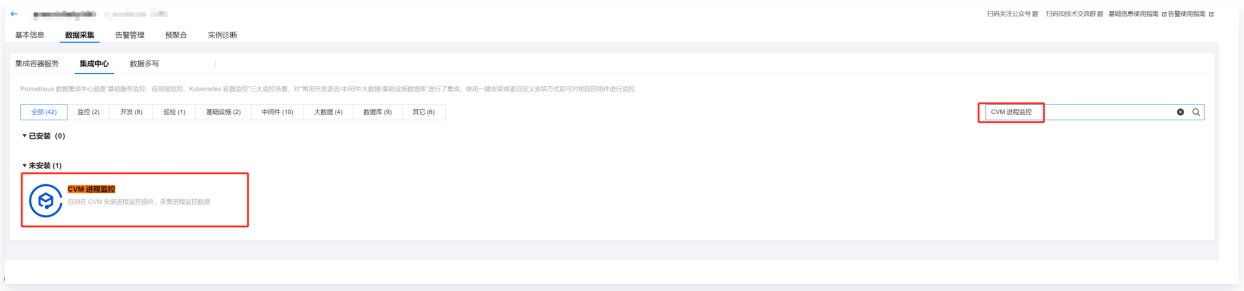
前提条件

CVM 已安装 腾讯云自动化助手（TencentCloud Automation Tools，TAT）。

操作步骤

> **说明：**
> 目前仅支持集成相同 VPC 下的 CVM。

1. 登录 Prometheus 监控服务控制台。
2. 在实例列表中，选择并进入对应的 Prometheus 实例。
3. 在实例详情页，选择**数据采集 > 集成中心**。
4. 在集成中心搜索 **CVM 进程监控**，找到后单击它即会弹出一个安装窗口。



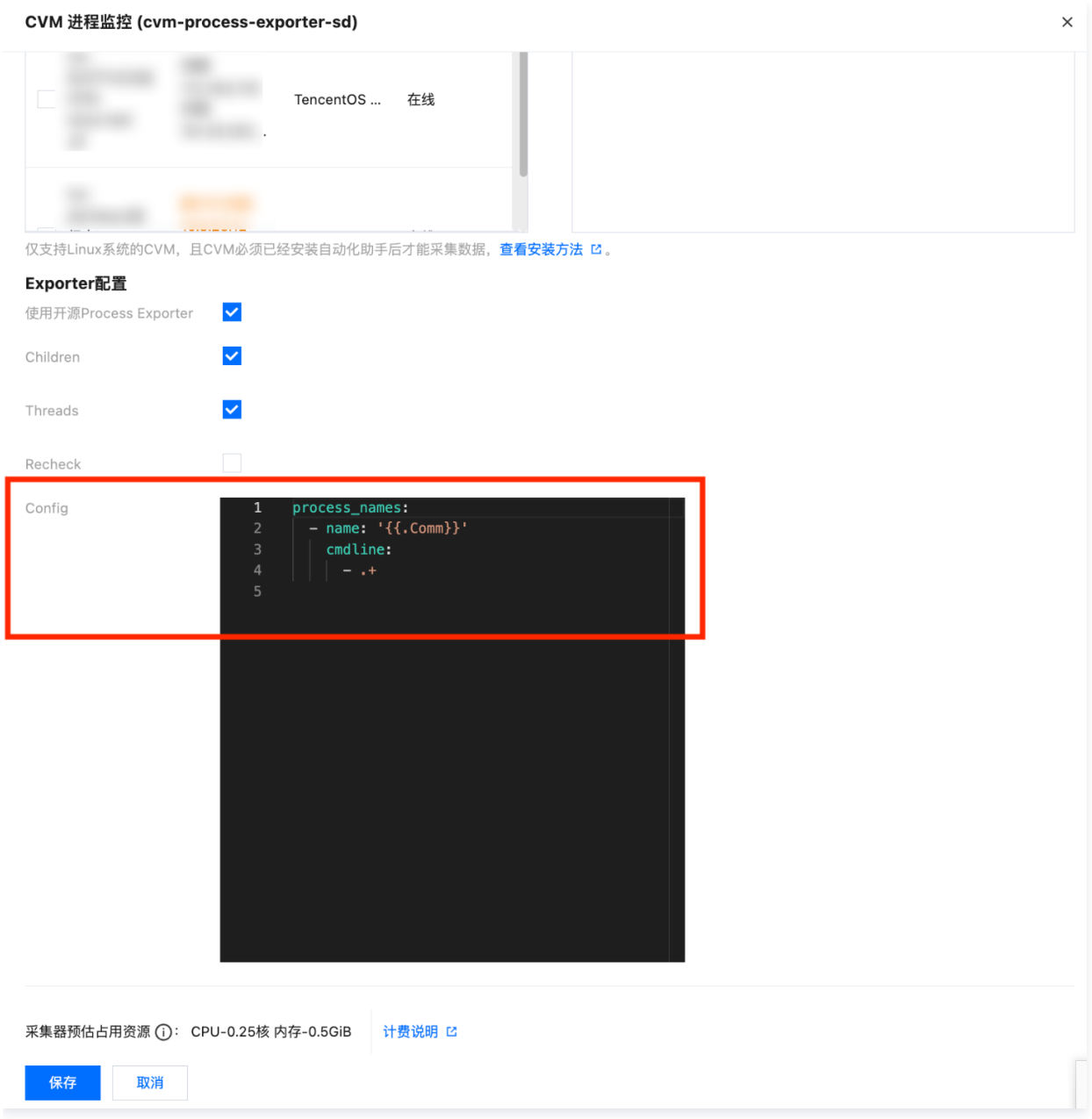
5. 进入 CVM 进程监控安装页面，勾选已安装腾讯云自动化助手的 CVM 实例，并单击右侧 按钮，完成后单击**保存**。
   您可以在实例列表的**自动化助手**列中可查看自动化助手安装状态。

6. 为了指标的完备性及配置的灵活性，建议勾选**使用开源 Process Exporter**，开源 exporter 详情参见 process-exporter 。

7. 保存成功后，等待安装。如下图，若运行状态为已部署，则安装成功。

自定义进程标识

若您使用的是开源 exporter（见上述操作步骤 6），则可通过 `groupname` 标签，提取想要监控的进程的标识。
在默认配置下，`groupname` 的值是进程执行程序的基本名称（不包含路径和扩展名），如下图所示：



该默认配置可根据需要修改。例如要想将完整命令提取作为 `groupname`，可以将开源 exporter 的配置部分改为：

```
process_names:
  - name: "{{.Matches.fullCmd}}"
    cmdline:
    - (?P<fullCmd>.*)
```

修改后的效果：

例如一条进程执行命令为 `/usr/local/bin/python3.8 /my/path/test.py` 。

- **使用默认配置时，提取出的 `groupname` 标签的值为 `python3.8` 。**
- **如果按上述配置，改为提取完整命令，则 `groupname` 标签的值为 `/usr/local/bin/python3.8 /my/path/test.py` 。**

用法示例：如果想针对该进程告警，则 PromQL 里可以用 `groupname` 将它筛选出来：

`namedprocess_namegroup_num_procs{groupname="/usr/local/bin/python3.8 /my/path/test.py"}==0` 。