Tencent Cloud Observability Platform

Prometheus Monitoring

Product Documentation

Contents

Prometheus Monitoring
Product Introduction
Overview

Last updated：2024-08-07 21:50:20

TencentCloud Managed Service for Prometheus (TMP) provides the highly available Prometheus service as well as the open-source visualization tool Grafana and Cloud Monitor alarms while inheriting the monitoring capabilities of the open-source Prometheus, which reduce your development and OPS costs.

Prometheus Overview

Prometheus is an open-source monitoring system. Similar to Kubernetes inspired by Google's Borgman monitoring system, it was inspired by Google's Borgman monitoring system. It was created and developed in 2012 by SoundCloud's internal engineers and released in January 2015. In May 2016, it became the second project after Kubernetes to officially join Cloud Native Computing Foundation (CNCF). Nowadays, it is usually used for monitoring in the most common Kubernetes container management systems.
Prometheus has the following features:
Custom multidimensional data models (a time series data entry is composed of a metric and a key-value pair called label).
Flexible and powerful query language PromQL, which can use multidimensional data to complete complex monitoring queries.
Independence from distributed storage and support for operations based on one single master node.
Time series data collection through HTTP pull.
Data push through Pushgateway.
Acquisition of collection target servers through dynamic scrape configuration or static configuration.
Integration with Grafana to easily support various visual charts and dashboards.

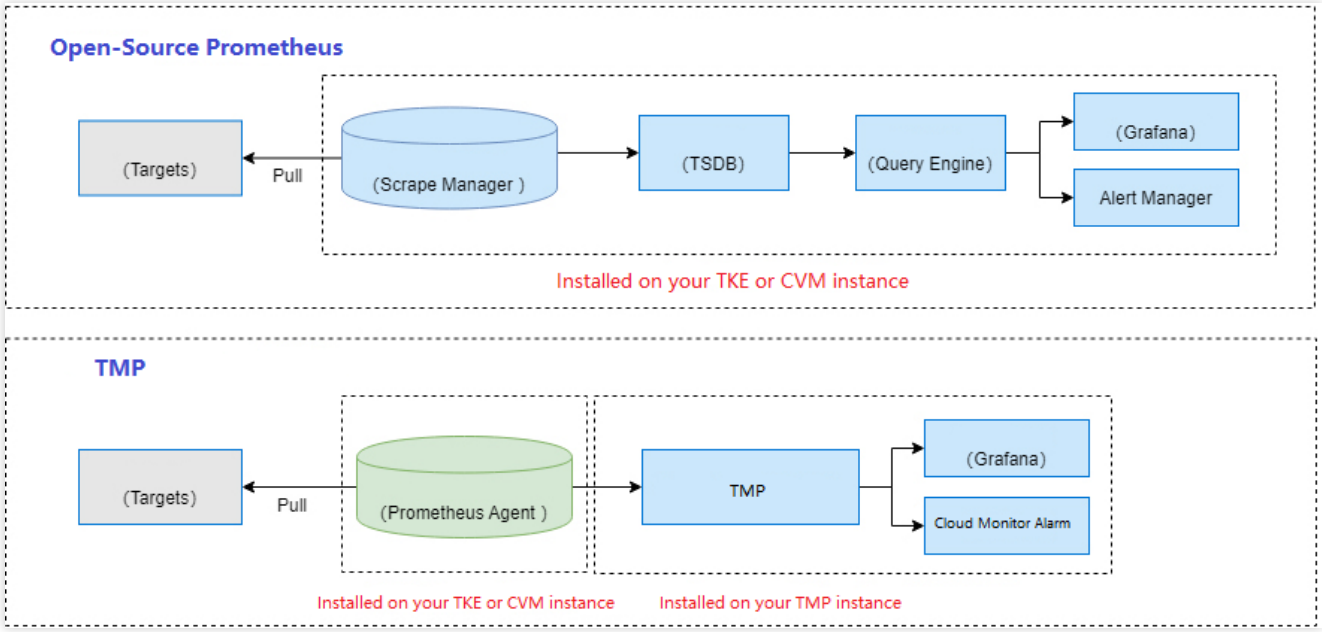
Features

According to the layering of monitoring, TMP covers business monitoring, application layer monitoring, middleware monitoring, and system layer monitoring. Together with Cloud Monitor's alarming capabilities and open-source Grafana, it can provide a one-stop all-round monitoring system to help your quickly identify and locate business problems and reduce the impact of various faults on your business.
System layer monitoring: CPU, memory, disk, network, etc.

Intermediate component layer monitoring: Kafka, MySQL, Redis, etc.
Application layer monitoring: application services such as JVM, HTTP, and RPC.
Business monitoring: golden business metrics such as the number of logins and the number of orders.

Strengths

TMP has the following strengths over the open-source Prometheus:



Lighter, more stable, and more available.
Fully compatible with the open-source Prometheus ecosystem.
Free of manual setup, saving the development costs.
Highly integrated with TKE, saving the development costs of integrating with Kubernetes.
Integrated with Cloud Monitor's alarming system, saving the costs of developing alarm notifications.
Integrated with commonly used Grafana dashboards and alerting rule templates.

Use Cases

Last updated：2024-08-07 21:50:26

Integrated Monitoring

TMP provides the one-stop out-of-the-box Prometheus monitoring service, which is natively integrated with Grafana dashboards and Cloud Monitor alarms for various monitoring scenarios such as basic services, application layer, and container services.

Application Service Monitoring

Scenario 1

An application provides external API services but their quality information cannot be secured. TMP can be integrated according to the development language to monitor the access request volume, delay, and success rate of APIs in real time.

Scenario 2

TMP also detects service exceptions to help you understand which APIs an exception responds to, which servers an exception occurs on, and whether an exception occurs on individual servers or in the entire cluster.

Scenario 3

For Java applications, TMP can monitor the GC, memory, and thread status of individual servers to help you fully understand the internal status of JVM.

CVM Monitoring

If your service is deployed in CVM, you must modify the Prometheus scrape configuration almost every time the service is scaled. For this kind of scenario, with the aid of the tagging capability of the Tencent Cloud platform and the tag discovery capability of the Prometheus agent, you only need to properly associate tags with CVM instances to easily manage and monitor target objects, which helps you get rid of continuously updating the configuration manually; for example:
1. Service A is deployed on 2 CVM instances at the same time, and the instances are associated with the tag "service name: A";
2. The original number of CVM instances cannot meet the requirements of a business campaign, and 3 more instances should be added. At this time, you only need to associate the tag "service name: A" with these new instances, and then the agent will automatically discover them and actively scrape their monitoring metrics;
3. After the campaign is over, 3 CVM instances are removed, and the agent can automatically discover the instance deactivation and stop scraping their monitoring metrics.

Custom Monitoring

You can use TMP to customize the reported metric monitoring data so as to monitor internal status of applications or services, such as the number of processed requests and the number of orders. You can also monitor the processing duration of some core logic, such as requesting external services.

Strengths

Last updated：2024-08-07 21:50:34

Lightweight Service

Compared with the open-source Prometheus monitoring service, TMP features a more lightweight overall structure. You can use a TMP instance simply after creating it in Cloud Monitor, where an agent can complete data scrape by using less than 1 GB of memory.

High Stability and Reliability

TMP only occupies megabytes of resources, much fewer than the open-source Prometheus does. It also integrates Tencent Cloud storage services and its own replica capabilities to reduce the number of system interruptions and provide services with a higher availability.

Great Openness

TMP offers the out-of-the-box Grafana service. It also integrates a wealth of Kubernetes basic monitoring services and dashboards for common service monitoring, which can be quickly used after activation.

Low Costs

TMP provides the native Prometheus monitoring service. After you purchase a TMP instance, you can quickly integrate it with TKE to monitor services running on Kubernetes, which eliminates the costs of setup, OPS, and development.

High Scalability

TMP features an unlimited data storage capacity not restricted to local disks. It can be dynamically scaled based on Tencent Cloud's proprietary sharding and scheduling technologies to meet your elastic needs. It also supports CLB for better load balancing. This helps solve the pain points that the open-source Prometheus cannot be scaled horizontally.

High Compatibility

TMP is fully compatible with Prometheus protocols, support core APIs, custom multidimensional data models, flexible query language PromQL, and collection target discovery through dynamic scrape configuration or static configuration, so you can easily migrate to it for smooth access.

Concepts

Last updated：2024-08-07 21:50:40

This document describes the basic concepts involved in using the TMP service, so that you can check and understand related concepts.

Concept	Description
Exporter	Exporter is a component that collects monitoring data and provides data externally based on Prometheus monitoring specifications. There are currently hundreds of official or third-party tools available, see Exporters and Integrations.
Job	A collection of configurations for a set of targets. After the scrape interval is defined, access restrictions will will be applied to the scraping job for a given set of targets.
TMP instance	The logical unit for managing monitoring data collection and data storage analysis provided by TMP.
TMP probe	Kubernetes cluster deployed on either the user or Tencent Cloud side. It is responsible for automatic discovery of collection targets, collection metrics and remote writing to other libraries.
PromQL	The query language for the TMP service, which supports instant query and timespan query. It has a variety of built-in functions and operators, so raw data can be aggregated, sliced, predicted, and federated.
Target	The collection target to be scraped by the Prometheus Agent. The collection target either exposes its own operation and business metrics or serves as a proxy for exposing the operation and business metrics of a monitored object.
Alerting rule	The alert configuration of alerting rule formats in TMP, Which can be described by PromQL.
Label	An pair of key-value used to describe the metric.
Scrape configuration	One of the features in TMP, which can automatically discover collection targets without static configuration. It supports Kubernetes SD, Consul, Eureka, and other ways of service discovery. It also allows collection targets to be exposed via Service Monitor and Pod Monitor.
Recording rule	The recording rule capacity in TMP. With recording rule, raw data can be processed into new metrics through PromQL to improve query efficiency.
Integration Center	It integrates all the services supported by TMP. You can install the corresponding services as instructed on the page. After successful installation, you can view the monitoring data on the monitoring panel.
Alerting rule	It is used to define how to trigger and send an alert.

Cloud product monitor	TMP integrates the monitoring data of Tencent Cloud products. You can quickly install the Agent to view the monitoring data.
Metrics	Metrics are used to collect a series of labeled data exposed by the target, which can fully reflect the operation or business status of the monitored object.
TPS	The total number of reported data points per second. It is an important metric to measure the processing power of the system.
Series limit	The maximum number of metrics. The upper limit of series = (single metric × the dimension combination of the metric) × the number of metrics.

Use Limits

Last updated：2024-08-07 21:50:46

Instance Limits

Each instance can have up to 4.5 million series. Free trial instance is limited to 2 million series. If you need to adjust the limit for a paid instance, contact us. In the case of adequate resources, we will adjust the related limit for you properly.

Note:

A series consists of a metric name and label. The same metric name and labels form a unique series.

Custom Reporting Limits

If you use TMP's custom monitoring feature to monitor data, there will be the following limits on metrics (series with a unique `__name__`).

Data reporting must carry a metric name, i.e., the `__name__` label, which can contain only ASCII letters, characters, digits, underscores, and colons and must start with a letter and match the regex `[a-zA-Z_:][a-zA-Z0-9_:]*`. For more information, see Metric names and labels.

Each metric can have up to 32 labels.

The label name can contain only ASCII letters, digits, and underscores. It must match the regex `[a-Za-Z_][a-Za-Z0-9_]*`. Labels beginning with `__` are reserved for internal use.

The label name and label value can contain up to 1,024 and 2,048 characters respectively.

Under the same metric, the dimension combinations of labels cannot exceed 100,000. When the histogram has many buckets, the histogram-type metrics cannot be adjusted.

Total number of data points reported per second: A paid instance cannot exceed 300,000, and a free trial instance cannot exceed 100,000.

Note

The role of labels: In Prometheus, data is stored as time series, which are uniquely identified by the metric name and a series of labels (key-value pairs). Different labels represent different time series, so you can query the specified data by label. The more labels you add, the finer the query dimension.

Prometheus Query Limits

To ensure the query efficiency and better user experience, Prometheus query has the following limits (which don't apply to metadata such as queries about labels and don't affect the Grafana metrics browser feature).
The number of time series involved in a single query cannot exceed 100,000.
The amount of data involved in a single query cannot exceed 100 MB.
There is no limit on the query frequency, but if the concurrency exceeds 15, there may be a certain queuing delay in slower large queries (the probability is low though). Large queries with a time span of more than two weeks will have a higher delay.
The above limits also apply to alarm rules and recording rules. We recommend that you limit the query scope based on your business scenario or appropriately split queries in other ways. You can also use the method of splitting first and then aggregating, such as aggregating recorded results again.

Other Configurations Limits

Configuration limits:
The maximum of alarm rules can be configured for each instance: 150.
The maximum of recording rules can be configured for each instance: 150.

Features

Last updated：2024-08-07 21:50:52

Monitoring Object Access

Feature	Description
Instance creation	You can create a TMP instance in multiple region.
Integration center	The integration center supports quick installation and custom access of various components. After successful installation, you can view monitoring data in Grafana.
Health check	Health check detects the service connectivity on a regular basis to monitor the service health, helping you stay up to date with the service health in real time and promptly discover exceptions to improve the SLA.
Custom monitoring	You can customize monitoring data reporting and monitor key business metrics, such as the number of requests, orders placed, and time consumed requesting external services.

Monitoring Metrics for Collection

Feature	Description
Scrape configuration	ServiceMonitor: A built-in scrape configuration feature in TMP, which is automatically enabled when connected. Currently, the default objects for scrape metrics collection are pods contained in all namespaces under the Kubernetes cluster. ServiceMonitor: It collects the monitoring data in the corresponding endpoints of services based on Prometheus Operator in the K8s ecosystem. PodMonitor: It collects the corresponding monitoring data in pods based on Prometheus Operator in the K8s ecosystem.
Targets	You can visually view the target being scraped through Targets and check whether the scraping status is normal. You can also view the metrics exposed in the target.

Monitoring Data

Feature	Description

Getting Remote Write address	The Remote Write feature can store TMP data as a remote database. You can use the Remote Write address to store the monitoring data of self-built Prometheus in the TMP instance to achieve remote storage, and visualize it in the same Grafana system.
Recording rule	A recording rule allows you to calculate some commonly used or complex metrics in advance and then store the calculated data in new data metrics. In this way, querying the calculated data will be faster than querying the original data. This can solve the problems of complicated user configuration and slow query.

Monitoring Data Display

Feature	Description
Grafana	There are numerous embedded Grafana dashboards available. You can also customize one if necessary. Plugins that are often used on the Grafana official website are also preset, and you can install them quickly in the console.
Instance monitoring	It supports the monitoring of TMP instance status and usage, including TMP instance storage, alert sending, Grafana requests and the number of dashboards, helping you check the usage of TMP instances in real time.
HTTP API	This API is used to get the TMP data address. You can use this address to connect the monitoring data of the TMP instance to the self-built Grafana dashboard for data display, or perform secondary development of the TMP monitoring data.

Alerts

Feature	Description
Creating alerting rules	There are numerous preset alerting rules available. You can also customize one for a specific monitoring object. TMP integrates the alarm notification template of Tencent Cloud Observability Platform (TCOP), which notifies you to take measures in time when certain metrics are abnormal.
Managing alerting rules	You can perform operations such as enabling, disabling, editing, and deleting alerting rules. You can also quickly import other instance alerts.

Service Regions

Last updated：2024-08-08 09:55:01

Note:

Service regions refer to locations of physical data centers or servers. You can regard them as the regions or countries where a service or resource is available. Once a resource is successfully created in a data center in a specific region, the region of the resource usually cannot be changed.

Regions and AZs supported by TMP are as follows:

Region	AZ
South China (Guangzhou) ap-guangzhou	Guangzhou Zone 3 ap-guangzhou-3
	Guangzhou Zone 4 ap-guangzhou-4
East China (Shanghai) ap-shanghai	Shanghai Zone 2 ap-shanghai-2
	Shanghai Zone 3 ap-shanghai-3
	Shanghai Zone 4 ap-shanghai-4
	Shanghai Zone 5 ap-shanghai-5
Hong Kong (China), Macao (China), and Taiwan (China) (Hong Kong (China)) ap-hongkong	Hong Kong (China) Zone 1 ap-hongkong-1
	Hong Kong (China) Zone 2 ap-hongkong-2
	Hong Kong (China) Zone 3 ap-hongkong-3
North China (Beijing) ap-beijing	Beijing Zone 3 ap-beijing-3
	Beijing Zone 4 ap-beijing-4
	Beijing Zone 5 ap-beijing-5
	Beijing Zone 6 ap-beijing-6
	Beijing Zone 7 ap-beijing-7
Southwest China (Chengdu) ap-chengdu	Chengdu Zone 1 ap-chengdu-1
	Chengdu Zone 2 ap-chengdu-2
Southwest China (Chongqing) ap-chongqing	Chongqing Zone 1 ap-chongqing-1
East China (Nanjing) ap-nanjing	Nanjing Zone 1 ap-nanjing-1

	Nanjing Zone 2 ap-nanjing-2
North America (Toronto) na-Toronto	Toronto Zone 1 na-Toronto-1
West US (Silicon Valley) na-siliconvalley	Silicon Valley Zone 1 na-siliconvalley-1
	Silicon Valley Zone 2 na-siliconvalley-2
Europe (Frankfurt) eu-frankfurt	Frankfurt Zone 1 eu-frankfurt-1
East US (Virginia) na-ashburn	Virginia Zone 1 na-ashburn-1
	Virginia Zone 2 na-ashburn-2
South America (Sao Paulo) sa-saopaulo	Sao Paulo Zone 1 sa-saopaulo-1
Southeast Asia (Singapore) ap-singapore	Singapore Zone 1 ap-singapore-1
	Singapore Zone 2 ap-singapore-2
	Singapore Zone 3 ap-singapore-3
	Singapore Zone 4 ap-singapore-4
Asia Pacific (Seoul) ap-seoul	Seoul Zone 1 ap-seoul-1
Asia Pacific (Mumbai) ap-bombay	Mumbai Zone 1 ap-bombay-1
	Mumbai Zone 2 ap-bombay-2
Asia Pacific (Bangkok) ap-bangkok	Bangkok Zone 2 ap-Bangkok-2
Asia Pacific (Tokyo) ap-tokyo	Tokyo Zone 1 ap-tokyo-1
	Tokyo Zone 2 ap-tokyo-2
Southeast Asia Pacific (Jakarta) spa-jakarta	Jakarta Zone 1 spa-Jakarta-1
	Jakarta Zone 2 spa-Jakarta-2

Access Guide
Scrape Configuration Description

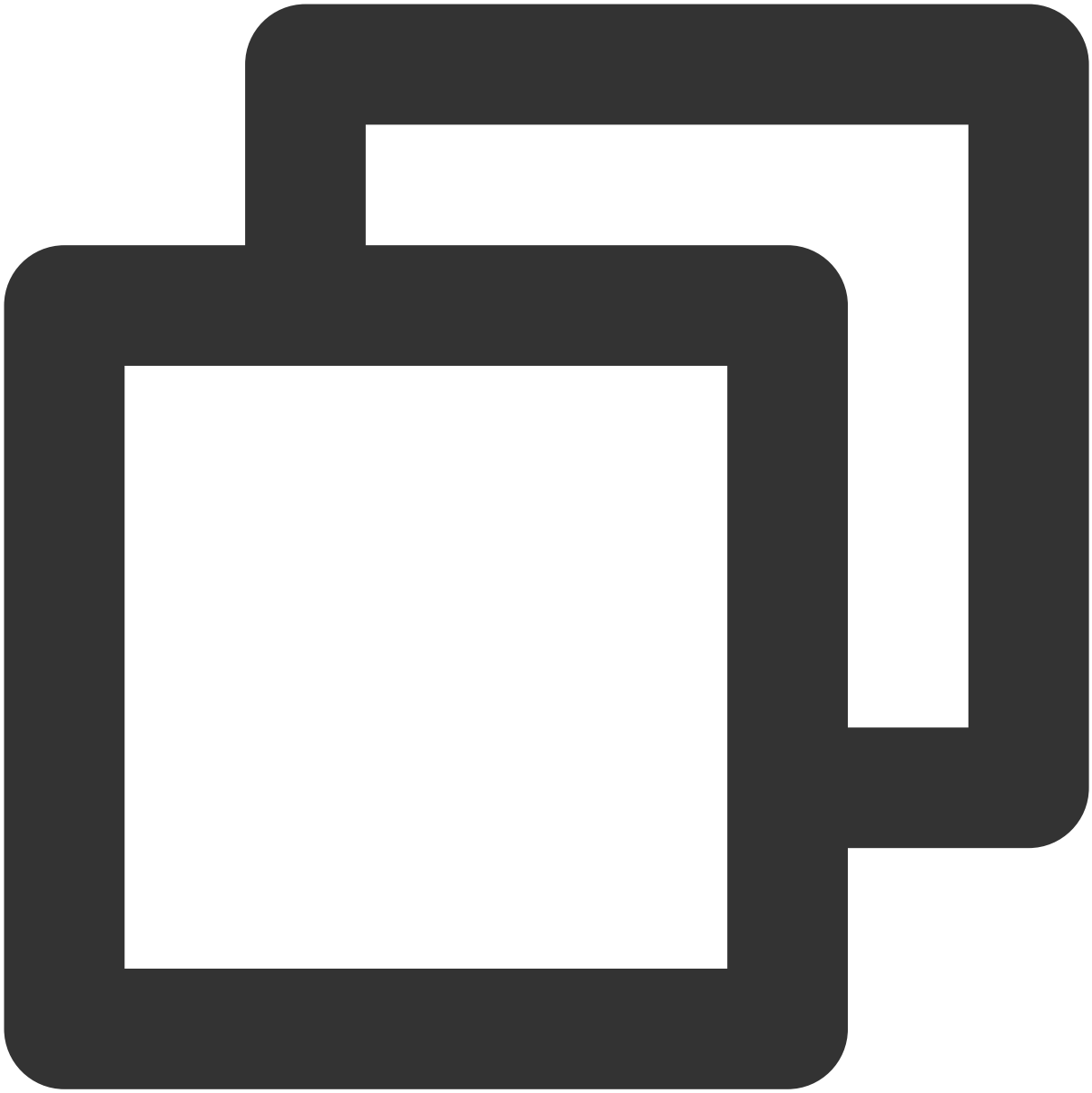
Last updated：2024-07-23 18:10:38

Overview

Prometheus mainly uses PULL to scrape the monitoring APIs exposed by the target service; therefore, you need to configure the corresponding scrape task to request the monitoring data and write it into the storage provided by Prometheus. Currently, Prometheus provides the configurations of the following tasks:
Native job configuration: the native scrape job configuration of Prometheus is provided.
PodMonitor: It collects the corresponding monitoring data in Pods based on Prometheus Operator in the K8s ecosystem.
ServiceMonitor: It collects the monitoring data in the corresponding Endpoints of Services based on Prometheus Operator in the K8s ecosystem.
Note:
Configuration items in `[]` are optional.

Native job configuration

The relevant configuration items are as detailed below:



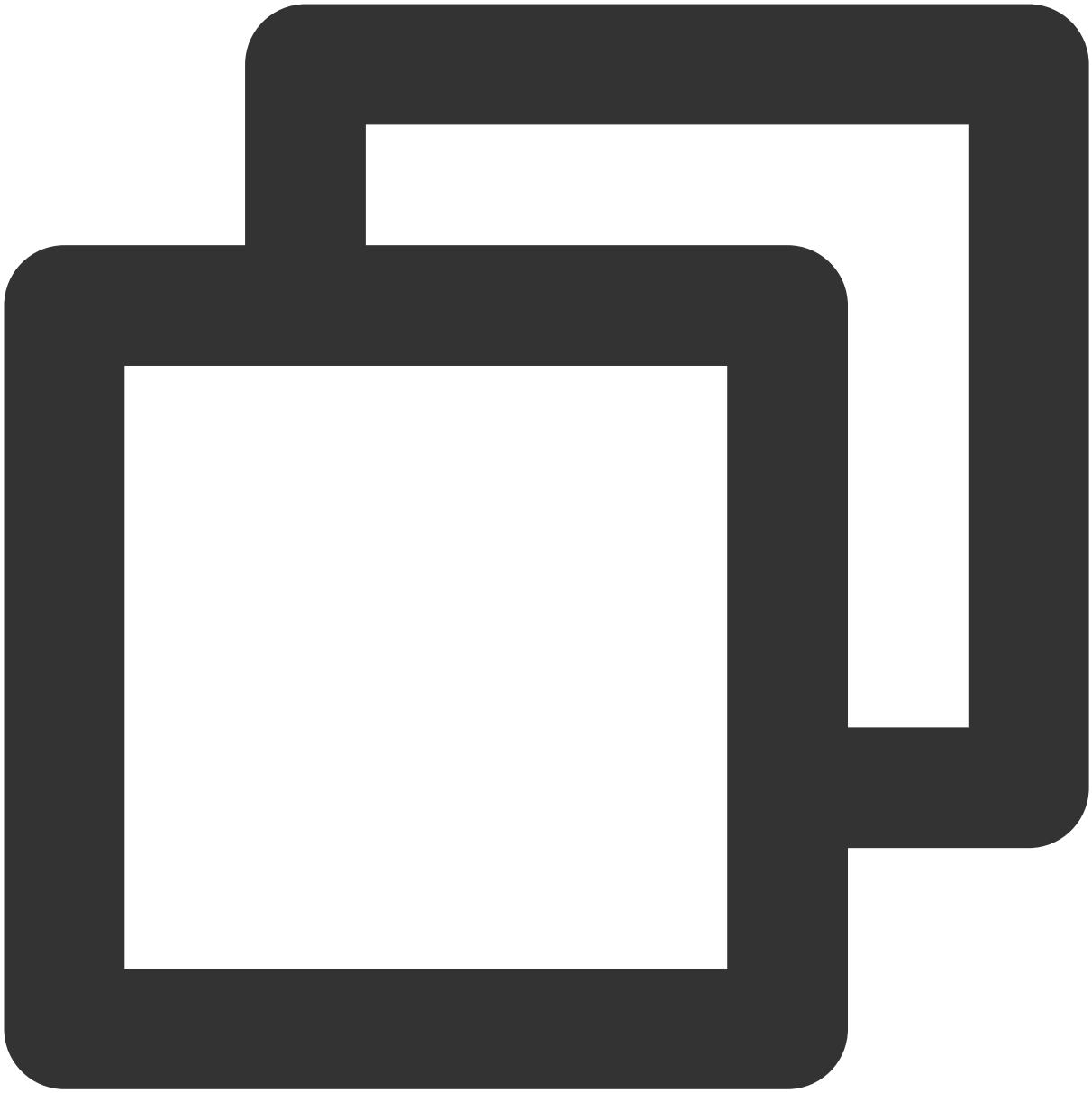
```
# Scrape task name. `label(job=job_name)` will be added to the corresponding metric
job_name: <job_name>

# Scrape task interval
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# Scrape request timeout period
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# Scrape task request URI path
```

```
[ metrics_path: <path> | default = /metrics ]

# Solve the conflict between the scraped label and the label added to Prometheus on
# true: Retain the scraped label and ignore the label conflicting with Prometheus o
# false: Add `exported_<original-label>` before the scraped label to add the label
[ honor_labels: <boolean> | default = false ]

# Whether to use the time generated on the scrape target
# true: Use the time on the target
# false: Directly ignore the time on the target
[ honor_timestamps: <boolean> | default = true ]

# Scrape protocol: HTTP or HTTPS
[ scheme: <scheme> | default = http ]

# URL parameter of the scrape request
params:
  [ <string>: [<string>, ...] ]

# Use `basic_auth` to set `Authorization` in the scrape request header. `password`
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Use `bearer_token` to set `Authorization` in the scrape request header. `bearer_t
[ bearer_token: <secret> ]

# Use `bearer_token` to set `Authorization` in the scrape request header. `bearer_t
[ bearer_token_file: <filename> ]

# Specify whether the scrape connection passes through a TLS secure channel and con
tls_config:
  [ <tls_config> ]

# Use a proxy service to scrape metrics on the target and enter the corresponding p
[ proxy_url: <string> ]

# Use static configuration to specify the target. For more information, see the des
static_configs:
  [ - <static_config> ... ]

# Set the CVM scrape configuration. For more information, see the description below
cvm_sd_configs:
  [ - <cvm_sd_config> ... ]

# After scraping the data, change the label on the target through the relabeling me
```

```
# For more information on `relabel_config`, see the description below
relabel_configs:
  [ - <relabel_config> ... ]

# After the data is scraped and before it is written, use the relabeling mechanism
# For more information on `relabel_config`, see the description below
metric_relabel_configs:
  [ - <relabel_config> ... ]

# Limit of data points in one scrape. 0: no limit. Default value: 0
[ sample_limit: <int> | default = 0 ]

# Limit of targets in one scrape. 0: no limit. Default value: 0
[ target_limit: <int> | default = 0 ]
```

`static_config` configuration

The relevant configuration items are as detailed below:



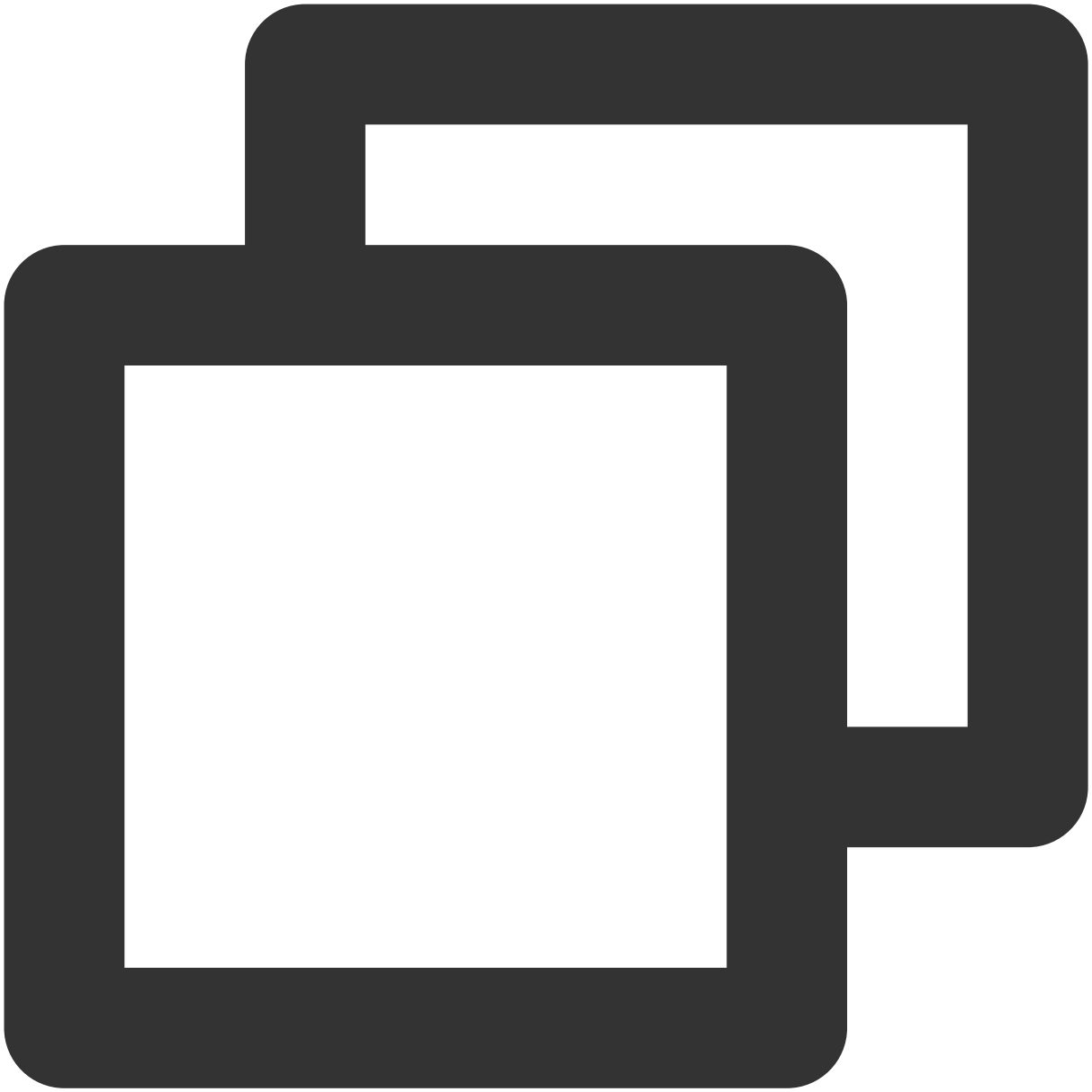
```
# Specify the corresponding target host value, such as `ip:port`
targets:
  [ - '<host>' ]

# Add the corresponding label to all targets, which is similar to a global label
labels:
  [ <labelname>: <labelvalue> ... ]
```

`cvm_sd_config` configuration

CVM scrape configuration uses TencentCloud API to automatically get the CVM instance list, and the CVM instance's private IP is used by default. Scrape configuration will generate the following meta labels, which can be used in relabeling configuration.

Label	Description
__meta_cvm_instance_id	Instance ID
__meta_cvm_instance_name	Instance name
__meta_cvm_instance_state	Instance status
__meta_cvm_instance_type	Instance model
__meta_cvm_OS	Instance OS
__meta_cvm_private_ip	Private IP
__meta_cvm_public_ip	Public IP
__meta_cvm_vpc_id	VPC ID
__meta_cvm_subnet_id	Subnet ID
__meta_cvm_tag_<tagkey>	Instance tag value
__meta_cvm_region	Instance region
__meta_cvm_zone	Instance AZ

CVM scrape configuration description:



```
# Tencent Cloud region. For the region list, visit https://cloud.tencent.com/docume
region: <string>

# Custom endpoint.
[ endpoint: <string> ]

# Credential information for accessing TencentCloud API. If it is not set, the valu
# Leave it empty if you use a CVM scrape task in **Integration Center** for configu
[ secret_id: <string> ]
[ secret_key: <secret> ]
```

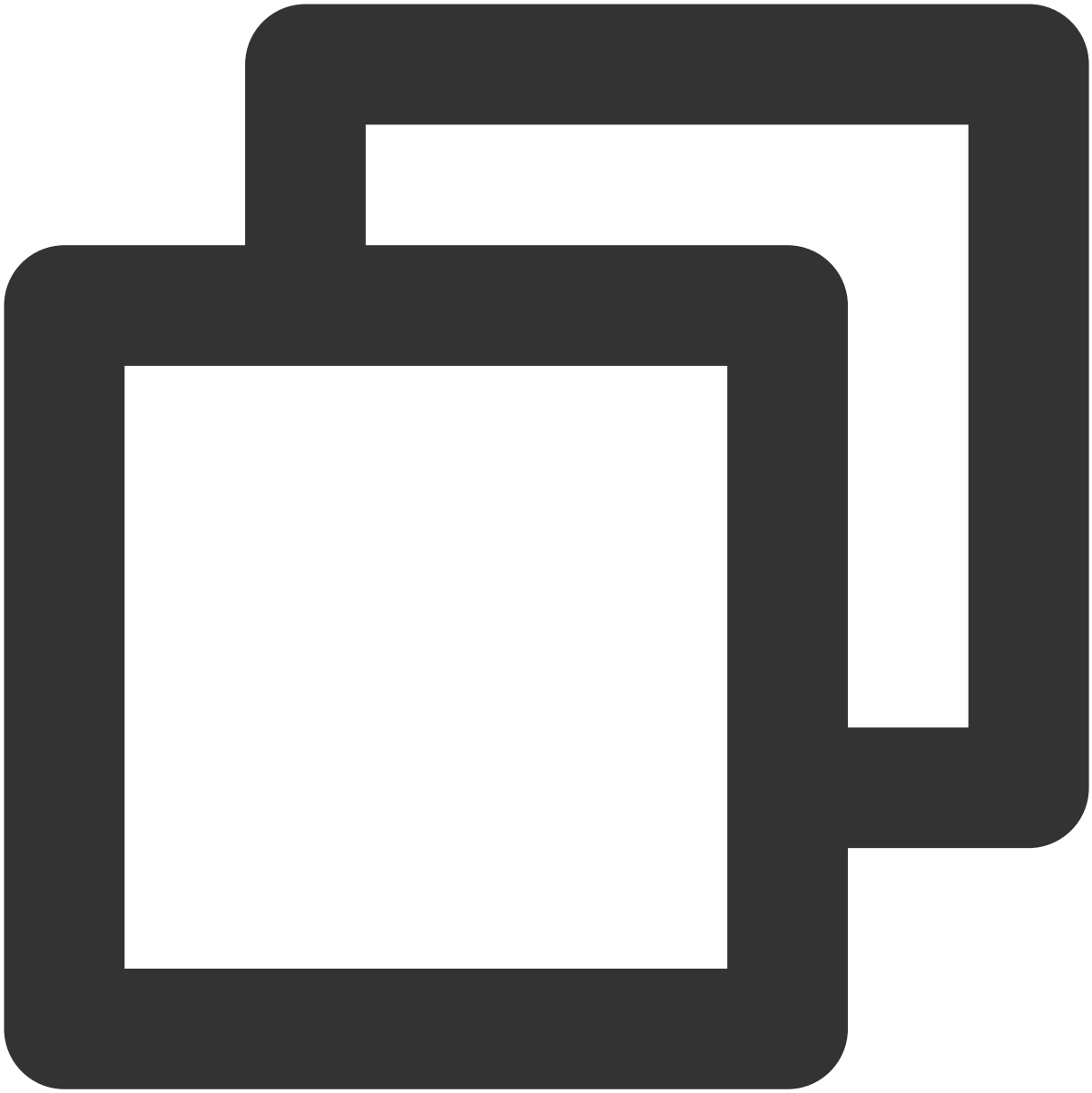
```
# CVM list refresh interval
[ refresh_interval: <duration> | default = 60s ]

# Port for scraping metrics
ports:
  - [ <int> | default = 80 ]

# CVM list filtering rule. For more information on the supported filtering rules, v
filters:
  [ - name: <string>
      values: <string>, [...] ]
```

Note:

If a CVM scrape task in **Integration Center** is used to configure `cvm_sd_configs`, the integration automatically uses the preset role authorization of the service for security considerations. You don't need to manually enter the `secret_id`, `secret_key`, and `endpoint` parameters.

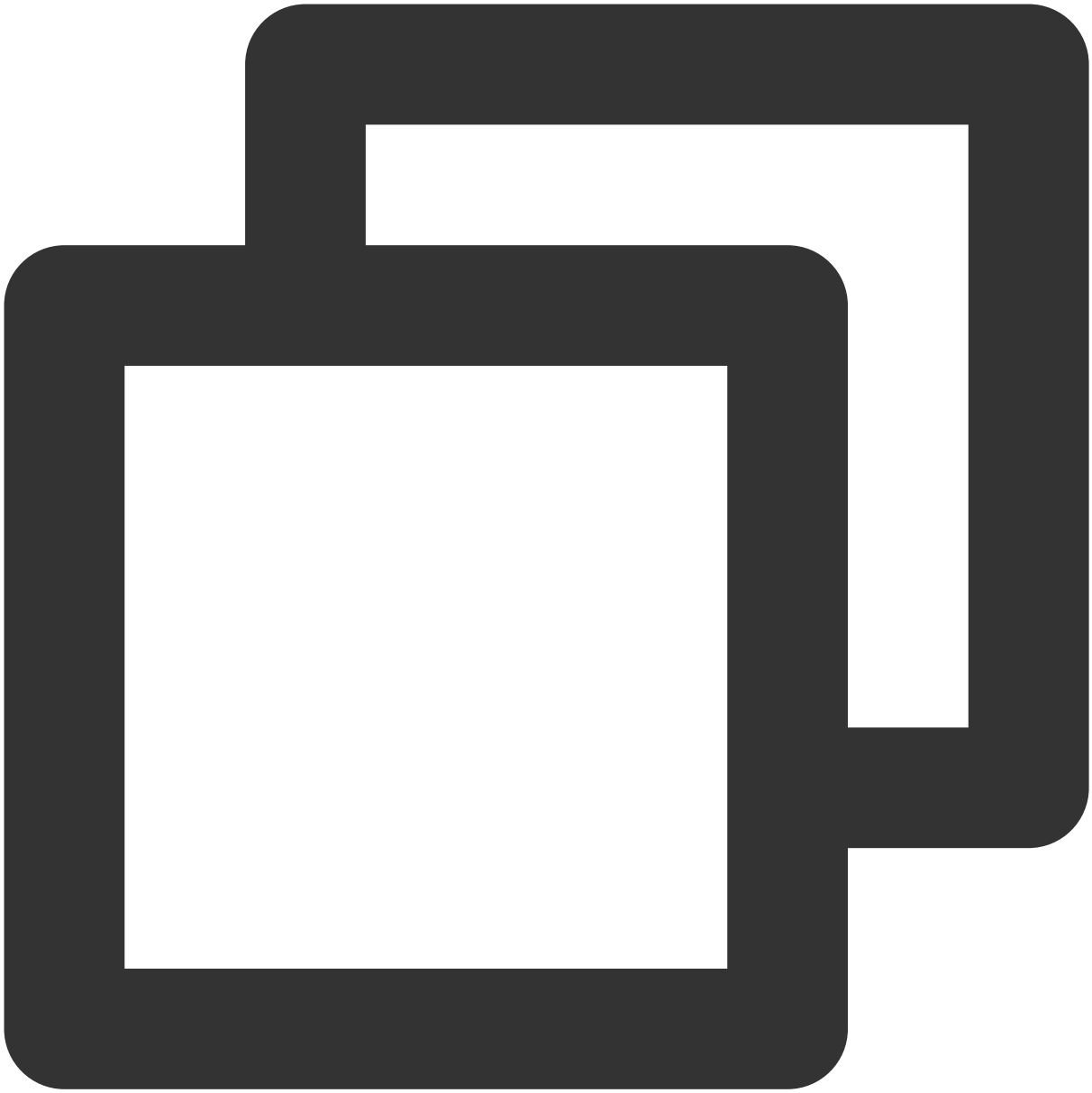
Sample

Static configuration



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

CVM scrape configuration

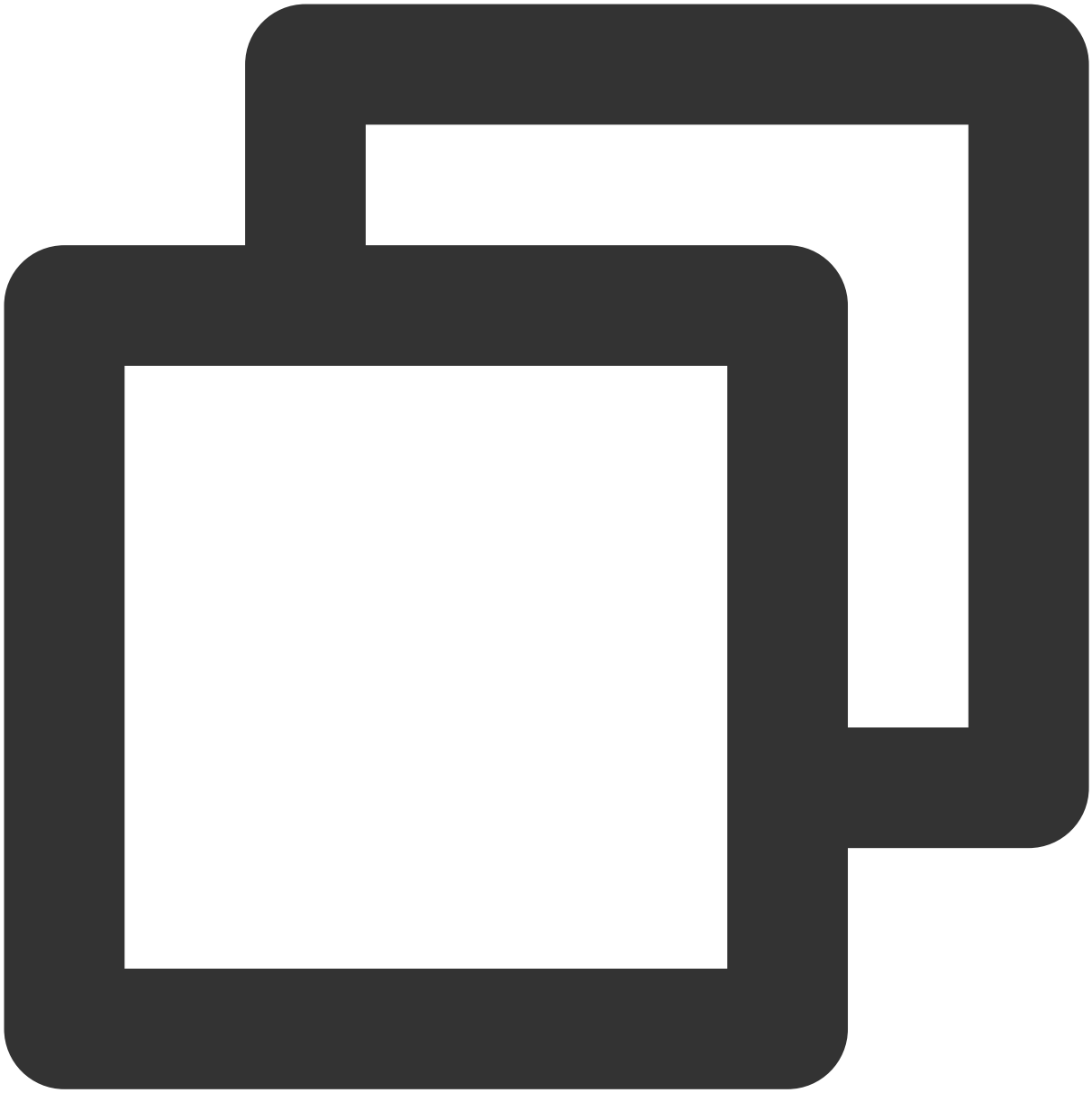


```
job_name: demo-monitor
cvm_sd_configs:
- region: ap-guangzhou
  ports:
  - 8080
  filters:
  - name: tag:service
    values:
    - demo
relabel_configs:
- source_labels: [__meta_cvm_instance_state]
```

```
    regex: RUNNING
    action: keep
  - regex: __meta_cvm_tag_(.*)
    replacement: $1
    action: labelmap
  - source_labels: [__meta_cvm_region]
    target_label: region
    action: replace
```

PodMonitor

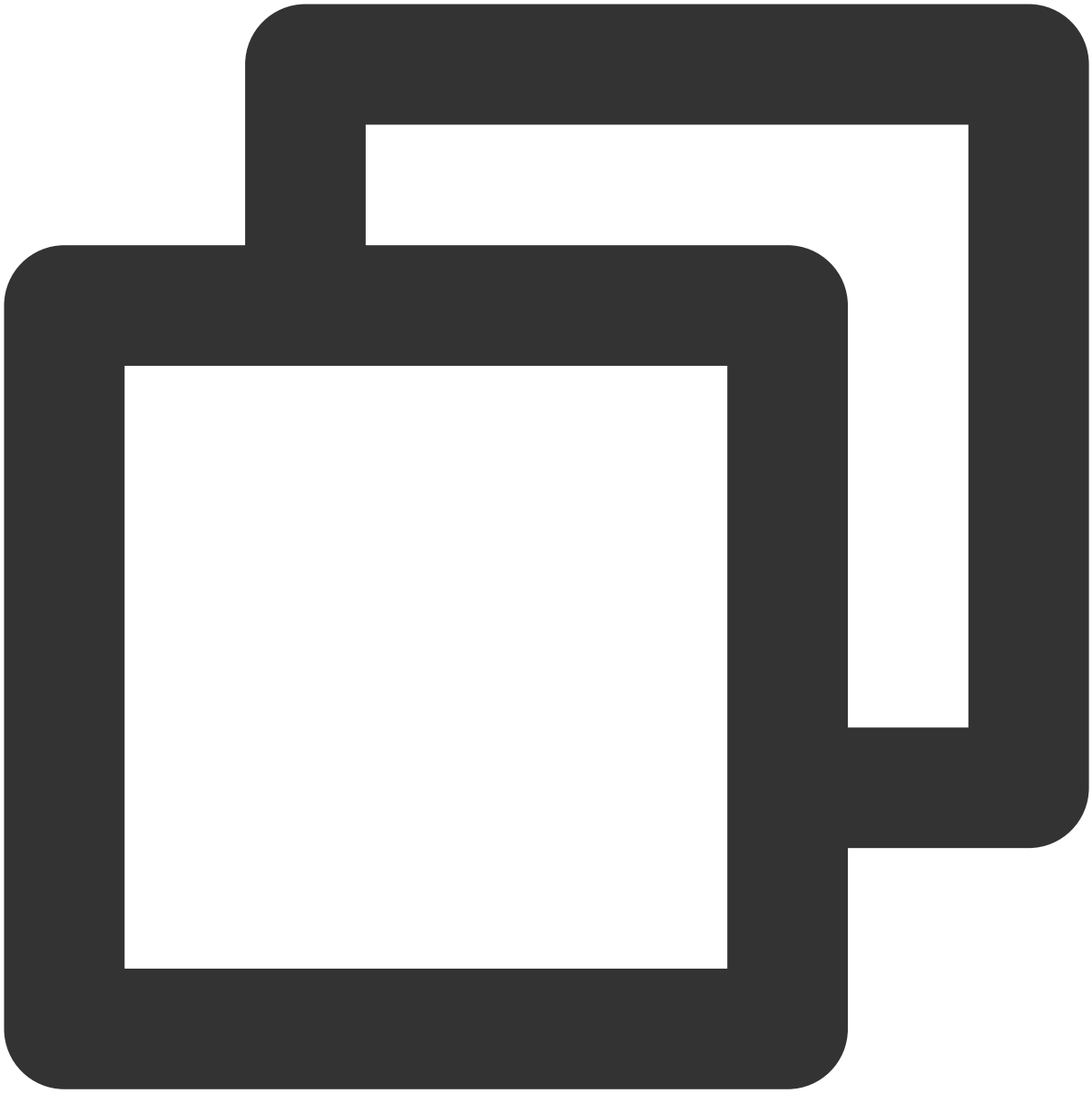
The relevant configuration items are as detailed below:



```
# Prometheus Operator CRD version
apiVersion: monitoring.coreos.com/v1
# Corresponding K8s resource type, which is PodMonitor here
kind: PodMonitor
# Corresponding K8s metadata. Here, only the `name` is concerned. If `jobLabel` is
metadata:
  name: redis-exporter # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
# Describe the selection of the scrape target Pod and the configuration of the scra
spec:
  # Enter the target Pod label. PodMonitor will use the corresponding value as the
```

```
  # If Pod YAML configuration is to be viewed, use the value in `pod.metadata.label
  # If `Deployment/Daemonset/Statefulset` is to be viewed, use `spec.template.metad
  [ jobLabel: string ]
  # Add the label on the corresponding Pod to the target label
  [ podTargetLabels: []string ]
  # Limit of data points in one scrape. 0: no limit. Default value: 0
  [ sampleLimit: uint64 ]
  # Limit of targets in one scrape. 0: no limit. Default value: 0
  [ targetLimit: uint64 ]
  # Configure the Prometheus HTTP port to be exposed and scraped. You can configure
  podMetricsEndpoints:
  [ - <endpoint_config> ... ] # For more information, see the endpoint description
  # Select the namespace where the Pod to be monitored resides. If it is not specif
  [ namespaceSelector: ]
    # Whether to select all namespaces
    [ any: bool ]
    # List of namespace to be selected
    [ matchNames: []string ]
  # Enter the label of the Pod to be monitored to locate the target Pod. For more i
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

Sample

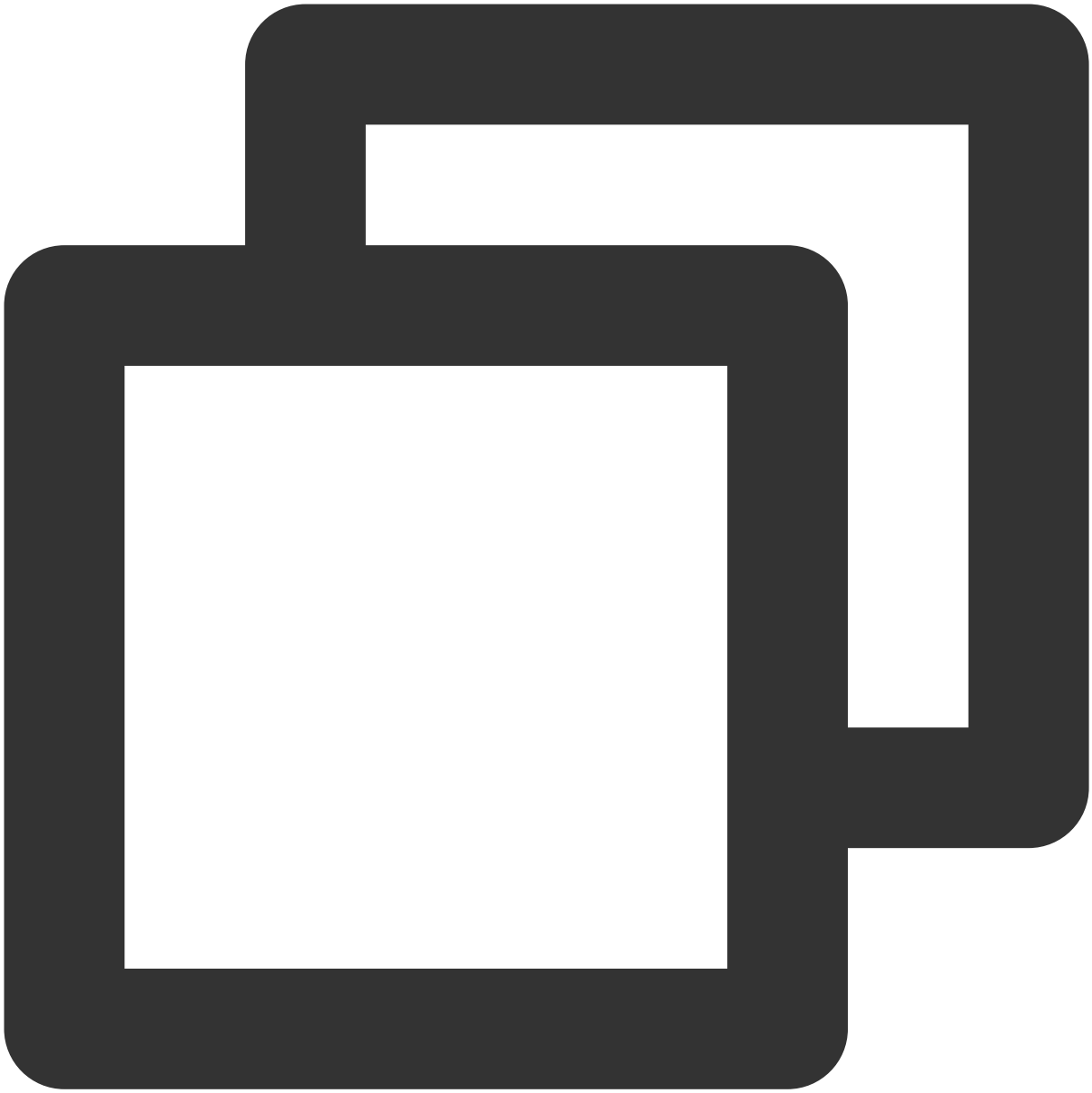


```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port  # Enter the name of the corresponding port of the Promethe
    path: /metrics  # Enter the value of the corresponding path of the Prometheus
    relabelings:
```

```
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # Change it to the corresponding Redis instance I
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # Change it to the corresponding Redis instance IP
  namespaceSelector:   # Select the namespace where the Pod to be monitored resid
    matchNames:
    - redis-test
  selector:    # Enter the label value of the Pod to be monitored to locate the t
    matchLabels:
      k8s-app: redis-exporter
```

ServiceMonitor

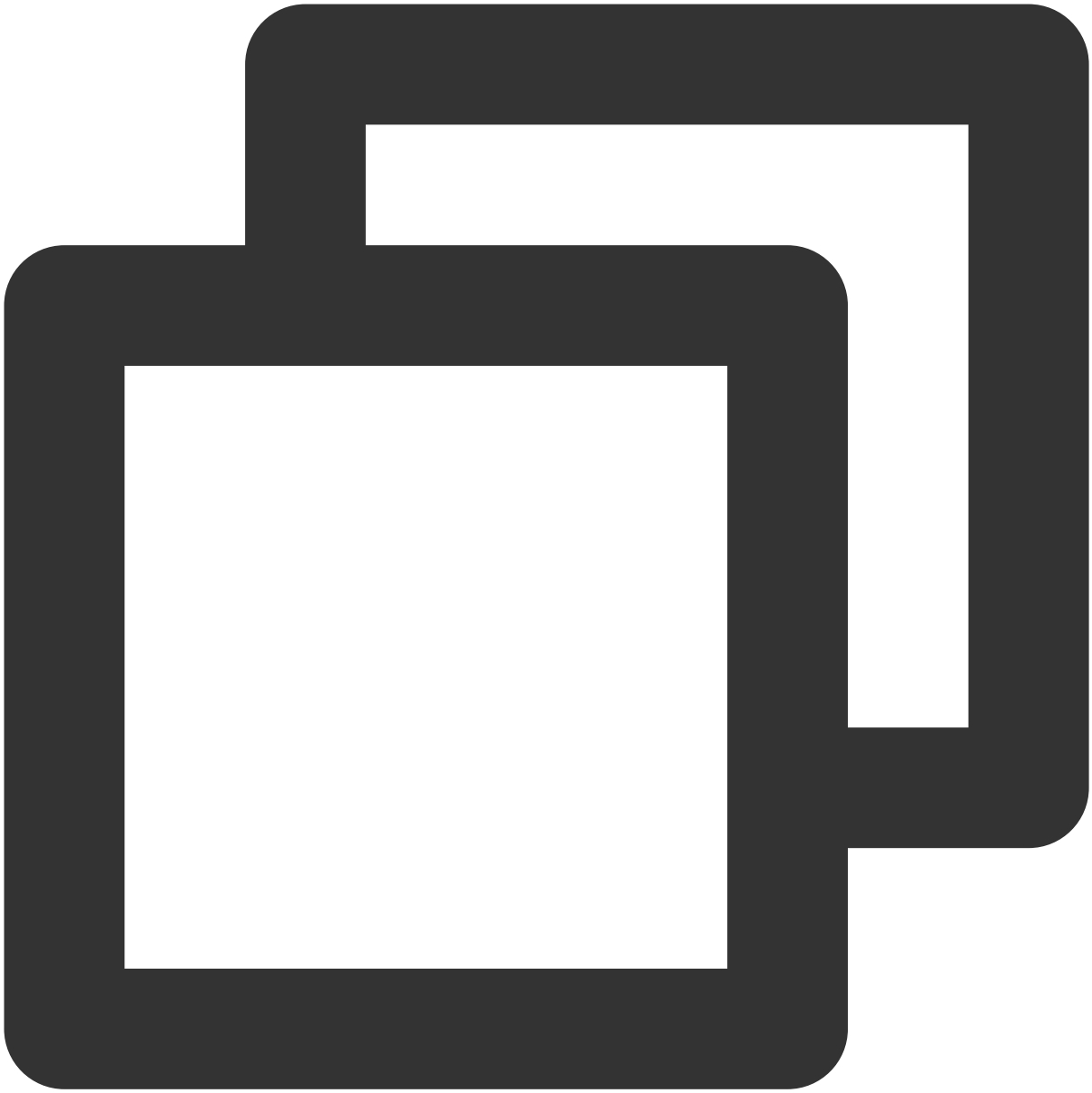
The relevant configuration items are as detailed below:



```
# Prometheus Operator CRD version
apiVersion: monitoring.coreos.com/v1
# Corresponding K8s resource type, which is ServiceMonitor here
kind: ServiceMonitor
# Corresponding K8s metadata. Here, only the `name` is concerned. If `jobLabel` is
metadata:
  name: redis-exporter # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
# Describe the selection of the scrape target Pod and the configuration of the scra
spec:
```

```
  # Enter the target Pod label (metadata/labels). ServiceMonitor will use the corre
  [ jobLabel: string ]
  # Add the label on the corresponding Service to the target label
  [ targetLabels: []string ]
  # Add the label on the corresponding Pod to the target label
  [ podTargetLabels: []string ]
  # Limit of data points in one scrape. 0: no limit. Default value: 0
  [ sampleLimit: uint64 ]
  # Limit of targets in one scrape. 0: no limit. Default value: 0
  [ targetLimit: uint64 ]
  # Configure the Prometheus HTTP port to be exposed and scraped. You can configure
  endpoints:
  [ - <endpoint_config> ... ] # For more information, see the endpoint description
  # Select the namespace where the Pod to be monitored resides. If it is not specif
  [ namespaceSelector: ]
    # Whether to select all namespaces
    [ any: bool ]
    # List of namespace to be selected
    [ matchNames: []string ]
  # Enter the label of the Pod to be monitored to locate the target Pod. For more i
  selector:
    [ matchExpressions: array ]
      [ example: - {key: tier, operator: In, values: [cache]} ]
    [ matchLabels: object ]
      [ example: k8s-app: redis-exporter ]
```

Sample

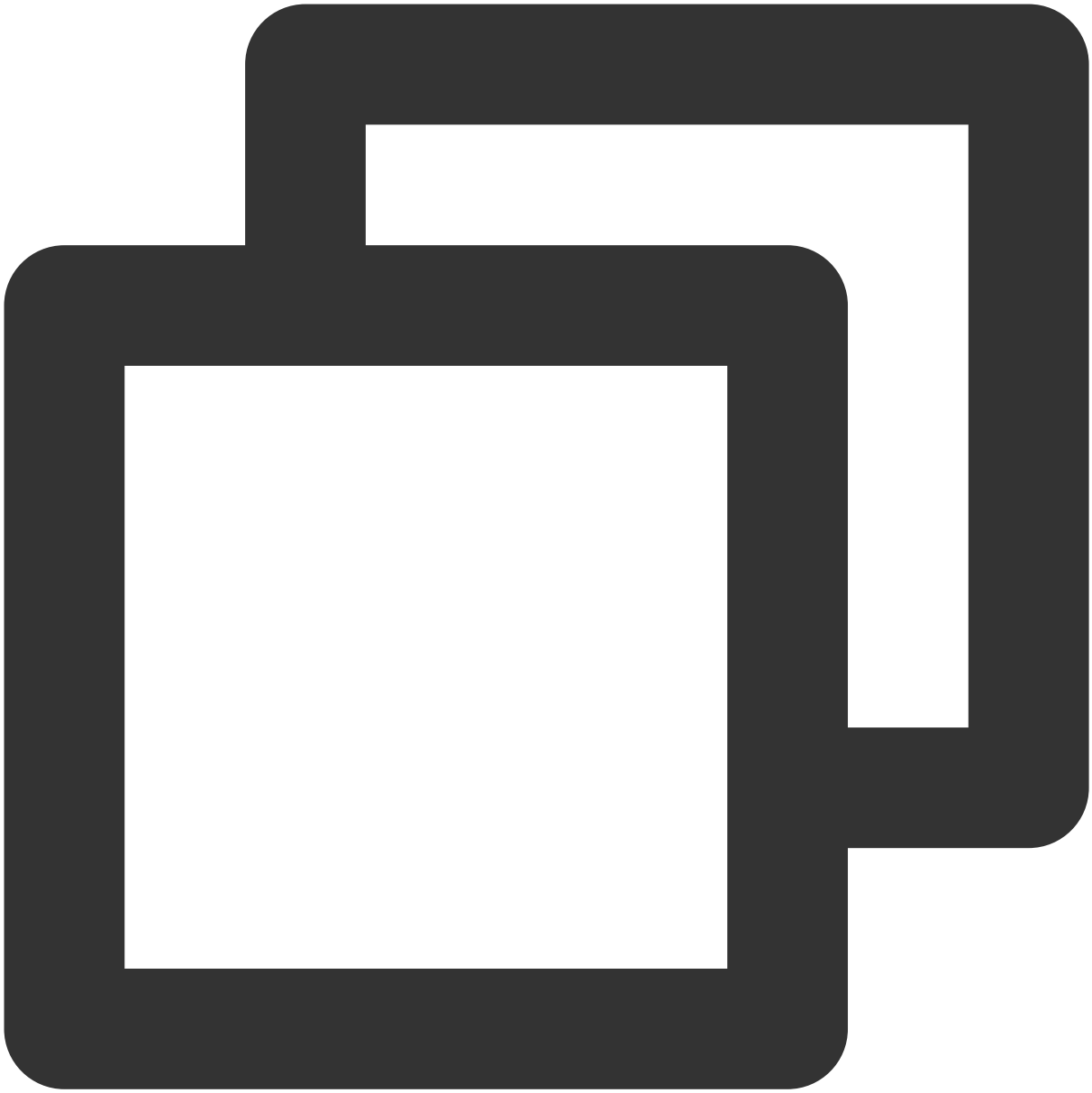


```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: go-demo    # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  endpoints:
  - interval: 30s
    # Enter the name of the corresponding port of the Prometheus exporter in the
    port: 8080-8080-tcp
    # Enter the value of the corresponding path of the Prometheus exporter. If it
```

```
    path: /metrics
    relabelings:
    # ** There must be a label named `application`. Here, suppose that K8s has a
    # Use the `replace` action of `relabel` to replace it with `application`
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # Select the namespace where the Service to be monitored resides
  namespaceSelector:
    matchNames:
    - golang-demo
  # Enter the label value of the Service to be monitored to locate the target Ser
  selector:
    matchLabels:
      app: golang-app-demo
```

endpoint_config configuration

The relevant configuration items are as detailed below:

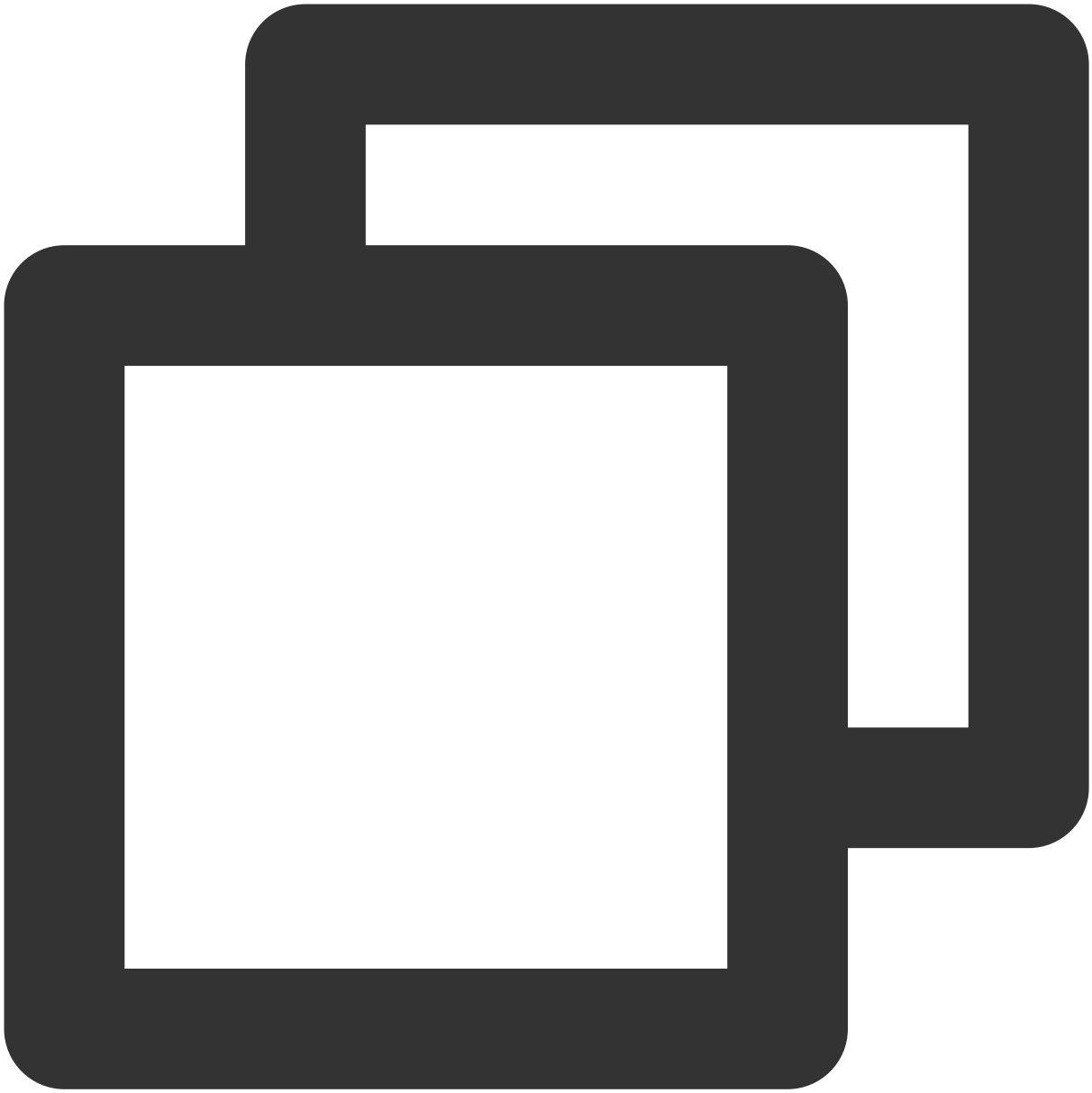


```
# Corresponding port name. Note that it is not the port number here. Default value:
# ServiceMonitor: `Service>spec/ports/name`
# PodMonitor description:
#   If Pod YAML configuration is to be viewed, use the value in `pod.spec.container
# If `Deployment/Daemonset/Statefulset` is to be viewed, use `spec.template.spec.co
[ port: string | default = 80]
# Scrape task request URI path
[ path: string | default = /metrics ]
# Scrape protocol: HTTP or HTTPS
[ scheme: string | default = http]
# URL parameter of the scrape request
```

```
[ params: map[string][]string]
# Scrape task interval
[ interval: string | default = 30s ]
# Scrape task timeout period
[ scrapeTimeout: string | default = 30s]
# Specify whether the scrape connection passes through a TLS secure channel and con
[ tlsConfig: TLSConfig ]
# Read the value of the bearer token through the corresponding file and add it to t
[ bearerTokenFile: string ]
# You can use the corresponding K8s secret key to read the bearer token. Note that
[ bearerTokenSecret: string ]
# Solve the conflict between the scraped label and the label added to Prometheus on
# true: Retain the scraped label and ignore the label conflicting with Prometheus o
# false: Add `exported_<original-label>` before the scraped label to add the label
[ honorLabels: bool | default = false ]
# Whether to use the time generated on the scrape target
# true: Use the time on the target
# false: Directly ignore the time on the target
[ honorTimestamps: bool | default = true ]
# `basic auth` authentication information. Enter the corresponding K8s secret key v
[ basicAuth: BasicAuth ]
# Use a proxy service to scrape metrics on the target and enter the corresponding p
[ proxyUrl: string ]
# After scraping the data, change the label on the target through the relabeling me
# For more information on `relabel_config`, see the description below
relabelings:
[ - <relabel_config> ...]
# After the data is scraped and before it is written, use the relabeling mechanism
# For more information on `relabel_config`, see the description below
metricRelabelings:
[ - <relabel_config> ...]
```

relabel_config configuration

The relevant configuration items are as detailed below:



```
# Specify which labels are to be taken from the original labels for relabeling. The
# The corresponding configuration item for PodMonitor/ServiceMonitor is `sourceLabe
[ source_labels: '[' <labelname> [, ...] ']' ]
# Define the separator symbol for concatenating the labels to be relabeled. Default
[ separator: <string> | default = ; ]

# If `action` is ` replace` or `hashmod`, you need to use the `target_label` to spe
# The corresponding configuration item for PodMonitor/ServiceMonitor is `targetLabe
[ target_label: <labelname> ]
```

```
# Regex for regular match of the values of source labels
[ regex: <regex> | default = (.*) ]

# Calculate the modulus of the MD5 value of the source label. The modulo operation
[ modulus: <int> ]

# If `action` is `replace`, use `replacement` to define the expression to be replac
[ replacement: <string> | default = $1 ]

# Perform an action based on the value matched by the regex. Valid values of `actio
# replace: Replace the matched value with that defined in `replacement` if the rege
# keep: Drop the value if the regex has no matches
# drop: Drop the value if the regex has any match
# hashmod: Calculate the modulus of the MD5 value of the source label based on the
# labelmap: Use `replacement` to replace the corresponding label name if the regex
# labeldrop: Delete the corresponding label name if the regex has any match
# labelkeep: Delete the corresponding label name if the regex has no matches
[ action: <relabel_action> | default = replace ]
```

Custom Monitoring

Last updated：2024-08-07 21:53:55

Overview

You can use TMP to customize the reported metric monitoring data so as to monitor internal status of applications or services, such as the number of processed requests and the number of orders. You can also monitor the processing duration of some core logic, such as requesting external services.
This document uses Go as an example to describe how to use TMP to customize reported metrics, visualization, and alerting.

Supported Programming Languages

Official SDKs from the native Prometheus community:
Go
Java or Scala
Python
Ruby
Third-Party SDKs for other programming languages:
Bash
C
C++
Common Lisp
Dart
Elixir
Erlang
Haskell
Lua for NGINX
Lua for Tarantool
.NET/C#
Node.js
Perl
PHP
R
Rust

For more information, please see CLIENT LIBRARIES.

Data Model

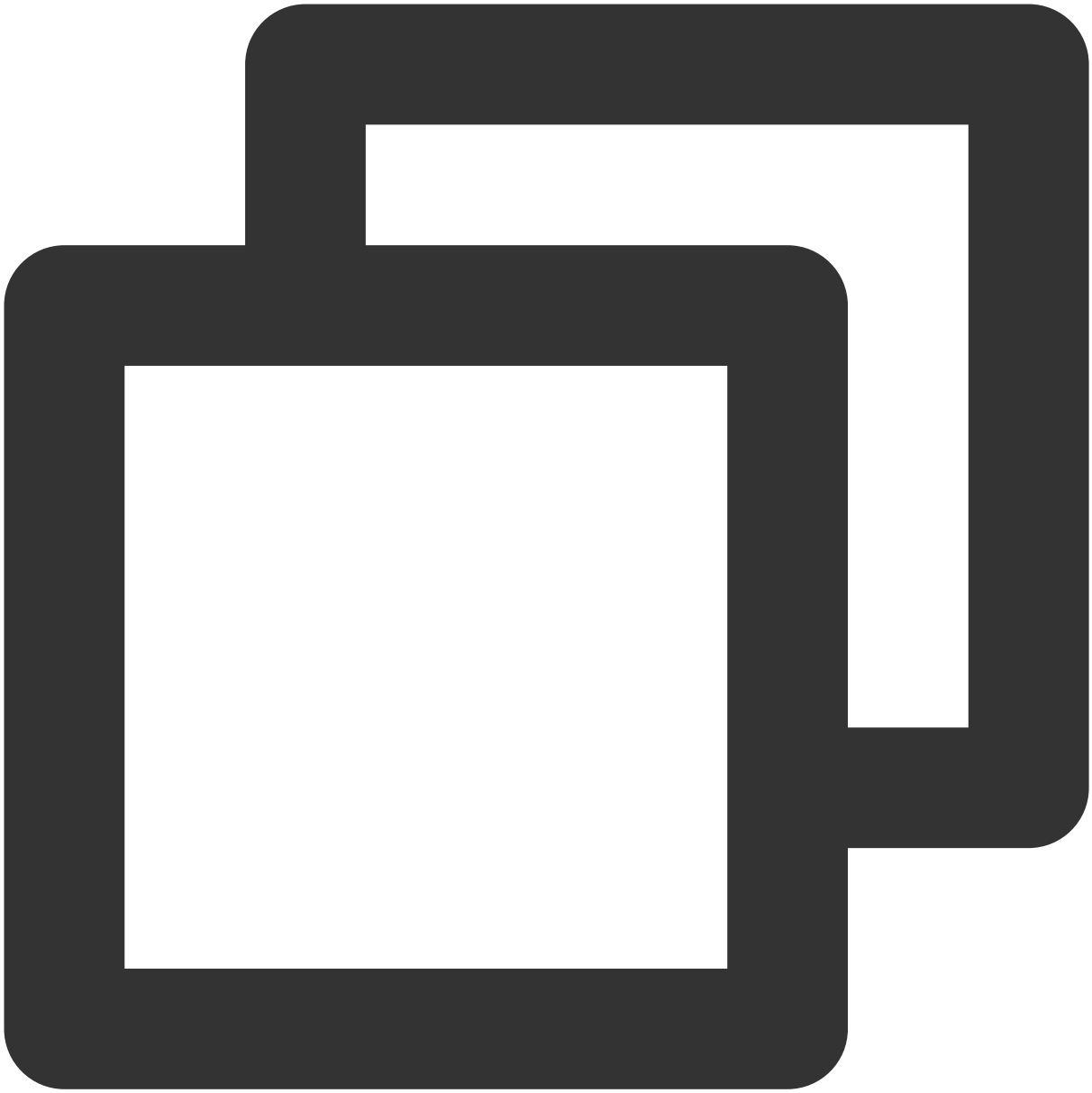
Prometheus has multidimensional analysis capabilities. A data model consists of the following parts:
`Metric Name` + `Labels` + `Timestamp` + `Value/Sample`
Metric Name: monitoring object (for example, `http_request_total` indicates the current total number of HTTP requests received by the system).
Labels: characteristics dimensions of the current sample, which are in K/V structure. Through such dimensions, Prometheus can filter, aggregate, and perform other operations on the sample data.
Timestamp: a timestamp accurate down to the millisecond
Value: a float64 value, which indicates the current sample value.
`Metric Name/Labels` can contain only ASCII characters, digits, underscores, and colons and must comply with the regular expression [a-zA-Z_:][a-zA-Z0-9_:]*.
For more information on a data model, please see DATA MODEL.
For the best practice of metric and label naming, please see METRIC AND LABEL NAMING.

Metric Tracking Method

Prometheus provides four metric types for different monitoring scenarios: `Counter` , `Gauge` , `Histogram` , and `Summary` , as described below. For more information, please see METRIC TYPES.
The Prometheus community provides SDKs for multiple programing languages, all of which are basically similar in usage but differ mostly in syntax. This document uses Go as an example to describe how to report custom monitoring metrics.

Counter

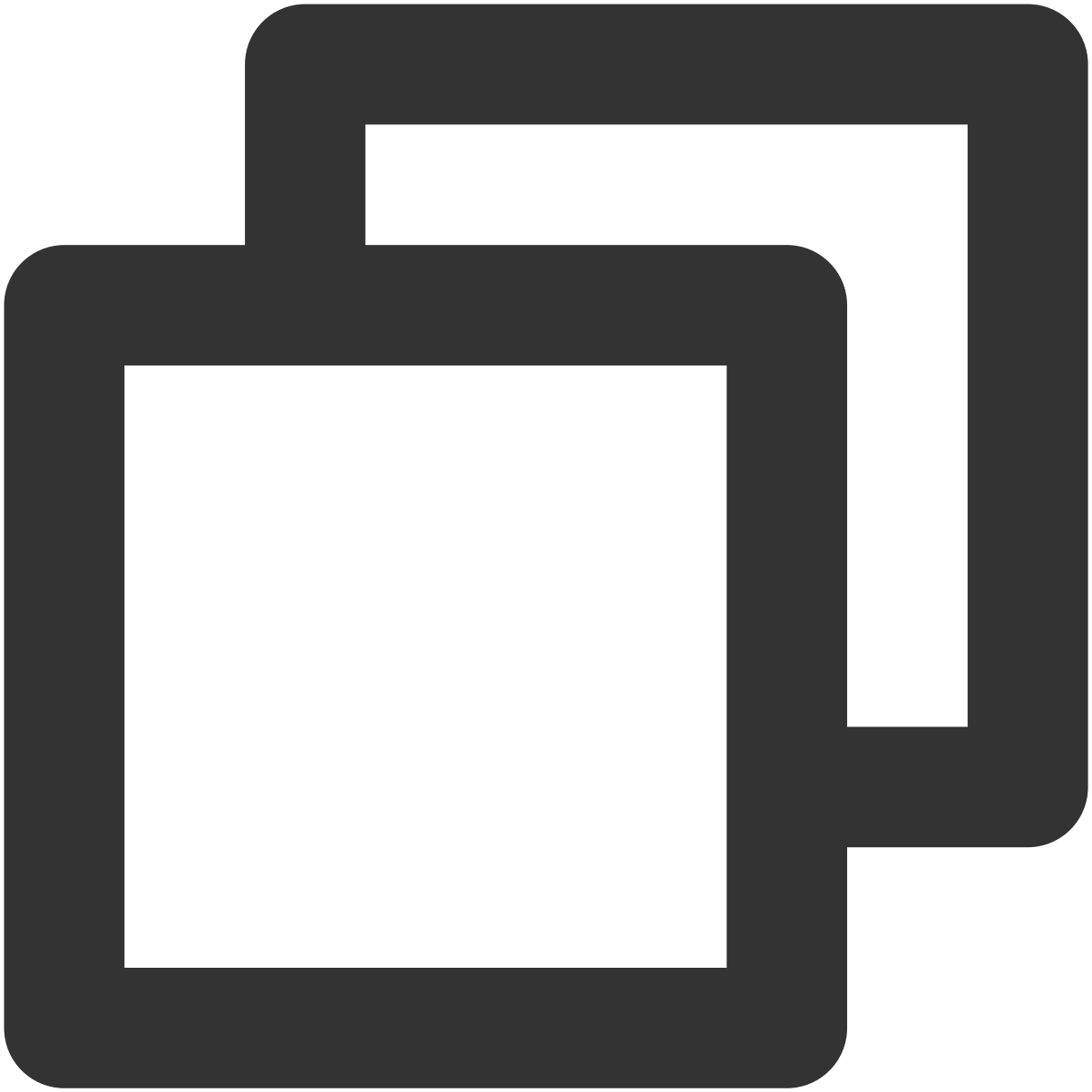
A metric in Counter type increases monotonically and will be reset after service restart. You can use counters to monitor the numbers of requests, exceptions, user logins, orders, etc.
You can use a counter to monitor the number of orders as follows:



```
package order

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

// Define the counter object to be monitored
var (
    opsProcessed = promauto.NewCounterVec(prometheus.CounterOpts{
        Name: "order_service_processed_orders_total",
```

```
        Help: "The total number of processed orders",
    }, []string{"status"}) // Processing status
)

// Process the order
func makeOrder() {
    opsProcessed.WithLabelValues("success").Inc() // Success
    // opsProcessed.WithLabelValues("fail").Inc() // Failure

    // Order placement business logic
}
```

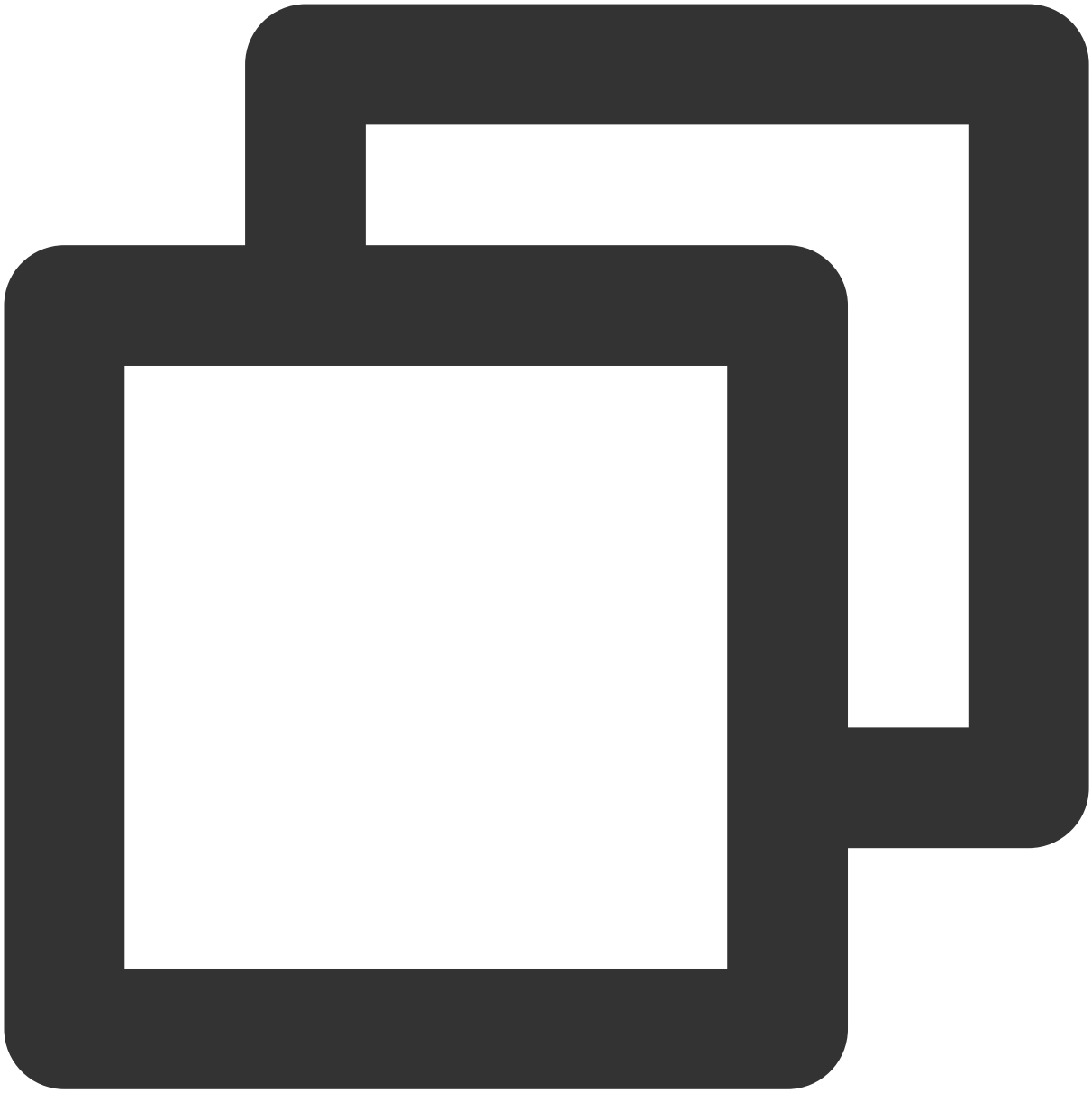
For example, you can use the `rate()` function to get the order increase rate:



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

Gauge

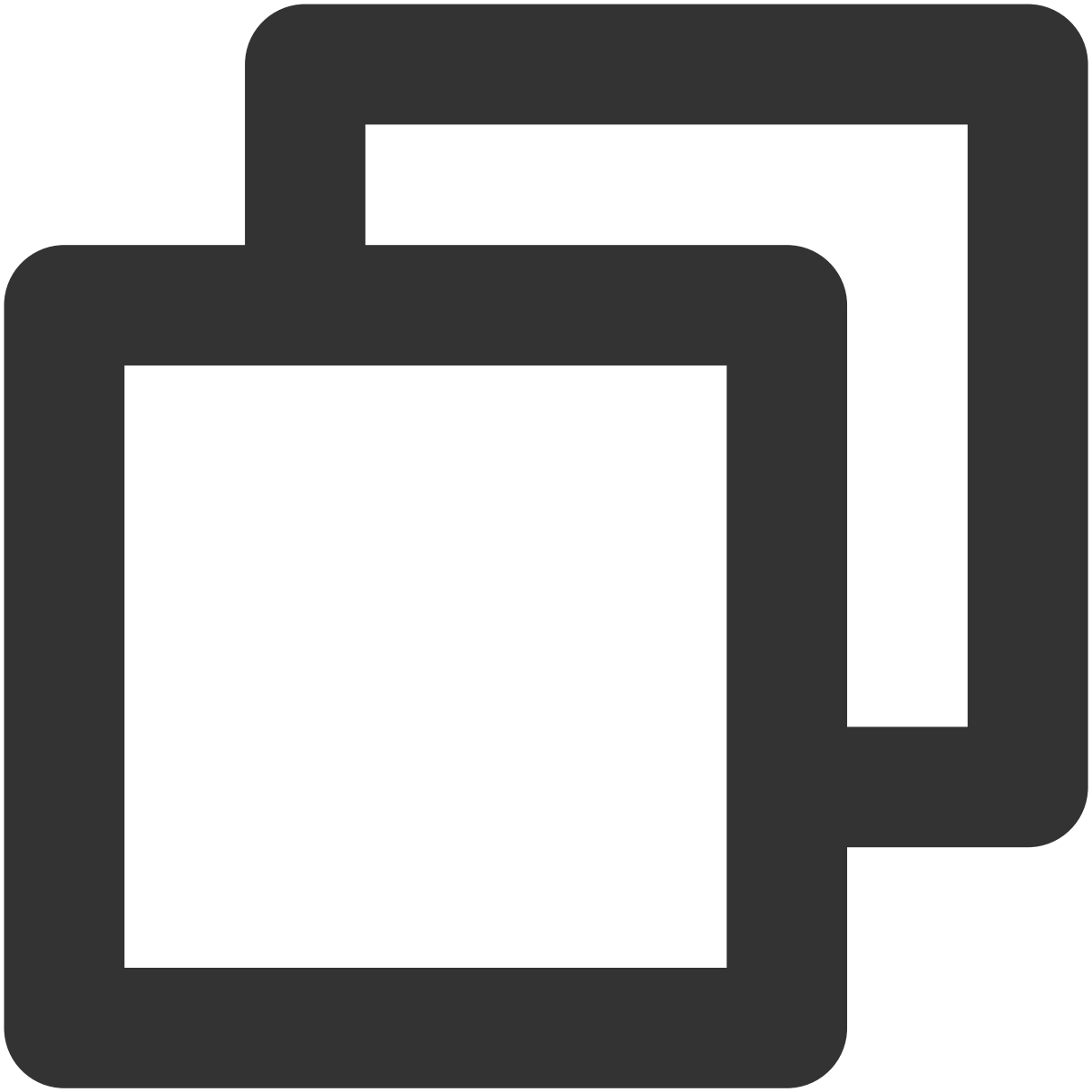
A gauge is a current value, which can be increased or reduced during metric timestamping. You can use gauges to monitor the current memory utilization, CPU utilization, current number of threads, queue size, etc.
You can use a gauge to monitor the size of an order queue as follows:



```
package order

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

// Define the gauge object to be monitored
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),
    }
}

// Produce an order message
func (q *OrderQueue)produceOrder() {
    // Produce an order message

    // Increase the queue size by 1
    queueSize.WithLabelValues("make_order").Inc() // Order placement queue
    // queueSize.WithLabelValues("cancel_order").Inc() // Order cancellation queue
}

// Consume an order message
func (q *OrderQueue)consumeOrder() {
    // Consume an order message

    // Reduce the queue size by 1
    queueSize.WithLabelValues("make_order").Dec()
}
```

You can use the gauge metric to directly view the current size of each type of queue of an order:



```
order_service_order_queue_size
```

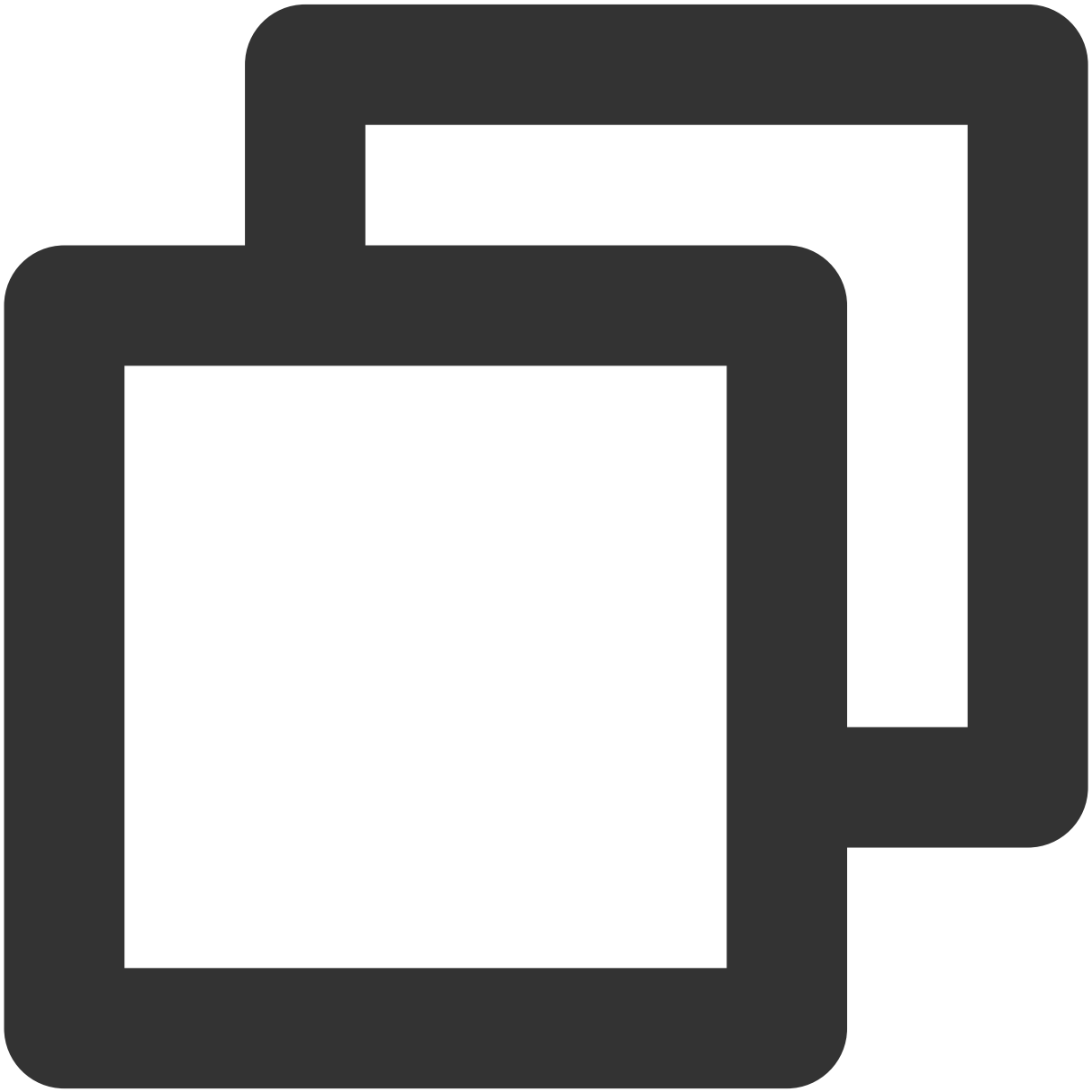
Histogram

Prometheus calculates the sample distribution based on the configured `Bucket` to generate a histogram, which can be processed subsequently and is generally used for duration monitoring. For example, you can use a histogram to calculate the latencies of P99, P95, and P50 and monitor the numbers of processed items. With histograms, you don't need to use counters to count items. In addition, you can use histograms to monitor metrics such as API response time and database access time.

A histogram can be used in a similar way to a summary, so you can directly refer to the summary usage.

Summary

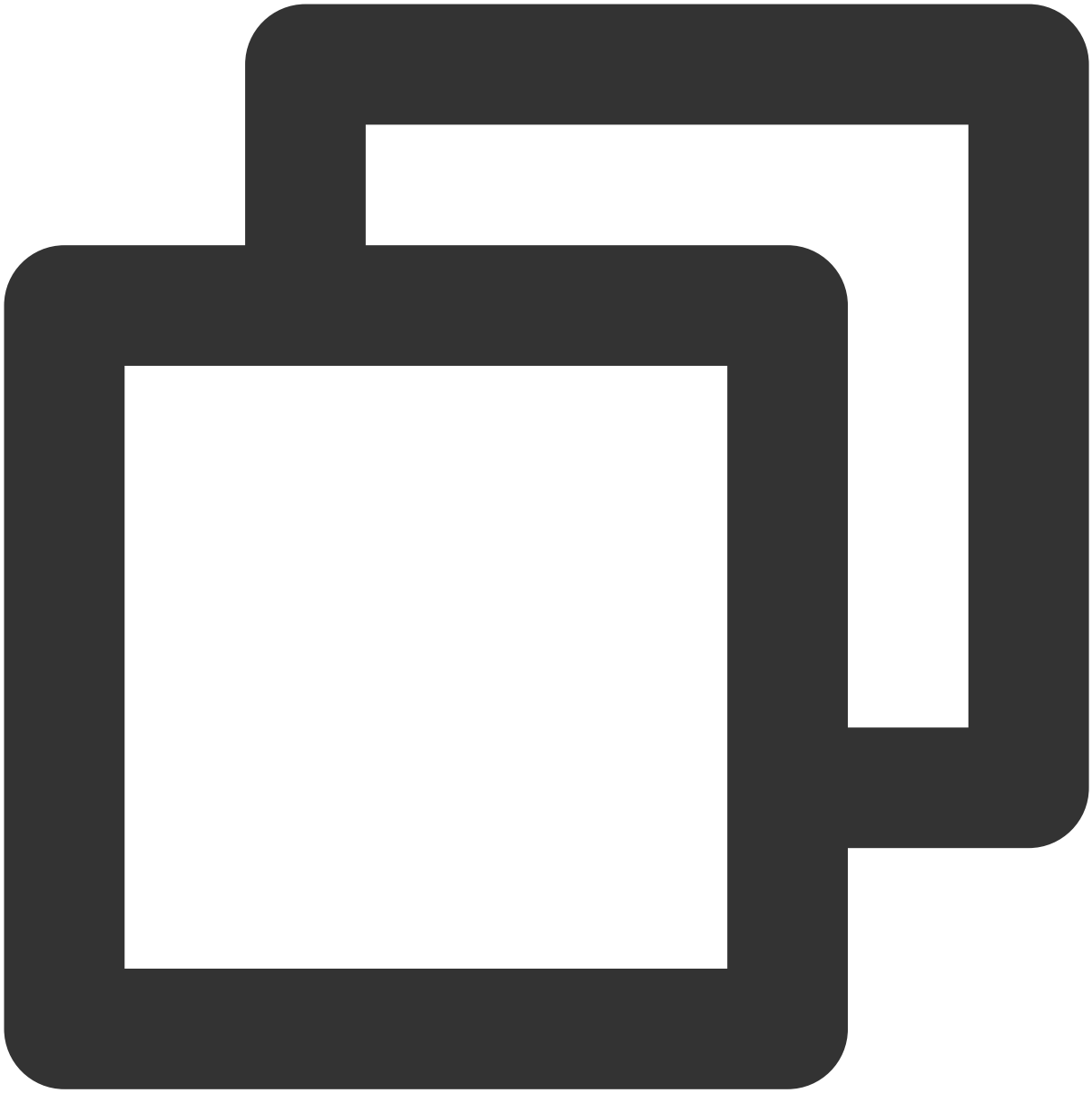
A summary is similar to a histogram, as it also calculates the sample distribution, but their differences lie in that a summary calculates the distribution (P99/P95/Sum/Count) on the client and therefore uses more client resources, and the data cannot be calculated and processed in an aggregated manner subsequently. You can use summaries to monitor metrics such as API response time and database access duration.
You can use a summary to monitor the order processing duration as follows:



```
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"
)

// Define the summary object to be monitored
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()
    // The order placement logic processing is completed, and the processing durati
    defer opsProcessCost.WithLabelValues("success").Observe((float64)(time.Now().Un

    // Order placement business logic
    time.Sleep(time.Second) // Simulate the processing duration
}
```

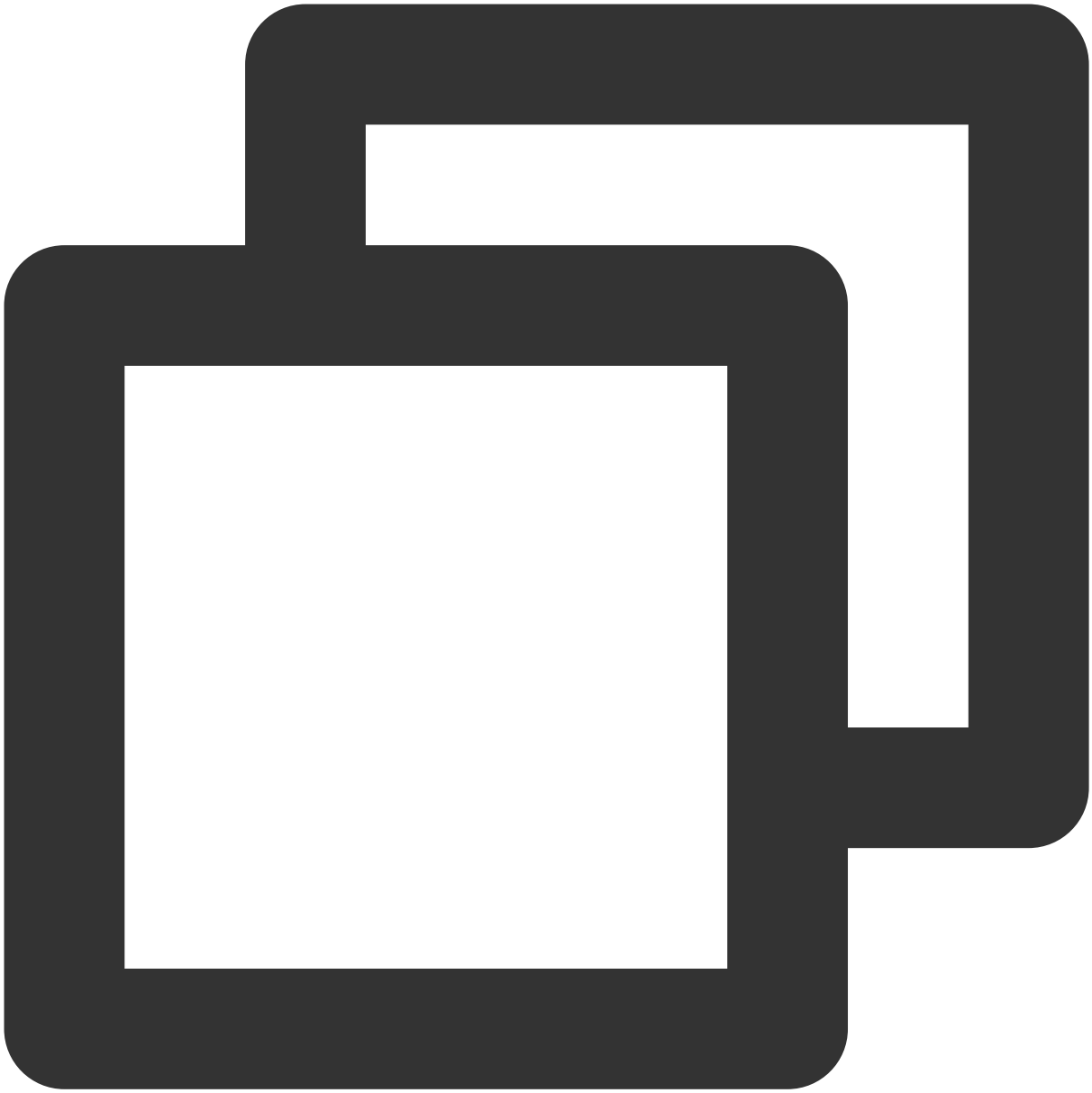
You can use a summary metric to directly view the average order placement processing duration:



```
order_service_processed_order_duration_sum / order_service_processed_order_duration
```

Exposing Prometheus metrics

Use `promhttp.Handler()` to expose the metric tracking data to the HTTP service.



```
package main

import (
    "net/http"

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

func main() {
        // Business code
```

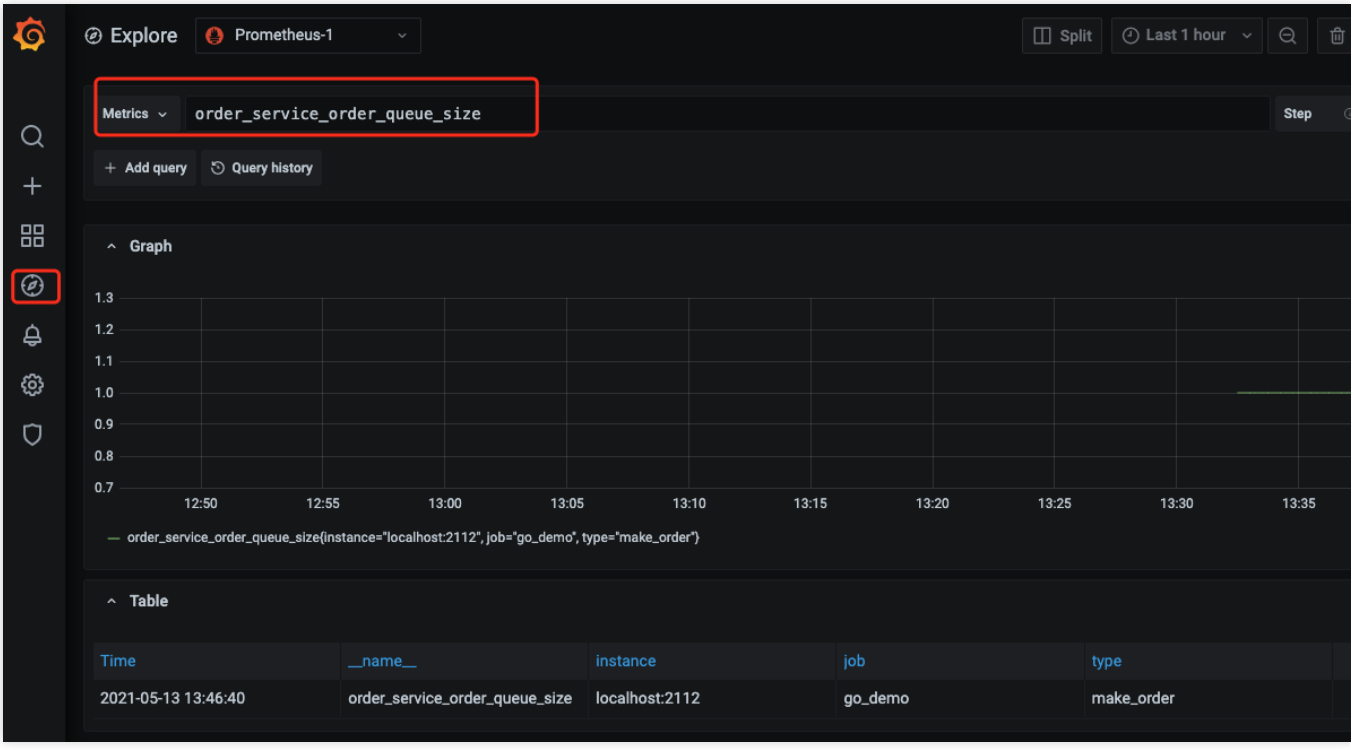
```
        // Expose Prometheus metrics in the HTTP service
        http.Handle("/metrics", promhttp.Handler())

        // Business code
}
```

Collecting Data

After the tracking of custom metrics for your business is completed and the application is released, you can use Prometheus to collect the monitoring metric data. For more information, please see Go Integration.

Viewing Monitoring Data and Alerts

Open the Grafana service that comes with TMP and use `Explore` to view the monitoring metric data as shown below. You can also customize Grafana monitoring dashboards.



You can use Prometheus together with the alarming capabilities of Cloud Monitor to trigger alerts for custom monitoring metrics in real time. For more information, please see Alert Overview and Usage.

EMR Integration
Flink Integration

Last updated：2024-08-07 21:59:47

Overview

When using Flink, you need to monitor its task running status to know whether the tasks run normally and troubleshoot faults. TMP integrates Pushgateway to allow Flink to write metrics and provides an out-of-the-box Grafana monitoring dashboard for it.

Prerequisites

1. The EMR product you purchased includes the Flink component, and a Flink task is running in your instance.
2. You have created a TKE cluster in the region and VPC of your TMP instance.

Directions

Integrating product

Getting Pushgateway access configuration

1. Log in to the **EMR console**, select the corresponding **instance**, and select **Basic Info** > **Instance Info** to get the Pushgateway address and token.

2. Get the `APPID` on the Account Info page.

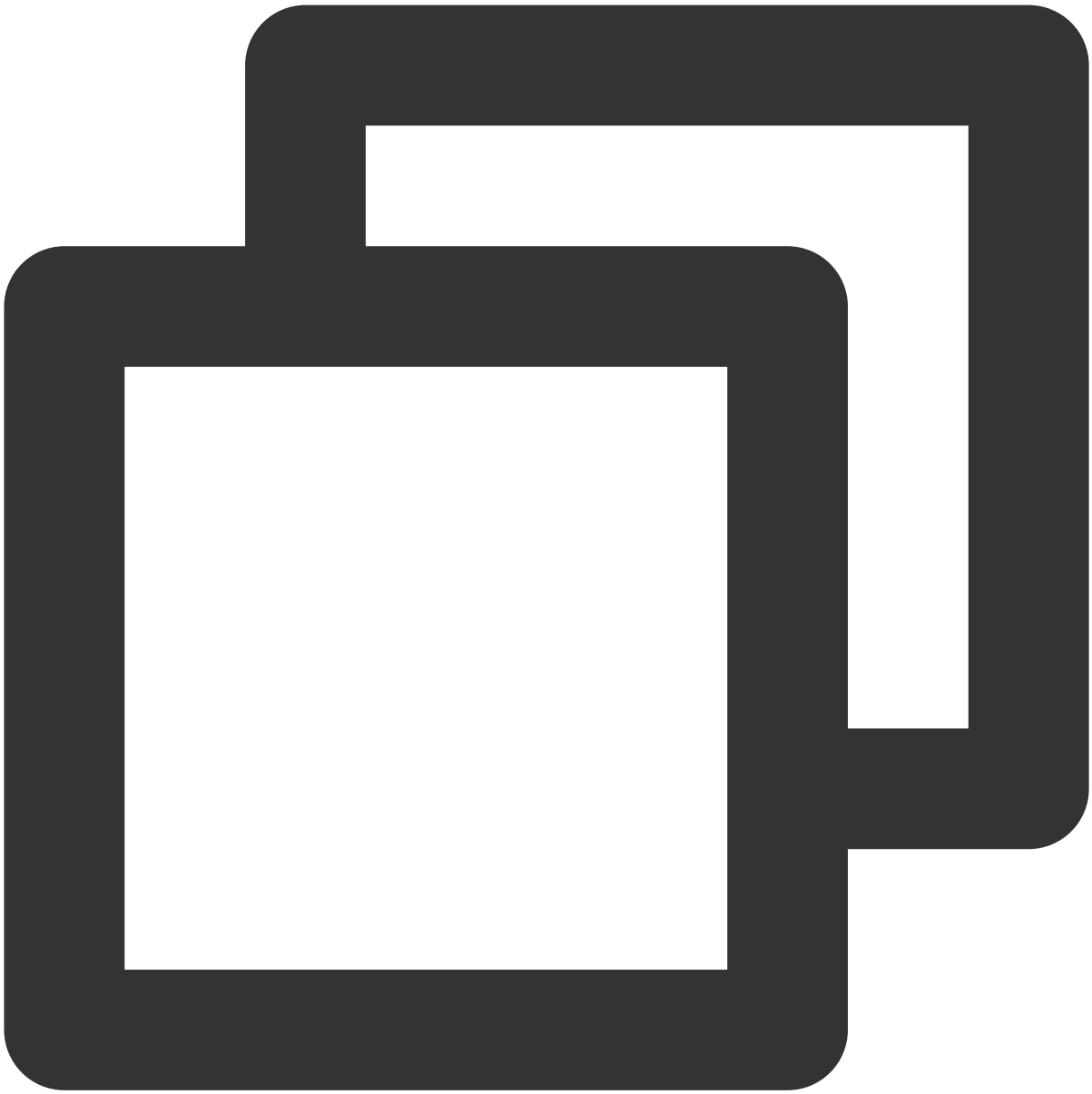
Modifying Flink configuration

1. Log in to the **EMR console**, select the corresponding **instance**, and select **Cluster Service**.
2. Find the **Flink** configuration item and select **Configuration Management** in the **Operation** column on the right to enter the configuration management page.
3. On the right of the page, click **Add Configuration Item** and add the following configuration items one by one:

Configuration Item	Default Value	Data Type	Description	Suggestion
metrics.reporter.promgateway.class	None	String	Name of the Java class for exporting metrics to Pushgateway	-
metrics.reporter.promgateway.jobName	None	String	Push task name	Specify an eas understandab string
metrics.reporter.promgateway.randomJobNameSuffix	true	Boolean	Whether to add a random string after the task name	Set it to `true`. no random stri is added, metr of different Flir tasks will overwrite each other
metrics.reporter.promgateway.groupingKey	None	String	Global label	Add the EMR

			added to each metric in the format of `k1=v1;k2=v2`	instance ID to distinguish between the d of different instances, suc as `instance_id=e xxx`
metrics.reporter.promgateway.interval	None	Time	Time interval for pushing metrics, such as 30s	We recommer you set the val to about 1 min
metrics.reporter.promgateway.host	None	String	Pushgateway service address	It is the servic address of the TMP instance the console
metrics.reporter.promgateway.port	-1	Integer	Pushgateway service port	It is the port of TMP instance the console
metrics.reporter.promgateway.needBasicAuth	false	Boolean	Whether the Pushgateway service requires authentication	Set it to `true`, the Pushgatev of TMP requir authentication
metrics.reporter.promgateway.user	None	String	Username for authentication	It is your `APP
metrics.reporter.promgateway.password	None	String	Password for authentication	It is the acces token of the TI instance in the console
metrics.reporter.promgateway.deleteOnShutdown	true	Boolean	Whether to delete the corresponding metrics on the Pushgateway after the Flink task is completed	Set it to `true`

Below is a sample configuration:



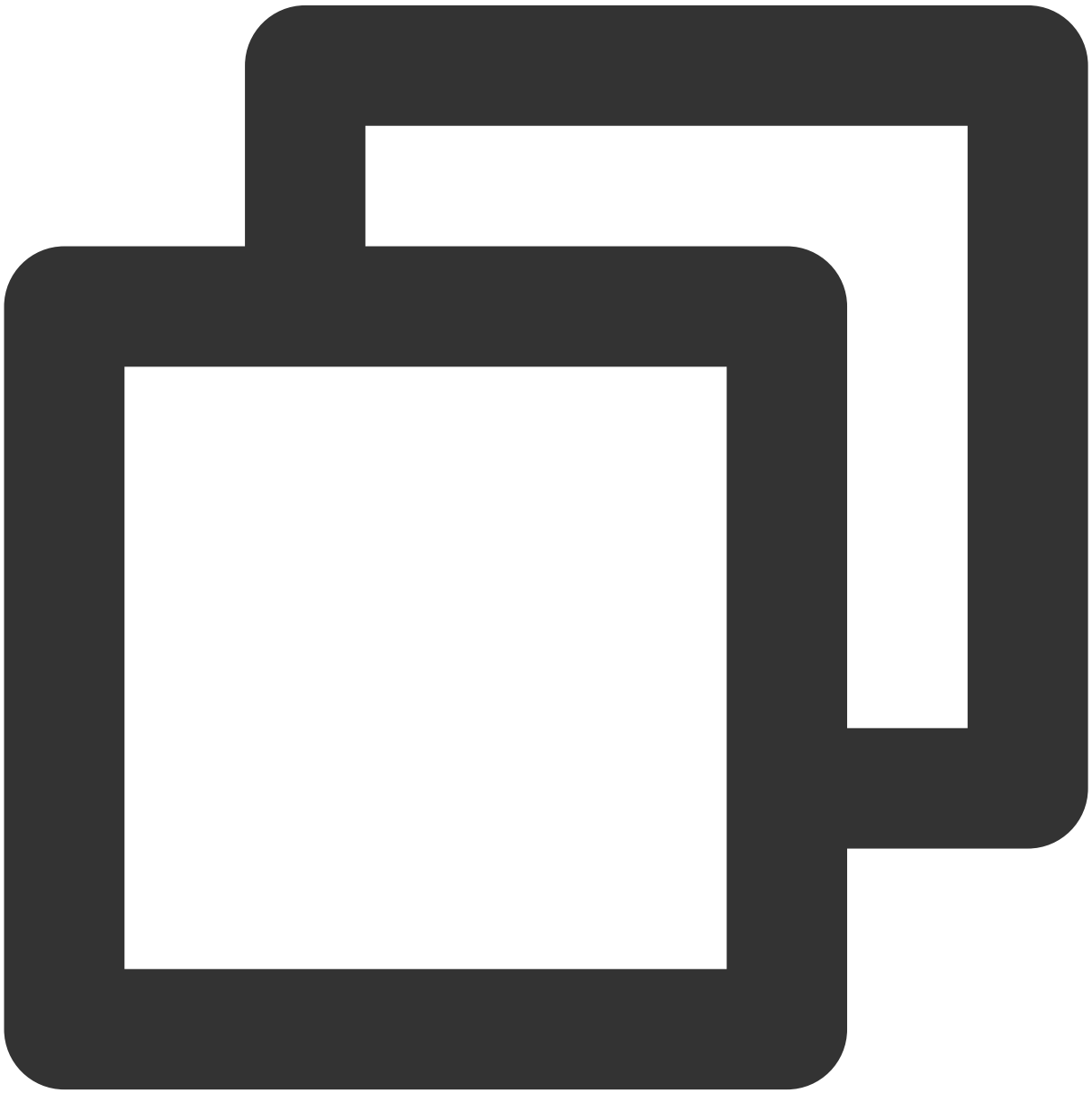
```
metrics.reporter.promgateway.class: org.apache.flink.metrics.prometheus.PrometheusP
metrics.reporter.promgateway.jobName: climatePredict
metrics.reporter.promgateway.randomJobNameSuffix:true
metrics.reporter.promgateway.interval: 60 SECONDS
metrics.reporter.promgateway.groupingKey:instance_id=emr-xxxx
metrics.reporter.promgateway.host: 172.xx.xx.xx
metrics.reporter.promgateway.port: 9090
metrics.reporter.promgateway.needBasicAuth: true
metrics.reporter.promgateway.user: appid
```

```
metrics.reporter.promgateway.password: token
```

Installing Flink Pushgateway plugin

The Pushgateway plugin in the official package currently does not support configuring the authentication information, but TMP requires authentication before data can be written. Therefore, we recommend you use the JAR package we provide. We have also submitted a pull request for supporting authentication to the Flink team.

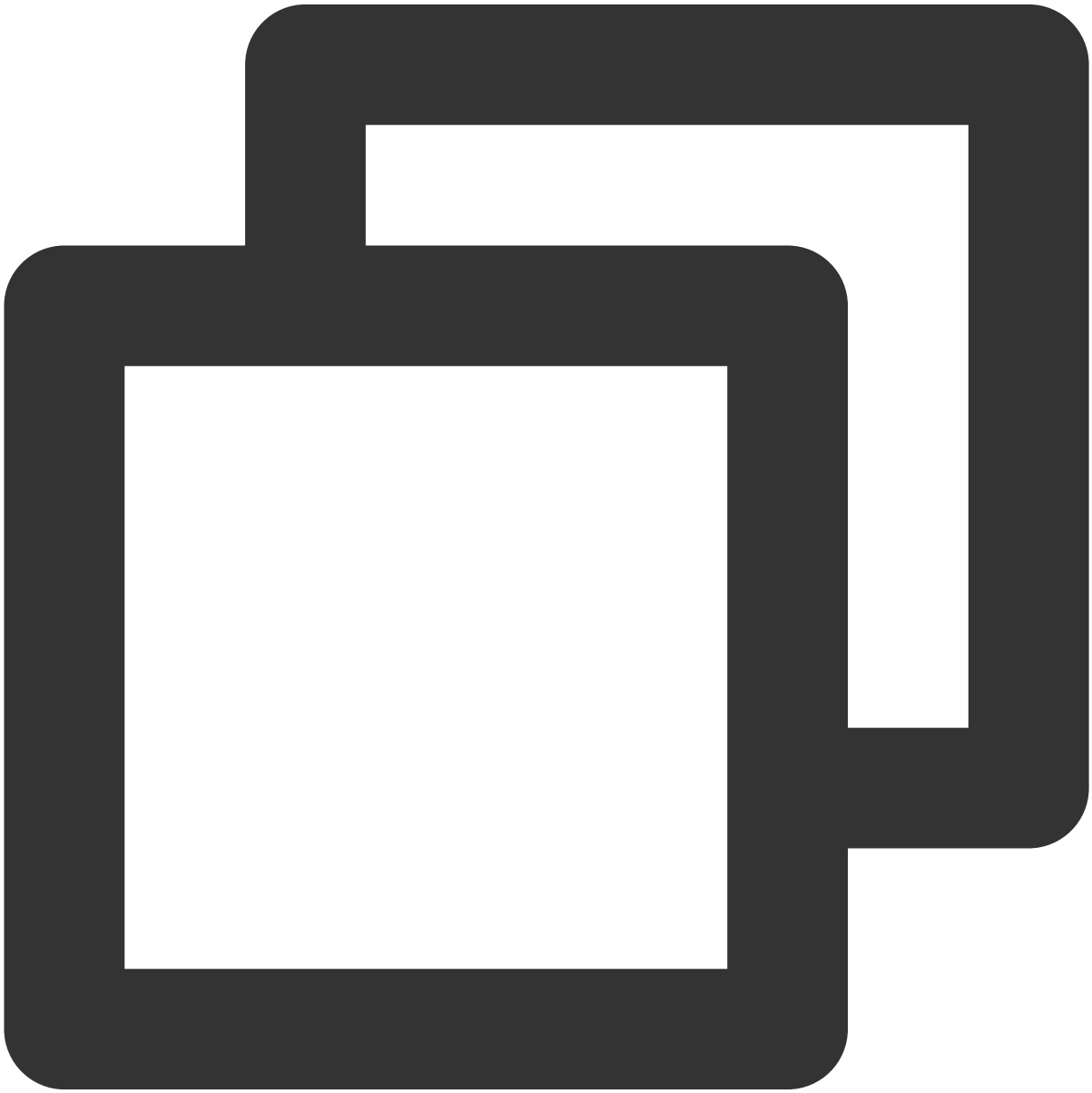
1. To prevent class conflicts, if you have already used the official Flink plugin, run the following command to delete it first:



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

2. In the **EMR console**, select the corresponding **instance**, and select **Cluster Resource** > **Resource Management** > **Master** to view the master node.

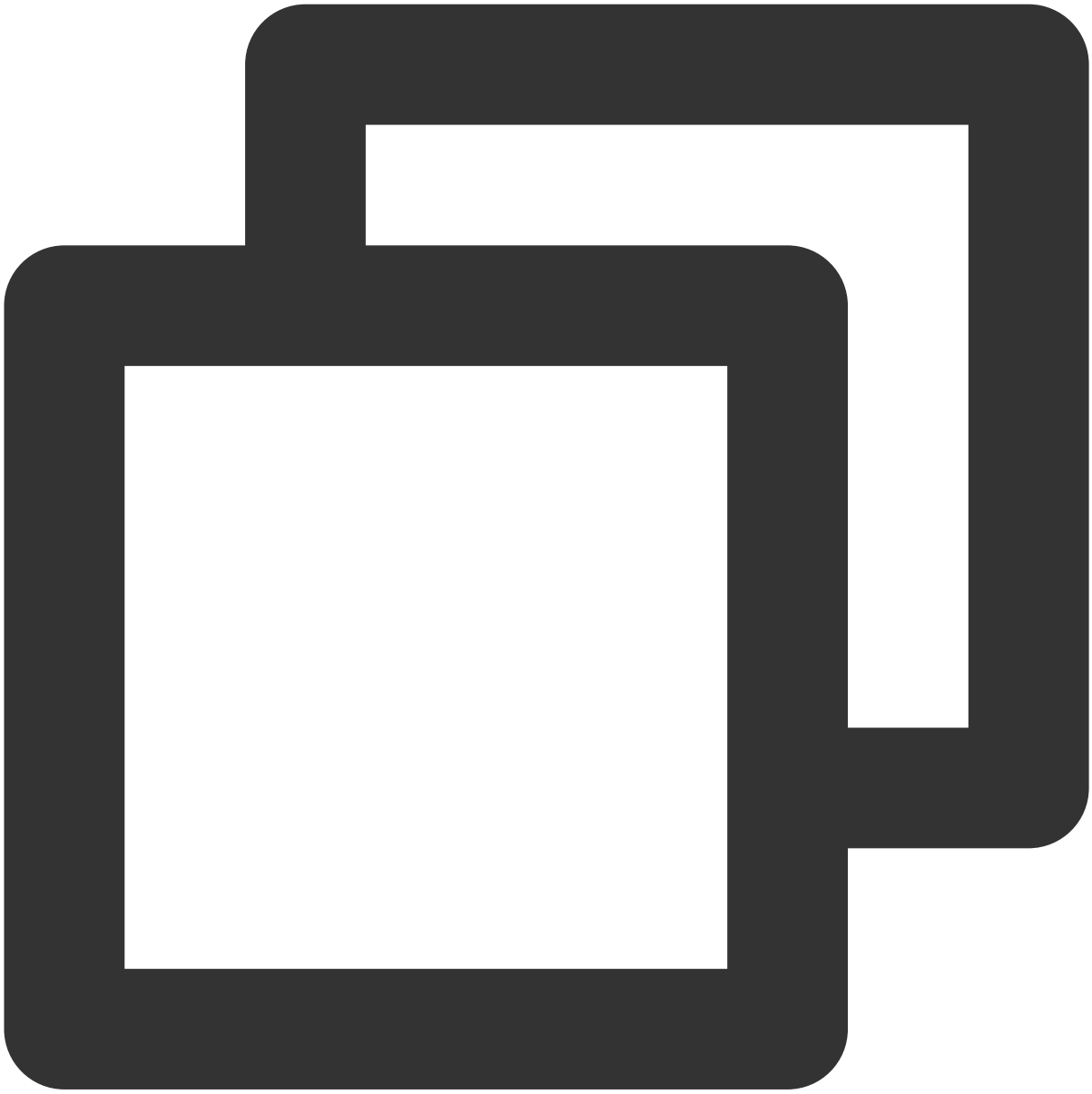
3. Click the instance ID to go to the CVM console, log in to the CVM instance, and run the following command to install the plugin:



```
cd /usr/local/service/flink/lib
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/flink/flink-metrics-prome
```

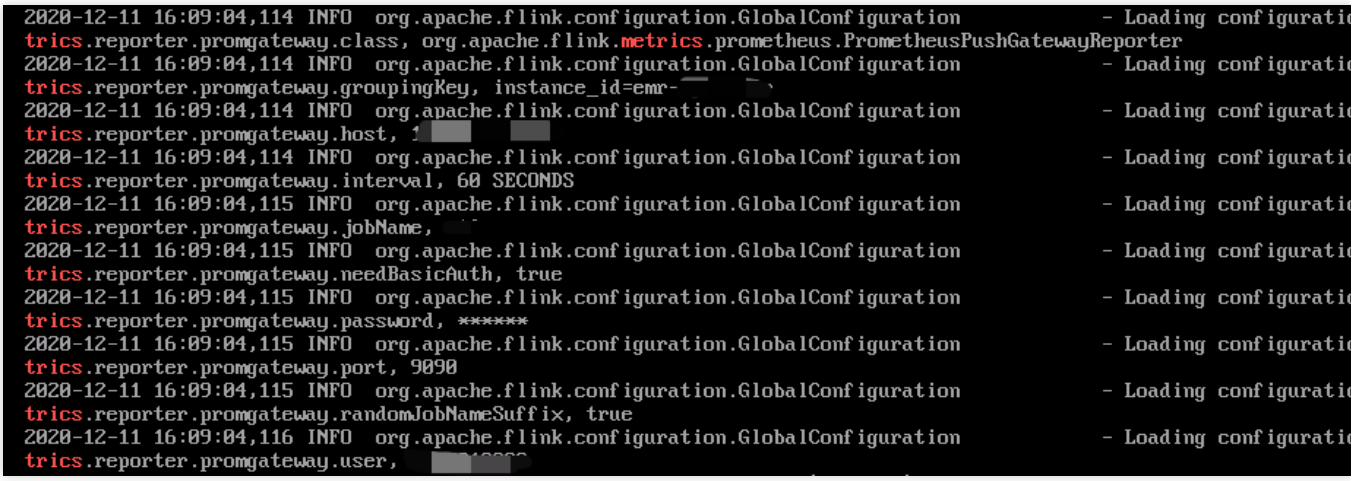
Verifying

1. Run the `flink run` command on the master node to submit a new task and view the task log:



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

2. If the log contains the following content, the configuration is successfully loaded:



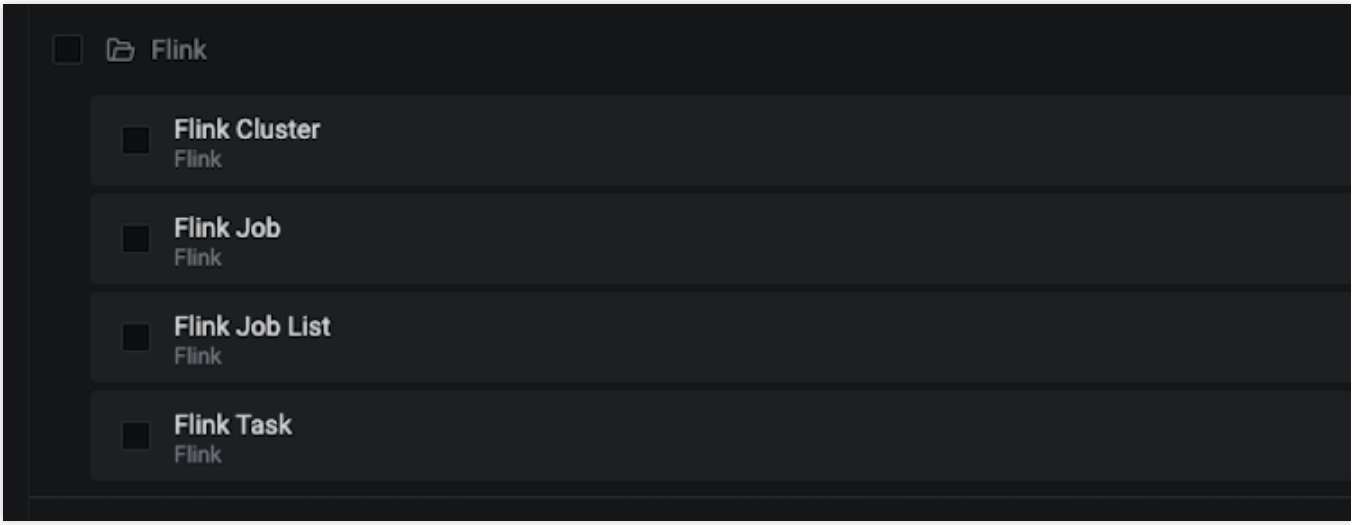
Note:

As tasks previously submitted in the cluster use the old configuration file, their metrics are not reported.

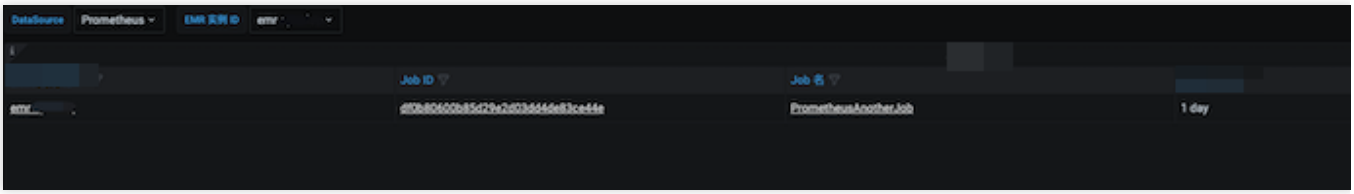
Viewing monitoring information

1. In **Integration Center** in the target TMP instance, find Flink monitoring, install the corresponding Grafana dashboard, and then you can enable the Flink monitoring dashboard.
2. Enter Grafana and click

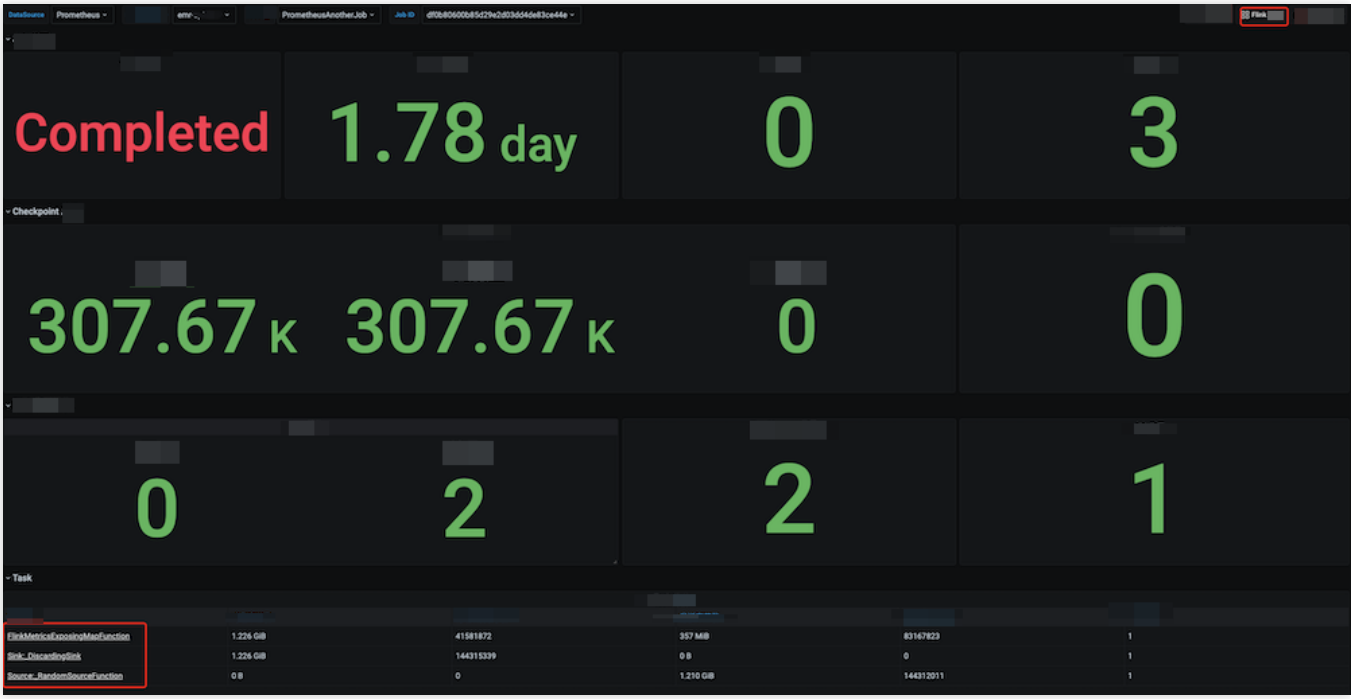
to expand the Flink monitoring panel.



3. Click **Flink Job List** to view the monitoring information.

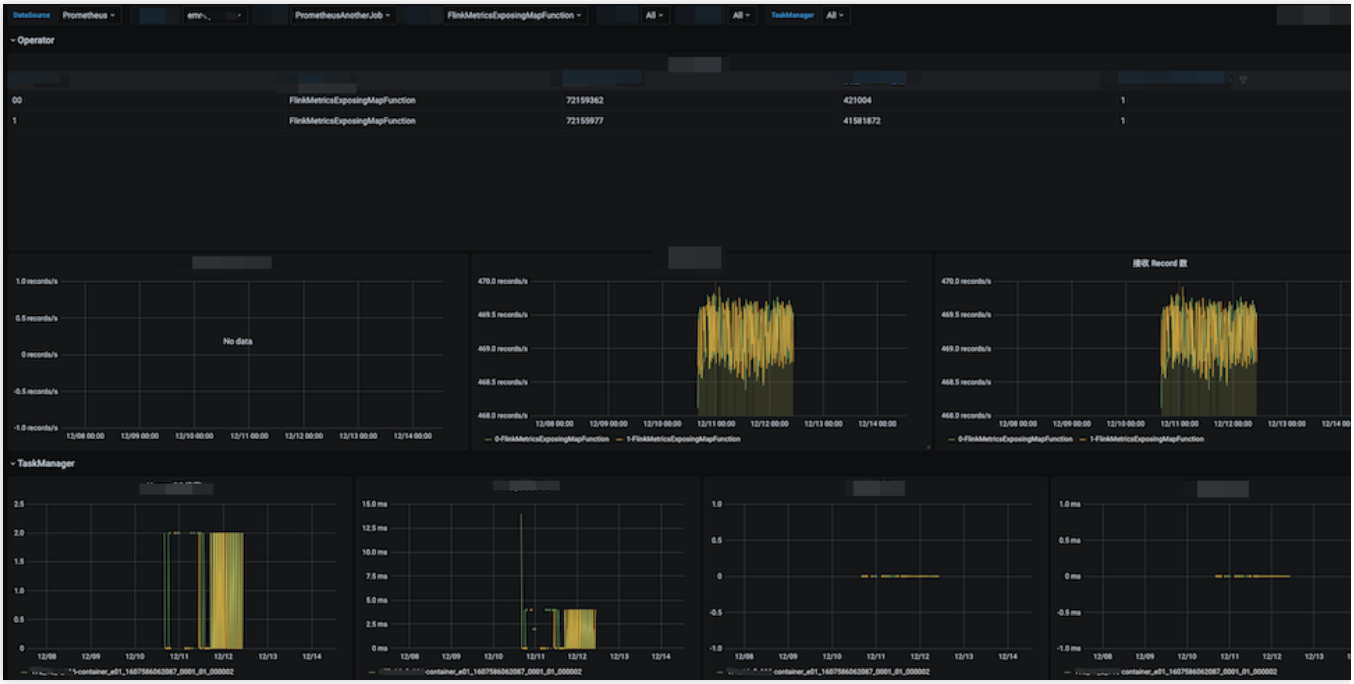


4. Click a **job name** or **job ID** in the table to view the job monitoring details.



5. Click **Flink Cluster** in the top-right corner to view the Flink cluster monitoring information.

6. Click a **task name** in the table to view the task monitoring details.



Integrating with alert feature

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Alerting Rule** and add the corresponding alerting rules. For more information, please see Creating Alerting Rule.

Java Application Integration
Spring Boot Integration

Last updated：2024-08-07 22:00:28

Overview

When using Spring Boot as the development framework, you need to monitor the status of applications such as JVM and Spring MVC. TMP collects data such as JVM data based on the Spring Boot Actuator mechanism. With the Grafana dashboard that comes with TMP, you can conveniently monitor the status of Spring Boot applications.
This document uses deploying a Spring Boot application in TKE as an example to describe how to use TMP to monitor the application status.

Prerequisites

Create a TKE cluster.
Use a private image repository to manage application images.
The image is developed based on the Spring Boot framework.

Directions

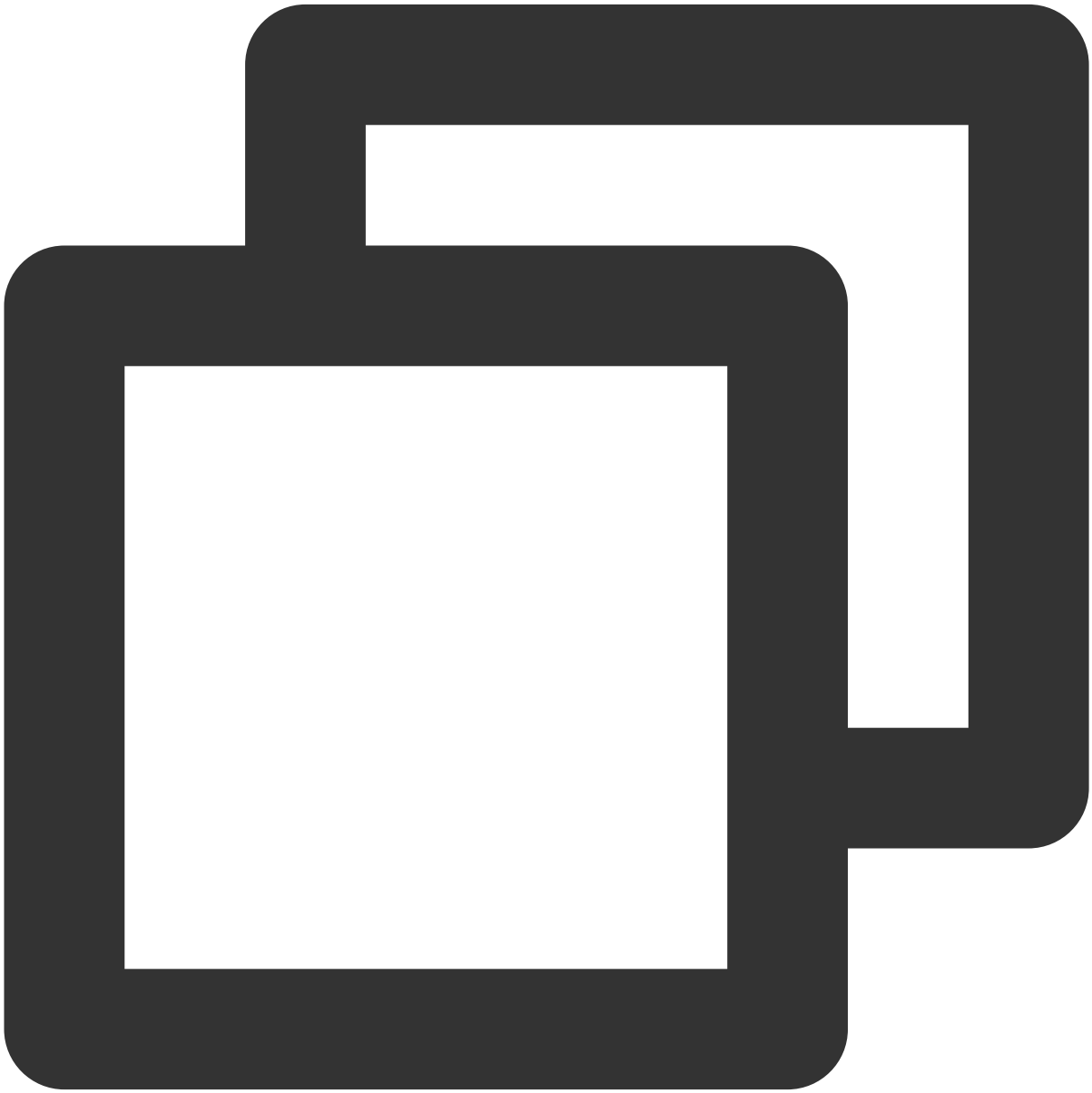
Note:
Spring Boot provides the Actuator component to monitor applications, which reduces the development costs. Therefore, Actuator is directly used in this document to track Spring Boot metrics. You should use Spring Boot v2.0 or above in the following steps, as lower versions may have different configurations.
If you use Spring Boot v1.5 for integration, the integration process will differ from that for v2.0, and you should note the following:
1. The address for accessing `prometheus metrics` is different from that for v2.0. On v1.5, the default address is `/prometheus`, i.e., `http://localhost:8080/prometheus`.
2. If error 401 is reported, it indicates no permissions (Whitelabel Error Page). On v1.5, security control is enabled for the `management` API by default, so you need to set `management.security.enabled=false`.
3. If `bootstrap.yml` is used to configure parameters in the project, modifying `management` in it will not work, which should be modified in `application.yml` due to the Spring Boot start and load sequence.

4. You cannot add `metric common tag` through YML; instead, you can add it only by adding a `bean` to the code.

Modifying application dependencies and configuration

Step 1. Modify POM dependencies

If `spring-boot-starter-web` is already imported in this project, add the `actuator/prometheus` Maven dependency to the `pom.xml` file.

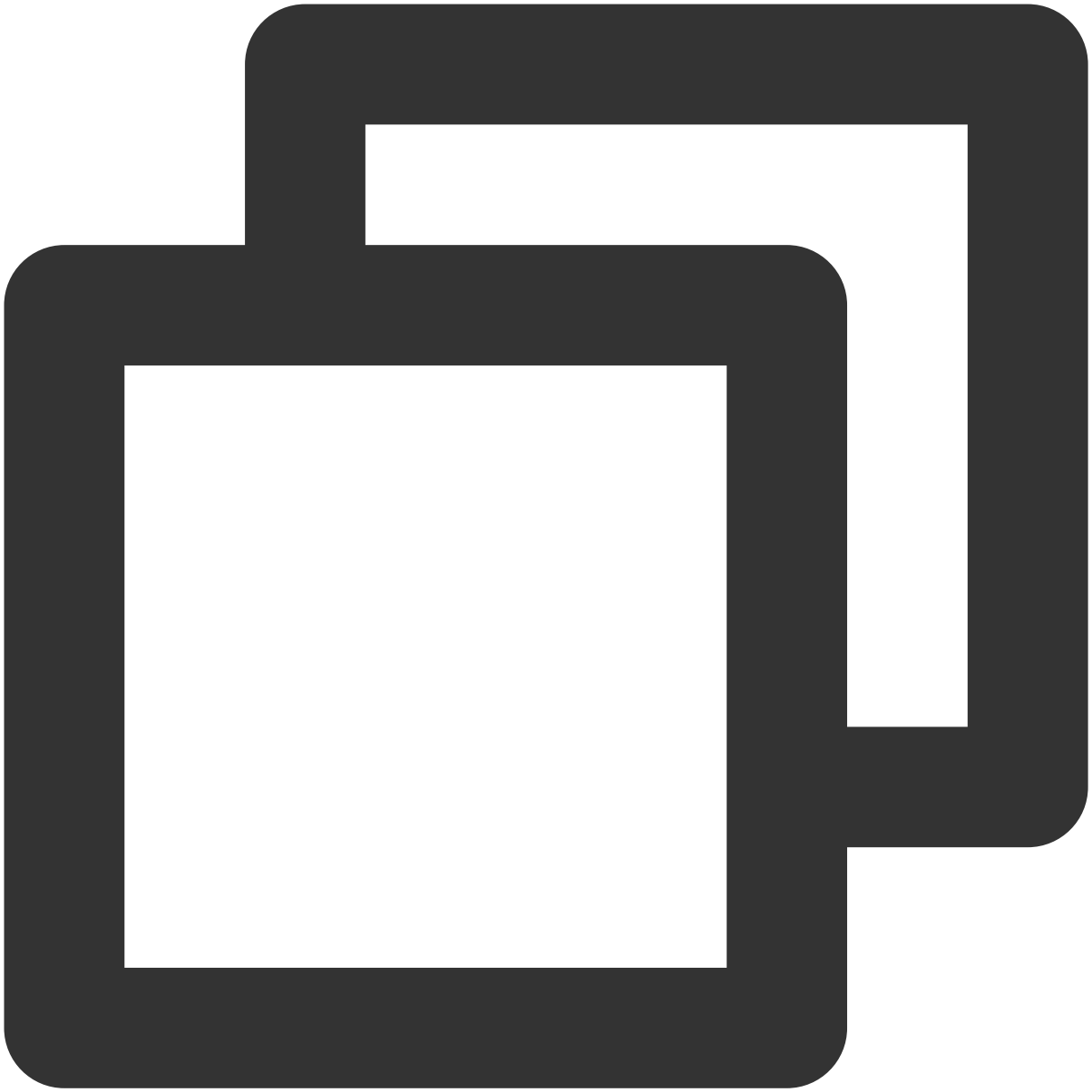


```
<dependency>
```

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

Step 2. Modify the configuration

Edit the `application.yml` file in the `resources` directory and modify the `actuator` configuration to expose the metric data in the Prometheus protocol.



```
management:
  endpoints:
    web:
      exposure:
        include: prometheus  # Web access path for opening Prometheus
  metrics:
    # We recommend you enable the following options to monitor P99 and P95 latencie
    distribution:
      sla:
        http:
          server:
```

```
            requests: 1ms,5ms,10ms,50ms,100ms,200ms,500ms,1s,5s
    # Add special labels to Prometheus
    tags:
      # You must add the corresponding application name, as the corresponding monit
      application: spring-boot-mvc-demo
```

Step 3. Perform local verification

In the current directory of the project, run `mvn spring-boot:run` . If you can access the metric data of the Prometheus protocol through `http://localhost:8080/actuator/prometheus` , the relevant dependency configuration is correct.

Note:

The default configurations of the port and path are used in the same, which should be replaced with those in your actual project.

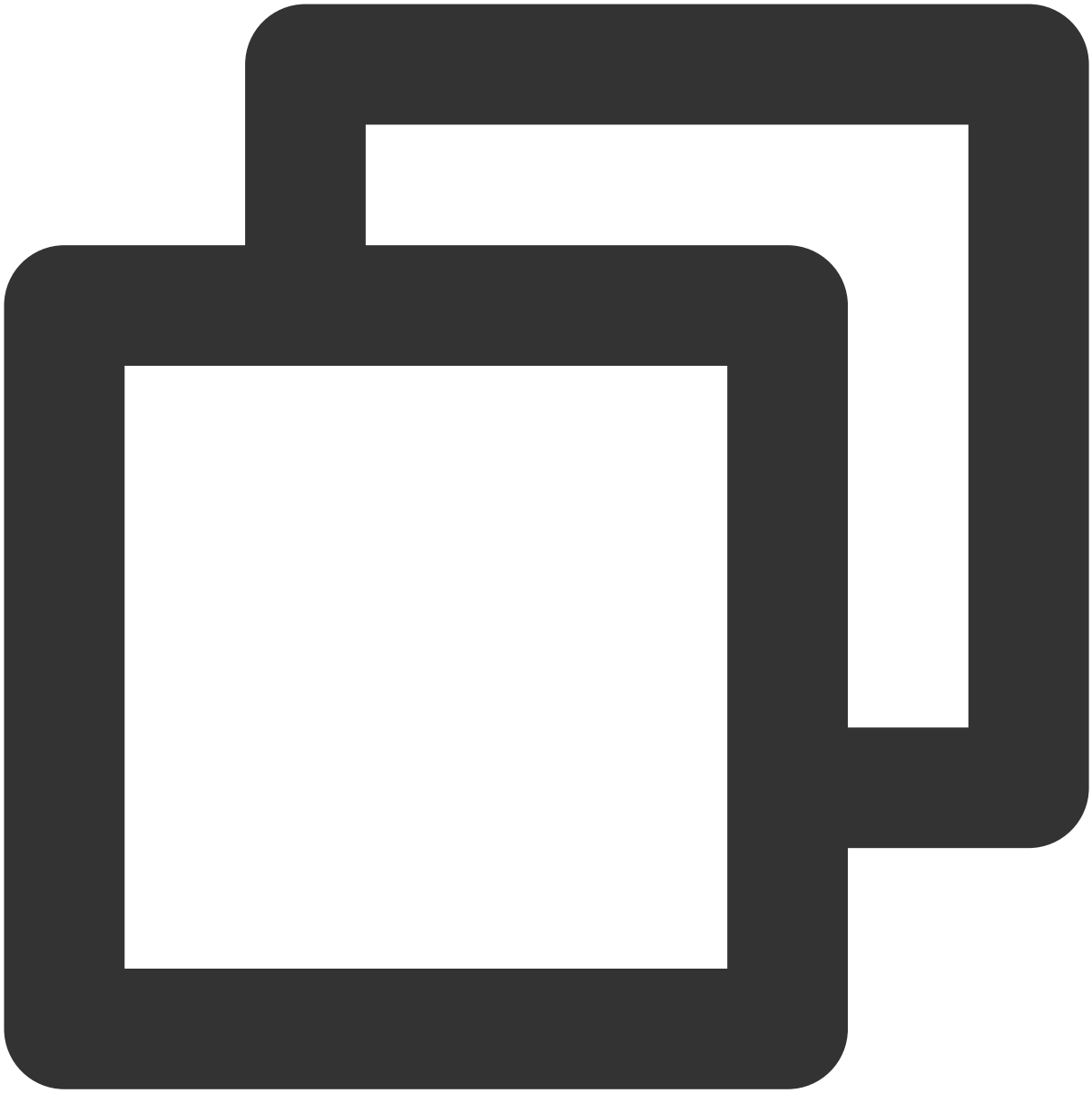
Releasing application to TKE

Step 1. Configure a Docker image environment locally

If you have already configured a Docker image environment locally, proceed to the next step; otherwise, configure one as instructed in Getting Started.

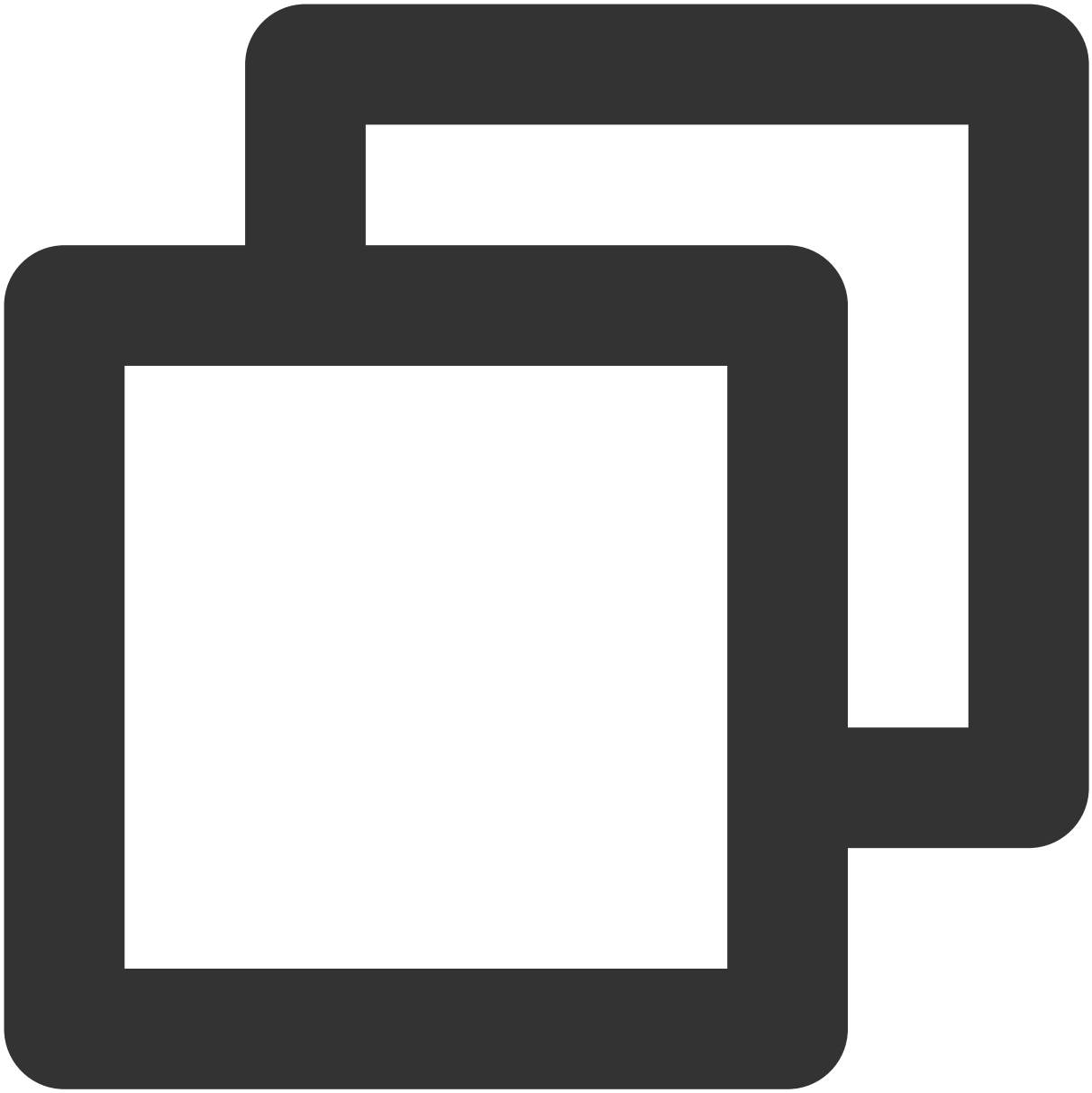
Step 2. Package and upload the image

1. Add `Dockerfile` in the root directory of the project. You can add it by referring to the following sample code and modify `Dockerfile` based on your actual project:



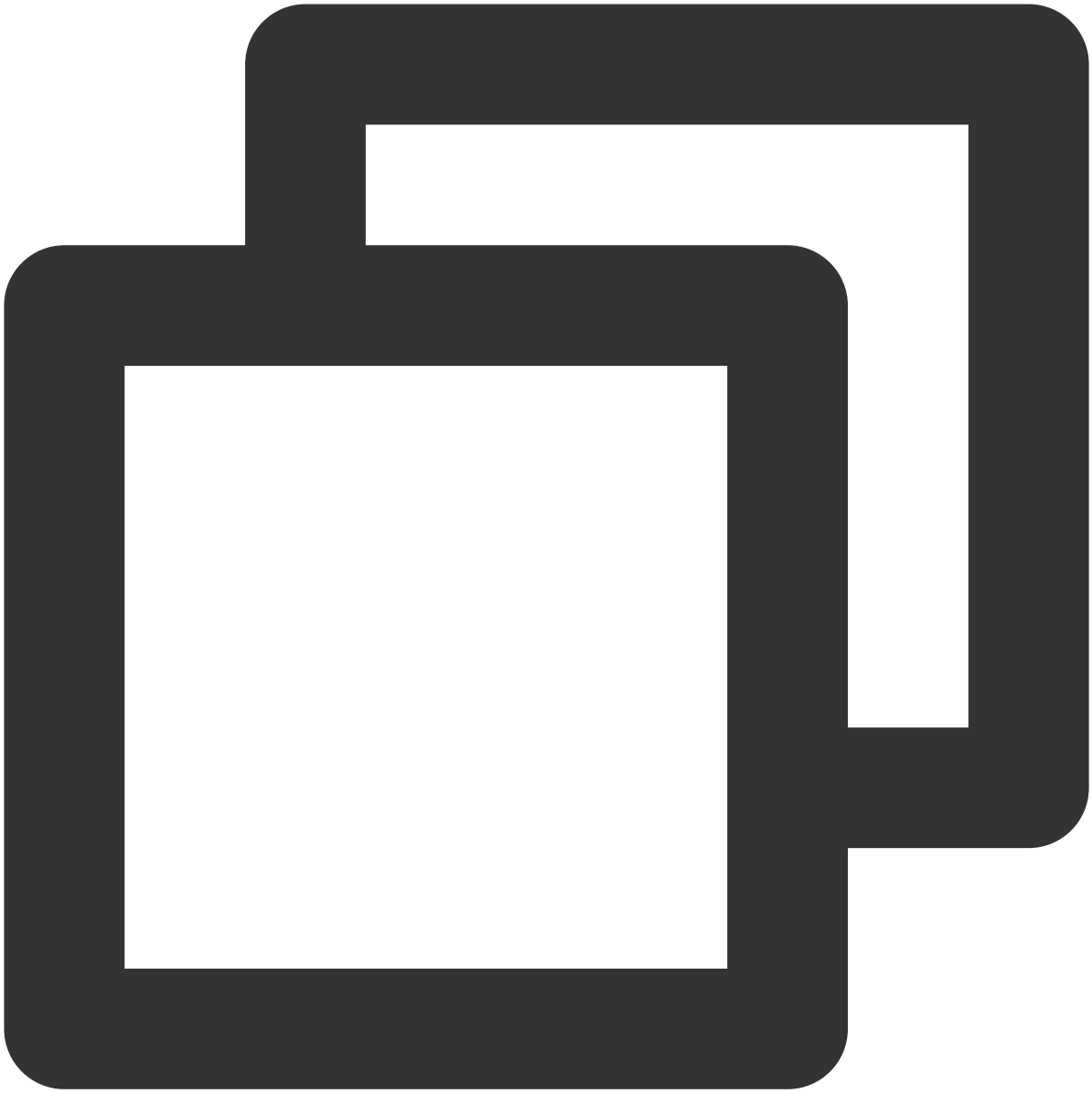
```
FROM openjdk:8-jdk
WORKDIR /spring-boot-demo
ADD target/spring-boot-demo-*.jar /spring-boot-demo/spring-boot-demo.jar
CMD ["java","-jar","spring-boot-demo.jar"]
```

2. Package the image by running the following command in the project root directory. You need to replace `namespace`, `ImageName`, and `image tag` as needed in your actual project.



```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

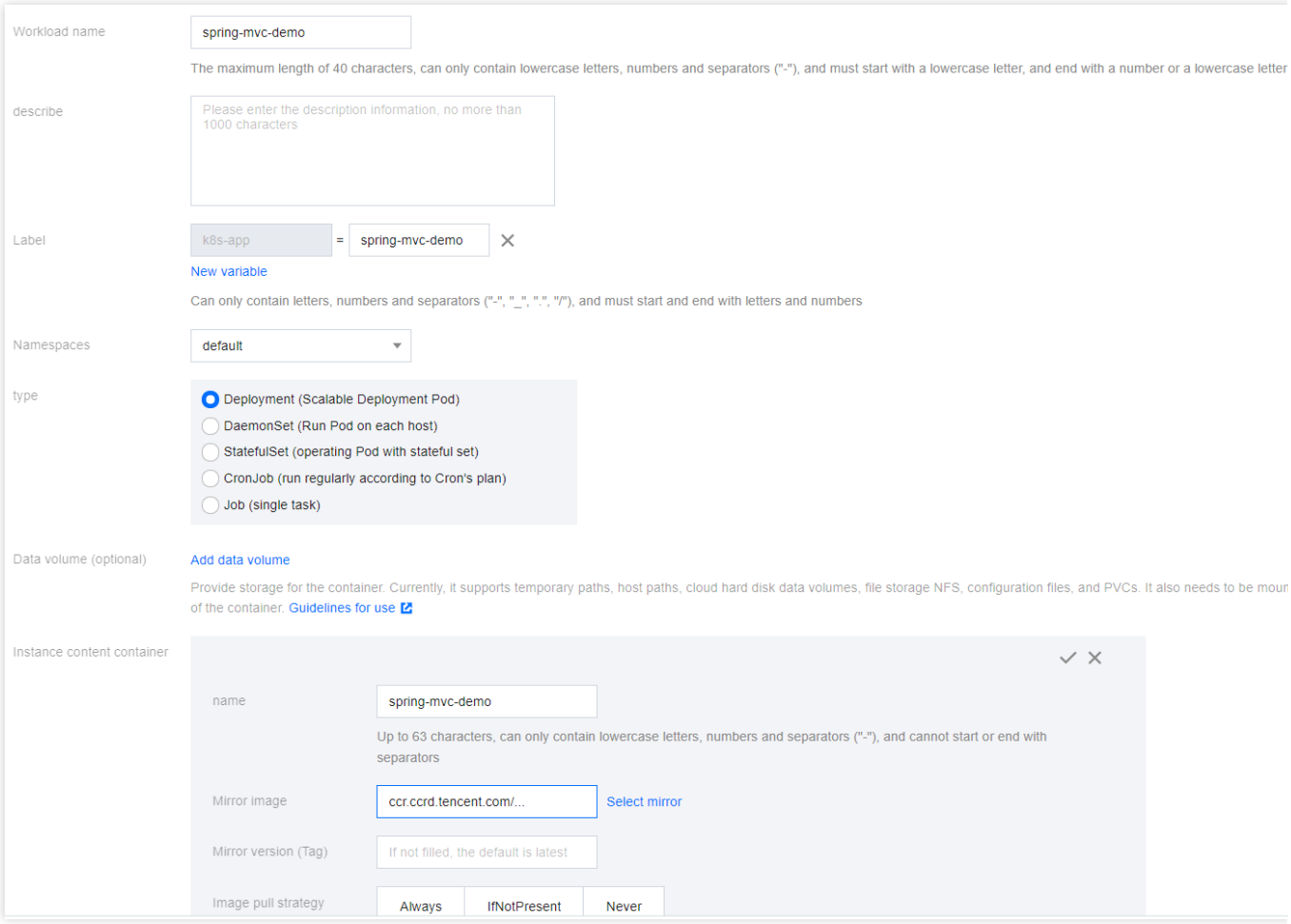
For example:

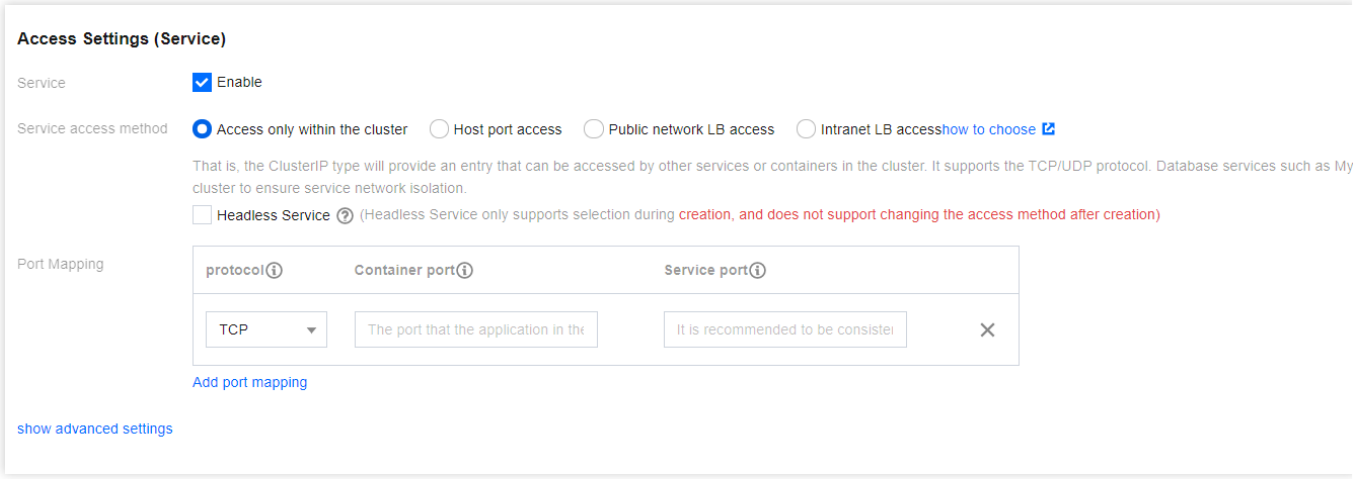


```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/prom_spring_demo/spring-boot-demo:latest
docker push ccr.ccs.tencentyun.com/prom_spring_demo/spring-boot-demo:latest
```

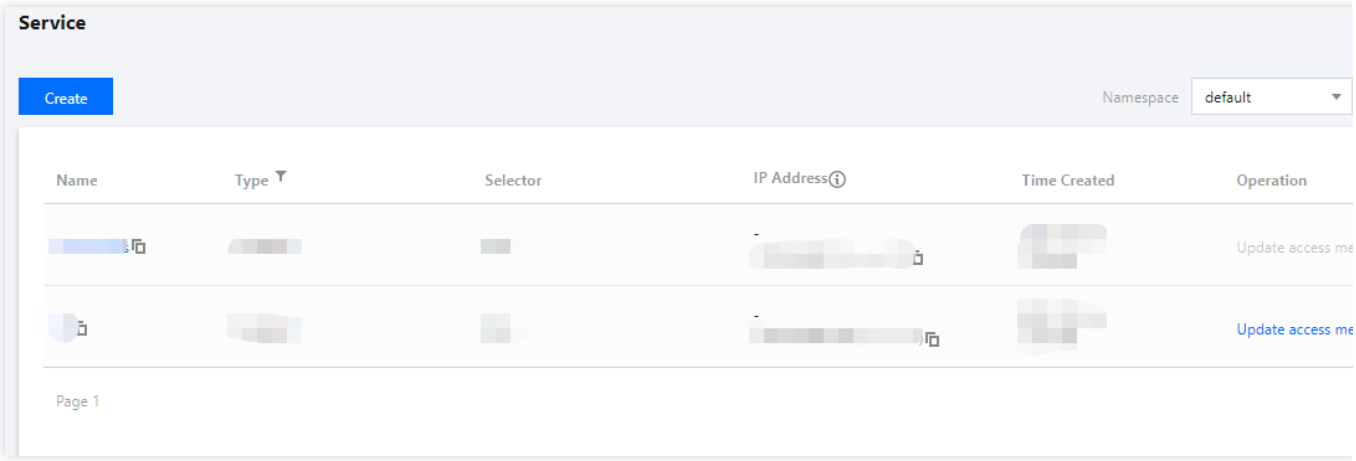
Step 3. Deploy the application

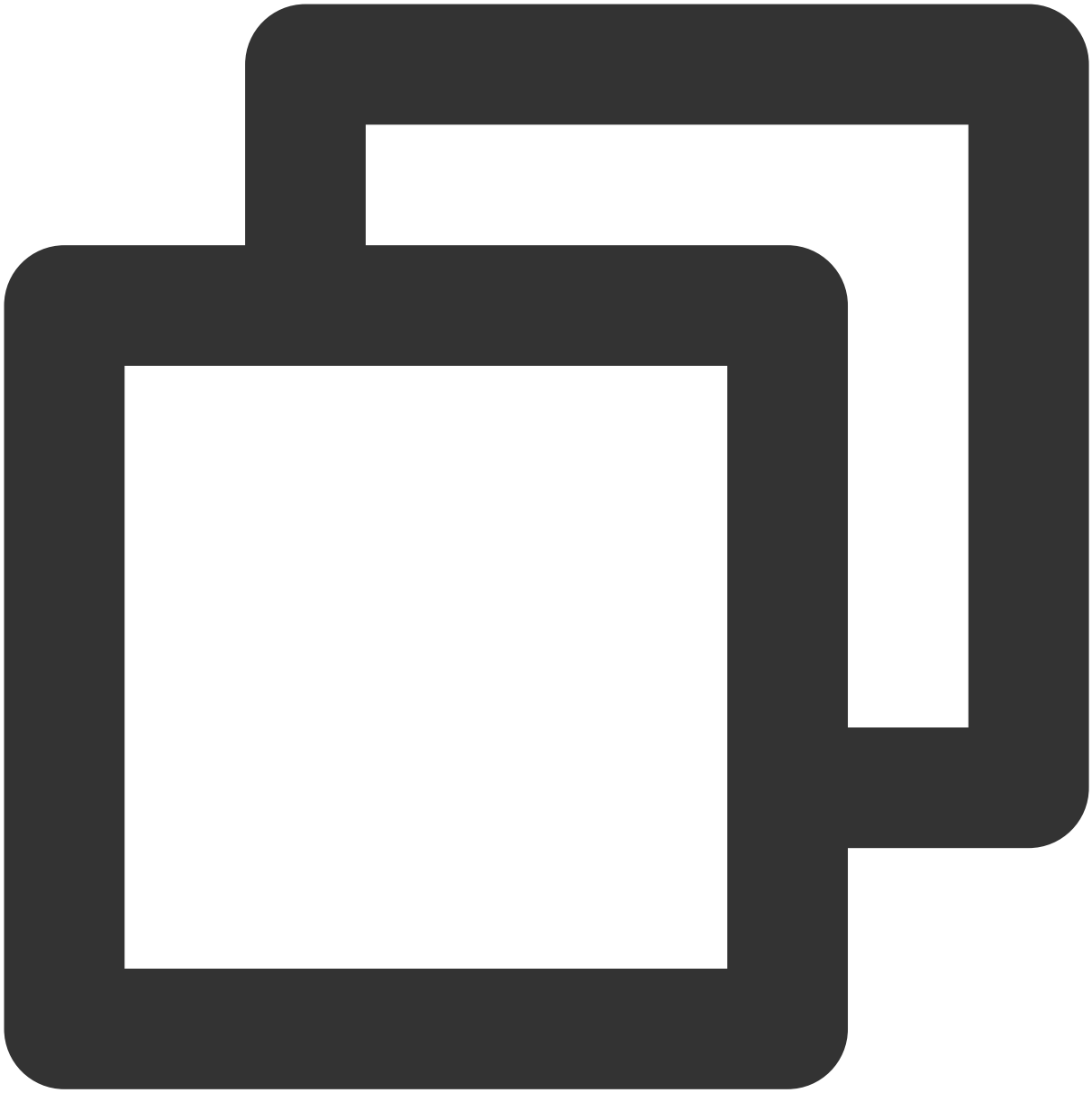
1. Log in to the TKE console and select the container cluster for deployment.
2. Click **Workload** > **Deployment** to enter the Deployment management page and select the corresponding namespace to deploy the service. Here, a workload is created in the console, and Service access is also enabled. You can also create one on the command line.





3. Add K8s labels to the corresponding Service. If the workload is created on the command line, you can directly add labels. Here, the configuration is adjusted in the TKE console. Select the TKE cluster that needs to be adjusted. Click **Services and Routes** > **Service** to enter the Service management page. Select the corresponding namespace to adjust the Service YAML configuration as shown below:





```
apiVersion: v1
kind: Service
metadata:
  labels: # Add the corresponding labels based on the actual conditions
  k8sapp: spring-mvc-demo
  name: spring-mvc-demo
  namespace: spring-demo
spec:
  ports:
  - name: 8080-8080-tcp  # Corresponding `port` value in the ServiceMonitor scrape
    port: 8080
```

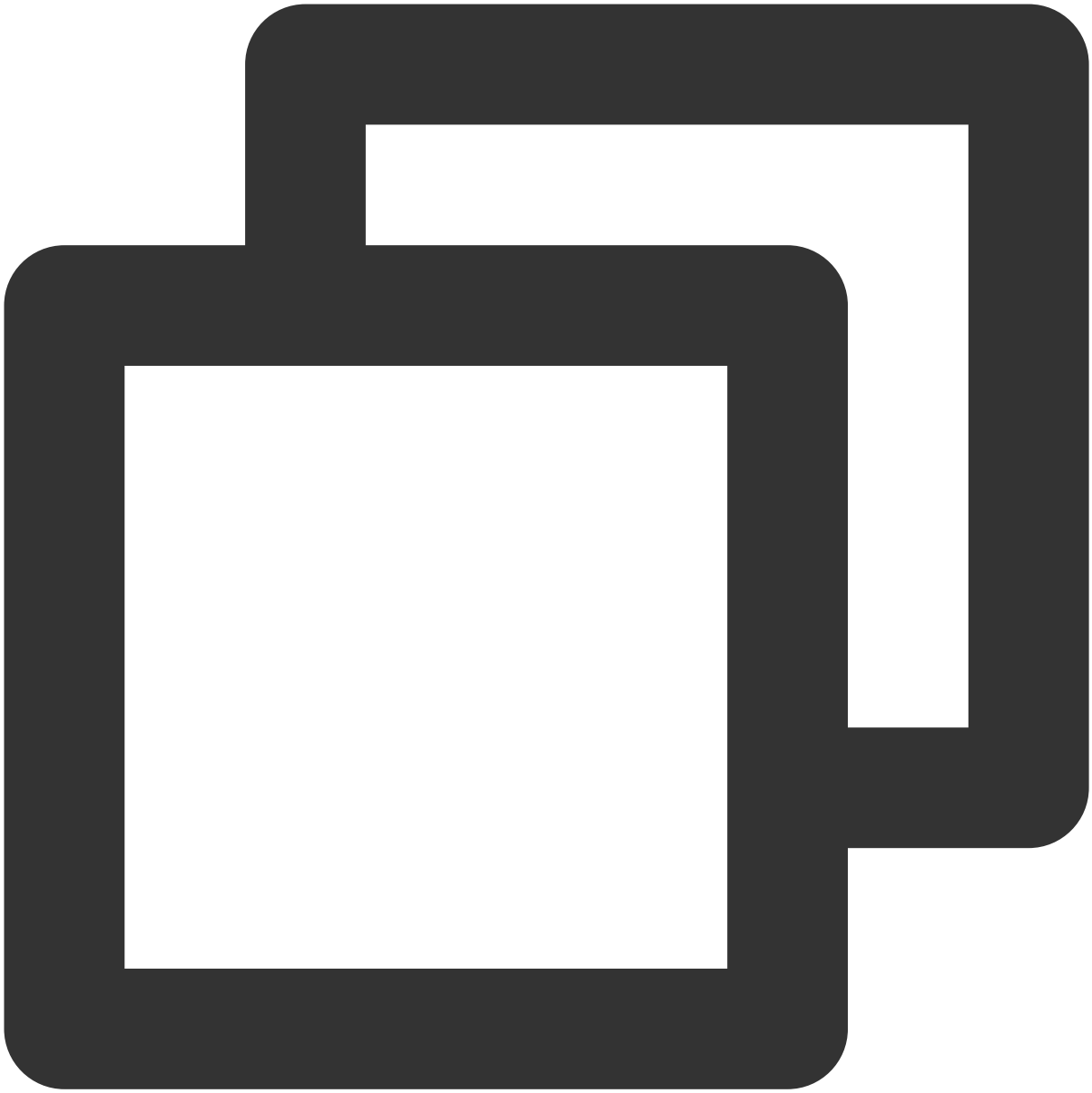
```
    protocol: TCP
    targetPort: 8080
  selector:
    k8s-app: spring-mvc-demo
    qcloud-app: spring-mvc-demo
  sessionAffinity: None
  type: ClusterIP
```

Step 4. Add a scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add a ServiceMonitor. Currently, TMP supports discovering the corresponding target instance address through labels; therefore, you can add some specific K8s labels to some services, which will be automatically identified by TMP after configuration, eliminating your need to add scrape tasks for all services one by one. The configuration information for the above sample is as follows:

Note:

Here, note that the `port` value is the `spec/ports/name` value in the Service YAML configuration file.



```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: spring-mvc-demo # Enter a unique name
  namespace: cm-prometheus # The namespace is fixed. Do not change it
spec:
  endpoints:
  - interval: 30s
    port: 8080-8080-tcp # Enter the name of the corresponding port of the Prometh
    path: /actuator/prometheus  # Enter the value of the corresponding path of th
  namespaceSelector:  # Select the namespace where the Service to be monitored re
```

```
      matchNames:
      - spring-demo
    selector: # Enter the label value of the Service to be monitored to locate the
      matchLabels:
        k8sapp: spring-mvc-demo
```

Step 5. View the monitoring information

Access the Grafana address of your TMP instance to view the application monitoring dashboard in **Dashboards > Manage > Application**.

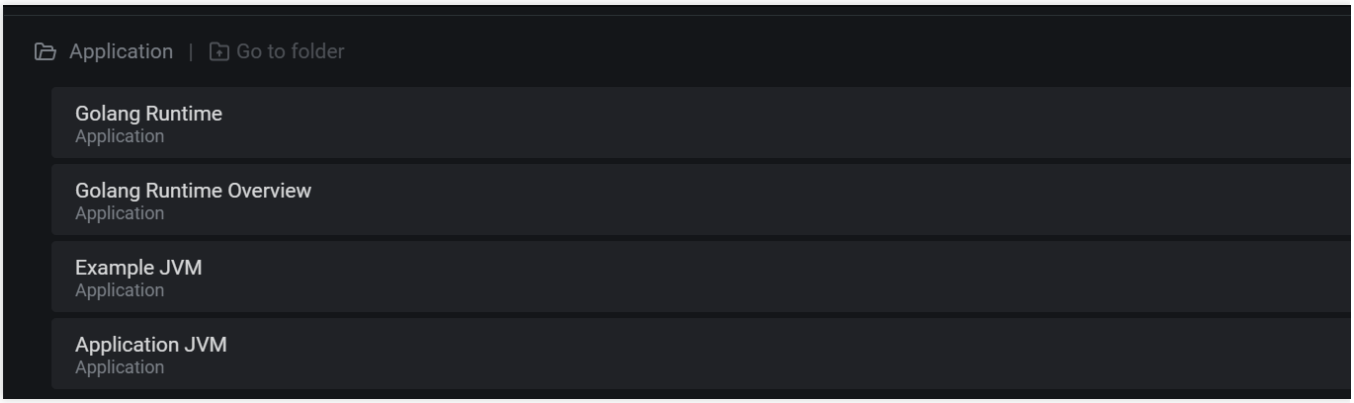
Spring MVC application: monitoring data of MVC status, such as the request latency, number of requests, success rate, and exception distribution.

Spring MVC API: API-level monitoring data, which supports multiple APIs to help you locate faulty APIs.

Tomcat: monitoring dashboard of internal Tomcat status, such as thread usage.

Application JVM: monitoring data of the status of all instances under an application. If you find a faulty instance, you can view its monitoring information at any time.

Instance JVM: detailed monitoring data of a single instance JVM.



JVM Integration

Last updated：2024-08-07 22:00:35

Overview

When using the Java programming language, you need to monitor JVM performance. TMP collects the JVM monitoring data exposed by applications and provides an out-of-the-box Grafana dashboard for it.
This document uses deploying a Java application in TKE as an example to describe how to use TMP to monitor the application status.
Note:
If you have already used Spring Boot as the development framework, please see Spring Boot Integration.

Prerequisites

Create a TKE cluster.
Use a private image repository to manage application images.

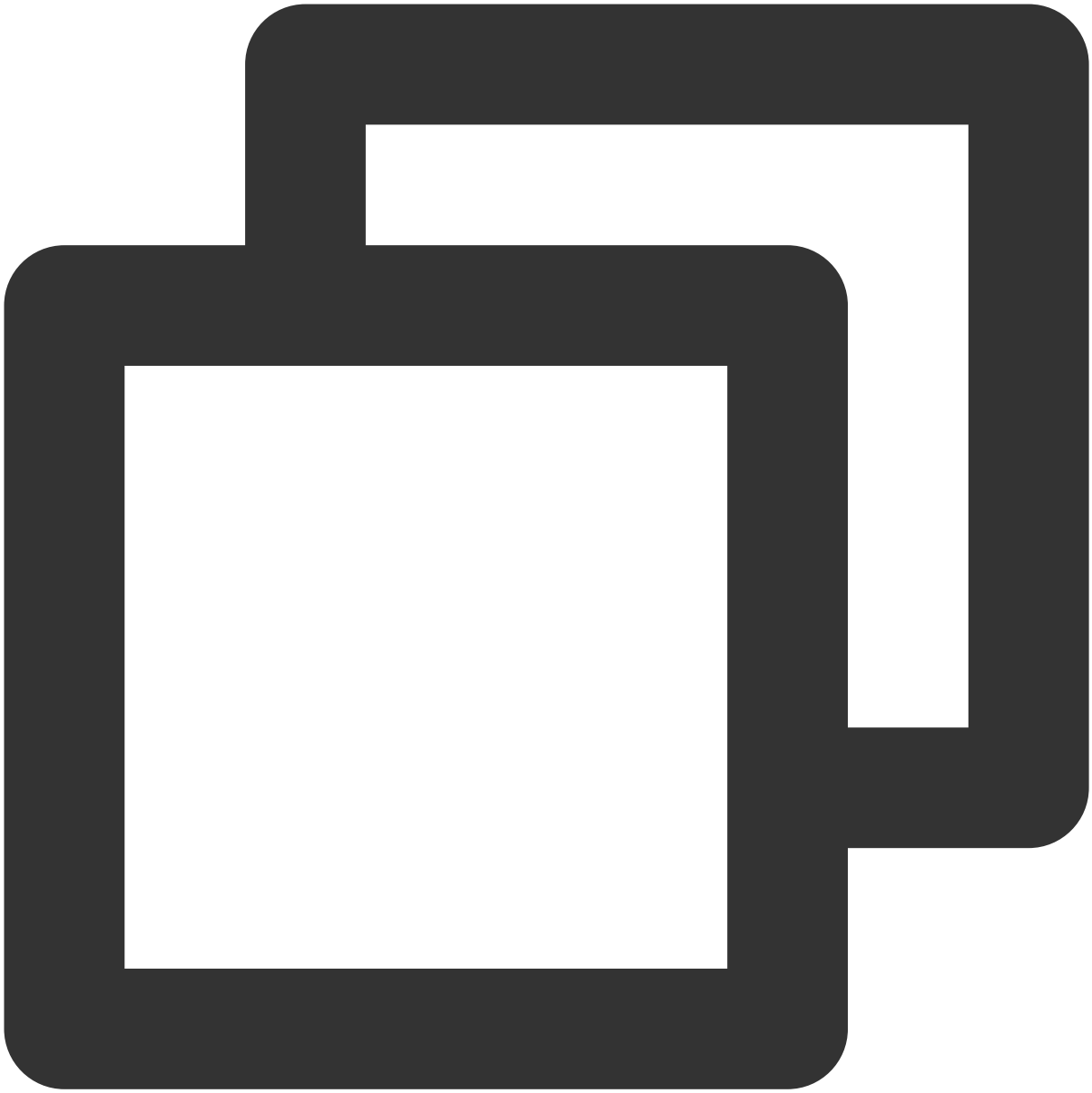
Directions

Note:
As a major programming language, Java has a comprehensive ecosystem, where Micrometer has been widely used as a metric timestamping SDK. This document uses Micrometer as an example to describe how to monitor JVM.

Modifying application dependencies and configuration

Step 1. Modify POM dependencies

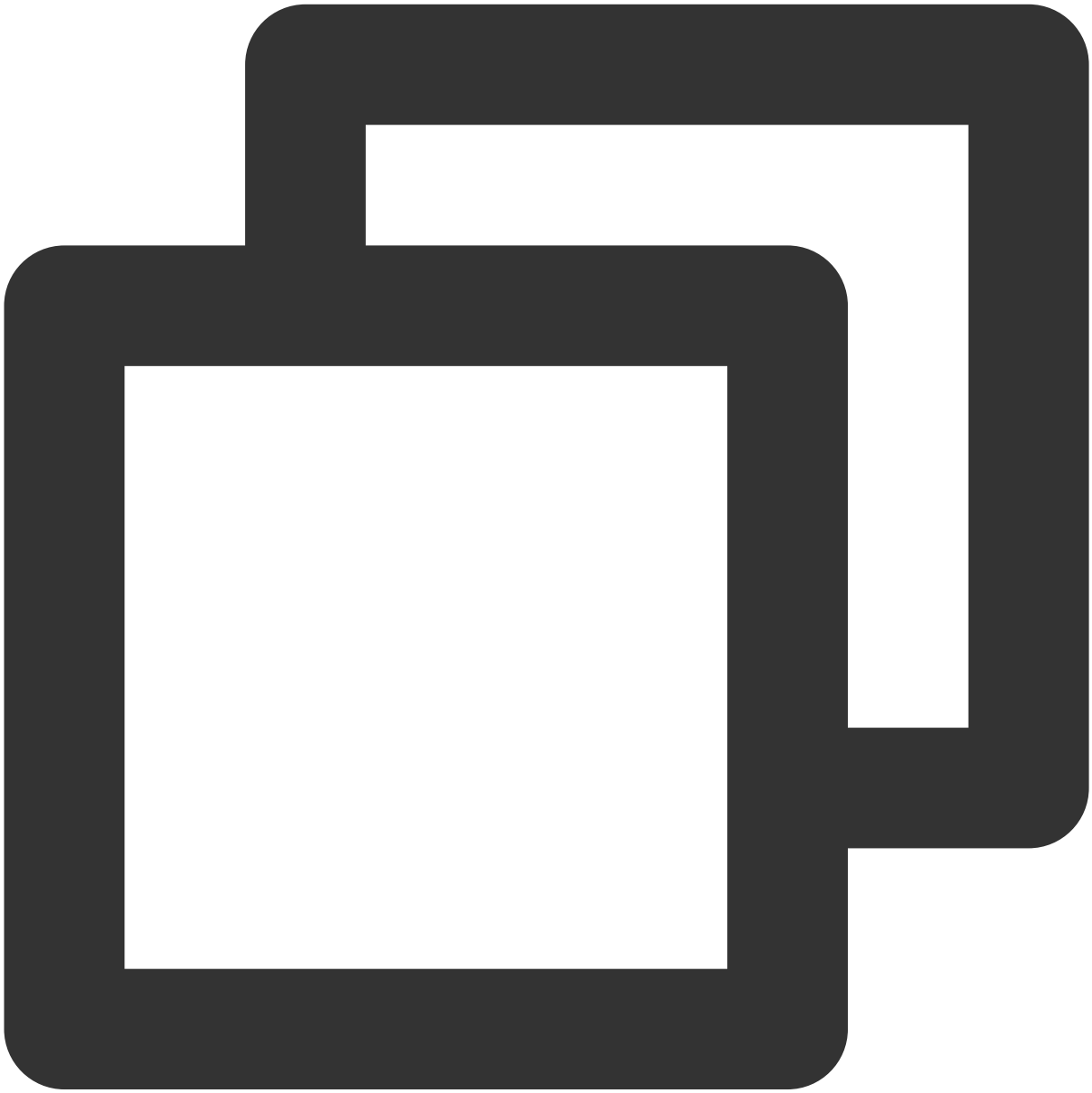
Add Maven dependencies to the `pom.xml` file and adjust the version as needed as follows:



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

Step 2. Modify the code

When the project is started, add the corresponding monitoring configuration. In addition, Micrometer also provides the collection of some common metrics, which are in the `io.micrometer.core.instrument.binder` package and can be added as needed as follows:



```
public class Application {
    // It can be used in custom monitoring as a global variable
    public static final PrometheusMeterRegistry registry = new PrometheusMeterRegis
    static {
```

```
        // Add a global Prometheus label. We recommend you add the corresponding ap
        registry.config().commonTags("application", "java-demo");
    }

    public static void main(String[] args) throws Exception {
        // Add JVM monitoring
        new ClassLoaderMetrics().bindTo(registry);
        new JvmMemoryMetrics().bindTo(registry);
        new JvmGcMetrics().bindTo(registry);
        new ProcessorMetrics().bindTo(registry);
        new JvmThreadMetrics().bindTo(registry);
        new UptimeMetrics().bindTo(registry);
        new FileDescriptorMetrics().bindTo(registry);
        System.gc(); // Test GC
        try {
            // Expose the Prometheus HTTP service. If it already exists, you can us
            HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
            server.createContext("/metrics", httpExchange -> {
                String response = registry.scrape();
                httpExchange.sendResponseHeaders(200, response.getBytes().length);
                try (OutputStream os = httpExchange.getResponseBody()) {
                    os.write(response.getBytes());
                }
            });

            new Thread(server::start).start();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Note:

As monitoring of JVM GC pauses is implemented through the GarbageCollector Notification mechanism, the monitoring data will be generated only after a GC occurs. The above sample actively calls `System.gc()` to make the test more straightforward.

Step 3. Perform local verification

After the application is started locally, you can access the metric data of the Prometheus protocol through `http://localhost:8080/metrics`.

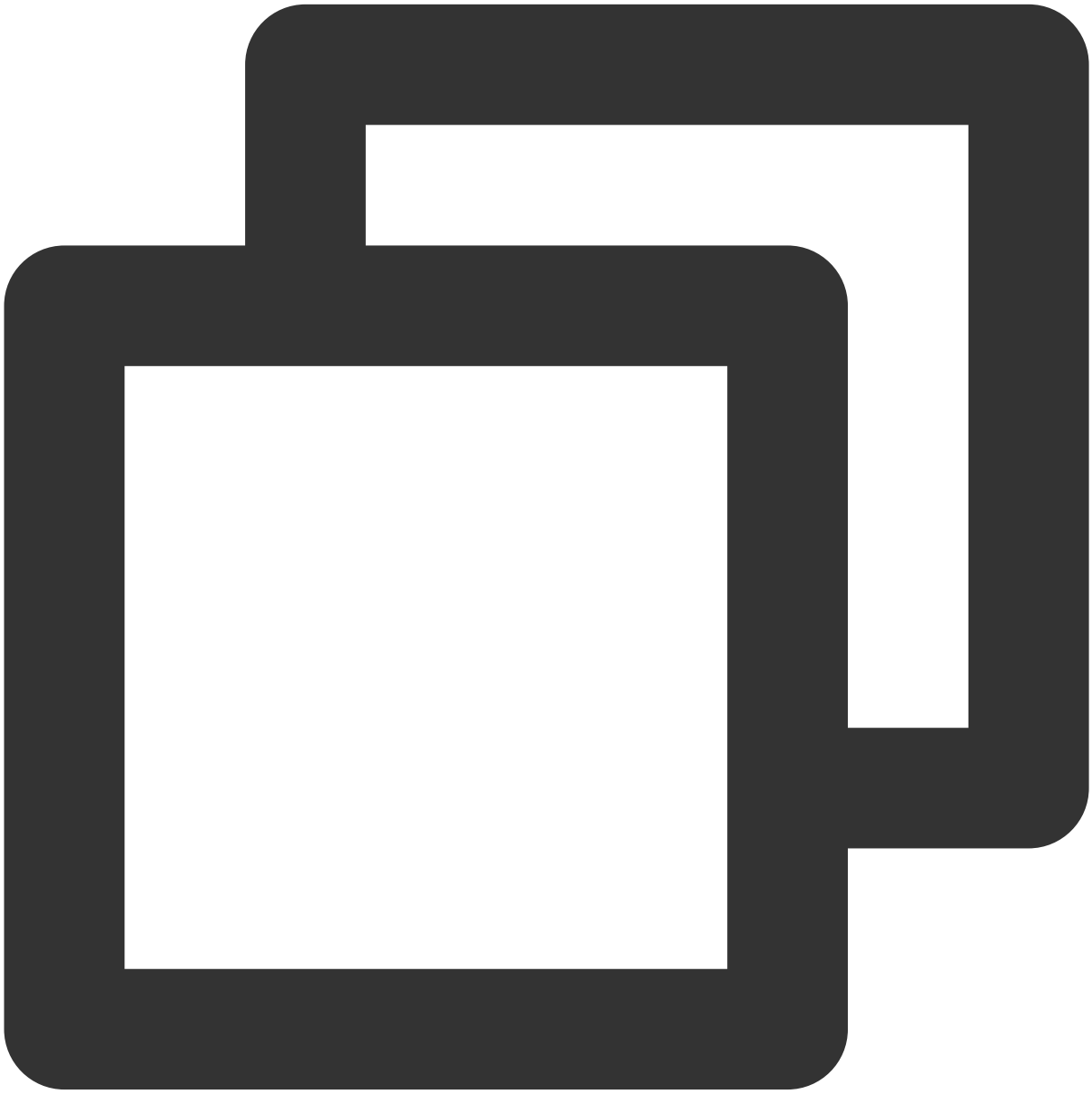
Releasing application to TKE

Step 1. Configure a Docker image environment locally

If you have already configured a Docker image environment locally, proceed to the next step; otherwise, configure one as instructed in Getting Started.

Step 2. Package and upload the image

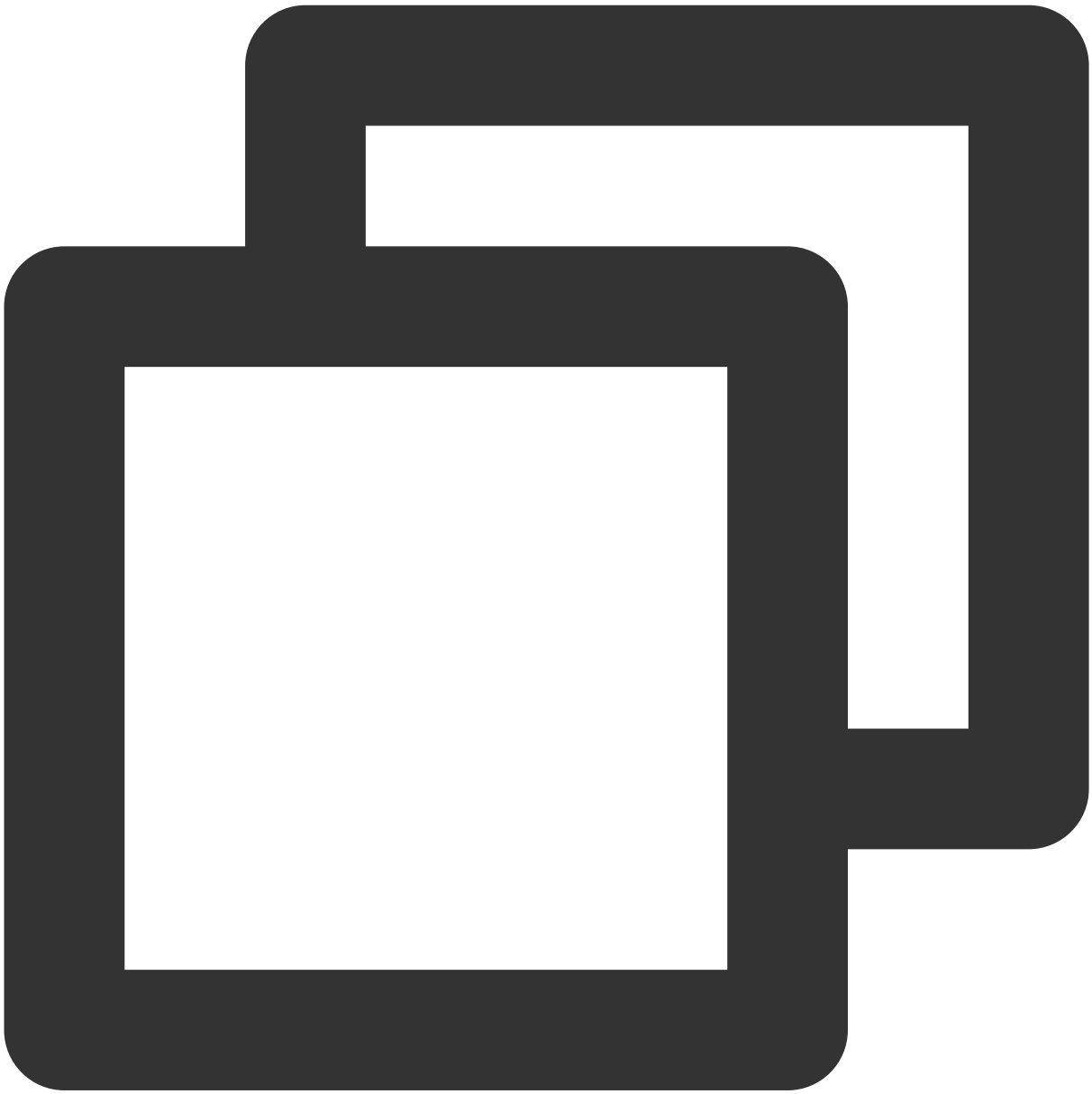
1. Add `Dockerfile` in the root directory of the project. Please modify it based on your actual project conditions as follows:



```
FROM openjdk:8-jdk
WORKDIR /java-demo
ADD target/java-demo-*.jar /java-demo/java-demo.jar
```

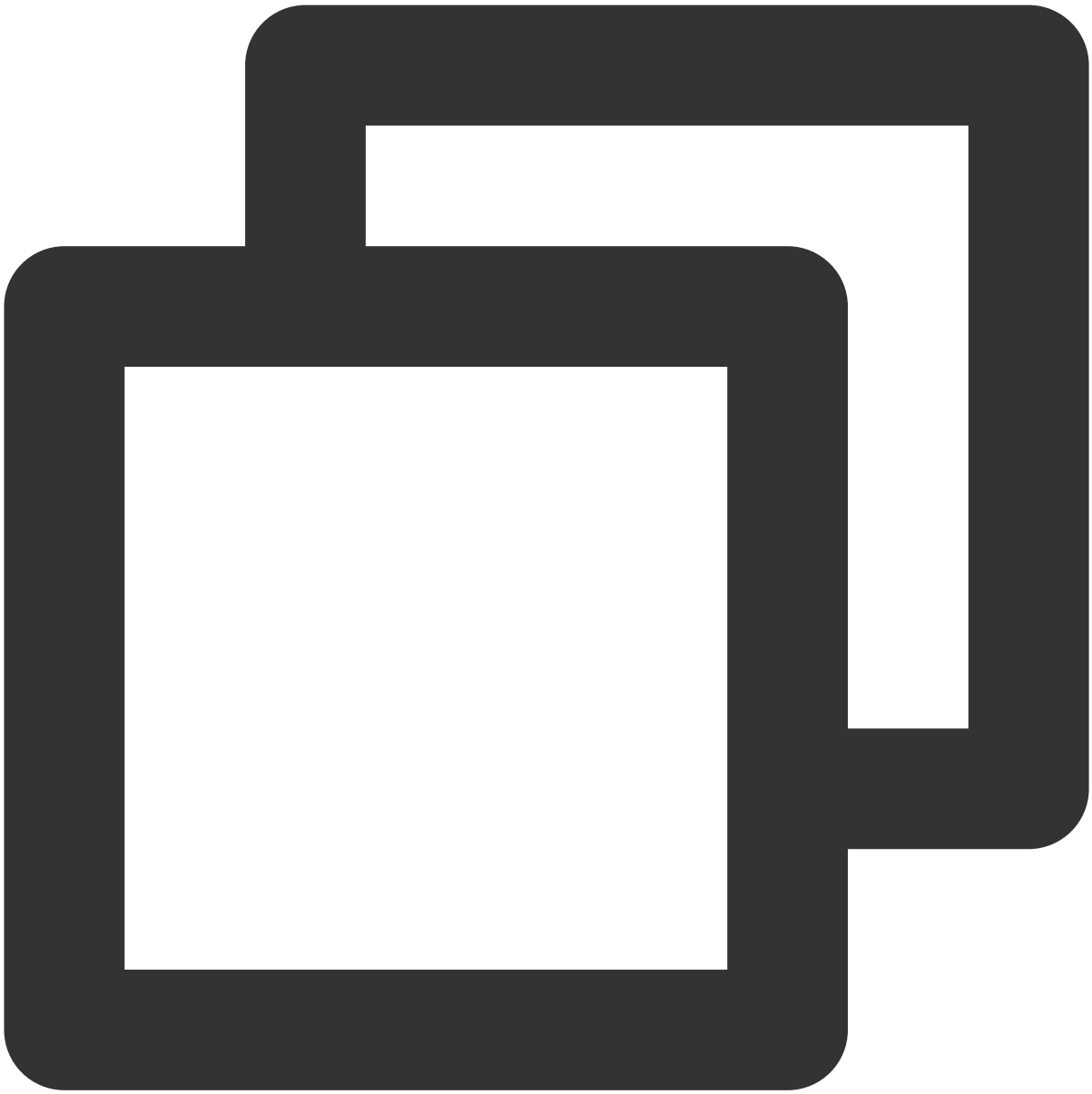
```
CMD ["java","-jar","java-demo.jar"]
```

2. Package the image by running the following command in the project root directory. You need to replace `namespace` , `ImageName` , and `image tag` as needed.



```
mvn clean package
 docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
 docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

Below is a sample:



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

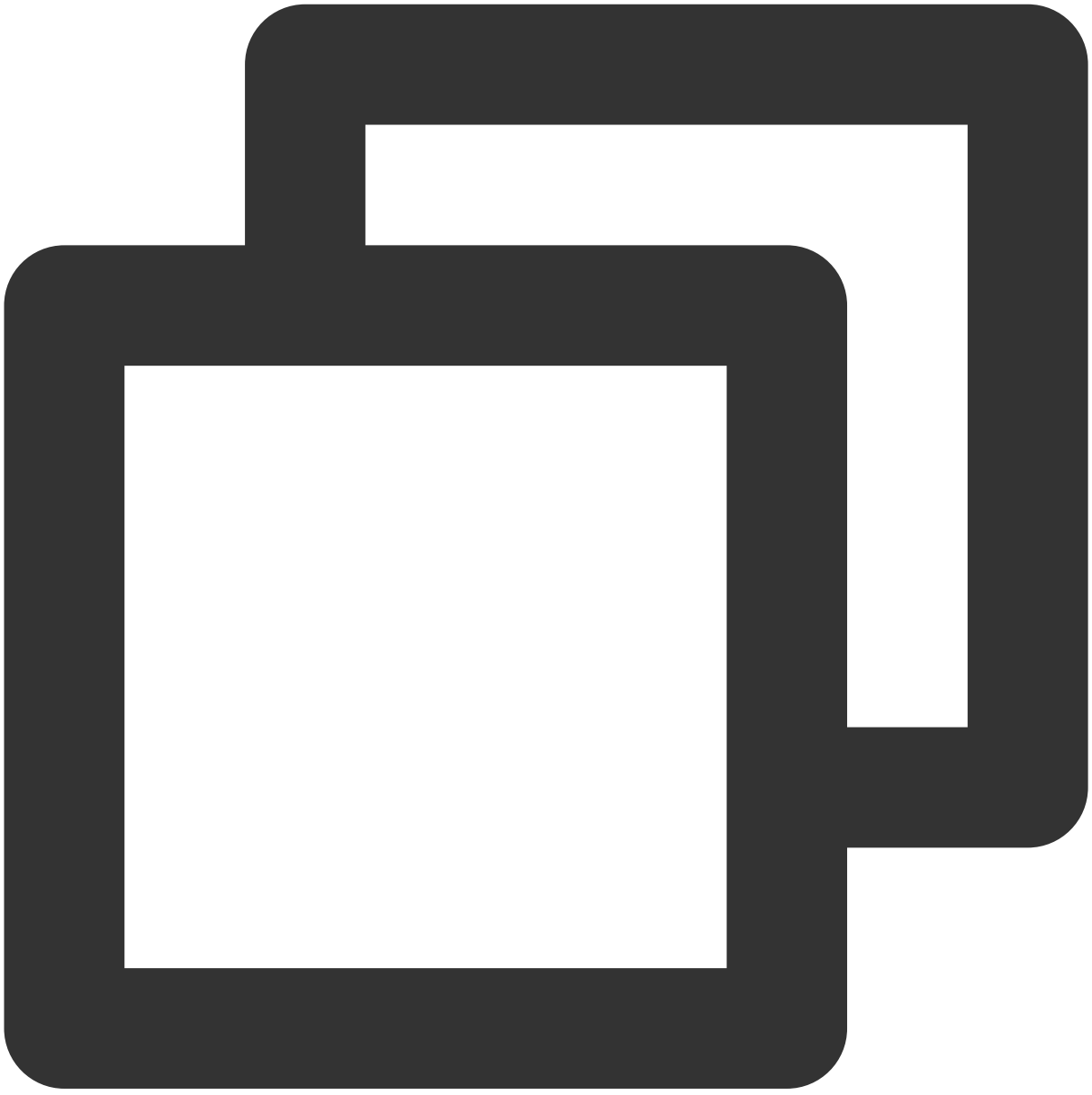
Step 3. Deploy the application

1. Log in to the TKE console and select the container cluster for deployment.

2. Select *Workload** > **Deployment** to enter the Deployment management page and select the corresponding `namespace` to deploy the service. Use the following YAML configuration to create the corresponding Deployment:

Note:

If you want to create in the console, please see Spring Boot Integration.

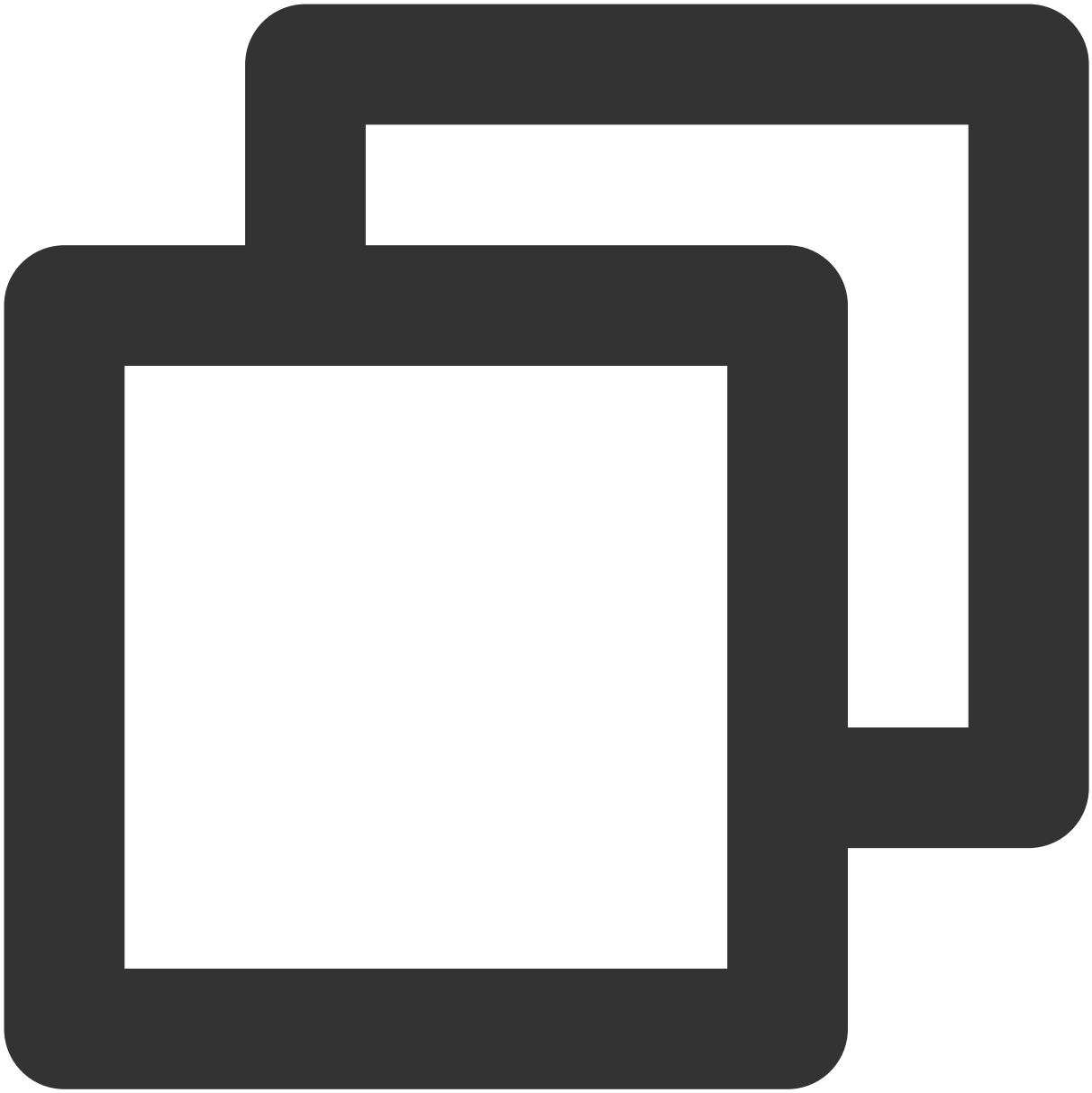


```
apiVersion: apps/v1
kind: Deployment
metadata:
    labels:
      k8s-app: java-demo
    name: java-demo
```

```
    namespace: spring-demo
spec:
    replicas: 1
    selector:
      matchLabels:
        k8s-app: java-demo
    template:
      metadata:
        labels:
          k8s-app: java-demo
    spec:
      containers:
      - image: ccr.ccs.tencentyun.com/prom_spring_demo/java-demo
        imagePullPolicy: Always
        name: java-demo
        ports:
        - containerPort: 8080
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      terminationGracePeriodSeconds: 30
```

Step 4. Add a scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:



```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: java-demo
  namespace: cm-prometheus
spec:
  namespaceSelector:
    matchNames:
    - java-demo
  podMetricsEndpoints:
  - interval: 30s
```

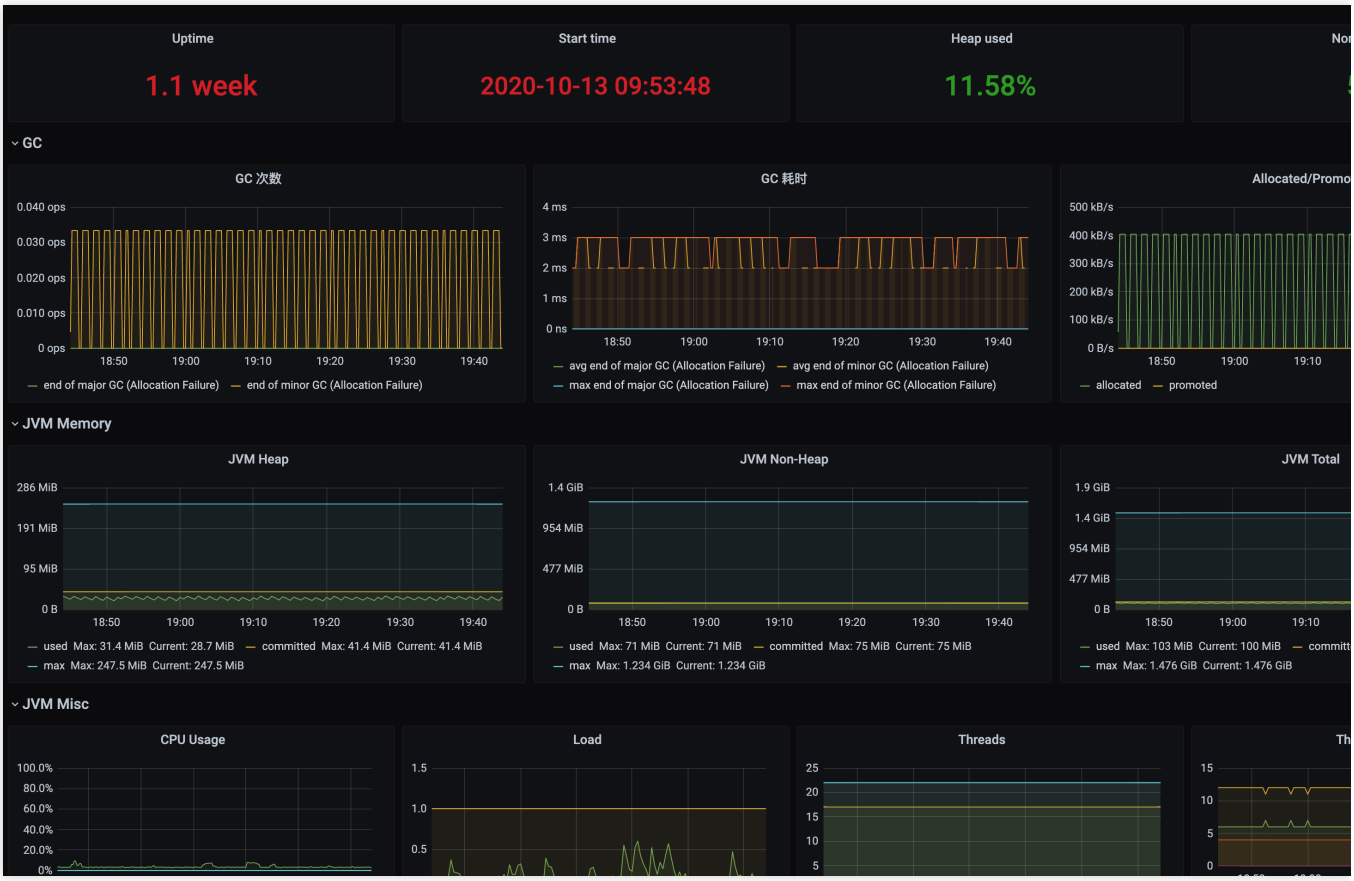
```
      path: /metrics
      port: metric-port
    selector:
      matchLabels:
        k8s-app: java-demo
```

Step 5. View the monitoring information

1. In **Integration Center** in the target TMP instance, find JVM monitoring, install the corresponding Grafana dashboard, and then you can enable the JVM monitoring dashboard.
2. Access the Grafana address of your TMP instance to view the application monitoring dashboard in **Dashboards > Manage > Application**.

Application JVM: monitoring data of the status of all instances under an application. If you find a faulty instance, you can view its monitoring information at any time.

Instance JVM: detailed monitoring data of a single instance JVM.



Go Application Integration

Last updated：2024-08-07 22:01:00

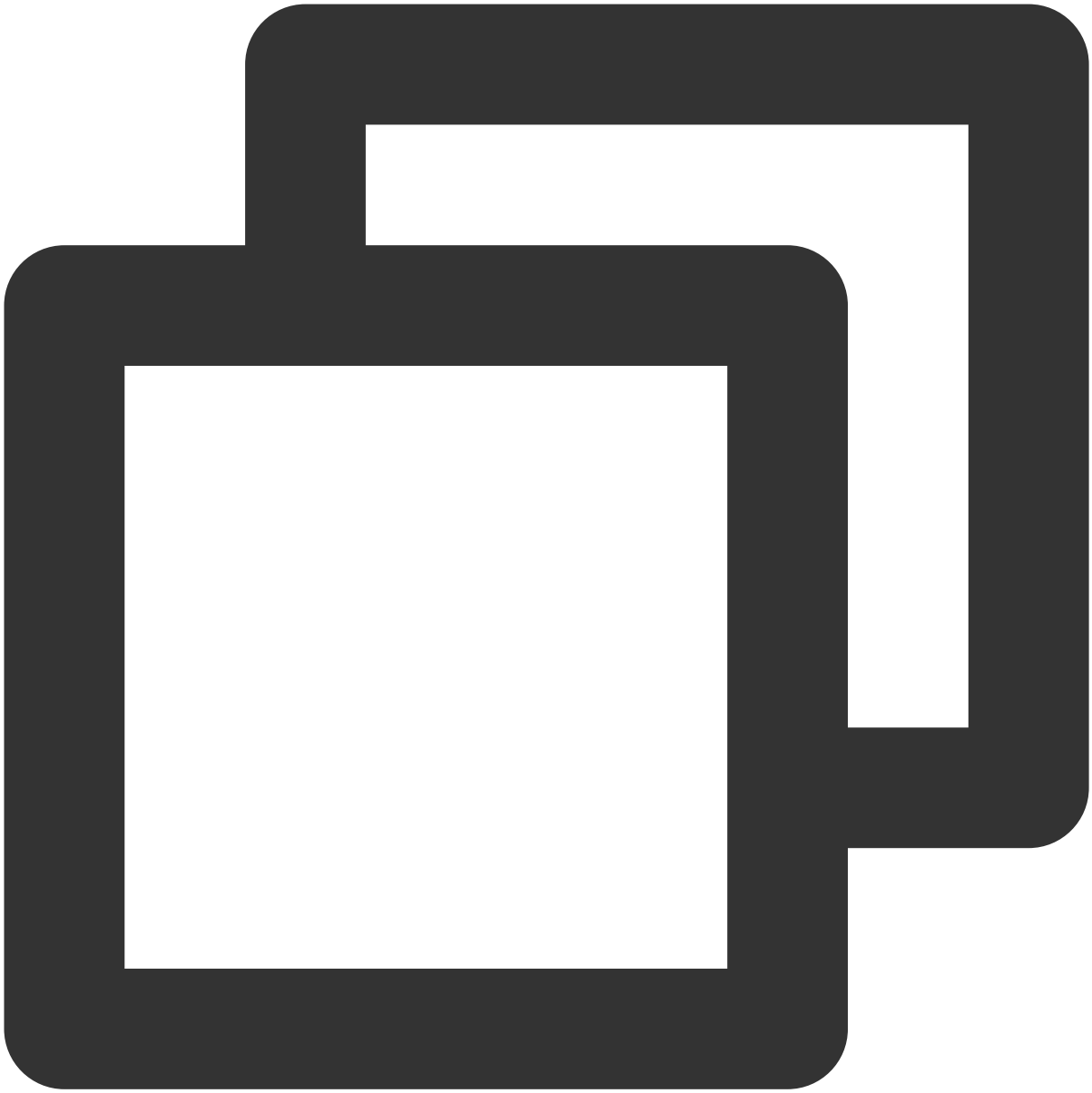
Prometheus provides an official Go library to collect and expose the monitoring data. This document describes how to use it to expose the Go runtime data and use TMP to collect metrics and display data with some basic samples.

Note:

For Go client API documentation, please see Prometheus Go client library.

Installation

You can run the following `go get` commands to install the relevant dependencies:

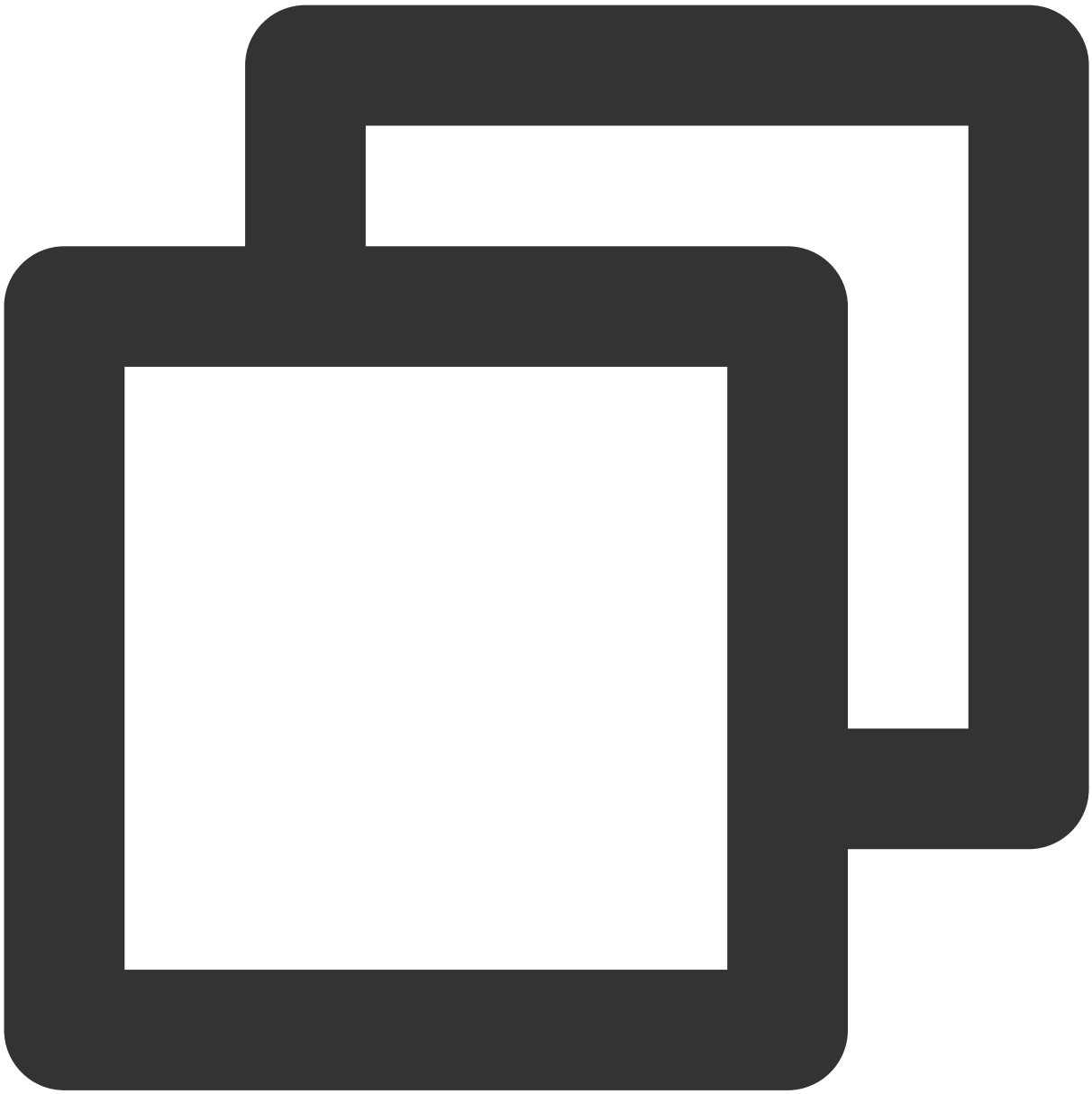


```
go get github.com/prometheus/client_golang/prometheus
go get github.com/prometheus/client_golang/prometheus/promauto
go get github.com/prometheus/client_golang/prometheus/promhttp
```

Start (Runtime Metrics)

1. Prepare an HTTP service with the commonly used path `/metrics` . You can directly use the `Handler` function provided in `prometheus/promhttp` .

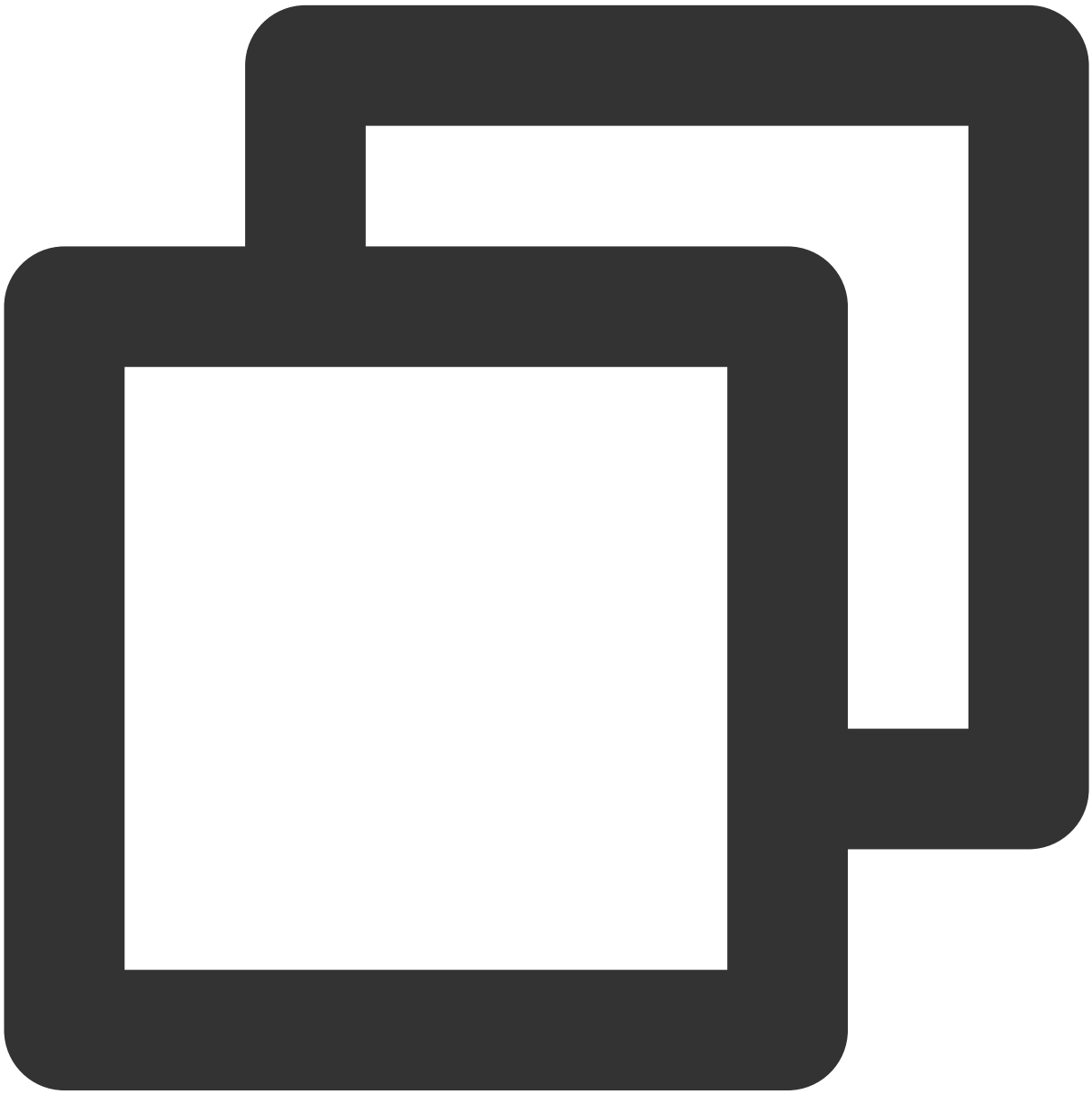
The following is a sample Go application, which exposes some default metrics (including runtime, process, and build metrics) through `http://localhost:2112/metrics` :



```
package main

import (
        "net/http"

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

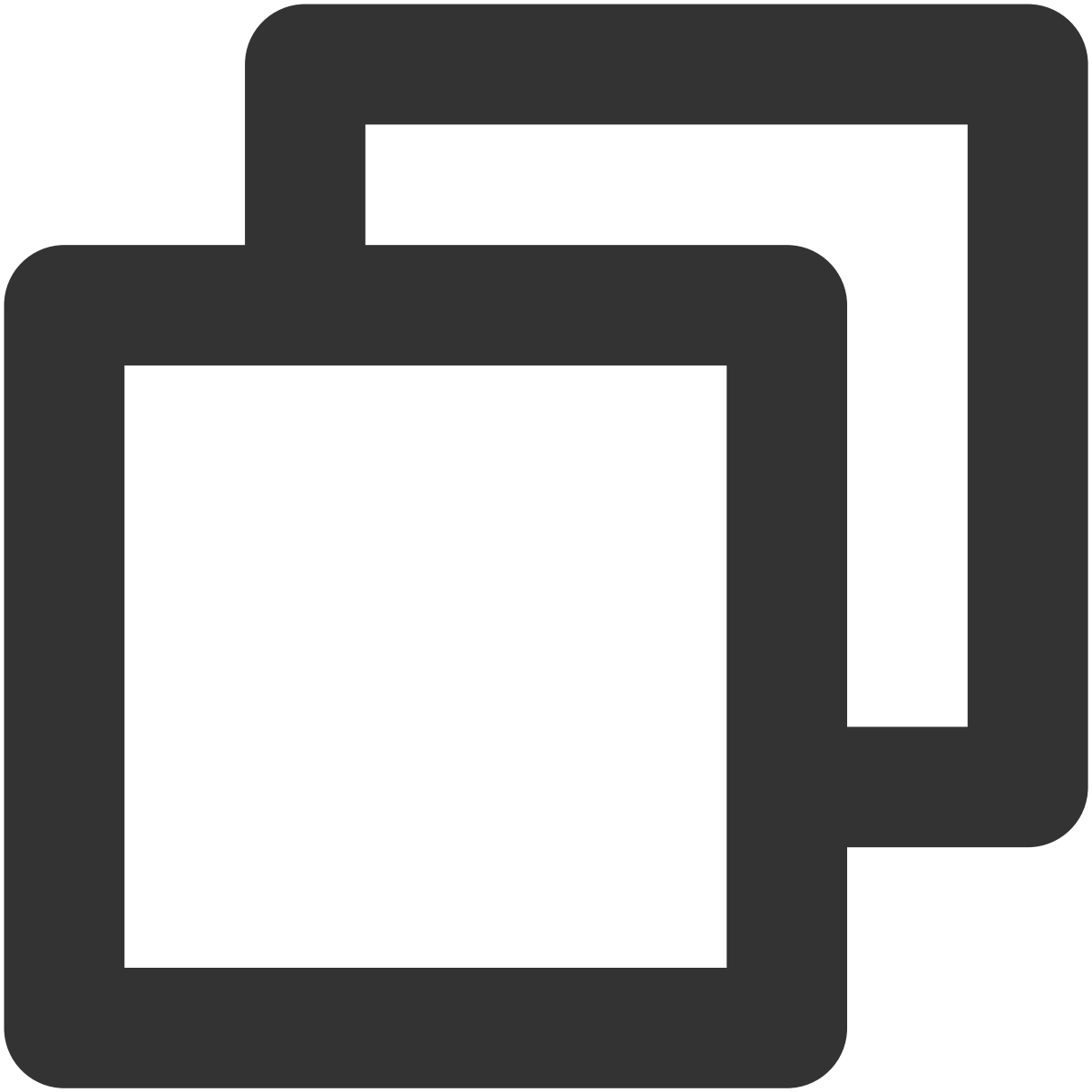
```
func main() {
        http.Handle("/metrics", promhttp.Handler())
        http.ListenAndServe(":2112", nil)
}
```

2. Run the following command to start the application:



```
go run main.go
```

3. Run the following command to access the basic built-in metric data:

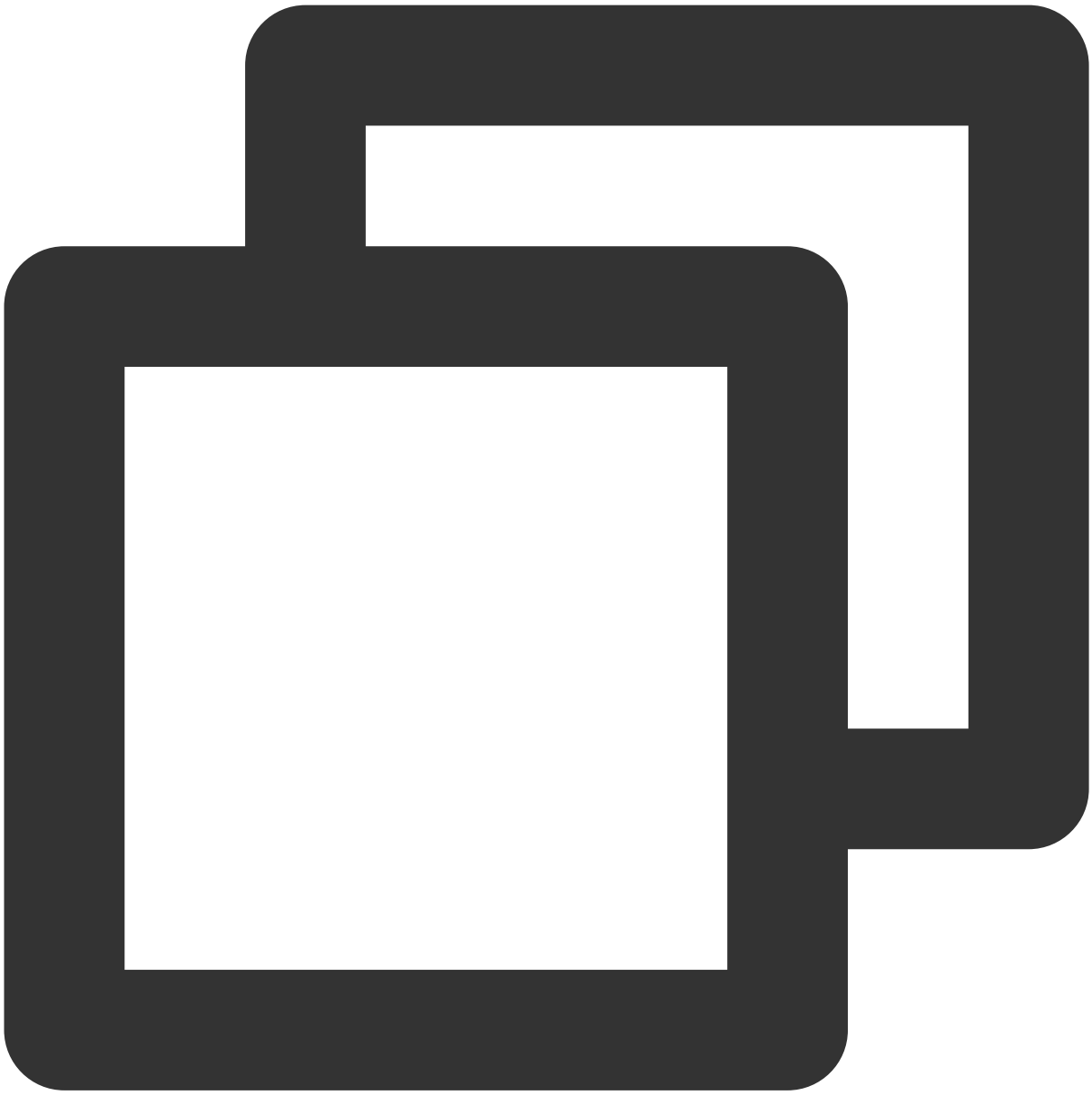


```
curl http://localhost:2112/metrics
```

Application Layer Metrics

1. The above sample only exposes some basic built-in metrics. For metrics at the application layer, you need to add them additionally (we will provide some SDKs in the future for easier integration). The following sample exposes a

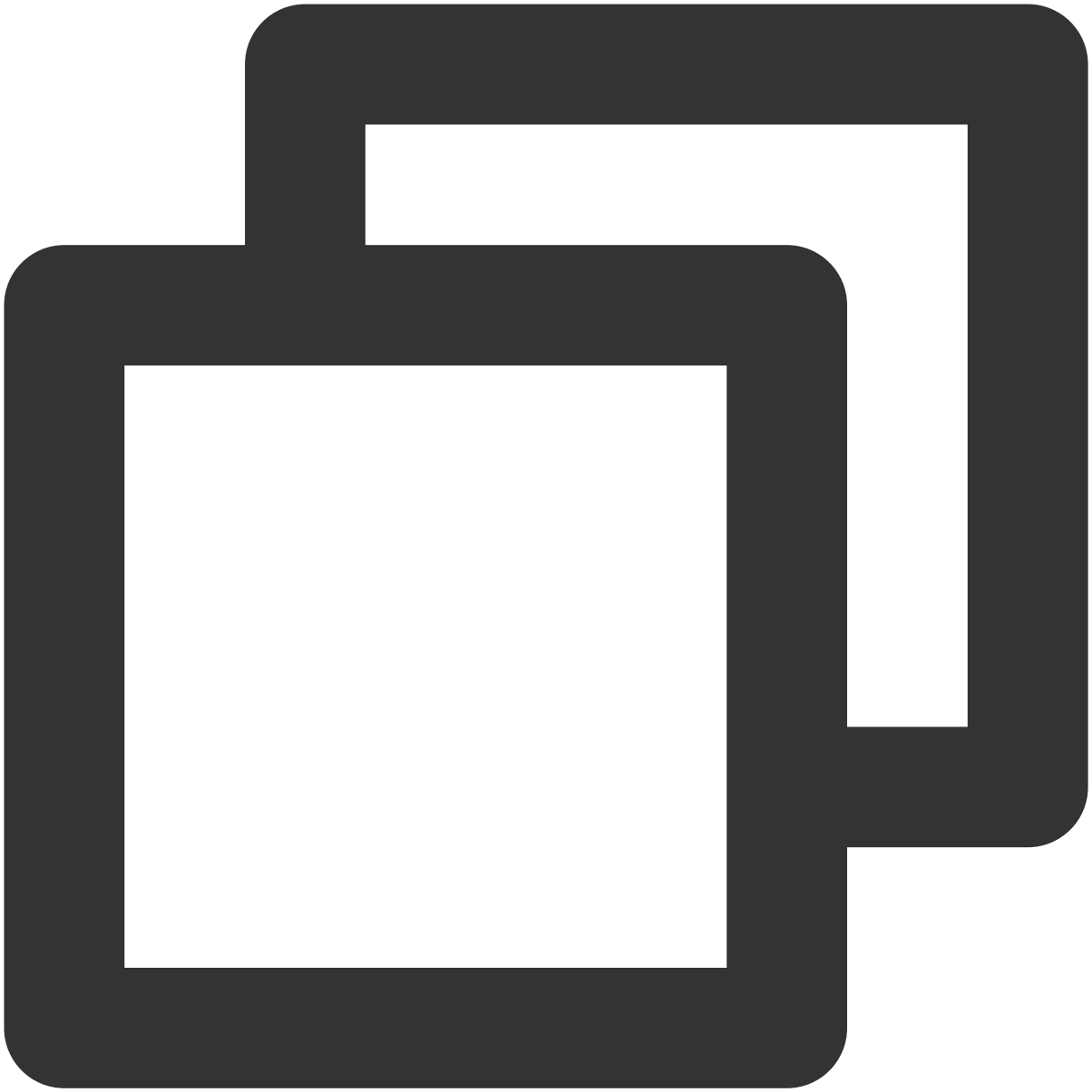
Counter metric named `myapp_processed_ops_total` to count the currently completed operations. The operation is performed once every 2 seconds, and the count increases by 1 each time:



```
package main

import (
        "net/http"
        "time"

        "github.com/prometheus/client_golang/prometheus"
        "github.com/prometheus/client_golang/prometheus/promauto"
```

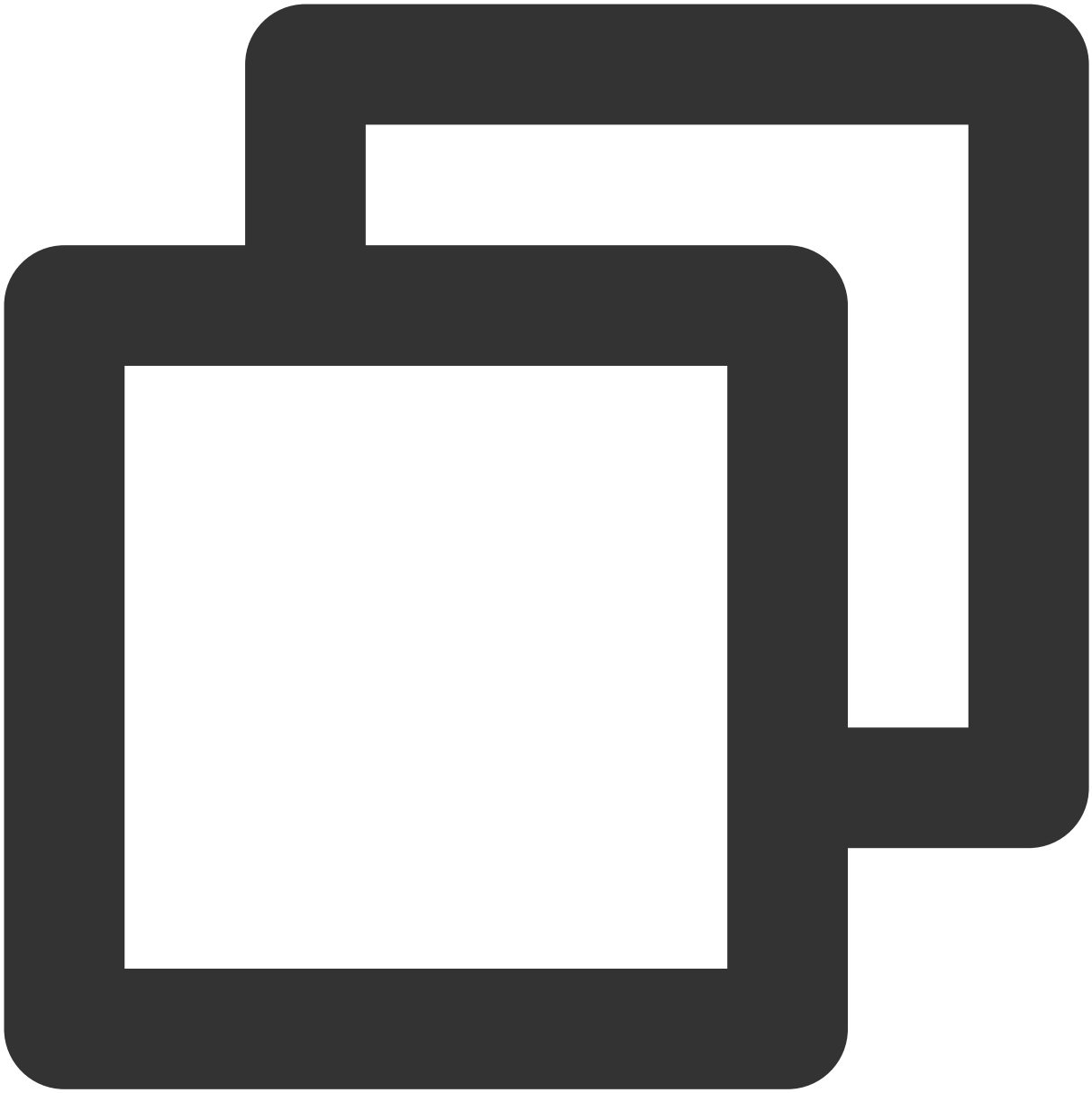
```
        "github.com/prometheus/client_golang/prometheus/promhttp"
)

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. Run the following command to start the application:



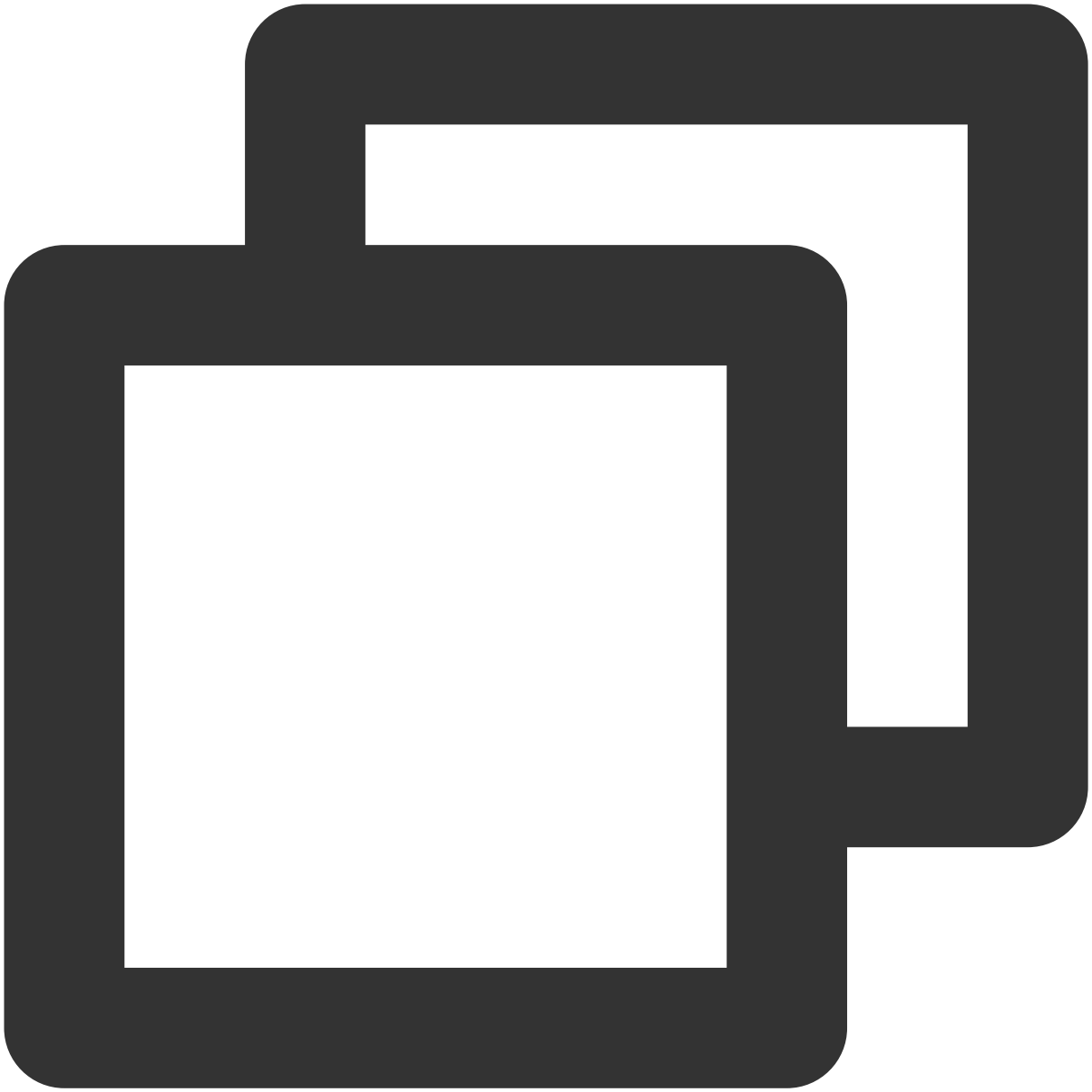
```
go run main.go
```

3. Run the following command to access the exposed metrics:



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

From the output result, you can see the information related to the `myapp_processed_ops_total` counter, including the help documentation, type information, metric name, and current value, as shown below:



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

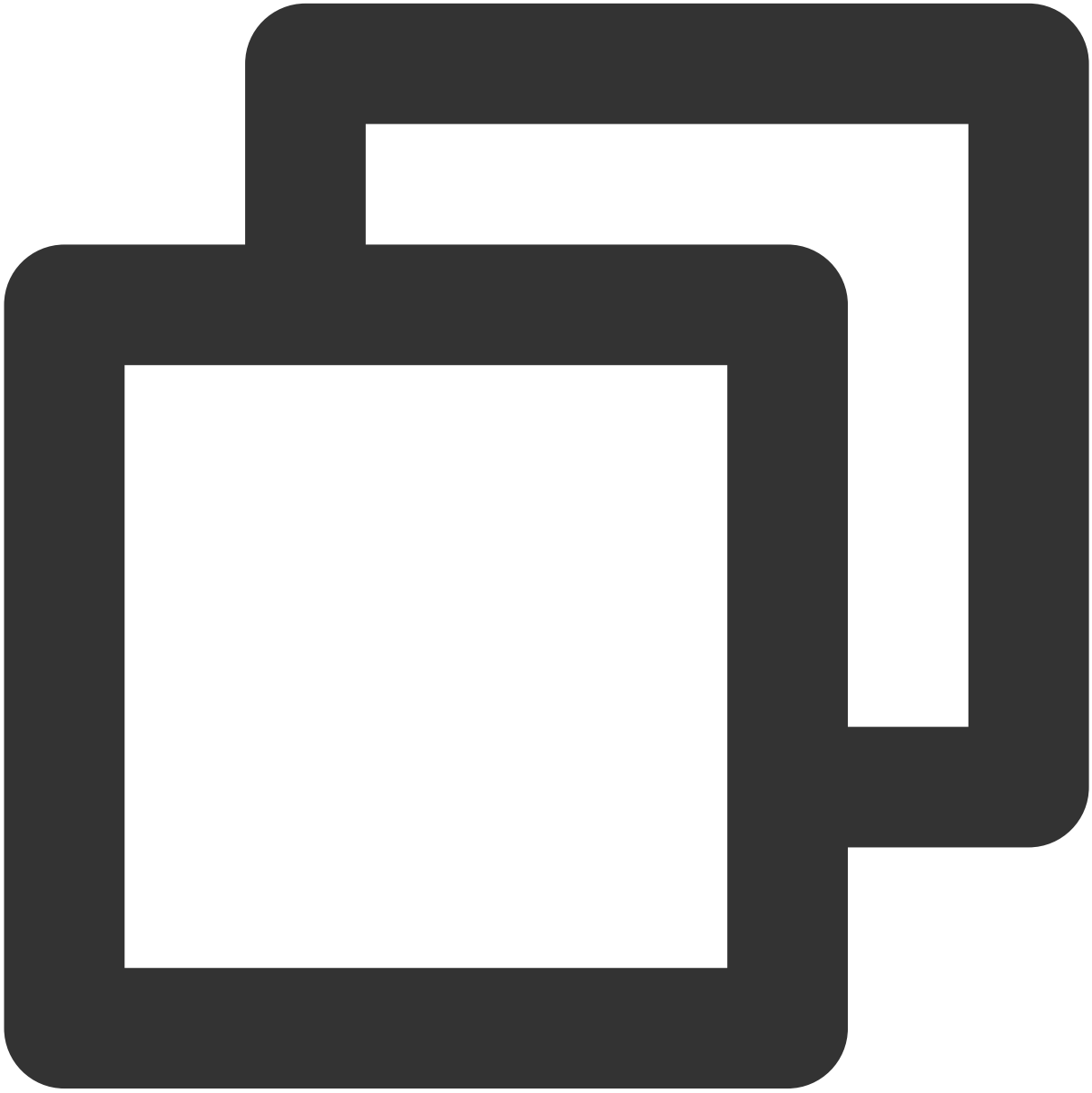
Using TMP

Two samples are used above to show how to use the Prometheus Go library to expose application metric data. However, because the exposed data is in text format, you'll need to set up and maintain an additional Prometheus

service to collect metrics, which may require additional Grafana dashboards for visual display.
In contrast, if you use TMP, you can directly skip the above steps and achieve the same purpose with just a few clicks. For more information, please see Getting Started.

Packaging and deploying application

1. A Go application generally can use a Dockerfile in the following format (it should be modified as needed):

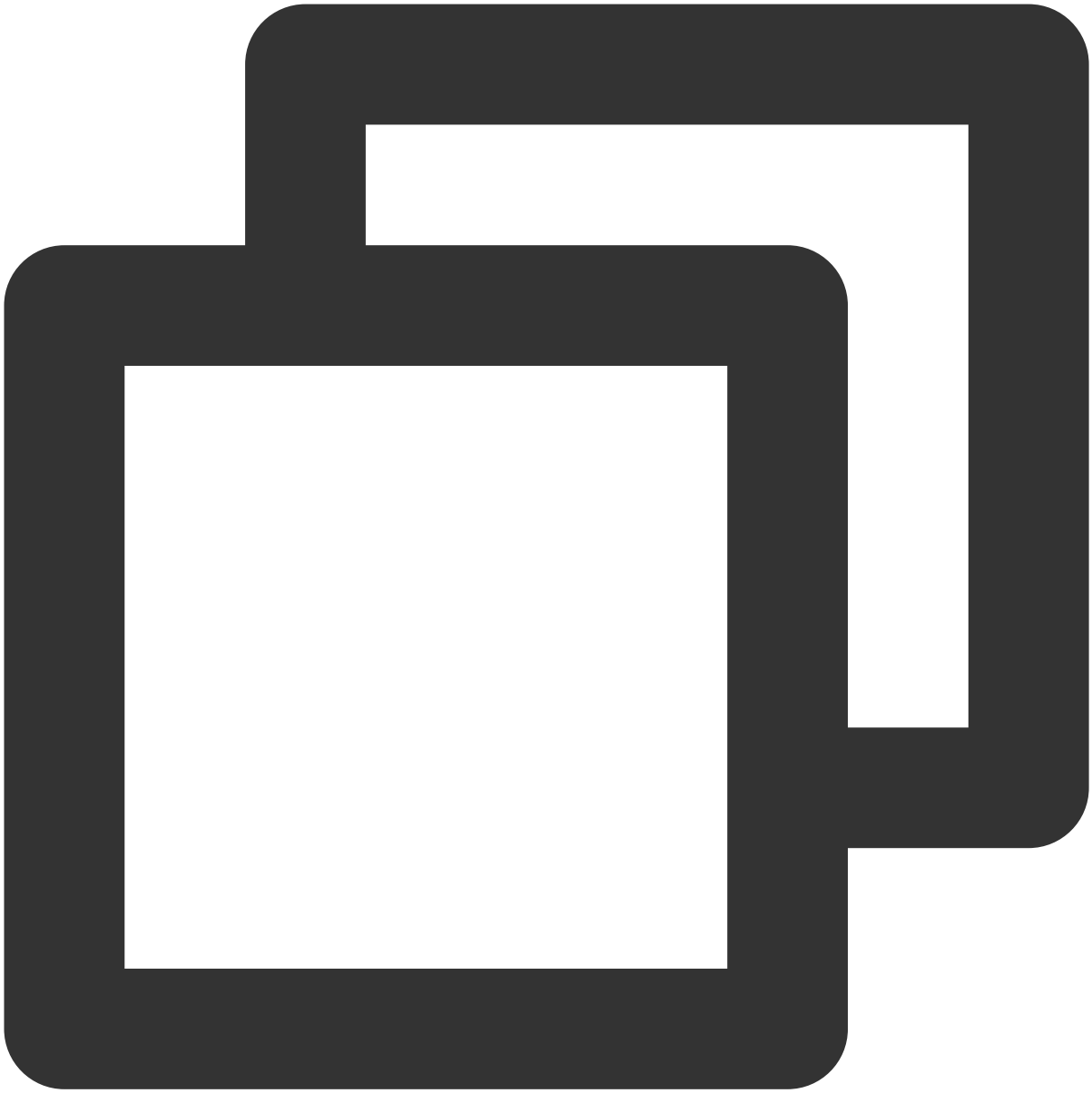


```
FROM golang:alpine AS builder
RUN apk add --no-cache ca-certificates \\
    make \\
```

```
        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. You can use an image from Tencent Cloud Image Registry or another public or self-built image registry.

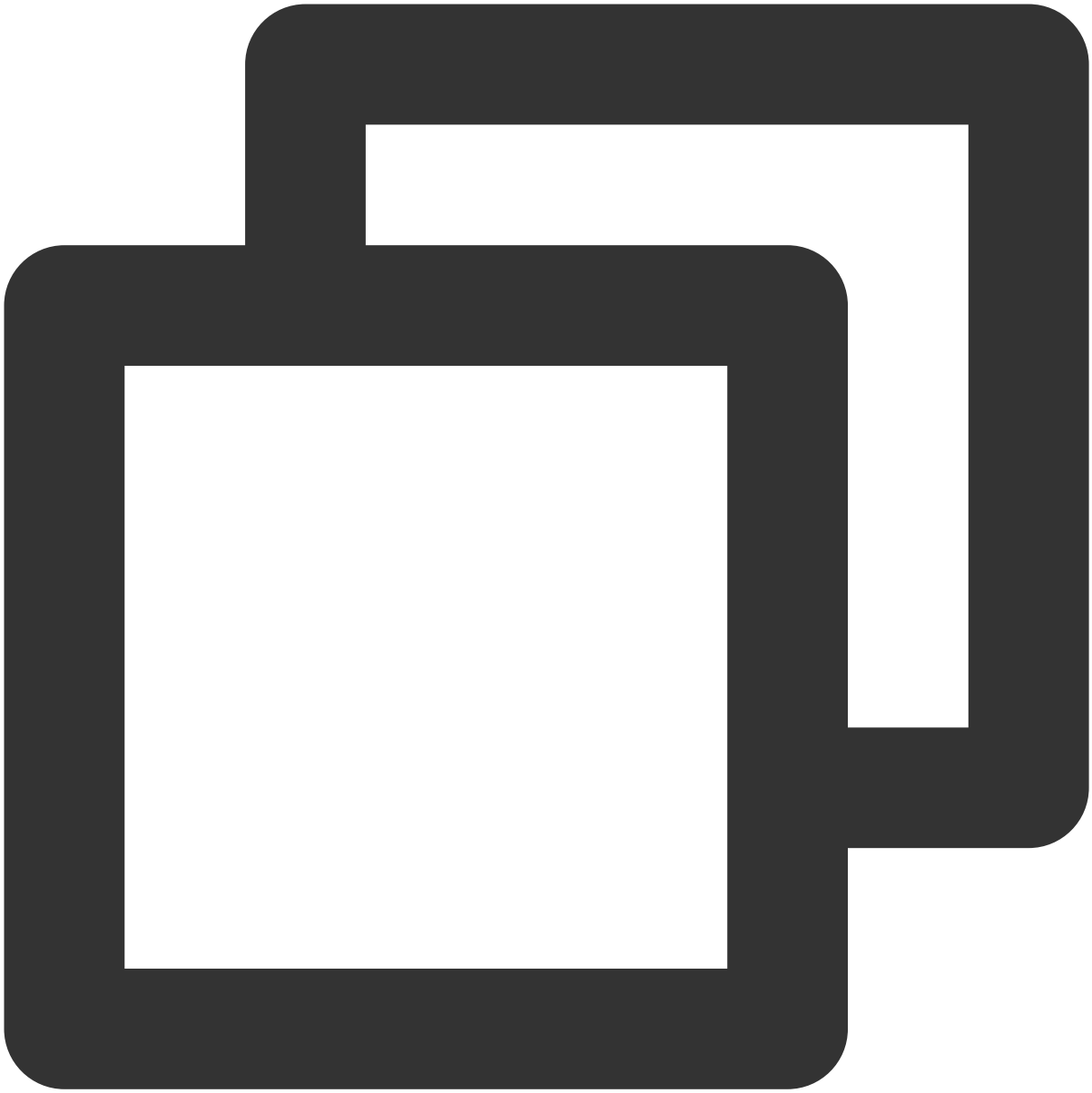
3. You need to define a Kubernetes resource based on your application type. Here, a Deployment is used as shown below:



```
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. You also need a Kubernetes Service for scrape configuration and load balancing.



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

Note:

You must add a label to identify the current application. The label name doesn't necessarily need to be app, but there must be a label with the similar meaning. You can add other extended labels by relabeling when adding a data collection task subsequently.

5. You can use the TKE console or directly use kubectl to submit the resource definitions to Kubernetes and wait for successful creation.

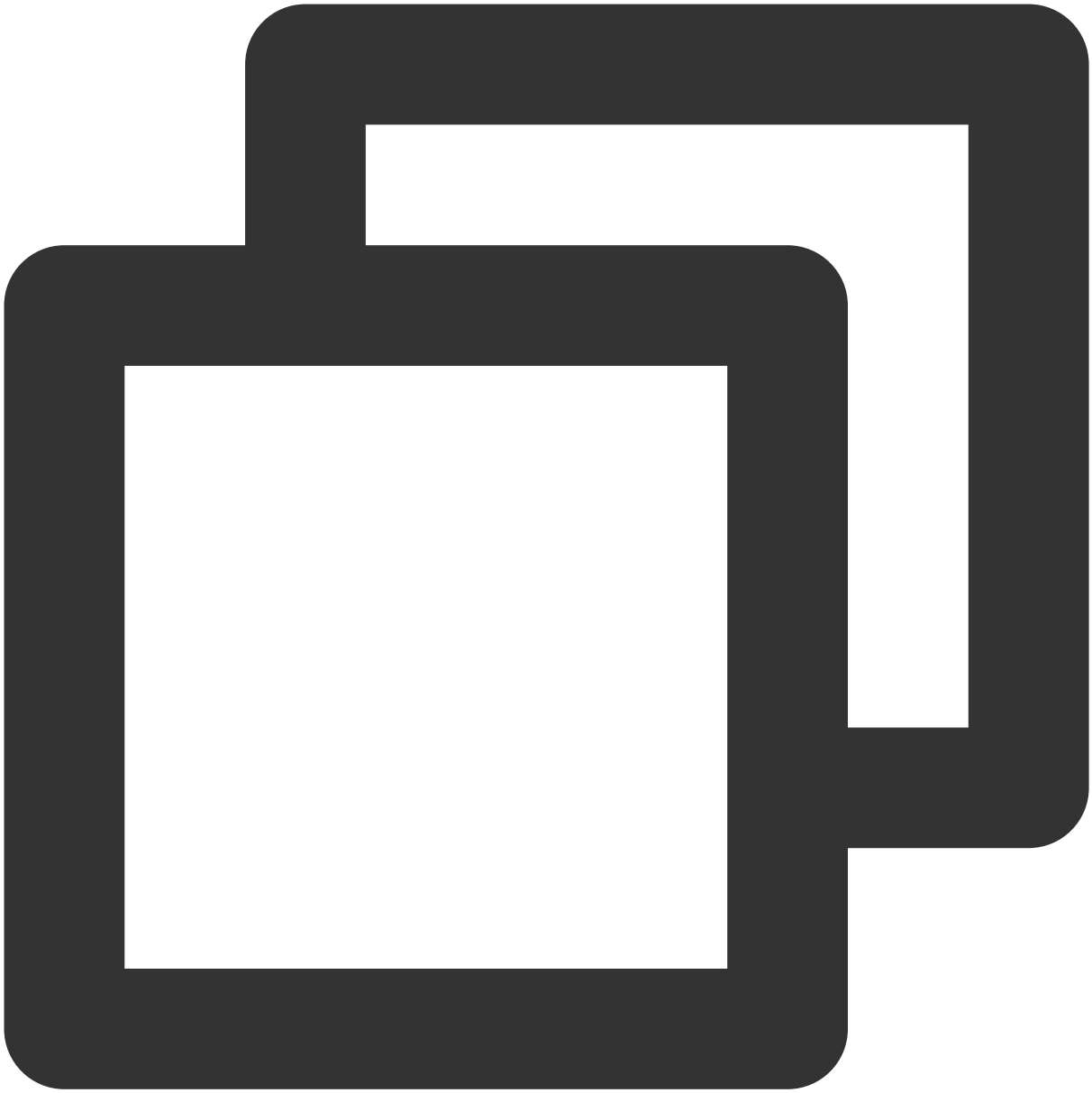
Adding data collection task

After the service runs, you need to configure TMP to discover and collect the monitoring metrics in the following steps:

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add a ServiceMonitor. Currently, TMP supports discovering the corresponding target instance address through labels; therefore, you can add some specific K8s labels to some services, which will be automatically identified by TMP after configuration, eliminating your need to add scrape tasks for all services one by one. The configuration information for the above sample is as follows:

Note:

The `port` value is the `spec/ports/name` value in the Service YAML configuration file.



```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: go-demo    # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  endpoints:
  - interval: 30s
    # Enter the name of the corresponding port of the Prometheus exporter in the
    port: 2112
    # Enter the value of the corresponding path of the Prometheus exporter. If it
```

```
    path: /metrics
    relabelings:
    # ** There must be a label named `application`. Here, suppose that K8s has a
    # Use the `replace` action of `relabel` to replace it with `application`
    - action: replace
      sourceLabels:  [__meta_kubernetes_pod_label_app]
      targetLabel: application
  # Select the namespace where the Service to be monitored resides
  namespaceSelector:
    matchNames:
    - golang-demo
    # Enter the label value of the Service to be monitored to locate the target S
  selector:
    matchLabels:
      app: golang-app-demo
```

Note:

You must configure the label named `application` in the sample; otherwise, you cannot use some other out-of-the-box integration features of TMP. For more advanced usage, please see ServiceMonitor or PodMonitor.

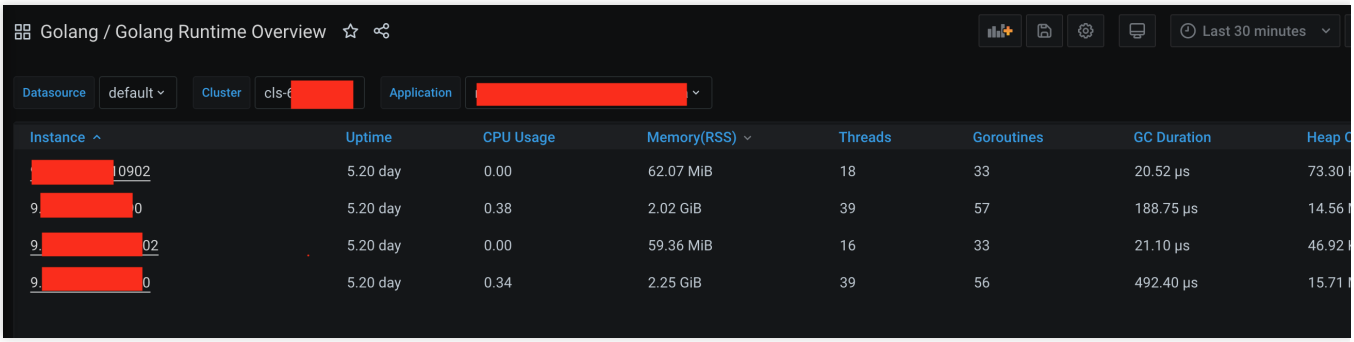
Viewing monitoring information

1. In the TMP instance list, find the corresponding TMP instance, click

on the right of the instance ID to open your Grafana page, and enter your account and password to access the Grafana visual dashboard operation section.

2. Enter Grafana, click the

icon to expand the monitoring dashboard, and click the name of the corresponding monitoring chart to view the monitoring data.



Summary

This document uses two samples to describe how to expose Go metrics to TMP and how to use the built-in visual charts to view monitoring data. This document only uses the Counter metrics. In other scenarios, you many need to use Gauge, Histogram, and Summary metrics. For more information, please see Metric Types.
For other use cases, TMP will integrate more frameworks to provide more out-of-the-box monitoring metrics, visual dashboards, and alerting templates.

Exporter Integration
Elasticsearch Exporter Integration

Last updated：2024-08-07 22:01:17

Overview

When using Elasticsearch, you need to monitor its running status, such as cluster and index status. TMP provides an exporter to monitor Elasticsearch and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to deploy the Elasticsearch exporter and integrate it with the alert feature.

Note:

For easier export installation and management, we recommend you use TKE for unified management.

Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance and created a namespace for the cluster. You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

Directions

Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. Perform the following steps to deploy an exporter: Using Secret to manage Elasticsearch connection string > Deploying Elasticsearch exporter > Verifying.

Using Secret to manage Elasticsearch connection string

1. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
2. In the top-right corner of the page, click **Create via YAML** to create a YAML configuration as detailed below:

You can use Kubernetes Secrets to manage and encrypt passwords. When starting the Elasticsearch exporter, you can directly use the Secret key but need to adjust the corresponding URI. Below is a sample YAML configuration:

Overview

When using Elasticsearch, you need to monitor its running status, such as cluster and index status. TMP provides an exporter to monitor Elasticsearch and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to deploy the Elasticsearch exporter and integrate it with the alert feature.

Note:

For easier export installation and management, we recommend you use TKE for unified management.

Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance and created a namespace for the cluster. You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

Directions

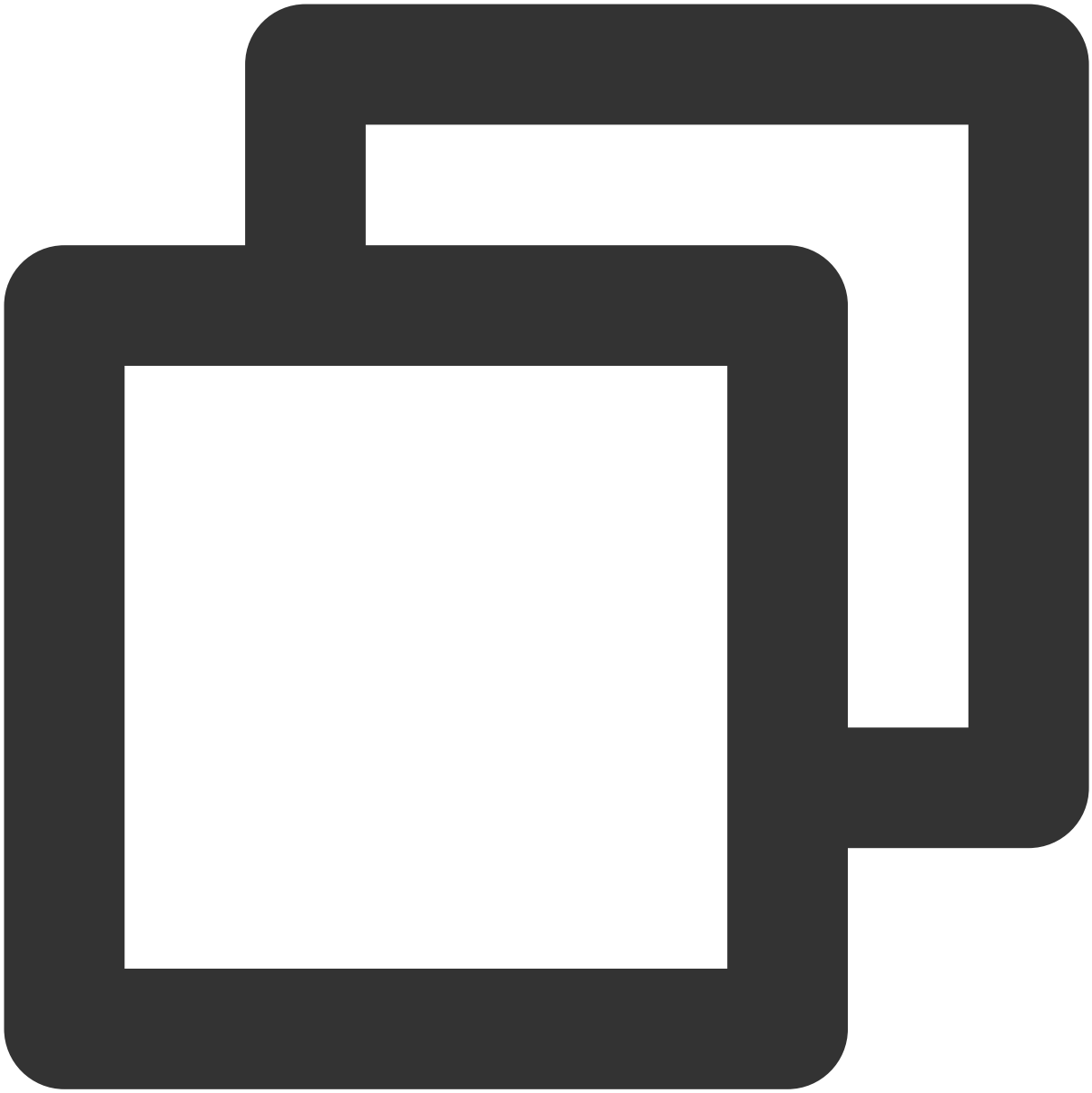
Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. Perform the following steps to deploy an exporter: Using Secret to manage Elasticsearch connection string > Deploying Elasticsearch exporter > Verifying.

Using Secret to manage Elasticsearch connection string

1. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
2. In the top-right corner of the page, click **Create via YAML** to create a YAML configuration as detailed below:

You can use Kubernetes Secrets to manage and encrypt passwords. When starting the Elasticsearch exporter, you can directly use the Secret key but need to adjust the corresponding URI. Below is a sample YAML configuration:



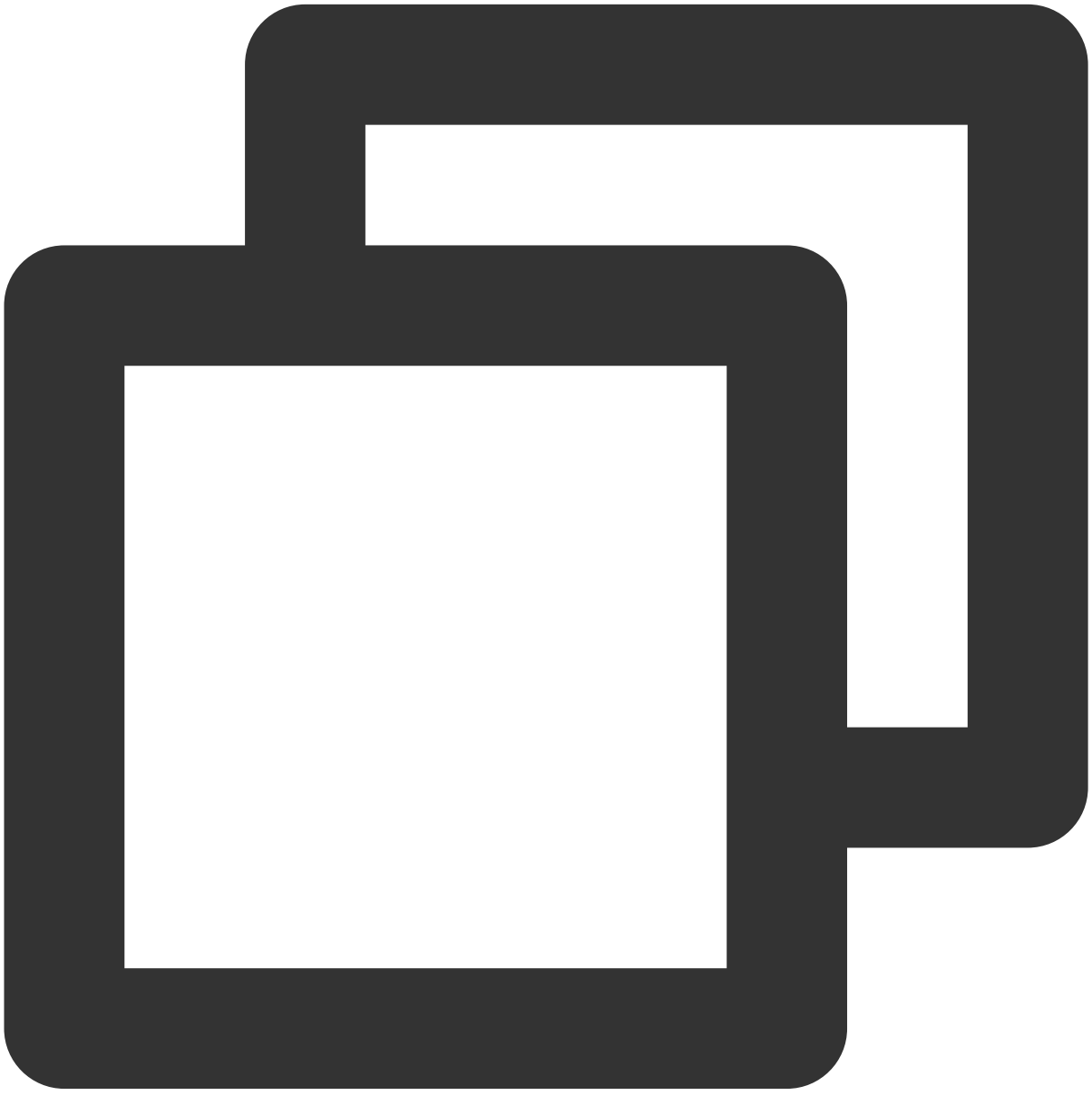
```
apiVersion: v1
kind: Secret
metadata:
  name: es-secret-test
  namespace: es-demo
type: Opaque
stringData:
  esURI: you-guess  # Corresponding Elasticsearch URI
```

Note:

The Elasticsearch connection string is in the format of `<proto>://<user>:<password>@<host>:<port>`, such as `http://admin:pass@localhost:9200`.

Deploying Elasticsearch exporter

On the Deployment management page, click **Create** and select the target **namespace** to deploy the service. You can create in the console. Here, YAML is used to deploy the exporter. Below is a sample YAML configuration:



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

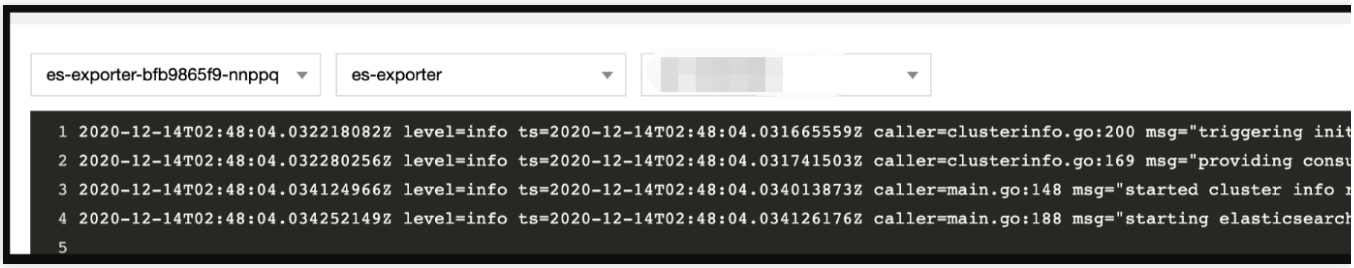
```
    labels:
      k8s-app: es-exporter
    name: es-exporter
    namespace: es-demo
  spec:
    replicas: 1
    selector:
      matchLabels:
        k8s-app: es-exporter
    template:
      metadata:
        labels:
          k8s-app: es-exporter
      spec:
        containers:
        - env:
            - name: ES_URI
              valueFrom:
                secretKeyRef:
                  name: es-secret-test
                  key: esURI
            - name: ES_ALL
              value: "true"
          image: bitnami/elasticsearch-exporter:latest
          imagePullPolicy: IfNotPresent
          name: es-exporter
          ports:
          - containerPort: 9114
            name: metric-port
          securityContext:
            privileged: false
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
        dnsPolicy: ClusterFirst
        imagePullSecrets:
        - name: qcloudregistrykey
        restartPolicy: Always
        schedulerName: default-scheduler
        securityContext: {}
        terminationGracePeriodSeconds: 30
```

Note:

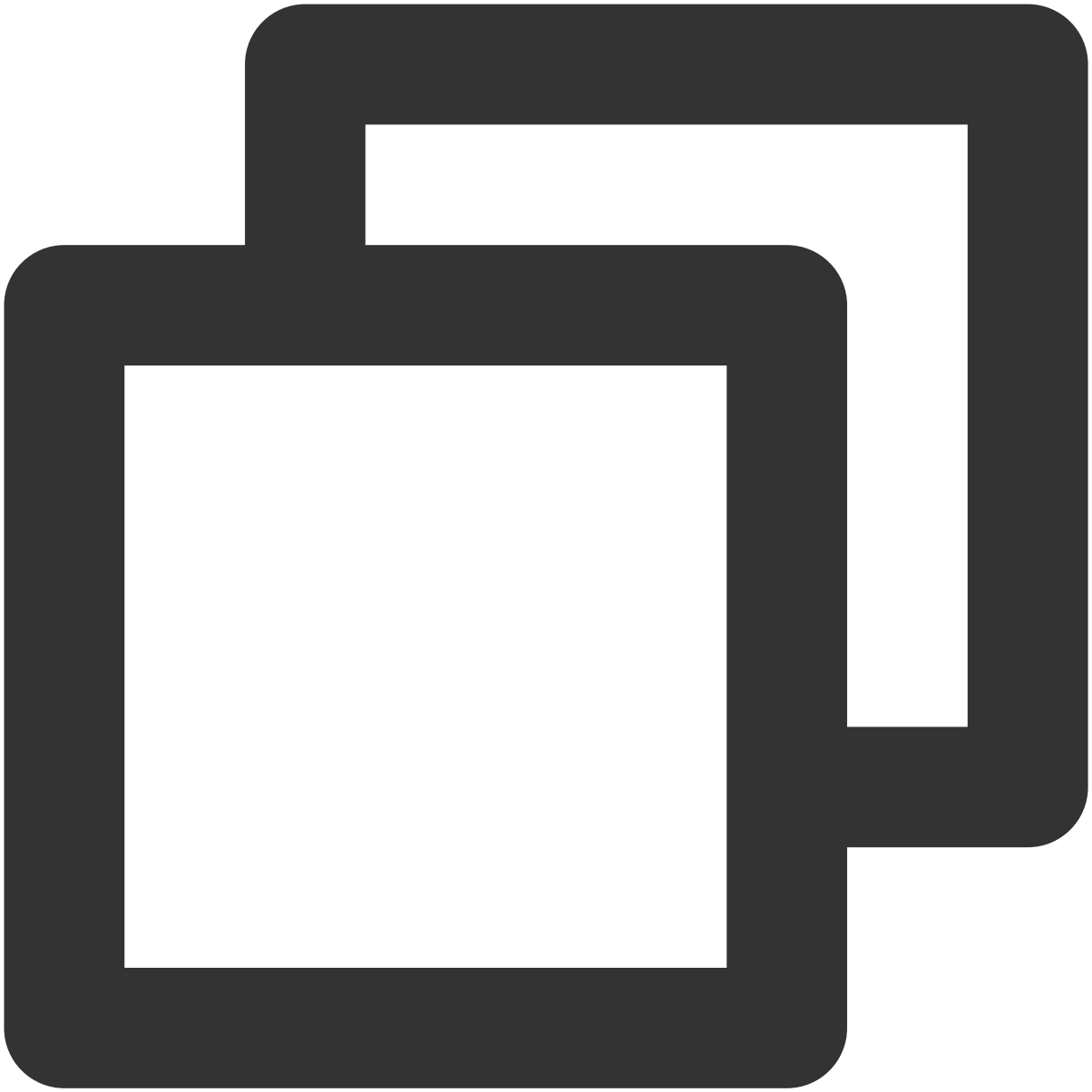
The above sample uses `ES_ALL` to collect all monitoring metrics of Elasticsearch, which can be adjusted through the corresponding parameters. For detailed exporter parameters, please see elasticsearch_exporter.

Verifying

1. Click the newly created Deployment on the **Deployment** page to enter the Deployment management page.
2. Click the **Log** tab, and you can see that the exporter is successfully started and its address is exposed as shown below:



3. Click the **Pod Management** tab to enter the Pod page.
4. In the **Operations** column on the right, click **Remote Login** to log in to the Pod. Run the following `curl` command with the address exposed by the exporter in the command line window, and you can get the corresponding Elasticsearch metrics normally. If no corresponding data is returned, please check whether the **connection string** is correct as shown below:



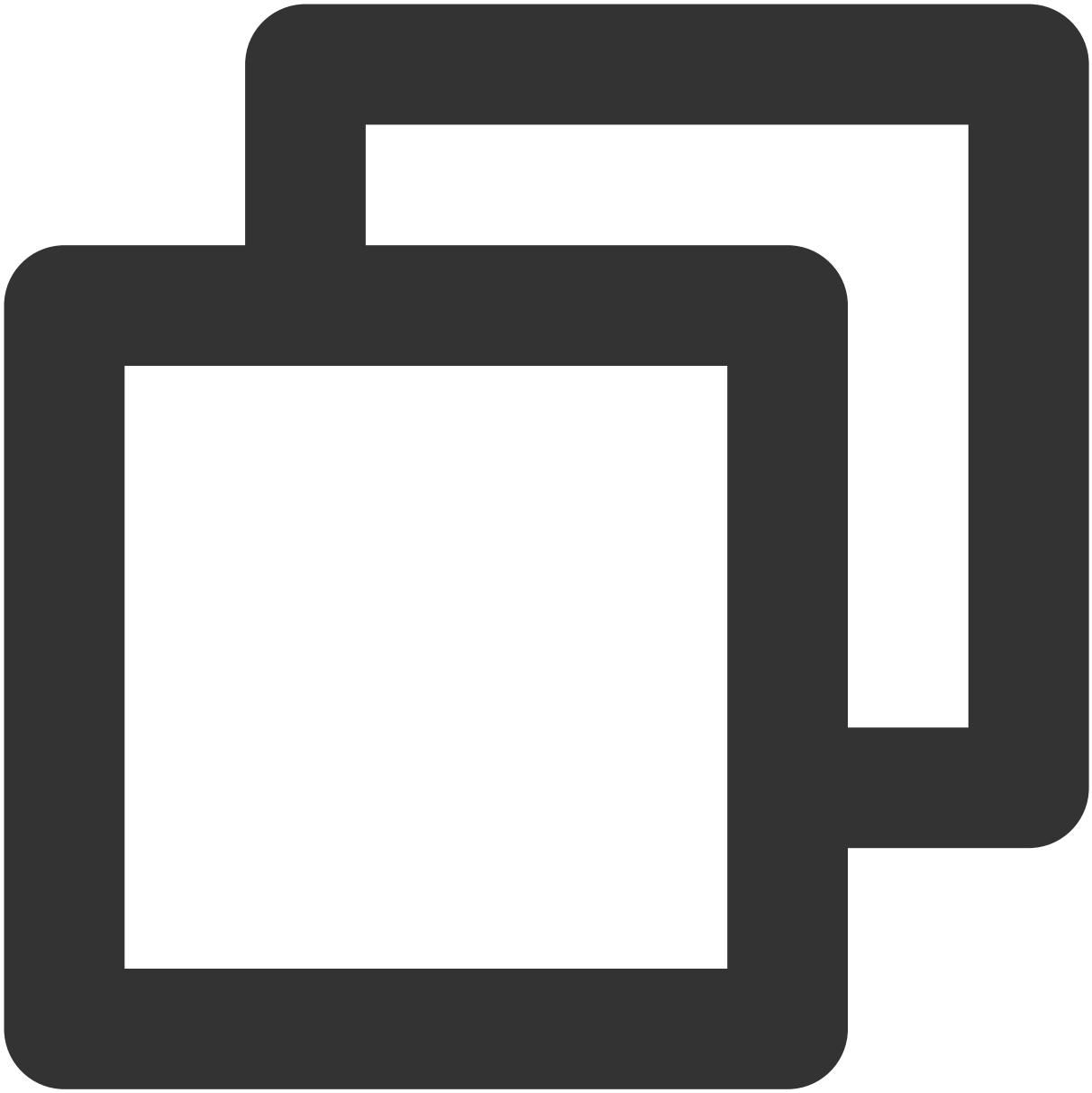
```
curl localhost:9114/metrics
```

The execution result is as shown below:

```
# HELP elasticsearch_breakers_estimated_size_bytes Estimated size i
# TYPE elasticsearch_breakers_estimated_size_bytes gauge
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",cl
2.0102643e+07
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",cl
1.9926654e+07
elasticsearch_breakers_estimated_size_bytes{breaker="accounting",cl
1.9685163e+07
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clu
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clu
elasticsearch_breakers_estimated_size_bytes{breaker="fielddata",clu
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_requ
0
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_requ
1167
elasticsearch_breakers_estimated_size_bytes{breaker="in_flight_requ
1167
elasticsearch_breakers_estimated_size_bytes{breaker="parent",cluste
2.0102643e+07
```

Adding scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:

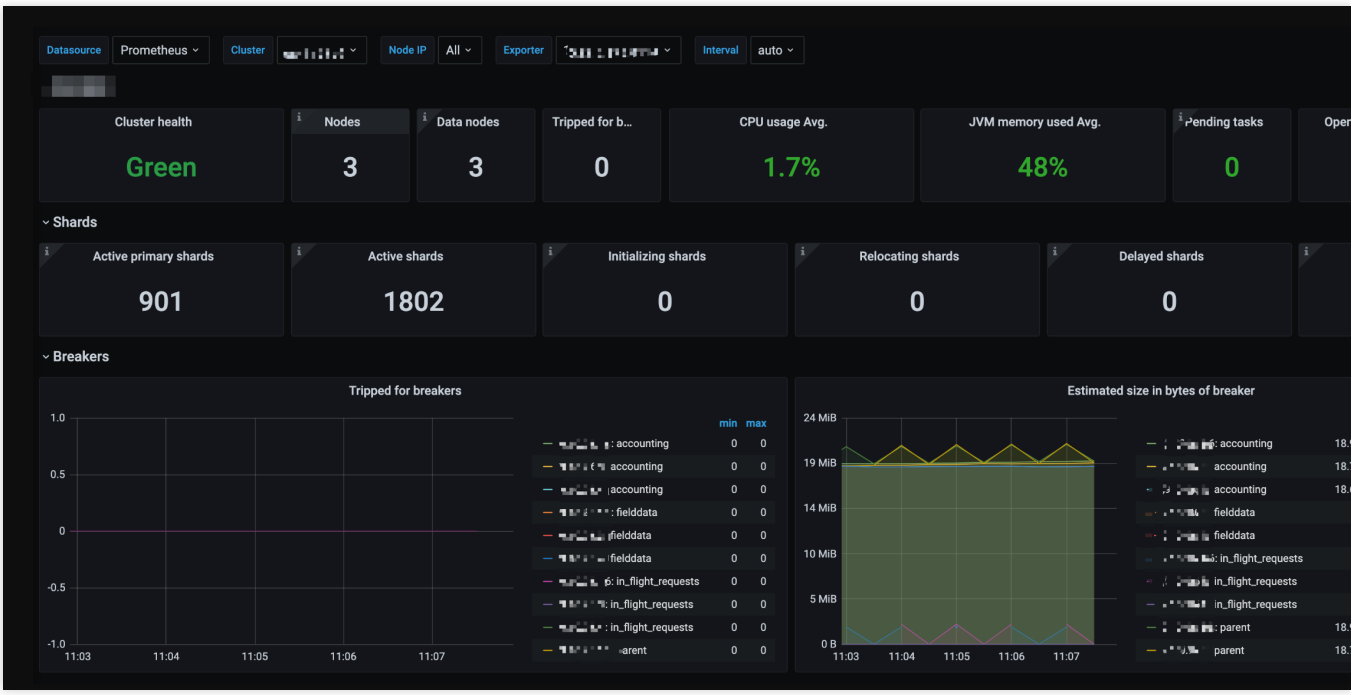


```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: es-exporter
  namespace: cm-prometheus
spec:
  namespaceSelector:
    matchNames:
      - es-demo
  podMetricsEndpoints:
  - interval: 30s
```

```
    path: /metrics
    port: metric-port
    selector:
    matchLabels:
    k8s-app: es-exporter
```

Viewing monitoring information

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Integration Center** to enter the **Integration Center** page. Find Elasticsearch monitoring, install the corresponding Grafana dashboard, and then you can enable the Elasticsearch monitoring dashboard to view instance monitoring data as shown below:



Integrating with alert feature

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Alerting Rule** and add the corresponding alerting rules. For more information, please see Creating Alerting Rule.

Kafka Exporter Integration

Last updated：2024-08-07 22:01:27

Overview

When using Kafka, you need to monitor its running status, such as cluster status and message heap. TMP provides an exporter to monitor Kafka and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to deploy the Kafka exporter and integrate it with the alert feature.

Note:

For easier export installation and management, we recommend you use TKE for unified management.

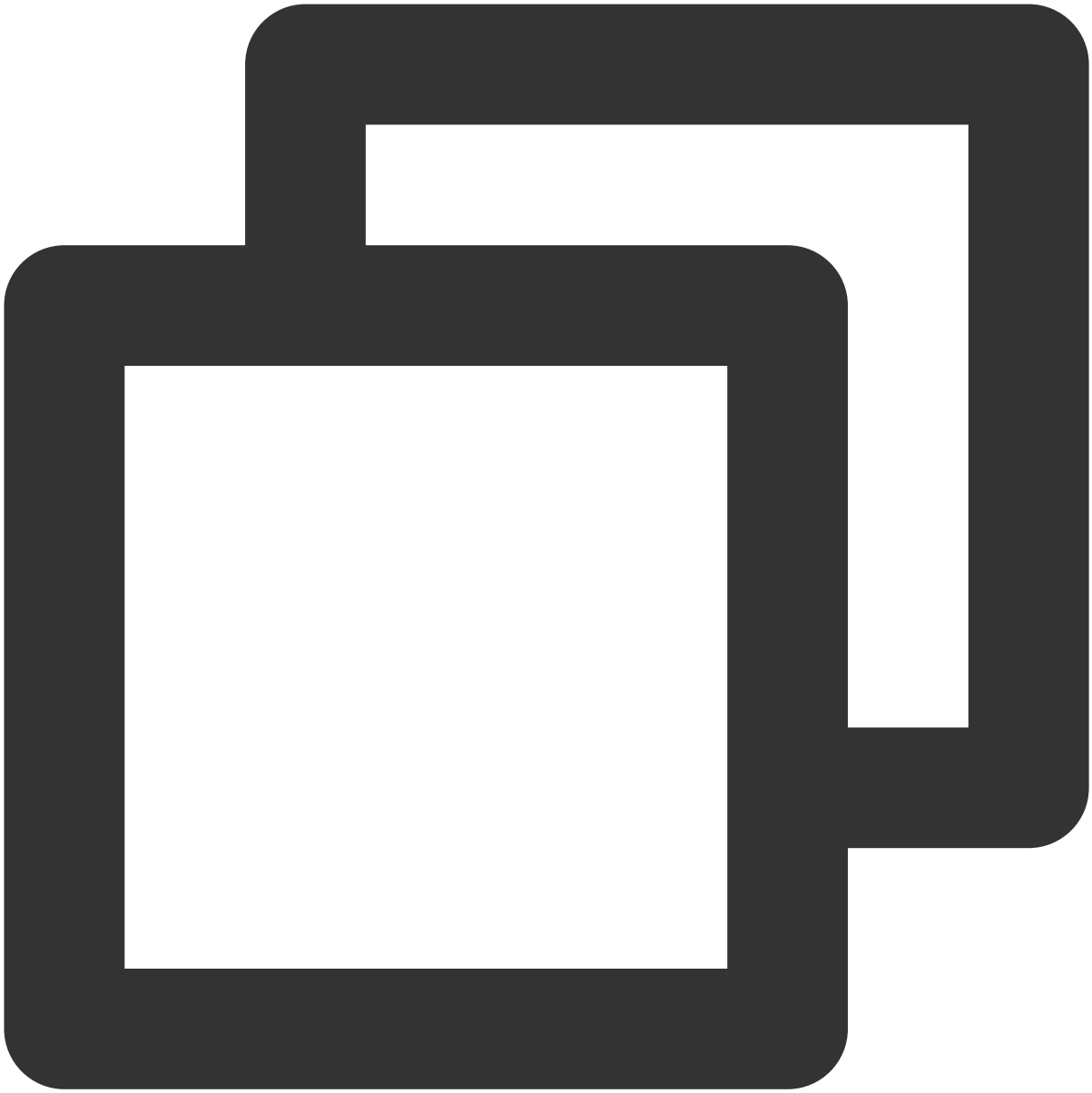
Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance and created a namespace for the cluster. You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

Directions

Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
4. On the Deployment management page, click **Create** and select the target **namespace** to deploy the service. You can create in the console. Here, YAML is used to deploy the exporter. Below is a sample YAML configuration:



```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: kafka-exporter # Rename the exporter based on the business needs. We r
  name: kafak-exporter # Rename the exporter based on the business needs. We recomm
  namespace: kafka-demo
spec:
  replicas: 1
  selector:
    matchLabels:
```

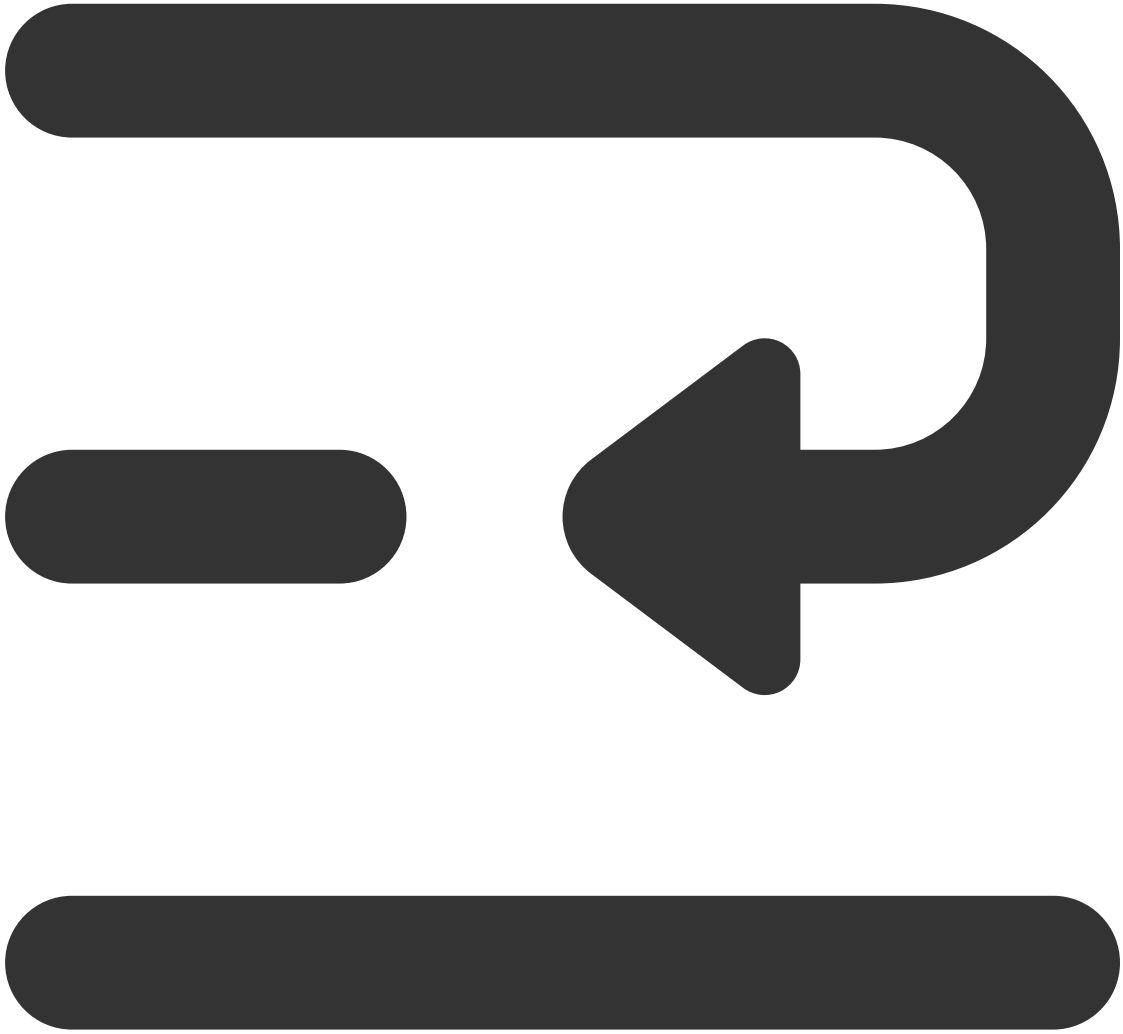
```
      k8s-app: kafka-exporter # Rename the exporter based on the business needs. We
  template:
    metadata:
      labels:
        k8s-app: kafka-exporter # Rename the exporter based on the business needs.
    spec:
      containers:
      - args:
        - --kafka.server=x.x.x.x:9092 # Corresponding Kafka instance address inform
        image: danielqsj/kafka-exporter:latest
        imagePullPolicy: IfNotPresent
        name: kafka-exporter
        ports:
        - containerPort: 9121
          name: metric-port  # This name is required during scrape task configurati
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

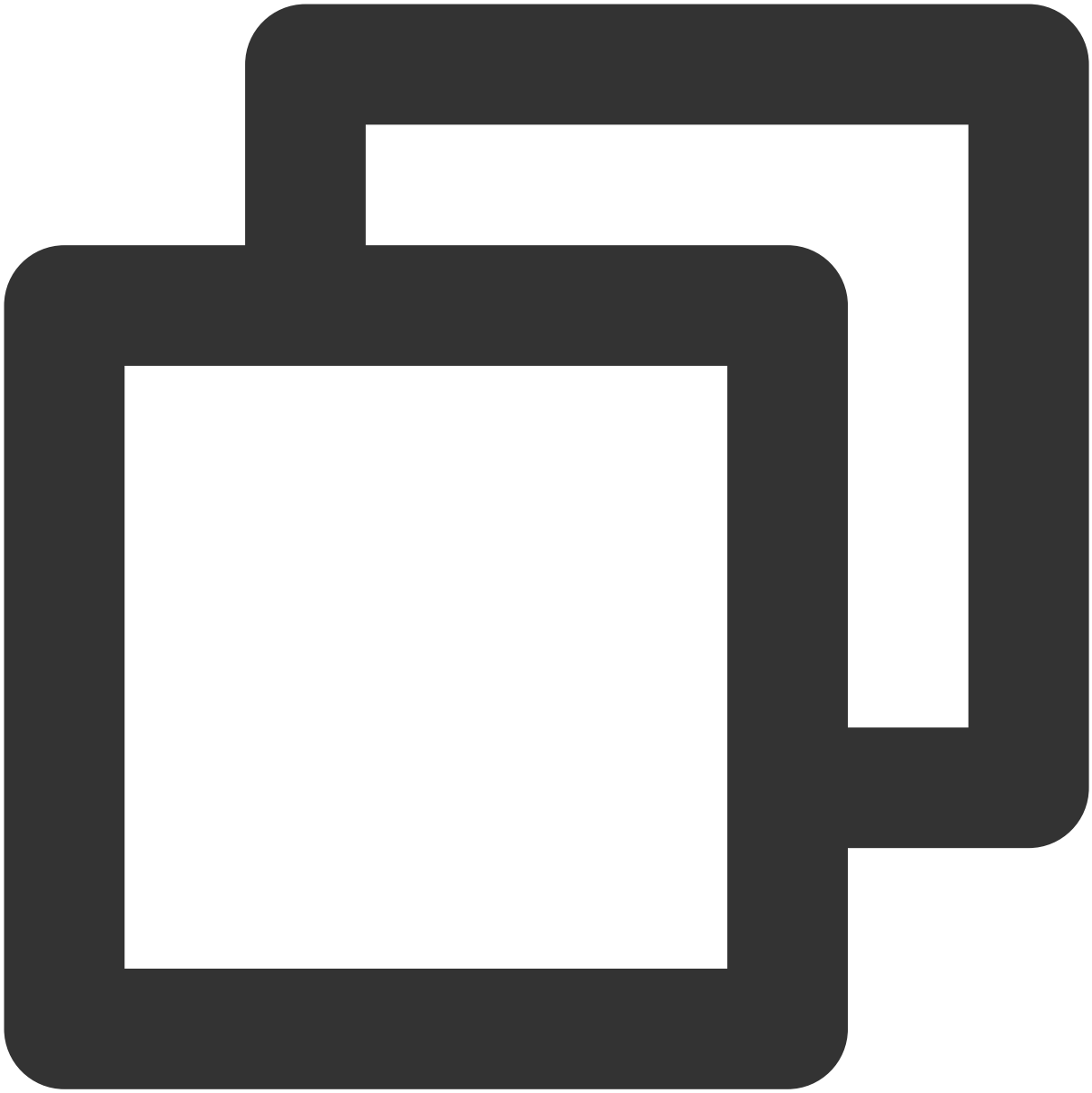
Note:

For detailed exporter parameters, please see kafka_exporter.

Adding scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:





```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: kafka-exporter  # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port # Enter the name of the corresponding port of the Prometheu
    path: /metrics # Enter the value of the corresponding path of the Prometheus
    relabelings:
```

```
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: instance
        replacement: 'ckafka-xxxxxx' # Change it to the corresponding Kafka instanc
      - action: replace
        sourceLabels:
        - instance
        regex: (.*)
        targetLabel: ip
        replacement: '1.x.x.x' # Change it to the corresponding Kafka instance IP
    namespaceSelector:
      matchNames:
      - kafka-demo
    selector:  # Enter the label value of the Pod to be monitored to locate the tar
      matchLabels:
        k8s-app: kafka-exporter
```

Note:

As the exporter and Kafka are deployed on different servers, we recommend you use the Prometheus relabeling mechanism to add the Kafka instance information to the monitoring metrics so as to locate problems more easily.

Viewing monitoring information

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Integration Center** to enter the **Integration Center** page. Find Kafka monitoring, install the corresponding Grafana dashboard, and then you can enable the Kafka monitoring dashboard to view instance monitoring data as shown below:

Integrating with alert feature

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Alerting Rule** and add the corresponding alerting rules. For more information, please see Creating Alerting Rule.

MongoDB Exporter Integration

Last updated：2024-08-07 22:01:35

Overview

When using MongoDB, you need to monitor its running status to know whether it runs normally and troubleshoot its faults. TMP provides an exporter to monitor MongoDB and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to deploy the MongoDB exporter and integrate it with the alert feature.

Note:

For easier export installation and management, we recommend you use TKE for unified management.

Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance.

You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

Directions

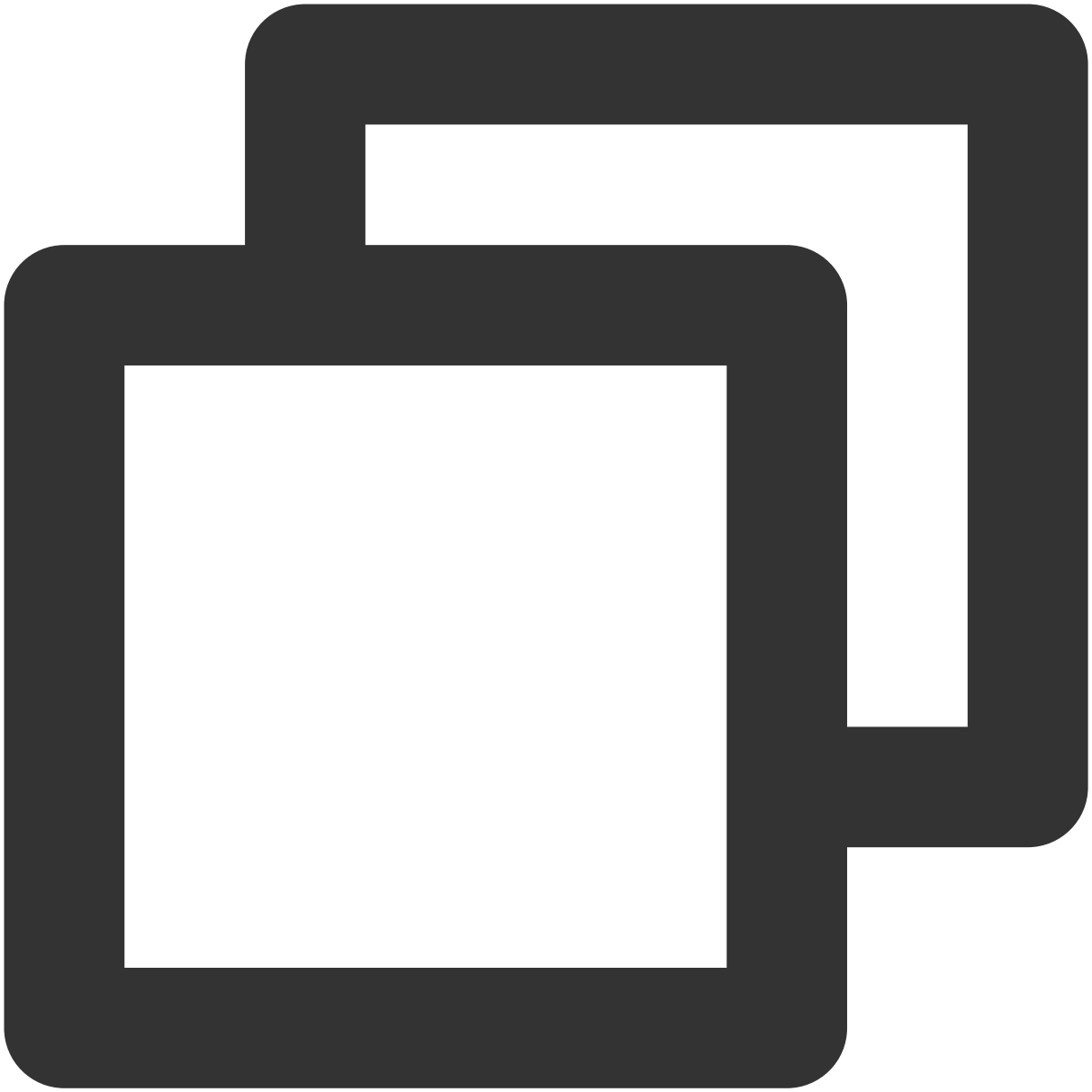
Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. Perform the following steps to deploy an exporter: Using Secret to manage MongoDB connection string > Deploying MongoDB exporter > Verifying.

Using Secret to manage MongoDB connection string

1. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
2. In the top-right corner of the page, click **Create via YAML** to create a YAML configuration as detailed below:

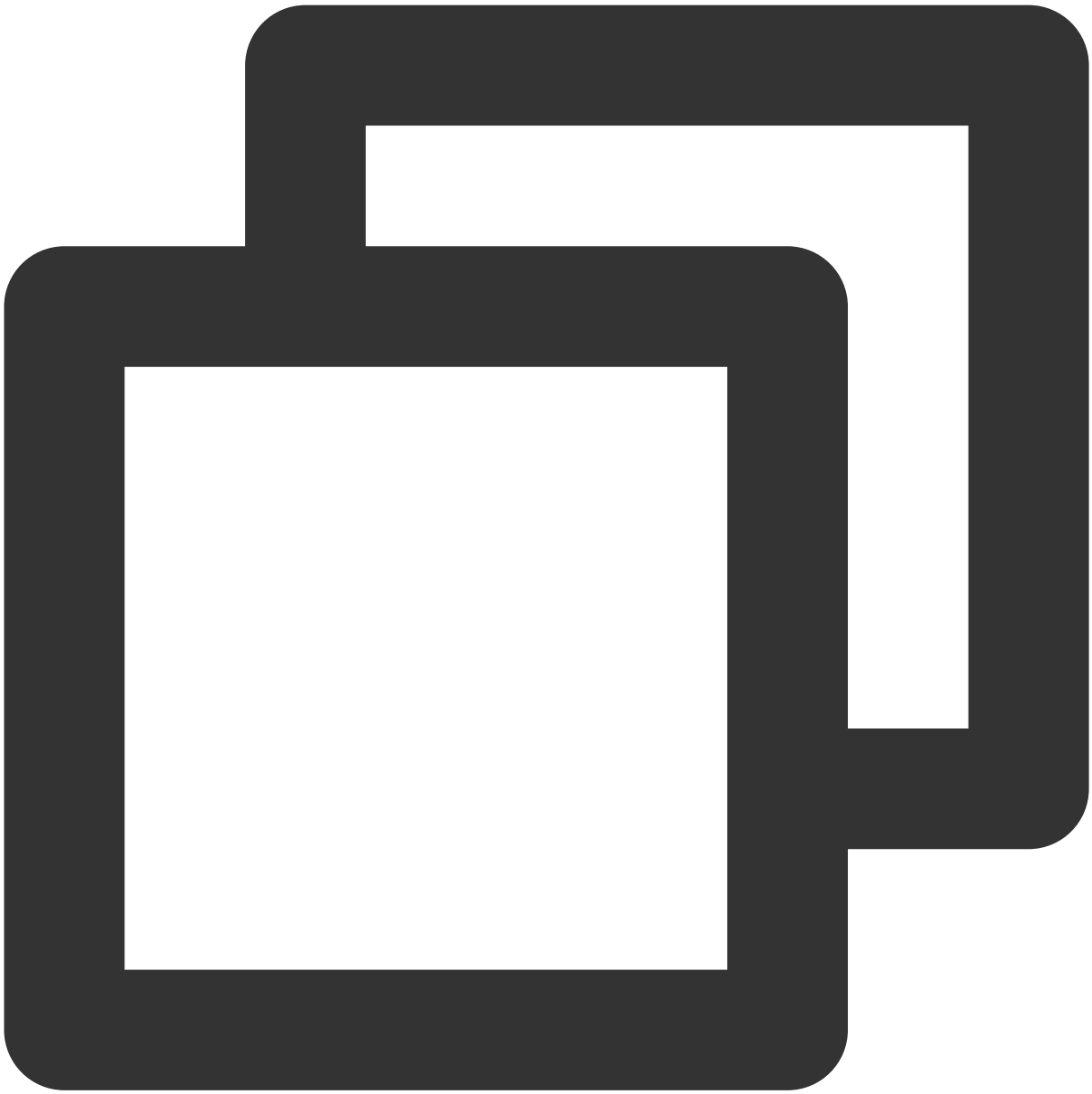
You can use Kubernetes Secrets to manage and encrypt passwords. When starting the MongoDB exporter, you can directly use the Secret key but need to adjust the corresponding URI. Below is a sample YAML configuration:



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

Deploying MongoDB exporter

On the Deployment management page, click **Create** and select the target **namespace** to deploy the service. You can create in the console. Here, YAML is used to deploy the exporter. Below is a sample YAML configuration:



```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mongodb-exporter # Rename the exporter based on the business needs. We
  name: mongodb-exporter # Rename the exporter based on the business needs. We reco
  namespace: mongodb-test
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mongodb-exporter # Rename the exporter based on the business needs.
  template:
    metadata:
      labels:
        k8s-app: mongodb-exporter # Rename the exporter based on the business needs
    spec:
      containers:
        - args:
            - --collect.database       # Enable the collection of `Database` metric
            - --collect.collection     # Enable the collection of `Collection` metr
            - --collect.topmetrics     # Enable the collection of `table top` metri
            - --collect.indexusage     # Enable the collection of `per index usage
            - --collect.connpoolstats  # Enable the collection of `MongoDB connpool
          env:
            - name: MONGODB_URI
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret-test
                  key: datasource
          image: ssheehy/mongodb-exporter
          imagePullPolicy: IfNotPresent
          name: mongodb-exporter
          ports:
            - containerPort: 9216
              name: metric-port  # This name is required during scrape task configu
          securityContext:
            privileged: false
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
        - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: { }
      terminationGracePeriodSeconds: 30
```

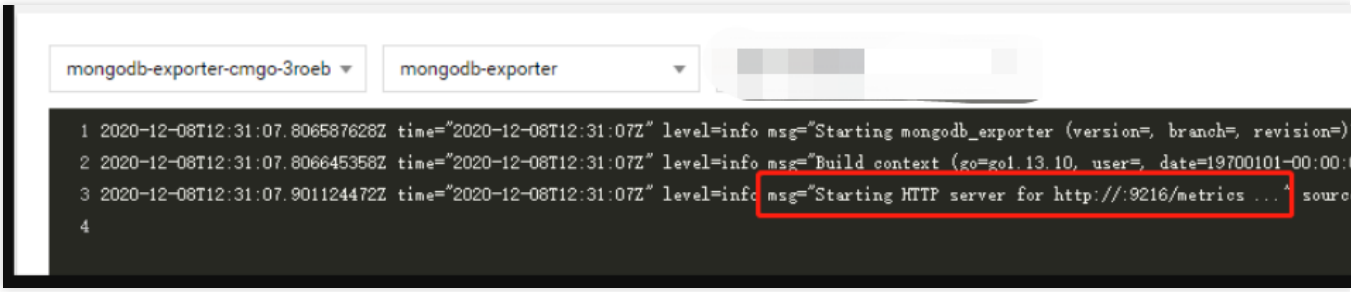
Note:

For detailed exporter parameters, please see mongodb_exporter.

Verifying

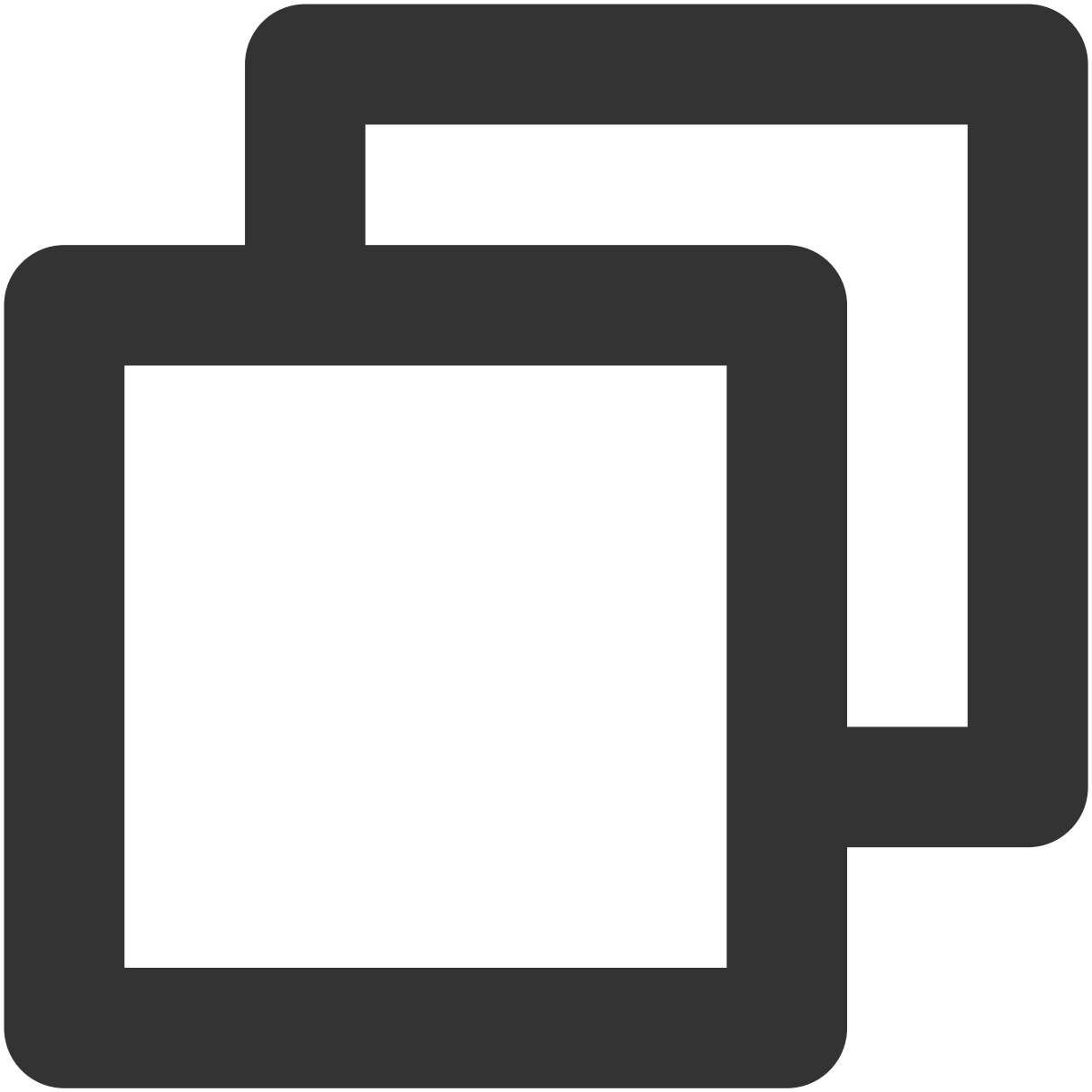
1. Click the newly created Deployment on the **Deployment** page to enter the Deployment management page.

2. Click the **Log** tab, and you can see that the exporter is successfully started and its address is exposed as shown below:



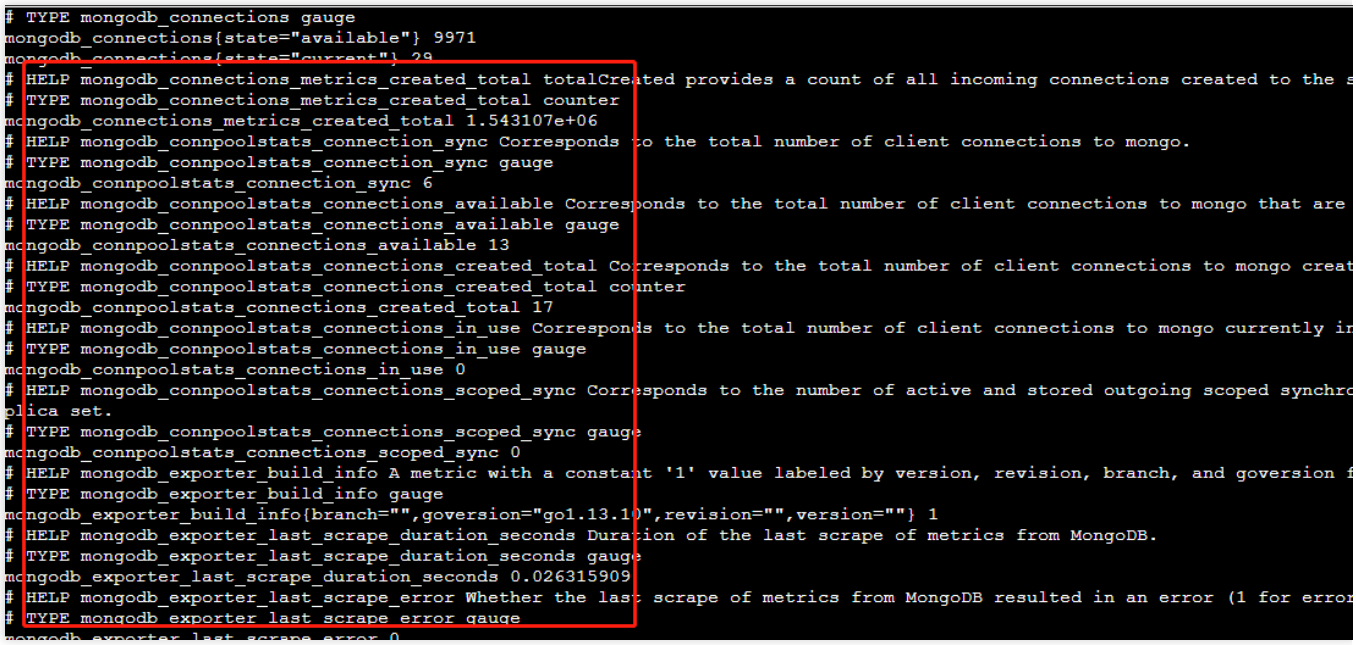
3. Click the **Pod Management** tab to enter the Pod page.

4. In the **Operations** column on the right, click **Remote Login** to log in to the Pod. Run the following `wget` command with the address exposed by the exporter on the command line, and you can get the corresponding MongoDB metrics normally. If no corresponding data is returned, please check whether the connection URI is correct as shown below:



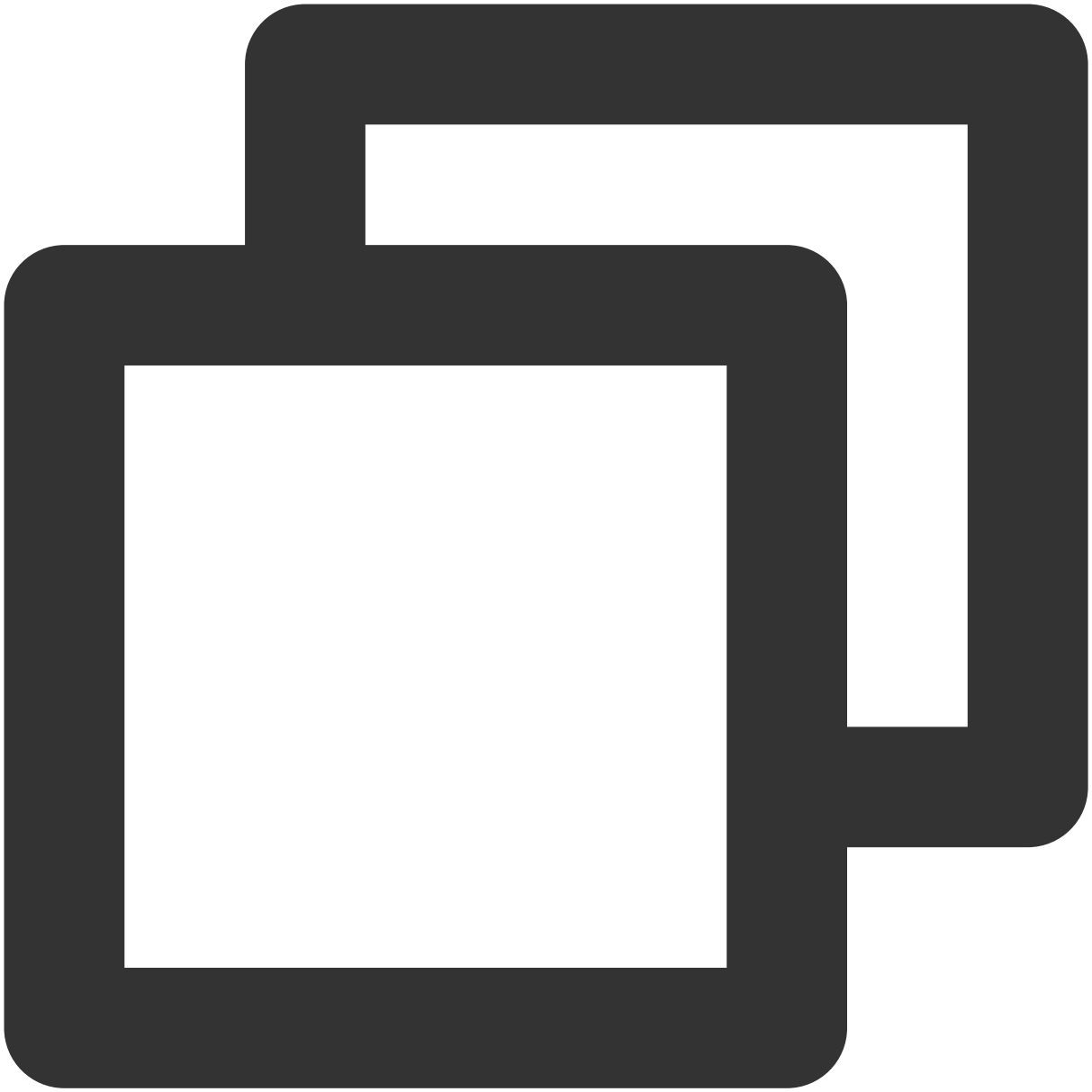
```
wget 127.0.0.1:9216/metrics
cat metrics
```

The command execution result is as shown below:



Adding scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:



```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: mongodb-exporter # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # Enter the name of the corresponding port of the Prometh
    path: /metrics  # Enter the value of the corresponding path of the Prometheus
    relabelings:
```

```
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'cmgo-xxxxxxxx' # Change it to the corresponding MongoDB insta
  namespaceSelector:  # Select the namespace where the Pod to be monitored reside
    matchNames:
    - mongodb-test
  selector: # Enter the label value of the Pod to be monitored to locate the targ
    matchLabels:
      k8s-app: mongodb-exporter
```

Note:

As the exporter and MongoDB are deployed on different servers, we recommend you use the Prometheus relabeling mechanism to add the MongoDB instance information to the monitoring metrics so as to locate problems more easily.

Viewing monitoring information

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Integration Center** to enter the **Integration Center** page. Find MongoDB monitoring, install the corresponding Grafana dashboard, and then you can enable the MongoDB monitoring dashboard to view instance monitoring data as shown below:

MongoDB Overview: you can view the status of each instance, such as number of documents, connection utilization, and read/write time. You can click an instance to view its details.

MongoDB Details: you can view the detailed status of an instance, such as metadata overview, core metrics, command operations, request traffic, and top reads/writes.

Note:

You can click **!** on the left of each chart to view the description.

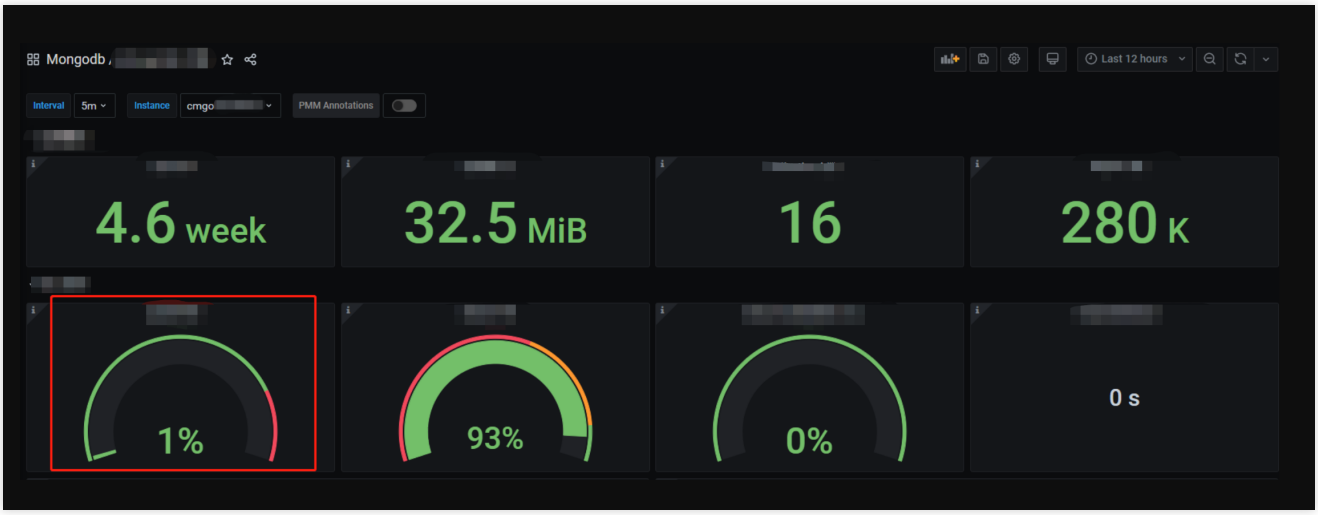
Integrating with alert feature

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Alerting Rule** and add the corresponding alerting rules. For more information, please see Creating Alerting Rule.

FAQs

The client reported an error "client checkout connect timeout". What should I do?

This is probably because that the connection pool utilization has reached 100%, resulting in a connection creation failure. You can check the **Connection Utilization** metric in **MongoDB Details > Core Metrics** on the Grafana dashboard for troubleshooting.



Write keeps timing out. What should I do?

Check whether the cache utilization is excessive and whether the number of available transactions is 0. You can check the **Available WiredTiger Transactions, WiredTiger Cache Utilization, and GetLastError Write Time** metrics in **MongoDB Details > Core Metrics** on the Grafana dashboard for troubleshooting.

PostgreSQL Exporter Integration

Last updated：2024-08-07 22:01:43

Overview

When using PostgreSQL, you need to monitor its running status to know whether it runs normally and troubleshoot its faults. TMP provides an exporter to monitor PostgreSQL and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to deploy the PostgreSQL exporter and integrate it with the alert feature.

Note:

For easier export installation and management, we recommend you use TKE for unified management.

Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance.

You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

Directions

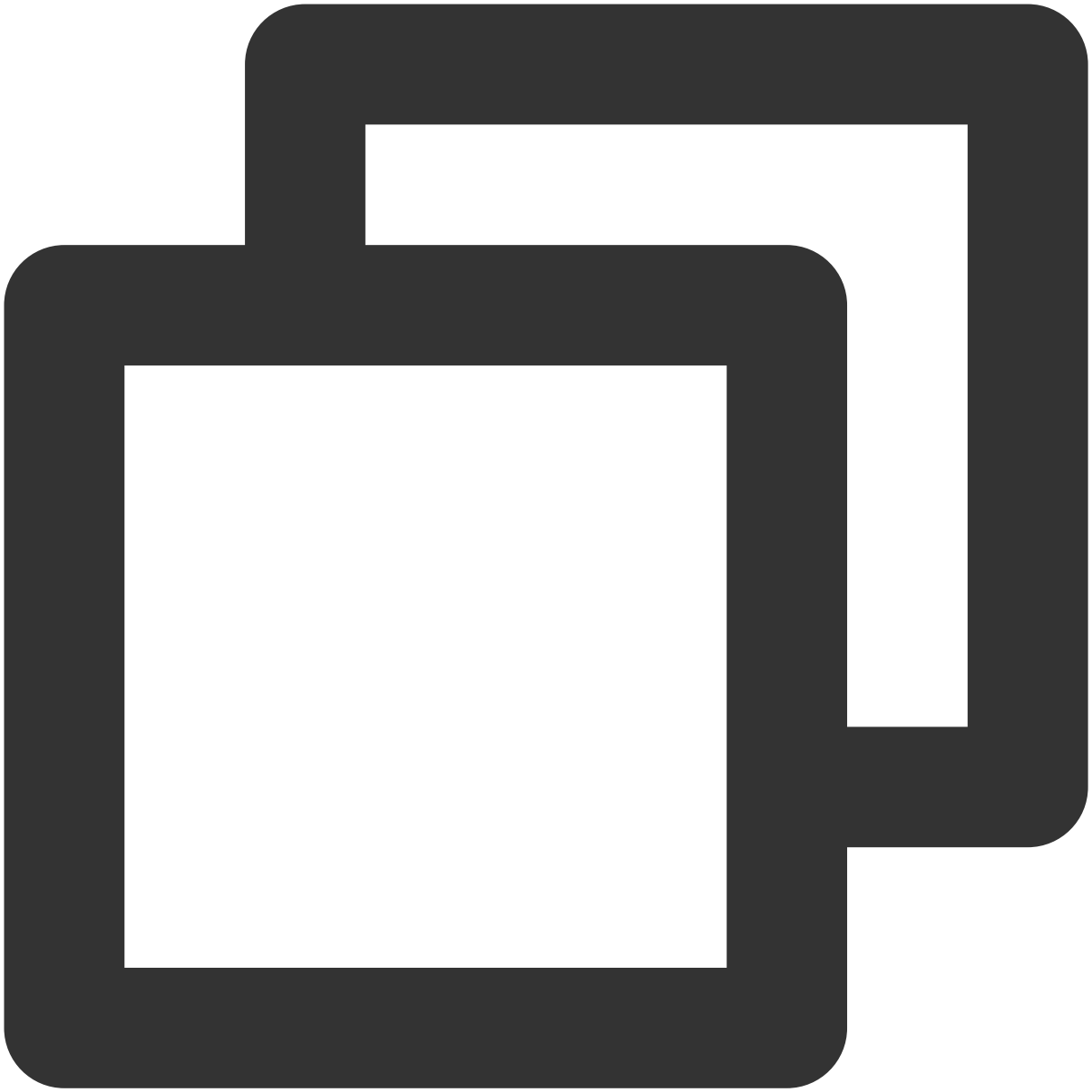
Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. Perform the following steps to deploy an exporter: Using Secret to manage PostgreSQL password > Deploying PostgreSQL exporter > Getting metric.

Using Secret to manage PostgreSQL password

1. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
2. In the top-right corner of the page, click **Create via YAML** to create a YAML configuration as detailed below:

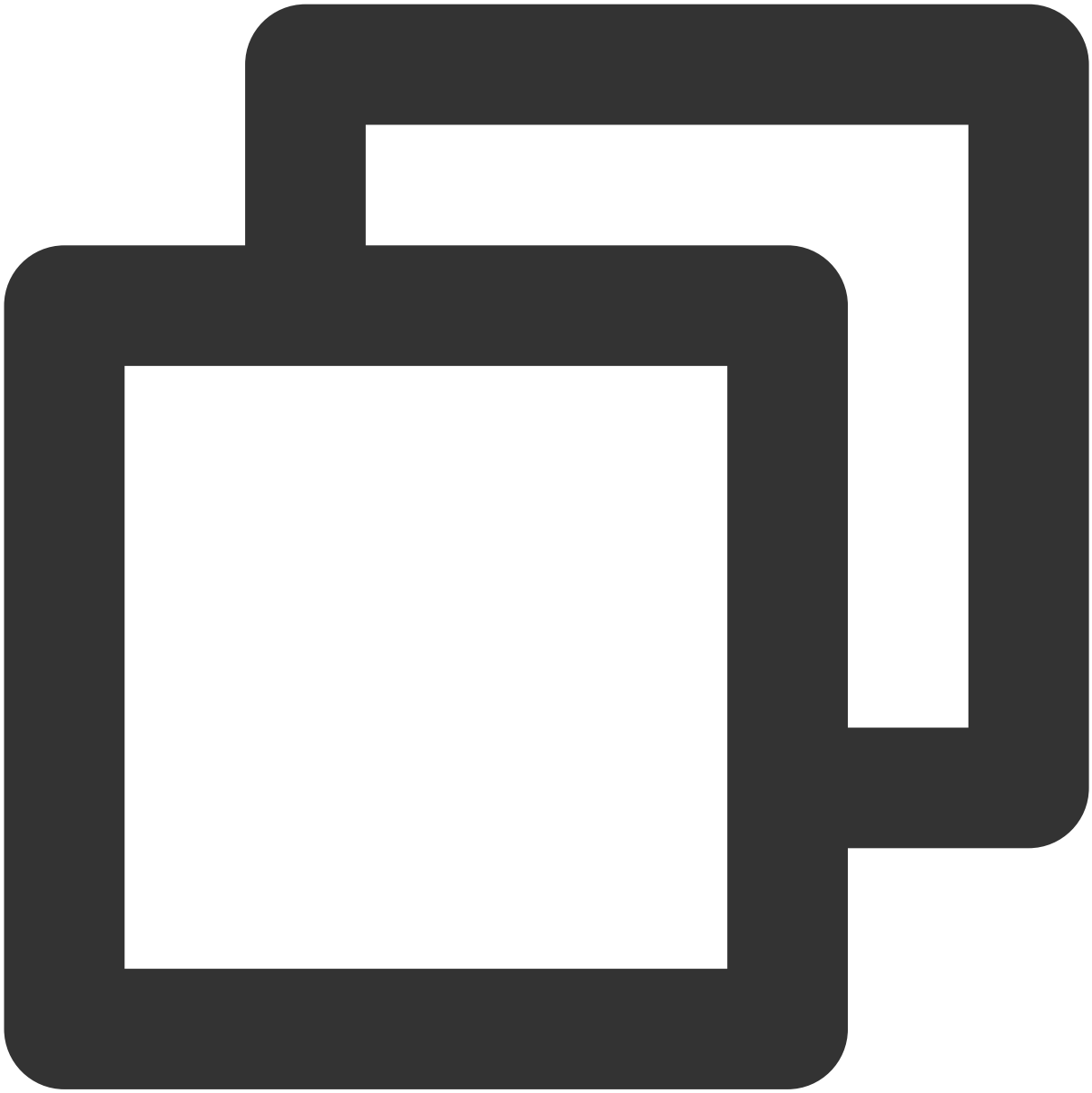
You can use Kubernetes Secrets to manage and encrypt passwords. When starting the PostgreSQL exporter, you can directly use the Secret key but need to adjust the corresponding `password` . Below is a sample YAML configuration:



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

Deploying PostgreSQL exporter

On the Deployment management page, click **Create** and select the target **namespace** to deploy the service. You can create in the console. Here, YAML is used to deploy the exporter. Below is a sample YAML configuration (please directly copy the following content and adjust the corresponding parameters based on your actual business needs):



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test
  namespace: postgres-test
  labels:
    app: postgres
```

```
    app.kubernetes.io/name: postgresql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
      app.kubernetes.io/name: postgresql
  template:
    metadata:
      labels:
        app: postgres
        app.kubernetes.io/name: postgresql
    spec:
      containers:
      - name: postgres-exporter
        image: wrouesnel/postgres_exporter:latest
        args:
          - "--web.listen-address=:9187"
          - "--log.level=debug"
        env:
          - name: DATA_SOURCE_USER
            valueFrom:
              secretKeyRef:
                name: postgres-test
                key: username
          - name: DATA_SOURCE_PASS
            valueFrom:
              secretKeyRef:
                name: postgres-test
                key: password
          - name: DATA_SOURCE_URI
            value: "x.x.x.x:5432/postgres?sslmode=disable"
        ports:
        - name: http-metrics
          containerPort: 9187
```

Note:

In the above sample, the username and password in `Secret` are passed in to the environment variables `DATA_SOURCE_USER` and `DATA_SOURCE_PASS`, so the username and password cannot be viewed in plaintext. You can also use `DATA_SOURCE_USER_FILE` / `DATA_SOURCE_PASS_FILE` to read the username and password from the file, or use `DATA_SOURCE_NAME` to put them in the connection string, such as `postgresql://login:password@hostname:port/dbname`.

Parameter description

The `query` part (after `?`) in the `DATA_SOURCE_URI` / `DATA_SOURCE_NAME` connection string supports the following parameters (the latest supported parameters listed in Connection String Parameters shall prevail):

Parameter	Description
sslmode	Whether to use SSL. Valid values:
- disable	Do not use SSL
- require	Always use (skip verification)
- verify-ca	Always use (check whether the certificate provided by the server is issued by a trusted CA)
- verify-full	Always use (check whether the certificate provided by the server is issued by a trusted CA and whether the hostname matches the certificate)
fallback_application_name	Alternative application_name
connect_timeout	Maximum connection wait time in seconds. `0` indicates to wait infinitely
sslcert	Certificate file path. The file data must be in PEM format
SSHKey	Private key file path. The file data must be in PEM format
sslrootcert	Root certificate file path. The file data must be in PEM format

Other supported exporter parameters are as detailed below (for more information, please see PostgreSQL Server Exporter):

Parameter	Description	Environment Variable
--web.listen-address	Listening address. Default value: :9487	PG_EXPORTER_WEB_LISTEN_ADDRESS
--web.telemetry-path	Path under which to expose metrics. Default value: /metrics	PG_EXPORTER_WEB_TELEMETRY_PATH
--extend.query-path	Path of a YAML file containing custom queries to run. For more information, please see queries.yaml	PG_EXPORTER_EXTEND_QUERY_PATH
--disable-default-metrics	Uses only metrics supplied from queries.yaml	PG_EXPORTER_DISABLE_DEFAULT_METRICS
--disable-settings-metrics	Skips scraping pg_settings metrics	PG_EXPORTER_DISABLE_SETTINGS_METRICS

--auto-discover-databases	Whether to discover the databases in the PostgreSQL instance dynamically	PG_EXPORTER_AUTO_DISCOVER_DATABASES
--dumpmaps	Prints the internal metric information to help troubleshoot custom queries (do not use it unless for debugging)	-
--constantLabels	Custom label provided in the format of key=value. Multiple labels are separated with ,	PG_EXPORTER_CONSTANT_LABELS
--exclude-databases	Database to be excluded. It takes effect only if --auto-discover-databases is enabled	PG_EXPORTER_EXCLUDE_DATABASES
--log.level	Log level. Valid values: debug, info, warn, error, fatal	PG_EXPORTER_LOG_LEVEL

Getting metric

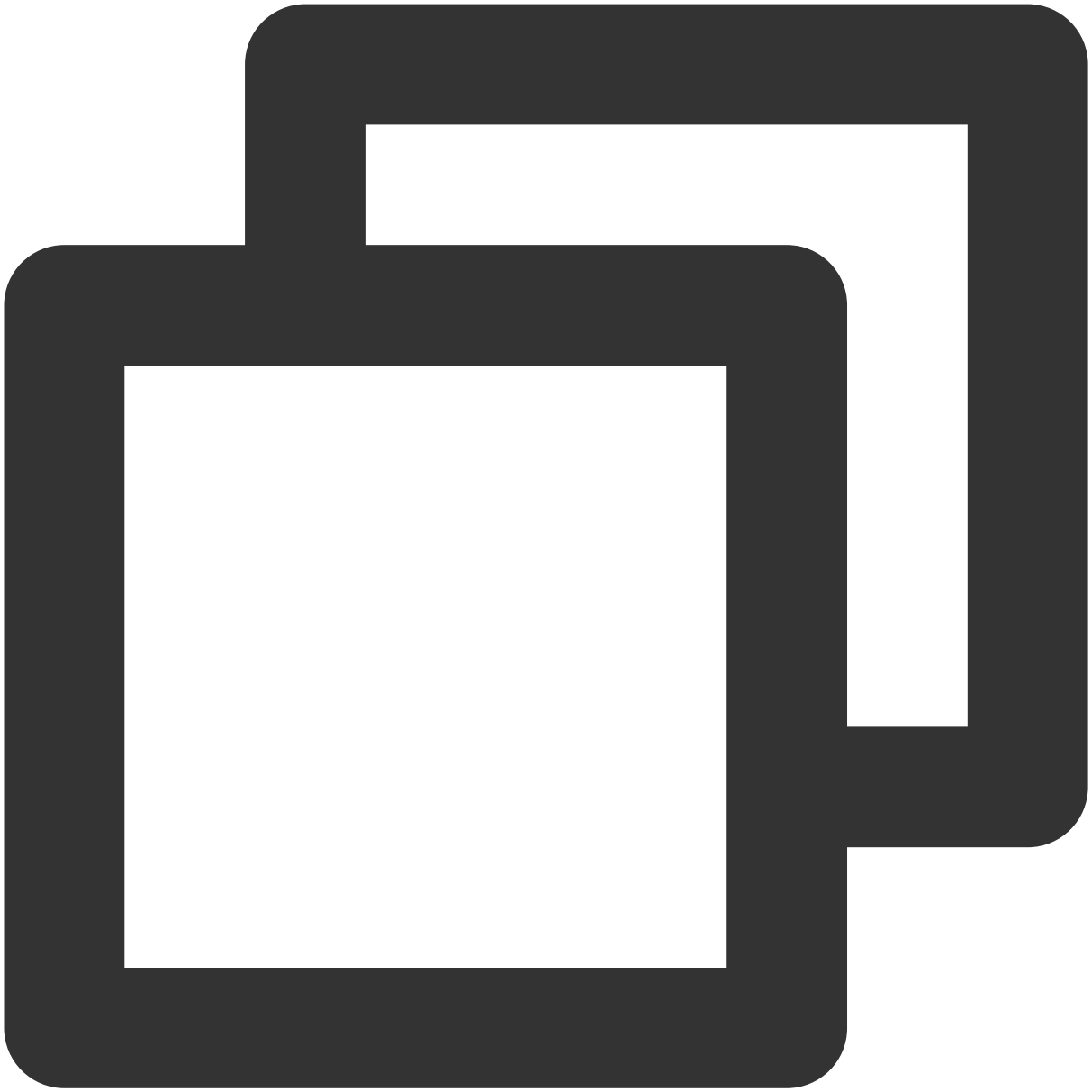
You cannot get the PostgreSQL instance operation time through `curl http://exporter:9187/metrics` .

You can define a `queries.yaml` file to get this metric:

1. Create a ConfigMap containing `queries.yaml` .

2. Mount the ConfigMap to a directory in the exporter as a volume.

3. Use the ConfigMap through `--extend.query-path` to aggregate the information of the aforementioned Secret and Deployment. The YAML file after aggregation is as shown below:

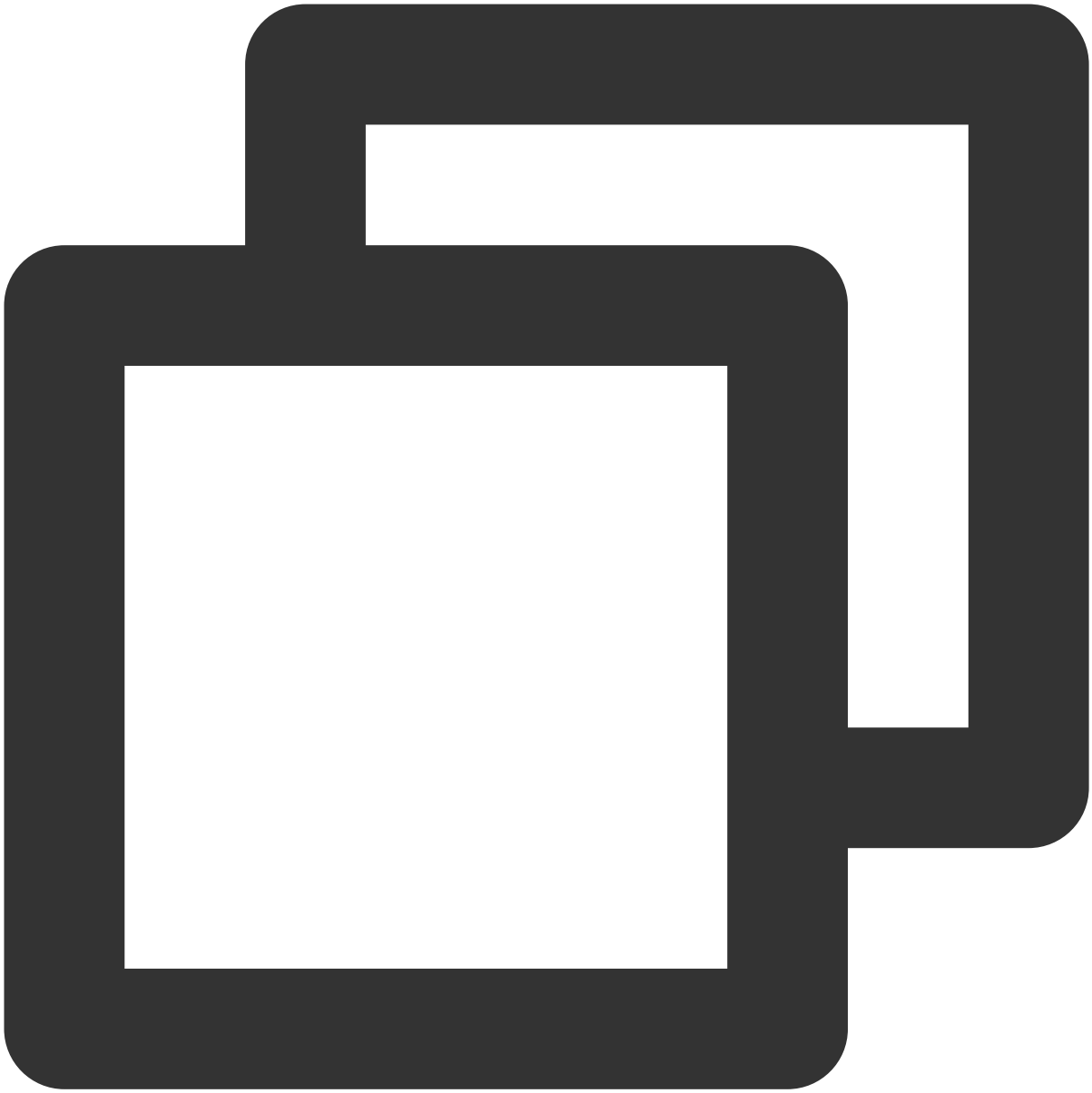


```
# Note: the following document sample code creates a namespace named `postgres-test
apiVersion: v1
kind: Namespace
metadata:
  name: postgres-test

# The following document sample code creates a Secret containing a username and pas
---
apiVersion: v1
kind: Secret
metadata:
```

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

# The following document sample code creates a `queries.yaml` file containing custo
---
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_postmas
      master: true
      metrics:
        - start_time_seconds:
            usage: "GAUGE"
            description: "Time at which postmaster started"

# The following document sample code mounts the Secret and ConfigMap and defines ex
---
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. Run `curl http://exporter:9187/metrics` , and you can use the custom `queries.yaml` to query the PostgreSQL instance start time as follows:



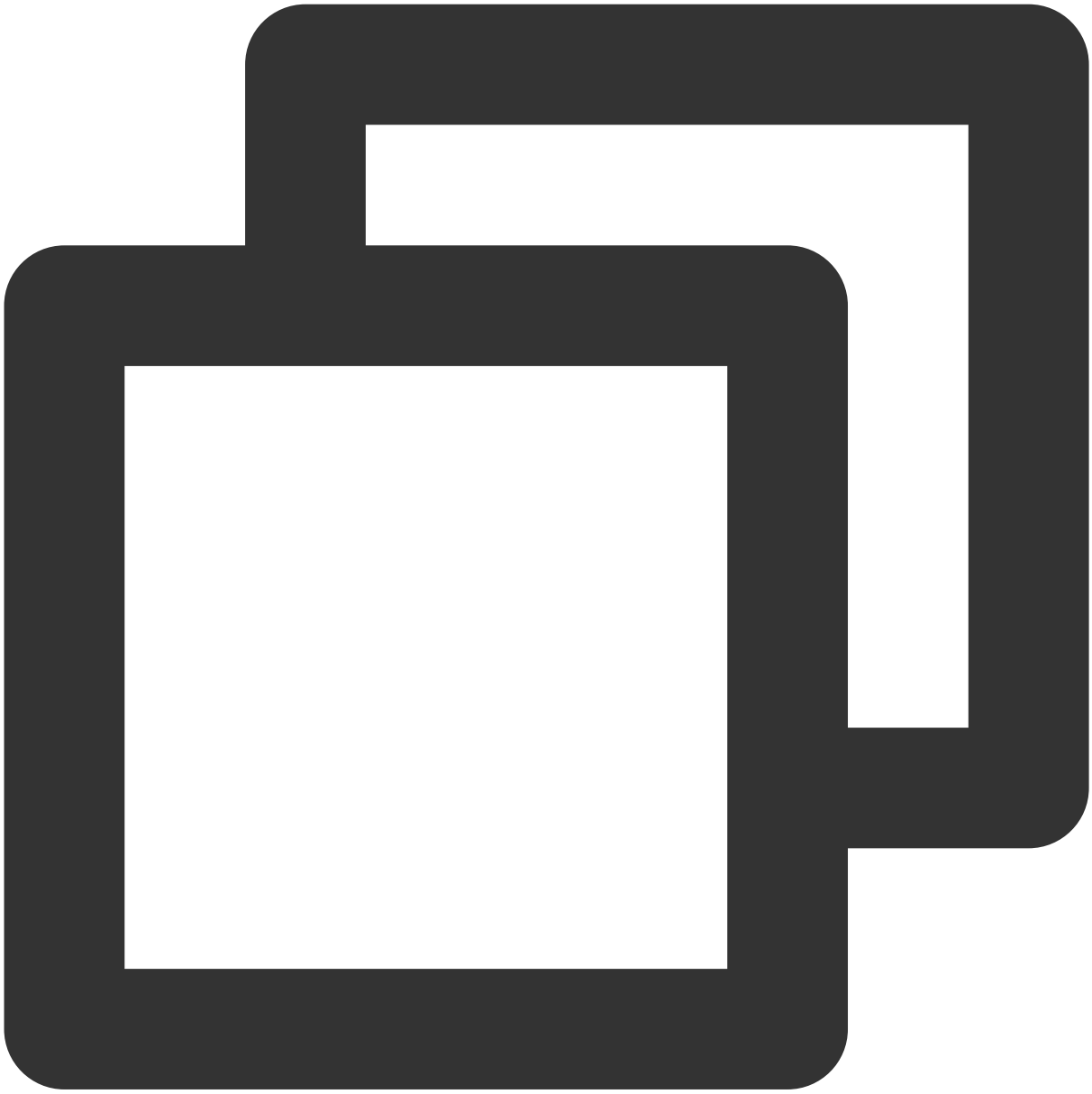
```
# HELP pg_postmaster_start_time_seconds Time at which postmaster started
# TYPE pg_postmaster_start_time_seconds gauge
pg_postmaster_start_time_seconds{server="x.x.x.x:5432"} 1.605061592e+09
```

Adding scrape task

After the exporter runs, you need to configure TMP to discover and collect the monitoring metrics in the following steps:

1. Log in to the TMP console and select the target TMP instance to enter the management page.

2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:



```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: postgres-exporter
  namespace: cm-prometheus
spec:
  namespaceSelector:
```

```
    matchNames:
    - postgres-test
  podMetricsEndpoints:
  - interval: 30s
    path: /metrics
    port: http-metrics # Port name of the aforementioned exporter container
    relabelings:
    - action: labeldrop
      regex: __meta_kubernetes_pod_label_(pod_|statefulset_|deployment_|controlle
    - action: replace
      regex: (.*)
      replacement: postgres-xxxxxx
      sourceLabels:
      - instance
      targetLabel: instance
  selector:
    matchLabels:
      app: postgres
```

Note:

For more advanced usage, please see ServiceMonitor and PodMonitor.

Visualizing Grafana dashboard

Note:

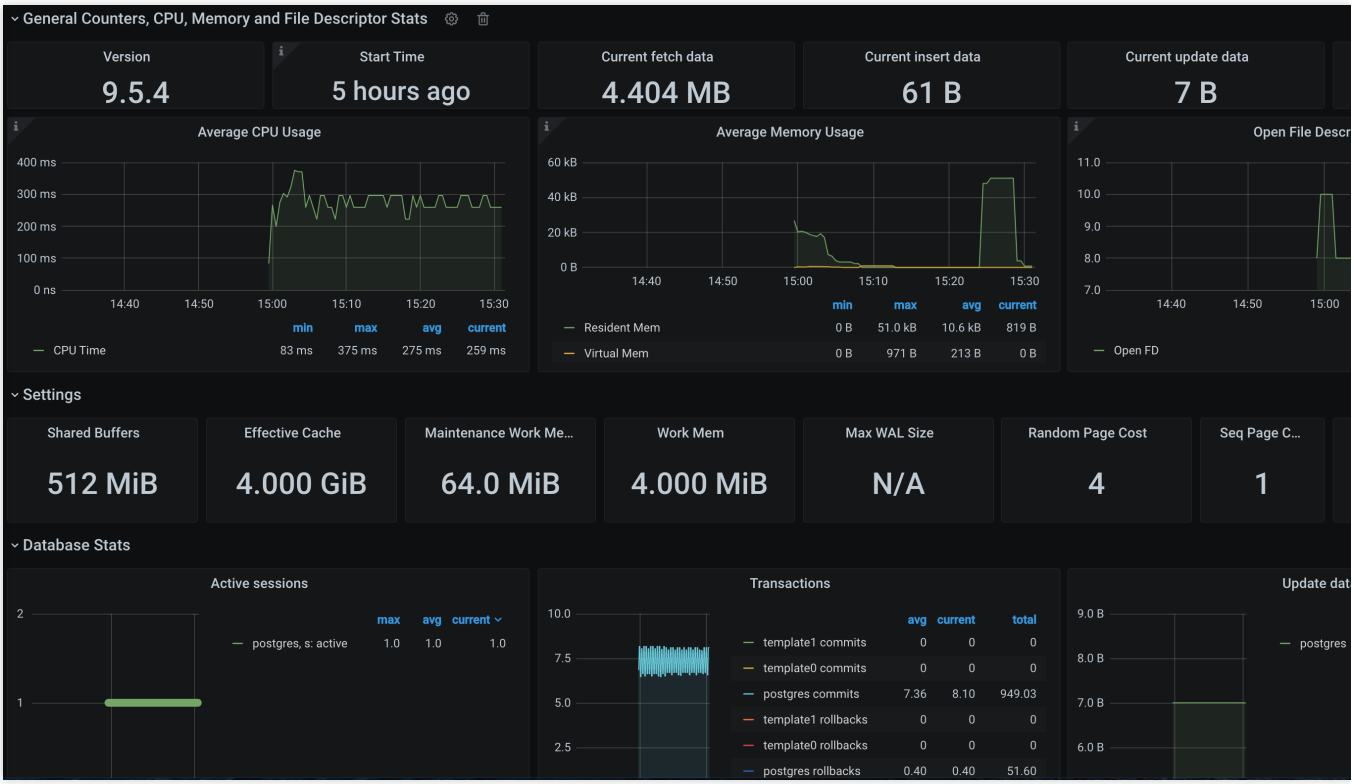
You need to use the configuration in Getting metric to get the PostgreSQL instance start time.

1. In the TMP instance list, find the corresponding TMP instance, click

on the right of the instance ID to open your Grafana page, and enter your account and password to access the Grafana visual dashboard operation section.

2. Enter Grafana, click the

icon to expand the monitoring dashboard, and click the name of the corresponding monitoring chart to view the monitoring data.



Integrating with alert feature

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Alerting Rule** and add the corresponding alerting rules. For more information, please see Creating Alerting Rule.

Note:

TMP will provide more PostgreSQL alerting templates in the near future.

NGINX Exporter Integration

Last updated：2024-08-07 22:01:48

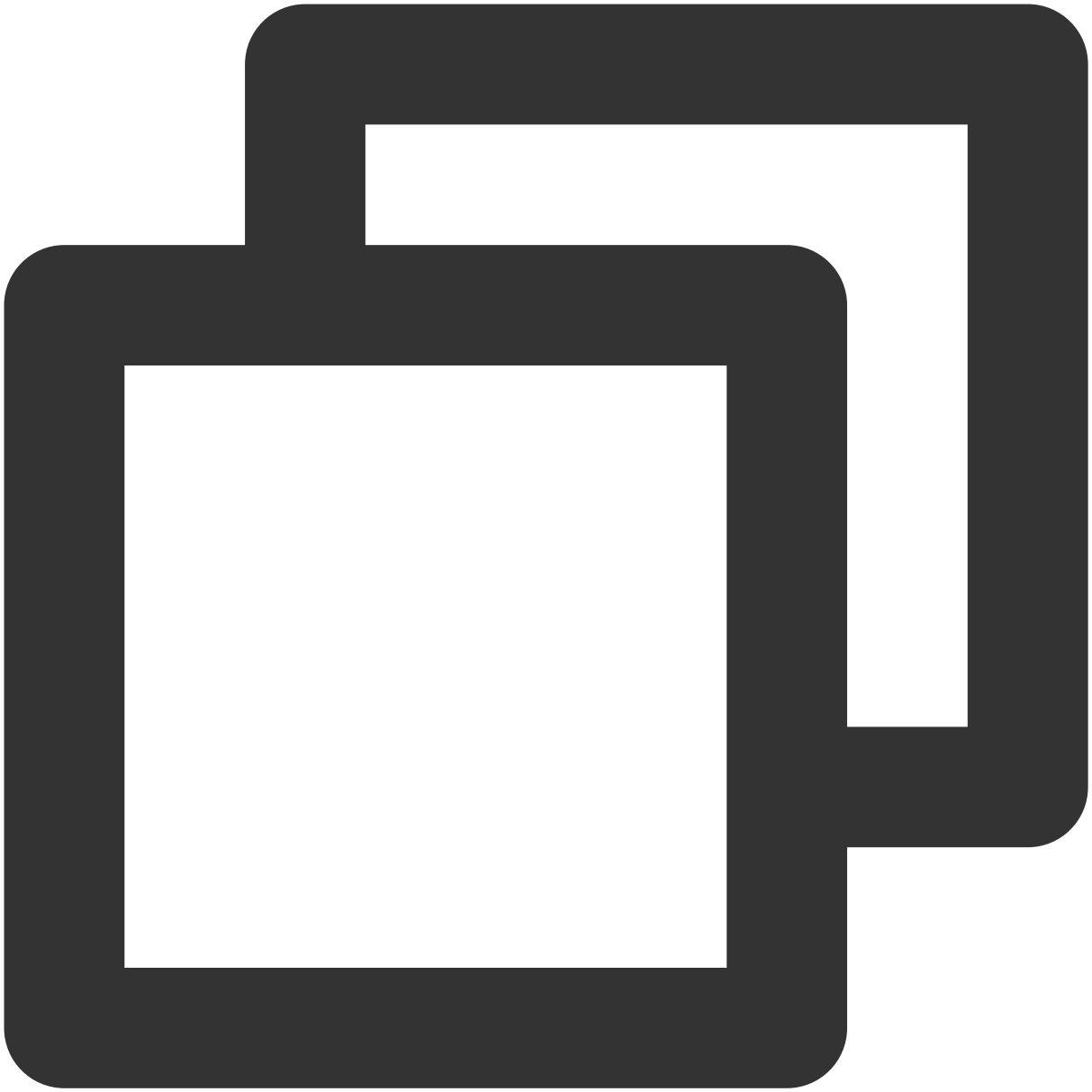
Overview

NGINX exposes certain monitoring metrics through the `stub_status` page. NGINX Prometheus Exporter collects the metrics of a single NGINX instance, converts them into monitoring data that can be used by Prometheus, and exposes the data to the Prometheus service for collection over the HTTP protocol. You can use the exporter to report the key monitoring metrics, which can be used for exception alerting and displayed on the dashboard.

Directions

Using Docker container to run exporter

Method 1. Use nginx-prometheus-exporter to quickly deploy the exporter in a Docker container. Run the following Docker command:



```
$ docker run -p 9113:9113 nginx/nginx-prometheus-exporter:0.8.0 -nginx.scrape-uri h
```

Method 2. Use the nginx-prometheus-exporter image to deploy the service in TKE and collect the monitoring data through TMP's self-discovery CRD PodMonitor or ServiceMonitor.

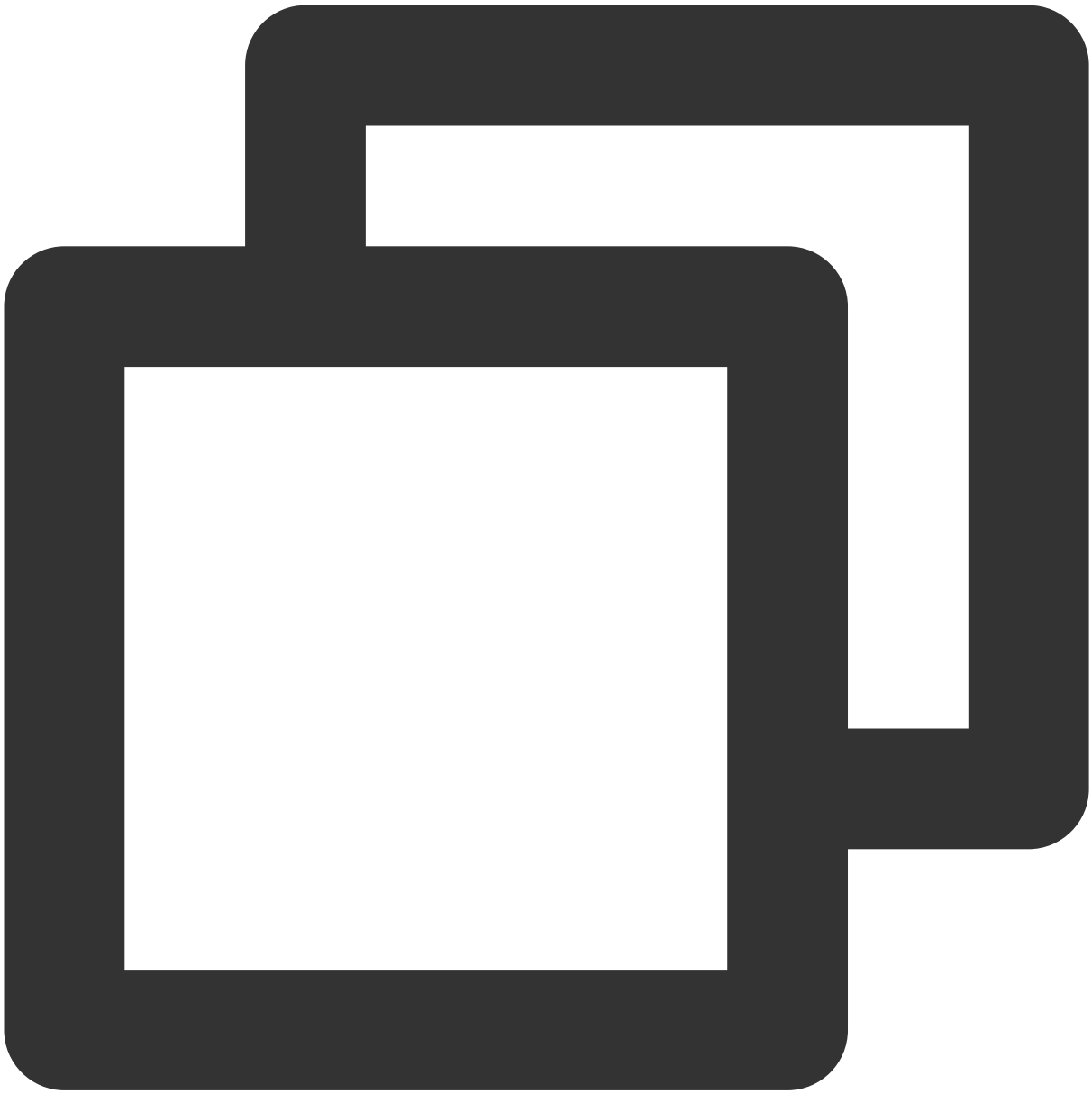
Using binary program to run exporter

Downloading and installing explorer

1. Download NGINX Prometheus Exporter from the community for your runtime environment.
2. Install NGINX Prometheus Exporter.

Enabling `NGINX stub_status` feature

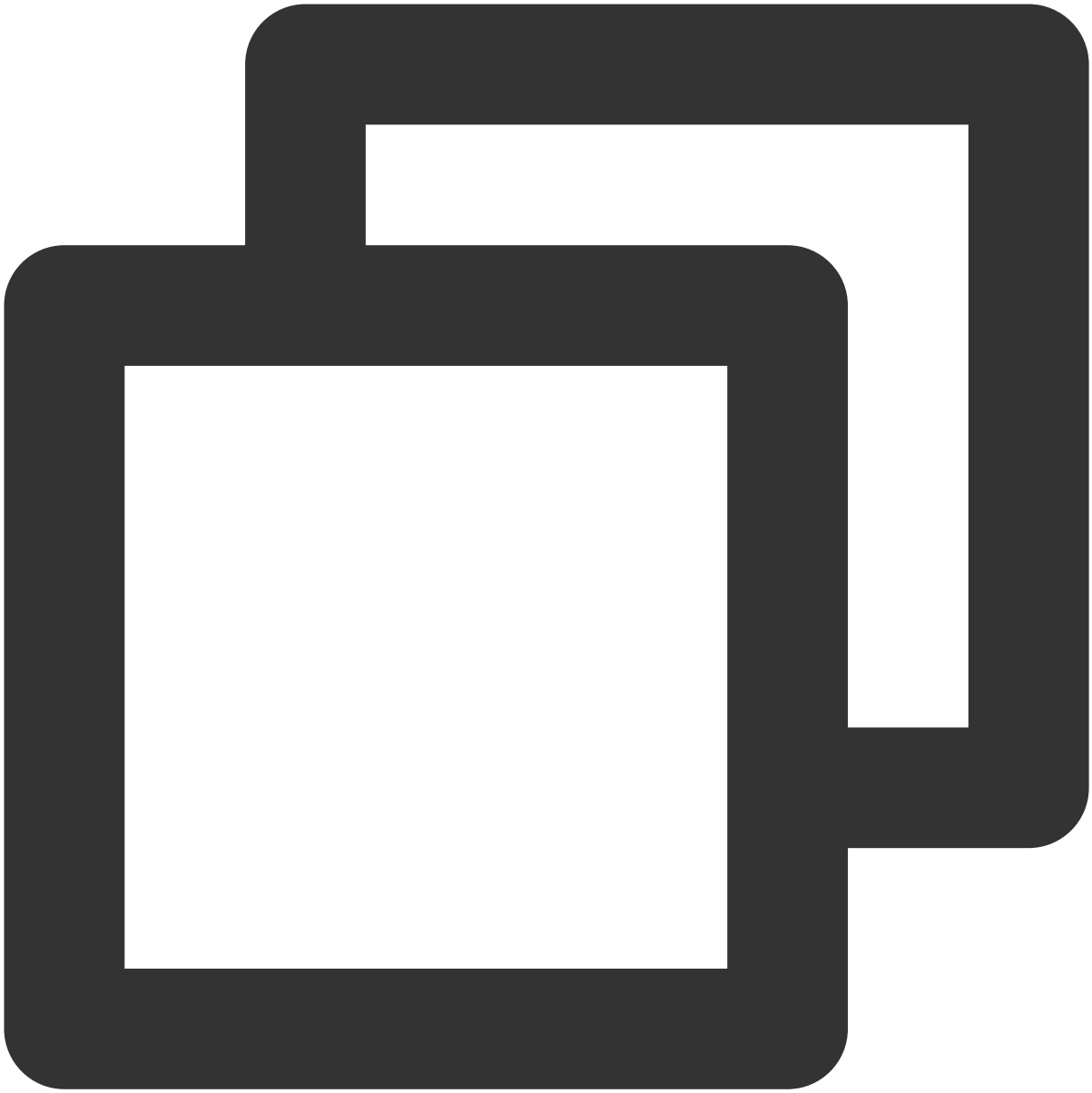
1. The stub_status module of open-source NGINX provides a simple page to display the status data. Run the following command to check whether this module is enabled in NGINX:



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

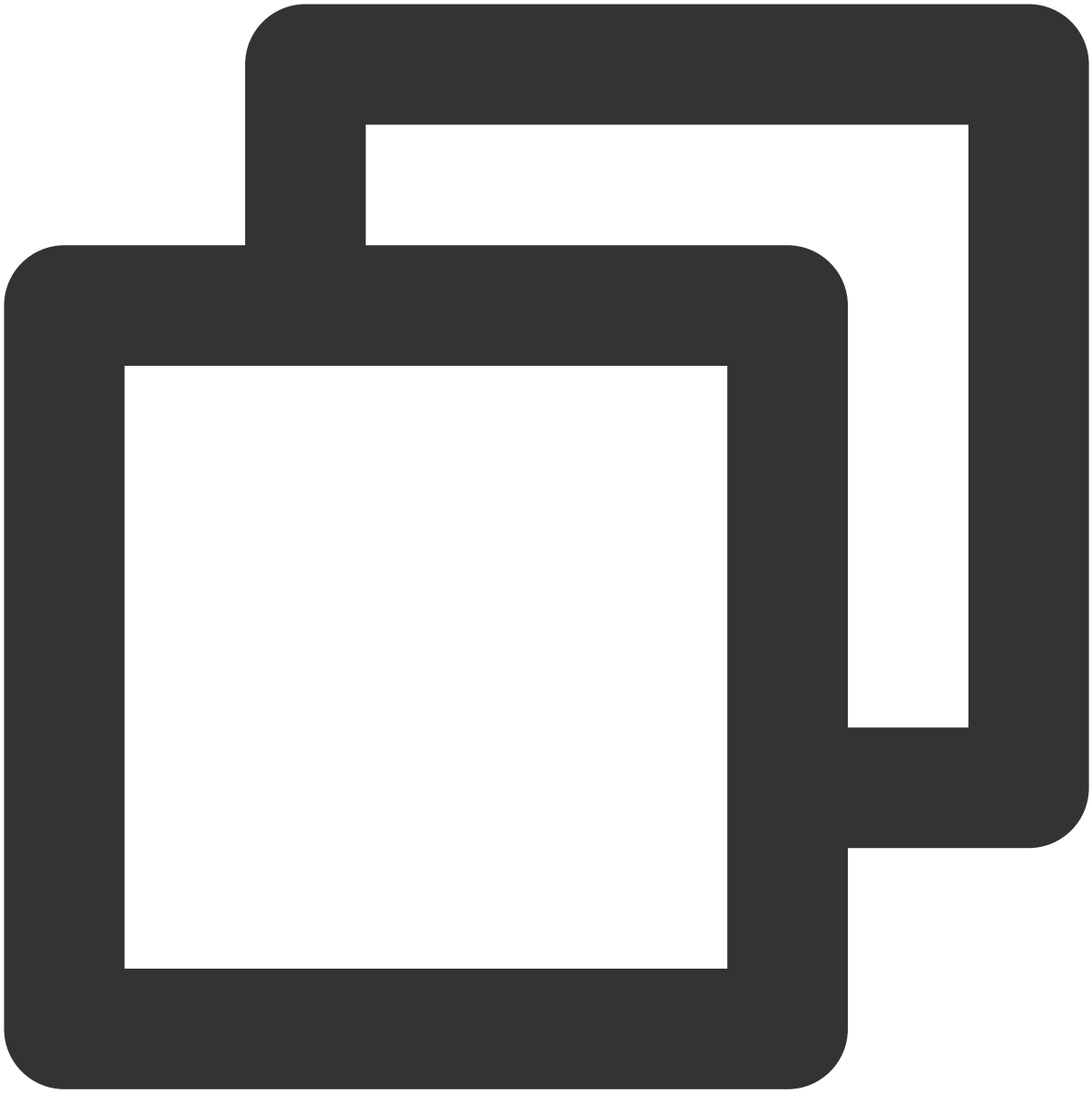
If `with-http_stub_status_module` is output on the terminal, the `stub_status` module in NGINX is enabled.

If no result is output, you can use the `--with-http_stub_status_module` parameter to configure and compile NGINX again from the source code. Below is the sample code:



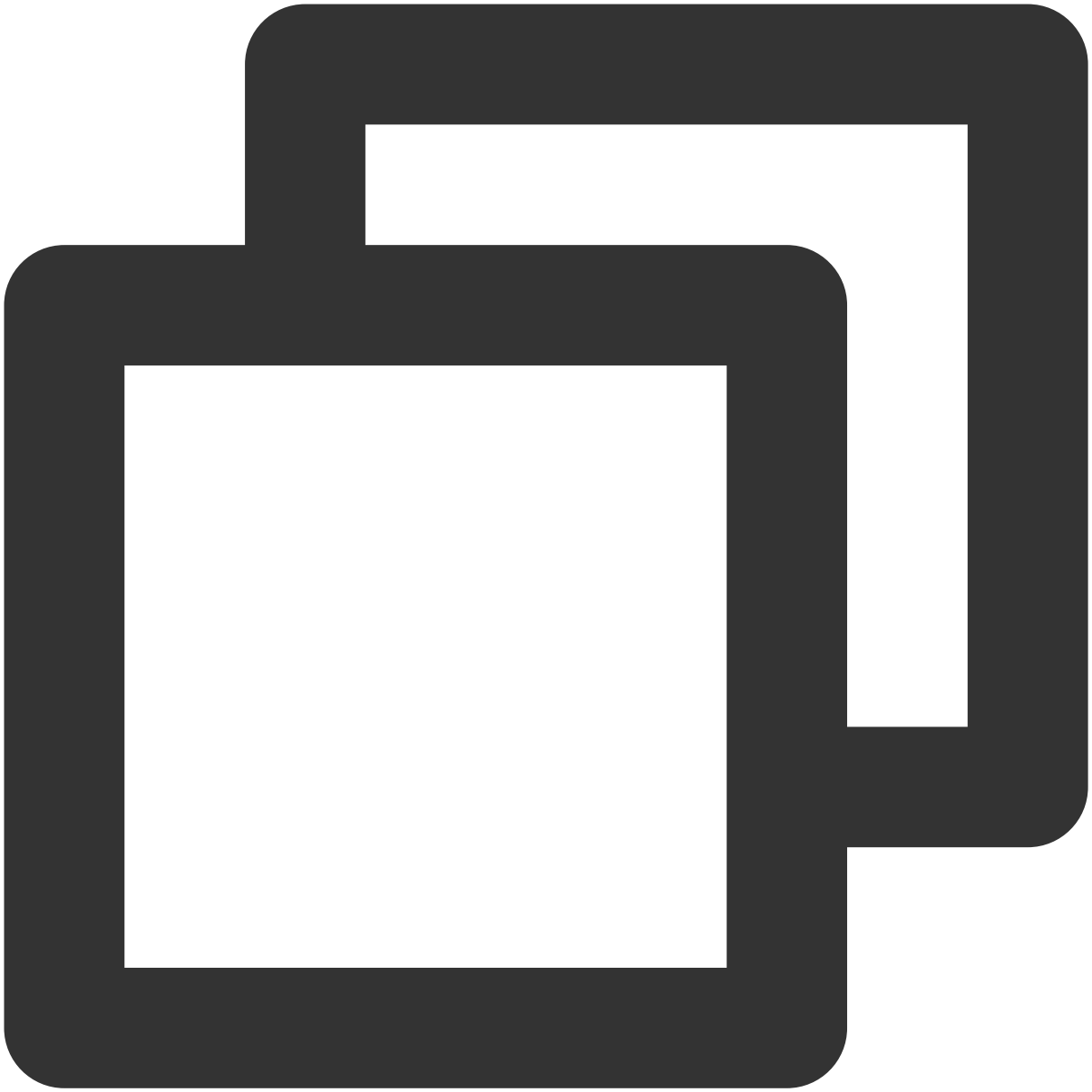
```
./configure \\
… \\--with-http_stub_status_module
make
sudo make install
```

2. After confirming that the `stub_status` module is enabled, modify the NGINX configuration file to specify the URL of the `stub_status` page as follows:



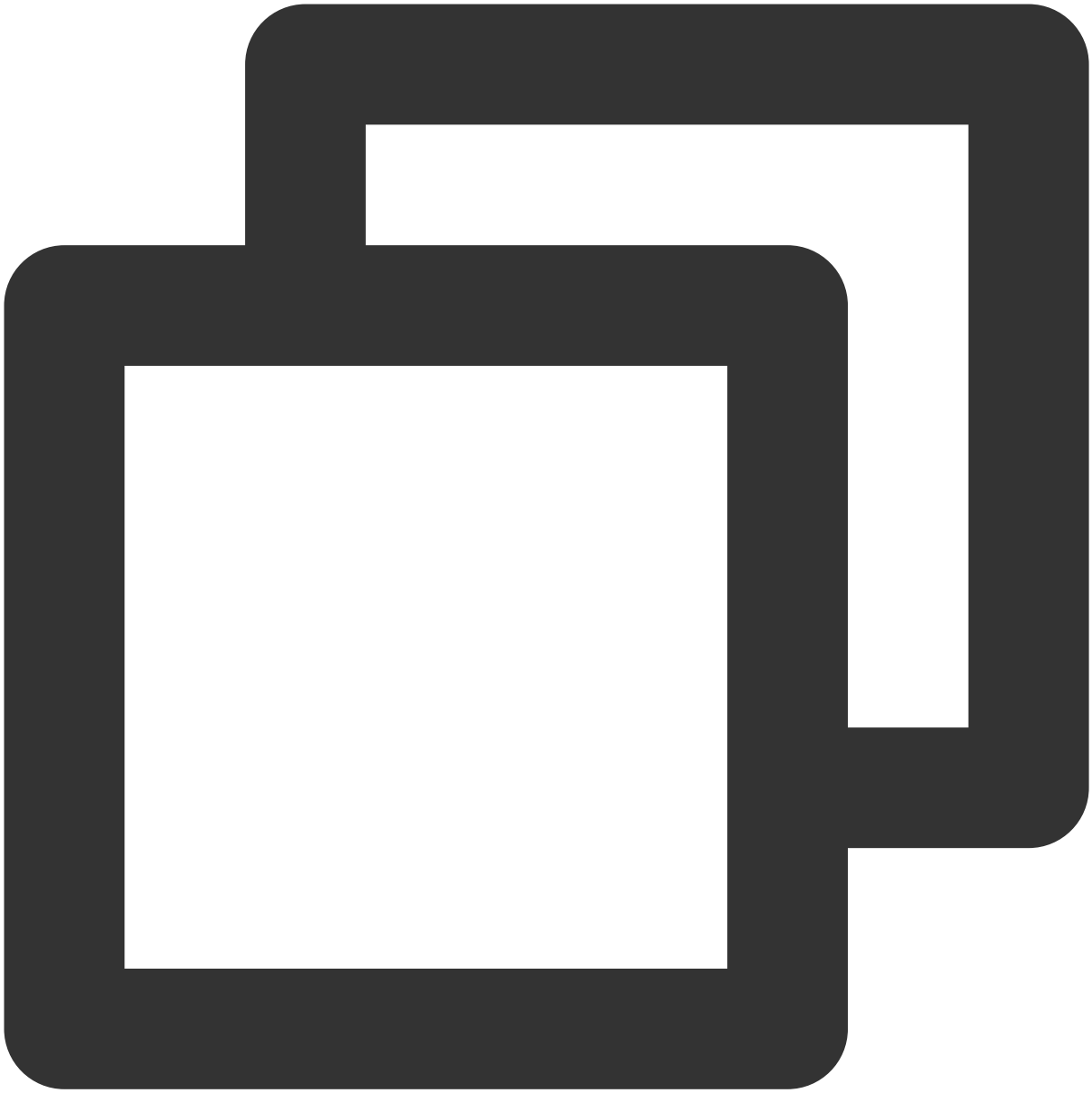
```
server {
 location /nginx_status {
     stub_status;

     access_log off;
     allow 127.0.0.1;
     deny all;
 }
}
```

3. Check NGINX and load it again to make the configuration take effect.



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

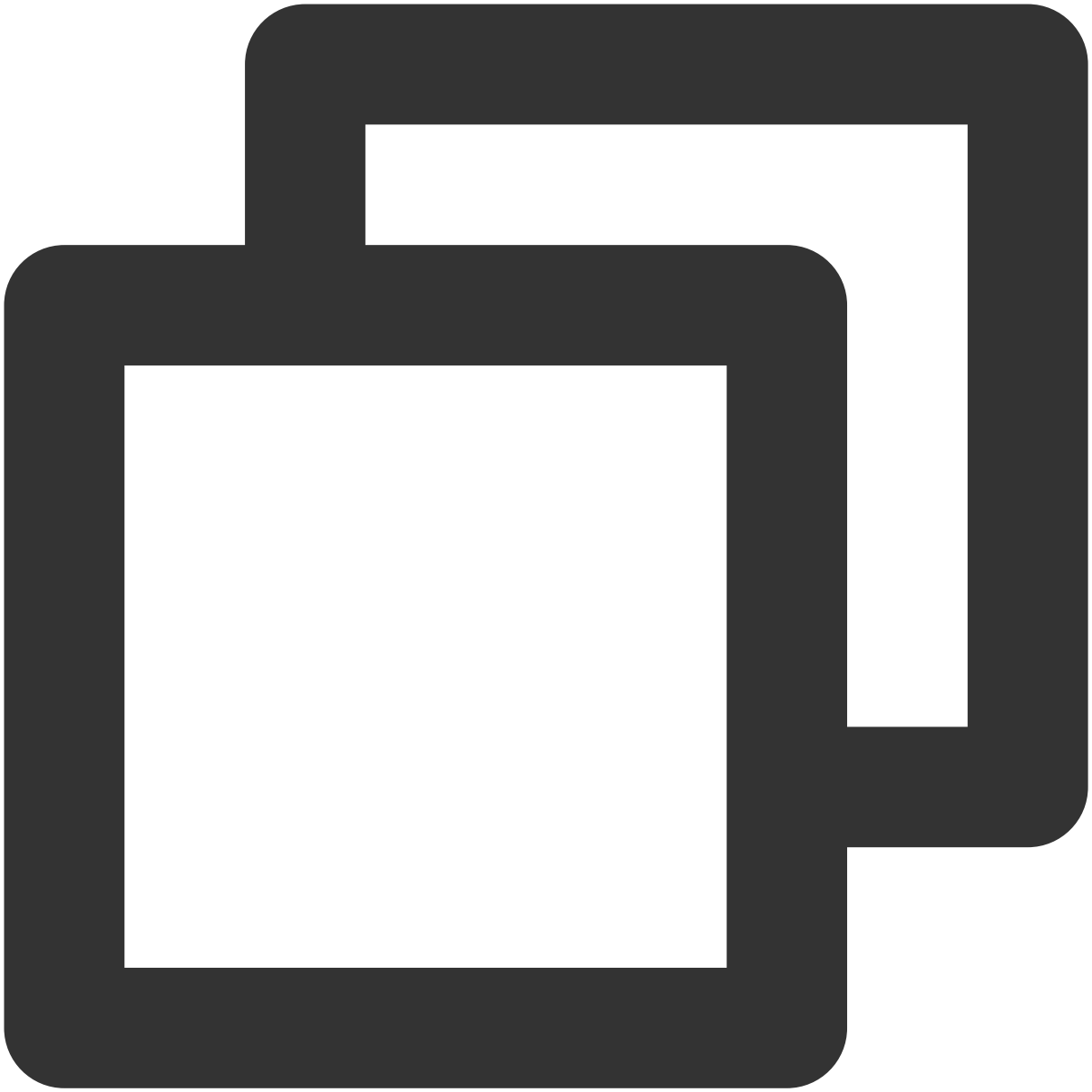
4. After completing the above steps, you can view the NGINX metrics through the configured URL.



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

Running NGINX Prometheus Exporter

Run the following command to start NGINX Prometheus Exporter:



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

Reported metrics

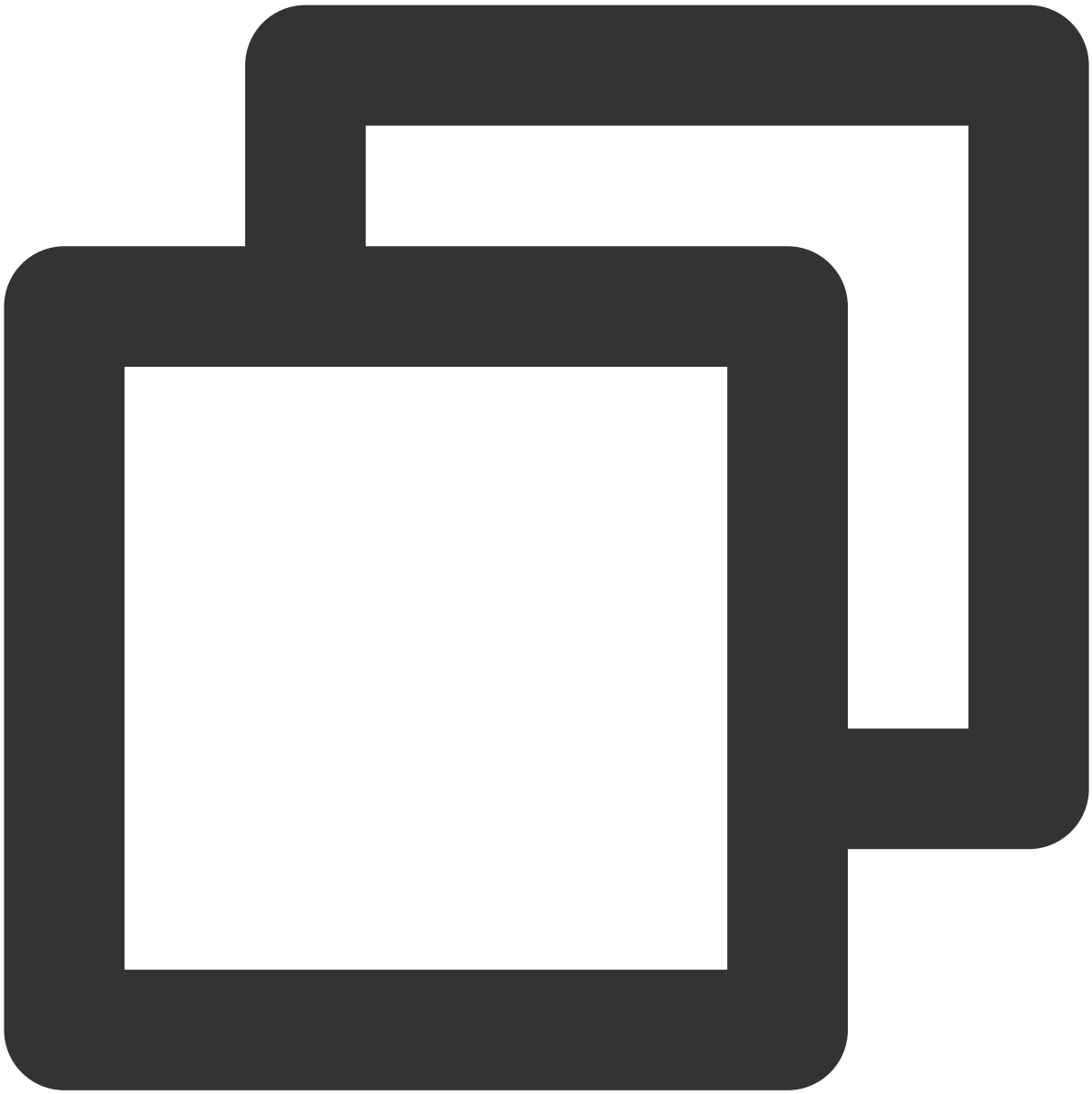
`nginxexporter_build_info` , which is the exporter compilation information.

All stub_status metrics.

`nginx_up` , which displays the last scrape status. `1` indicates success, while `0` indicates failure.

Configuring Prometheus scrape job

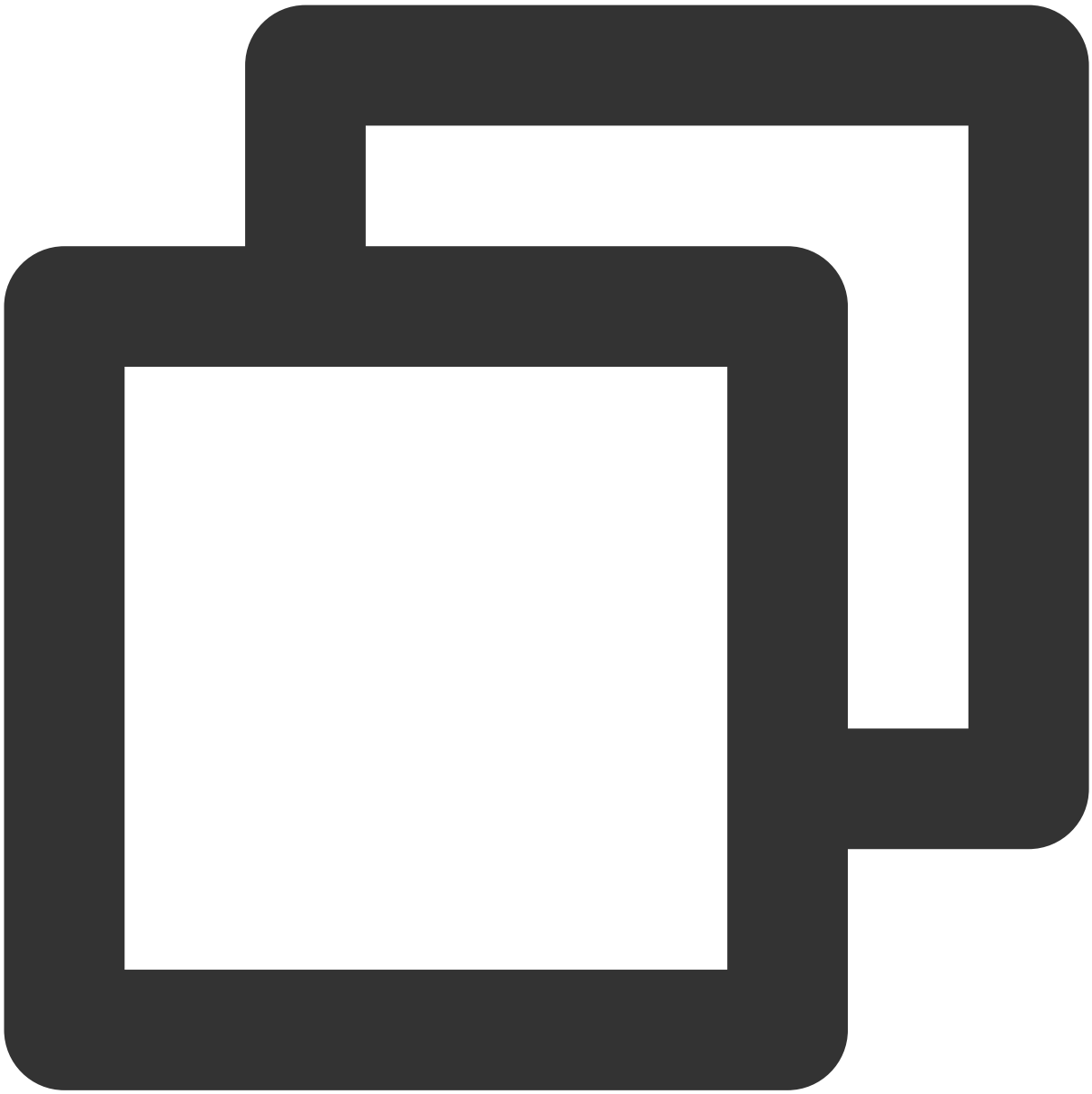
1. After NGINX Prometheus Exporter runs normally, run the following command to add a job to the Prometheus scrape task.



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

2. Generally, the exporter and NGINX do not run together, so the `instance` of the reported data cannot describe the real instance. To facilitate data search and observation, you can modify the `instance` label and replace it with

the real IP to make the label more intuitive as follows:



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

Enabling database monitoring dashboard

TMP provides a preconfigured NGINX exporter dashboard in Grafana. You can view the NGINX monitoring data in the following steps.

1. Log in to the TMP console.
2. Click

on the right of the corresponding instance ID to view the data.

Redis Exporter Integration

Last updated：2024-08-07 22:01:56

Overview

When using Redis, you need to monitor its running status to know whether it runs normally and troubleshoot its faults. TMP provides an exporter to monitor Redis and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to use TMP to monitor Redis.

Note:

For easier export installation and management, we recommend you use TKE for unified management.

Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance and created a namespace for the cluster. You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

Directions

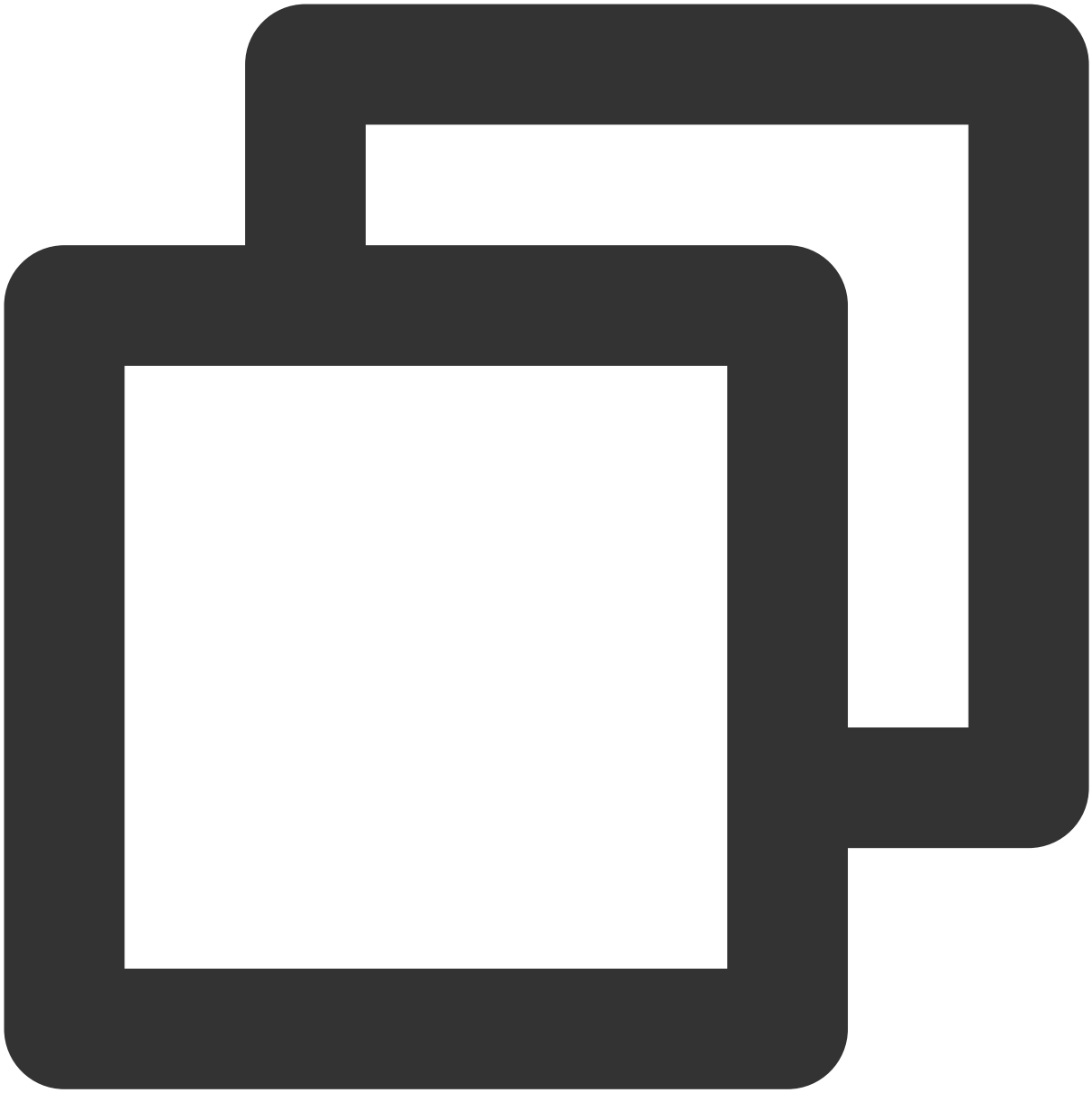
Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. Perform the following steps to deploy an exporter: Using Secret to manage Redis password > Deploying Redis exporter > Verifying.

Using Secret to manage Redis password

1. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
2. In the top-right corner of the page, click **Create via YAML** to create a YAML configuration as detailed below:

You can use Kubernetes Secrets to manage and encrypt passwords. When starting the Redis exporter, you can directly use the Secret key but need to adjust the corresponding `password` . Below is a sample YAML configuration:



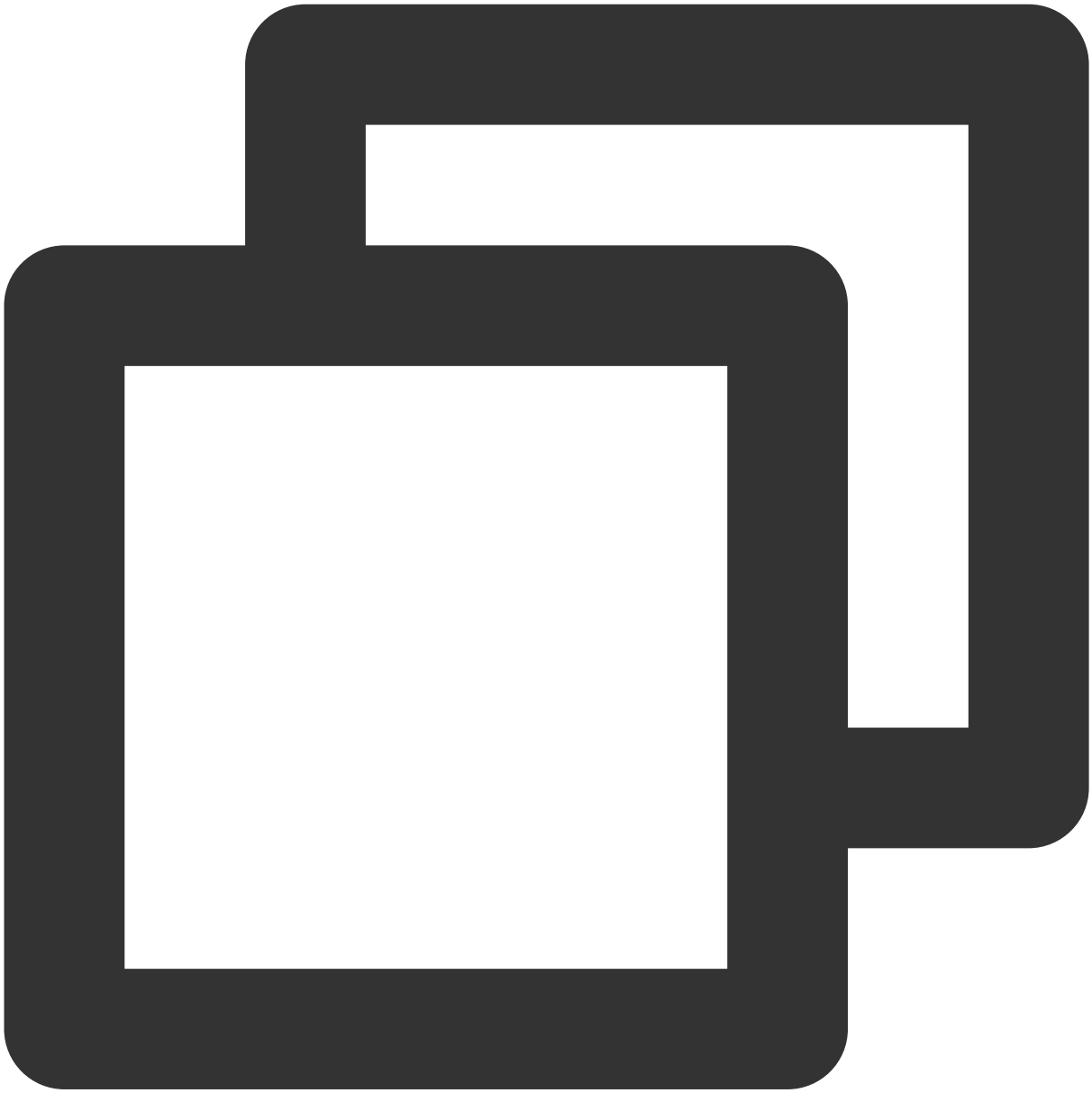
```
apiVersion: v1
kind: Secret
metadata:
    name: redis-secret-test
    namespace: redis-test
type: Opaque
stringData:
    password: you-guess  # Corresponding Redis password
```

Deploying Redis exporter

On the Deployment management page, click **Create** and select the target **namespace** to deploy the service. You can create in the console. Here, YAML is used to deploy the exporter. Below is a sample YAML configuration:

Note:

For more information on the detailed exporter parameters, please see redis_exporter.



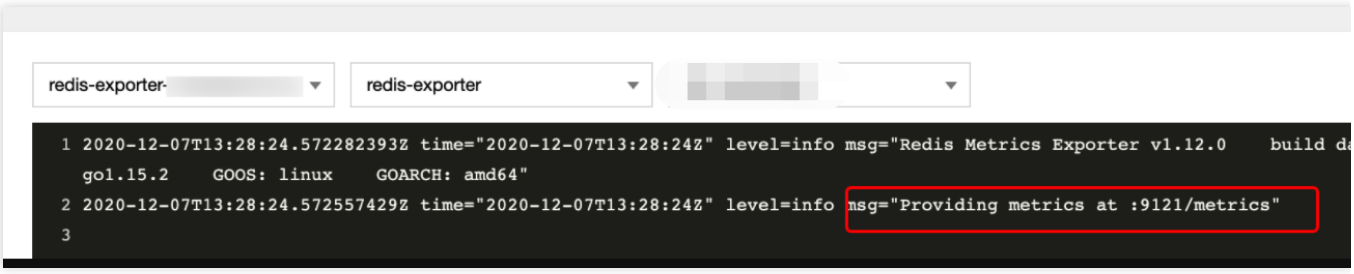
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: redis-exporter # Rename the exporter based on the business needs. We r
  name: redis-exporter # Rename the exporter based on the business needs. We recomm
```

```
  namespace: redis-test
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: redis-exporter # Rename the exporter based on the business needs. We
  template:
    metadata:
      labels:
        k8s-app: redis-exporter # Rename the exporter based on the business needs.
    spec:
      containers:
      - env:
        - name: REDIS_ADDR
          value: ip:port # `ip:port` of the corresponding Redis instance
        - name: REDIS_PASSWORD
          valueFrom:
            secretKeyRef:
              name: redis-secret-test
              key: password
        image: ccr.ccs.tencentyun.com/redis-operator/redis-exporter:1.12.0
        imagePullPolicy: IfNotPresent
        name: redis-exporter
        ports:
        - containerPort: 9121
          name: metric-port  # This name is required during scrape task configurati
        securityContext:
          privileged: false
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

Verifying

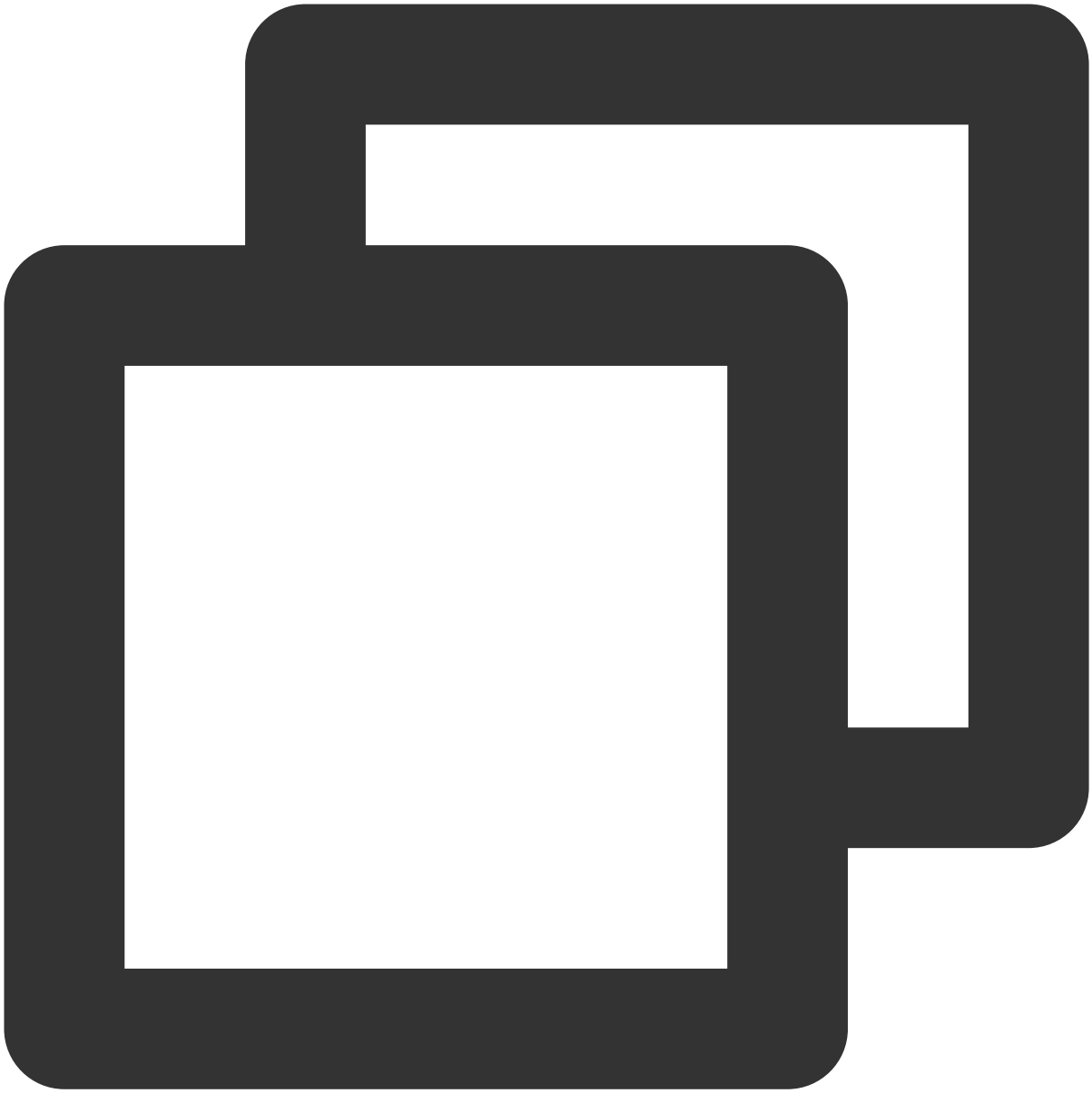
1. Click the newly created Deployment on the **Deployment** page to enter the Deployment management page.

2. Click the **Log** tab, and you can see that the exporter is successfully started and its address is exposed as shown below:



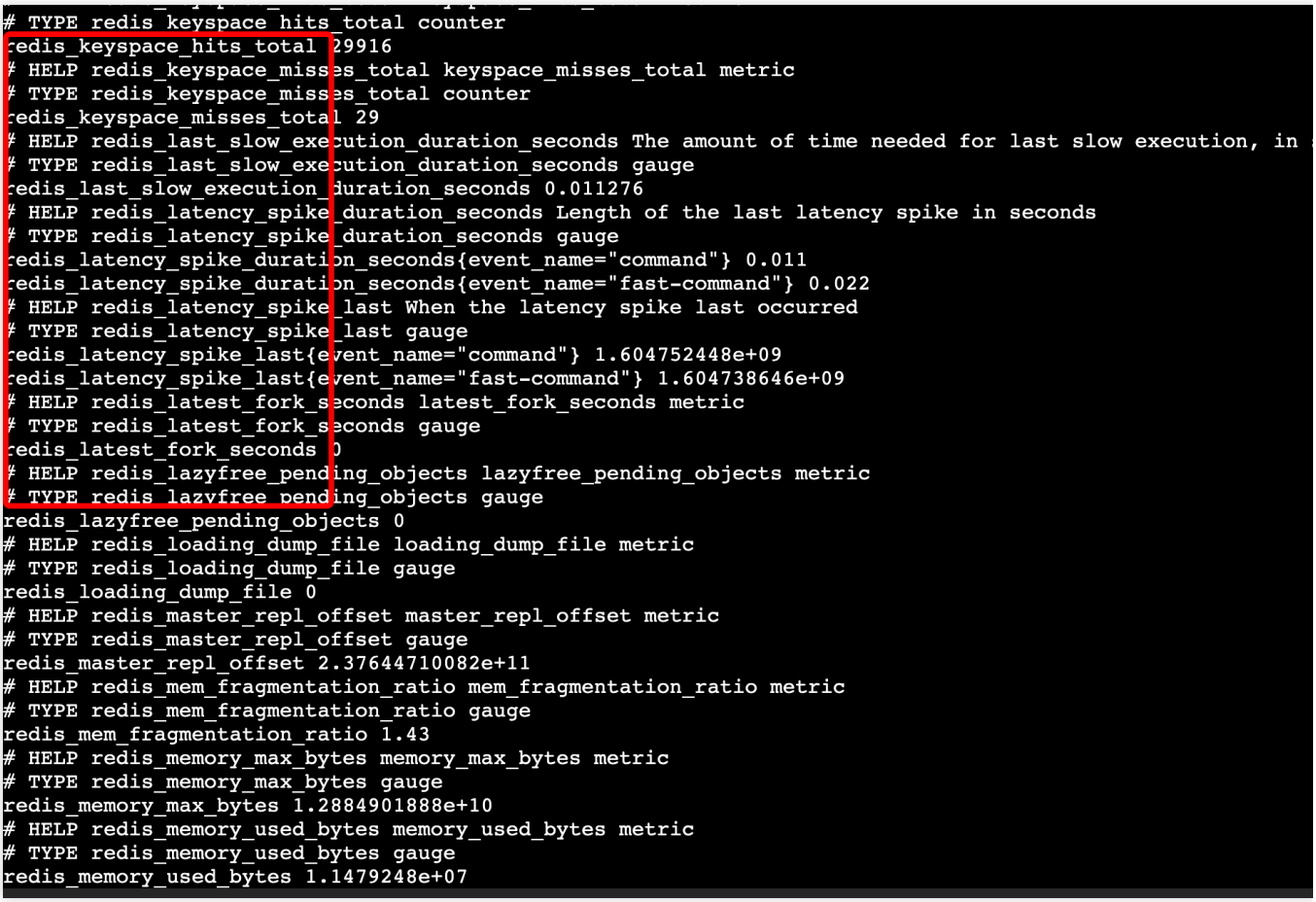
3. Click the **Pod Management** tab to enter the Pod page.

4. In the **Operations** column on the right, click **Remote Login** to log in to the Pod. Run the following `curl` command with the address exposed by the exporter in the command line window, and you can get the corresponding Redis metrics normally. If no corresponding data is returned, please check whether `REDIS_ADDR` and `REDIS_PASSWORD` are correct as shown below:



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

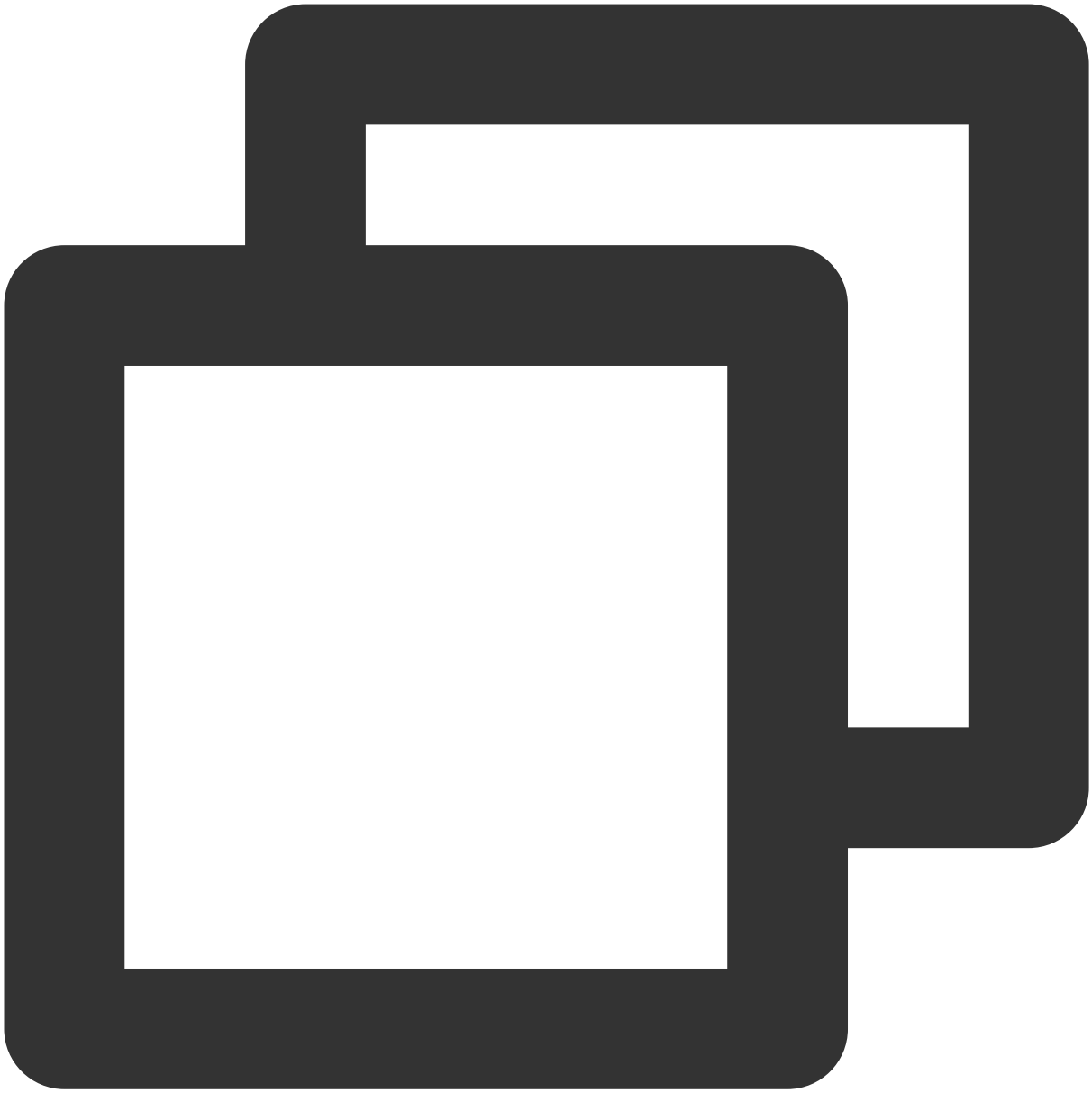
The command execution result is as shown below:



```
# TYPE redis_keyspace_hits_total counter
redis_keyspace_hits_total 29916
# HELP redis_keyspace_misses_total keyspace_misses_total metric
# TYPE redis_keyspace_misses_total counter
redis_keyspace_misses_total 29
# HELP redis_last_slow_execution_duration_seconds The amount of time needed for last slow execution, in
# TYPE redis_last_slow_execution_duration_seconds gauge
redis_last_slow_execution_duration_seconds 0.011276
# HELP redis_latency_spike_duration_seconds Length of the last latency spike in seconds
# TYPE redis_latency_spike_duration_seconds gauge
redis_latency_spike_duration_seconds{event_name="command"} 0.011
redis_latency_spike_duration_seconds{event_name="fast-command"} 0.022
# HELP redis_latency_spike_last When the latency spike last occurred
# TYPE redis_latency_spike_last gauge
redis_latency_spike_last{event_name="command"} 1.604752448e+09
redis_latency_spike_last{event_name="fast-command"} 1.604738646e+09
# HELP redis_latest_fork_seconds latest_fork_seconds metric
# TYPE redis_latest_fork_seconds gauge
redis_latest_fork_seconds 0
# HELP redis_lazyfree_pending_objects lazyfree_pending_objects metric
# TYPE redis_lazyfree_pending_objects gauge
redis_lazyfree_pending_objects 0
# HELP redis_loading_dump_file loading_dump_file metric
# TYPE redis_loading_dump_file gauge
redis_loading_dump_file 0
# HELP redis_master_repl_offset master_repl_offset metric
# TYPE redis_master_repl_offset gauge
redis_master_repl_offset 2.37644710082e+11
# HELP redis_mem_fragmentation_ratio mem_fragmentation_ratio metric
# TYPE redis_mem_fragmentation_ratio gauge
redis_mem_fragmentation_ratio 1.43
# HELP redis_memory_max_bytes memory_max_bytes metric
# TYPE redis_memory_max_bytes gauge
redis_memory_max_bytes 1.2884901888e+10
# HELP redis_memory_used_bytes memory_used_bytes metric
# TYPE redis_memory_used_bytes gauge
redis_memory_used_bytes 1.1479248e+07
```

Adding scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:



```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port  # Enter the name of the corresponding port of the Promethe
    path: /metrics  # Enter the value of the corresponding path of the Prometheus
    relabelings:
```

```
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # Change it to the corresponding Redis instance I
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # Change it to the corresponding Redis instance IP
  namespaceSelector:   # Select the namespace where the Pod to be monitored resid
    matchNames:
    - redis-test
  selector:    # Enter the label value of the Pod to be monitored to locate the t
    matchLabels:
      k8s-app: redis-exporter
```

Note:

As the exporter and Redis are deployed on different servers, we recommend you use the Prometheus relabeling mechanism to add the Redis instance information to the monitoring metrics so as to locate problems more easily.

Viewing monitoring information

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Integration Center** to enter the **Integration Center** page. Find Redis monitoring, install the corresponding Grafana dashboard, and then you can enable the Redis monitoring dashboard to view instance monitoring data as shown below:

Integrating with alert feature

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Alerting Rule** and add the corresponding alerting rules. For more information, please see Creating Alerting Rule.

MySQL Exporter Integration

Last updated：2024-08-07 22:02:03

Overview

The MySQL exporter is specially designed and developed by the Prometheus community to collect MySQL/MariaDB database monitoring metrics. The exporter reports core database metrics, which can be used for exception alerting and displayed on the monitoring dashboard. TMP supports integration with the MySQL exporter and provides an out-of-the-box Grafana monitoring dashboard.
Currently, the exporter supports MySQL 5.6 or above and MariaDB 10.1 or above. If MySQL or MariaDB is below 5.6 or 10.1 respectively, some monitoring metrics may fail to be collected.
Note:
For easier export installation and management, we recommend you use TKE for unified management.

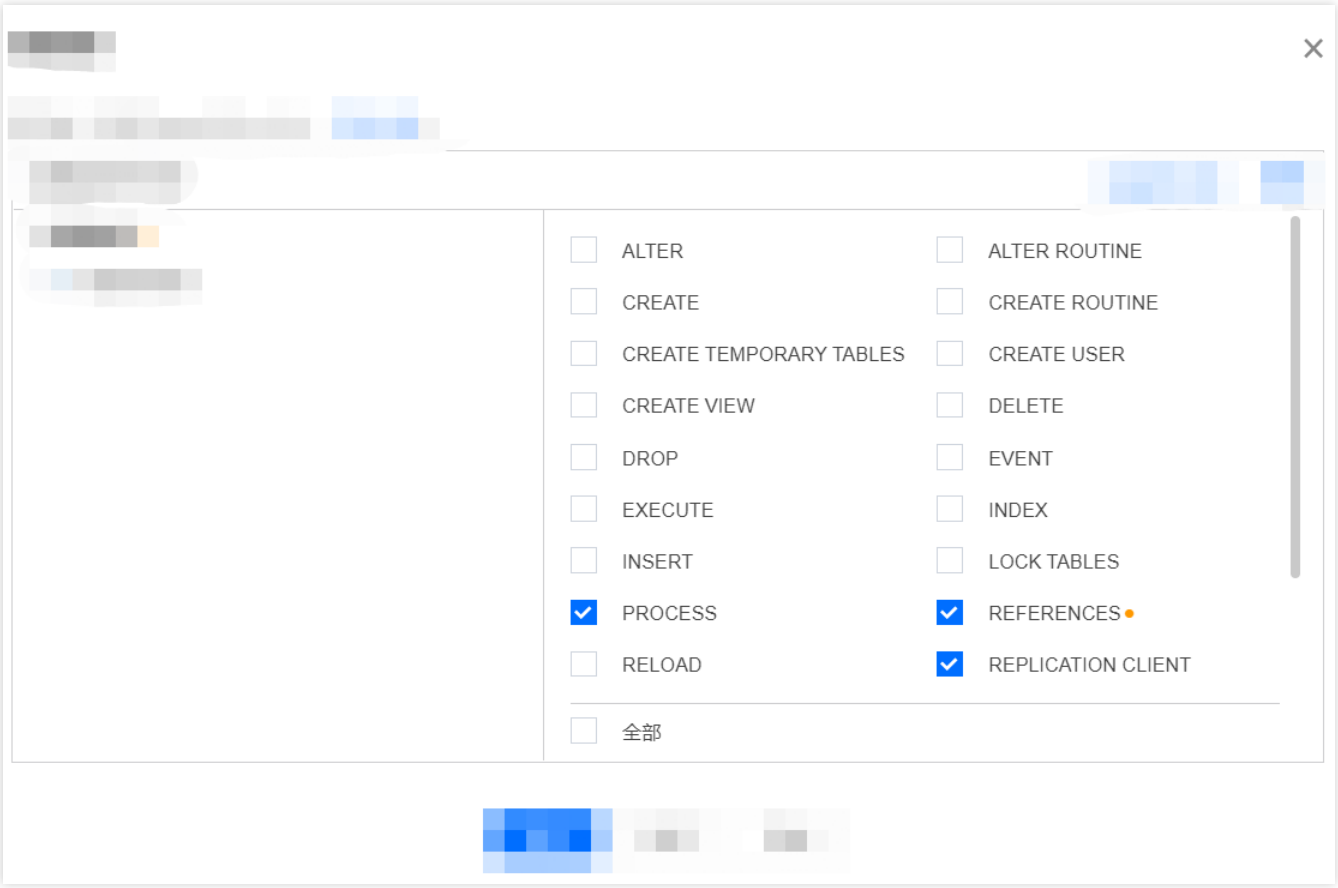
Prerequisites

You have created a TKE cluster in the region and VPC of your TMP instance and created a namespace for the cluster.
You have located and integrated the target TKE cluster in the **Integrate with TKE** section of the **target TMP instance** in the **TMP console**. For more information, please see Agent Management.

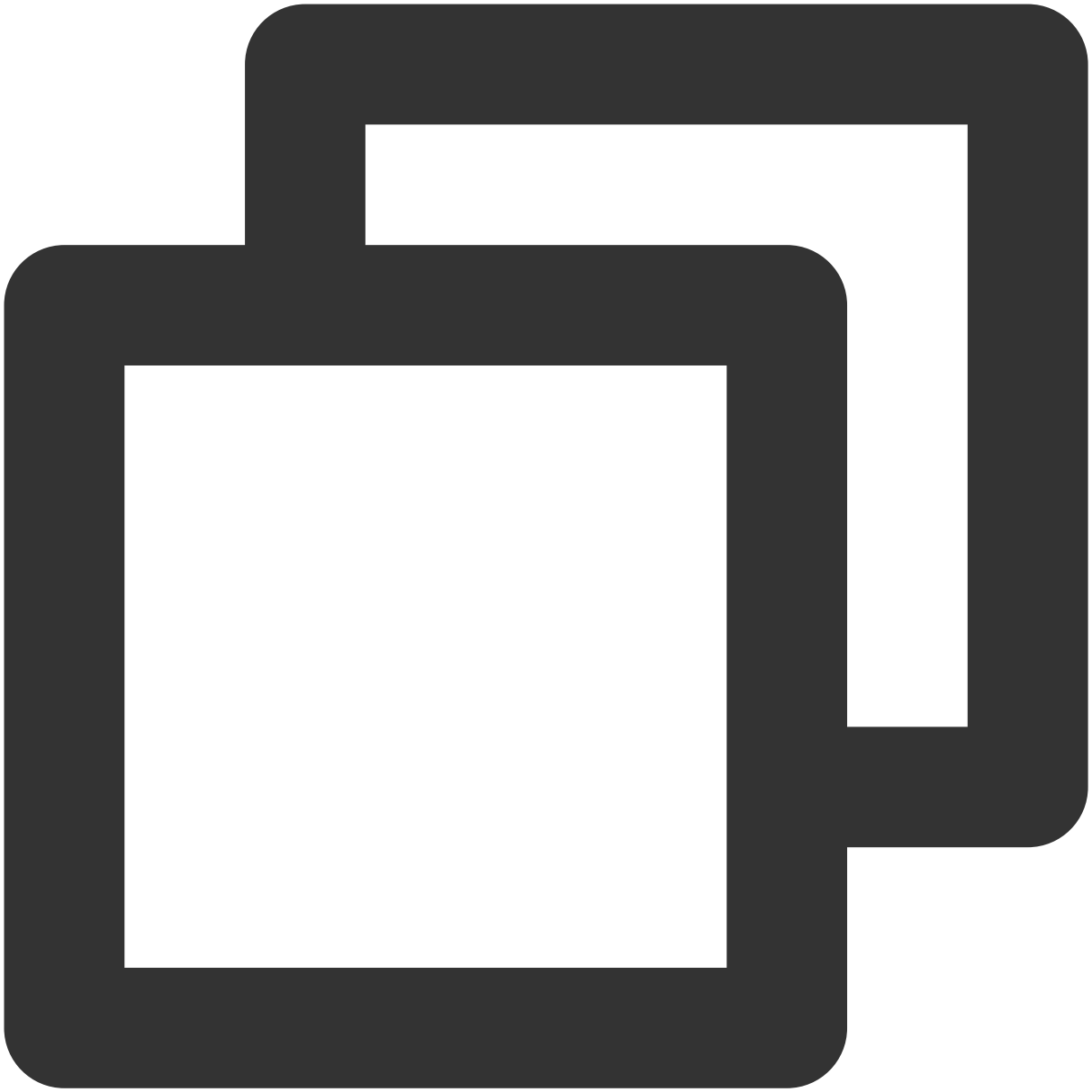
Directions

Authorizing in database

As the MySQL exporter monitors a database by querying its status data, you need to grant the exporter access to the corresponding database instance. The account and password should be set based on the actual conditions. The authorization steps are as follows:
1. Log in to the TencentDB for MySQL console.
2. On the instance list page, click the name of the database for which to authorize the exporter to enter the database details page.
3. Select **Database Management** > **Account Management** to enter the account management page and create an account for monitoring based on the actual business needs.
4. Click **Modify Permissions** in the **Operation** column on the right of the account to modify the corresponding permissions as shown below:



You can run the following command for authorization:



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

Note:

We recommend you set the allowed maximum number of connections for the account to avoid any impact on the database due to monitoring data collection. However, not all database versions support this configuration, for example, MariaDB 10.1. For more information, please see Resource Limit Options.

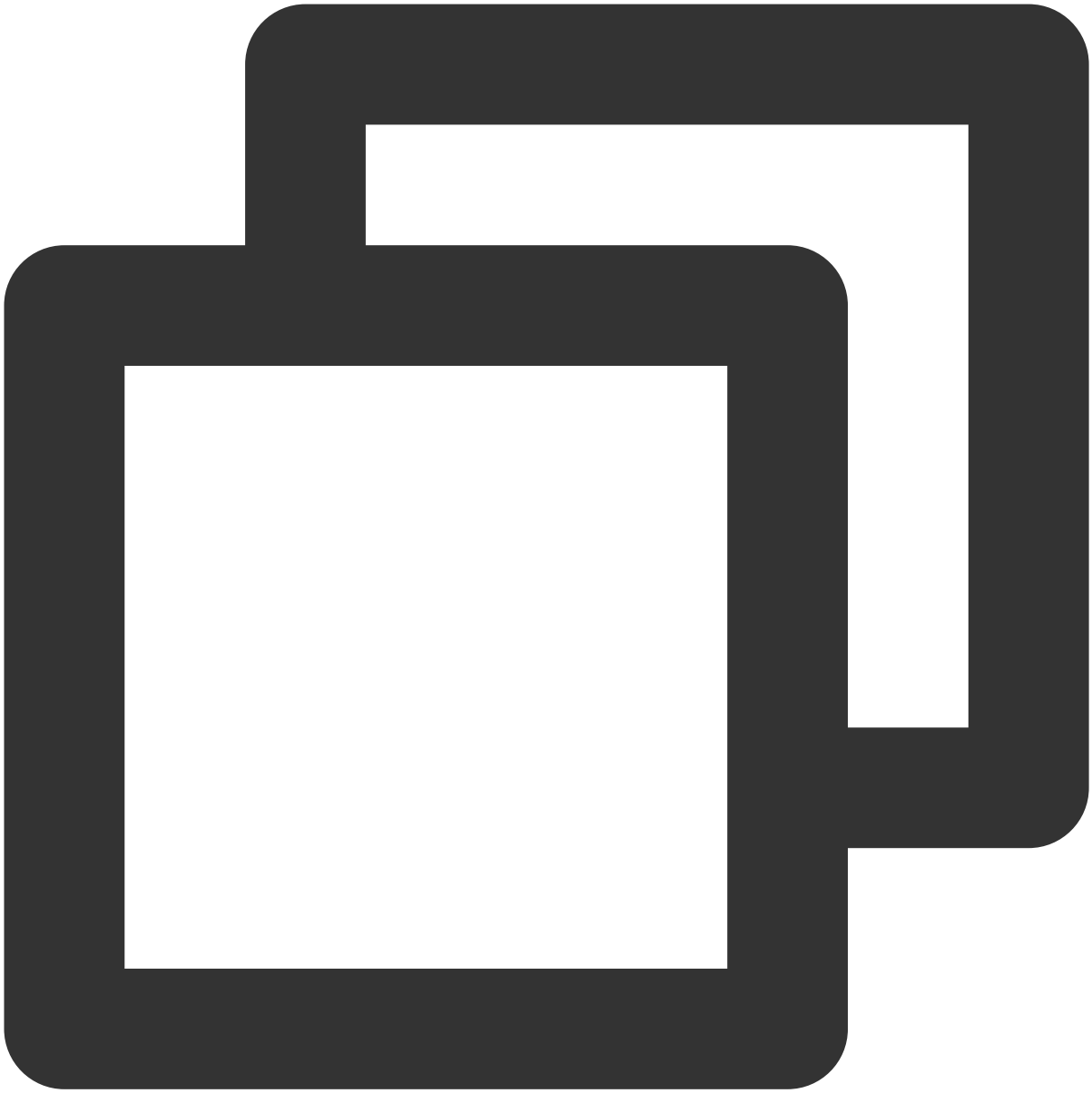
Deploying exporter

1. Log in to the TKE console.
2. Click the ID/name of the cluster whose access credential you want to get to enter the cluster management page.
3. Perform the following steps to deploy an exporter: Using Secret to manage MySQL connection string > Deploying MySQL exporter > Verifying.

Using Secret to manage MySQL connection string

1. On the left sidebar, select **Workload** > **Deployment** to enter the **Deployment** page.
2. In the top-right corner of the page, click **Create via YAML** to create a YAML configuration as detailed below:

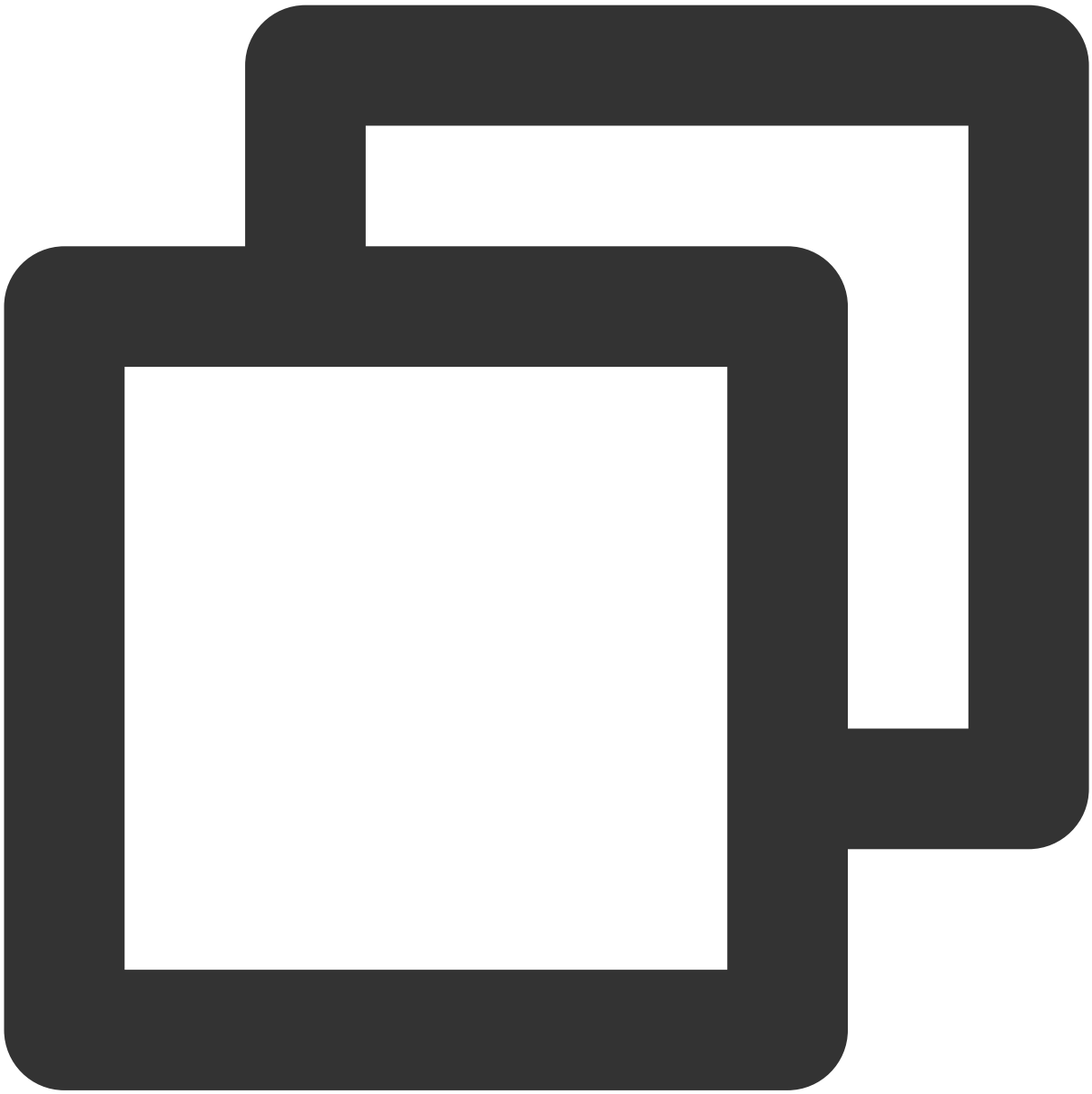
You can use Kubernetes Secrets to manage and encrypt connection strings. When starting the MySQL exporter, you can directly use the Secret key but need to adjust the corresponding **connection string**. Below is a sample YAML configuration:



```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret-test
  namespace: mysql-demo
type: Opaque
stringData:
  datasource: "user:password@tcp(ip:port)/"  # Corresponding MySQL connection strin
```

Deploying MySQL exporter

On the Deployment management page, select the target namespace to deploy the service. You can create in the console. Here, YAML is used to deploy the exporter. Below is a sample configuration:

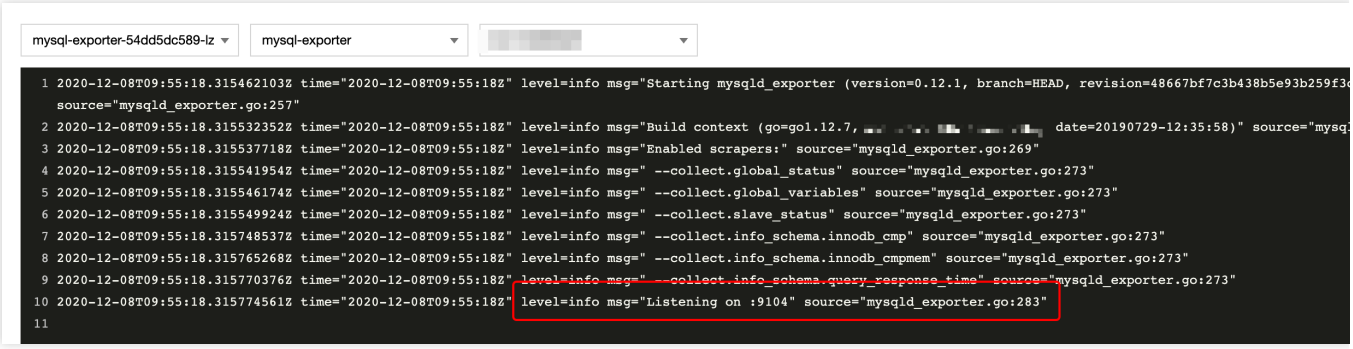


```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mysql-exporter  # Rename the exporter based on the business needs. We
  name: mysql-exporter  # Rename the exporter based on the business needs. We recom
  namespace: mysql-demo
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mysql-exporter  # Rename the exporter based on the business needs. W
  template:
    metadata:
      labels:
        k8s-app: mysql-exporter  # Rename the exporter based on the business needs.
    spec:
      containers:
      - env:
        - name: DATA_SOURCE_NAME
          valueFrom:
            secretKeyRef:
              name: mysql-secret-test
              key: datasource
        image: ccr.ccs.tencentyun.com/k8s-comm/mysqld-exporter:0.12.1
        imagePullPolicy: IfNotPresent
        name: mysql-exporter
        ports:
        - containerPort: 9104
          name: metric-port
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      imagePullSecrets:
      - name: qcloudregistrykey
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

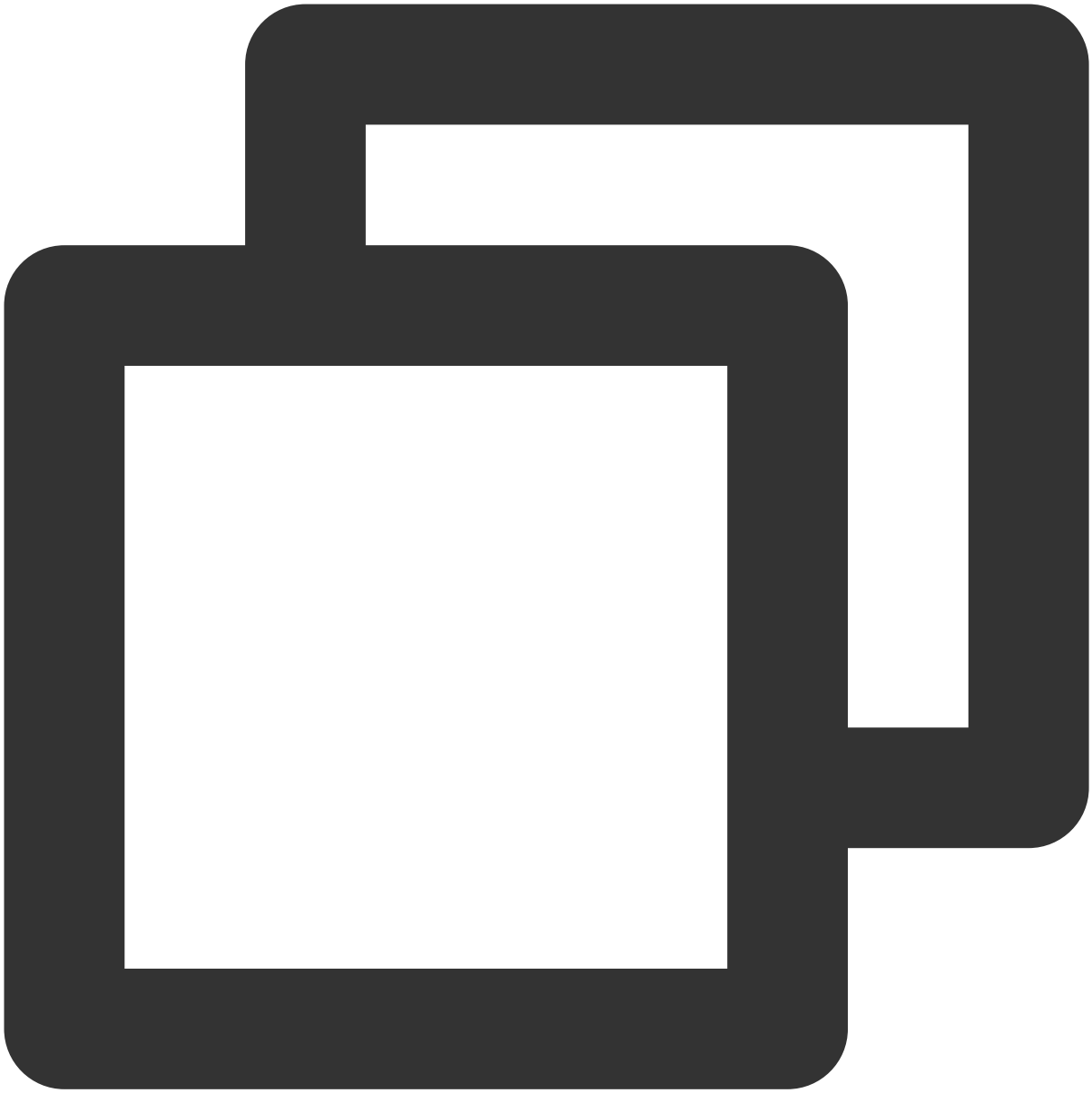
Verifying

1. Click the newly created Deployment on the **Deployment** page to enter the Deployment management page.
2. Click the **Log** tab, and you can see that the exporter is successfully started and its address is exposed as shown below:



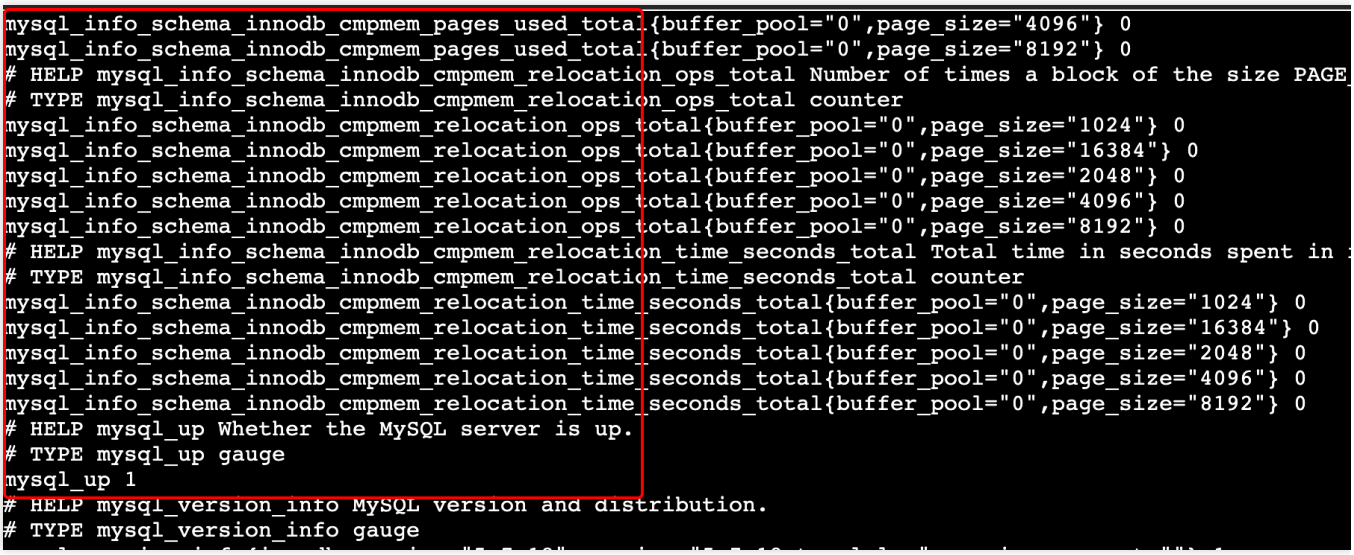
3. Click the **Pod Management** tab to enter the Pod page.

4. In the **Operations** column on the right, click **Remote Login** to log in to the Pod. Run the following `curl` command with the address exposed by the exporter in the command line window, and you can get the corresponding MySQL metrics normally. If no corresponding data is returned, please check whether the **connection string** is correct as shown below:



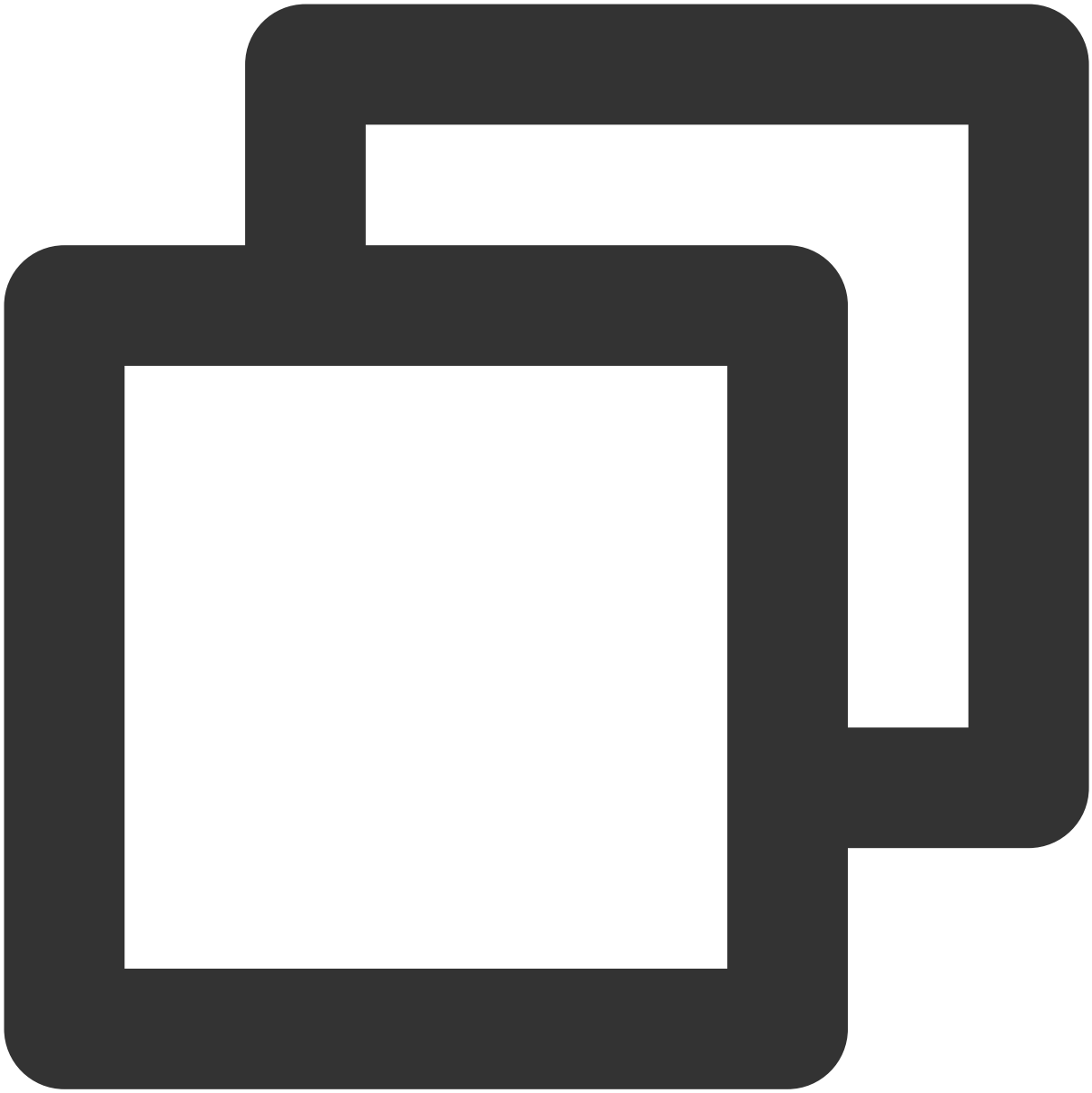
```
curl localhost:9104/metrics
```

The execution result is as shown below:



Adding scrape task

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:



```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: mysql-exporter  # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port    # Enter the name of the corresponding port of the Promet
    path: /metrics  # Enter the value of the corresponding path of the Prometheus
    relabelings:
```

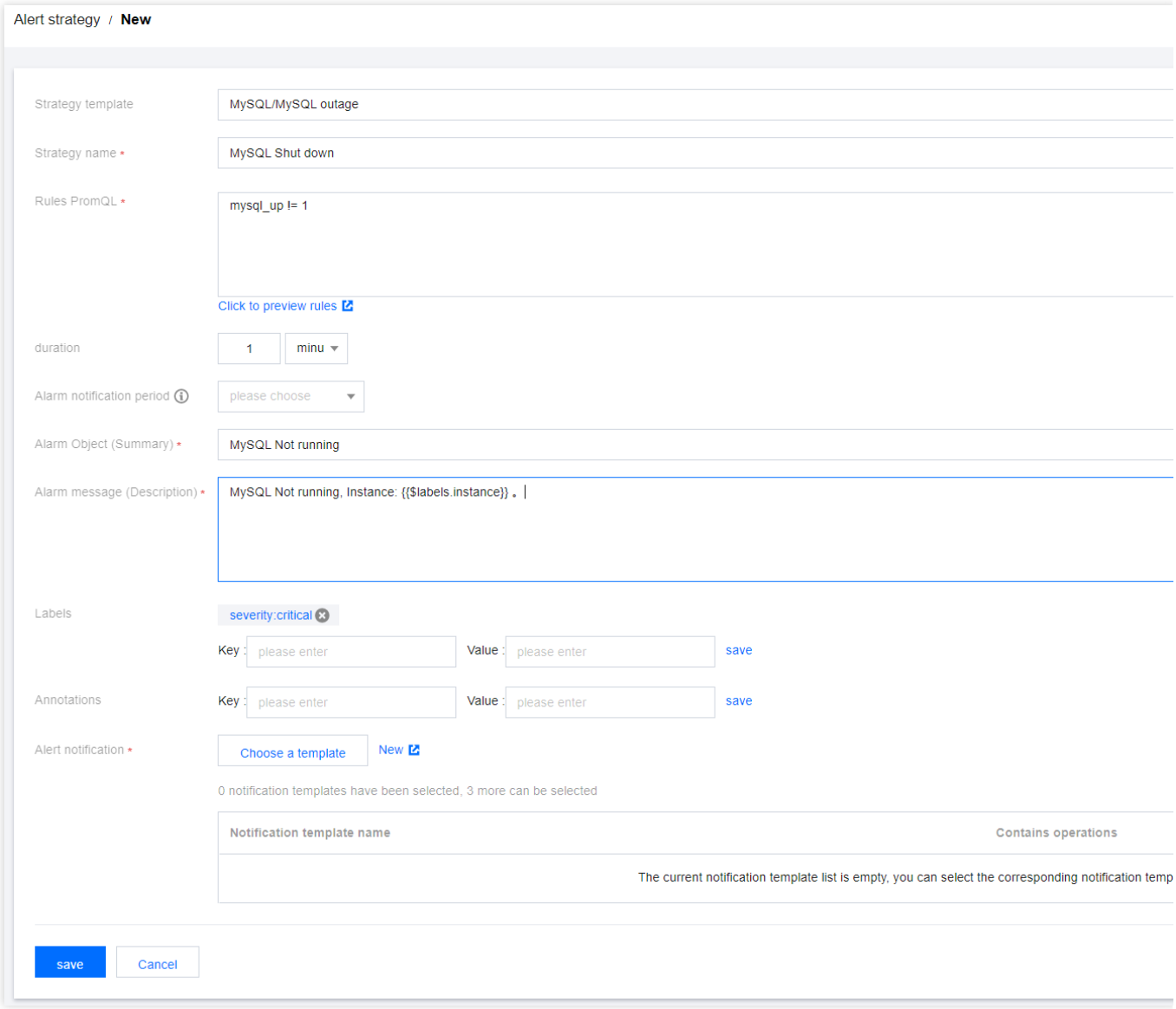
```
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: instance
      replacement: 'crs-xxxxxx' # Change it to the corresponding MySQL instance I
    - action: replace
      sourceLabels:
      - instance
      regex: (.*)
      targetLabel: ip
      replacement: '1.x.x.x' # Change it to the corresponding MySQL instance IP
  namespaceSelector:    # Select the namespace where the Pod to be monitored resid
    matchNames:
    - mysql-demo
  selector:  # Enter the label value of the Pod to be monitored to locate the tar
    matchLabels:
      k8s-app: mysql-exporter
```

Viewing monitoring information

1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click **Integration Center** to enter the **Integration Center** page. Find MySQL monitoring, install the corresponding Grafana dashboard, and then you can enable the MySQL monitoring dashboard to view instance monitoring data as shown below:

Integrating with alert feature

TMP has some built-in MySQL alerting rule templates. You can adjust the corresponding thresholds to add alerting rules based on your actual business conditions. For more information, please see Creating Alerting Rule.



MySQL Exporter Collection Parameter Description

The MySQL exporter uses various `collectors` to enable/disable data collection. The specific parameters are as listed below:

Parameter	MySQL Version	Description
collect.auto_increment.columns	5.1	Collects `auto_increment` columns

collect.binlog_size	5.1	Collects the current size of all registered
collect.engine_innodb_status	5.1	Collects the status data from `SHOW EN`
collect.engine_tokudb_status	5.6	Collects the status data from `SHOW EN`
collect.global_status	5.1	Collects the status data from `SHOW GL`
collect.global_variables	5.1	Collects the status data from `SHOW GL`
collect.info_schema.clientstats	5.5	If `userstat=1` is set, this parameter collection.
collect.info_schema.innodb_metrics	5.6	Collects the monitoring data from `info`
collect.info_schema.innodb_tablespaces	5.7	Collects the monitoring data from `information_schema.innodb_sy`
collect.info_schema.innodb_cmp	5.5	Collects the monitoring data of compress `information_schema.innodb_cm`
collect.info_schema.innodb_cmpmem	5.5	Collects the monitoring data of InnoDB b `information_schema.innodb_cm`
collect.info_schema.processlist	5.1	Collects the monitoring data of the thread `information_schema.processli`
collect.info_schema.processlist.min_time	5.1	Minimum time a thread must be in each s
collect.info_schema.query_response_time	5.5	Collects query response time distribution to `ON` .
collect.info_schema.replica_host	5.6	Collects the status data from `informa`
collect.info_schema.tables	5.1	Collects the status data from `informa`
collect.info_schema.tables.databases	5.1	Sets the list of databases to collect table
collect.info_schema.tablestats	5.1	If `userstat=1` is set, this parameter statistics.
collect.info_schema.schemastats	5.1	If `userstat=1` is set, this parameter statistics.
collect.info_schema.userstats	5.1	If `userstat=1` is set, this parameter statistics.
collect.perf_schema.eventsstatements	5.6	Collects the monitoring data from

		`performance_schema.events_st`
collect.perf_schema.eventsstatements.digest_text_limit	5.6	Sets the maximum length of the normaliz
collect.perf_schema.eventsstatements.limit	5.6	Limits the number of event statements. D
collect.perf_schema.eventsstatements.timelimit	5.6	Limits how old the 'last_seen' events stat 86400.
collect.perf_schema.eventsstatementssum	5.7	Collects the monitoring data from `performance_schema.events_st summed` .
collect.perf_schema.eventswaits	5.5	Collects the monitoring data from `performance_schema.events_wa`
collect.perf_schema.file_events	5.6	Collects the monitoring data from `performance_schema.file_summ`
collect.perf_schema.file_instances	5.5	Collects the monitoring data from `performance_schema.file_summ`
collect.perf_schema.indexiowaits	5.6	Collects the monitoring data from `performance_schema.table_io_`
collect.perf_schema.tableiowaits	5.6	Collects the monitoring data from `performance_schema.table_io_`
collect.perf_schema.tablelocks	5.6	Collects the monitoring data from `performance_schema.table_loc`
collect.perf_schema.replication_group_members	5.7	Collects the monitoring data from `performance_schema.replicati`
collect.perf_schema.replication_group_member_stats	5.7	Collects the monitoring data from `performance_schema.replicati`
collect.perf_schema.replication_applier_status_by_worker	5.7	Collects the monitoring data from `performance_schema.replicati`
collect.slave_status	5.1	Collects the monitoring data from `SHOW`
collect.slave_hosts	5.1	Collects the monitoring data from `SHOW`
collect.heartbeat	5.1	Collects the monitoring data from heartb
collect.heartbeat.database	5.1	Database from where to collect heartbea

collect.heartbeat.table	5.1	Table from where to collect heartbeat da
collect.heartbeat.utc	5.1	Uses UTC for timestamps of the current `utc`). Default value: false.

Global configuration parameters

Item	Description
config.my-cnf	Path of `.my.cnf` file to read MySQL credentials from. Default value: `~/.my.cnf` .
log.level	Log level. Default value: info.
exporter.lock_wait_timeout	Sets a `lock_wait_timeout` (in seconds) on the connection to avoid long metadata locking. Default value: 2.
exporter.log_slow_filter	Adds a `log_slow_filter` to avoid slow query logging of scrapes. Note: not supported by Oracle MySQL.
web.listen-address	Web port listening address.
web.telemetry-path	Metric API path.
version	Prints the version information.

Heartbeat detection

If `collect.heartbeat` is enabled, mysqld_exporter will scrape replication delay measured by heartbeat mechanisms.

Consul Exporter Integration

Last updated：2024-08-07 22:02:11

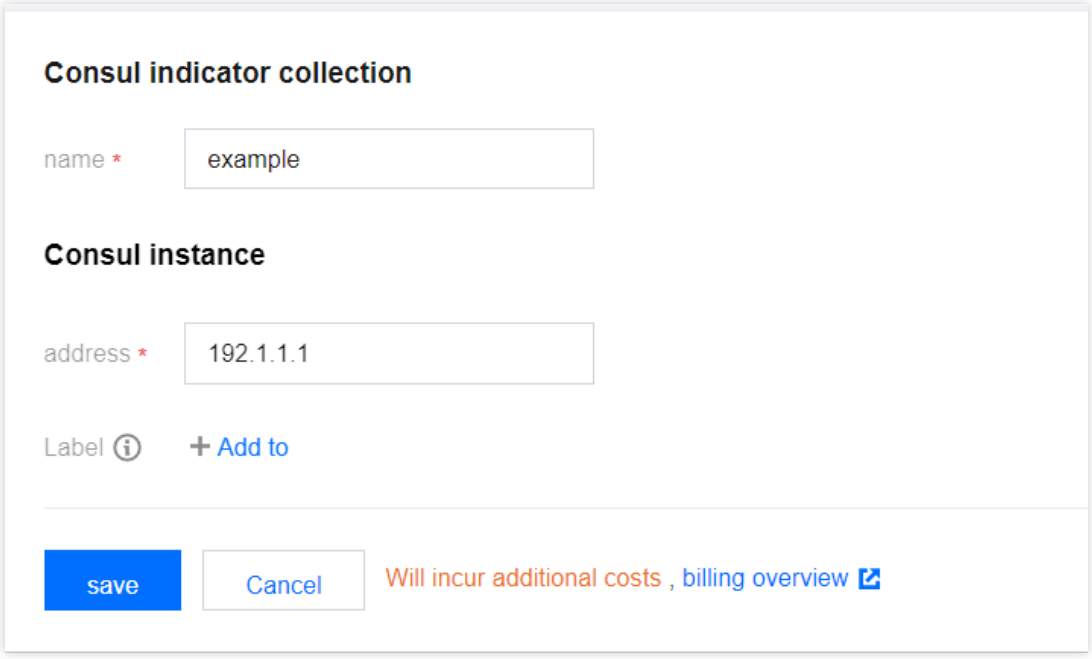
Overview

When using Consul, you need to monitor its running status to know whether it runs normally and troubleshoot its faults. TMP provides an exporter to monitor Consul and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to use TMP to monitor Consul.

Directions

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance.
3. Enter the instance details page and click **Integration Center**.
4. Select `Consul` in the Integration Center and click **Install** for integration.

Configuration description



Item	Description
Name	Unique integration name

Address	Address and port of the Consul instance to be collected
Label	Label with business meaning, which will be automatically added to Prometheus labels

Viewing monitoring information

You can clearly view the following monitoring metrics on the monitoring dashboard:

1. Status of Consul cluster nodes.
2. Status of services registered in Consul.

Memcached Exporter Integration

Last updated：2024-08-07 22:02:18

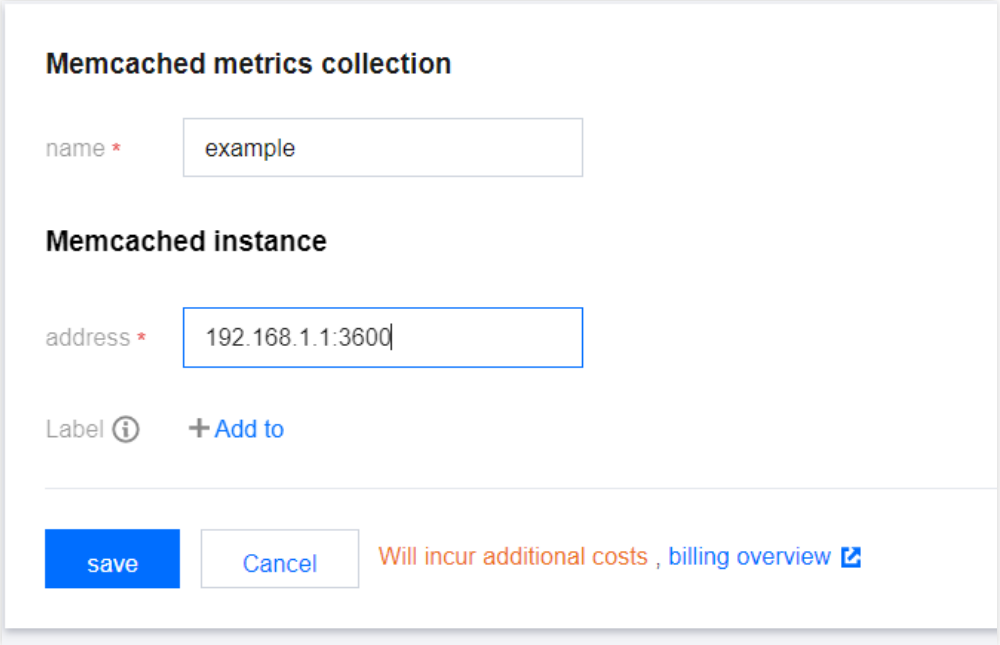
Overview

When using Memcached, you need to monitor its running status to know whether it runs normally and troubleshoot its faults. TMP provides an exporter to monitor Memcached and offers an out-of-the-box Grafana monitoring dashboard for it. This document describes how to use TMP to monitor Memcached.

Directions

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance.
3. Enter the instance details page and click **Integration Center**.
4. Select `Memcached` in the Integration Center and click **Install** for integration.

Configuration description



Item	Description
Name	Unique integration name

Address	Address and port of the Memcached instance to be collected
Label	Label with business meaning, which will be automatically added to Prometheus labels

Viewing monitoring information

You can clearly view the following monitoring metrics on the monitoring dashboard:

1. Memory utilization. The used memory and total memory are also displayed.
2. Current hit rate of `Get` commands. The hit and miss rates of `Get` commands during the service operation are also displayed.
3. Old data eviction rate and expired data reclaim rate of Memcached. The total numbers of evictions and reclaims during the service operation are also displayed.
4. Total amount of data stored in Memcached.
5. Number of bytes read from and written by the network.
6. Current number of open connections.
7. Ratio of `Get` and `Set` commands during the service operation.
8. Current generation rate of each command.

Integration with Other Exporters

Last updated：2024-08-07 22:02:28

Overview

TMP currently provides integration methods for common basic components and corresponding out-of-the-box monitoring dashboards. As TMP is compatible with the native Prometheus, you can also install other exporters available in the community.

Directions

If there is no integration method available for the basic component you want to use, you can integrate it as follows and customize a monitoring dashboard to meet your monitoring requirements:
1. Find your component in EXPORTERS AND INTEGRATIONS and integrate it as instructed.
2. Refer to the integration method for MySQL.

CVM Node Exporter

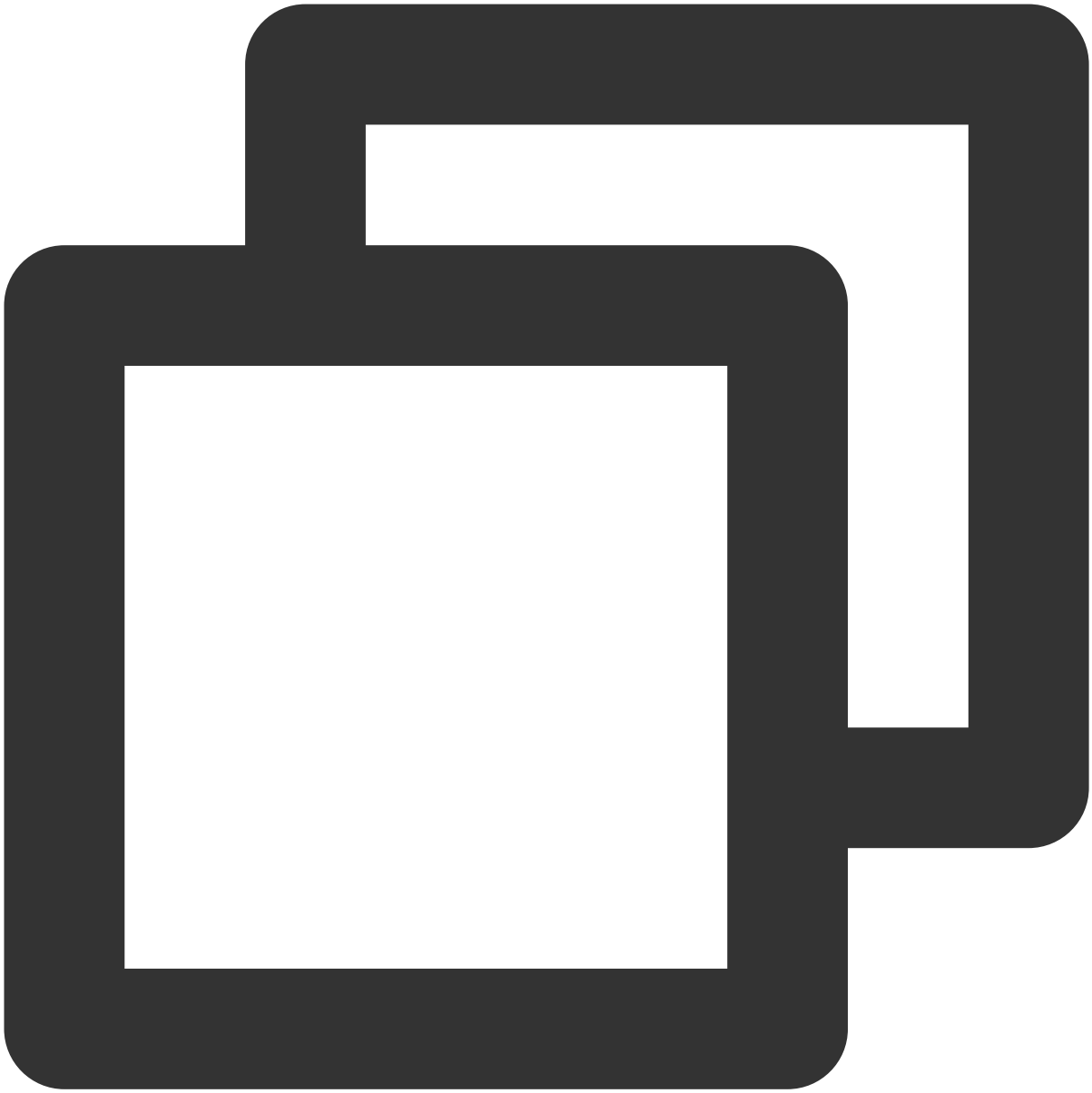
Last updated：2024-08-07 22:02:34

This document describes how to install Node Exporter to expose CVM basic metrics to TMP.

Directions

Step 1. Download and install Node Exporter

Download and install Node Exporter (used to collect basic metric data) in the target CVM instance. Click here or run the following command for download:



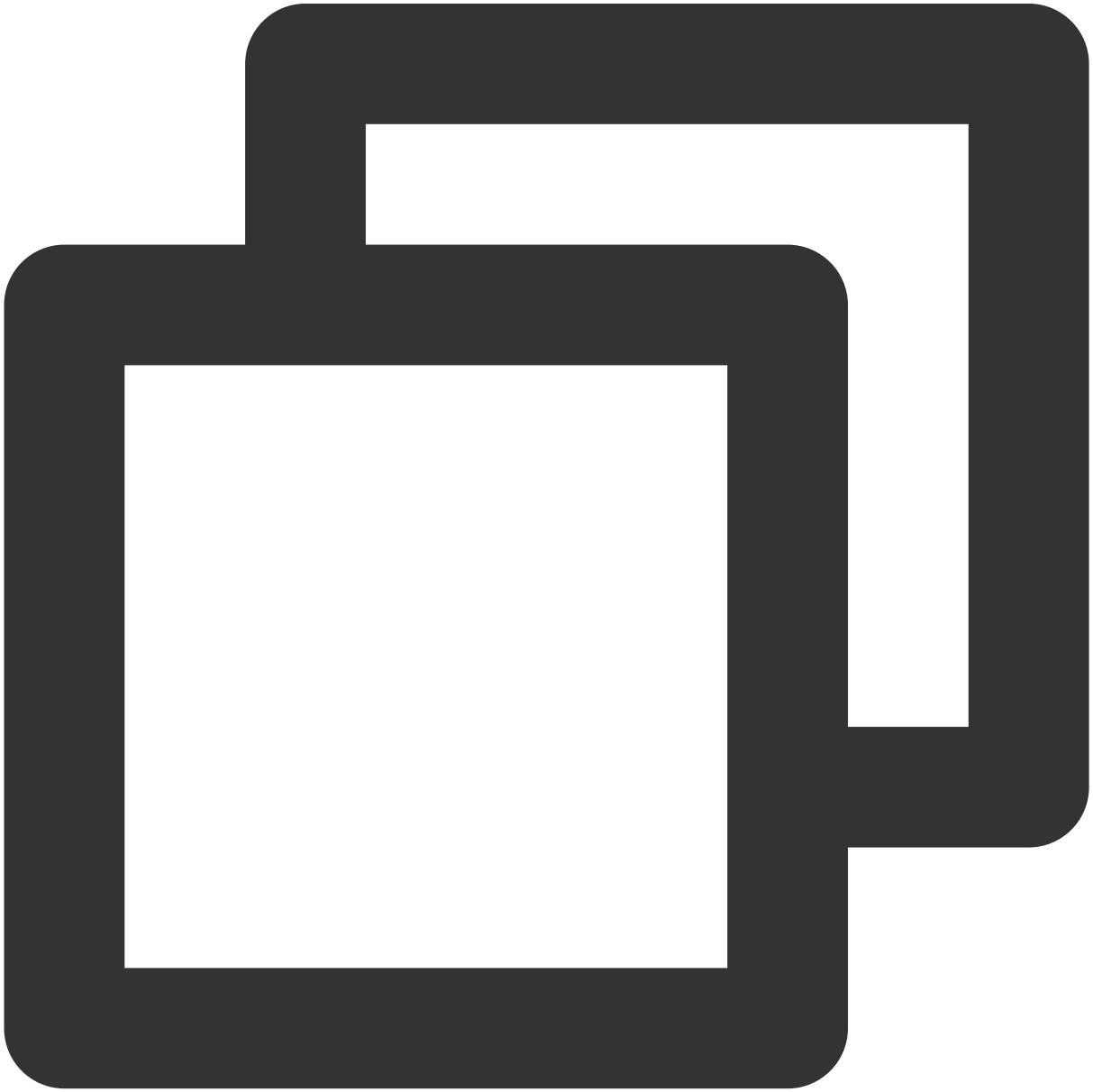
```
wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_expo
```

The file directory is as follows:

```
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ll
total 18080
-rw-r--r-- 1 3434 3434    11357 Aug  6  2021 LICENSE
-rwxr-xr-x 1 3434 3434 18494215 Aug  6  2021 node_exporter
-rw-r--r-- 1 3434 3434      463 Aug  6  2021 NOTICE
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ./node_exporter
```

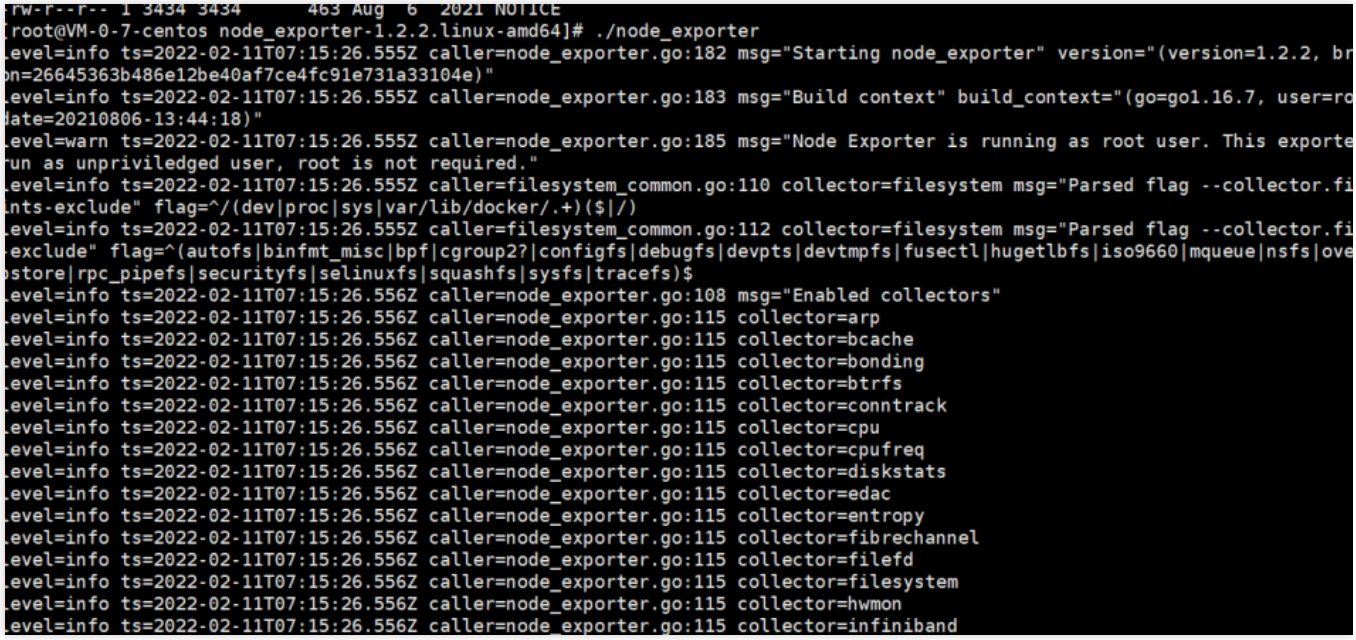
Step 2. Run Node Exporter to collect basic monitoring data

1. Go to the target folder and run Node Exporter.

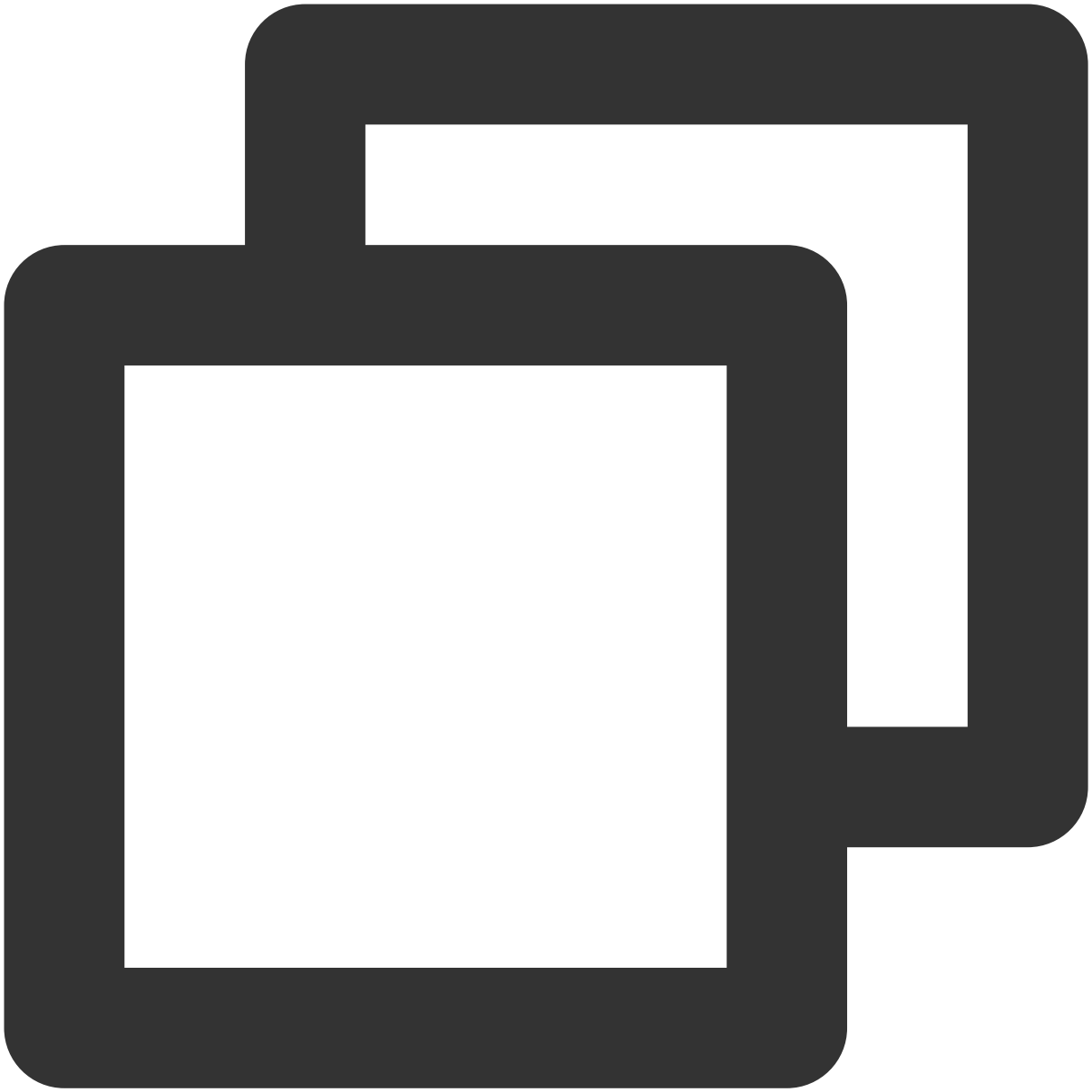


```
cd node_exporter-1.3.1.linux-amd64
./node_exporter
```

If the following result is displayed, basic monitoring data has been collected successfully.

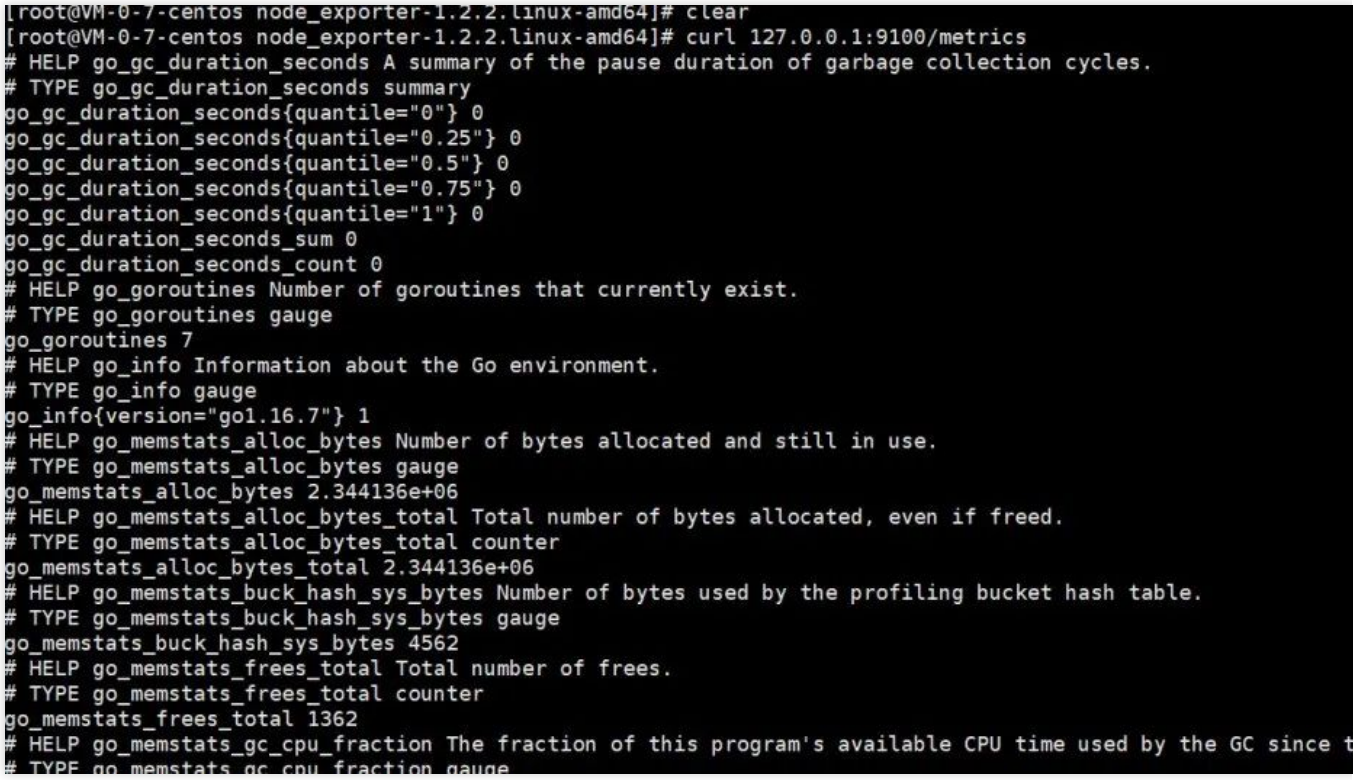


2. Run the following command to expose the basic monitoring data to port 9100:



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

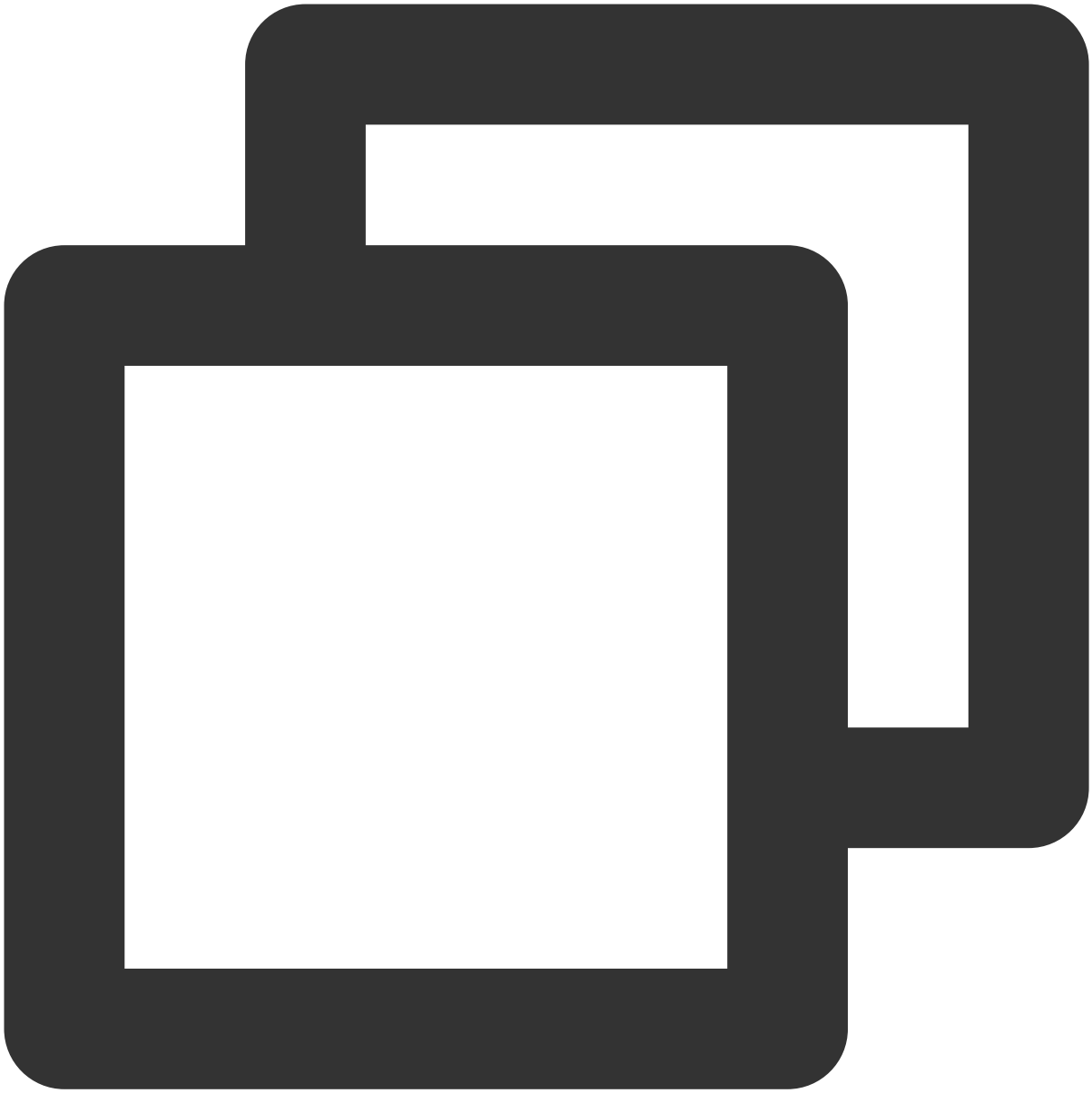
You can see the following metric monitoring data that is exposed after the command is executed.



Step 3. Configure the collection

Log in to the TMP console, select **Integration Center** > **CVM**, and configure the information in **Task Configuration** as prompted.

Below is a sample configuration of a scrape task:



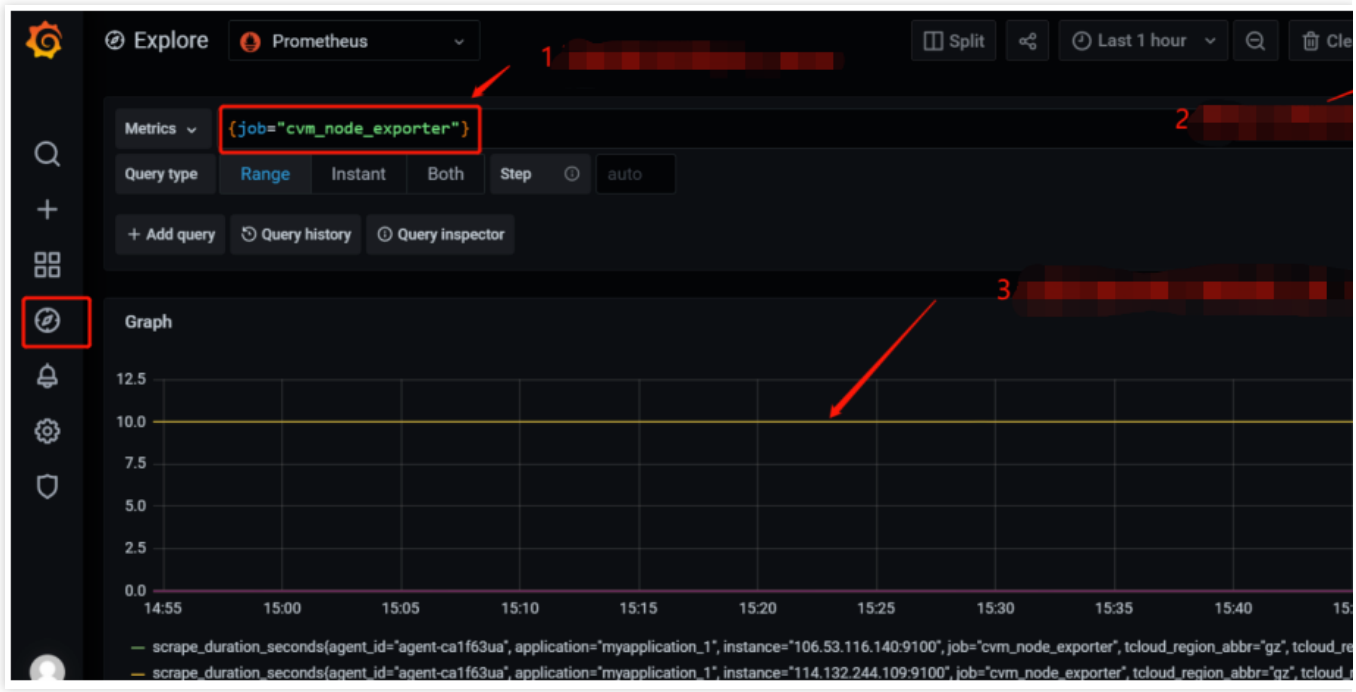
```
job_name: example-job-name
metrics_path: /metrics
cvm_sd_configs:
- region: ap-guangzhou
  ports:
  - 9100
  filters:
  - name: tag: Sample tag key
    values:
    - Sample tag value
relabel_configs:
```

```
- 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
```

Step 4. Check whether data is reported successfully

Log in to the TMP console and click the Grafana icon to enter Grafana.

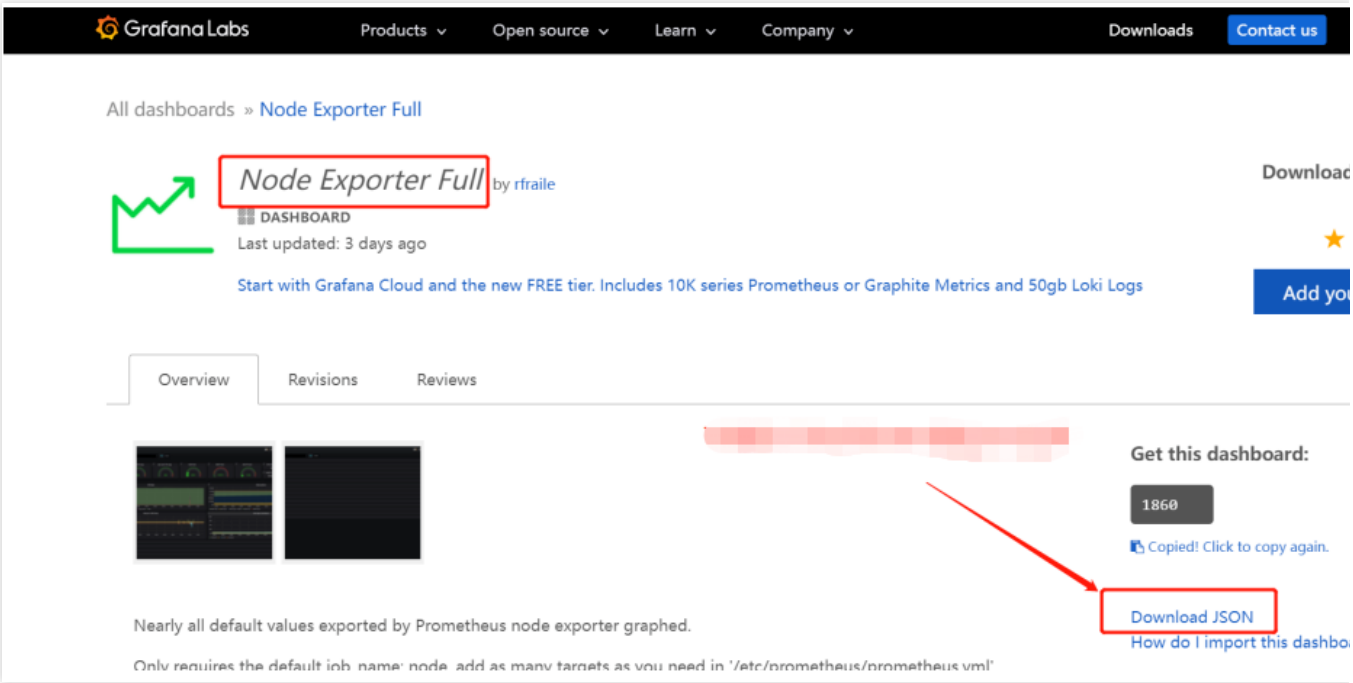
Search for `{job="cvm_node_exporter"}` in **Explore** to see whether there is data, and if so, data is reported successfully.



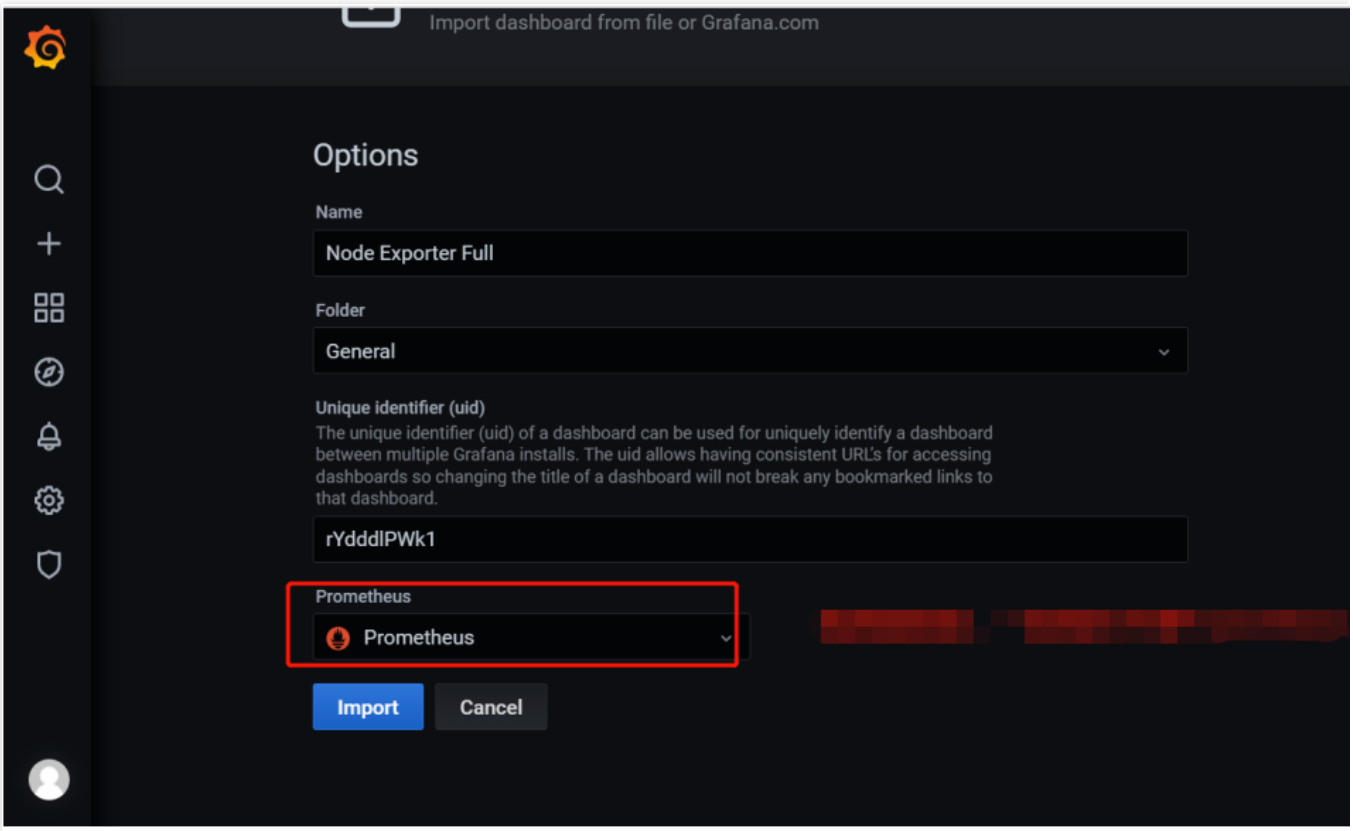
Step 5. Configure the dashboard

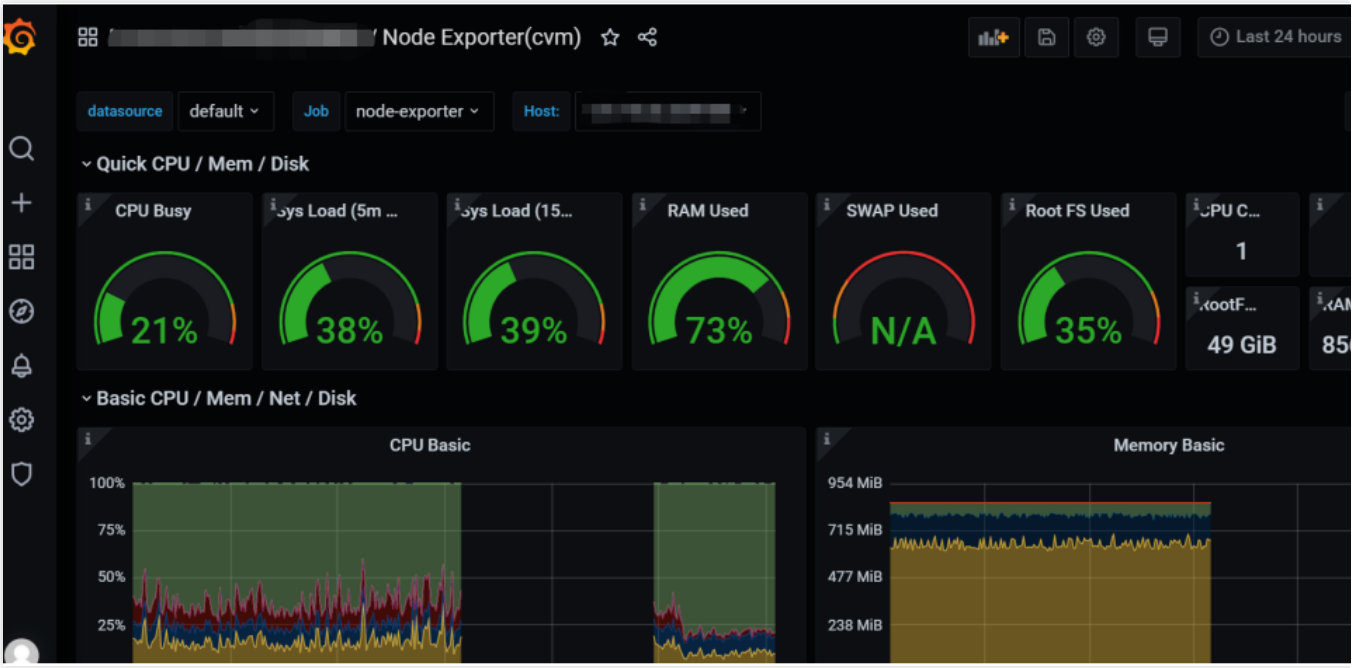
Every product has some existing JSON files that can be directly imported into the dashboard.

1. **Download a dashboard file**: Go to the **Dashboard** page, search for `node_exporter`, and select the latest dashboard for download.



2. **Import a JSON file into the dashboard**: Log in to the TMP console, select **Basic Info** > **Grafana Address** to enter Grafana. In the Grafana console, select **Create** > **Import** and upload the dashboard file in **Upload JSON file**.





Health Check

Last updated：2024-08-07 22:02:40

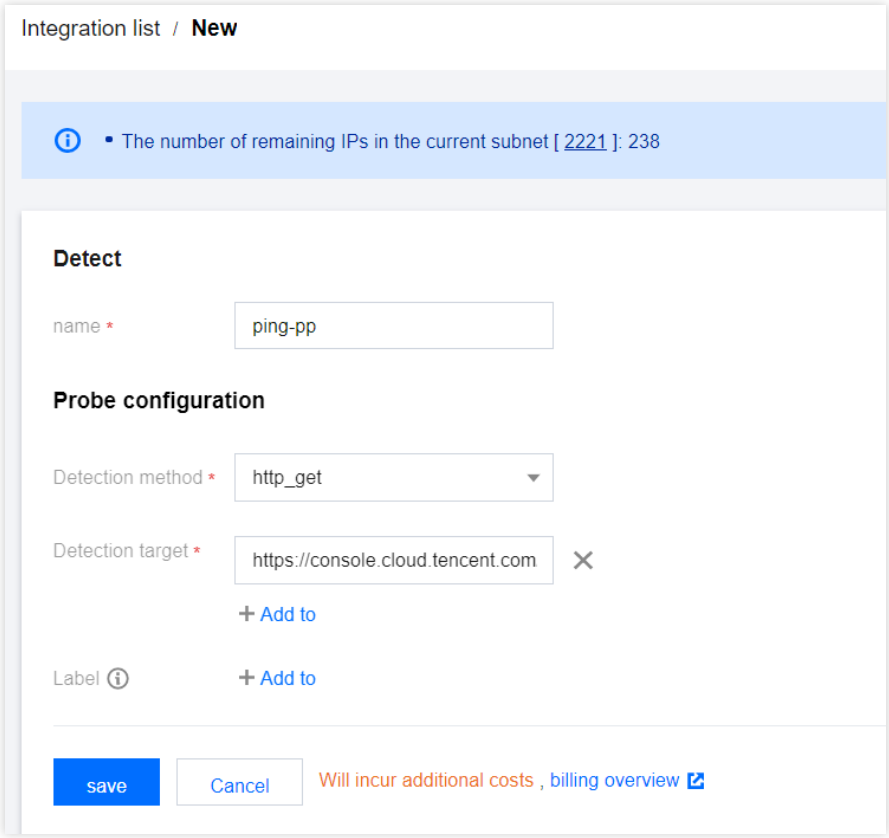
Overview

Health check detects the service connectivity on a regular basis to monitor the service health, helping you stay up to date with the service health in real time and promptly discover exceptions to improve the SLA.

Directions

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance.
3. Enter the instance details page and click **Integration Center**.
4. Select **Health Check** in **Integration Center** to configure the detection of the corresponding service.

Detection description



Parameter	Description
Name	Unique detection task name, which corresponds to the detection group on the Grafana monitoring dashboard
Detection Method	Currently, the following detection methods are supported: http_get http_post tcp ssh ping
Detection Target	Address of the service to be detected
Label	Label with business meaning, which will be automatically added to Prometheus labels

Viewing monitoring information

You can clearly view the following status on the monitoring dashboard:

1. Service access latency and health status.
2. Latency in each processing phase of service access.
3. Expiration time of certificate in case of HTTPS
4. Status of various detection types.


Datasouce
Prometheus
tencent-cloud
https://cloud.tencent.com/
https://cloud.tencent.com/
No data
https://cloud.tencent.com/ status
Status
UP
HTTP Status Code
200
HTTP Version
2
SSL
YES
SSL Expiry
12 months, 0 weeks, 3 days
min
max
avg
current
延时
20.03 ms
130.97 ms
47.32 ms
26.23 ms
connect
processing
resolve
tls
transfer
DNS Lookup
1.15 ms
36.34 ms

Cloud Monitoring

Last updated：2024-08-07 22:02:48

Overview

TMP collects, stores, and visualizes the basic monitoring data of Tencent Cloud products.

Directions

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance.
3. Enter the instance details page and click **Integration Center**.
4. Select **Cloud Monitoring** in the integration center. Define the integration name, configure the Exporter, and select the corresponding cloud product.

Note

Time offset: Select the time range. End time = current time - time offset.

The integrated monitoring data contains the tag data of the selected cloud products. Any Chinese or special characters in the tag will be filtered out.

Data collection frequency in this module: 1 minute.

Read Cloud-Hosted Prometheus Instance Data via Remote Read

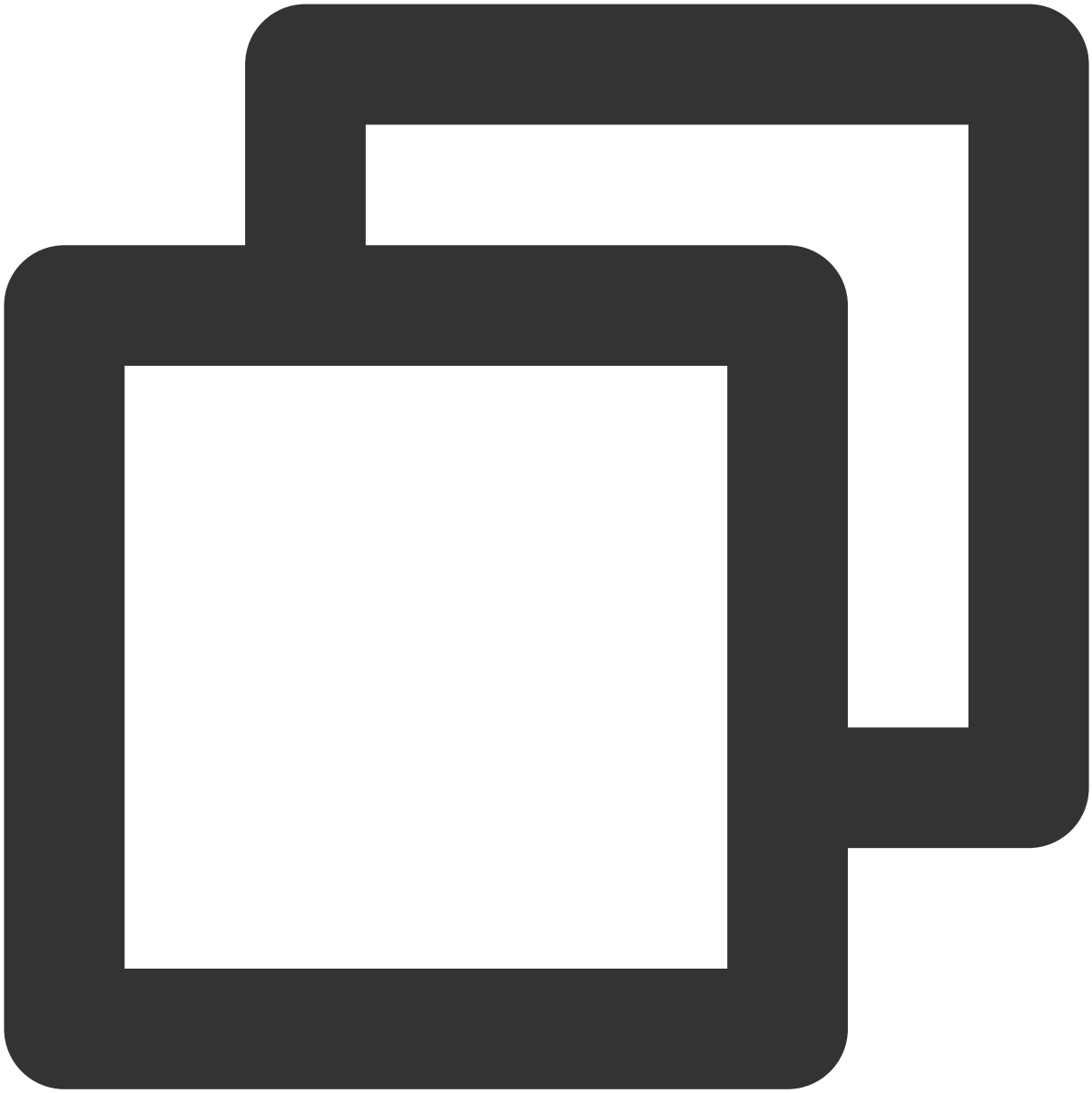
Last updated：2024-08-07 22:02:54

Overview

TMP provides the remote read API, which supports organizing a series of data sources of the Prometheus protocol into a single data source for query. This document describes how to use self-built Prometheus to read data from a cloud-managed TMP instance through the remote read API.

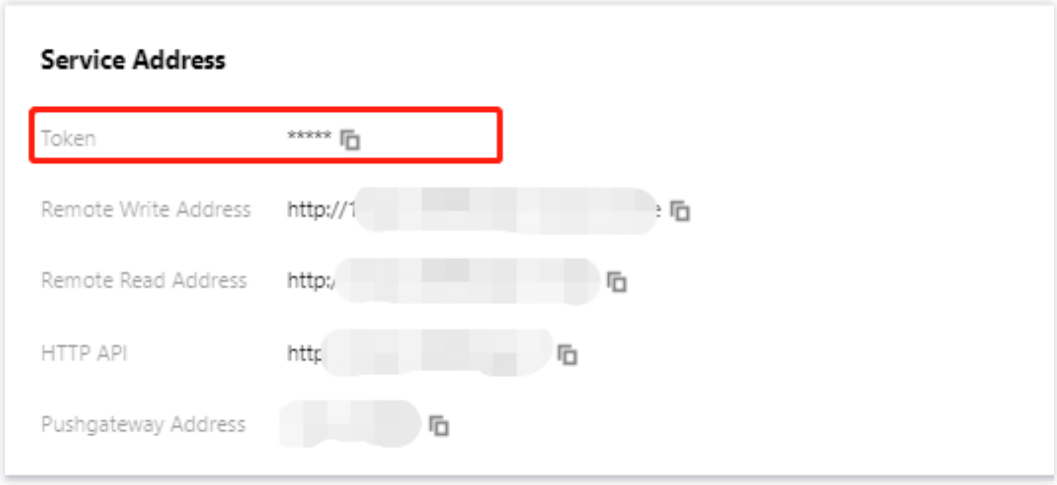
Remote Read Configuration

The recommended configuration for `prometheus.yml` is as follows:



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

It is recommended to use the Basic Auth method to access the cloud-managed TMP instance. The username is the account AppID and the password is the token obtained on **Basic Info** > **Service Address** in the Prometheus console.



Note

Configure `global:external_labels` carefully for TMP instances with remote read enabled:

As `external_labels` will be appended to the query condition of remote read, an inaccurate label may prevent you from querying the necessary data.

The `filter_external_labels: false` configuration item can avoid adding `external_labels` to the query condition (supported in v2.34 and later).

Avoid identical series:

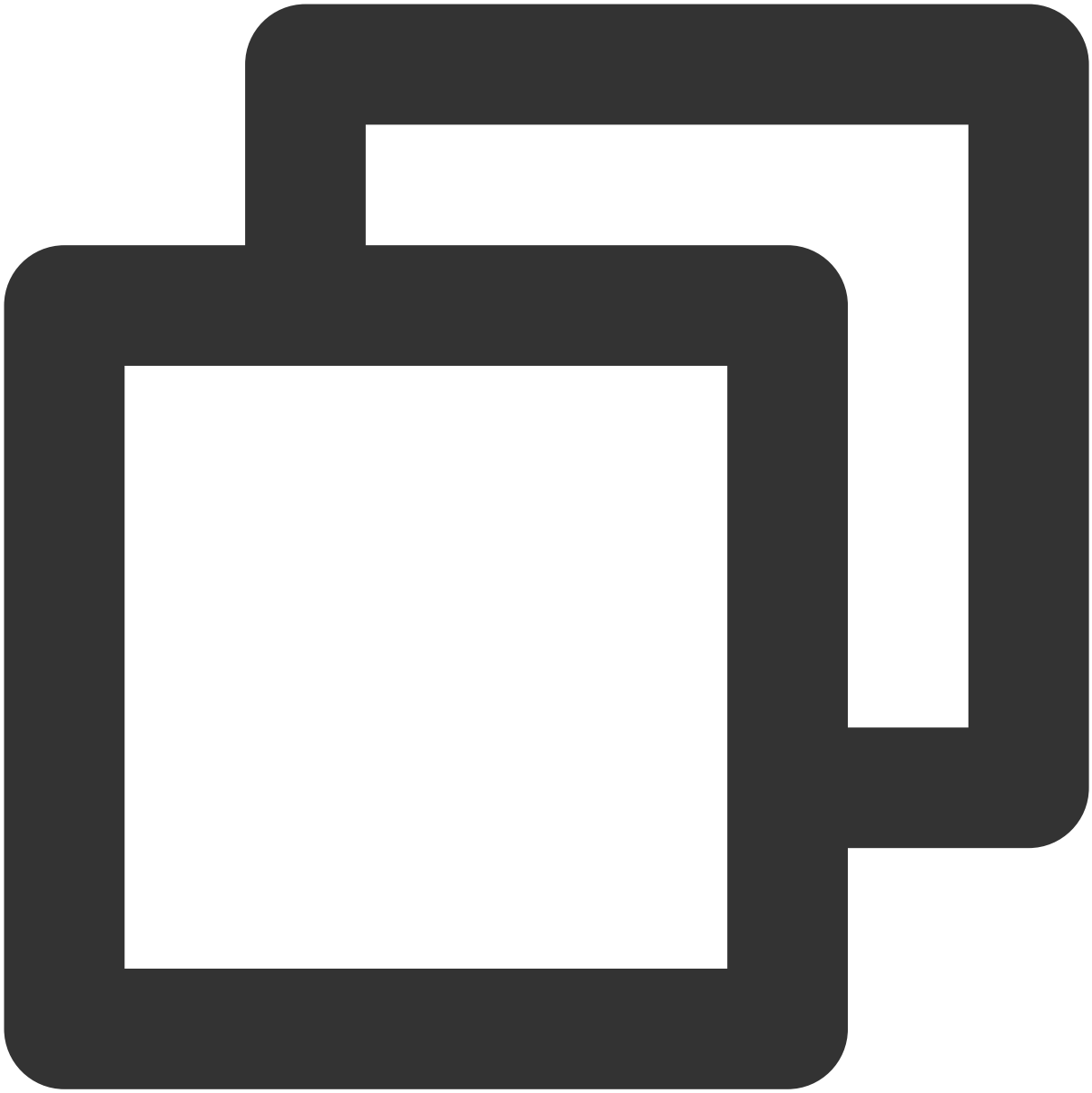
For two identical series, TMP will randomly select a series value at each time point to form a new series as the query result during query merging, which will lead to inaccurate query results.

Since there is no multi-copy redundant storage in the design concept of TMP, identical series will not be supported.

Remote Read Configuration Items

Note

The configuration items in `[]` are optional. This document shows Prometheus v2.40 configuration, and some configuration items may be missing in lower versions. For more information, see Prometheus official documentation.



```
# The API address of the target TMP instance for remote read
url: <string>

# Identify a unique remote read configuration name
[ name: <string> ]

# The PromQL must contain the following label filter conditions to perform remote r
required_matchers:
  [ <labelname>: <labelvalue> ... ]

# The timeout for remote read query
```

```
[ remote_timeout: <duration> | default = 1m ]

# Customize the headers attached to the remote read request. You can't overwrite th
headers:
  [ <string>: <string> ... ]

# Whether to perform remote read query in the time range with complete local data s
[ read_recent: <boolean> | default = false ]

# Add Authorization header to each remote read request, and choose password or pass
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Customize authorization header configuration
authorization:
  # Authentication type
  [ type: <string> | default: Bearer ]
  # Authentication key. You can choose credentials or credentials_file.
  [ credentials: <secret> ]
  # Get the key from the file
  [ credentials_file: <filename> ]

# OAuth2.0 authentication, which cannot be used with basic_auth authorization at th
oauth2:
  [ <oauth2> ]

# TLS configuration
tls_config:
  [ <tls_config> ]

# Proxy URL
[ proxy_url: <string> ]

# Query whether the request accepts 3XX redirection
[ follow_redirects: <boolean> | default = true ]

# Whether to enable HTTP2
[ enable_http2: <bool> | default: true ]

# Whether to append `external_labels` for remote read
[ filter_external_labels: <boolean> | default = true ]
```

Agent Self-Service Access

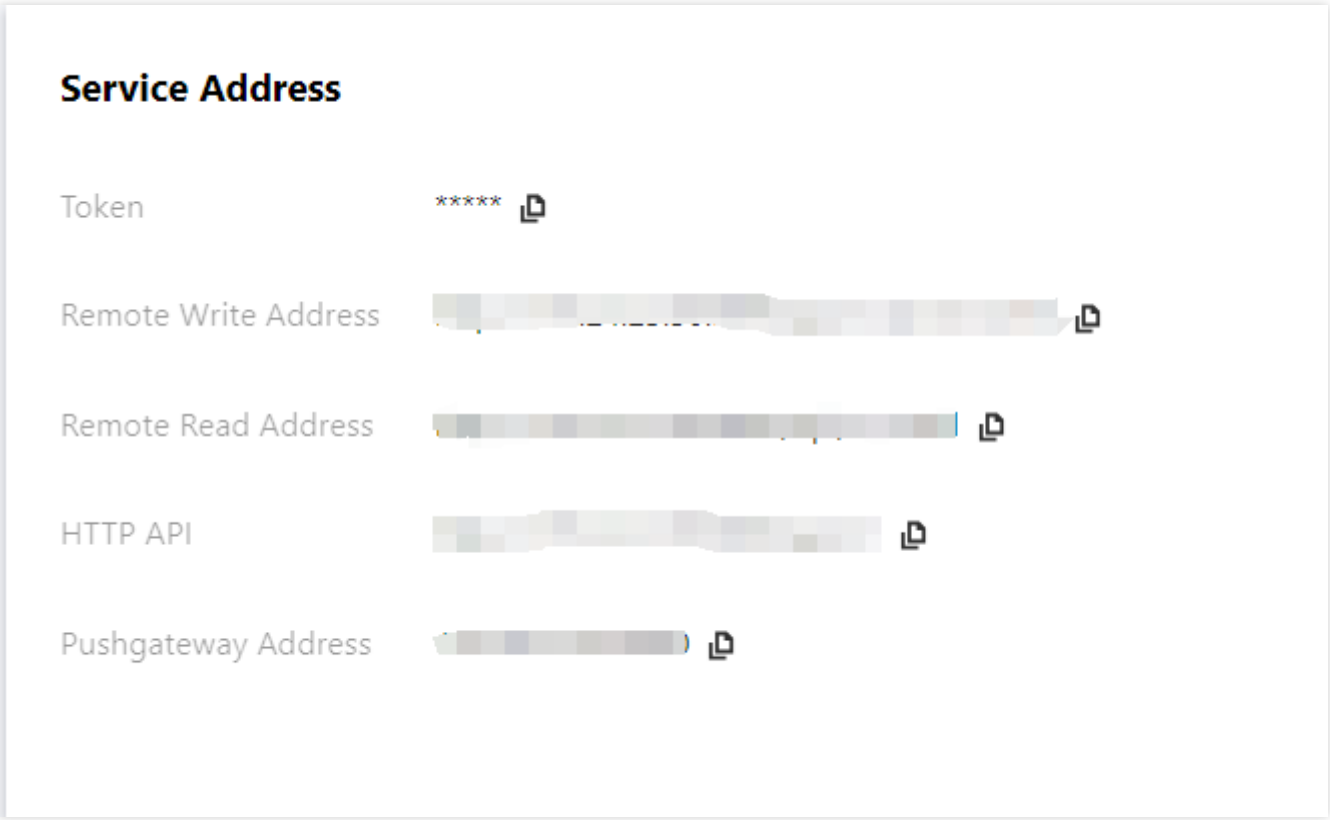
Last updated：2024-08-15 16:10:55

Application Scenario

To collect services on self-built IDC, deploy Agent and manage collection configurations, and report monitoring data to the cloud TMP. For cloud services, we recommend using Integration Center, which will manage Agent, offering automated integration for multiple middlewares and scraping tasks.

Obtaining Prometheus Instance Access Configuration

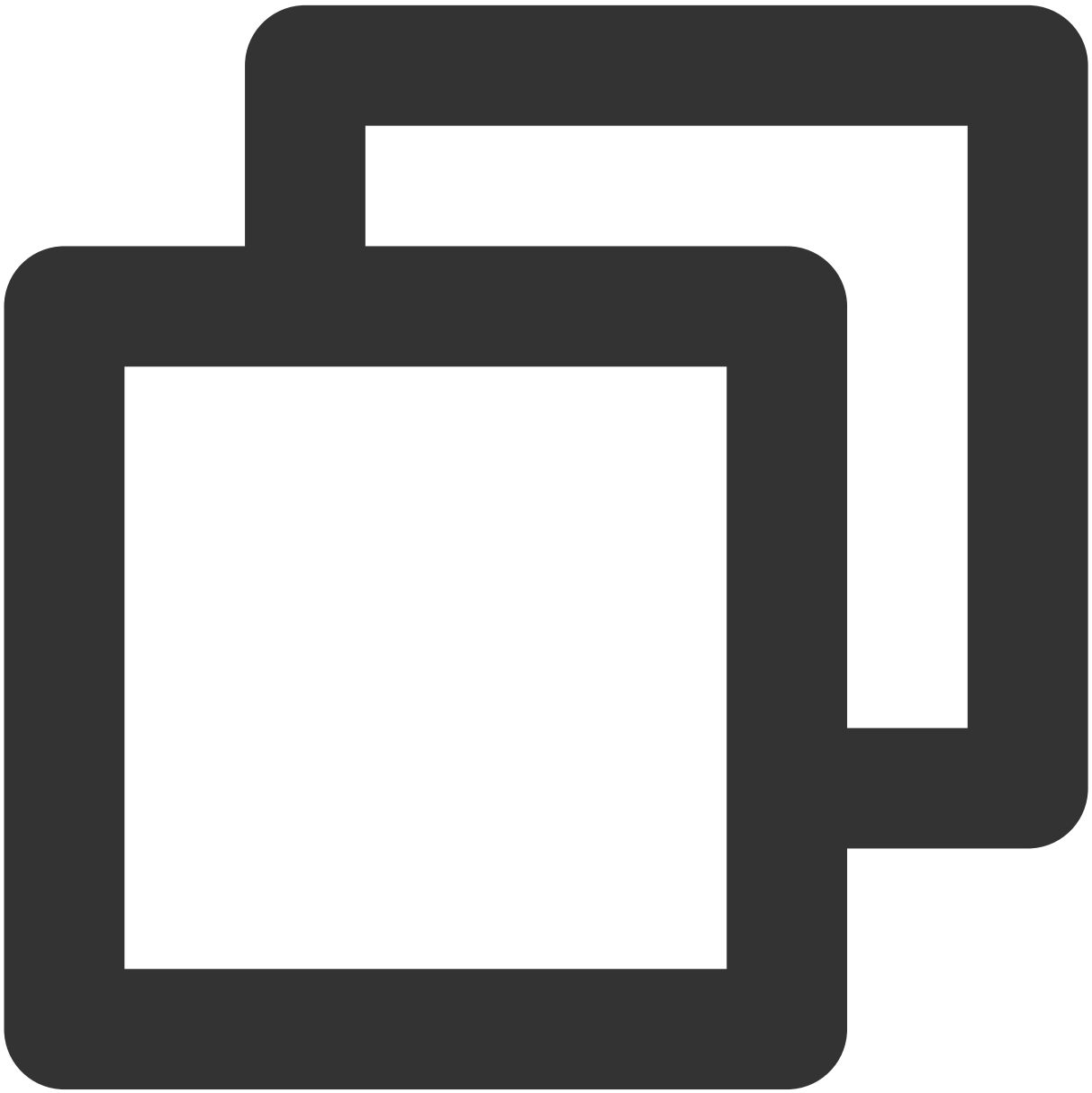
1. Go to Prometheus Monitoring Console, select the corresponding instance ID/Name, and on the **Basic Info** > **Service Address** page, obtain the Remote Write address and Token.



2. Obtain APPID on the Account Information page.

Confirming the Network Environment and Connectivity with Cloud Instances

Based on the acquired RemoteWrite address, execute the following command. If the network is connected, the returned information will include `401 Unauthorized` .



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

Installing and Starting vmagent

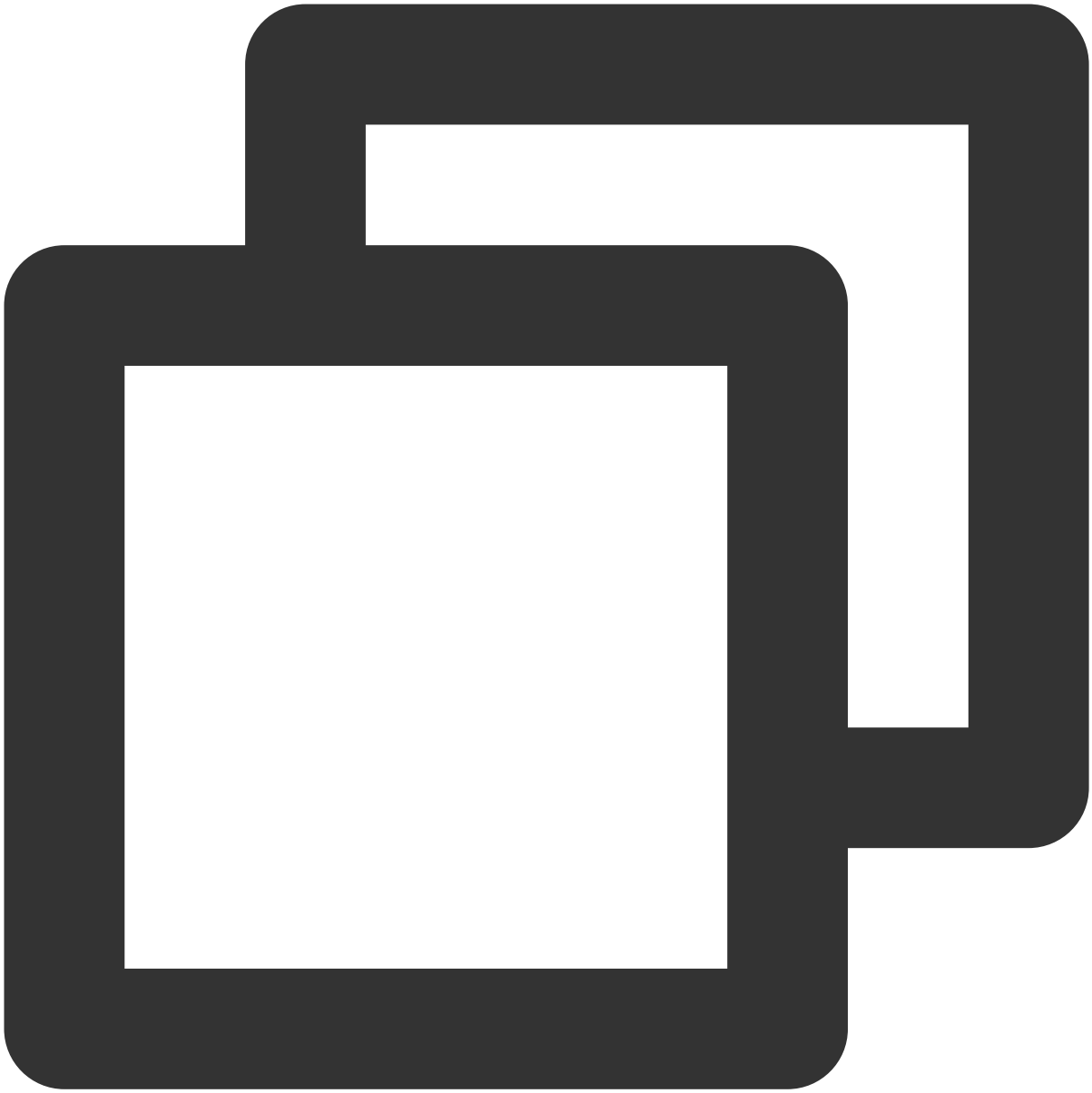
vmagent uses fewer resources and is widely used due to its compatibility with Prometheus collection configuration and Remote Write protocol. This document only describes common startup options for vmagent, managed through Systemd or Docker. For more detailed information, please see the official documentation.

Common Startup Options

-promscrape.noStaleMarkers: If the collection target disappears, a stale marker for all associated metrics is generated and written to remote storage by default. Setting this option disables this behavior and can reduce memory usage.
-loggerTimezone: The time zone for the time in logs, for example, `Asia/Shanghai, Europe/Berlin` or `Local` (UTC by default).
-remoteWrite.tmpDataPath: The file path for temporary data storage to be written after collection.
-remoteWrite.url: The URL where data is written to remote storage.
-remoteWrite.basicAuth.username: Remote storage -remoteWrite.url corresponding basic auth username.
-remoteWrite.basicAuth.password: Remote storage -remoteWrite.url corresponding basic auth password.
-promscrape.config: Path of the collection configuration, which can be a file path or HTTP URL. For more details, please see Reference Documentation.
-promscrape.configCheckInterval: Interval for checking the -promscrape.config configuration changes. For configuration updates, please see Reference Documentation.

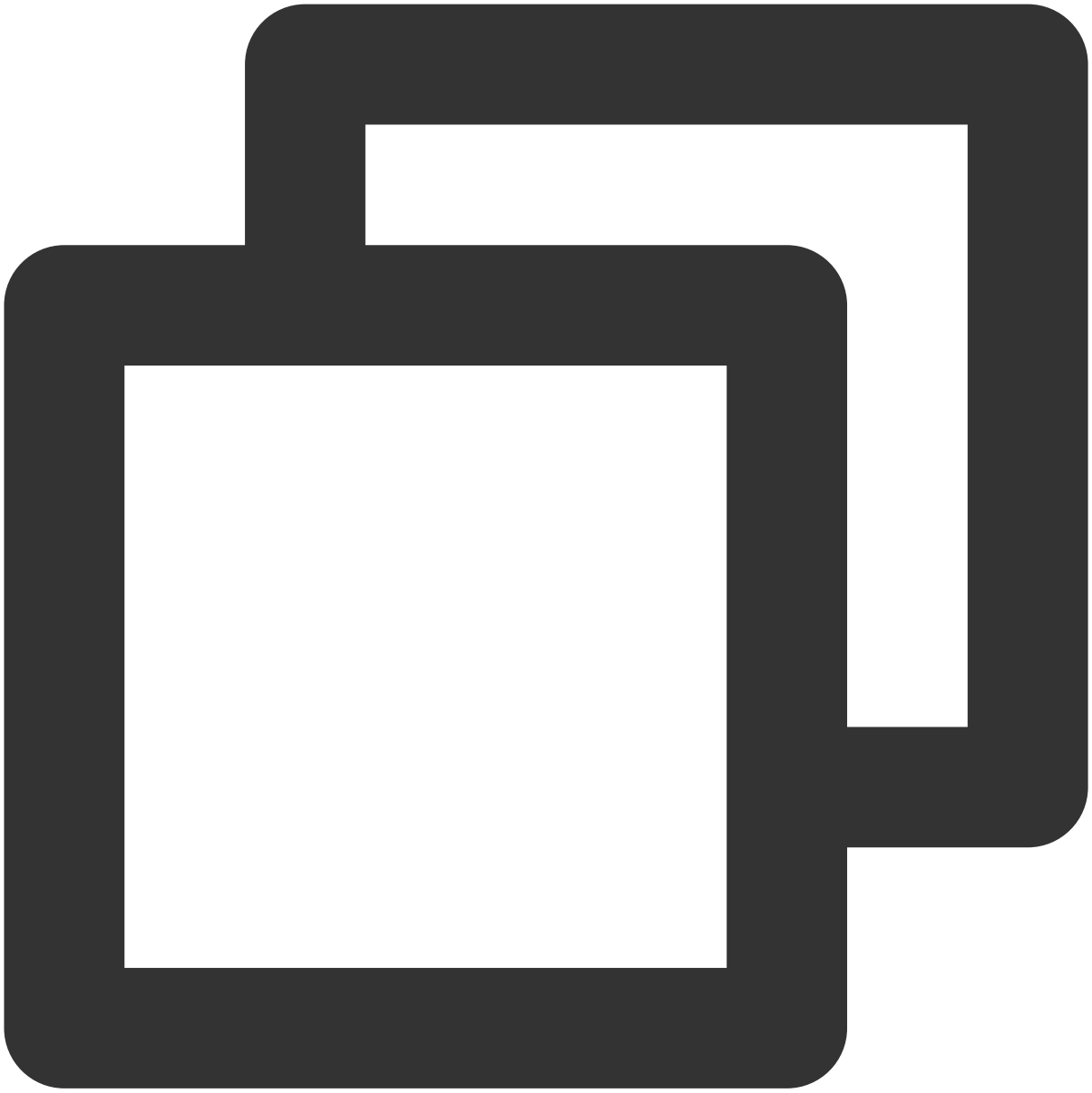
Managing via Docker

1. On the vmagent Release Page, select the image version. It is recommended to use latest.
2. Replace the Prometheus instance information in the script and start vmagent.



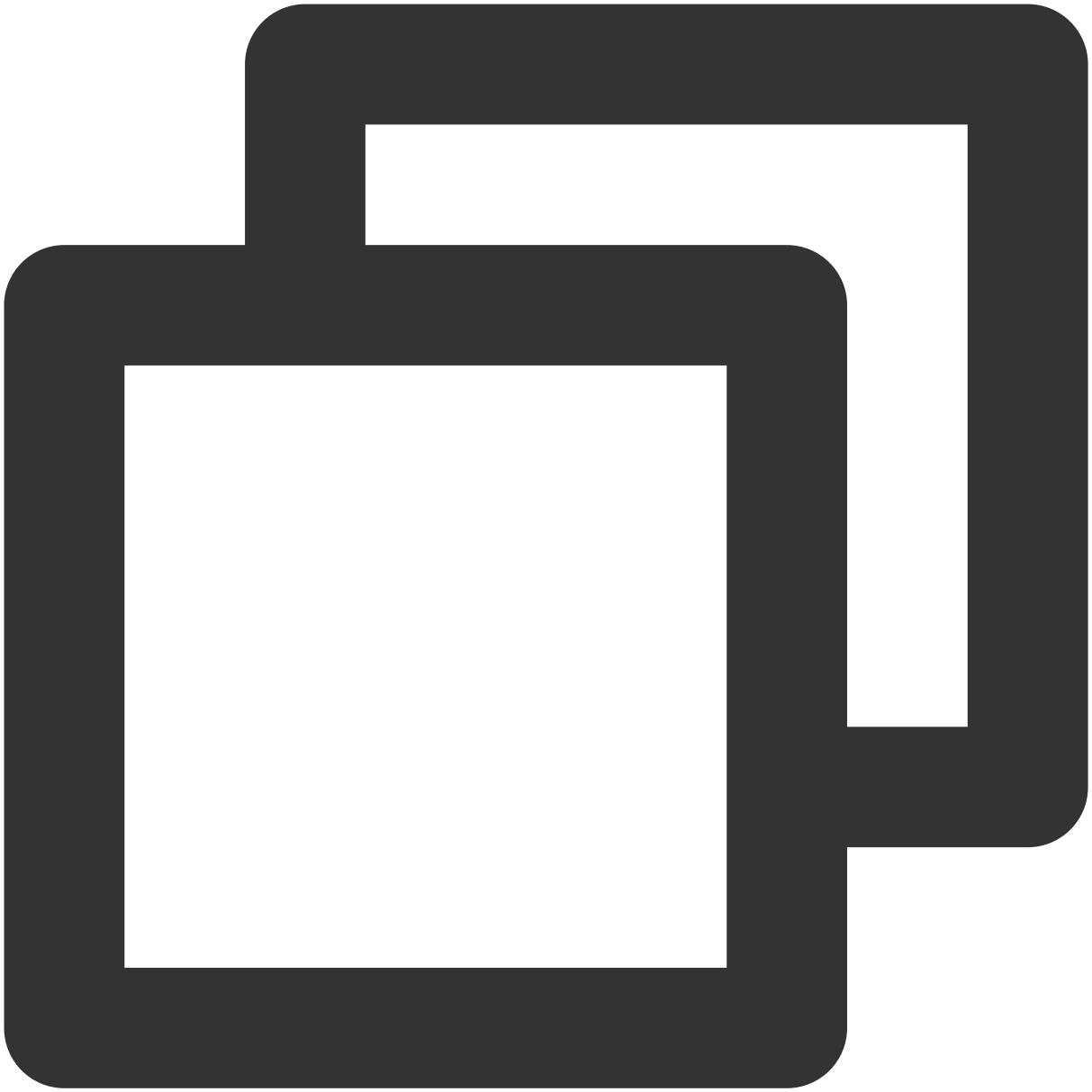
```
mkdir /etc/prometheus
touch /etc/prometheus/scrape-config.yaml
docker run -d --name vmagent --restart always --net host -v /etc/prometheus:/etc/pr
-promscrape.noStaleMarkers \\
-loggerTimezone=Local \\
-remoteWrite.url="${RemoteWriteURL}" \\
-remoteWrite.basicAuth.username="${APPID}" \\
-remoteWrite.basicAuth.password='${Token}'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

3. View vmagent logs



```
docker ps
docker logs vmagent
```

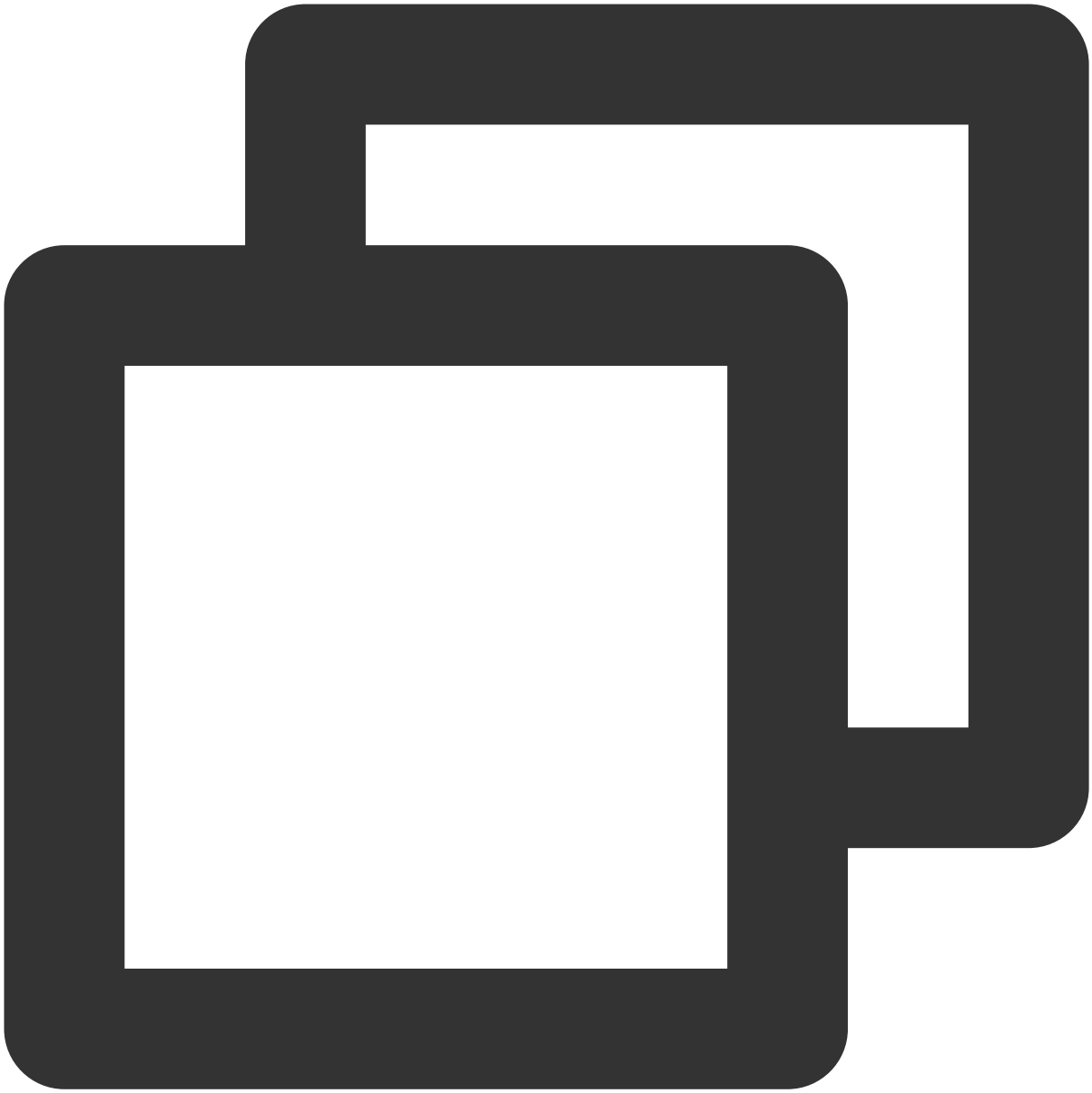
If it starts normally, executing the following command will return `OK` .



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

Managing via Systemd

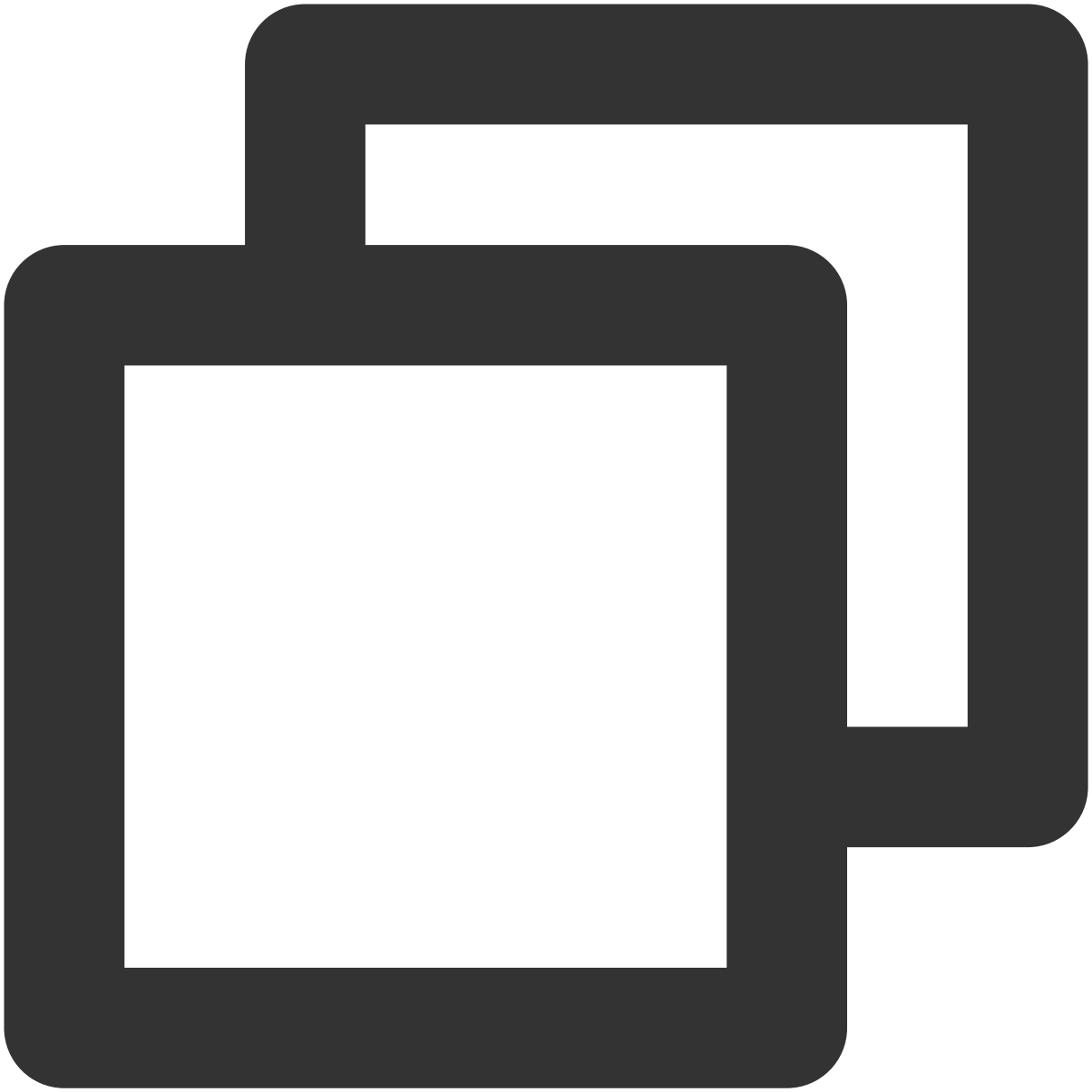
1. On the vmagent Release page, download the corresponding vmutils-* compressed package according to your operating system and CPU architecture, and decompress it.
2. Replace the access information of the Prometheus instance in the script and start vmagent.



```
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 \\
```

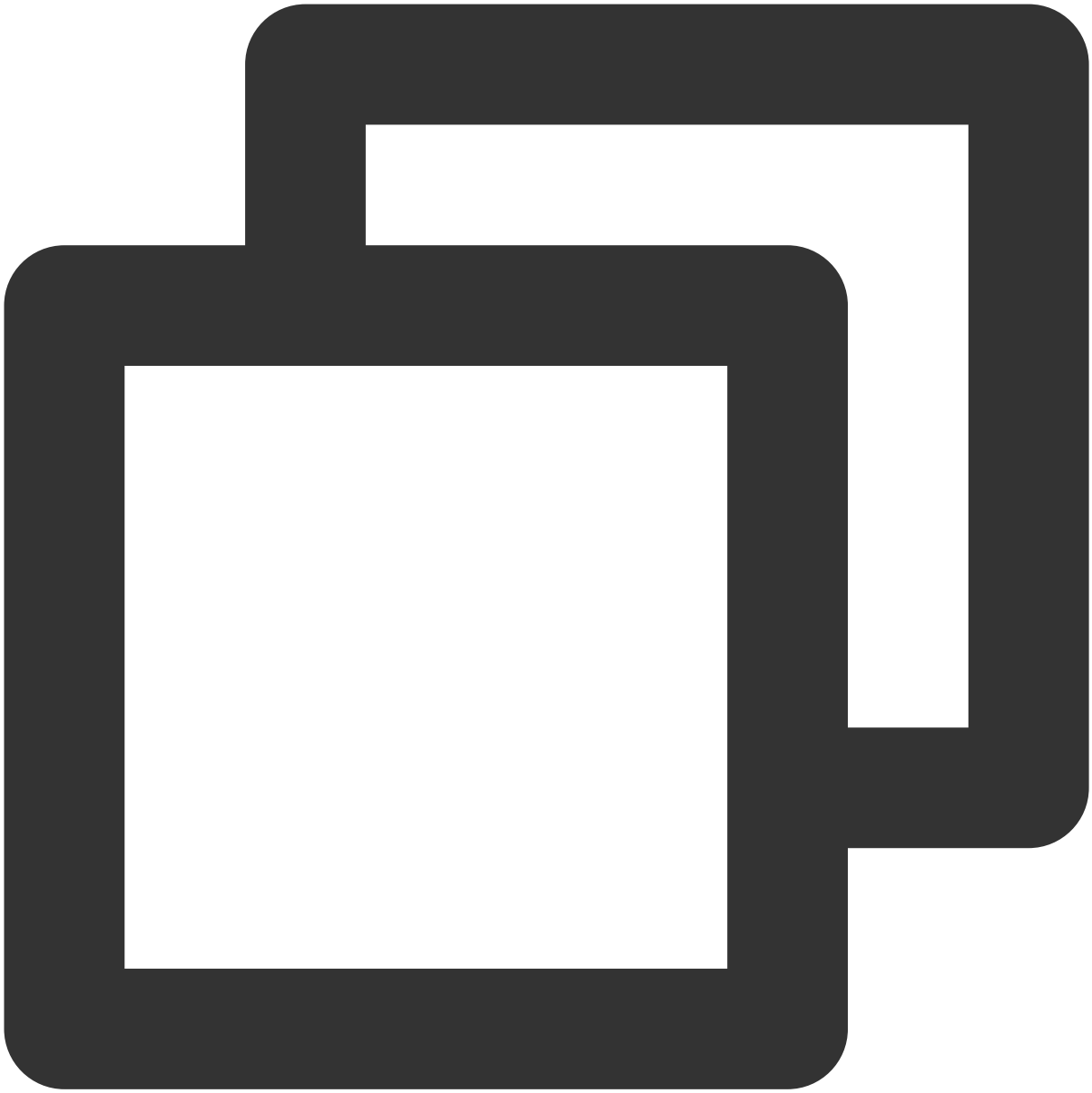
```
-loggerTimezone=Local   \\
-remoteWrite.url="${RemoteWriteURL}" \\
-remoteWrite.basicAuth.username="${APPID}" \\
-remoteWrite.basicAuth.password="${Token}" \\
-remoteWrite.tmpDataPath=/var/lib/vmagent \\
-promscrape.config=/etc/prometheus/scrape-config.yaml \\
-promscrape.configCheckInterval=5s
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
systemctl enable vmagent
systemctl start vmagent
sleep 3
systemctl status vmagent
```

3. View logs



```
journalctl -u vmagent
```

If it starts normally, executing the following command will return `OK` .

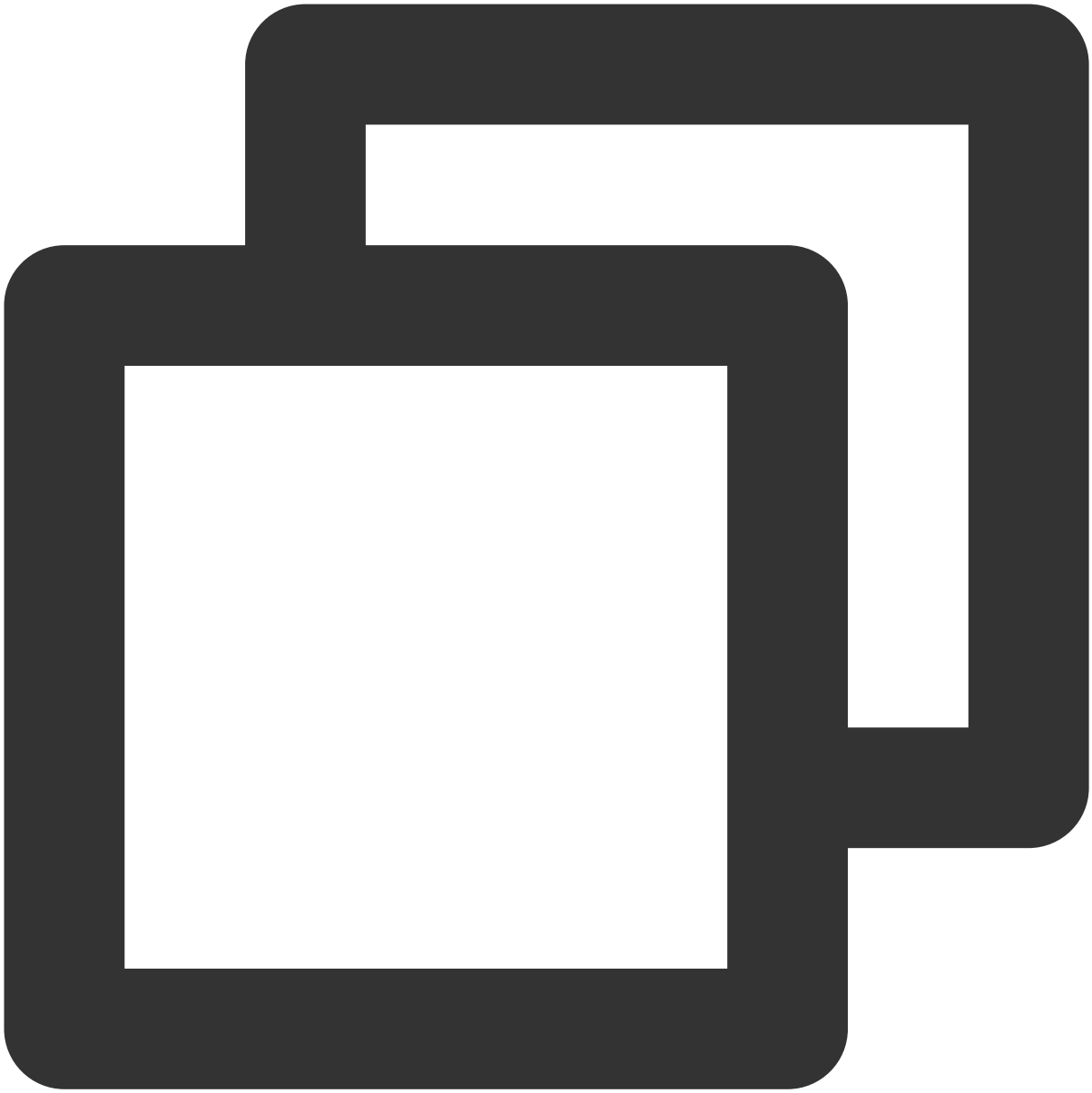


```
curl localhost:8429/health
```

Managing the Configuration

Modifying the Configuration File

Edit the collection configuration file `/etc/prometheus/scrape-config.yaml` to add/update/delete collection tasks. For Prometheus collection task configuration, see Official Documentation.

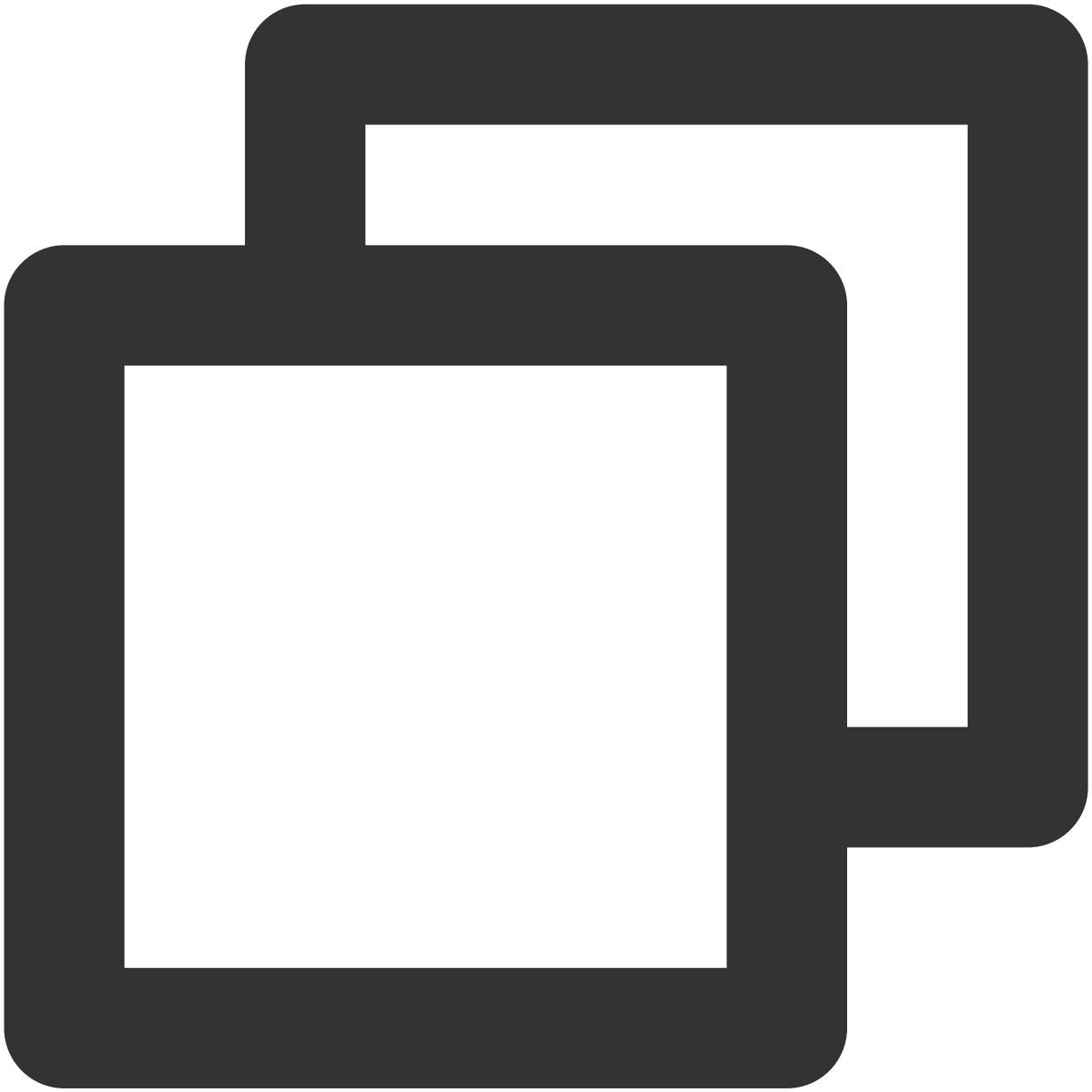


```
global:
   scrape_interval: 30s
scrape_configs:
   - job_name: agent-monitor
     static_configs:
        - targets:
             - localhost:8429
```

After the configuration is modified, it will only take effect after the time set by the option `-promscrape.configCheckInterval`.

Viewing Monitoring Target Information

Execute the following command to view the collection target and check whether the configuration is effective and meets expectations.



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

Instructions for Installing Components in the TKE Cluster

Last updated：2024-07-23 18:11:09

Overview

This document describes the features, use permissions, and resource consumption of various components installed in the user's TKE cluster during the TKE Integration process of TMP.

proxy-agent

Component Overview

The TKE cluster has independent network environment. Therefore, the proxy-agent is deployed within the cluster to provide access proxies for collection components outside the cluster. On one hand, external collection components discover resources within the cluster through the proxy-agent service; on the other hand, they scrape metrics through the proxy-agent and write them to the time series storage of the Prometheus instance.

Resource Objects Deployed in the Cluster

Namespace	Kubernetes Object Name	Type	Resource Amount	Description
<Prometheus instance ID>	proxy-agent	Deployment	0.25C256Mi*2	Collection proxy
<Prometheus instance ID>	<Prometheus instance ID>	ServiceAccount	-	Permission carrier
-	<Prometheus instance ID>	ClusterRole	-	Collection permissions related
-	<Prometheus instance ID>-crb	ClusterRoleBinding	-	Collection permissions related

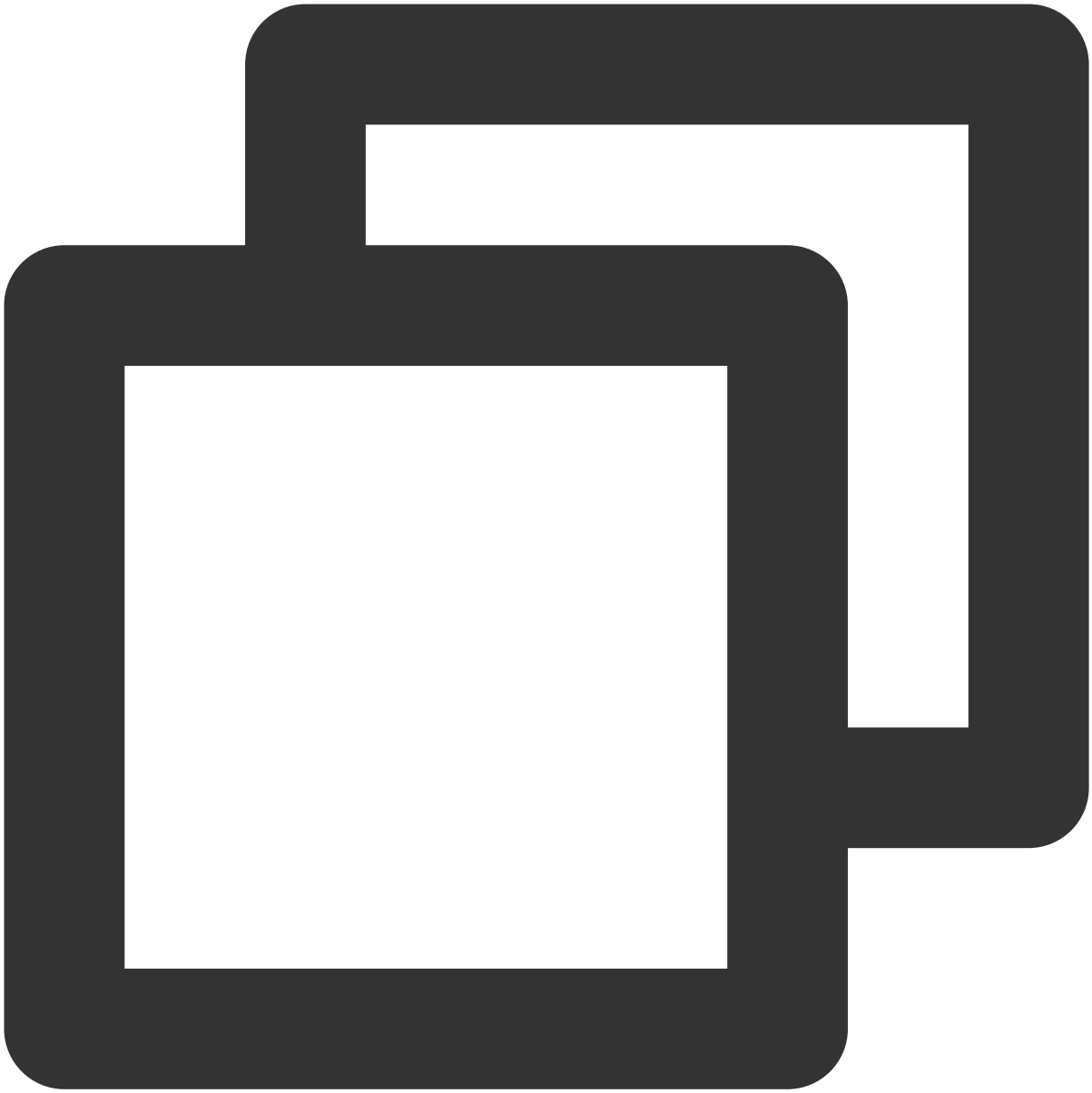
Component Permission Description

Permission Scenarios

Feature	Involved Objects	Involve

		Operati Permiss
Collection configuration management	scrapeconfigs,servicemonitors,podmonitors,probes,configmaps,secrets,namespaces	get/list/
Service discovery	services,endpoints,nodes,pods,ingresses	get/list/
Scraping some system component metrics	nodes/metrics,nodes/proxy,pods/proxy	get/list/
Scraping metrics with RBAC authentication	/metrics,/metrics/cadvisor	get

Permission Definition



```
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
  - 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
```

tke-kube-state-metrics

Component Overview

tke-kube-state-metrics uses the open-source component kube-state-metrics, listens to the cluster's API server, and generates status metrics for various objects within the cluster.

Resource Objects Deployed in the Cluster

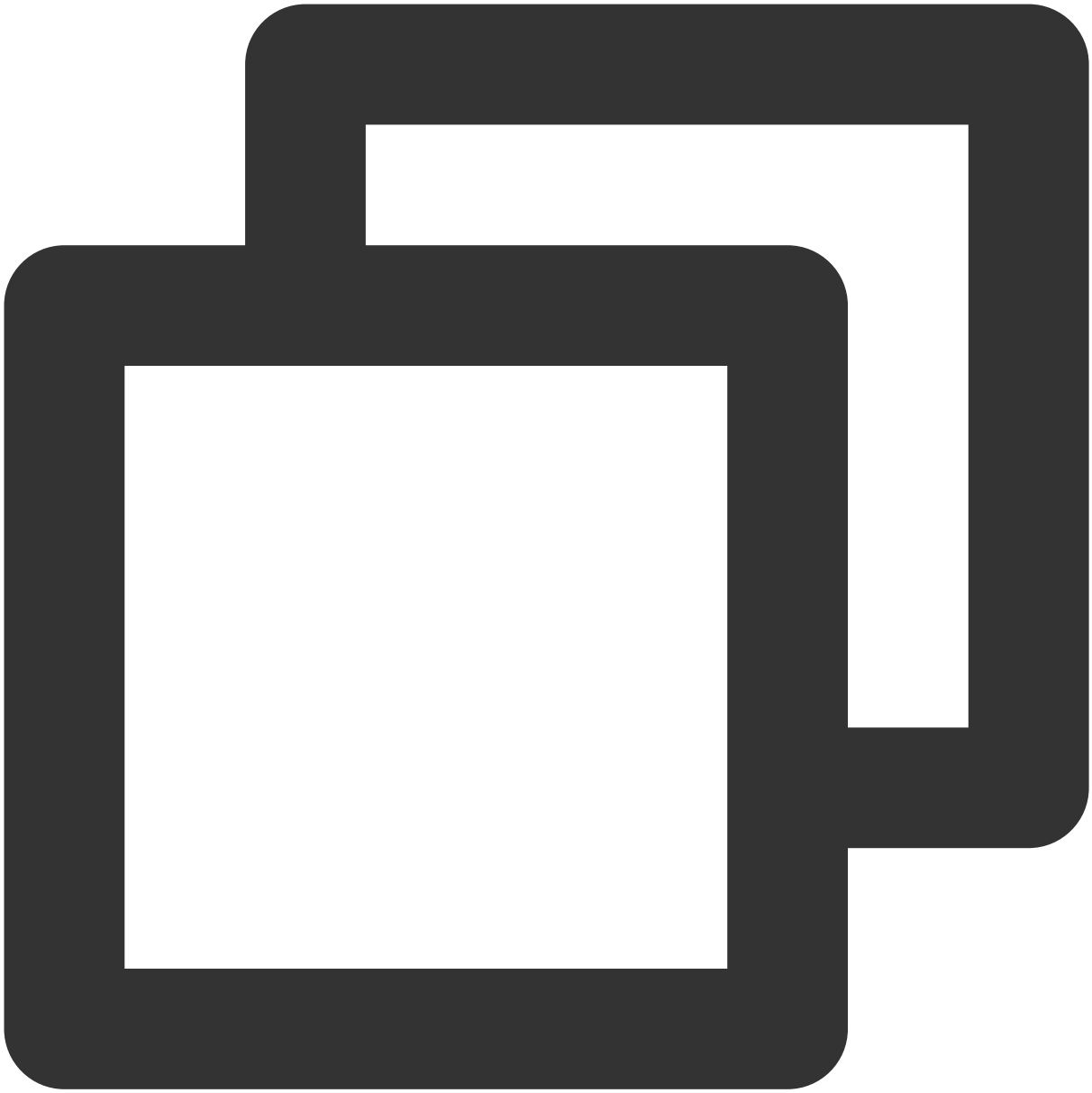
Namespace	Kubernetes Object Name	Type	Resource Amount	Description
kube-system	tke-kube-state-metrics	Statefulset	0.5C512Mi	Collection program
kube-system	tke-kube-state-metrics	ServiceAccount	-	Permission carrier
-	tke-kube-state-metrics	ClusterRole	-	Collection permissions related
-	tke-kube-state-metrics	ClusterRoleBinding	-	Collection permissions related
kube-system	tke-kube-state-metrics	Service	-	Collection agent corresponding service, for service discovery use
kube-system	tke-kube-state-metrics	ServiceMonitor	-	Collection configuration
kube-system	tke-kube-state-metrics	Role	-	Shard collection permission related
kube-system	tke-kube-state-metrics	RoleBinding	-	Shard collection permission related

Component Permission Description

Permission Scenarios

Feature	Involved Objects	Involved Operation Permissions
Listening to the status of various resources in the cluster	Most Kubernetes resources	list/watch
Get the shard number of the collection pod	statefulsets, pods	get

Permission Definition



```
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

Component Overview

tke-node-exporter uses the open-source project node_exporter, deployed on each node in the cluster to collect hardware and Unix-like operating system metrics.

Resources Deployed in the Cluster

Namespace	Kubernetes	Type	Resource Amount	Description

	Object Name			
kube-system	tke-node-exporter	DaemonSet	0.1C180Mi*node amount	Collection program
kube-system	tke-node-exporter	Service	-	Collection program corresponding service, for service discovery use
kube-system	tke-node-exporter	ServiceMonitor	-	Collection configuration

Component Permission Description

This component does not use any cluster permissions.

Security Group Open Description

Last updated：2024-08-15 16:32:27

Overview

This document describes the port that needs to be opened for security groups of managed clusters and user clusters during the process of integrating TKE for TMP. It also describes solutions for security group related issues that arise when managed clusters and user clusters are bound.

Managed Cluster

Managed cluster Security Groups are created by TMP and generally do not need modifications.

Security Group

Rule	Protocol Port	Policy
Inbound rule	TCP:9093, 9090, 10901, 10902, 9990, 3000, 8080, and 8008	Allow
Inbound rule	TCP:8100-8200	Allow
Outbound rule	ALL	Allow

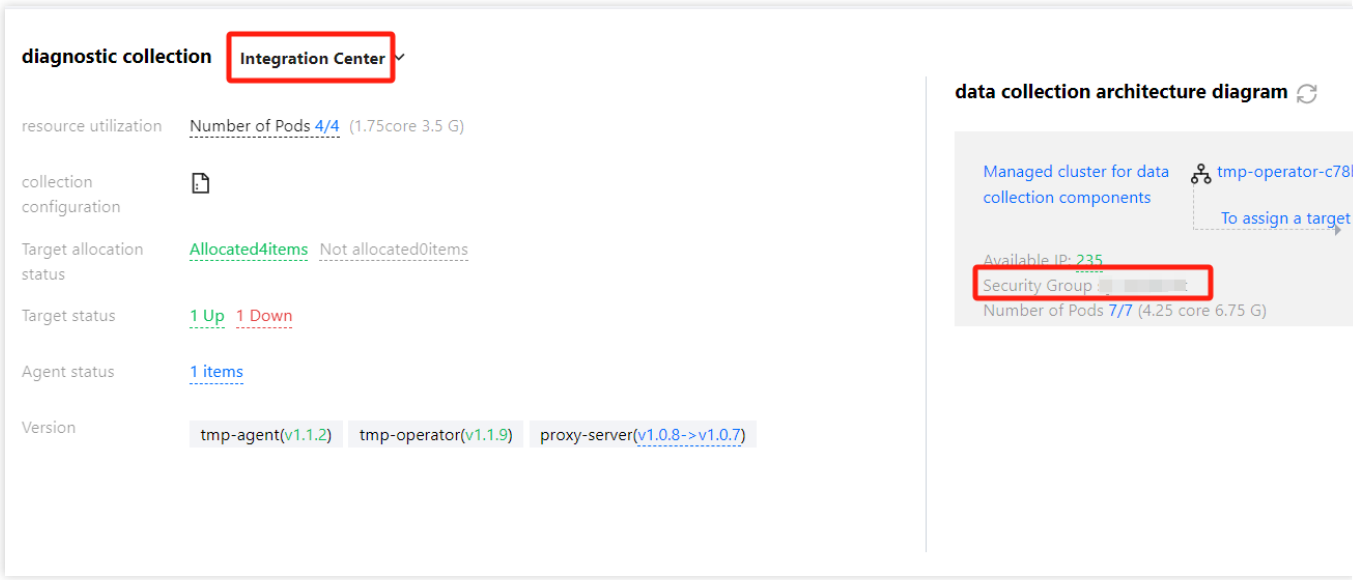
Port Description

Port	Function	Remarks
TCP:8008	proxy-server listens for the proxy-agent connection port	-
TCP:8080	Cluster internal API calls port	-
TCP:3000	grafana proxy port	-
TCP:9990	cm-notify synchronization port	About to be decommissioned
TCP:10901,10902	thanos sidecar listening address	-
TCP:9090	Configure reload port, and collect data query API	-

TCP:9093	Alarm port	-
TCP:8100-8200	proxy-server listening collection port	Since the collection port range is 100, the maximum number of associated clusters cannot exceed 100.

Viewing Method

log in to Prometheus Monitoring, select the instance's ID/Name > **instance diagnostics**, choose **Integration Center** for diagnostics, in the **data collection architecture diagram** you can see the Managed Cluster Security Group, click it to jump to the security group interface via hyperlink to view the Managed Cluster Security Group.



User Cluster

The user cluster security group is specified when the user creates a node. If not specified, the default security group will be used.

Security Group

Rule	Cluster Type	Protocol Port	Policy	Description
Outbound rule	-	TCP:8008	Allow	Ensure that the proxy-agent and proxy-server can establish a connection
Inbound	Standard	-		The standard cluster does not need

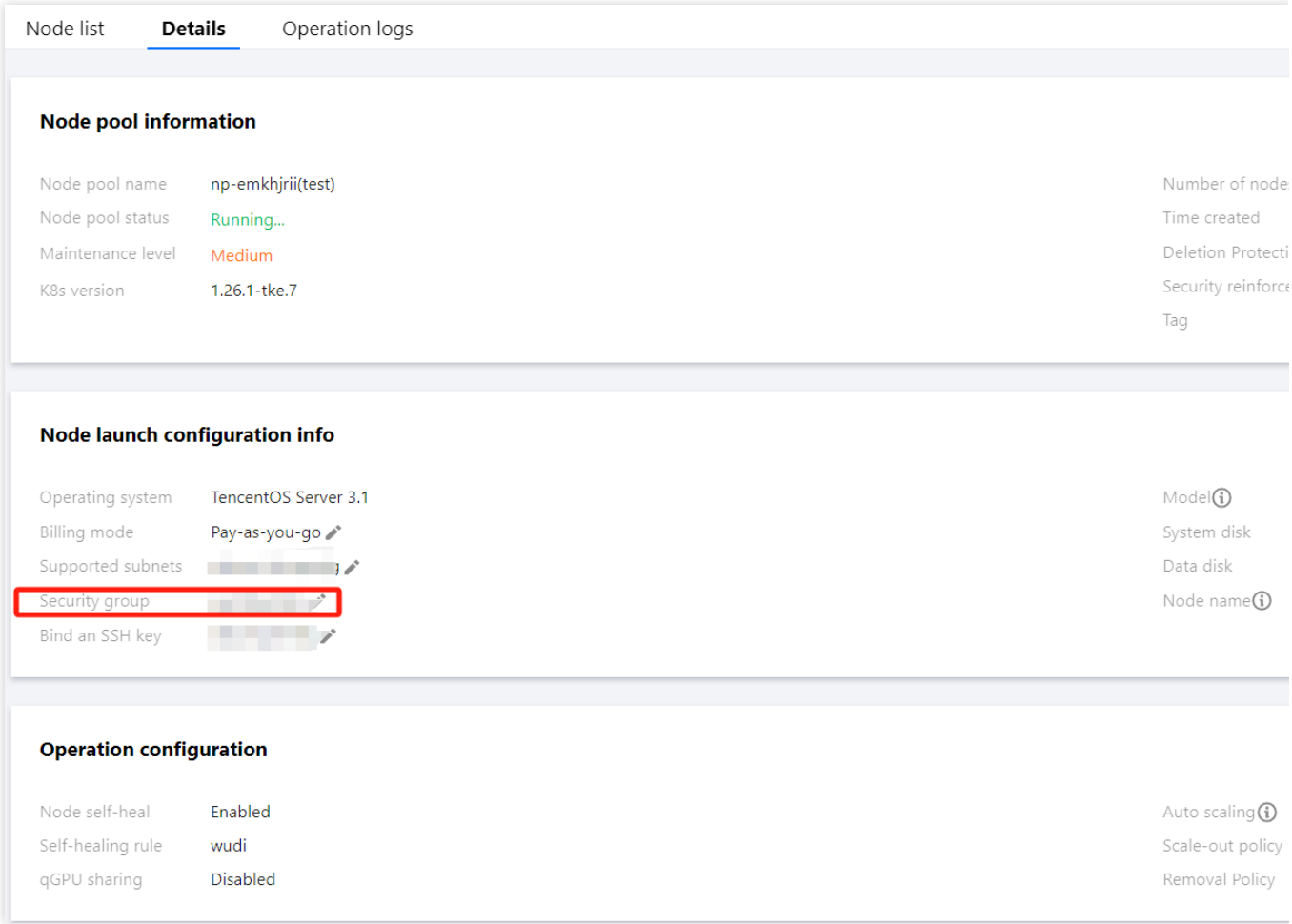
rule	cluster			opening ports.
Inbound rule	Independent cluster	TCP: 9092, 8180, 443, 10249, 9100, 60002, 10252, 10257, 10259, and 10251	Allow	The independent cluster needs to open additional master node-related ports to ensure proxy-agent can pull master node-related monitoring data

Viewing Method

log in to Prometheus Monitoring, select the instance ID/Name > **Data Collection**, and click the cluster ID/Name to jump to the cluster's TKE interface.

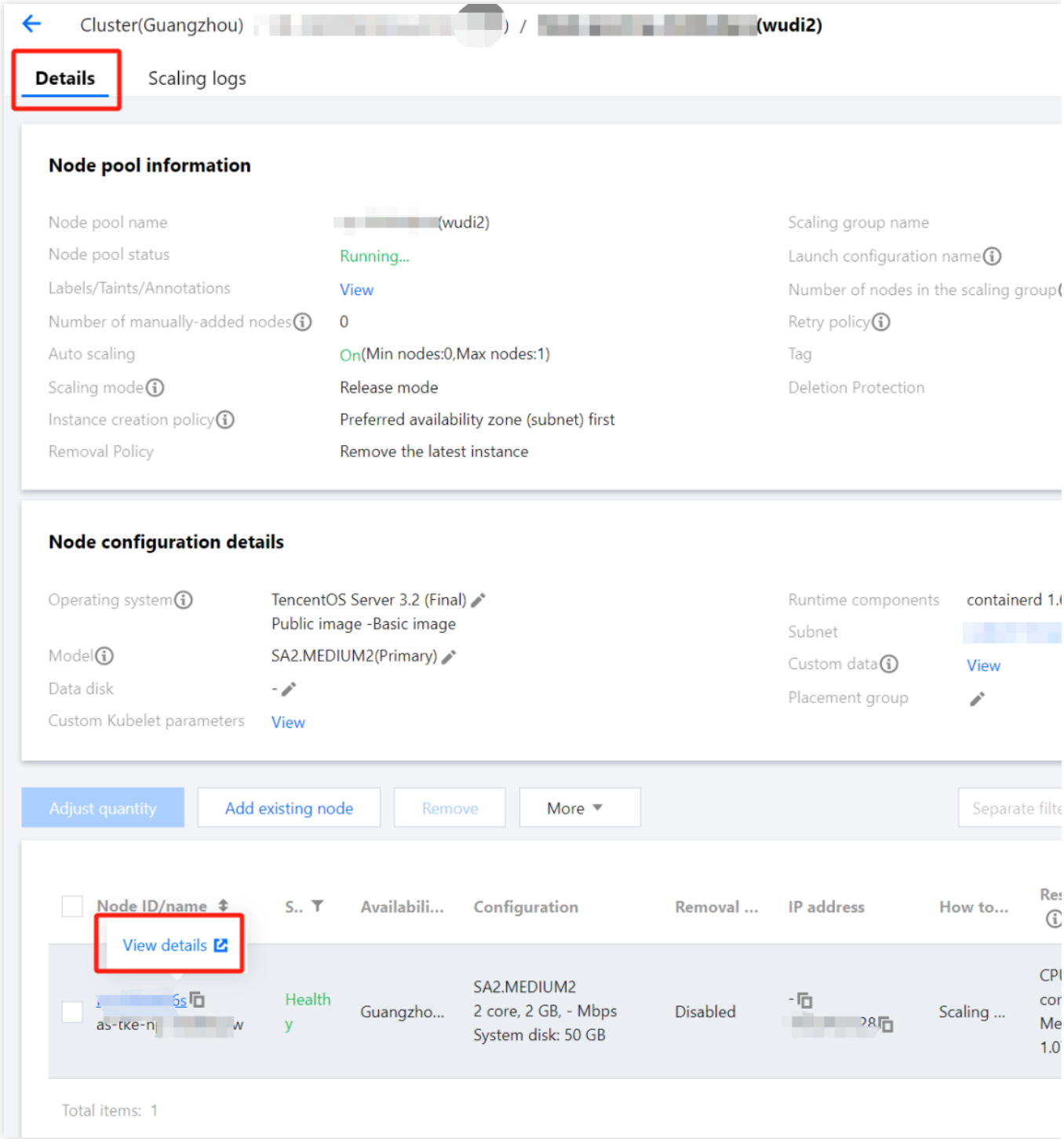
Native Nodes

Click **Node Management** > **Worker Node** > **Node pool**, and **click Node Pool ID**. In the **Details** page, you can see the security group. In the **Security group**, search by security group ID to view specific rules.

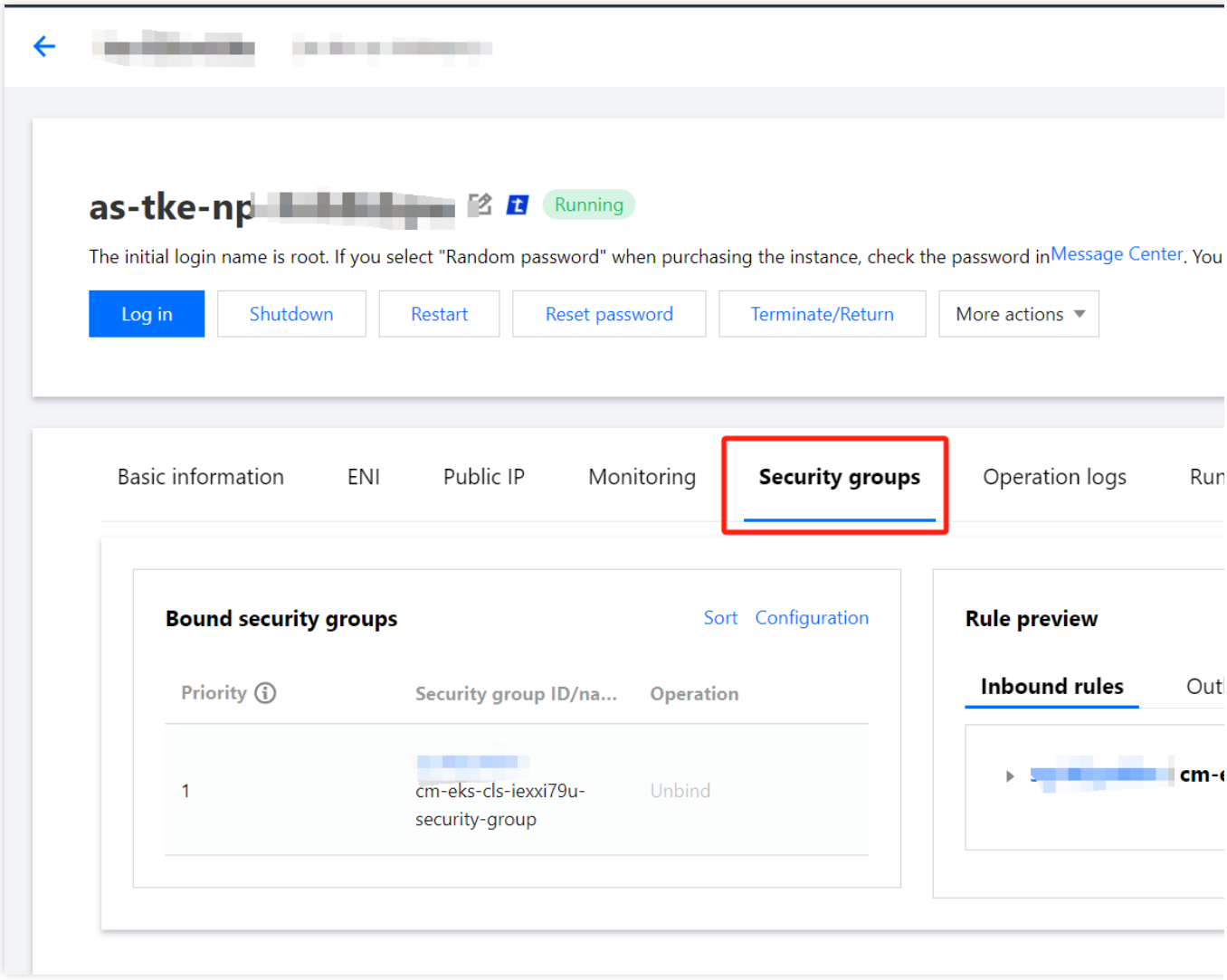


Common Nodes

Click **Node Management** > **Worker Node** > **Node Pool**, and click Node Pool ID. In the Details page, hover over the Node ID and click **Details**:

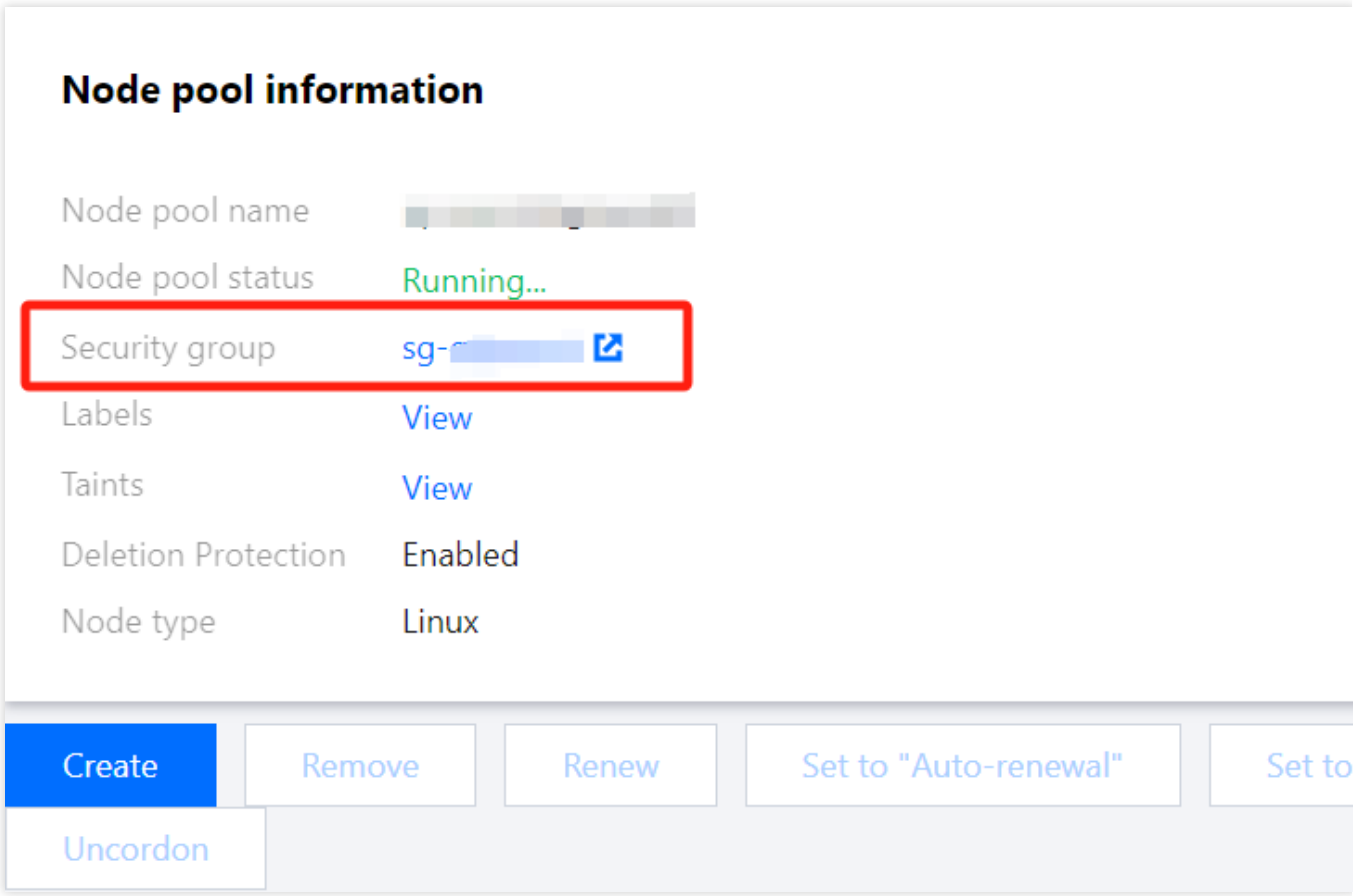


After navigating to the Instance Details page, click **Security groups** to view specific security group information:



Super Nodes

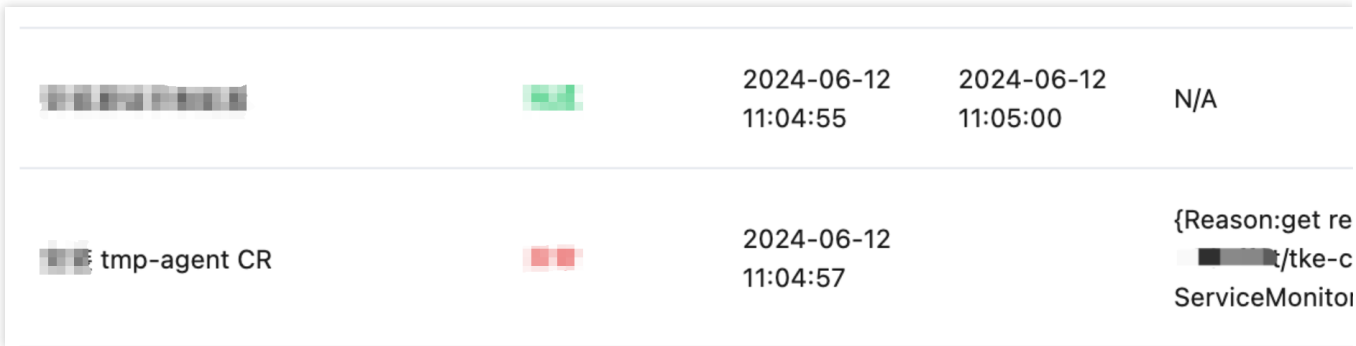
Click **Node Management** > **Worker Node** > **Node Pool**, and click Node Pool ID. In **Node pool information**, you can view the security group:



Related Issues

Issue Description

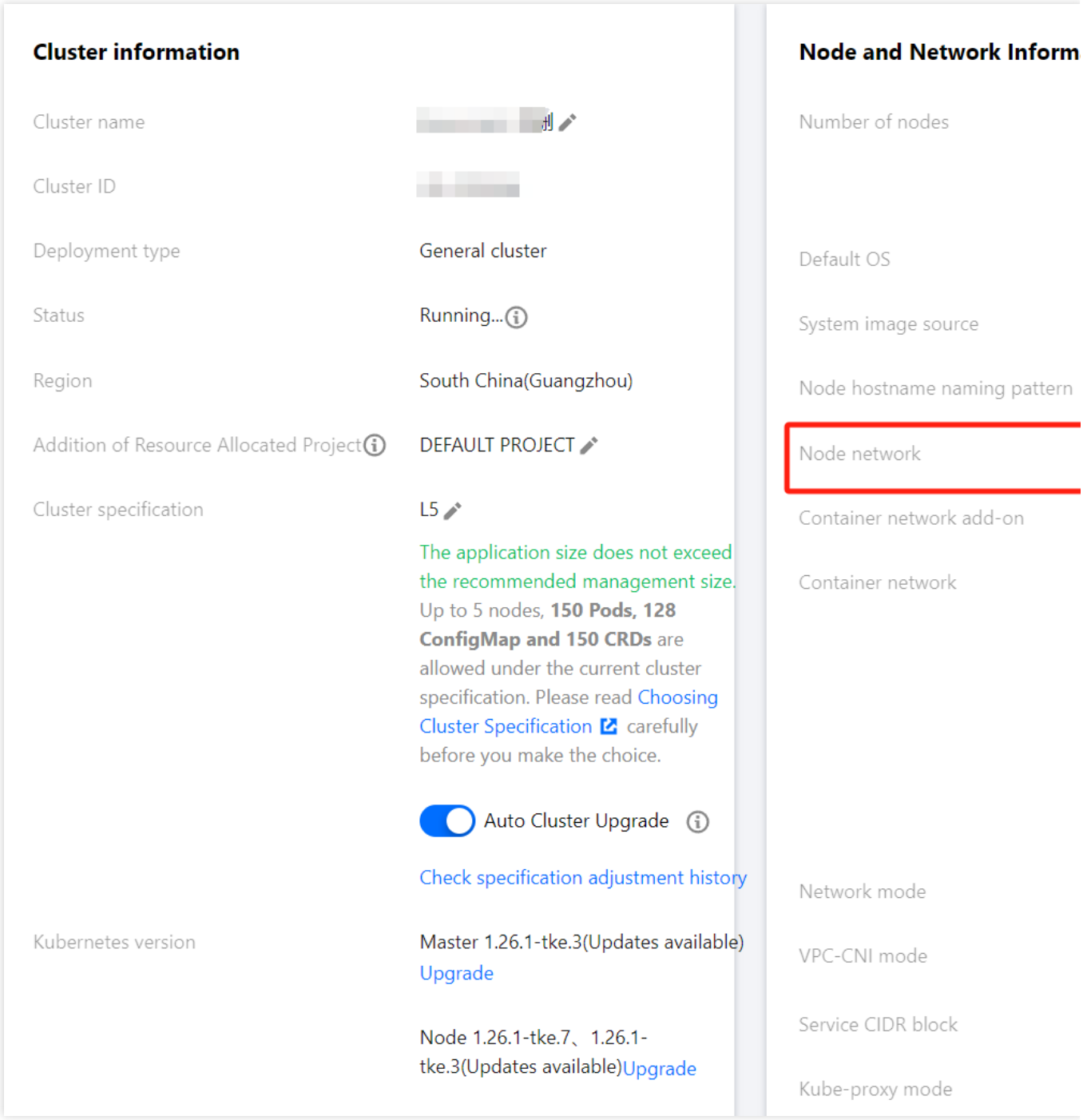
Abnormal binding status, "Install tmp-agent CR" step shows "context deadline exceeded":



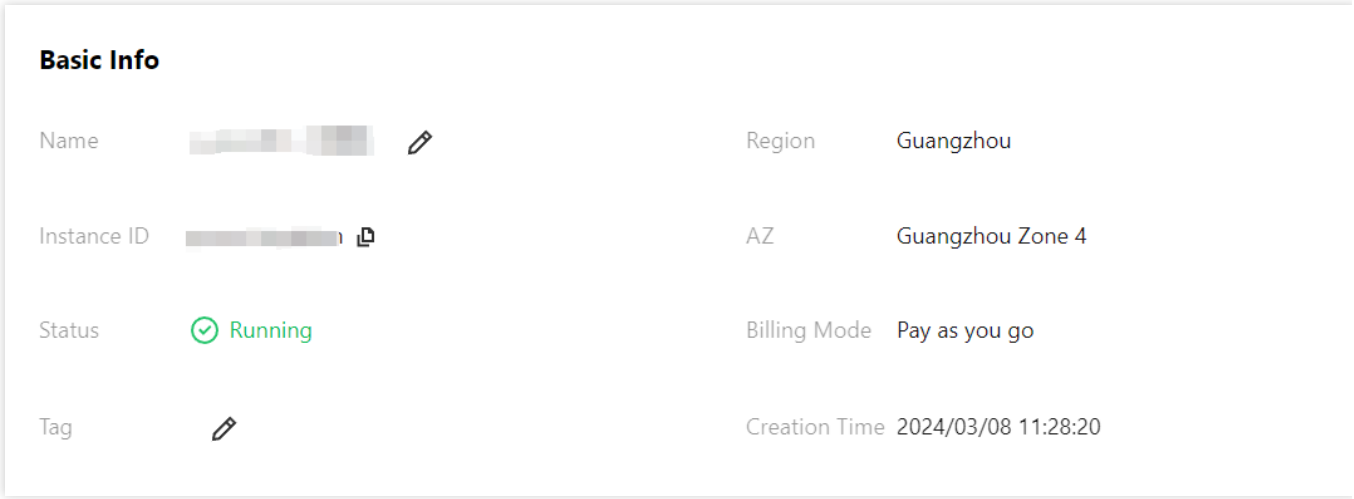
Troubleshooting

Is the VPC the Same or Interconnected?

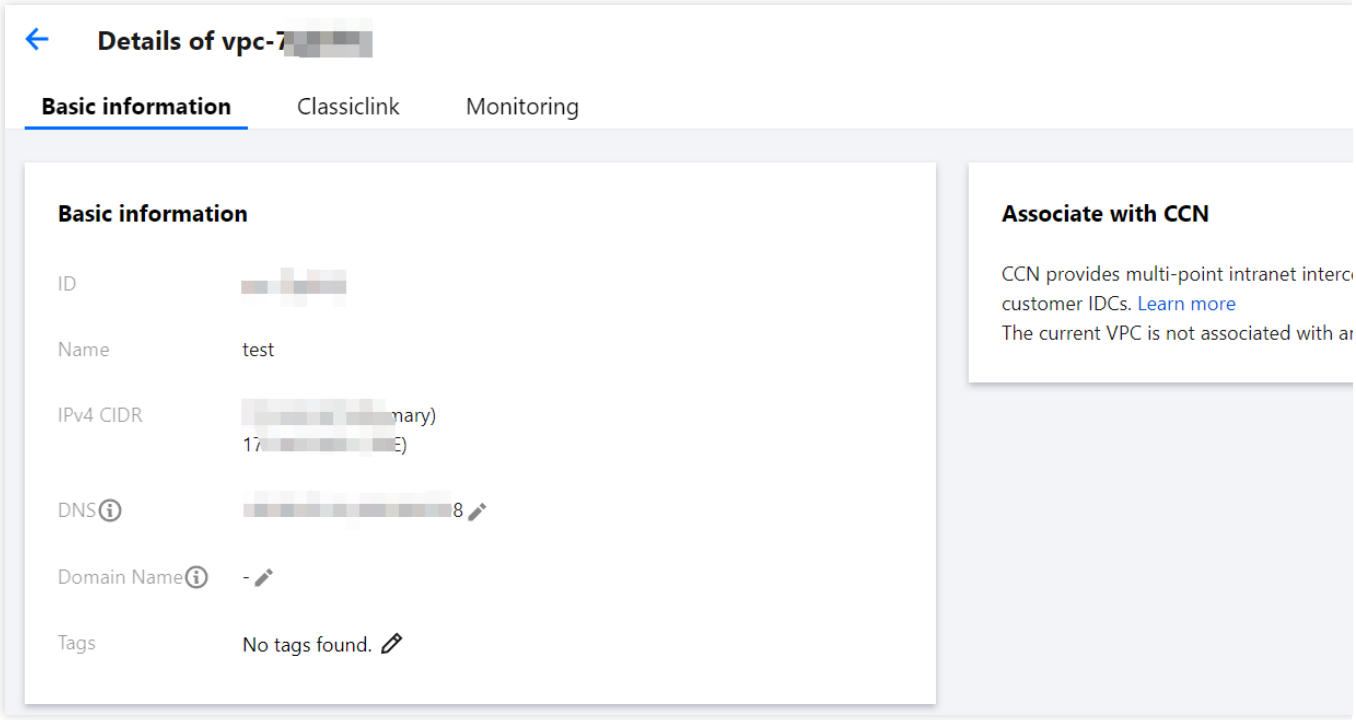
1. Click the user cluster link, open the associated cluster, and view the cluster node network (i.e., vpcid):



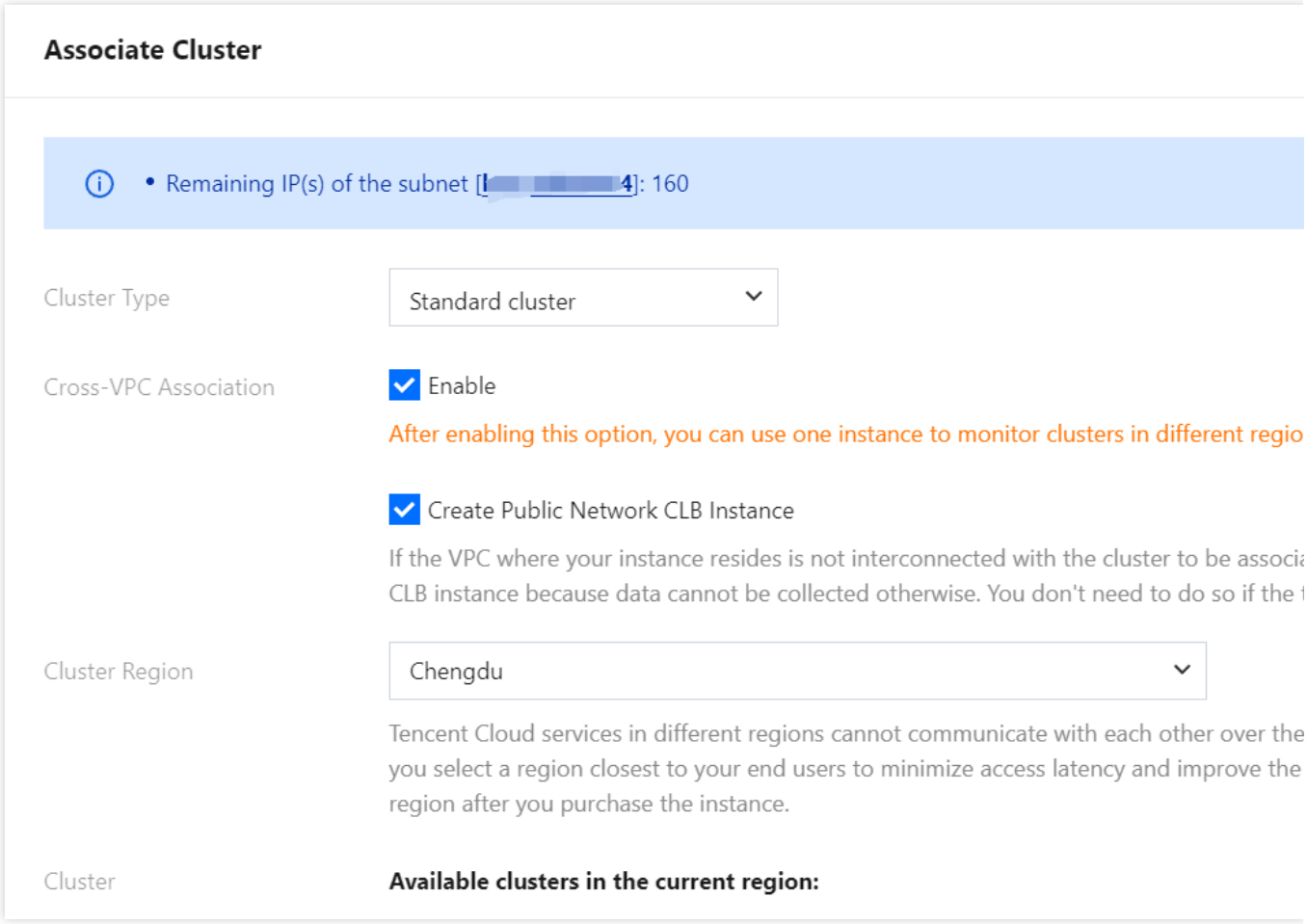
2. On the Prometheus Instance's **Basic Info** page, click **Network** to view the cluster network:



3. Compare the vpcid. If they are different, check if the VPCs are interconnected via CCN. If not, you need to associate the CCN to interconnect both VPCs or select **Create Public Network CLB Instance** when associating clusters. If CCN is interconnected but still unsuccessful, check if the CCN bandwidth limit is reached. If so, increase the CCN bandwidth limit.
Associate with CCN:

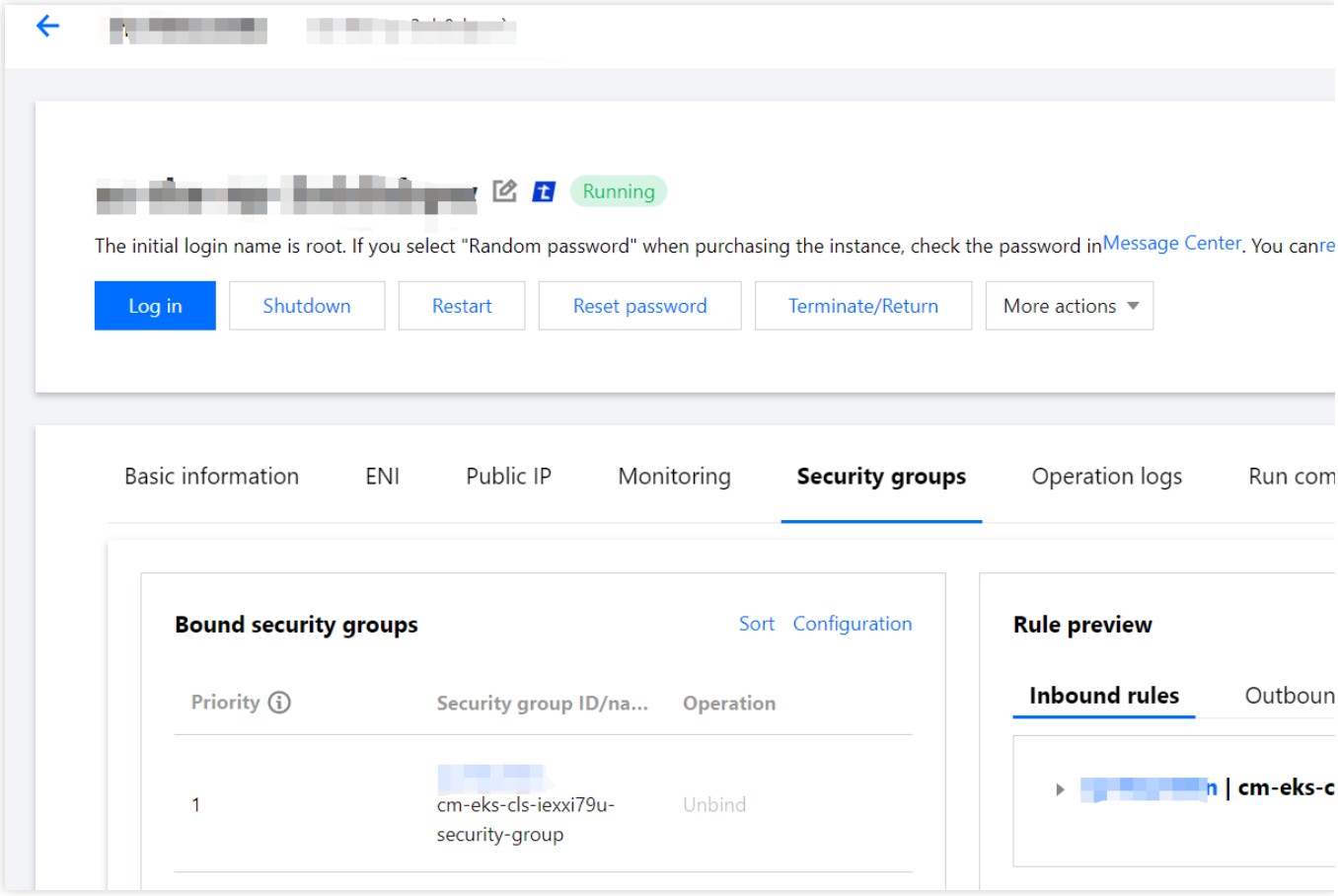


Select Create Public Network CLB Instance:



Does the Security Group Allow Access?

1. View the user cluster security group. For viewing methods, see User Cluster Security Group Viewing Method. Check if the rules meet the requirements.
2. If the user cluster is an independent cluster, view the Master&Etcd security group information. Click **Node Management > Master&Etcd > Node Pool**, click the Node Pool ID, hover over the Node ID, and then click **Jump to CVM Instance Details Page**. On the CVM **Security groups** page, you can view specific security group information:



Check if the security group rules meet the requirements.

Operation Guide
Instance
Creating Instance

Last updated：2024-08-07 22:03:06

This document uses custom configuration as an example to describe how to create a TMP instance.

Preparations

Before creating a TMP instance, you must complete the following operations:
Sign up for a Tencent Cloud account and complete identity verification.
Create a VPC in the target region and create a subnet in the target AZ in the VPC.

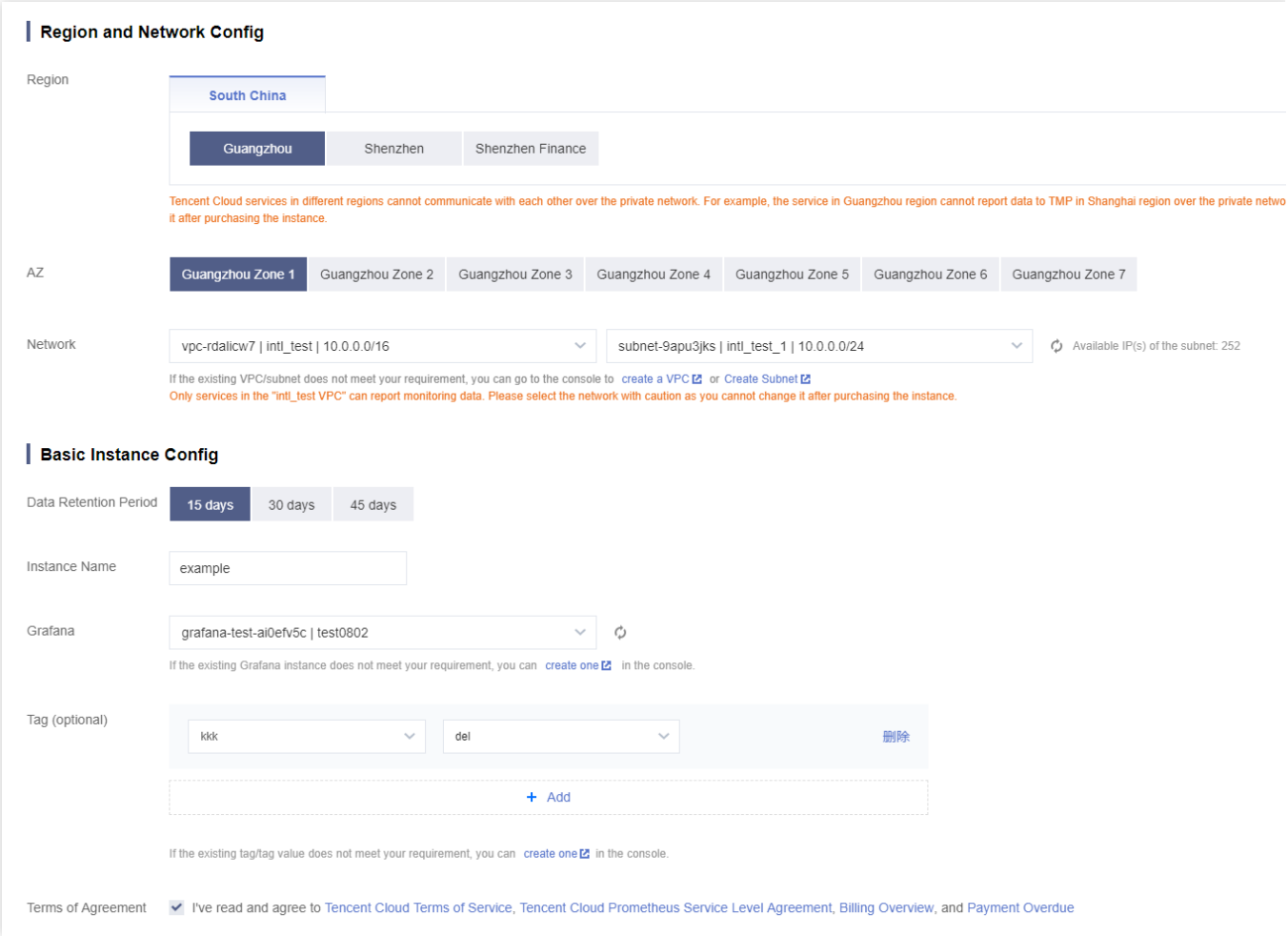
Directions

1. Log in to the TMP console.
2. Click **Create** and configure the following information as prompted:

Type	Required	Configuration Description
Region	Yes	Select a region based on the region of your Tencent Cloud service. The price may vary by region, and the real-time price displayed on the purchase page shall prevail. Tencent Cloud services in different regions cannot communicate with each other over the private network; for example, a service in the Guangzhou region cannot report data to a TMP instance in the Shanghai region over the private network. Once an instance is purchased, the region cannot be changed. Therefore, please select the region with caution.
AZ	Yes	Select as needed.
Network	Yes	It indicates a logically isolated network space in Tencent Cloud. A VPC consists of at least one subnet. The system will provide a default VPC and subnet for you in each region. If the existing VPCs/subnets don't meet your requirements, you can create new ones as instructed in Creating VPCs or Creating Subnets.
Product Specs	Yes	The price varies by product specs. For more information, please see Pricing.
Data Retention	Yes	The price varies by data retention period. For more information, please see

Period		Pricing.
Instance Name	Yes	Enter a custom name of the TMP instance.
Grafana/Grafana Password	Yes	Grafana is enabled by default. After the instance is successfully created, the system will generate a domain name accessible over the public network. The default account of the Grafana service is `admin`, and the password is user-defined.
Validity Period	Yes	Multiple validity periods are available for your choice.

3. After completing the configuration, click **Buy Now**.



Searching for Instance

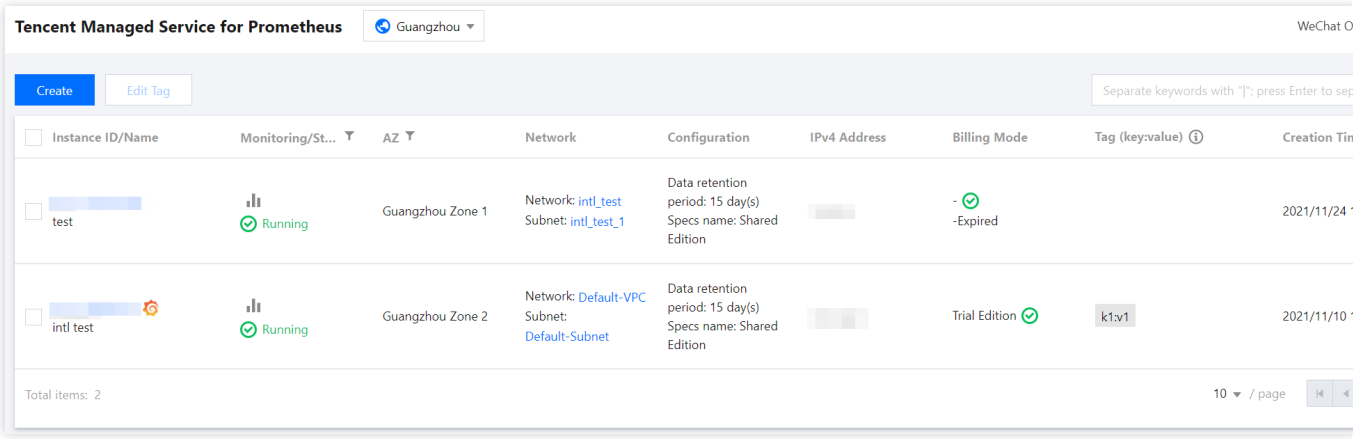
Last updated：2024-08-07 22:03:19

By default, TMP instances in the current region are displayed in the TMP console. To help you quickly find the instances in the current region, Tencent Cloud provides the search feature and allows you to filter instances by resource attributes such as instance ID, name, status, AZ, IPv4 address, and tag.

Directions

1. Log in to the TMP console.
2. Enter the conditions in the search box as needed and click

.



3. You can filter instances by different conditions. Currently, the following dimensions are supported:

Filter Condition	Description
Instance ID	You can directly enter multiple instance IDs for quick filtering. Each instance ID supports only exact match-based filtering.
Instance name	You can enter only one instance names for filtering. Fuzzy match-based filtering is supported.
Instance status	You can enter multiple status values for filtering. You can also configure the list header for quick filtering.
AZ	You can enter multiple AZs for filtering. After the region is switched, the corresponding AZs in the region will be displayed. You can also configure the list header for quick filtering.

IPv4 address	You can enter multiple IPv4 addresses for filtering. Each IPv4 address supports only exact match-based filtering.
Tag	You can enter multiple tags to filter instances. You can also directly click a tag value in the instance list for filtering.

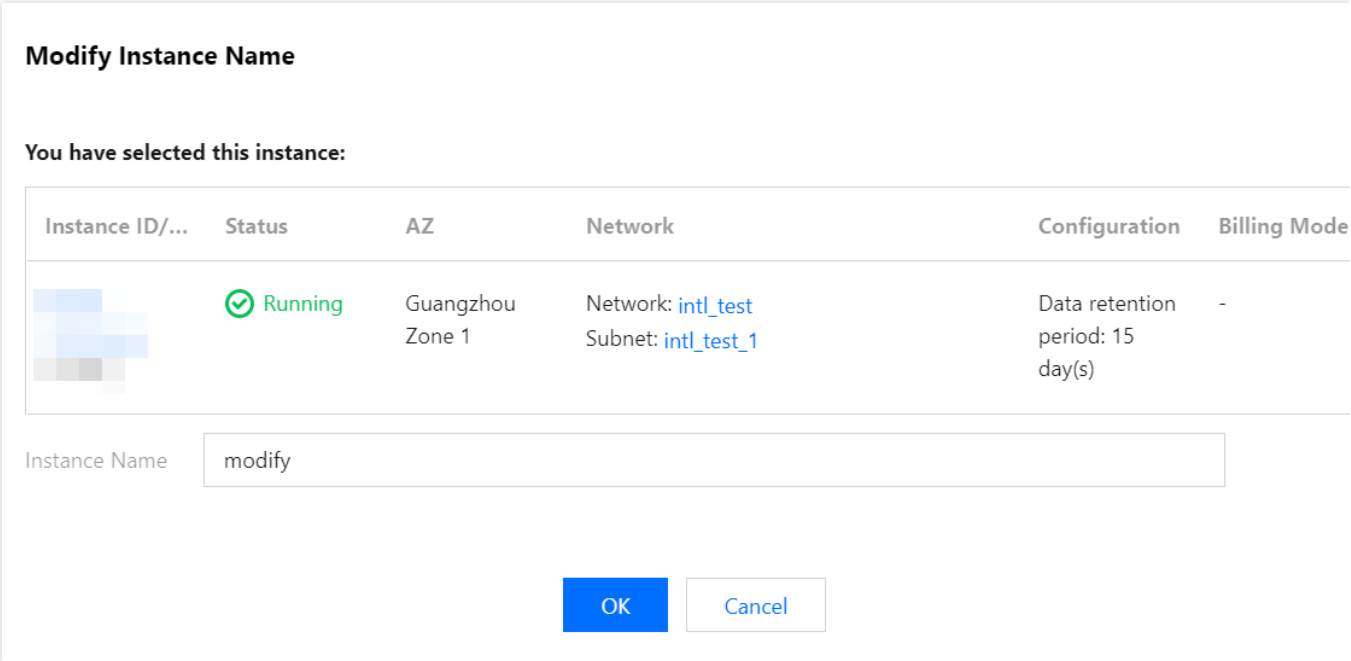
Renaming Instance

Last updated：2024-08-07 22:03:26

To help you quickly identify TMP instance names for easier instance management in the TMP console, Tencent Cloud allows you to name all instances and rename them at any time with immediate effect.

Directions

1. Log in to the TMP console.
2. In the instance list, select the TMP instance to be renamed and click **More** > **Instance Configuration** > **Rename** on the right.
3. In the **Rename** window that pops up, enter the new instance name and click **OK**.



Terminating Instance

Last updated：2024-08-07 22:03:31

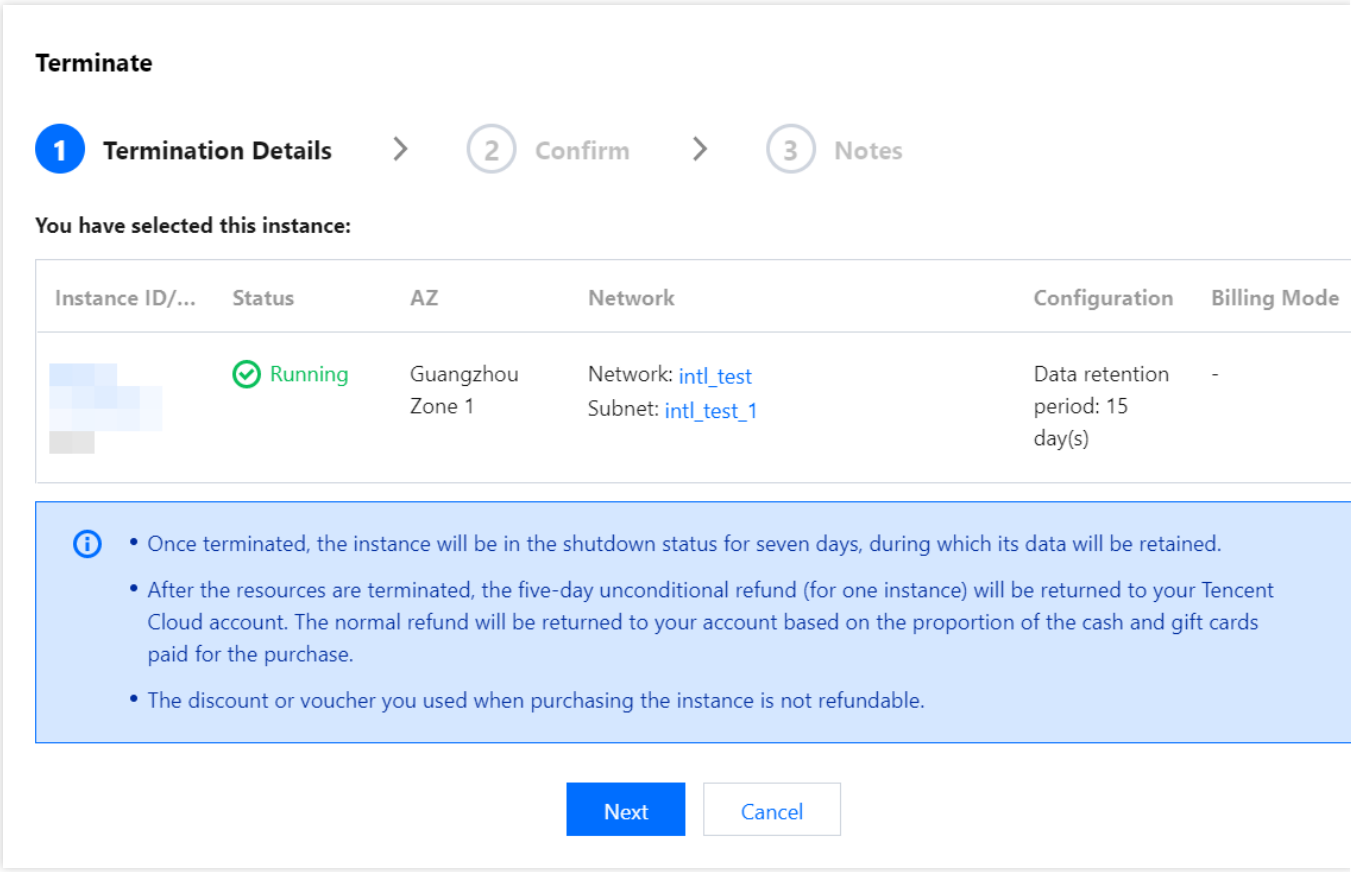
If you no longer use an instance, you can terminate it, and it will be suspended once terminated. You can reboot suspended instances according to different scenarios and needs.

Relevant Impact

Once an instance is suspended, its data will be affected as follows:
IP: the corresponding IP address is retained but does not provide normal services.
Grafana: Grafana cannot be accessed at the corresponding domain name.
Data: data on the corresponding instance will be retained for a certain number of days. The specific number displayed in the confirmation information during instance termination in the console shall prevail.

Directions

1. Log in to the TMP console.
2. In the instance list, select the TMP instance to be terminated and click **More** > **Instance Status** > **Terminate** on the right.
3. As termination is a high-risk operation, in the **Terminate** window that pops up, complete the termination steps as prompted and click **OK**.



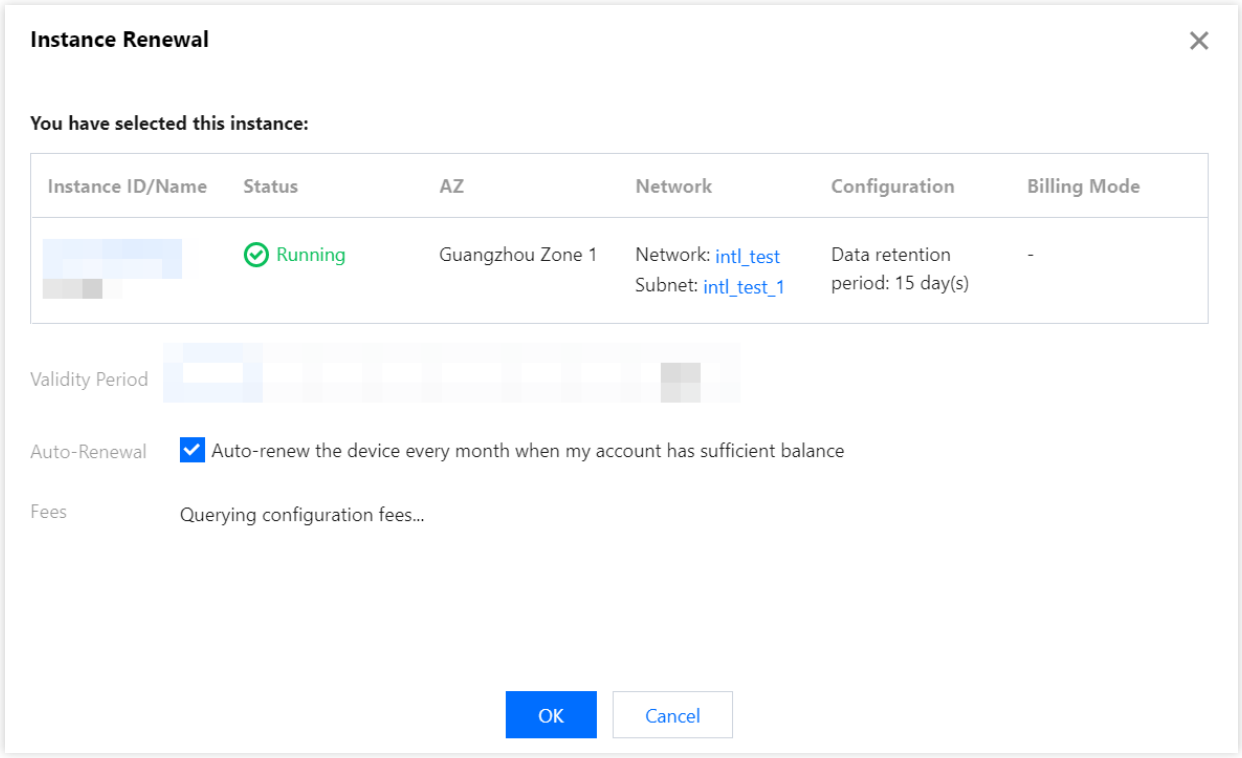
Rebooting Instance

Last updated：2024-08-07 22:03:36

You can reboot a suspended or abnormal instance in the console.

Directions

1. Log in to the TMP console.
2. In the instance list, select the TMP instance to be rebooted and click **More** > **Instance Status** > **Reboot** on the right.
3. In the **Reboot Notes** window that pops up, click **OK**.



Note:

If you reboot a running instance, the service will be interrupted during reboot. Please confirm the operation risks first.

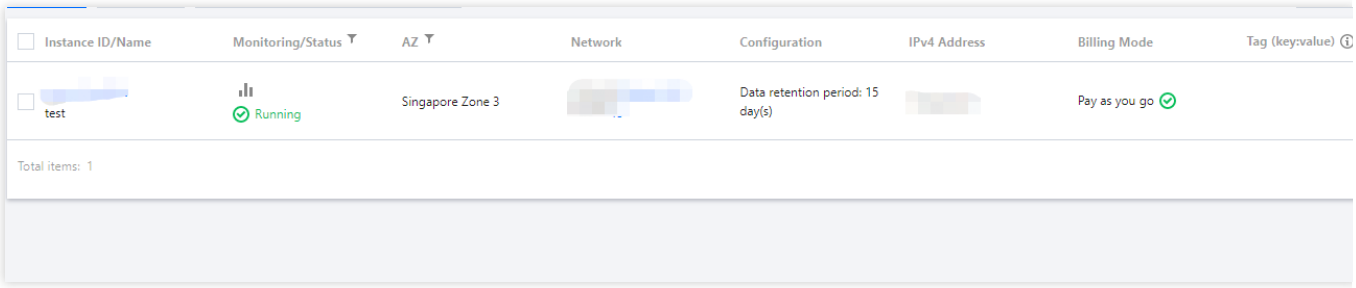
Modifying Storage Period

Last updated：2024-08-07 22:03:43

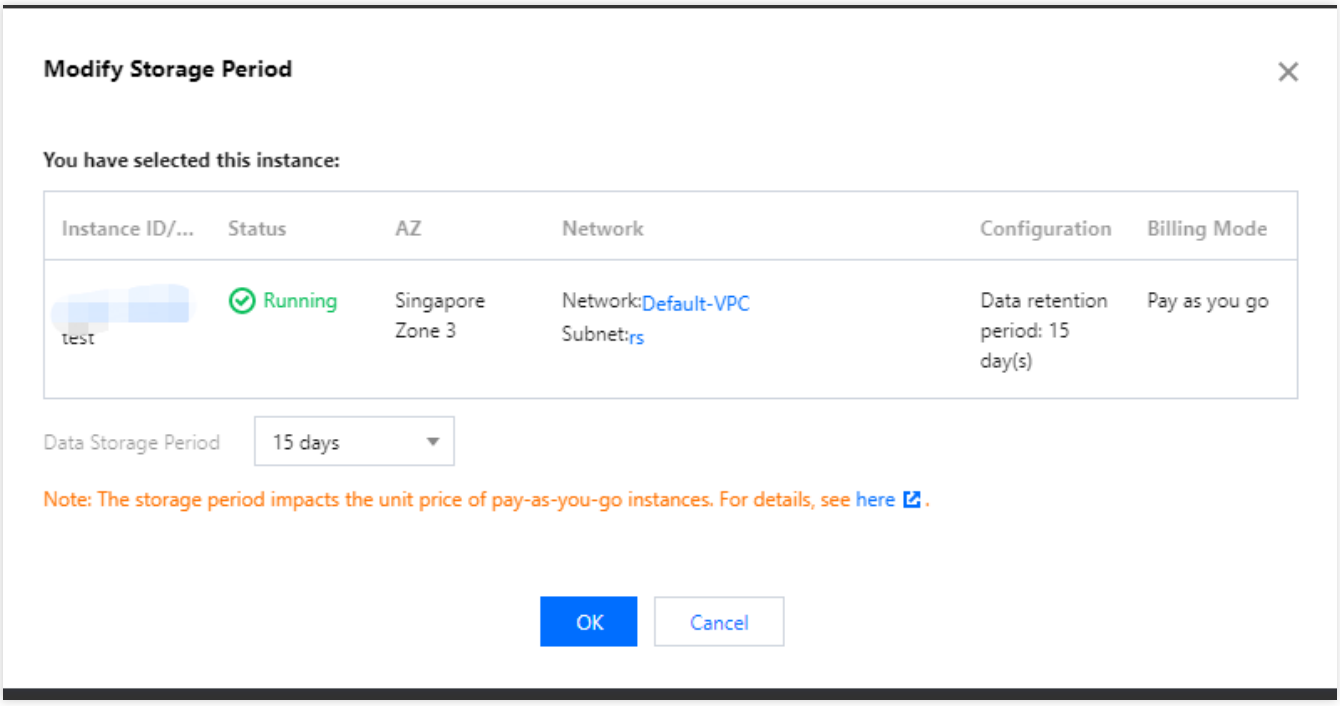
TMP allows you to modify the data storage period after creating an instance. The longer the storage period, the higher the unit price of the instance. For billing details, see Pay-as-You-Go.

Directions

1. Log in to the TMP console.
2. In the instance list, find the target TMP instance and click **More** > **Instance Configuration** > **Modify Storage Period** on the right.



3. In the pop-up window, select the target storage period and click **OK**.



Note:

After the modification, newly collected data will be billed based on the new storage period and unit price from 0:00 AM the next day.

Historical data is still stored based on the storage period configured before the modification.

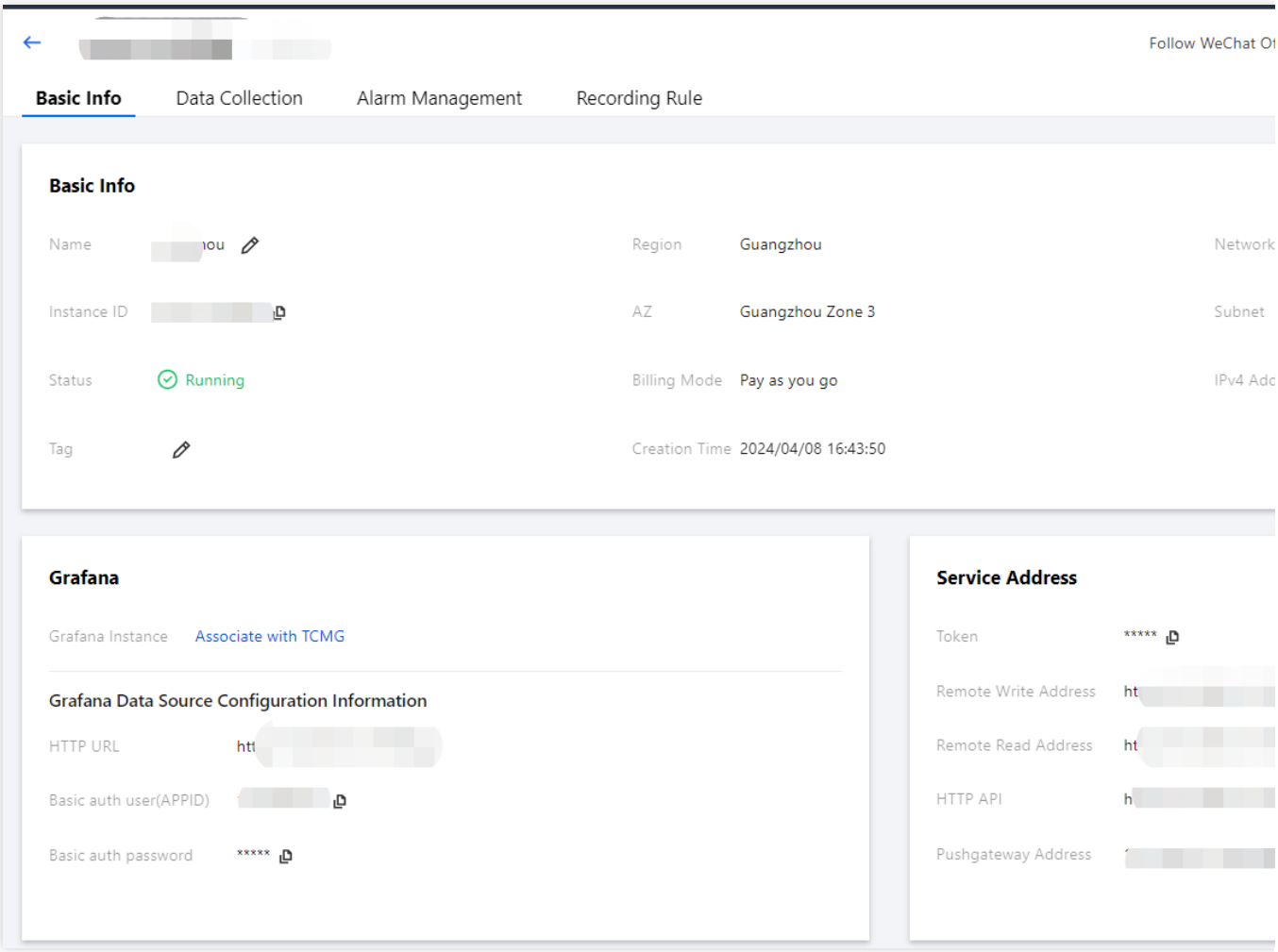
Viewing Instance's Basic Information

Last updated：2024-07-31 10:44:36

To help you view the basic information of TMP instances, TMP allows you to select a desired instance on the instance list page to enter its management page and view its basic information.

Directions

1. Log in to the TMP console.
2. In the instance list, select the TMP instance to be viewed and click its **instance ID** or **Manage** on the right.



3. You can perform the following operations on the basic information page:

Rename the instance.

Edit instance tags.

Change the Grafana Admin password.

Upgrade preset Grafana dashboards.

TKE

Integration with TKE

Last updated：2024-08-07 22:04:22

You can monitor the TKE service in business scenarios after integrating it. This document describes how to integrate the TKE service.

Tencent Kubernetes Engine (TKE) provides container-centric solutions based on native Kubernetes. It can solve environment-related issues in the process of development, testing, and Ops, helping you reduce costs and improve efficiency. Kubernetes is an open-source container orchestration tool developed by Google and has been used by Google for more than 15 years. As the de facto standard in the container field, Kubernetes can greatly simplify the management and deployment complexity of applications. By integrating with the TKE service, you can monitor the status of Kubernetes and the services running on it much more easily through Prometheus.

Note

In order to ensure normal operation, existing instances will automatically update the component version when you edit collection configuration and associate new clusters. During the update process, data breakpoints may occur in the associated clusters.

Directions

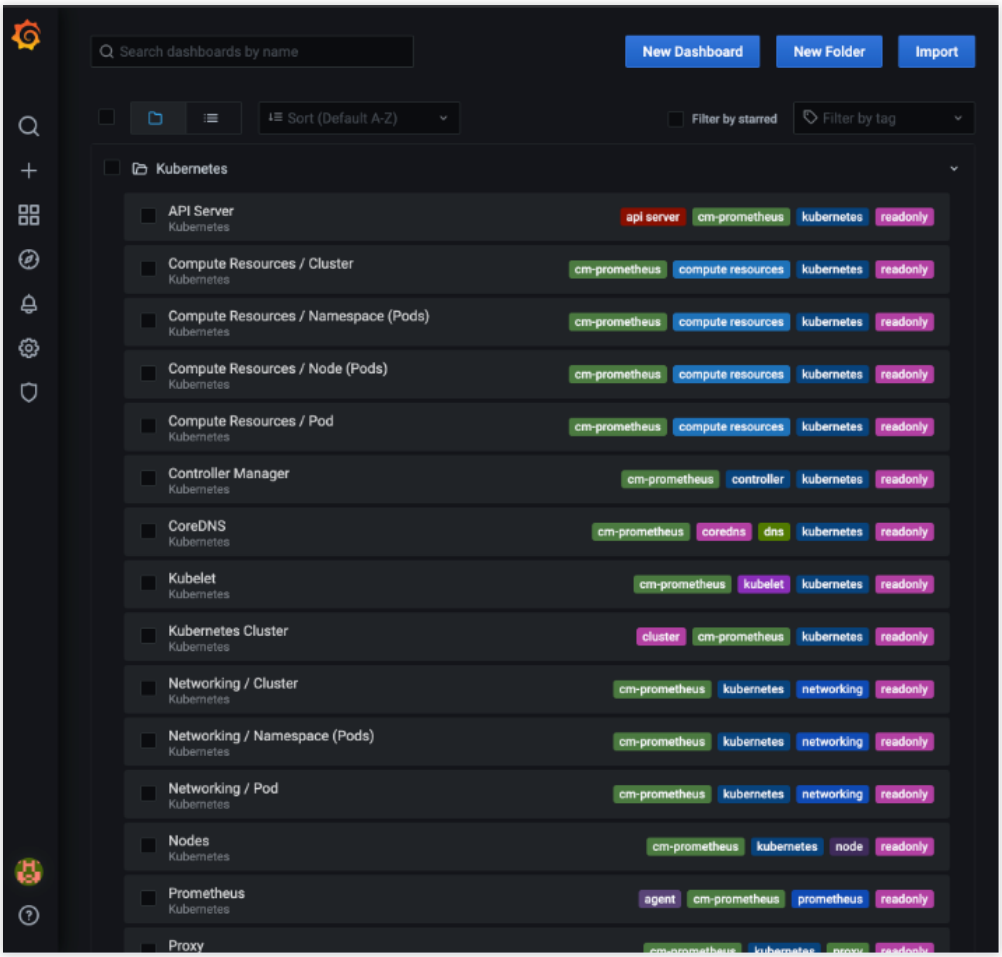
1. Log in to the TMP console.
2. In the TMP instance list, click the **ID/Name** of the newly created instance.
3. Go to the TMP management center and click **Integrate with TKE** on the left sidebar.
4. Perform the following operations on the cluster monitoring page:

Associate a cluster: Associate a cluster with a TMP instance as instructed in Associating with Cluster.

Configure data collection: Create a data collection rule to monitor your business data by adding the configuration in the console or via a YAML file. For more information, see Data Collection Configurations.

Streamline basic monitoring metrics: Select only the required metrics to avoid unnecessary fees as instructed Streamlining Monitoring Metrics.

5. After completing the above operations, you can view the monitoring data of your TKE service in Grafana.



Integration Center

Last updated：2024-08-07 22:04:30

TMP integrates commonly used programming languages, middleware, big data, and infrastructure databases. It supports quick installation and custom installation. You only need to follow the instructions to monitor the corresponding components. It also provides out-of-the-box Grafana monitoring dashboards. The integration center covers three major monitoring scenarios of basic service monitoring, application layer monitoring, and TKE cluster monitoring, making it easier for you to integrate and use.

List of Supported Services

Service Type	Service	Monitoring Metric	Quick Installation	Integration Guide
Big data	Elasticsearch	Cluster/Index/Node monitoring	Supported	Elasticsearch Exporter Integration
	Flink	Cluster/Job/Task monitoring	Not supported	Flink Integration
Development	CVM	The extended `cvm_sd_config` can be used to configure a CVM scrape task and collect Node Exporter or custom business metrics.	Supported	CVM Node Exporter
	Go	GC/Heap/Thread/Goroutine monitoring	Not supported	Go Application Integration
	JVM	Heap/Thread/GC/CPU/File monitoring	Not supported	JVM Integration
	Spring MVC	HTTP API/Exception/JVM monitoring	Not supported	Spring Boot Integration
Middleware	Kafka	Broker/Topic/Consumer group monitoring	Supported	Kafka Exporter Integration
	Consul	Consul monitoring	Supported	Consul Exporter Integration

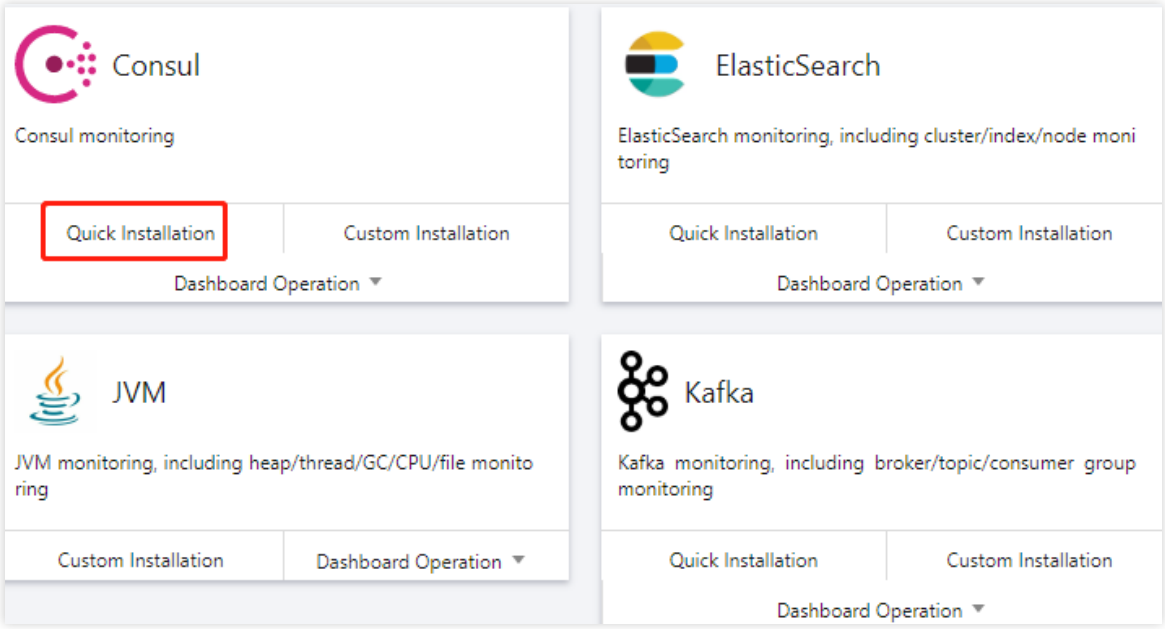
	Etcd	Etcd monitoring	Not supported	-
	Istio	Istio monitoring	Not supported	-
Infrastructure	Kubernetes	API server/DNS/Workload/Network monitoring	Supported	Agent Management
Database	TencentDB for MongoDB	File count/Read and write performance/Network traffic monitoring	Supported	MongoDB Exporter Integration
	TencentDB for MySQL	Network/Connection count/Slow query monitoring	Supported	MySQL Exporter Integration
	TencentDB for PostgreSQL	CPU/Memory/Transaction/Lock/Read/Write monitoring	Supported	PostgreSQL Exporter Integration
	TencentDB for Redis	Memory utilization/Connection count/Command execution status monitoring	Supported	Redis Exporter Integration
	TencentDB for Memcached	Memcached monitoring	Supported	Memcached Exporter Integration
Inspection	Health check	Blackbox can be used to regularly test the connectivity of the target service, helping you stay up to date with the service health and discover exceptions in time.	Supported	Health Check
CM	CM	Tencent Cloud service monitoring	Supported	-
Custom	Scrape task	The native `static_config` can be used to configure a scrape task.	Supported	Scrape Configuration Description
	CVM scrape task	The extended `cvm_sd_config` can be used to configure a CVM scrape task.	Supported	Scrape Configuration Description

Directions

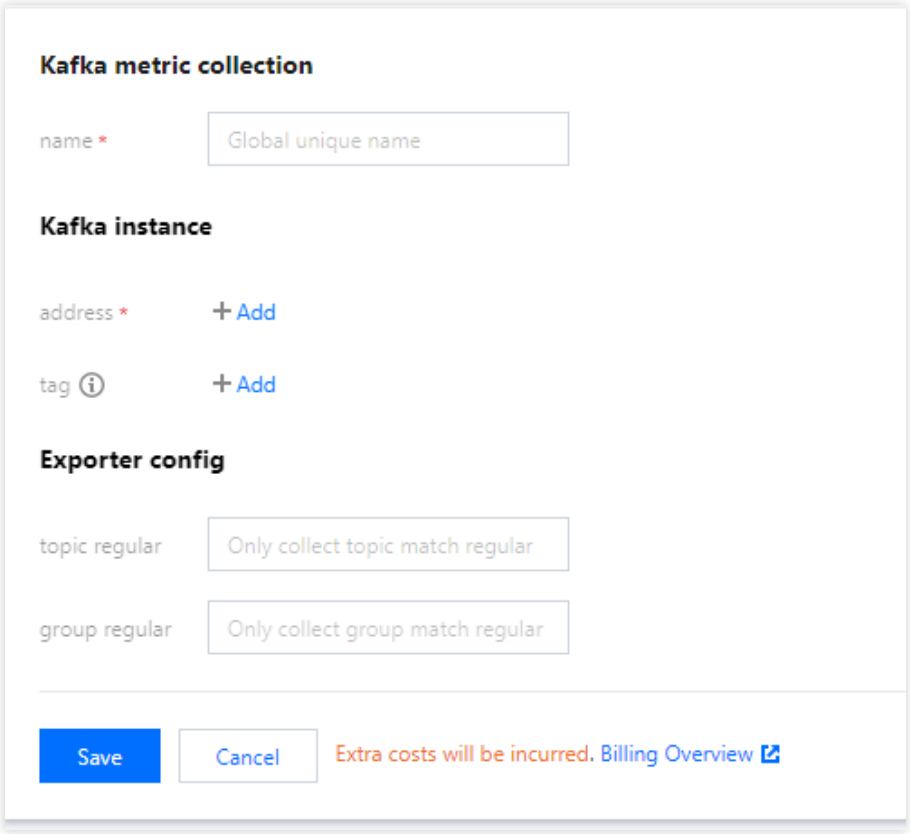
Quick installation

Some services support quick agent installation. For more information, see Integration Center > List of Supported Services.

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance.
3. Enter the instance details page and click **Integration Center**.
4. In the **Integration Center**, select the service that supports quick installation and click **Install** in the bottom-left corner.



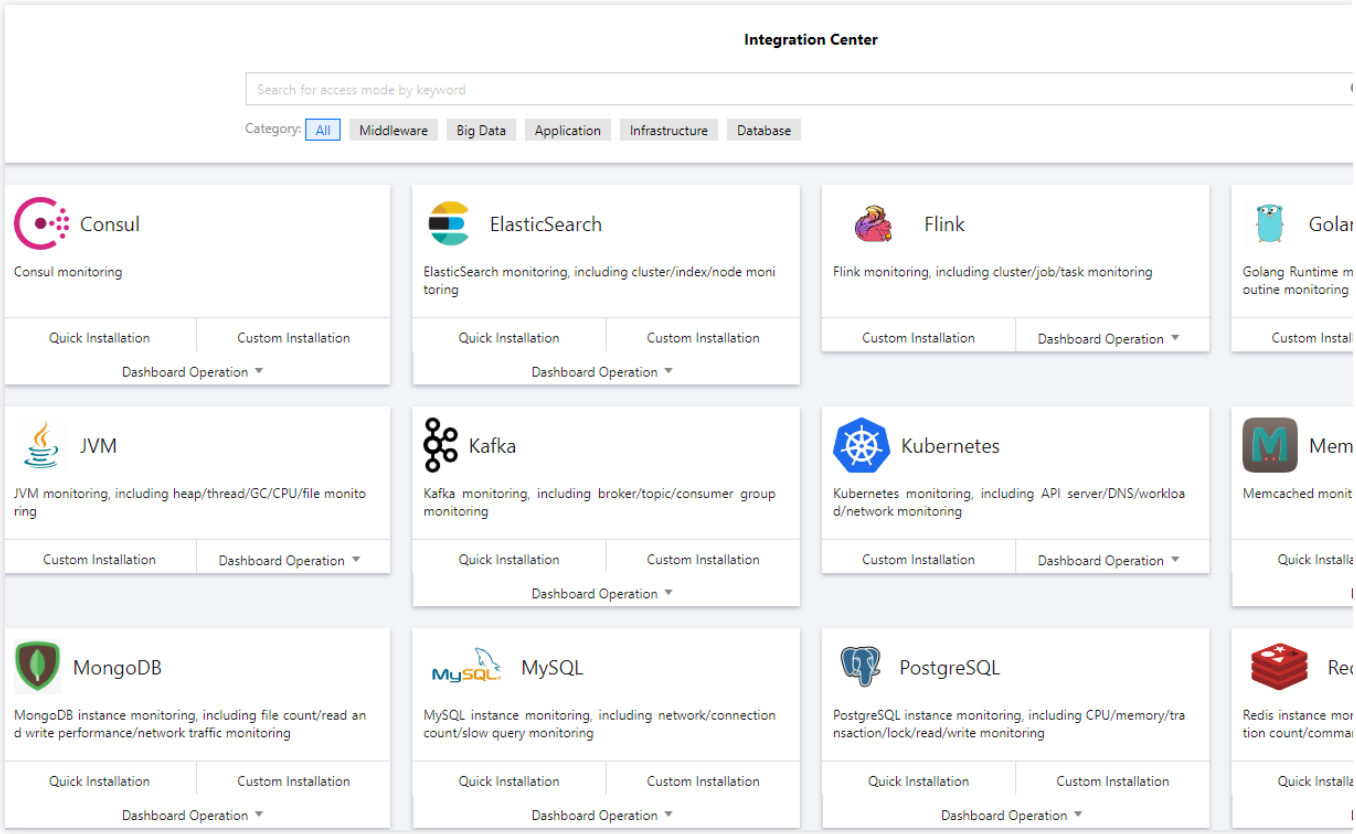
5. On the **Integration List** page, enter the metric collection name and address and click **Save**. Below is a sample for Kafka:



Custom installation

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance.
3. Enter the instance details page and click **Integration Center**.
4. Select the target service in the integration center. You can click **Integration Guide** to view the integration guide.

After successful integration, you can monitor the corresponding service in real time. You can also click **Install/Upgrade** in **Dashboard Operation** to install or upgrade the Grafana dashboard for the service.



Data Multi-Write

Last updated：2024-08-15 16:16:34

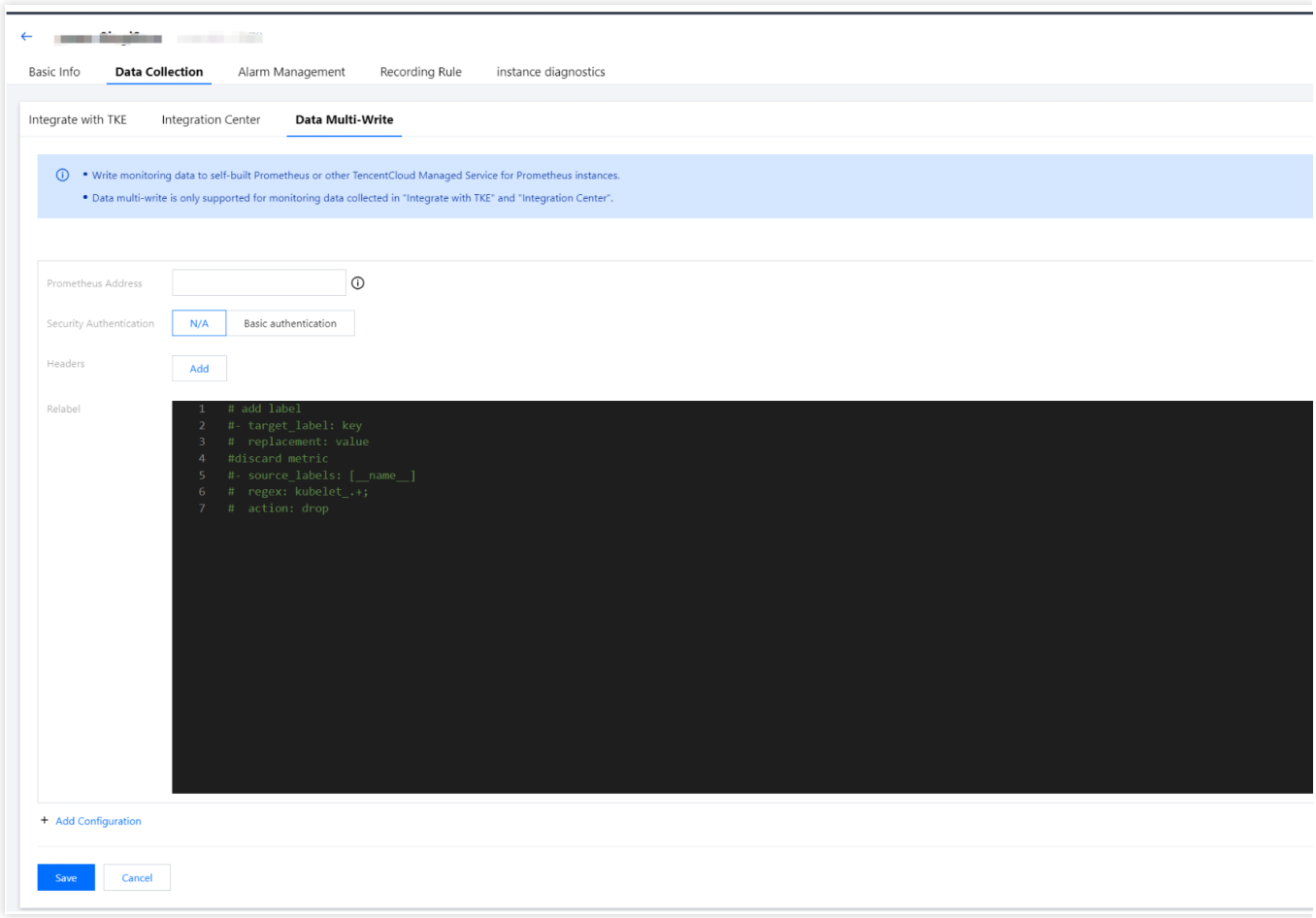
The data multi-write feature supports writing monitoring data to self-built Prometheus or other Prometheus monitoring instances.

Note:

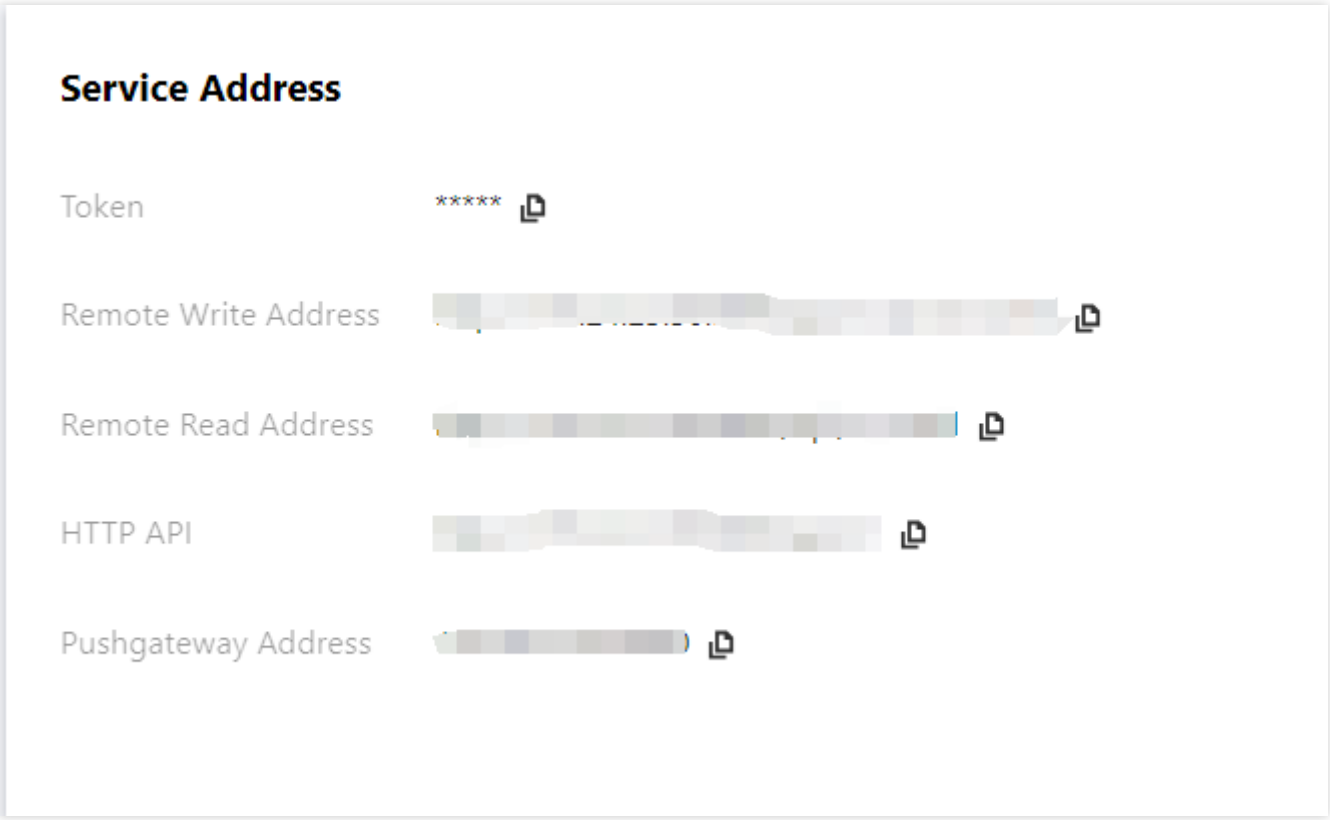
Currently, only the multi-write of monitoring data collected by Integrated Container Service and Integrated Center is supported, and others are not supported yet.

Operation Guide

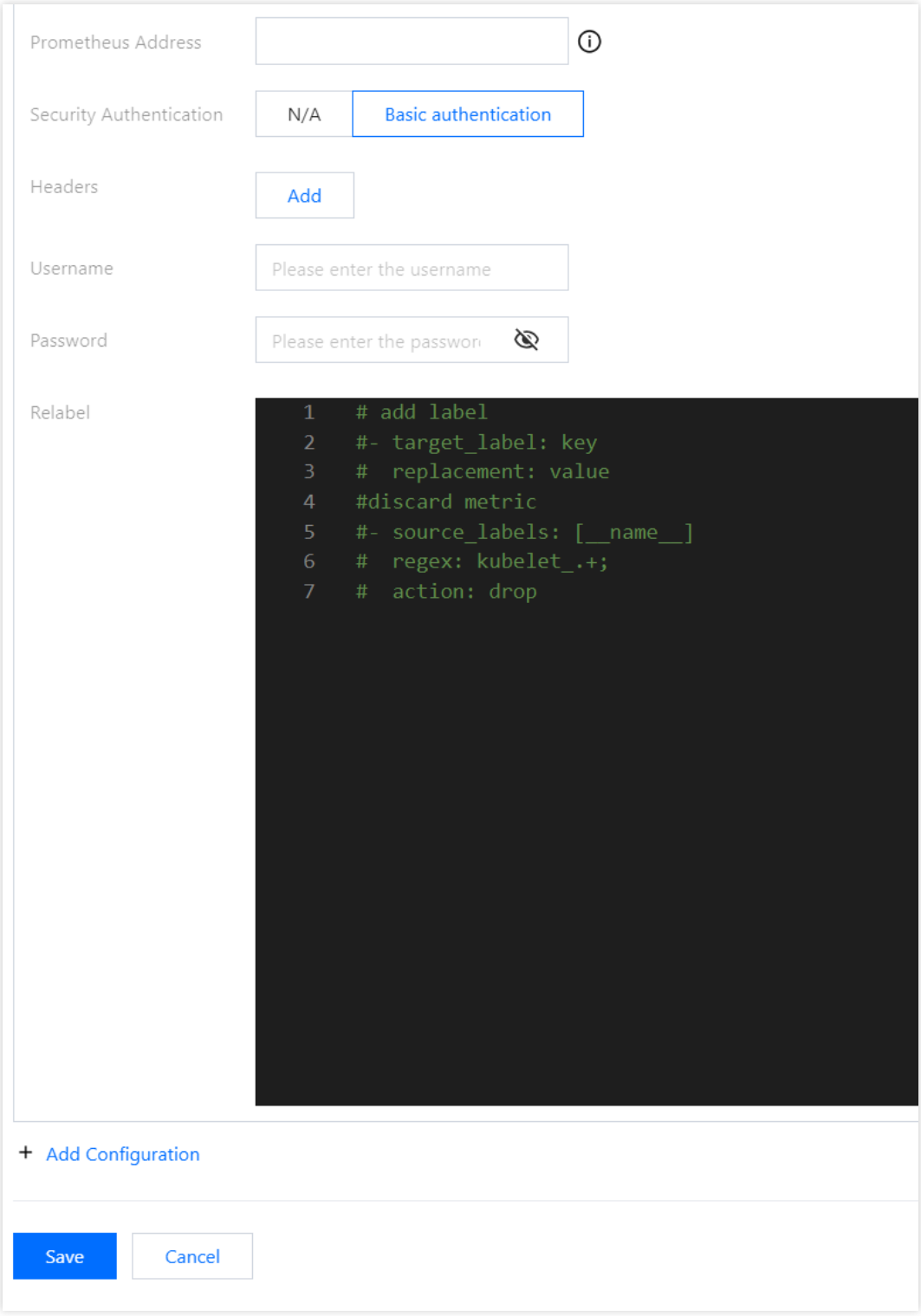
1. Log in to TMP Console.
2. In the left sidebar, click **TencentCloud Managed Service for Prometheus.**
3. Click the corresponding instance name to enter the instance details page, then click **Data Collection** > **Data Multi-Write** on the top navigation bar.
4. In the Data Multi-Write sub-window, click **Add** and see the following description to configure the data multi-write information.



Prometheus address: Obtain the HTTP API address from the basic information of instances.



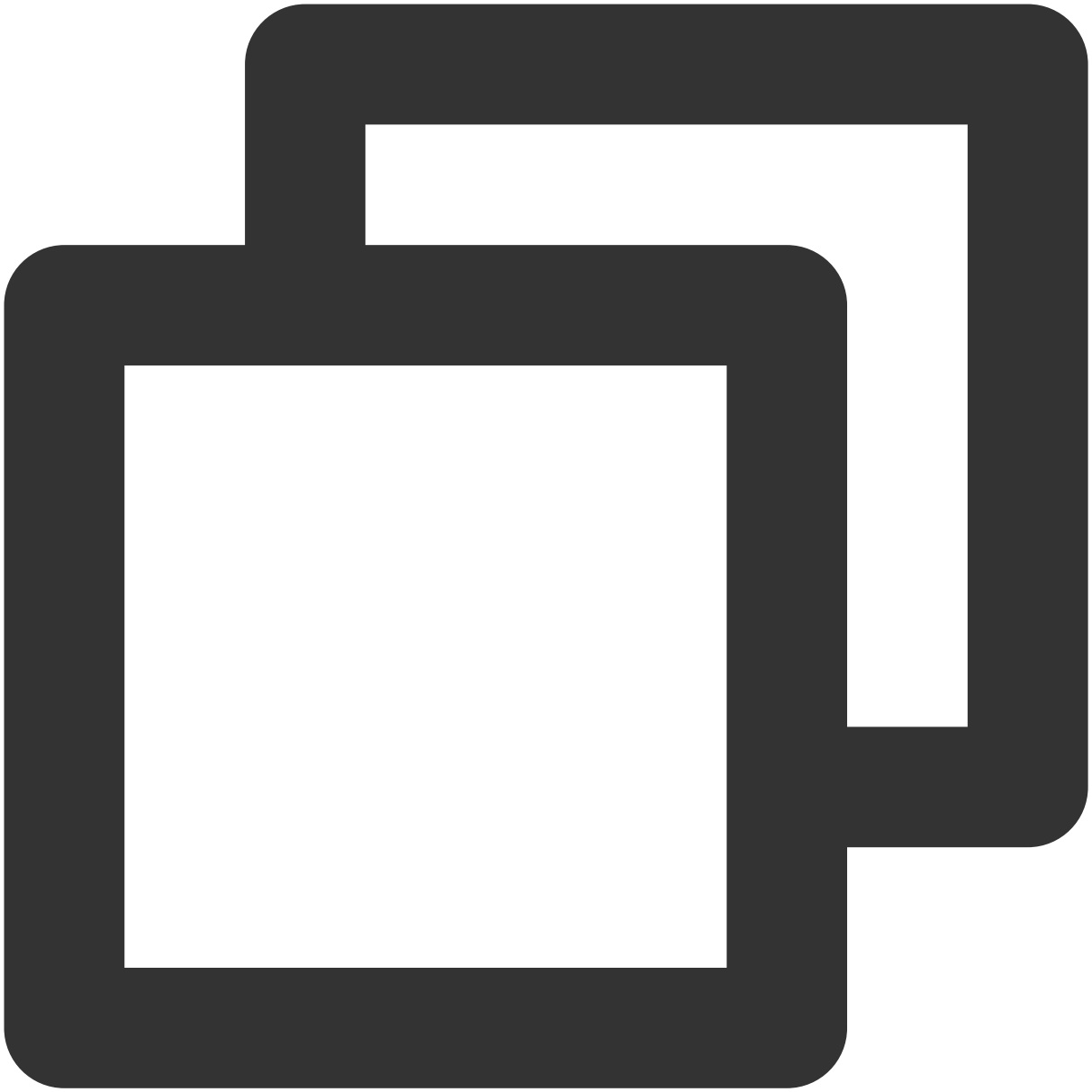
Security verification: Whether to enable security authentication. Once enabled, the username and password needs to be customized.



Recording Rule Overview

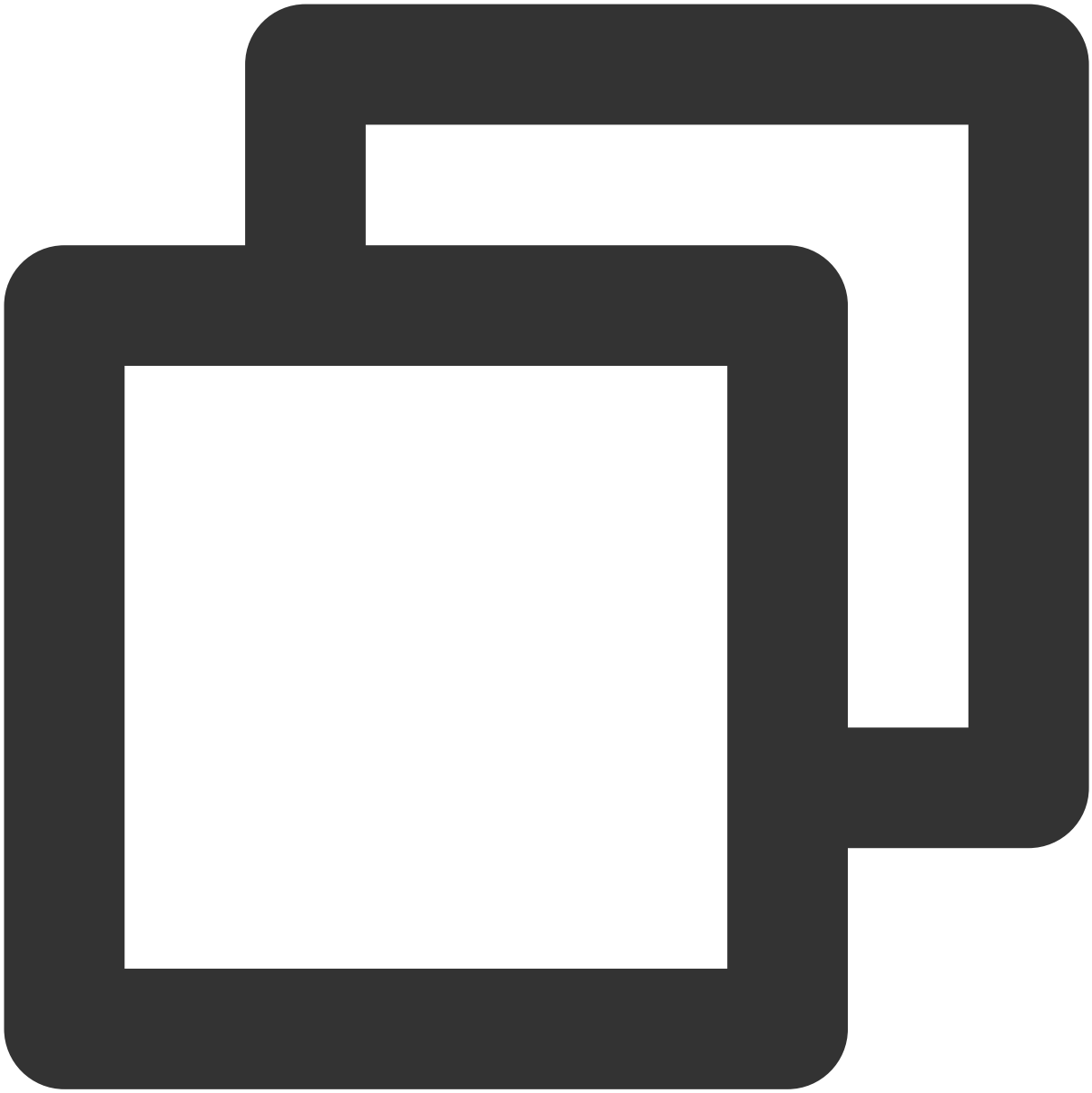
Last updated：2024-08-07 22:04:42

A recording rule allows you to calculate some commonly used or complex metrics in advance and then store the calculated data in new data metrics. In this way, querying the calculated data will be faster and easier than querying the original data. This is very suitable for dashboard scenarios and can solve the problems of complicated user configuration and slow query.
Recording rules exist in the form of rule group, and rules in the same group are executed sequentially at a certain interval. Rule names must conform to the corresponding Prometheus specification.
Generally, a rule file is as follows:



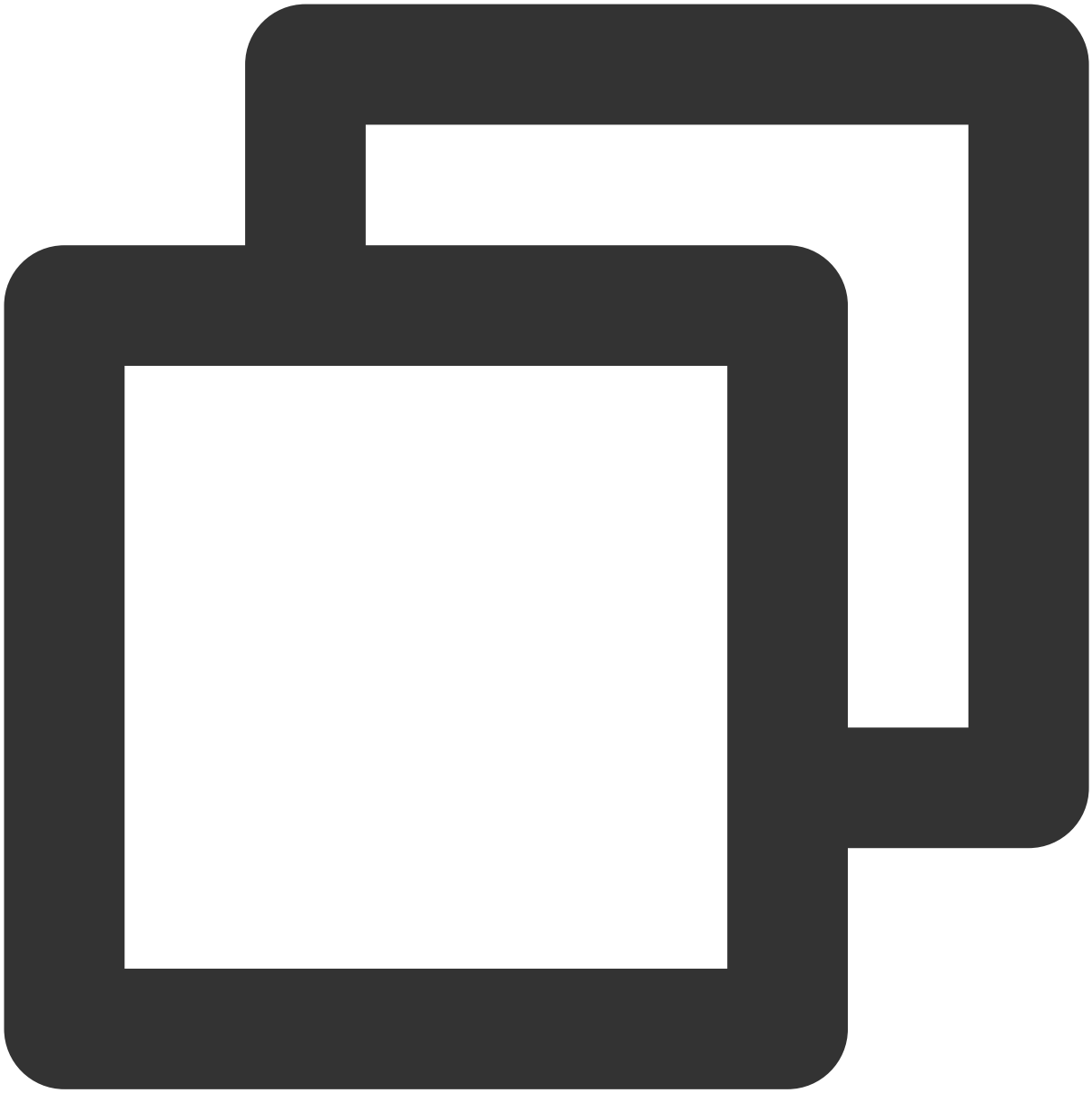
```
groups:
  [ - <rule_group> ]
```

Below is a simple example of recording rule:



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

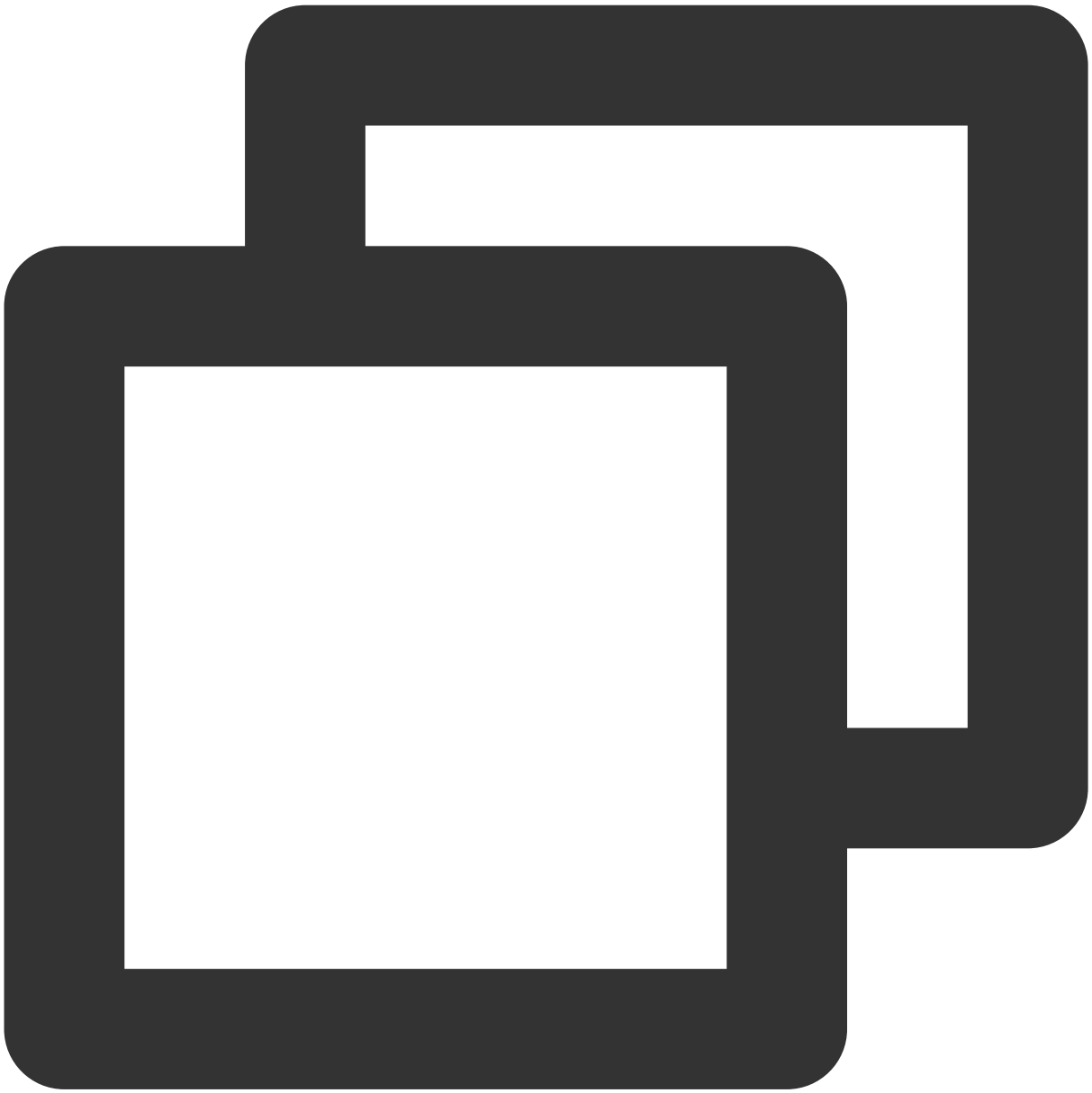
Rule Group



```
# A rule group name must be unique in the same file
name: <string>

# Rule detection interval
[ interval: <duration> | default = global.evaluation_interval ]

rules:
  [ - <rule> ... ]
```

Rule

The recording rule syntax is as follows:



```
# The generated new metric name, which must be valid
record: <string>

# PromQL expression. Each calculated data entry will be stored in the new metric na
expr: <string>

# The label to be added or overwritten in the data to be stored
```

```
labels:
  [ <labelname>: <labelvalue> ]
```

Recommended Name Format

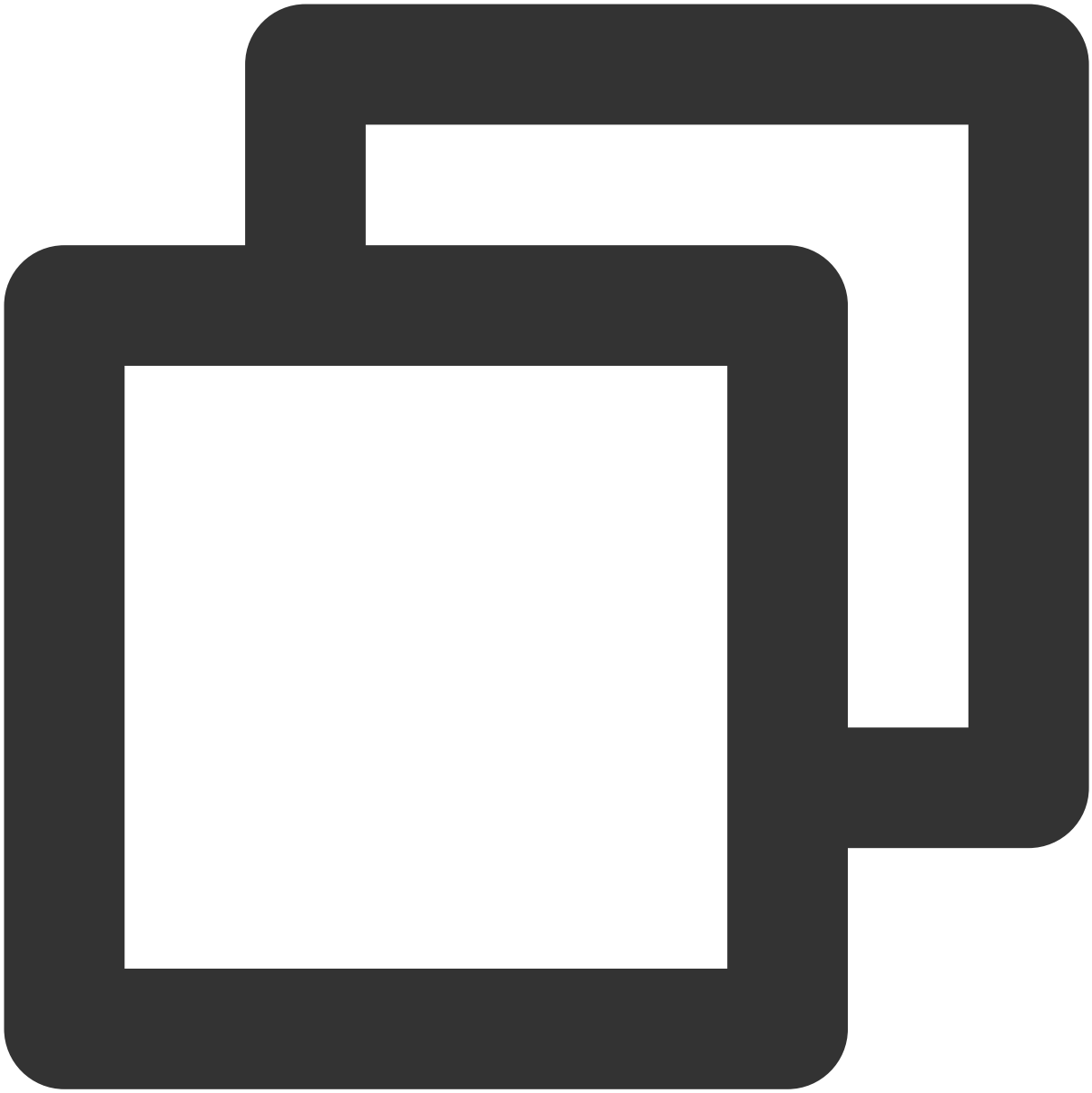
The recommended format for naming recording rules is `level:metric:operations` .

level: indicates the recording level and the output label of the rule.

metric: indicates the metric name.

operations: indicates the list of operations applied to the metric.

Example:



```
- record: instance_path:requests:rate5m
  expr: rate(requests_total{job="myjob"}[5m])

- record: path:requests:rate5m
  expr: sum without (instance)(instance_path:requests:rate5m{job="myjob"})
```

Rule Management

Last updated：2024-08-07 22:04:47

Overview

You can manage the TMP recording rules in the TMP console to avoid the hassle of having to modify the configuration file in native Prometheus.

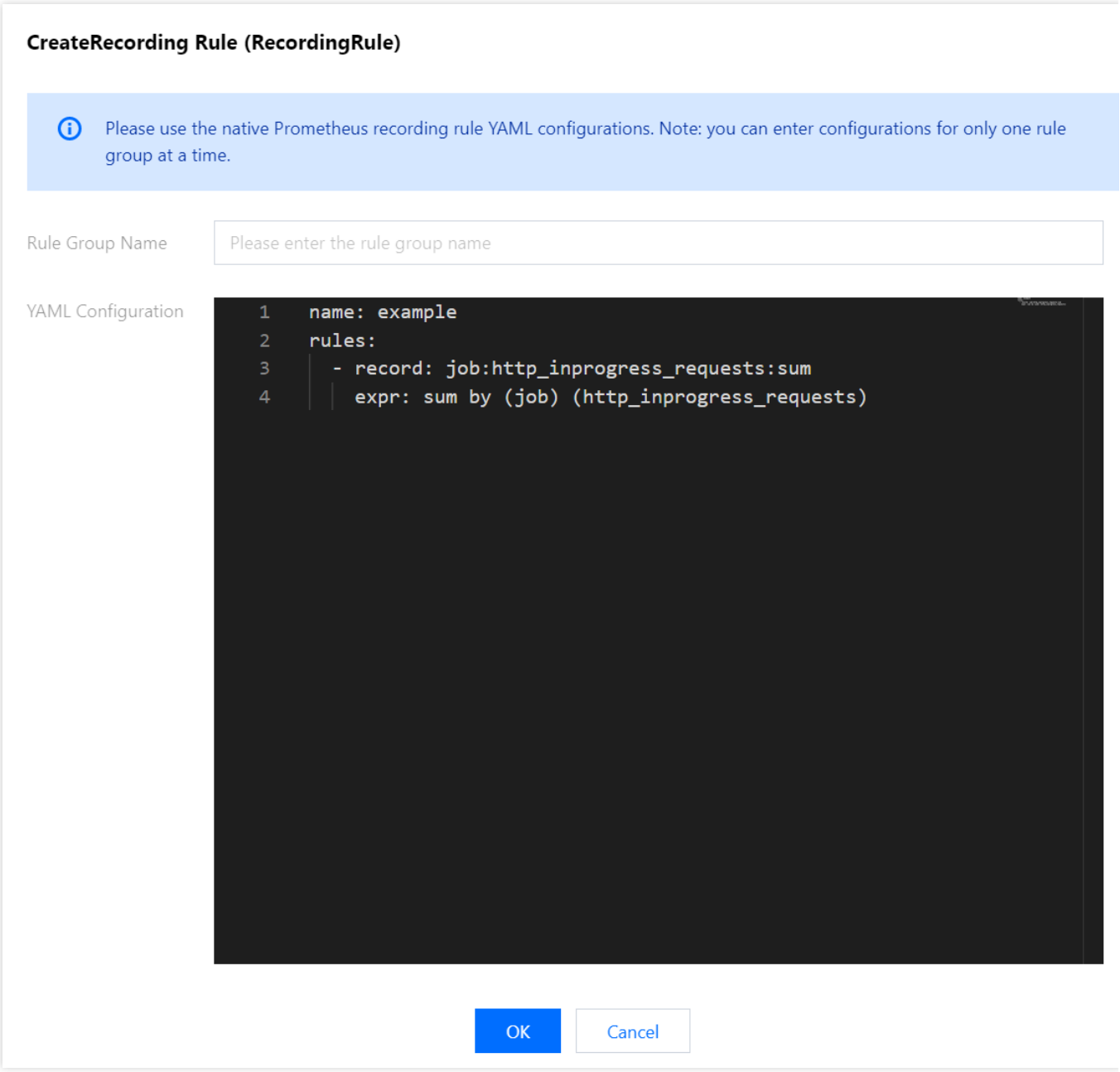
Preparations

1. Log in to the TMP console.
2. Create a TMP instance as instructed in Creating Instance.
3. Enter the TMP instance management page through the instance list.
4. Manage recording rules as instructed in Overview.

Directions

Creating rule

1. In the menu on the left of the instance management page, click **Recording Rule** > **Create** to enter the rule creation page, adjust the rule expression and the name of the new metric to be recorded according to your actual needs as shown below. For specific terms, please see Overview.



2. Click **OK**.

Managing rule

In the rule list, you can temporarily **disable** rules or enable rules that are **not enabled**. Once disabled, a rule will stop working, and the collection of related recording metrics will also stop.

Deleting rule

1. You can delete rules that are no longer used.
2. Select the rule to be deleted in the list and confirm in the pop-up window. Once deleted, a rule will stop working.

List of Default Recording Rules

Last updated：2024-08-07 22:04:52

Recording rules are created for associated clusters by default. For free metrics in pay-as-you-go mode, the following new metrics will be created after recording and will be billed normally. If you don't need them for data collection, you can disable the default recording rules when you associate the cluster for the first time, or later in the recording rule list on the recording page.

Metric	Preset Dashboard	A F T
:node_memory_MemAvailable_bytes:sum	Kubernetes / Compute Resources / Cluster	-
node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate	Kubernetes / Compute Resources / ClusterKubernetes / Compute Resources / WorkloadKubernetes / Compute Resources / PodKubernetes / Compute Resources / Node (Pods)Kubernetes / Compute Resources / Namespace (Workloads)Kubernetes / Compute Resources / Namespace (Pods)Kubernetes / Networking / WorkloadKubernetes / Networking / Namespace (Workload)	-
node_namespace_pod_container:container_memory_working_set_bytes		-
node_namespace_pod_container:container_memory_rss		-
node_namespace_pod_container:container_memory_cache		-
node_namespace_pod_container:container_memory_swap		-
namespace_workload_pod:kube_pod_owner:relabel		-
namespace:kube_pod_container_resource_requests_memory_bytes:sum	-	K re
namespace:kube_pod_container_resource_requests_cpu_cores:sum	-	K re

Instance Diagnosis

Last updated：2024-08-07 22:04:57

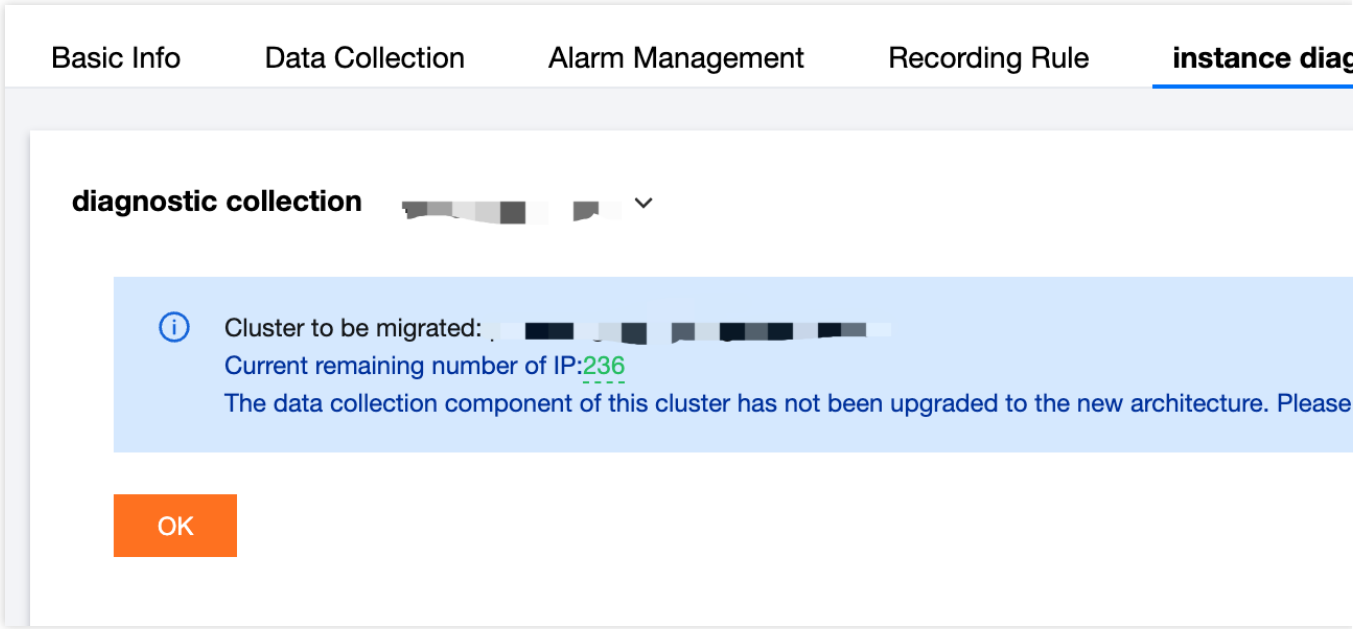
Background

To enhance the user experience with the Prometheus monitoring collection terminal, we now offer a new collection terminal architecture. The upgraded new architecture supports instance diagnosis, system health checks, and improves the resource utilization rate of the collection agent and the stability of metric collection. This document will guide users in upgrading the old collection architecture to the new one via the console instance diagnosis page and obtain detailed information about the current instance collection and storage for a better experience.

Directions

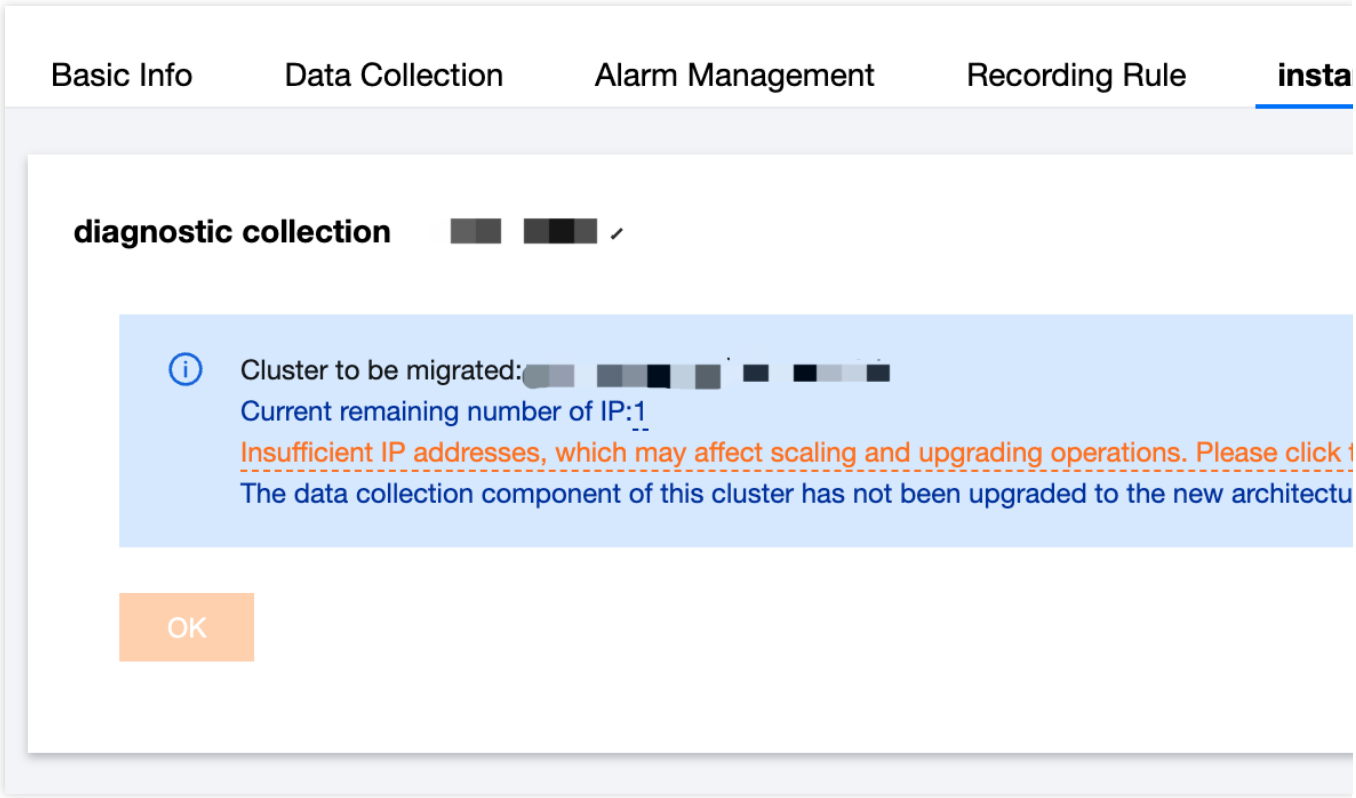
Upgrading to the New Architecture

1. Log in to TMP Console.
2. In the Prometheus instance list, click **Instance ID/Name**.
3. In the Prometheus management center, click **instance diagnostics** on the top navigation bar, and select the corresponding **collection cluster**.
Click **OK** to upgrade to the new architecture.



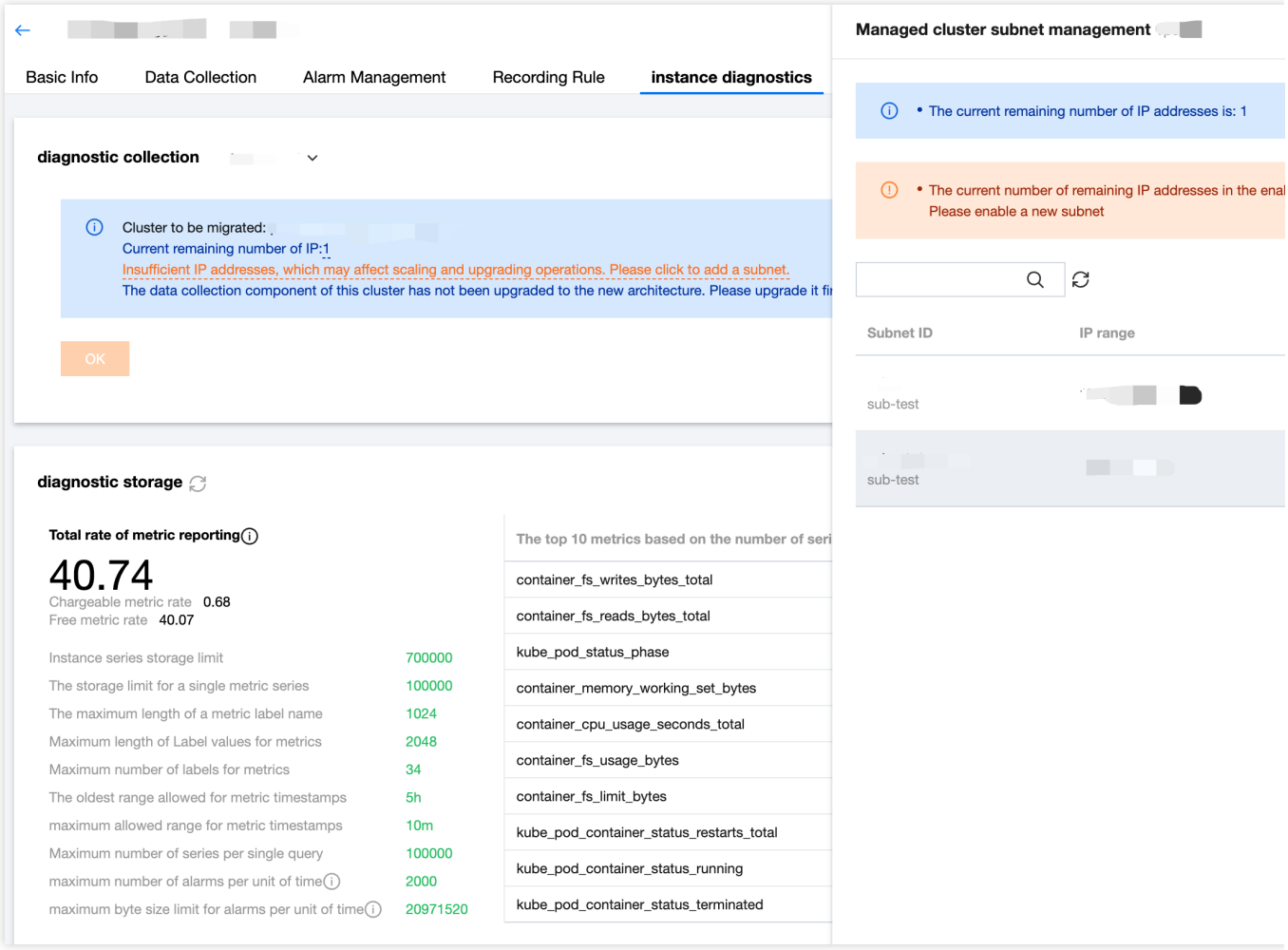
Note:

The upgrade process is expected to take 5 minutes and may experience a 1-2 minute metric interruption.
If the number of IPs in the managed cluster of the instance is less than 10, a risk warning will appear. To prevent the issue of insufficient IPs from causing component upgrade failures, please increase the available IPs as guided below.



Adding Subnet

1. On the instance diagnosis page, click subnet information to enter the **Managed Cluster Subnet Management** page.



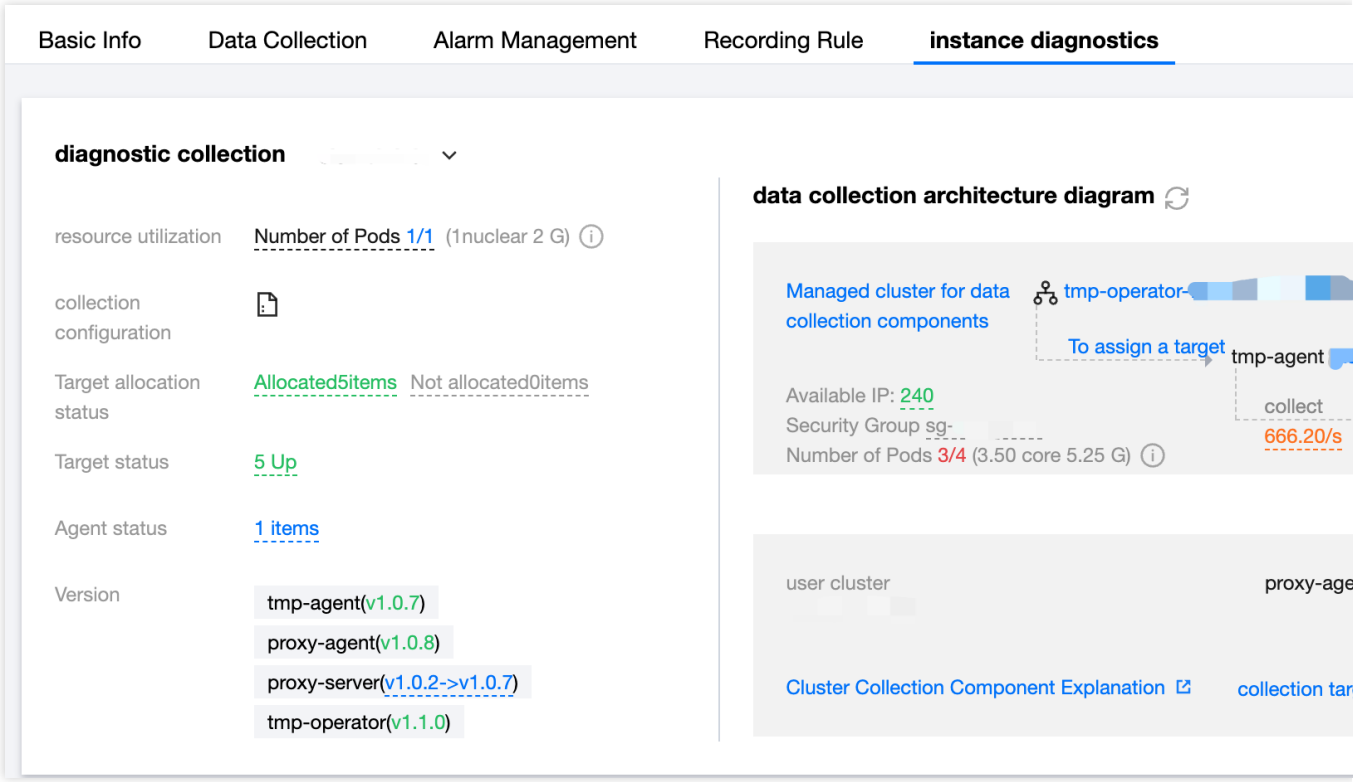
2. The page displays the subnets that have already been added to the managed cluster and those that have not in the current VPC. You can click **Enable Subnet** to enable subnets with sufficient remaining IPs as per your plan.
3. If there are no available subnets, please Add Subnet first and then enable it on the current page.

Instance Diagnosis

After the architecture is upgraded, the instance diagnosis page will include content on both collection and storage, helping users understand the operating status of Prometheus collection and storage to locate issues more quickly.

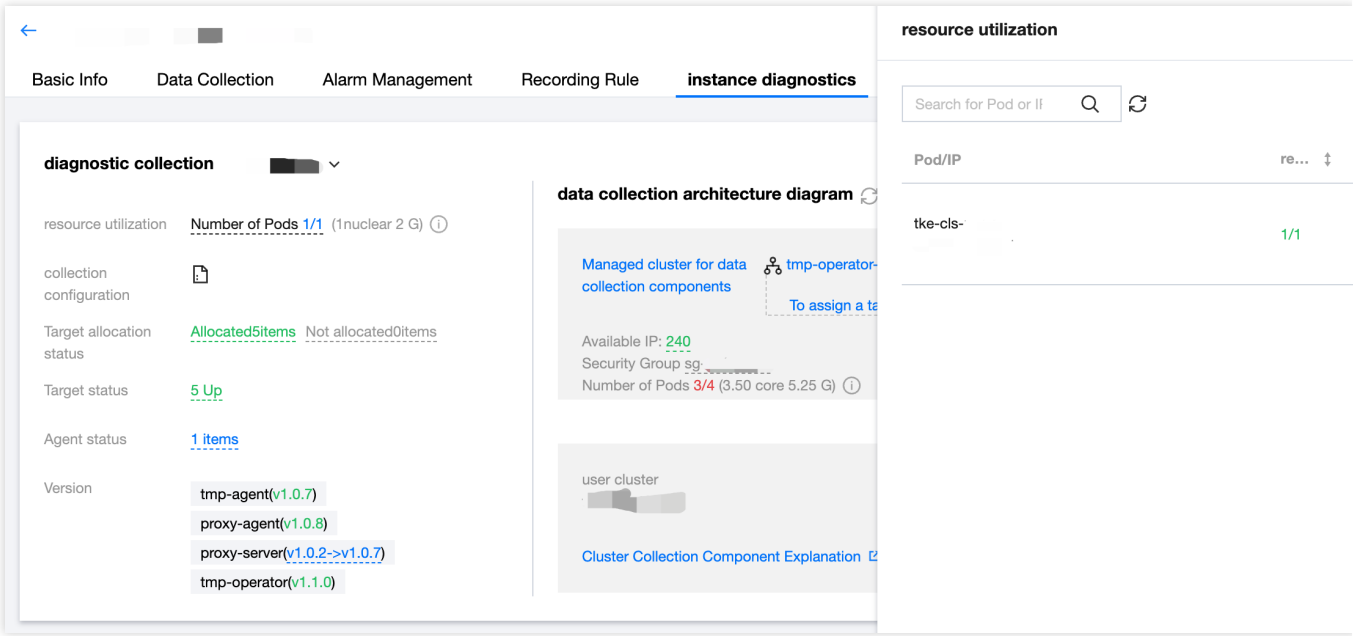
Diagnostic Collection

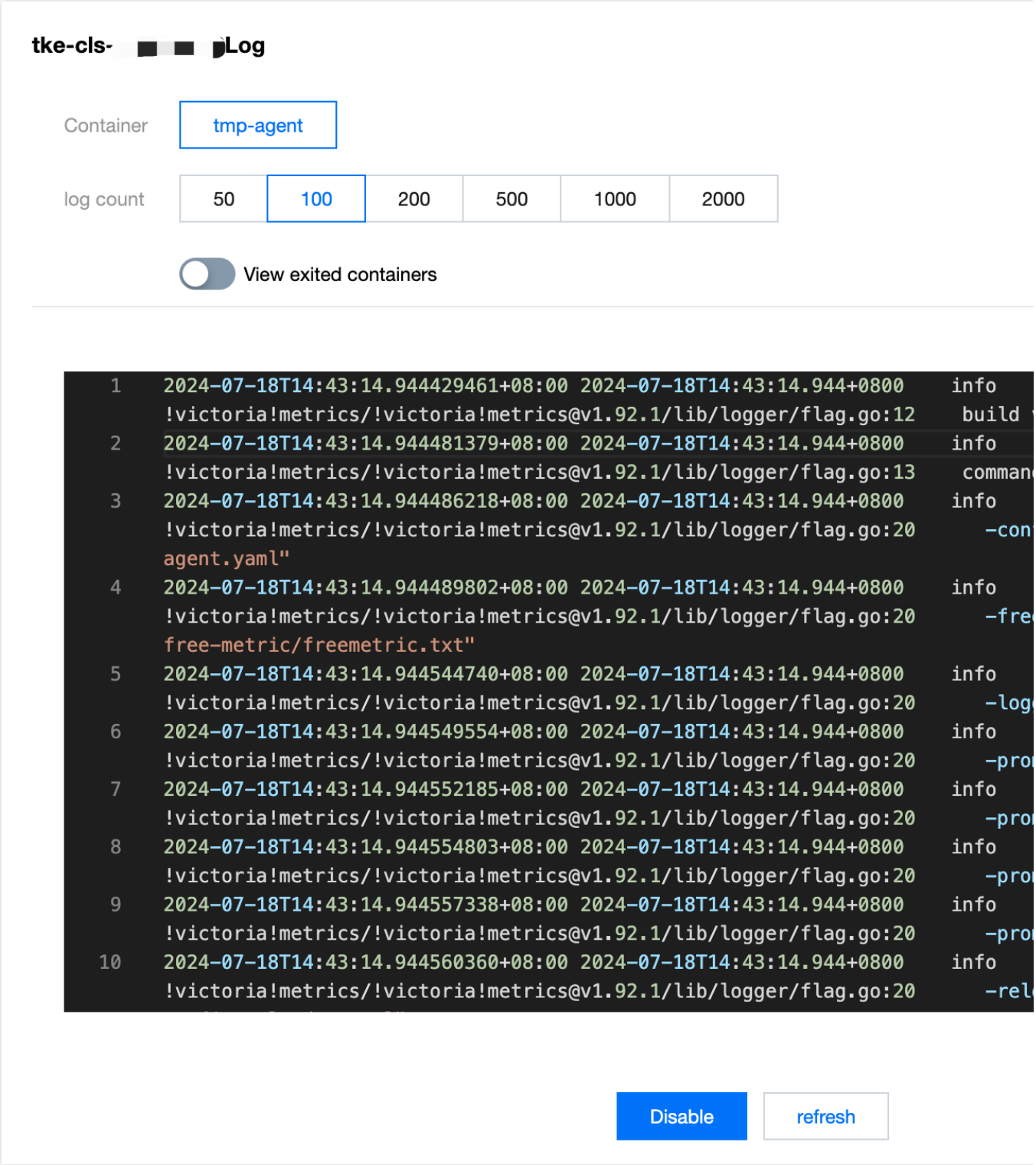
As shown in the following figure, collection diagnosis includes resource occupancy of the corresponding collection, collection configuration, target allocation status, target status, agent status, component version, and the collection architecture diagram.



Resource Utilization

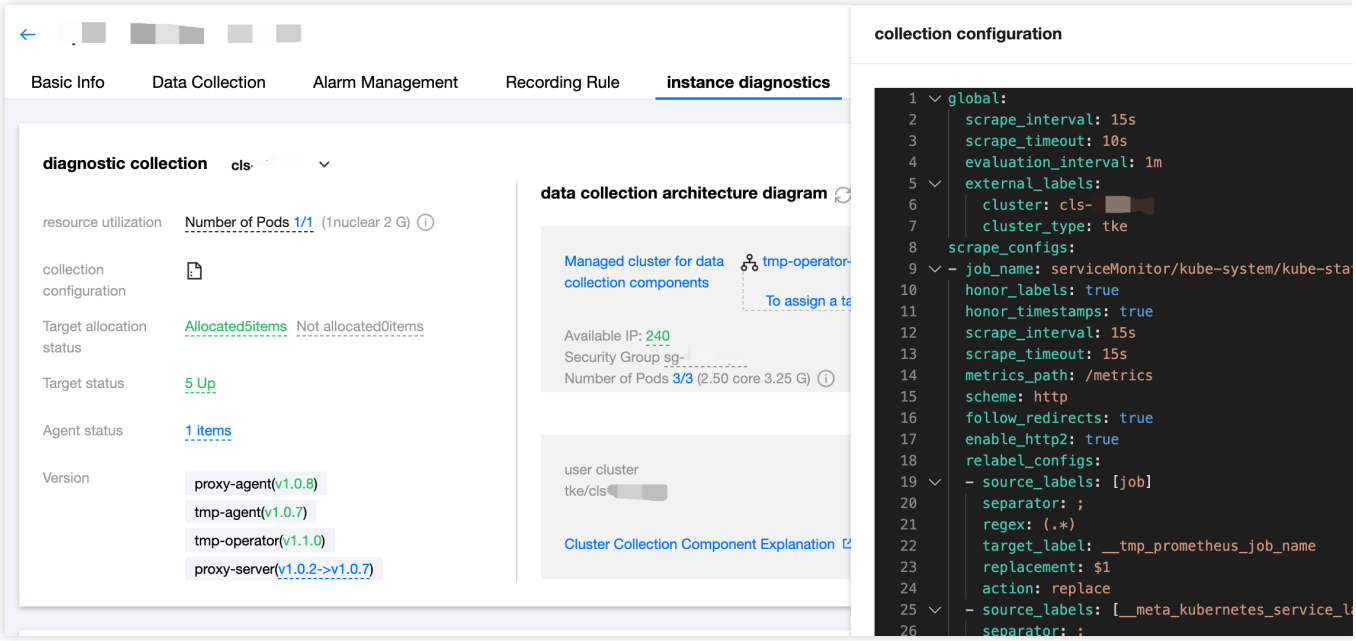
Resource occupancy of the collection shard displays information including CPU, memory limit and occupancy, and inbound and outbound traffic. Click **View logs** to see the logs of the Pod, facilitating the review of running details and troubleshooting of exceptions.





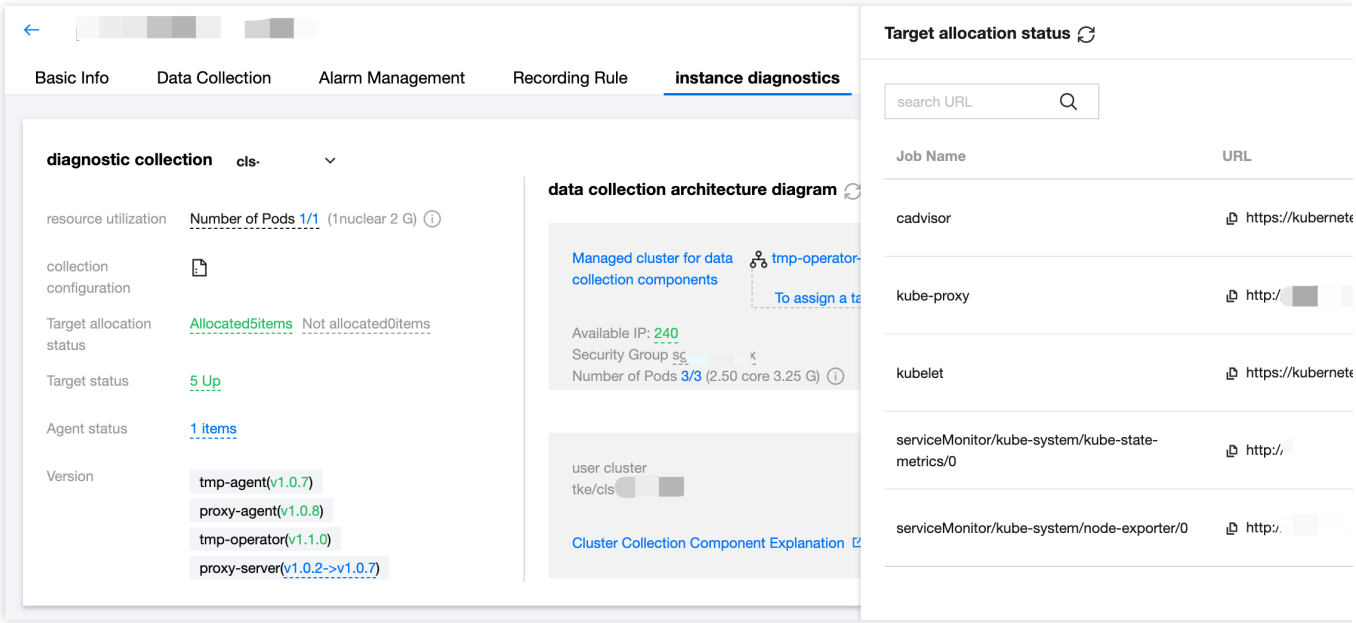
Collection Configuration

It displays the specific collection configuration of the current collection.



Target Allocation Status

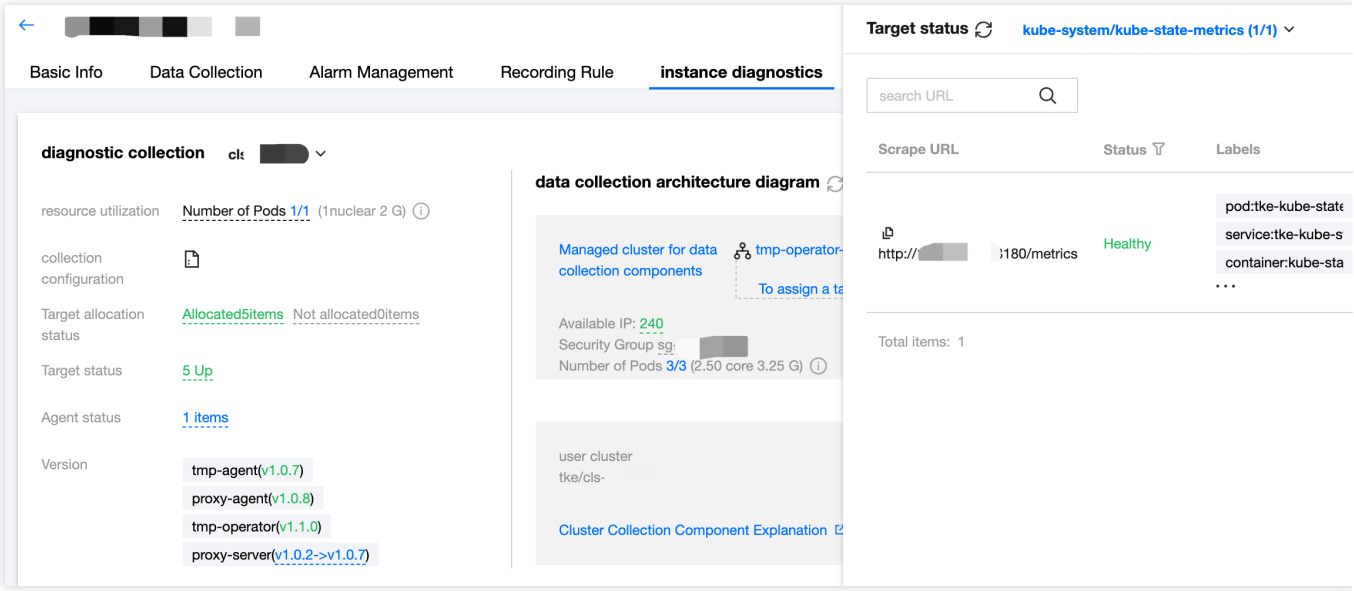
It shows the URL of the collection target, the name of the collection job to which the target belongs, and which collection shard is currently collecting it.



Target Status

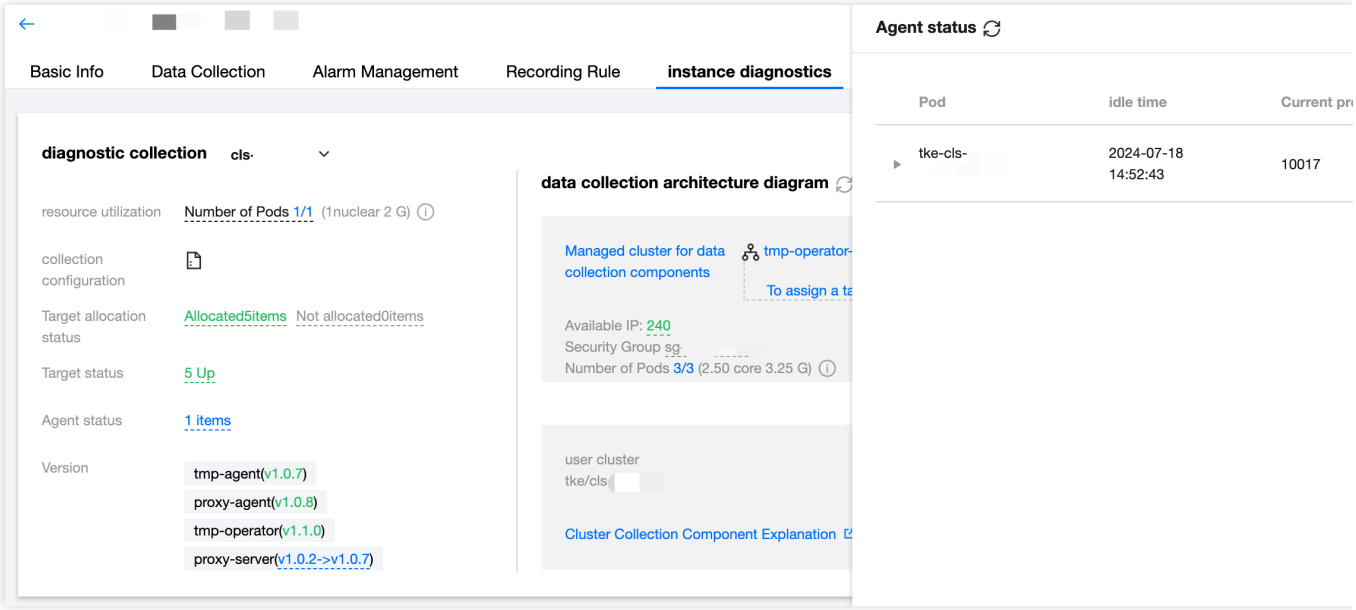
You can filter active collection targets based on the collection job, and obtain information about the corresponding target such as status, Labels, and Discovered Labels. The target status is "Healthy" in normal conditons, but if it is "abnormal" and there has already been the last scrape time, you can troubleshoot based on the error message on the

far right. Common issues may include the target itself being unhealthy, incorrect permission configuration, and network errors.



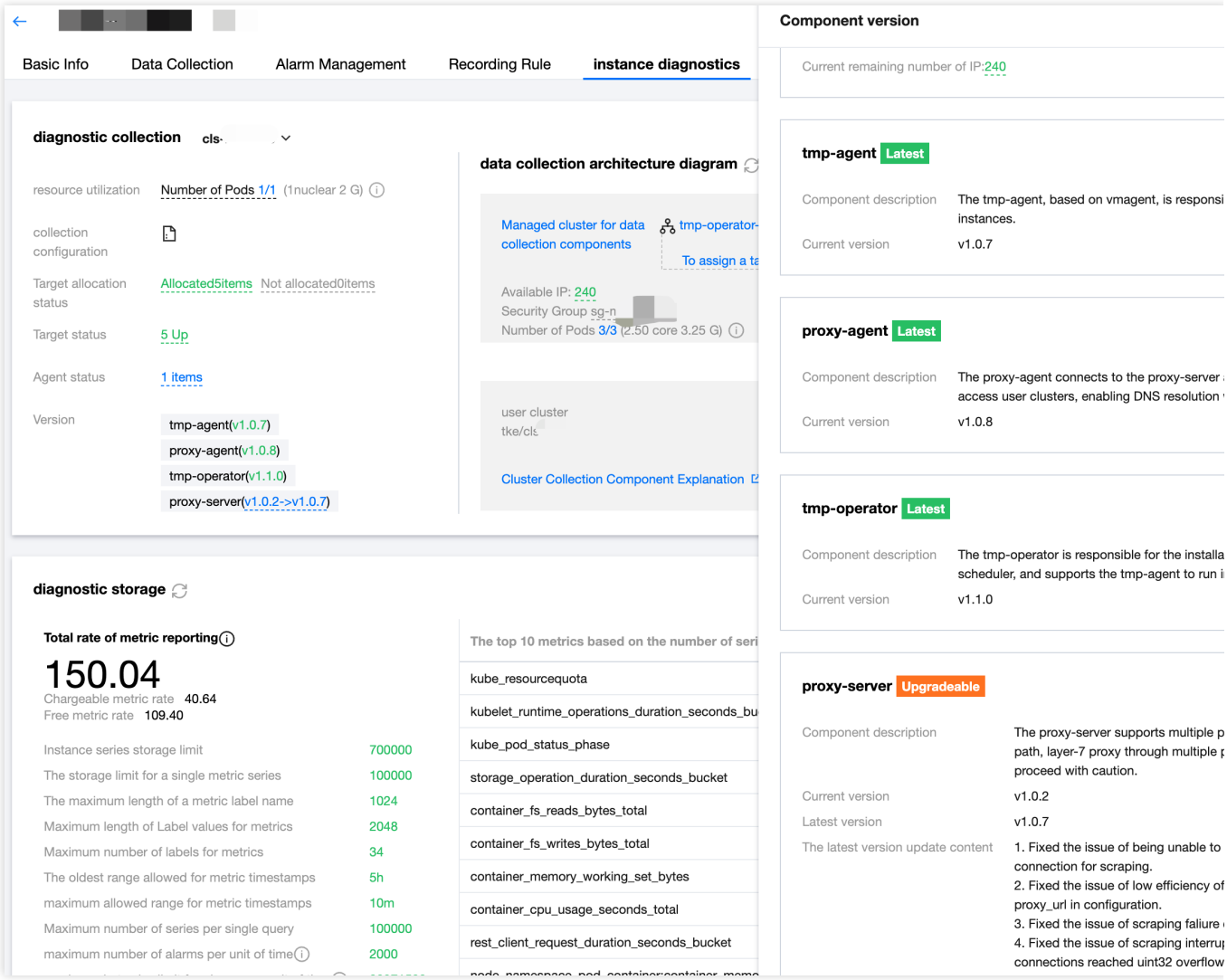
Agent Status

It shows the running status of the collection shard agent. Click **View logs** to see the logs of the Pod to understand running details and troubleshoot exceptions.



Component Version

It shows the component version information of the current collection. On the **Component version** page, it displays IP quantity check and the current version, latest version, and component description, upgrade description of each component version.



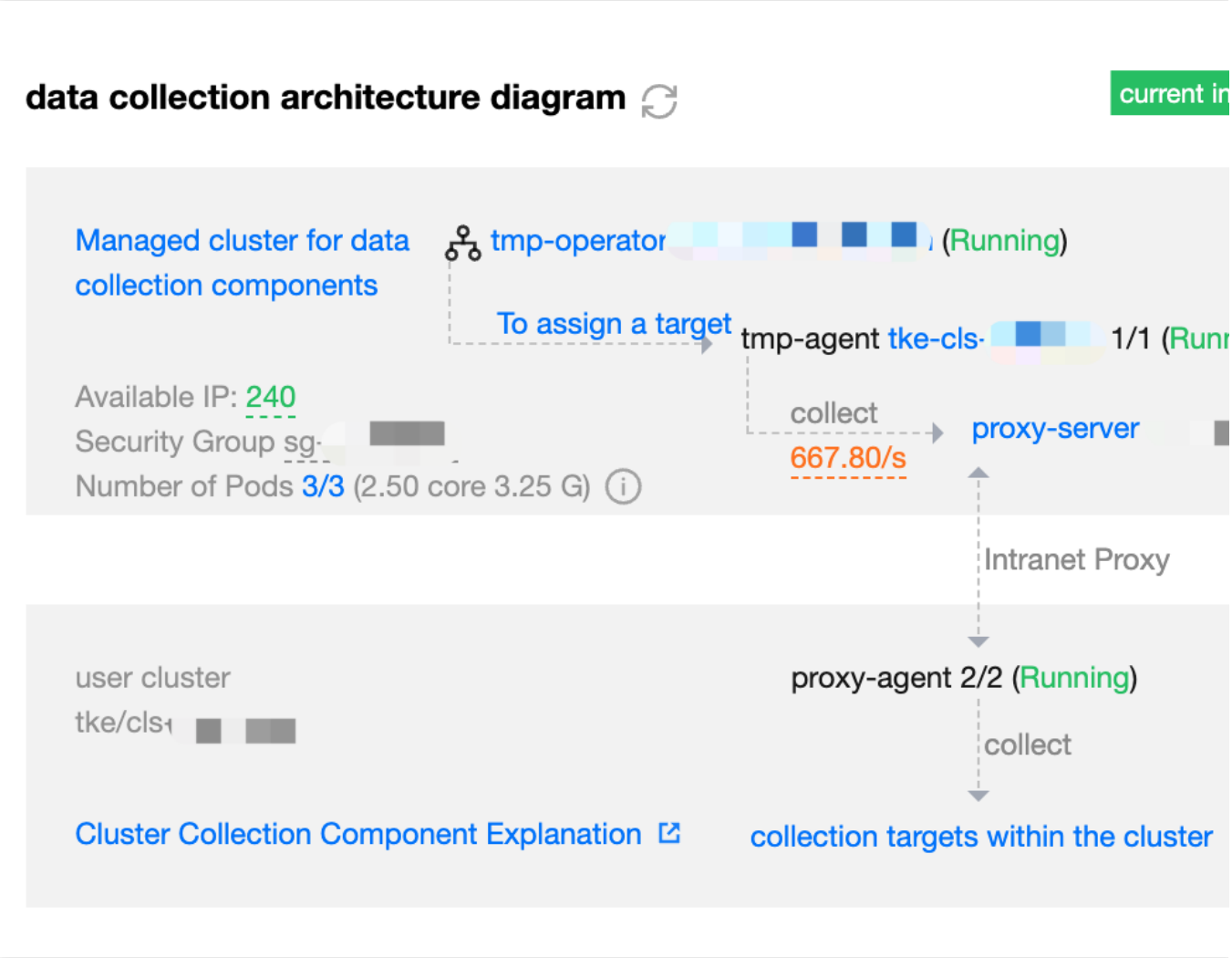
Note:

Please try to keep the collection component in the **latest version**, which can be upgraded through the **Upgrade** of the corresponding component.

Components tmp-operator, tmp-agent, and proxy-agent can be upgraded without impact under normal circumstances. During the upgrade of the proxy-server component, there will be collection breakpoints. The breakpoint duration is the time for eks to activate the component Pod, and it will affect the collection of the entire Prometheus instance (including the integration center and container clusters). Please operate cautiously.

Data Collection Architecture Diagram

The collection architecture diagram provides information about the current collection architecture.



Collection Component Managed Cluster:

Available IP count. When the number of IPs is insufficient, an **Insufficient IP Count** alert will appear. Click to enter the **Managed Cluster Subnet Management** page. For specific operations, see Adding Subnet.

Managed cluster's security group and its pass-through requirements during normal operation. Some network issues during the collection may be caused by the security group not allowing pass-through.

The number of Pods and resource utilization related to the current collection

Running status of the collection scheduling component tmp-operator

Collection target allocation status

Running status of the collection shard component tmp-agent

Metric collection rate and inbound bandwidth

Running status of the proxy component proxy-server

Metric write rate and outbound bandwidth

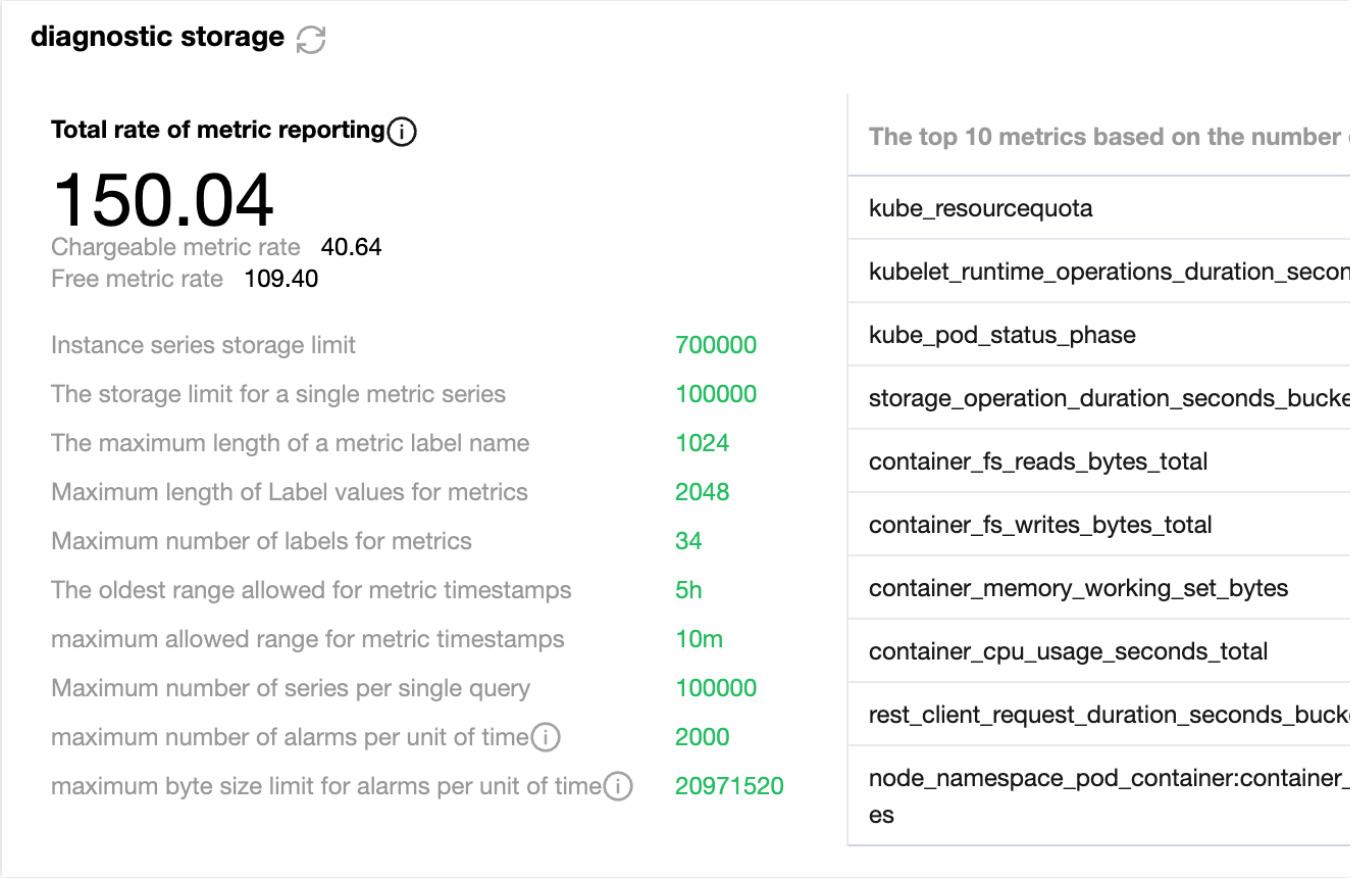
Write target status, including data overwrite of the current instance's corresponding storage and user configuration.

User Cluster:

Status of the public network proxy CLB (if enabled)
Running status of the proxy component proxy-agent;
Status of the collection target within the cluster

Storage Diagnosis

As shown in the figure, the storage diagnosis includes storage-related status and limitations.



Parameter Description

Parameter	Description
Total rate of metric reporting	Includes free metrics and paid metrics.
Instance series storage limit	The number of active series exceeding this value will cause the corresponding instance series discarded due to limit exceeded.
The storage limit for a single metric series	Different labels for the same metric name constitute different series. Exceeding this value will result in discard due to limit exceeded.
The maximum length of a metric label name	Exceeding this value will result in discard due to invalid length.

Maximum number of labels for metrics	Exceeding this value will result in discard due to invalid length.
The oldest range allowed for metric timestamps	Indicates the oldest timestamp acceptable in a single series (out of order not allowed).
maximum allowed range for metric timestamps	Indicates the latest timestamp acceptable in a single series (out of order not allowed).
Maximum number of series per single query	Indicates the maximum number of series involved in data query. It is recommended to shorten the range query time or use instant query in suitable scenarios.
Maximum number of alarms per unit of time	Indicates the maximum number of alarms triggered within 5 minutes.
Maximum byte size limit for alarms per unit of time	Indicates the total size limit of fields (such as lable and annotation) for alarms triggered within 5 minutes.
The top 10 metrics based on the number of series	Same metric name and different label keys are considered different series. Storage is subject to the upper limit for single metric series, and a large quantity can cause high cardinality issue.

Alerting Rule Overview

Last updated：2024-08-07 22:05:10

TMP allows you to define alert conditions based on Prometheus expressions. Once a metric reaches an alert condition, you will receive notifications through email and SMS, so you can take corresponding measures promptly.

Note:

Alerting in TMP combines the alarming capabilities of Cloud Monitor and the alerting capabilities of the open-source Prometheus ecosystem, making alerting more accurate and reasonable.

Features

TMP supports Alertmanager features such as grouping and silencing to make alerts more reasonable.

You can group the metric data and regularly check and calculate the alerting rules to make alerting quicker and more convenient.

Alerting rules can be defined based on PromQL, making alerting more flexible.

Cloud Monitor's alarm notification templates can be used, which support multiple receiving channels such as email and SMS.

Components

Term	Description
Rule name	Custom alerting rule name.
Alerting rule	It contains alert trigger condition and duration and should be defined through PromQL.
Alert object	Custom alert title.
Alert message	Custom alert content.
Label	A set of specified labels to be added to the alert.
Annotation	Custom additional alert message.
Notification template	It contains the template name, notification type, recipient, receiving channel. You can define the receiving channel and grouping method.

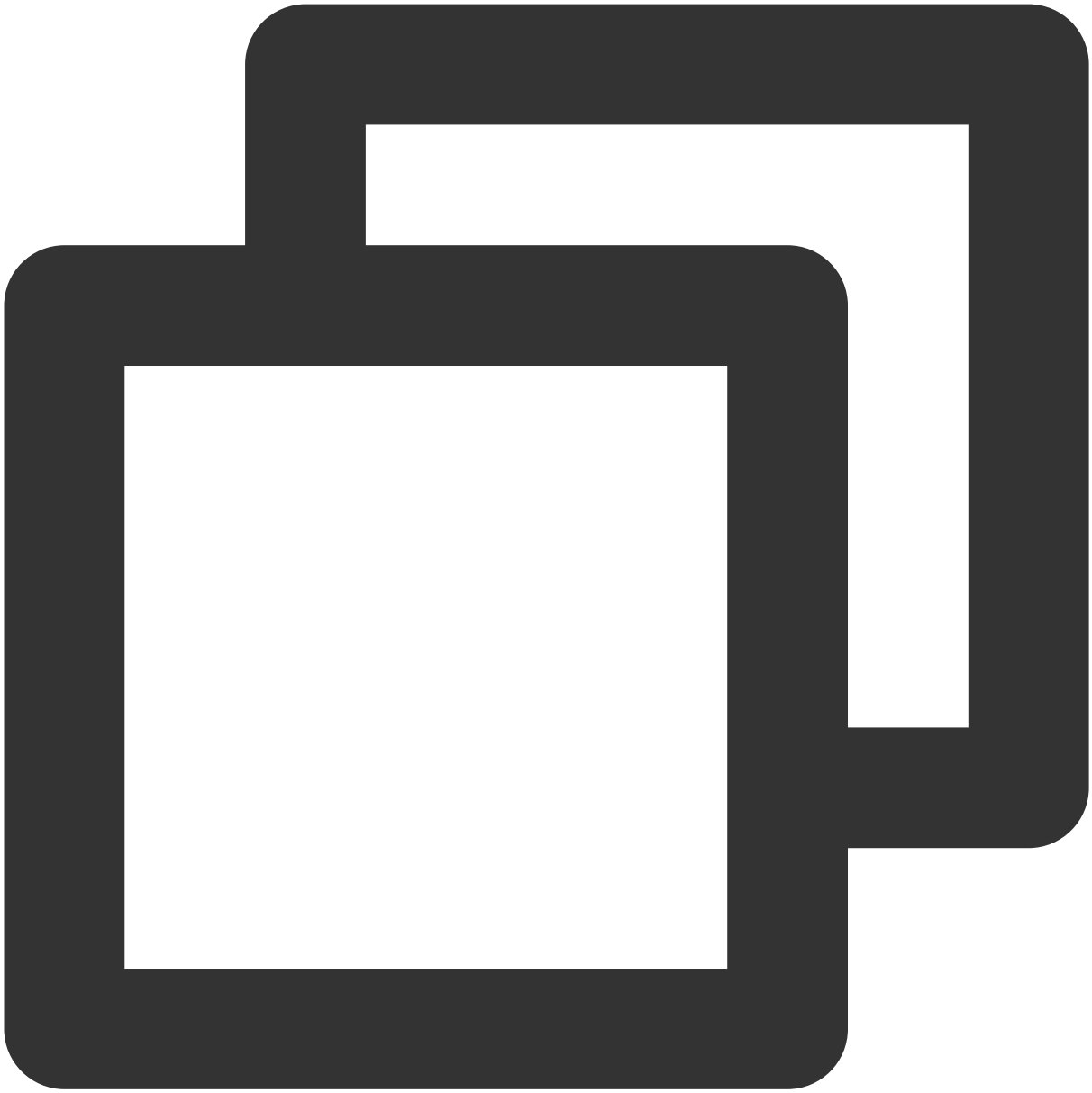
Alerting Rule Description

Last updated：2024-08-07 22:05:17

You can set alert conditions based on Prometheus expressions to monitor the service status in real time and receive prompt notifications when the service is exceptional.

Defining Alerting Rule

Defining an alerting rule in TMP is very similar to defining a recording rule. Below is a sample alerting rule:

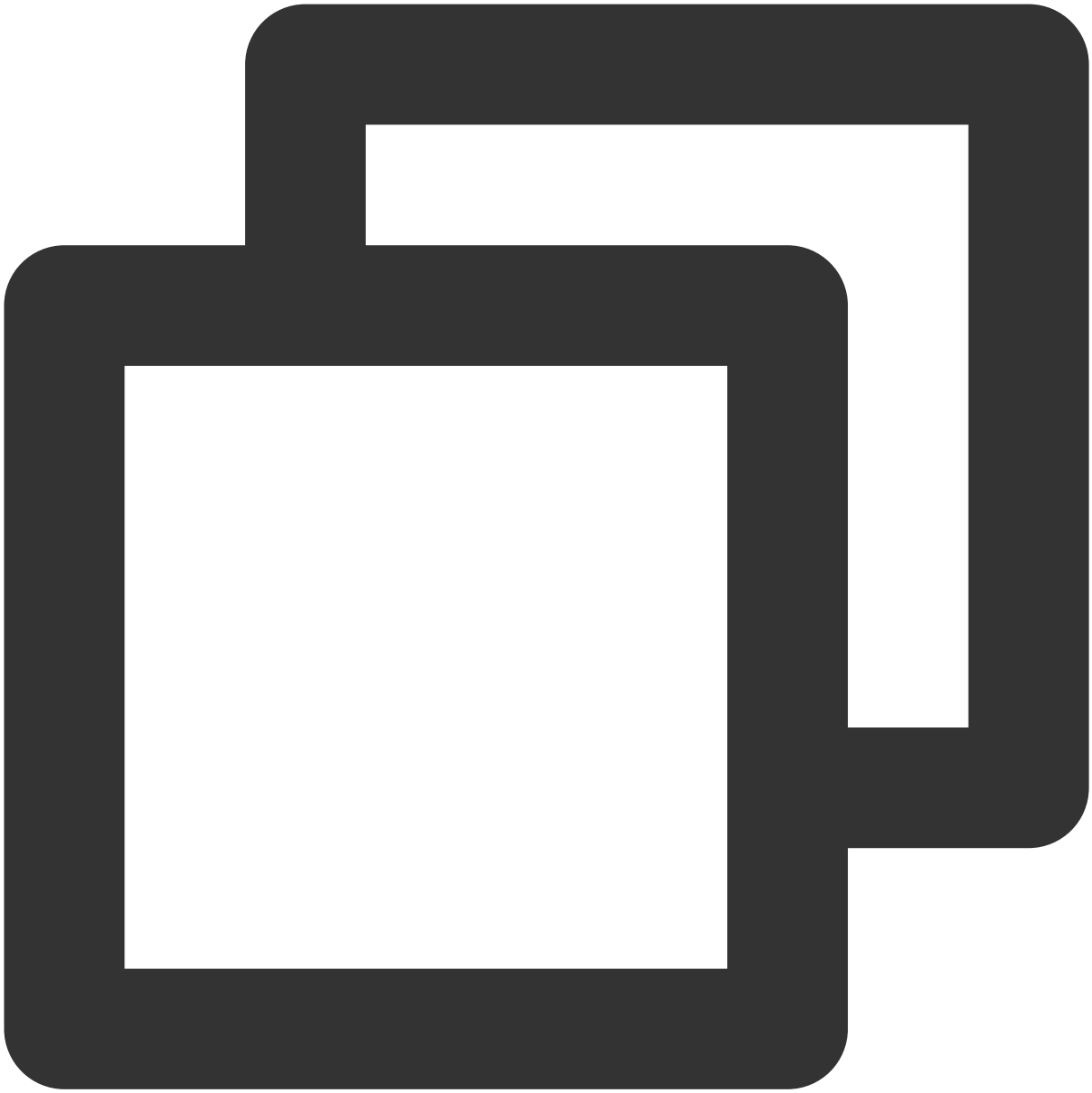


```
groups:
- name: example
  rules:
  - alert: HighRequestLatency
    expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5
    for: 10m
    labels:
      severity: page
    annotations:
      summary: High request latency
```

In an alerting rule file, you can define a set of relevant rules in the same group. In each group, you can define multiple alerting rules. A rule mainly consists of the following parts:
alert: alerting rule name.
expr: alert trigger condition based on a PromQL expression, which is used to calculate whether there is time series data meeting the condition.
for: assessment wait time, which is optional. It indicates how long a trigger condition can last before an alert is sent. New alerts generated during the wait time are in "Pending" status.
labels: custom labels, which are a set of specified labels to be added to alerts.
annotations: it is used to specify a set of additional information, such as text that describes alert details. It will be sent to Alertmanager as a parameter when an alert is generated.

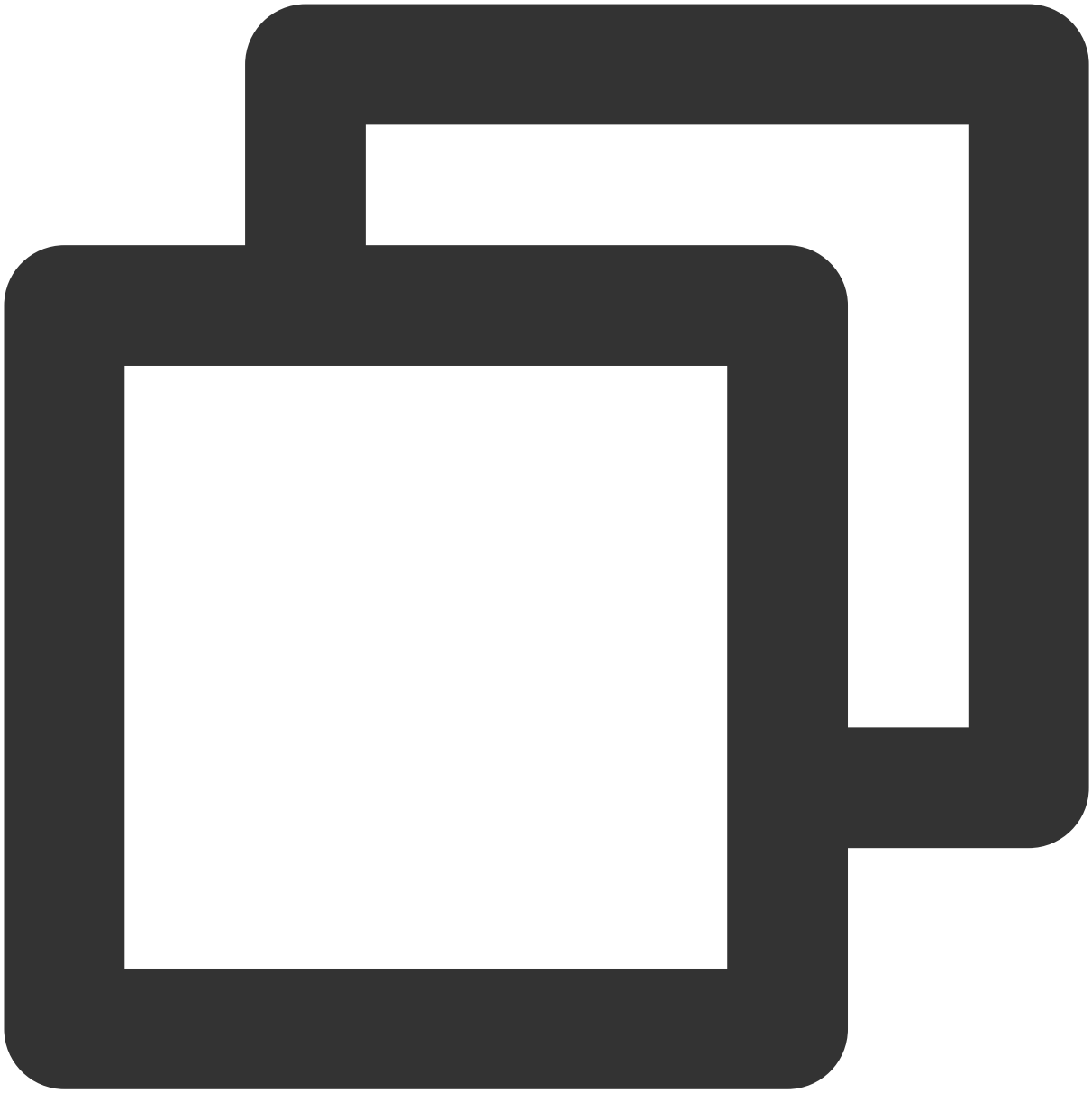
Template

Generally, `annotations` in an alerting rule file uses `summary` to describe the summary of alerts and `description` to describe alert details. In addition, Alertmanager UI will also display the alert information based on the two label values. To make the alert information more readable, TMP allows you to convert label values in `labels` and `annotations` into a template.
You can use the `$labels.<labelname>` variable to access the value of the specified label on the current alert instance and use `$value` to get the sample value calculated through the current PromQL expression.



```
# To insert a firing element's label values:
{{ $labels.<labelname> }}
# To insert the numeric expression value of the firing element:
{{ $value }}
```

For example, you can use a template to optimize the readability of the content of `summary` and `description`:



```
groups:
- name: example
  rules:

  # Alert for any instance that is unreachable for >5 minutes.
  - alert: InstanceDown
    expr: up == 0
    for: 5m
    labels:
      severity: page
    annotations:
```

```
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down f

  # Alert for any instance that has a median request latency >1s.
  - alert: APIHighRequestLatency
    expr: api_http_request_latencies_second{quantile="0.5"} > 1
    for: 10m
    annotations:
      summary: "High request latency on {{ $labels.instance }}"
      description: "{{ $labels.instance }} has a median request latency above 1s (c
```

Creating Alerting Rule

Last updated：2024-08-07 22:05:23

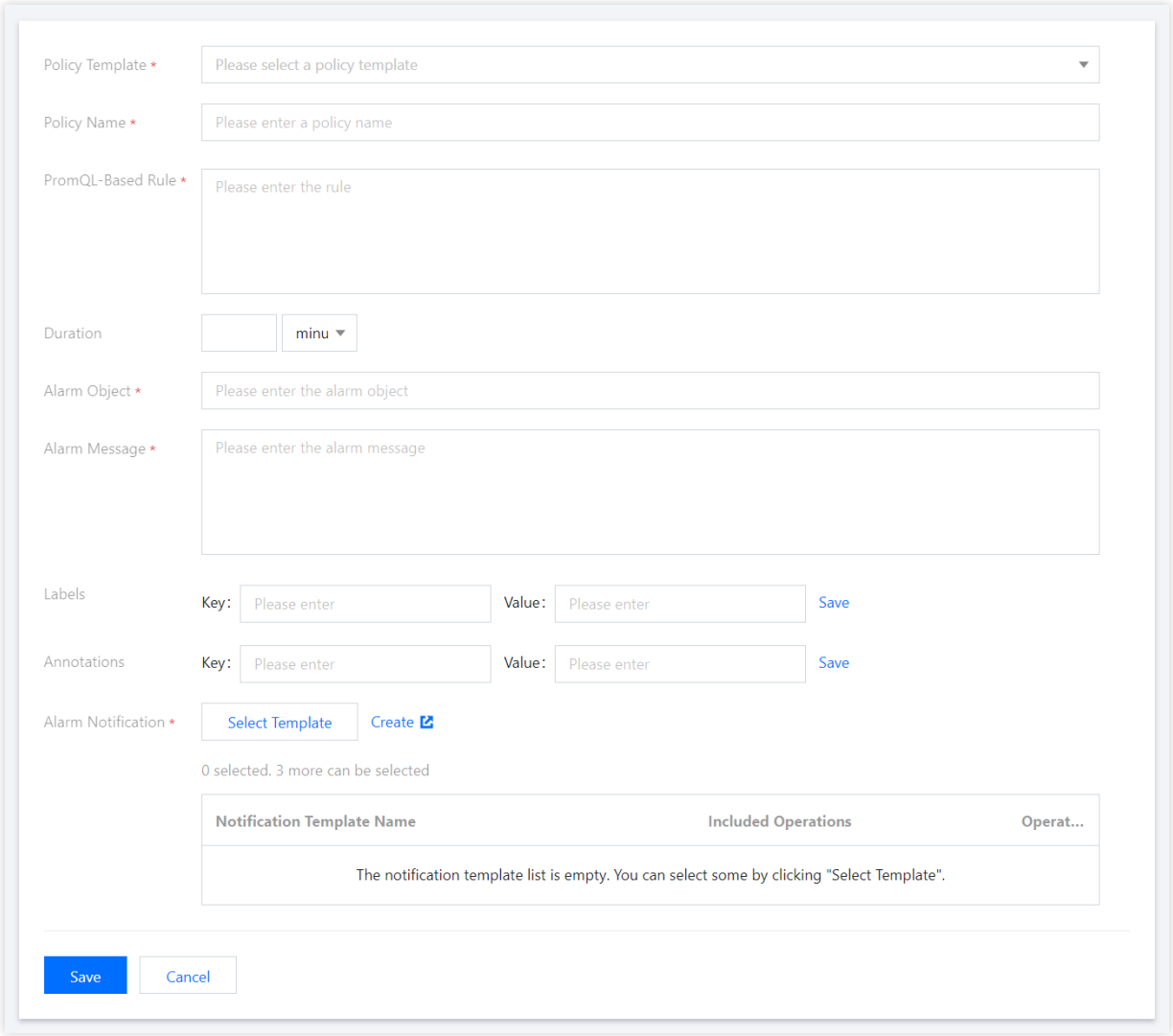
This document describes how to create an alerting rule in the TMP console. With such a rule, you will receive notifications when some metrics are exceptional, so that you can take corresponding measures promptly.

Prerequisites

You have created a managed TKE cluster.
You have created a TMP instance.
You have installed the Prometheus agent and other monitoring components.

Directions

For more information on alerting rule, please see Alerting Rule Description.
1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance and click **Alerting Rule** on the left.
3. On the alerting rule management page, click **Create** and configure the alerting rule information in the pop-up window.
Rule Template Type: select a rule template type. For more information, please see Alerting Rule Type Description.
Rule Name: you can use the default rule name or customize it.
PromQL-Based Rule: you can use the default value or customize it. It indicates an alert trigger condition based on a PromQL expression, which is used to calculate whether there is time series data meeting the condition.
Duration: you can use the default value or customize it. It indicates how long a trigger condition can last before an alert is sent.
Alert Object: you can customize the alert title.
Alert Message: you can use the default value or customize it. It indicates the custom alert content.
Advanced Configuration: you can toggle it on for configuration, which contains configuration items for labels and annotations.
Labels: you can use the default value or customize it. It indicates a set of specified labels to be added to alerts. You can match the corresponding processing method based on the labels of a received alert.
Annotations: you can use the default value or customize it. It indicates the user-defined additional alert information.
Alert Notification: you can customize the alert notification template, which contains the template name, notification type, recipient, and receiving channel. For more information, please see Notification Template.



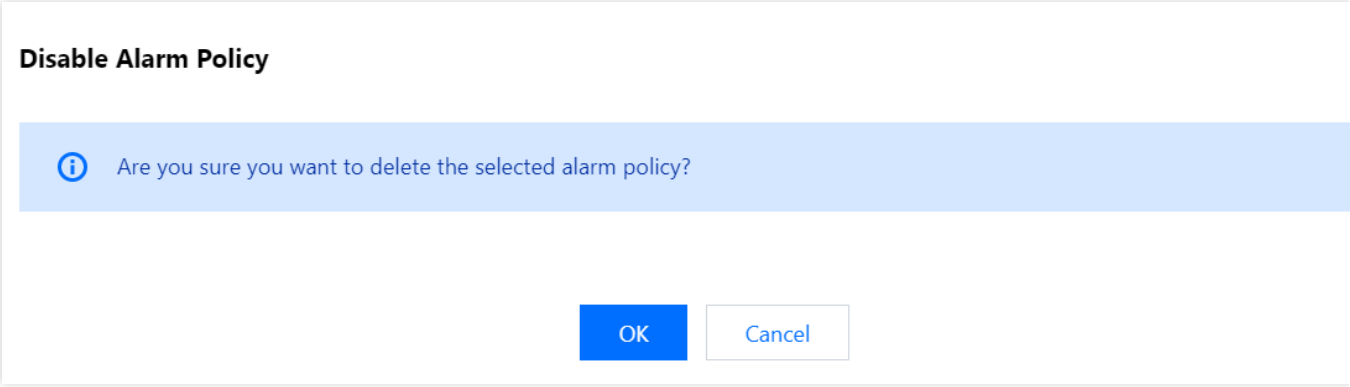
Disabling Alerting Rule

Last updated：2024-08-07 22:05:28

This document describes how to disable an alerting rule on the target instance in the TMP console.

Directions

1. Log in to the TMP console.
2. In the instance list, select the corresponding TMP instance and click **Alerting Rule** on the left.
3. Find the alerting rule to be disabled and click **Disable** in the **Operation** column.
4. In the pop-up window, click **OK**.



Rule Type Description(old)

Last updated：2024-08-07 22:05:33

TMP presets the master component, kubelet, resource use, workload, and node alert templates for TKE clusters.

Kubernetes master component

The following metrics are provided for non-managed clusters:

Rule Name	Rule Expression	Duration	Desc
Error with client access to APIServer	(sum(rate(rest_client_requests_total{code=~"5.."}[5m])) by (instance, job, cluster_id) / sum(rate(rest_client_requests_total[5m])) by (instance, job, cluster_id))> 0.01	15m	The rate acce the APIS abov
Imminent expiration of the client certificate for APIServer access	apiserver_client_certificate_expiration_seconds_count{job="apiserver"} > 0 and on(job) histogram_quantile(0.01, sum by (cluster_id, job, le) (rate(apiserver_client_certificate_expiration_seconds_bucket{job="apiserver"}[5m]))) < 86400	None	The certif for APIS acce expir hours
Recording API error	sum by(cluster_id, name, namespace) (increase(aggregator_unavailable_apiservice_count[5m])) > 2	None	The recor API repor error last 5 minu
Low recording API availability	(1 - max by(name, namespace, cluster_id) (avg_over_time(aggregator_unavailable_apiservice[5m]))) * 100 < 90	5m	The avail of the recor API s in the

			minu belov
APIServer fault	absent(sum(up{job="apiserver"}) by (cluster_id) > 0)	5m	APIS disap from colle targe
Scheduler fault	absent(sum(up{job="kube-scheduler"}) by (cluster_id) > 0)	15m	The sche disap from colle targe
Controller manager fault	absent(sum(up{job="kube-controller-manager"}) by (cluster_id) > 0)	15m	The contr mana disap from colle targe

Kubelet

Rule Name	Rule Expression	Duration	Des
Exceptional node status	kube_node_status_condition{job=~".*kube-state-metrics",condition="Ready",status="true"} == 0	15m	The stat exc for min
Unreachable node	kube_node_spec_taint{job=~".*kube-state-metrics",key="node.kubernetes.io/unreachable",effect="NoSchedule"} == 1	15m	The unre and wor be sch aga
Too many Pods	count by(cluster_id, node) ((kube_pod_status_phase{job=~".*kube-state-metrics",phase="Running"} == 1) * on(instance,pod,namespace,cluster_id)	15m	The of P

running on node	group_left(node) topk by(instance,pod,namespace,cluster_id) (1, kube_pod_info{job=~".*kube-state-metrics"}))/max by(cluster_id, node) (kube_node_status_capacity_pods{job=~".*kube-state-metrics"} != 1) > 0.95		runr the clos upp
Node status fluctuation	sum(changes(kube_node_status_condition{status="true",condition="Ready"} [15m])) by (cluster_id, node) > 2	15m	The stat fluc betv norr exc
Imminent expiration of the kubelet client certificate	kubelet_certificate_manager_client_ttl_seconds < 86400	None	The clie cert will 24 h
Imminent expiration of the kubelet server certificate	kubelet_certificate_manager_server_ttl_seconds < 86400	None	The serv cert will 24 h
Kubelet client certificate renewal error	increase(kubelet_certificate_manager_client_expiration_renew_errors[5m]) > 0	15m	An occ whil rene kub cert
Kubelet server certificate renewal error	increase(kubelet_server_expiration_renew_errors[5m]) > 0	15m	An occ whil rene kub serv cert
Time-Consuming PLEG	histogram_quantile(0.99, sum(rate(kubelet_pleg_relist_duration_seconds_bucket[5m])) by (cluster_id, instance, le) * on(instance, cluster_id) group_left(node) kubelet_node_name{job="kubelet"}) >= 10	5m	The perc PLE ope dura exc sec

Time-Consuming Pod start	histogram_quantile(0.99, sum(rate(kubelet_pod_worker_duration_seconds_bucket{job="kubelet"}[5m])) by (cluster_id, instance, le)) * on(cluster_id, instance) group_left(node) kubelet_node_name{job="kubelet"} > 60	15m	The perc Poc dura exc sec
Kubelet fault	absent(sum(up{job="kubelet"}) by (cluster_id) > 0)	15m	Kub disa fron coll targ

Kubernetes Resource Use

Rule Name	Rule Expression	Duration	Description
Cluster CPU resource overload	sum by (cluster_id) (max by (cluster_id, namespace, pod, container) (kube_pod_container_resource_requests_cpu_cores{job=~".*kube-state-metrics"}) * on(cluster_id, namespace, pod) group_left() max by (cluster_id, namespace, pod) (kube_pod_status_phase{phase=~"Pending|Running"} == 1))/sum by (cluster_id) (kube_node_status_allocatable_cpu_cores)>(count by (cluster_id) (kube_node_status_allocatable_cpu_cores)-1) / count by (cluster_id) (kube_node_status_allocatable_cpu_cores)	5m	Too many CPU cores are applied for by Pods in the cluster, and no more failed nodes can be tolerated
Cluster memory resource overload	sum by (cluster_id) (max by (cluster_id, namespace, pod, container) (kube_pod_container_resource_requests_memory_bytes{job=~".*kube-state-metrics"}) * on(cluster_id, namespace, pod) group_left() max by (cluster_id, namespace, pod) (kube_pod_status_phase{phase=~"Pending|Running"} == 1))/sum by (cluster_id) (kube_node_status_allocatable_memory_bytes) > (count by (cluster_id) (kube_node_status_allocatable_memory_bytes)-1) / count by (cluster_id) (kube_node_status_allocatable_memory_bytes)	5m	Too much memory is applied for by Pods in the cluster, and no more failed nodes can be tolerated
Cluster CPU	sum by (cluster_id) (kube_resourcequota{job=~".*kube-state-metrics", type="hard", resource="cpu"})/sum by (cluster_id) (kube_node_status_allocatable_cpu_cores) > 1.5	5m	The CPU quota in the cluster

quota overload			exceeds the total number of allocable CPU cores
Cluster memory quota overload	sum by (cluster_id) (kube_resourcequota{job=~".*kube-state-metrics", type="hard", resource="memory"}) / sum by (cluster_id) (kube_node_status_allocatable_memory_bytes) > 1.5	5m	The memory quota in the cluster exceeds the total amount of allocable memory
Imminent runout of quota resources	sum by (cluster_id, namespace, resource) kube_resourcequota{job=~".*kube-state-metrics", type="used"} / sum by (cluster_id, namespace, resource) (kube_resourcequota{job=~".*kube-state-metrics", type="hard"} > 0) >= 0.9	15m	The quota resource utilization exceeds 90%
High proportion of restricted CPU execution cycles	sum(increase(container_cpu_cfs_throttled_periods_total{container!="", }[5m])) by (cluster_id, container, pod, namespace) /sum(increase(container_cpu_cfs_periods_total{}[5m])) by (cluster_id, container, pod, namespace) > (25 / 100)	15m	The proportion of restricted CPU execution cycles is high
High Pod CPU utilization	sum(rate(container_cpu_usage_seconds_total{job="kubelet", metrics_path="/metrics/cadvisor", image!="", container!="POD"}[1m])) by (cluster_id, namespace, pod, container) / sum(kube_pod_container_resource_limits_cpu_cores) by (cluster_id, namespace, pod, container) > 0.75	15m	The Pod CPU utilization exceeds 75%
High Pod memory utilization	sum(rate(container_memory_working_set_bytes{job="kubelet", metrics_path="/metrics/cadvisor", image!="", container!="POD"}[1m])) by (cluster_id, namespace, pod, container) /sum(kube_pod_container_resource_limits_memory_bytes) by (cluster_id, namespace, pod, container) > 0.75	15m	The Pod memory utilization exceeds 75%

Kubernetes Workload

Rule Name	Rule Expression	Duration
Frequent Pod restarts	increase(kube_pod_container_status_restarts_total{job=~".*kube-state-metrics"}[5m]) > 0	15m
Exceptional Pod status	sum by (namespace, pod, cluster_id) (max by(namespace, pod, cluster_id) (kube_pod_status_phase{job=~".*kube-state-metrics", phase=~"Pending|Unknown"}) * on(namespace, pod, cluster_id) group_left(owner_kind) topk by(namespace, pod) (1, max by(namespace, pod, owner_kind, cluster_id) (kube_pod_owner{owner_kind!="Job"}))) > 0	15m
Exceptional container status	sum by (namespace, pod, container, cluster_id) (kube_pod_container_status_waiting_reason{job=~".*kube-state-metrics"}) > 0	1h
Deployment version mismatch	kube_deployment_status_observed_generation{job=~".*kube-state-metrics"} !=kube_deployment_metadata_generation{job=~".*kube-state-metrics"}	15m
Deployment replica quantity mismatch	(kube_deployment_spec_replicas{job=~".*kube-state-metrics"} != kube_deployment_status_replicas_available{job=~".*kube-state-metrics"}) and (changes(kube_deployment_status_replicas_updated{job=~".*kube-state-metrics"}[5m]) == 0)	15m
StatefulSet version mismatch	kube_statefulset_status_observed_generation{job=~".*kube-state-metrics"} != kube_statefulset_metadata_generation{job=~".*kube-state-metrics"}	15m
StatefulSet	(kube_statefulset_status_replicas_ready{job=~".*kube-state-metrics"} !=	15m

replica quantity mismatch	kube_statefulset_status_replicas{job=~".*kube-state-metrics"}) and (changes(kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"}[5m]) == 0)	
Ineffective StatefulSet update	(maxwithout(revision)(kube_statefulset_status_current_revision{job=~".*kube-state-metrics"}unless kube_statefulset_status_update_revision{job=~".*kube-state-metrics"})*(kube_statefulset_replicas{job=~".*kube-state-metrics"}!=kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"})) and (changes(kube_statefulset_status_replicas_updated{job=~".*kube-state-metrics"}[5m])==0)	15m
Frozen DaemonSet change	((kube_daemonset_status_current_number_scheduled{job=~".*kube-state-metrics"}!=kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"}) or (kube_daemonset_status_number_misscheduled{job=~".*kube-state-metrics"}!=0) or (kube_daemonset_updated_number_scheduled{job=~".*kube-state-metrics"}!=kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"}) or (kube_daemonset_status_number_available{job=~".*kube-state-metrics"}!=kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"})) and (changes(kube_daemonset_updated_number_scheduled{job=~".*kube-state-metrics"}[5m])==0)	15m
DaemonSet not scheduled on some nodes	kube_daemonset_status_desired_number_scheduled{job=~".*kube-state-metrics"} - kube_daemonset_status_current_number_scheduled{job=~".*kube-state-metrics"} > 0	10m
Faulty scheduling of DaemonSet on some nodes	kube_daemonset_status_number_misscheduled{job=~".*kube-state-metrics"} > 0	15m
Excessive Job execution	kube_job_spec_completions{job=~".*kube-state-metrics"} - kube_job_status_succeeded{job=~".*kube-state-metrics"} > 0	12h
Job execution failure	kube_job_failed{job=~".*kube-state-metrics"} > 0	15m
Mismatch between replica	(kube_hpa_status_desired_replicas{job=~".*kube-state-metrics"} != kube_hpa_status_current_replicas{job=~".*kube-state-metrics"}) and	15m

quantity and HPA	changes(kube_hpa_status_current_replicas[15m]) == 0	
Number of replicas reaching maximum value in HPA	kube_hpa_status_current_replicas{job=~".*kube-state-metrics"} == kube_hpa_spec_max_replicas{job=~".*kube-state-metrics"}	15m
Exceptional PersistentVolume status	kube_persistentvolume_status_phase{phase=~"Failed|Pending",job=~".*kube-state-metrics"} > 0	15m

Kubernetes Node

Rule Name	Rule Expression	Duration	Description
Imminent runout of filesystem space	(node_filesystem_avail_bytes{job="node-exporter",fstype!=""}/node_filesystem_size_bytes{job="node-exporter",fstype!=""}*100<15 and predict_linear(node_filesystem_avail_bytes{job="node-exporter",fstype!=""}[6h],4*60*60)<0 and node_filesystem_readonly{job="node-exporter",fstype!=""}==0)	1h	It is estimated that the filesystem space will be used up in 4 hours
High filesystem space utilization	(node_filesystem_avail_bytes{job="node-exporter",fstype!=""}/node_filesystem_size_bytes{job="node-exporter",fstype!=""}*100<5 and node_filesystem_readonly{job="node-exporter",fstype!=""}==0)	1h	The available filesystem space is below 5%
Imminent runout of filesystem inodes	(node_filesystem_files_free{job="node-exporter",fstype!=""}/node_filesystem_files{job="node-exporter",fstype!=""}*100<20 and predict_linear(node_filesystem_files_free{job="node-exporter",fstype!=""}[6h],4*60*60)<0 and node_filesystem_readonly{job="node-exporter",fstype!=""}==0)	1h	It is estimated that the filesystem inodes will be used up in 4 hours
High	(node_filesystem_files_free{job="node-	1h	The proportion of

filesystem inode utilization	exporter",fstype!=""}/node_filesystem_files{job="node-exporter",fstype!=""}*100<3 and node_filesystem_readonly{job="node-exporter",fstype!=""}==0)		available inodes is below 3%
Unstable network interface status	changes(node_network_up{job="node-exporter",device!~"veth.+"}[2m])	2m	The network interface status is unstable and frequently changes between "up" and "down"
Network interface data reception error	increase(node_network_receive_errs_total[2m]) > 10	1h	An error occurred while the network interface received data
Network interface data sending error	increase(node_network_transmit_errs_total[2m]) > 10	1h	An error occurred while the network interface sent data
Unsynced server clock	min_over_time(node_timex_sync_status[5m]) == 0	10m	The server time has not been synced recently. Please check whether NTP is correctly configured
Server clock skew	(node_timex_offset_seconds>0.05 and deriv(node_timex_offset_seconds[5m])>=0) or (node_timex_offset_seconds<-0.05 and deriv(node_timex_offset_seconds[5m])<=0)	10m	The server clock skew exceeds 300 seconds. Please check whether NTP is correctly configured

Notification Template

Last updated：2024-08-07 22:05:39

This document describes how to create a notification template in the Cloud Monitor alarming module.

Use Cases

One template can be quickly reused for multiple policies, eliminating the need to repeatedly configure user notifications.
User notification methods can be configured in a more personalized way. For example, you can configure the alarm receiving channel as SMS/email by day and phone by night.

Prerequisites

View notification templates: the sub-account must have the read permission of Cloud Monitor.
Create and edit notification templates: the sub-account must have the write permission of Cloud Monitor.
Note:
For more information on how to grant sub-accounts permissions, please see Cloud Access Management (CAM).

Use Limits

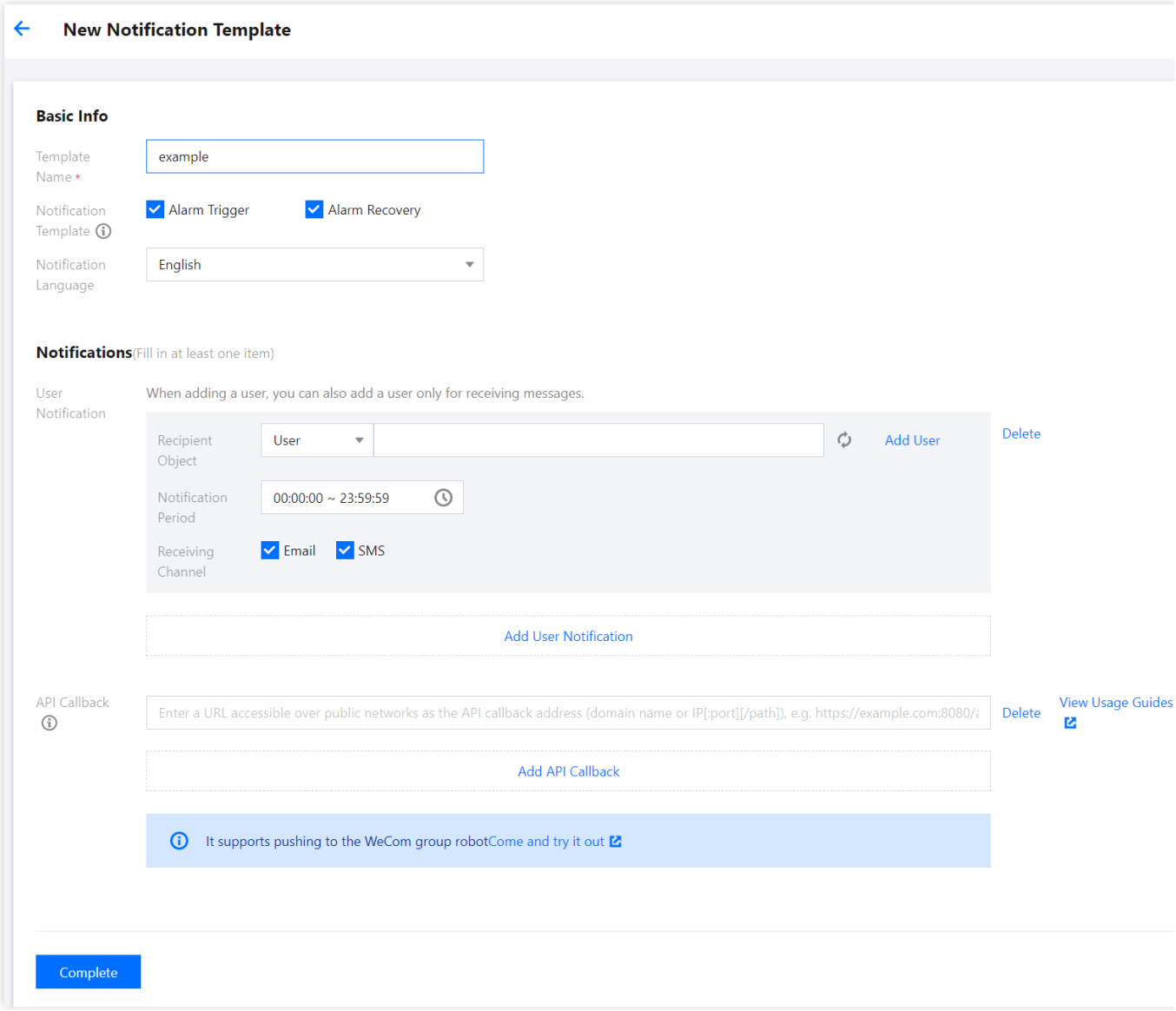
Feature	Limit
User notification	Up to five items can be added
API callback	Up to three URLs accessible over the public network can be entered

Directions

Creating notification template

1. Enter the Alarm Notification Template page in the Cloud Monitor console.
2. Click **Create** and enter relevant information in **Create Notification Template**.
Template Name: enter a custom template name.
Notification type:

Alarm triggered: a notification will be sent when an alarm is triggered.
Alarm cleared: a notification will be sent when an alarm is resolved.
User notification:
Recipient Object: you can choose a recipient group or recipient. If you need to create a group, please see Creating Alarm Recipient Group.
Notification Period: define the time period for receiving alarms.
Receiving Channel: four alarm channels are supported: email, SMS, WeChat, and phone. You can also set different channels and notification periods in different user dimensions.
Description of phone alarm settings:
Polling Times: the maximum number of dials for each polled recipient when there is no valid reach.
Polling Sequence: alarm calls will be dialed according to the order of the recipients. You can adjust the order of calling by dragging up and down recipients.
Polling Interval: time interval at which alarm calls will be dialed according to the order of the recipients.
Reach Notification: notifications will be to all recipients after successful reception of the call or calling all recipients. SMS messages are counted against the quota.
API Callback: you can enter a URL accessible over the public network as the callback API address, and Cloud Monitor will push alarm messages to it promptly. If the HTTP response returns code 200, the verification is successful. For more information on alarm callback fields, please see Alarm Callback.
Note:
After you save the callback URL, the system will automatically verify your URL once. The timeout threshold for this verification is 5 seconds. When an alarm policy created by the user is triggered or the alarm is resolved, the alarm messages will be pushed through the API callbacks. An alarm message can be pushed up to three times, and the timeout threshold for each request is 5 seconds.
When an alarm policy created by the user is triggered or the alarm is resolved, the alarm messages will be pushed through the API callbacks. API callbacks also support repeated alarms.
The outbound IP of the Cloud Monitor callback API is dynamically and randomly allocated, so no specific IP information can be provided to you, but the IP port is fixed at 80. We recommend you configure a weighted opening policy in the security group based on port 80.



Default notification template

The system automatically creates a default notification template for you as detailed below:

Feature	Default Configuration
Template name	Preset notification template
Notification type	Alarm trigger, alarm recovery
Alarm recipient	Root account admin
Notification period	00:00:00–23:59:59 (all day)
Receiving channel	Email, SMS

Deleting template

Note:

The default notification template cannot be deleted.

1. Find the name of the template to be deleted and click **Delete** in the **Operation** column.
2. In the pop-up window, click **OK**.

Replicating template

1. Find the name of the template to be replicated and click **Replicate** in the **Operation** column.
2. Modify the information in the redirected page or click **Complete** directly.

Tag
Examples

Last updated：2024-08-07 22:05:51

Overview

A tag is a key-value pair provided by Tencent Cloud to identify a resource in the cloud.
You can use tags to classify TMP resources based on various factors such as service, usage, and owner. With tags, you can quickly sift through the resource pool and find the corresponding resources. The values of tag keys do not mean anything to Tencent Cloud semantically and will be parsed and matched strictly according to the string. Tencent Cloud will not use your set tags, which are used only for resource management.
Below is a specific use case to show how a tag is used.

Use Case Background

A company owns 10 TMP instances in Tencent Cloud. Distributed in three departments (ecommerce, gaming, and entertainment), these instances are used to serve internal business lines such as marketing, game A, game B, and post-production. The OPS owners of the three departments are John, Jane, and Harry, respectively.

Setting Tag

To facilitate management, the company categorizes its TMP resources with tags and defines the following tag key-value pairs:

Tag Key	Tag Value
Department	Ecommerce, gaming, and entertainment
Business	Marketing, game A, game B, and post-production
OPS owner	John, Jane, and Harry

These tags are bound to TMP instances in the following way:

Instance ID	Department	Business	OPS Owner
prom-1jqwv1	Ecommerce	Marketing	Harry
prom-1jqwv12	Ecommerce	Marketing	Harry
prom-1jqwv13	Gaming	Game A	John
prom-1jqwv13	Gaming	Game B	John

prom-1jqwv14	Gaming	Game B	John
prom-1jqwv15	Gaming	Game B	Jane
prom-1jqwv16	Gaming	Game B	Jane
prom-1jqwv17	Gaming	Game B	Jane
prom-1jqwv18	Entertainment	Post-production	Harry
prom-1jqwv19	Entertainment	Post-production	Harry
prom-1jqwv110	Entertainment	Post-production	Harry

Using Tag

Filter out the TMP instances in the charge of Harry

Filter out the TMP instances where the OPS owner is "Harry". For detailed directions, please see Using Tag.

Filter out the TMP instances in the charge of Jane in the gaming department

Filter out the TMP instances where the department is "gaming" and OPS owner is "Jane". For detailed directions, please see Using Tag.

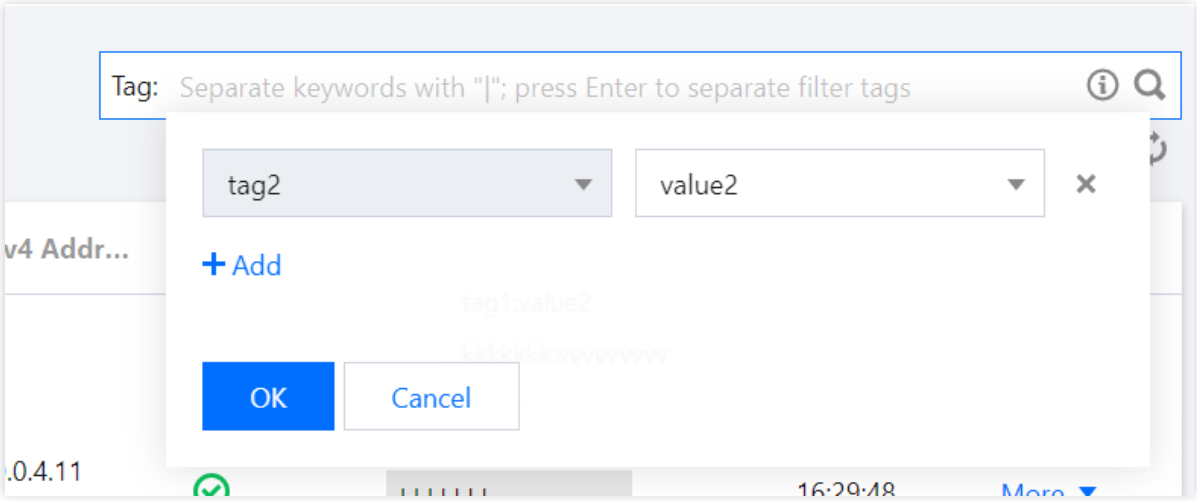
Using Tag

Last updated：2024-08-07 22:06:01

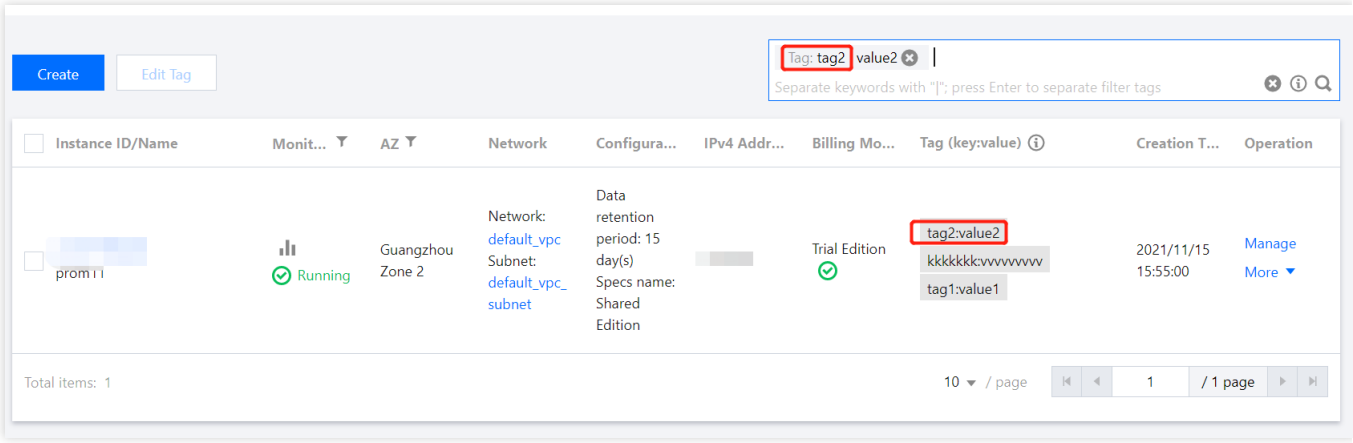
This document describes how to filter instance resources by tags in the TMP console.

Directions

1. Log in to the TMP console.
2. Select the region at the top of the instance list page.
3. In the search box in the top-right corner of the instance list, click the blank space and select **Tag**.



4. Select the corresponding conditions in the tag filter selection box and click **OK**.
5. If you need to adjust the tag conditions, click the tag content after **Tag:** in the search box to edit it.
6. You can also directly click the corresponding tag value in the instance list to filter instances as shown below:



Editing Tag

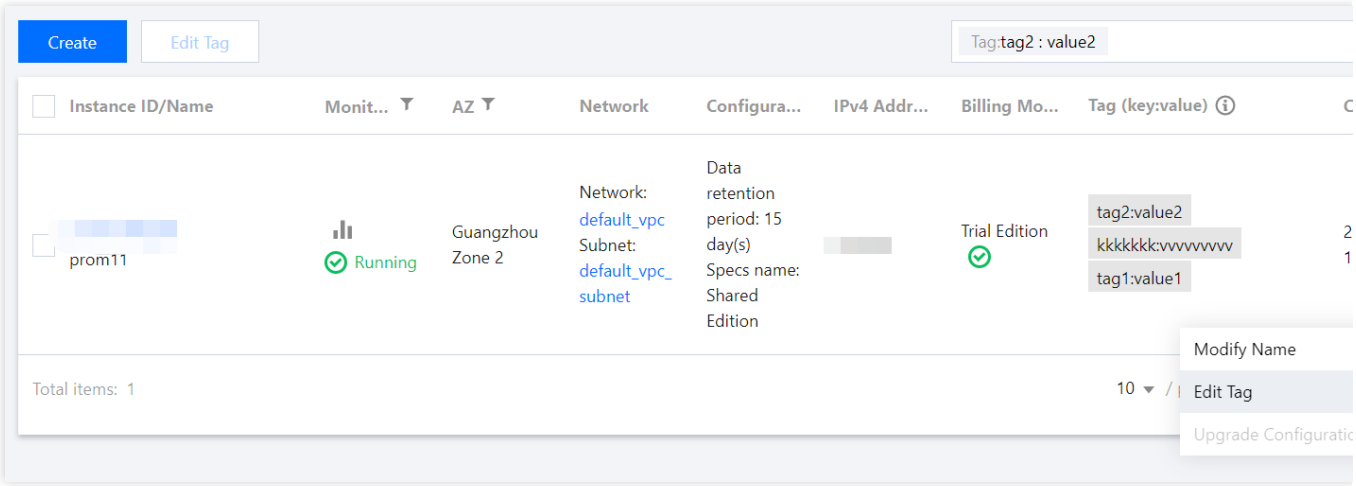
Last updated：2024-08-07 22:06:06

This document describes how to edit the tags of an instance in the TMP console.

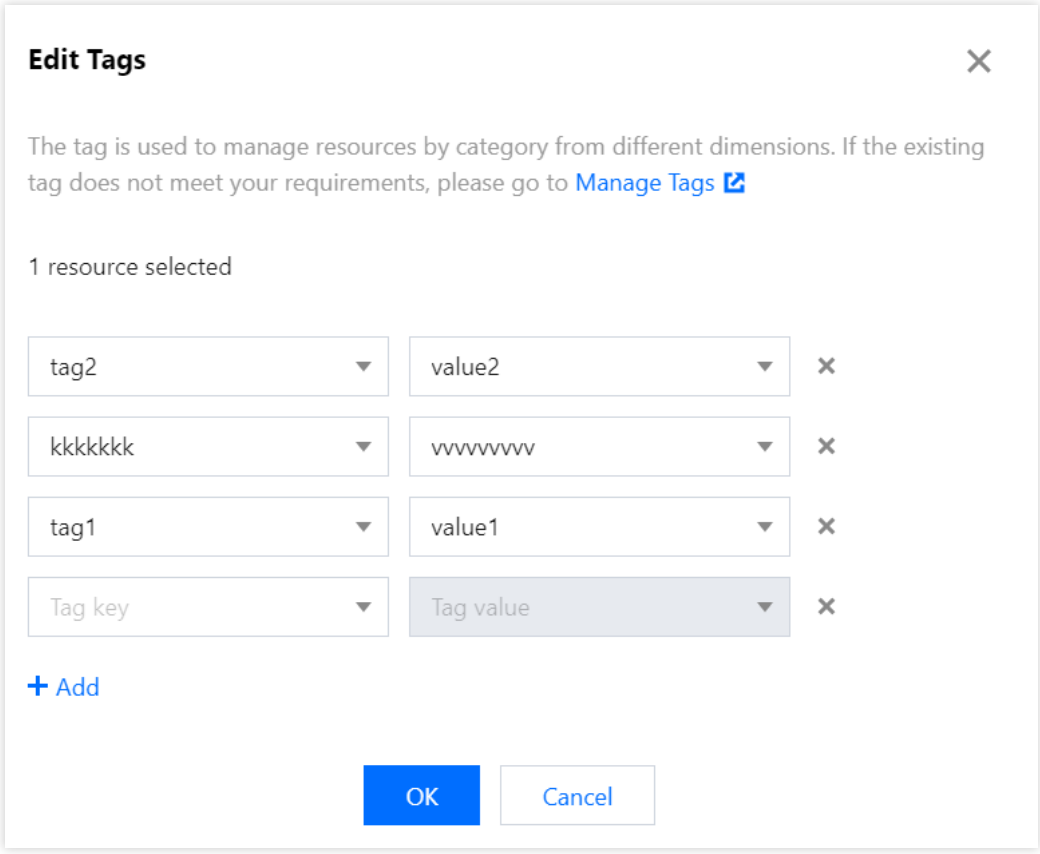
Directions

Edit tag for a single instance

1. Log in to the TMP console.
2. On the instance management page, select the instance for which to edit tags and click **More** > **Instance Configuration** > **Edit Tag** as shown below:



3. In the **Selected 1 resource** window that pops up, add, modify, or delete tags based on your actual needs.



Use Limits

Last updated：2024-08-07 22:06:11

A tag is a key-value pair. You can set tags for TMP instances in the TMP console to manage them in a categorized manner. Then, you can easily filter and find desired resources with tags.

Quantity Limit

One Tencent Cloud resource can have up to 50 tags.

Tag Key Limit

It cannot begin with "qcloud", "tencent", or "project" as they are reserved by the system.
It can contain only letters, digits, spaces, and certain special symbols (+, -, =, ., _, :, /, @).
It can contain up to 255 characters.

Tag Value Limit

It can contain only letters, digits, spaces, and certain special symbols (+, -, =, ., _, :, /, @).
It can contain up to 127 characters.

Access Control Overview

Last updated：2024-08-07 22:06:26

If you have multiple users managing the TMP service, and they all share your Tencent Cloud account access key, you may face the following problems:
The risk of your key being compromised is high since multiple users are sharing it.
Your users might introduce security risks from maloperations due to the lack of user access control.
You can avoid the above problems by allowing different users to manage different services through sub-accounts. By default, sub-accounts have no permissions to use TMP. Therefore, you need to create a policy to grant different permissions to sub-accounts.

Overview

Cloud Access Management (CAM) is a web-based Tencent Cloud service that helps you securely manage and control access permissions of your Tencent Cloud resources. Using CAM, you can create, manage, and terminate users (groups), and control the Tencent Cloud resources that can be used by the specified user through identity and policy management.
When using CAM, you can associate a policy with a user or user group to allow or forbid them to use specified resources to complete specified tasks. For more information on CAM policies, please see Element Reference. For more information on how to use CAM policies, please see Policy.
You can skip this section if you don't need to manage permissions of TMP resources for sub-accounts. This won't affect your understanding and use of the other sections of the document.

Getting Started

A CAM policy must authorize or deny the use of one or more TMP operations. At the same time, it must specify the resources that can be used for the operations (which can be all resources or partial resources for certain operations). A policy can also include the conditions set for the manipulated resources.
Certain APIs of TMP don't support resource-level permissions, which means that for this type of API operations, you cannot specify a given resource for use when they are performed; instead, you must specify all resources for use.

Setting Policy

Last updated：2024-08-07 22:06:32

Overview

Access policies can be used to grant access to TMP instances. They use JSON-based access policy syntax. You can authorize specified principals to perform specified operations on specified TMP resources through the access policy syntax.
The access policy syntax describes the basic elements and usage of the policy. For the description of the policy syntax, please see Permission.

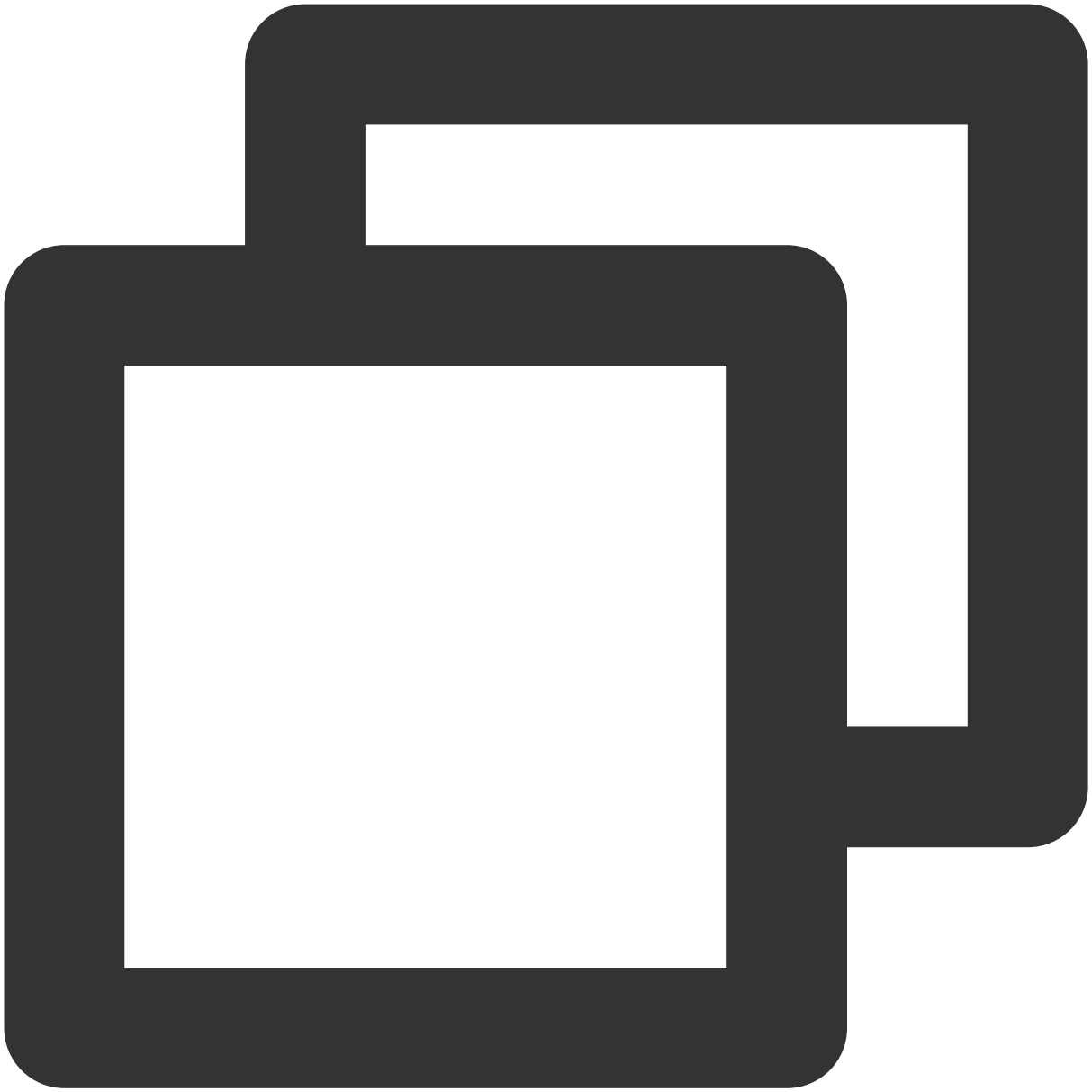
Elements in Access Policy

An access policy contains the following elements with basic meanings:
statement: it describes the details of one or more permissions. It contains a permission or permission set of multiple other elements such as `effect` , `action` , `resource` , and `condition` . One policy must and can have only one `statement` .
effect: it is required and describes the result of a statement. The result can be an "allow" or "explicit deny".
action: it is required and describes the allowed or denied action (operation). An operation can be an API (prefixed with "name") or a feature set (a set of specific APIs prefixed with "permid").
resource: it is required and describes the details of authorization. A resource is described in a six-segment format. Detailed resource definitions vary by product.
condition: it is optional and describes the condition for the policy to take effect. A condition consists of operator, action key, and action value. A condition value may contain information such as time and IP address. Some services allow you to specify additional values in a condition.

Element Usage

Specifying effect

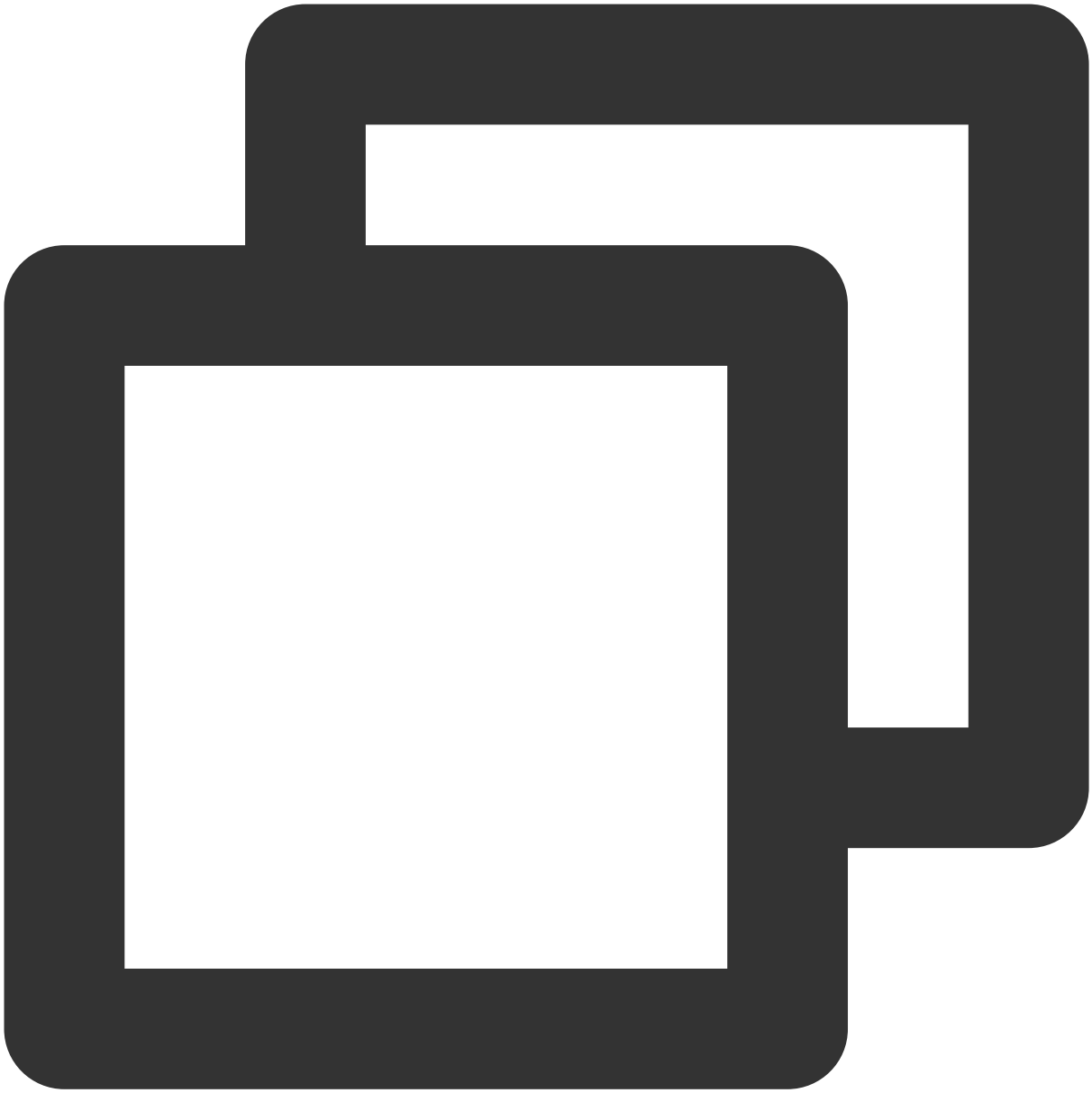
If access to a resource is not explicitly granted (allowed), then it is implicitly denied. It can also be explicitly denied, which ensures that users cannot access the resource even if they are granted the access permission by other policies. Below is an example of specifying the "allow" effect:



```
"effect" : "allow"
```

Specifying action

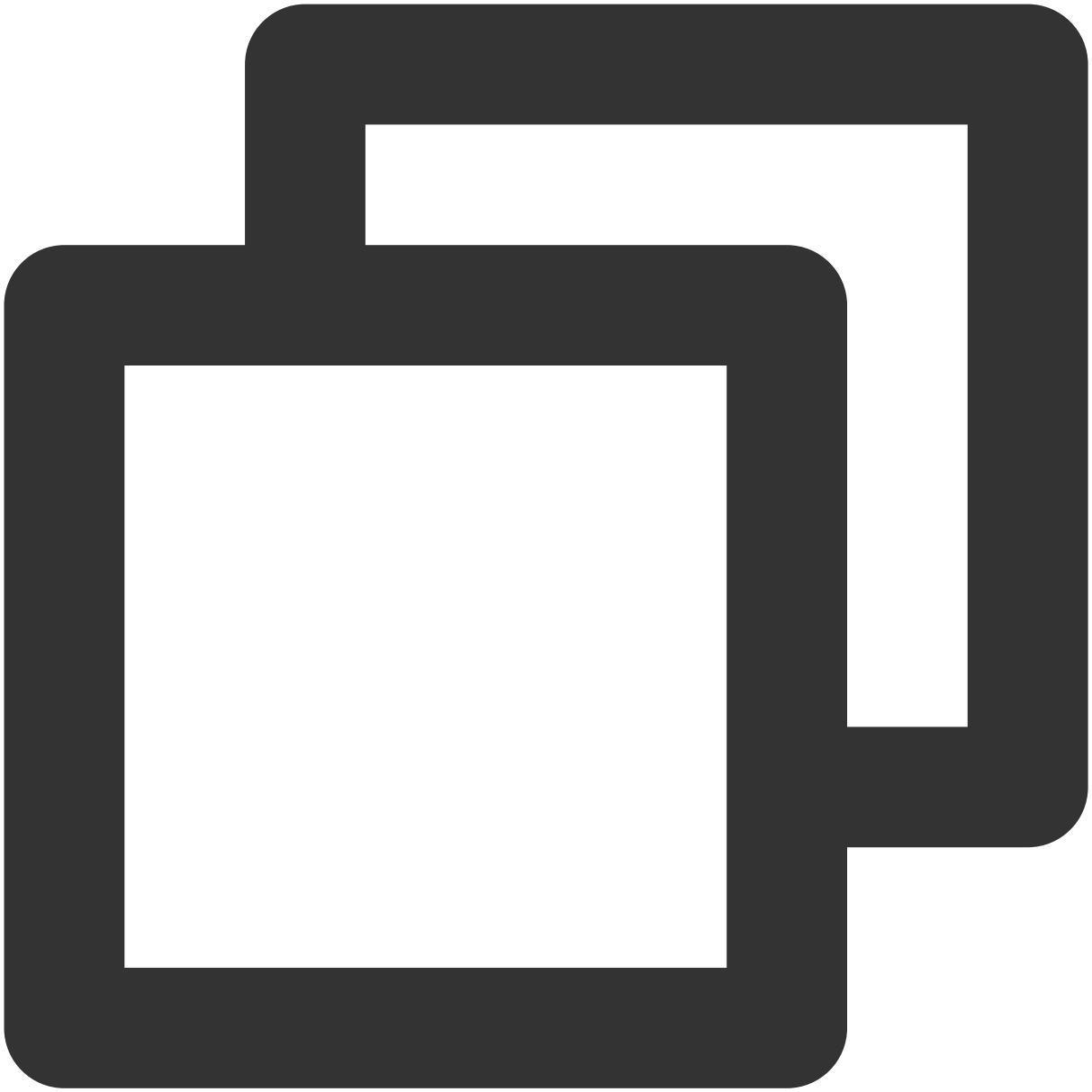
Cloud Monitor defines console operations that can be specified in a policy. The specified operations are divided into reading part of APIs (monitor:Describe*) and all APIs (monitor:*) according to the operation nature.
Below is an example of specifying the allowed operations:



```
"action": [
  "name/monitor:Describe*"
]
```

Specifying resource

The `resource` element describes one or more operation objects, such as TMP resources. All resources can use the following six-segment format:



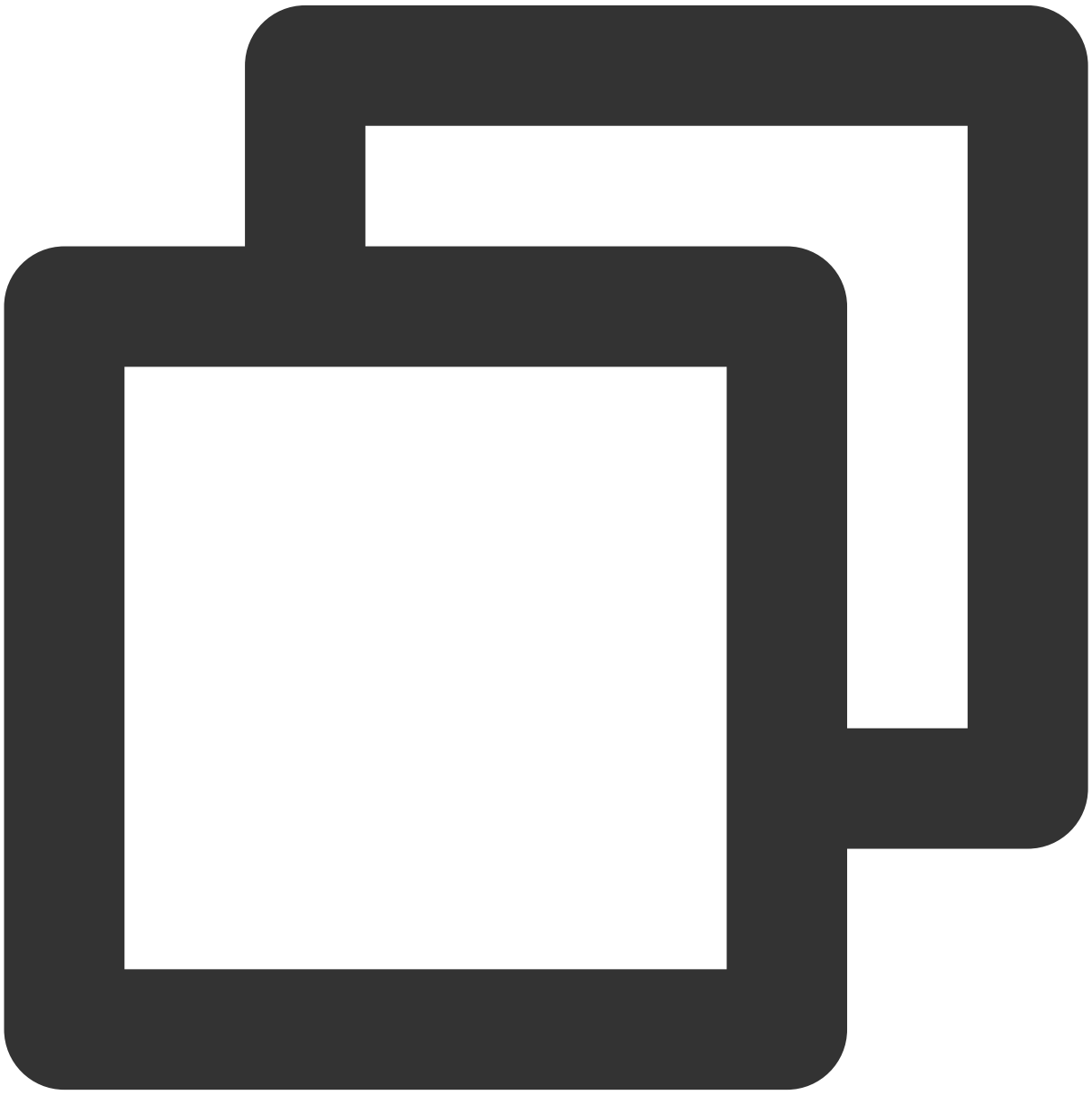
```
qcs:project_id:service_type:region:account:resource
```

The parameters are as detailed below:

Parameter	Description	Required
qcs	Tencent Cloud service abbreviation, which indicates a service of Tencent Cloud	Yes
project_id	Project information, which is only used to enable compatibility with legacy CAM logic and generally can be left empty	No

service_type	Product abbreviation, which is `monitor` here	Yes
region	Region information	Yes
account	Root account information of the resource owner, i.e., root account ID in the format of `uin/${OwnerUin}` , such as uin/100000000001	Yes
resource	Resource details prefixed with `instance`	Yes

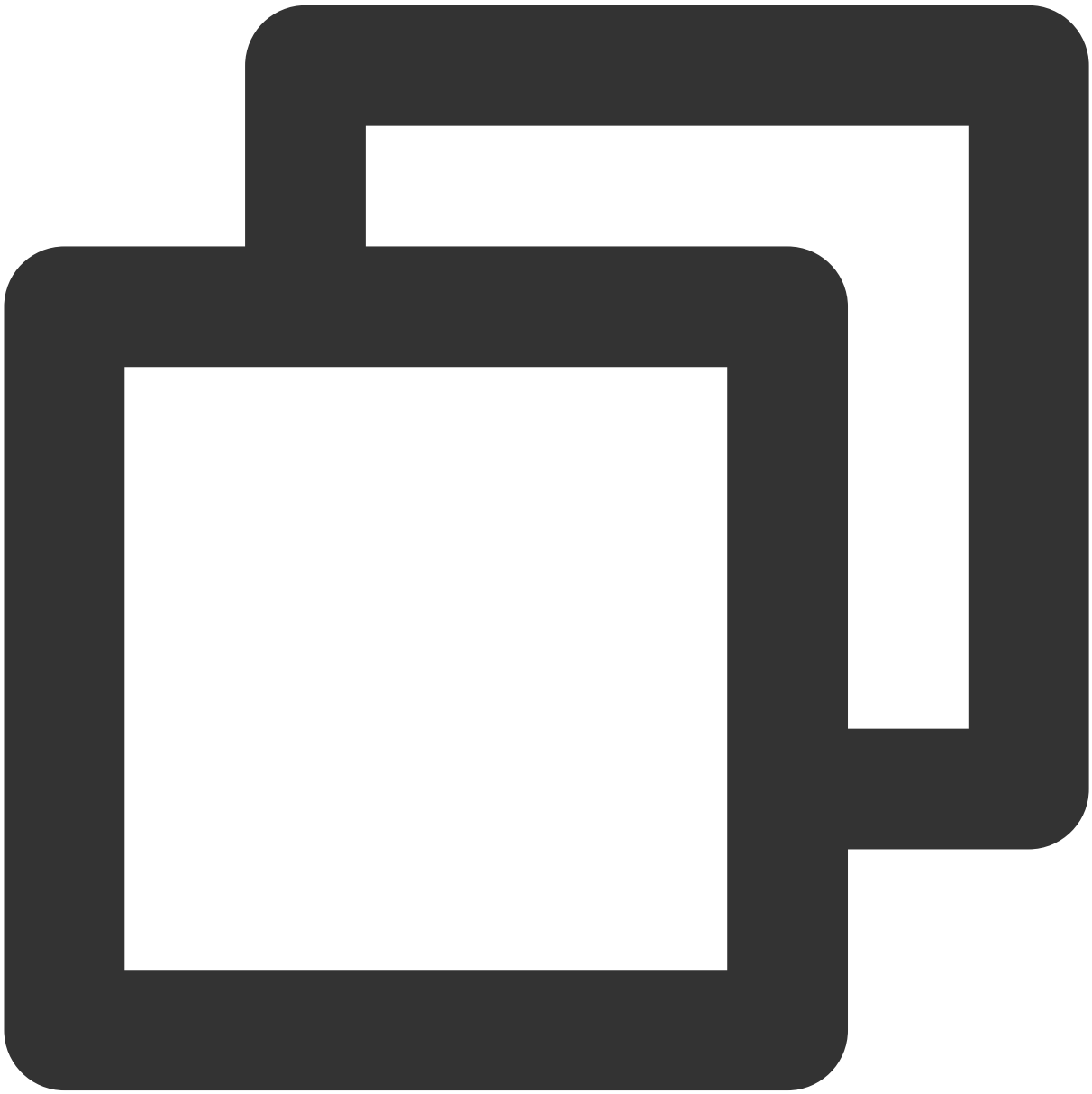
Below is a sample six-segment TMP resource description:



```
"resource":["qcs::monitor:ap-guangzhou:uin/100000000001:prom-instance/prom-73jingds
```

Specifying condition

The access policy syntax allows you to specify the condition when granting permissions, which is mainly used to set tag authentication. The tag condition takes effect only for clusters bound with the tag. Below is a sample tag policy:



```
"condition": {
    "for_any_value:string_equal": {
        "qcs:tag": [
```

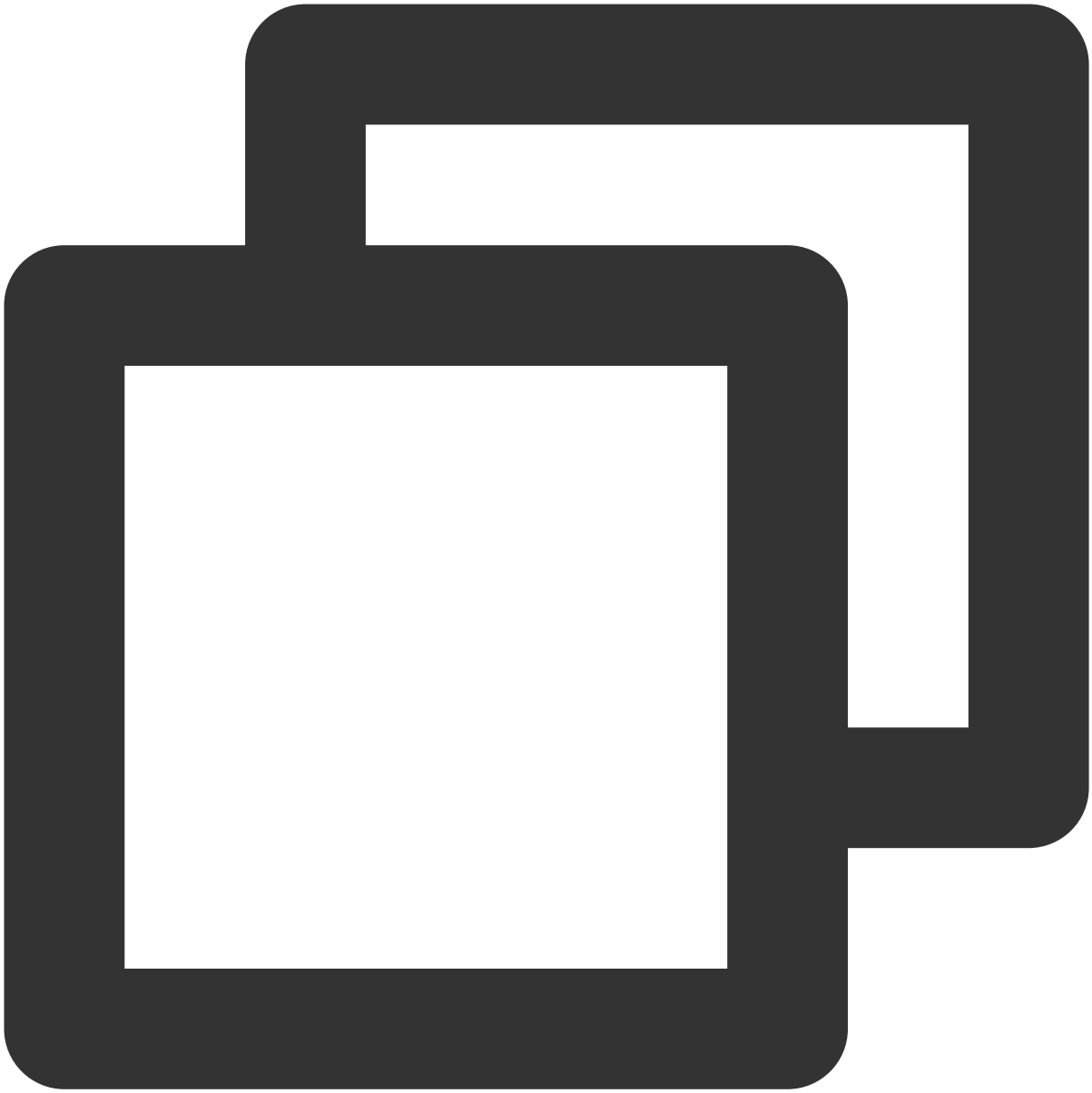
```
                "testkey&testvalue"
            ]
        }
    }
```

This statement means that the policy contains resources whose tag key is `testkey` and tag value is `testvalue` .

Use Cases

Based on tag

In the following case, the policy is to allow access to the resource whose instance ID is `prom-73jingds` under UIN 1250000000 and the resources whose tag key is `testkey` and tag value is `testvalue` (if this instance doesn't have the `testkey& testvalue` tag, it cannot be accessed).



```
{
   "version": "2.0",
    "statement": [
        {
            "action": [
                "name/monitor:*"
            ],
            "condition": {
               "for_any_value:string_equal": {
                    "qcs:tag": [
                        "testkey&testvalue"
```

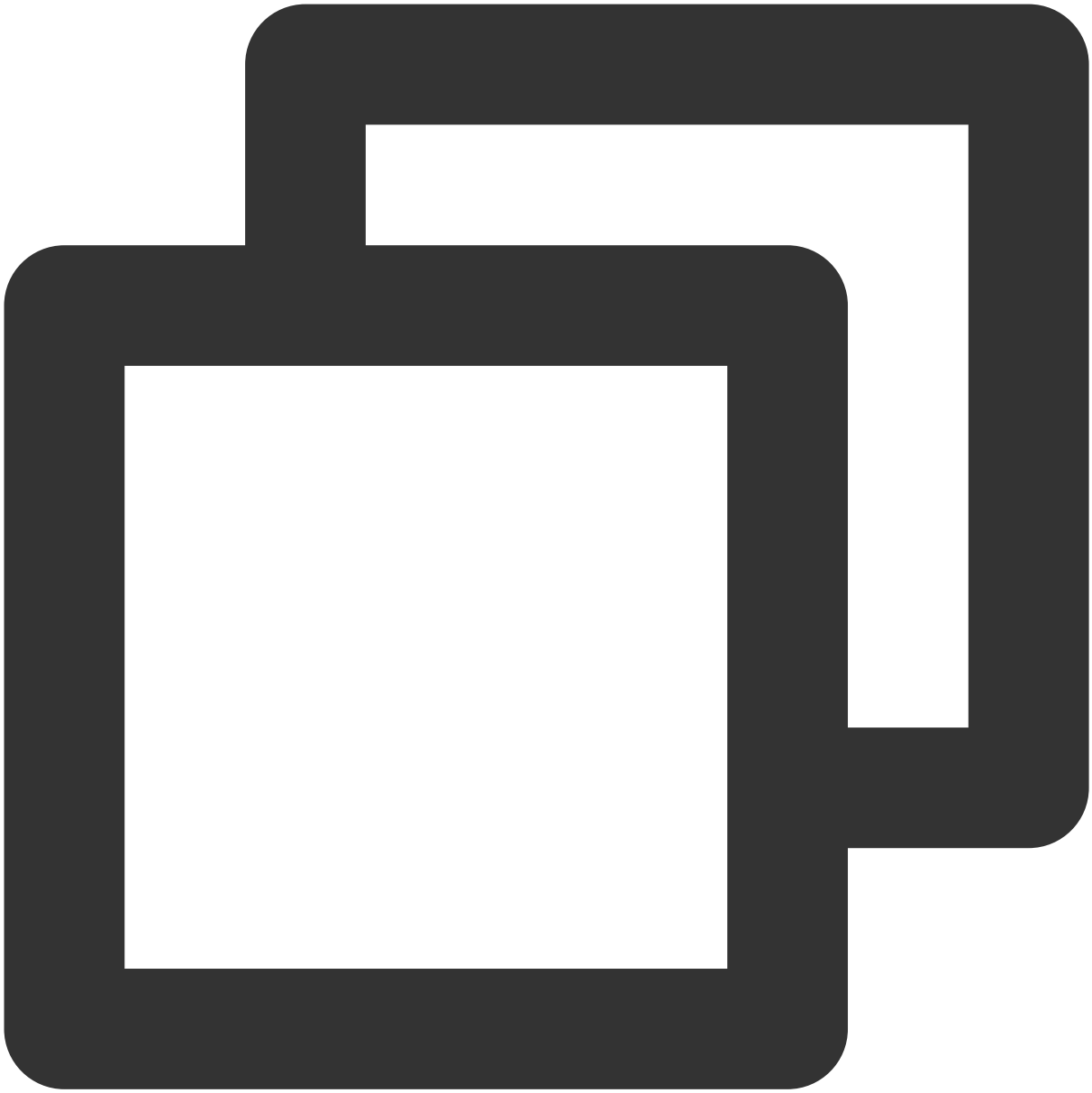
```
                    ]
                }
            },
            "effect": "allow",
            "resource": [
                "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-73jing
            ]
        }
    ]
}
```

Based on resource

Grant the read and write permissions of specified resources based on resource ID. The root account ID is 1250000000:

Configure read-only access to resources in the Guangzhou region.

Sample: granting the sub-user read-only access to the instance1(prom-73jingds) and instance2(prom-65jidfafk) resources.

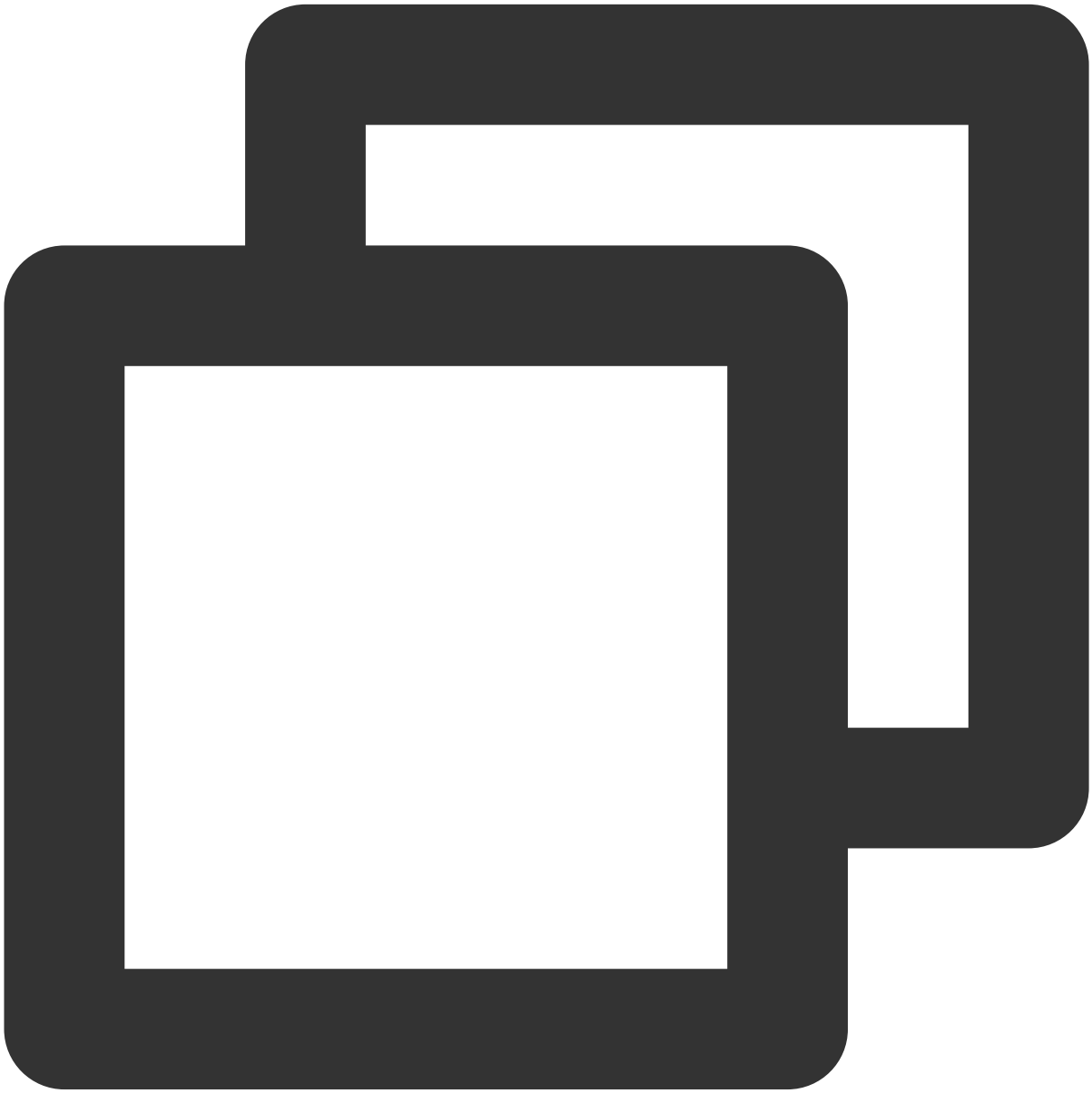


```
{
  "version": "2.0",
  "statement": [{
      "effect": "allow",
      "action": [
          "name/monitor:Describe*"
      ],
      "resource": [
          "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-73jingds",
          "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-65jidfafk"
      ],
```

```
        "condition": []
    }]
}
```

Configure partial read/write access to resources in the Guangzhou region.

Sample: granting the sub-user the permission to delete the instance1(prom-73jingds) resource



```
{
  "version": "2.0",
  "statement": [{
      "effect": "allow",
```

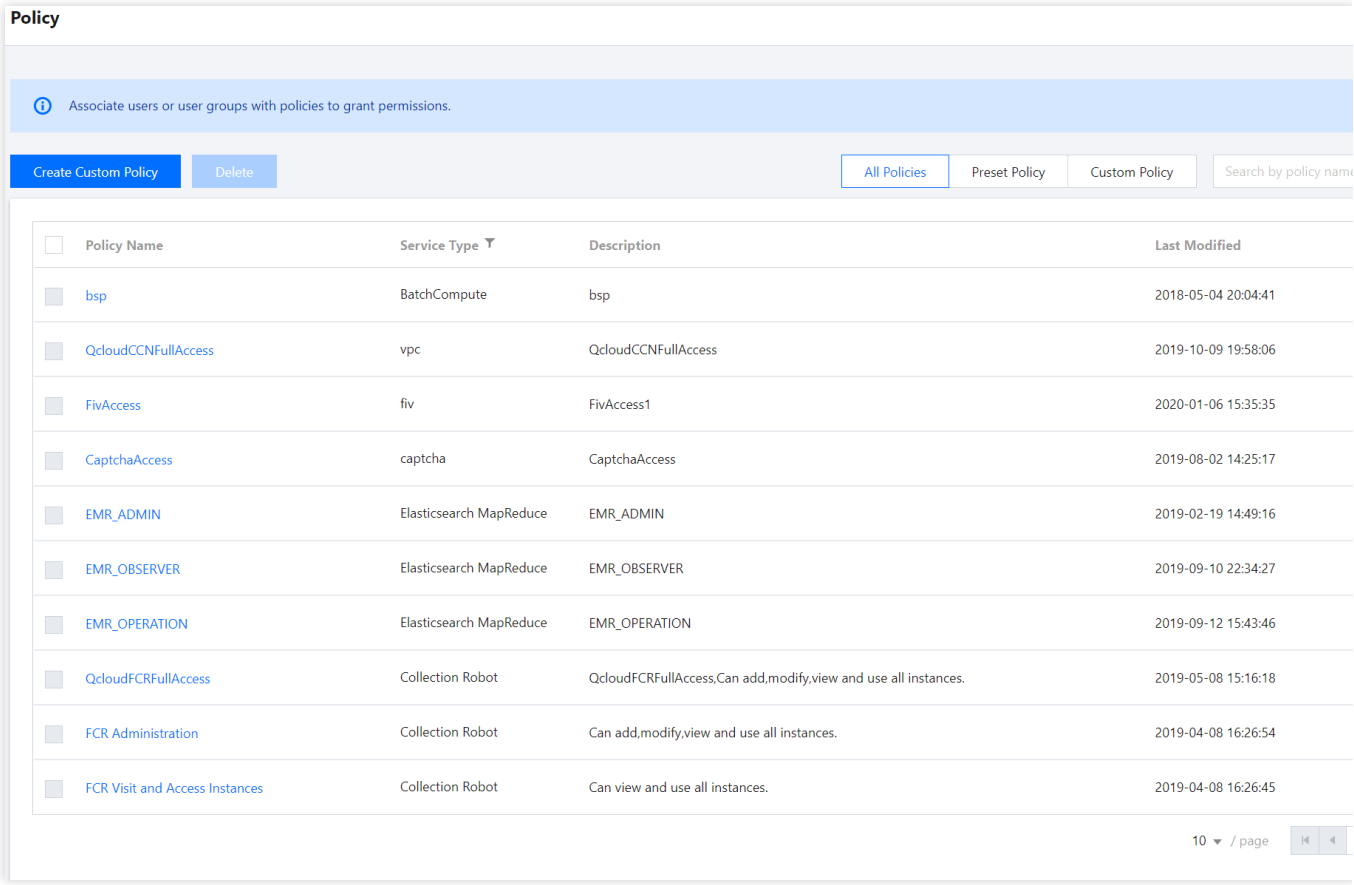
```
        "resource": [
            "qcs::monitor:ap-guangzhou:uin/1250000000:prom-instance/prom-73jingds"
        ],
        "action": [
            "name/monitor:TerminatePrometheusInstances"
        ]
    }]
}
```

Granting Policy

Last updated：2024-08-07 22:06:37

Custom Policy for TMP

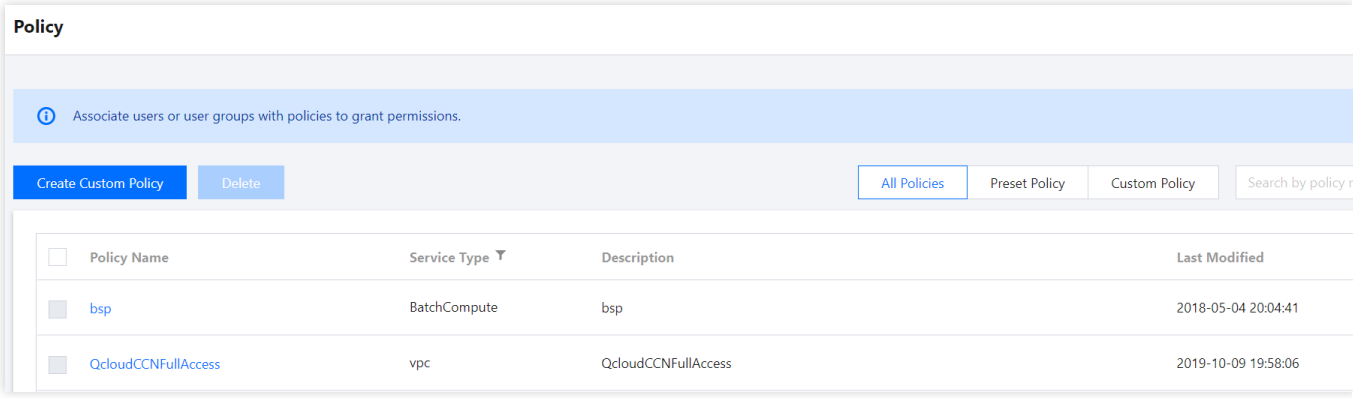
If preset policies cannot meet your needs, you can click **Create Custom Policy** to create custom policies.



For the method of custom policy creation, please see Setting Policy.

Policy Authorization

A configured policy can grant permissions by associating user groups or sub-users.



Resource Types Authorizable by Custom Policy

Resource-Level permission can be used to specify which resources a user can manipulate. TMP supports certain resource-level permissions. This means that for TMP operations that support resource-level permission, you can control the time when a user is allowed to perform operations or to use specified resources. The following table describes the types of resources that can be authorized in CAM.

Resource Type	Resource Description Method in Authorization Policy
TMP	`qcs::monitor:$region:$account:prom-instance/*` `qcs::monitor:$region:$account:prom-instance/$instanceId`

The following table describes the TMP API operations that currently support resource-level permissions. When setting a policy, you can enter the API operation name in the `action` field to control the individual API. You can also use `*` as a wildcard to set the `action` .

List of APIs supporting resource-level authorization

API Operation	API Description
DescribePrometheusInstances	Lists all TMP instances of the user
TerminatePrometheusInstances	Terminates TMP instance
RecreatePrometheusInstance	Reboots TMP instance
ModifyPrometheusInstanceAttributes	Modifies TMP instance attributes
ChangeGrafanaAdminPassword	Changes Grafana admin Password
UpgradeGrafanaDashboard	Upgrades Grafana dashboard

DescribePrometheusKubeClusters	Lists TKE clusters that can be integrated with TMP
InstallPrometheusAgent	Installs Prometheus agent
UninstallPrometheusAgent	Uninstalls Prometheus agent
DescribeServiceDiscovery	Lists TMP scrape configurations
CreateServiceDiscovery	Creates TMP scrape configuration
UpdateServiceDiscovery	Updates TMP scrape configuration
DeleteServiceDiscovery	Deletes TMP scrape configuration
DescribePrometheusKubeBasicMonitor	Queries basic monitoring status
EnablePrometheusKubeBasicMonitor	Enables basic monitoring
DisablePrometheusKubeBasicMonitor	Disables basic monitoring
DescribePrometheusAgentRuntime	Gets the runtime status of Prometheus agent
DescribePrometheusJobTargets	Lists the status information of TMP metric scrape tasks
DescribeRecordingRules	Queries recording rules
CreateRecordingRule	Creates recording rule
UpdateRecordingRule	Updates recording rule
DeleteRecordingRules	Deletes recording rule
DescribeAlertRules	Queries alarming rules
DeleteAlertRules	Deletes alarming rule
UpdateAlertRuleState	Updates alarming rule status
CreateAlertRule	Creates alarming rule
UpdateAlertRule	Updates alarming rule

List of APIs not supporting resource-level authorization

For TMP API operations that don't support resource-level authorization, you can still authorize a user to perform them, but you must specify `*` as the resource element in the policy statement.

API Operation	API Description

CreatePrometheusInstance	Creates TMP instance

Description of Role Permissions Related to Service Authorization

Last updated：2024-08-07 22:06:42

When you use TMP, in order to use related Tencent Cloud resources, you will encounter a variety of scenarios that require service authorization. The `CM_QCSRole` service role is mainly involved in the process of using TMP. This document describes the details, scenarios, and steps of each authorization policy by role.
The preset policies associated with the `CM_QCSRole` role by default include the following:
QcloudAccessForCMRoleInPromHostingService: TKE permission required by TMP.

Use Cases

After you successfully create a TMP instance, you need to monitor the services running on TKE. In order to integrate the TKE service more conveniently, you need to access TKE-related APIs. In this case, your authorization is required before TKE can be normally accessed to install basic monitoring components and get their running status information. This role doesn't need to actively look for configuration. If its permission hasn't been granted, after you successfully create a TMP instance, the authorization page will automatically pop up when you enter the **Integrate with TKE** page for instance management.

Authorization Steps

Authorizing by root account

1. After you successfully create a TMP instance, an authorization window will pop up when you access the **Integrate with TKE** page, and you need to authorize Cloud Monitor permissions as shown below:
2. Click **Authorize Now** in the window.
3. On the **CAM** > **Role Management** page, click **Grant**, and the system will prompt that the authorization is successful.

Note:
This authorization window will appear only once. If you have already authorized, it will not appear again.

Granting permissions to sub-account

After the root account completes the above authorization operations and successfully creates the `CM_QCSRole` role, the sub-account doesn't have permission to access it. The sub-account must be granted the `PassRole`

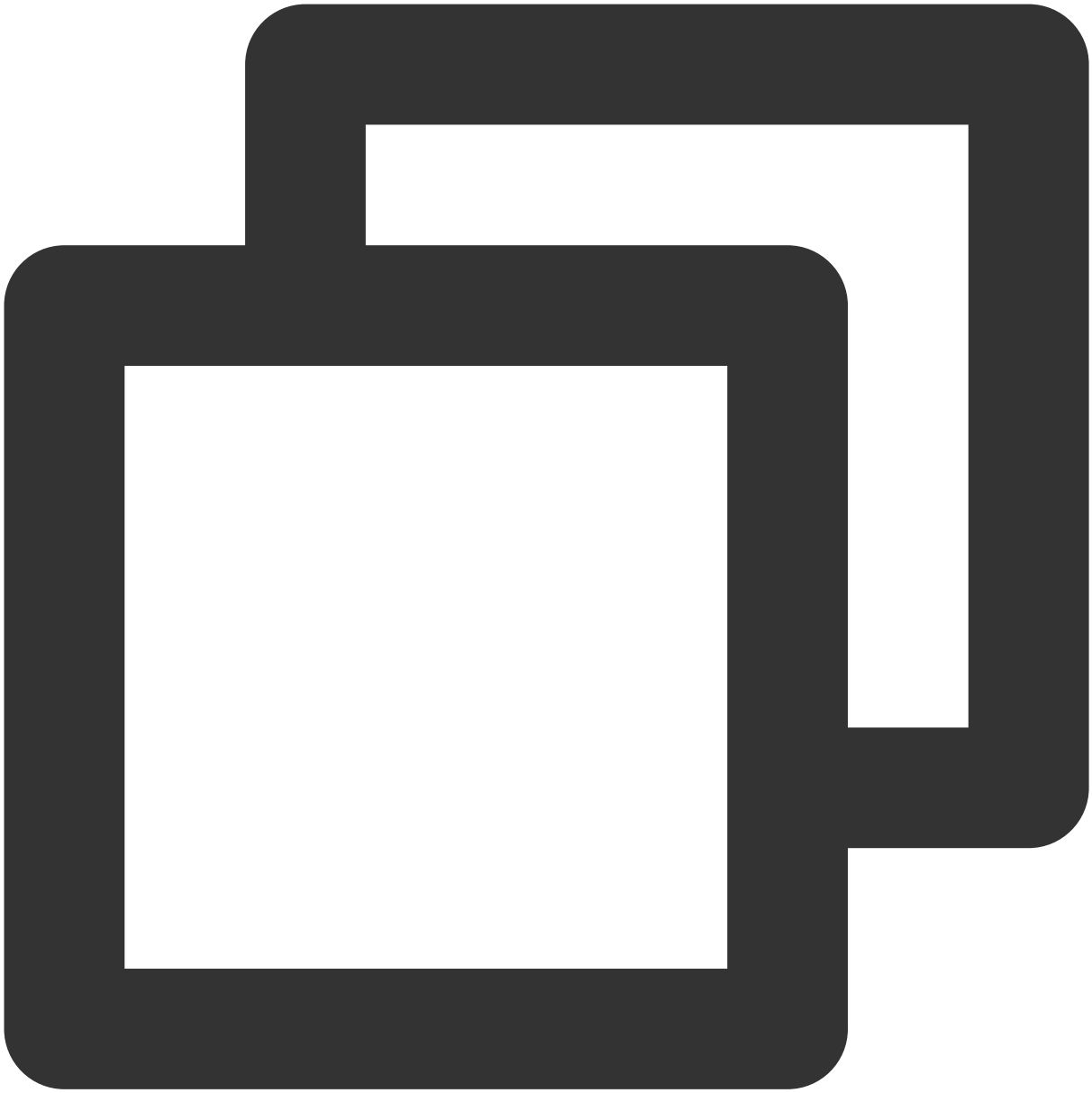
permission by the root account before it can normally access TKE in TMP; otherwise, an error will be displayed when it accesses the TKE cluster list.

When granting the `PassRole` permission to your sub-account, please make sure that your sub-account has the following permissions:

Permission Description	Granted Policy
The sub-account needs to be granted access to CAM before granting the `PassRole` permission to the sub-account by the root account can take effect	`QcloudCamReadOnlyAccess` or `QcloudCamFullAcces`
The Cloud Monitor policy depends on the Tencent Cloud service policy; therefore, before granting the `PassRole` permission to the sub-account, you need to make sure that the sub-account can normally access TKE resources	For more information, please see Permission Management

To ensure that the above permissions are granted successfully, please grant the `cam:PassRole` permission to the sub-account in the following steps.

1. Use the root account or a sub-account with administrative permissions to create the following custom policy:



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

2. After creation, associate the sub-account with the custom policy as instructed in CAM - Authorization Management. After granting the sub-account the `cam:PassRole` permission, access the **Integrate with TKE** page of the corresponding TMP instance, and an authorization window will pop up.

Grafana

Last updated：2024-08-07 22:06:47

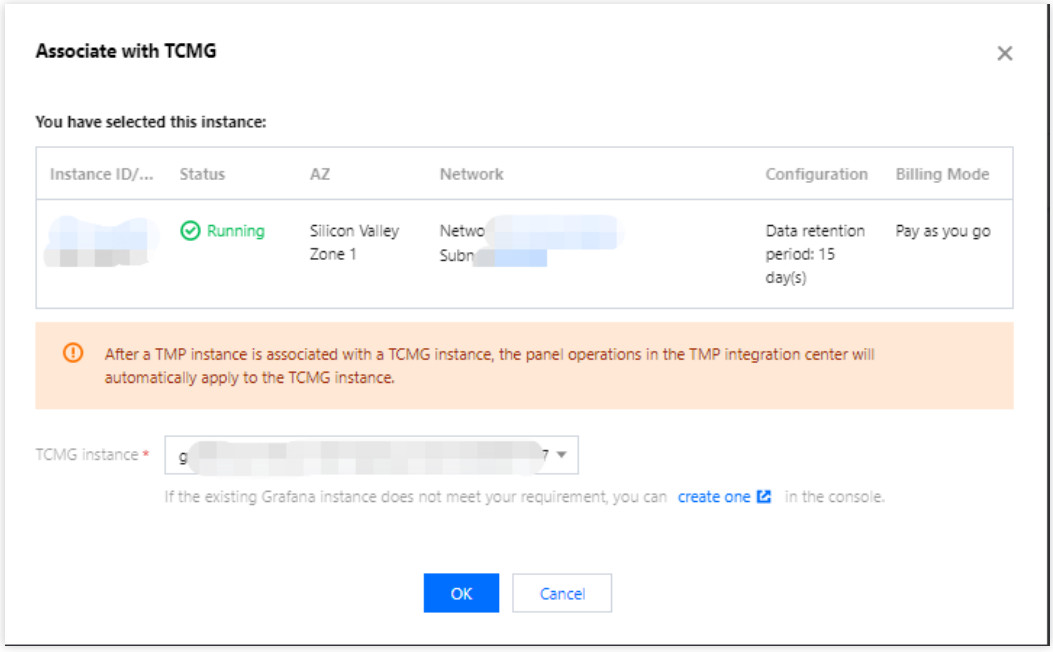
TMP is highly integrated with TencentCloud Managed Service for Grafana (TCMG). One TCMG instance can be bound by multiple TMP instances at the same time to visualized TMP data in a unified way.

Directions

TMP allows you to associate with a TCMG instance when creating a TMP instance. If you don't associate with the TCMG instance when purchasing the TMP service, you can follow the instructions below for binding.

Associating with a TCMG instance

1. Log in to the TMP console.
2. Find the corresponding TMP instance in the TMP instance list, and click **More**>**Grafana**>**Associate with TCMG** in the **Operation** column.
3. Select the TCMG instance in the pop-up window and click **OK**.

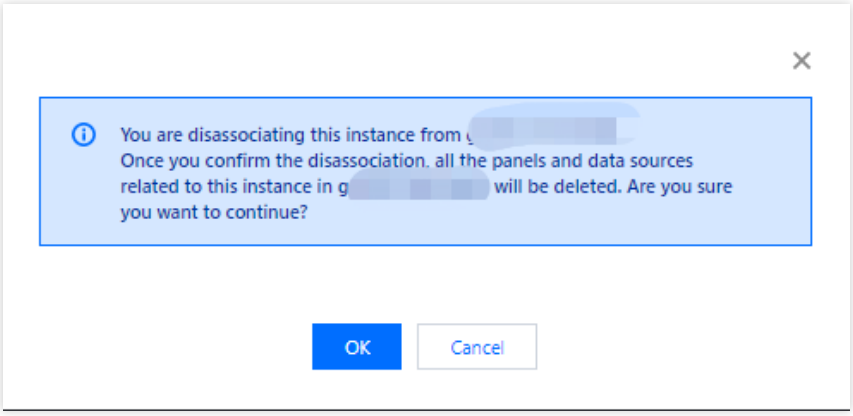


Note

You can only select the TCMG instance in the same VPC (private network) as the TMP instance. If there is no suitable TCMG instance, see Creating Instance to create one.

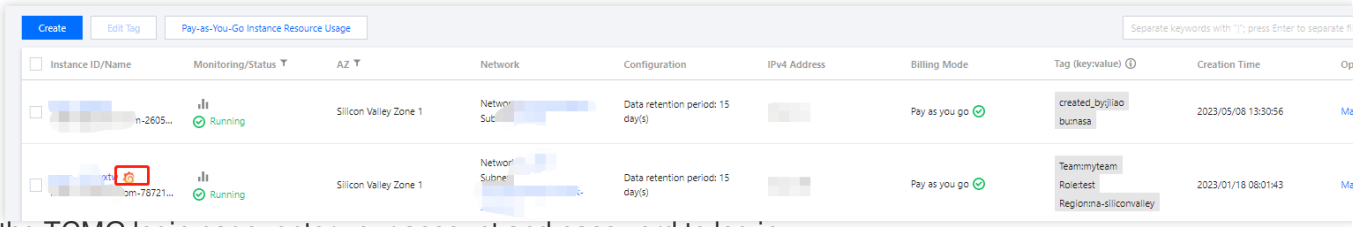
Disassociating from a TCMG instance

1. Log in to the TMP console.
2. Find the corresponding TMP instance in the TMP instance list, and click **More**>**Grafana**>**Disassociate from TCMG** in the **Operation** column.
3. In the pop-up window, click **OK**.



Logging in to the TCMG instance

1. Log in to the TMP console.
2. Find the corresponding TMP instance in the TMP instance list, and click the Grafana icon to the right of the instance ID/name.



3. On the TCMG login page, enter your account and password to log in.

Note

For more TCMG operations such as configuration and image rendering, see TencentCloud Managed Service for Grafana (TCMG).

API Guide
Overview

Last updated：2024-08-07 22:06:58

HTTP API

All stable HTTP APIs of Prometheus are under the path `/api/v1`. When you need to query monitoring data, you can request data through query APIs. To submit data, you can use the remote write protocol or Pushgateway.

Supported APIs

API	Description	Authentication Required	Method
/api/v1/query	Query	Yes	GET/POST
/api/v1/query_range	Range query	Yes	GET/POST
/api/v1/series	Series query	Yes	GET/POST
/api/v1/labels	Labels query	Yes	GET/POST
/api/v1/label/<label_name>/values	Label value query	Yes	GET
/api/v1/prom/write	Data submission through remote write	Yes	remote write
Pushgateway	Data submission through Pushgateway	Yes	SDK

Authentication Method

Authentication is enabled by default, so all APIs require authentication, and all authentication methods support bearer token and basic authentication.

Bearer token

A bearer token is generated as an instance is generated and can be queried in the console. For more information on bearer token, please see Bearer Authentication.

Basic auth

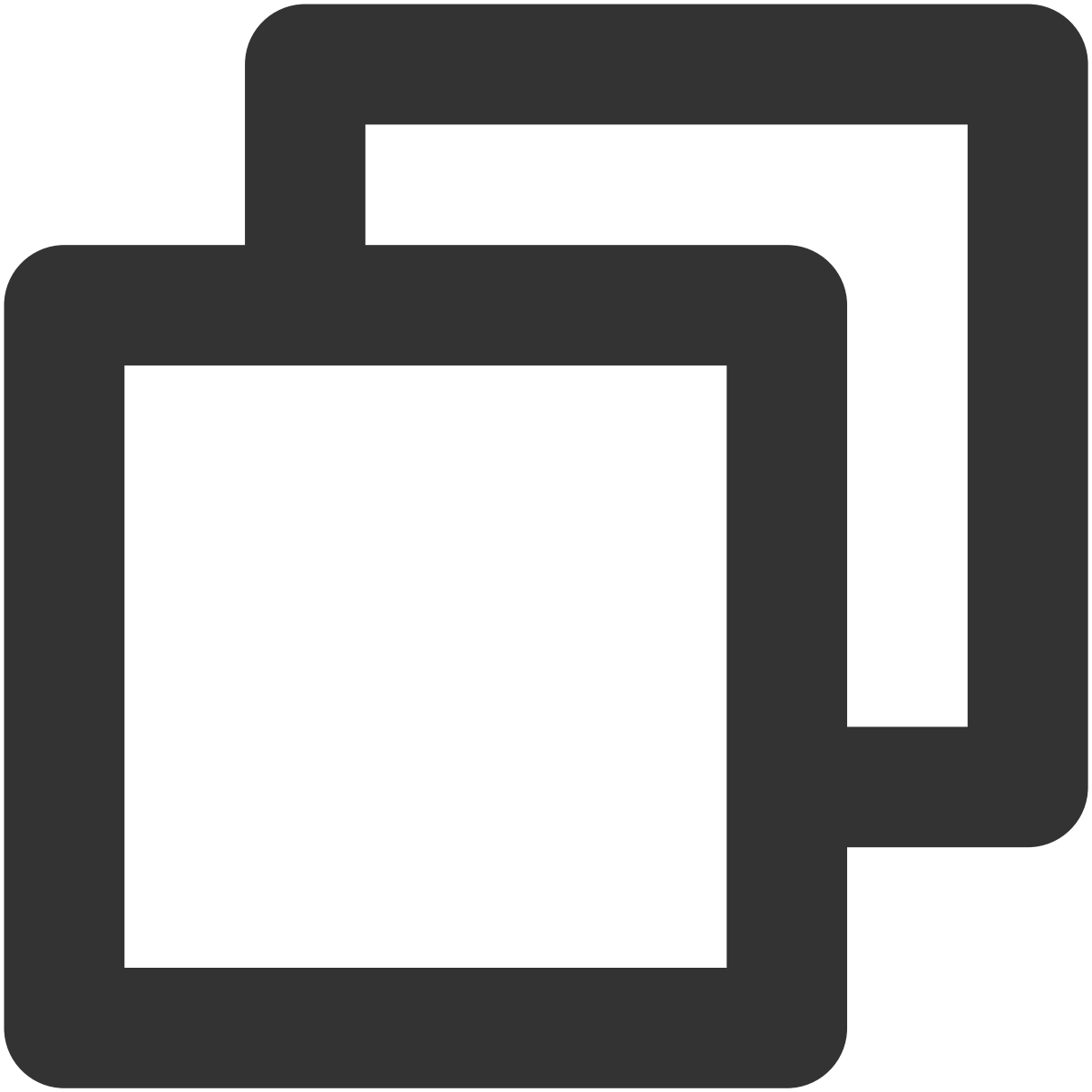
Basic auth is compatible with the native Prometheus query authentication method. The username is your `APPID`, and the password is the bearer token (generated when the instance is generated), which can be queried in the console. For more information on basic auth, please see Basic Authentication.

Data Return Format

The response data of all APIs is in JSON format. Every successful request will return a status code of `2xx`.
An invalid request will return a piece of JSON data containing an error object and a status code as described below:

Status Code	Description
401	Authentication failed
400	A parameter was missing or incorrect
422	An invalid expression couldn't be specified (RFC4918)
503	The query was unavailable or canceled

Below is a response template for invalid requests:



```
{
  "status": "success" | "error",
  "data": <data>,

  // When the `status` is `error`, the following data will be returned
  "errorType": "<string>",
  "error": "<string>",

  // When there is a warning message during request execution, this field will be f
  "warnings": ["<string>"]
}
```

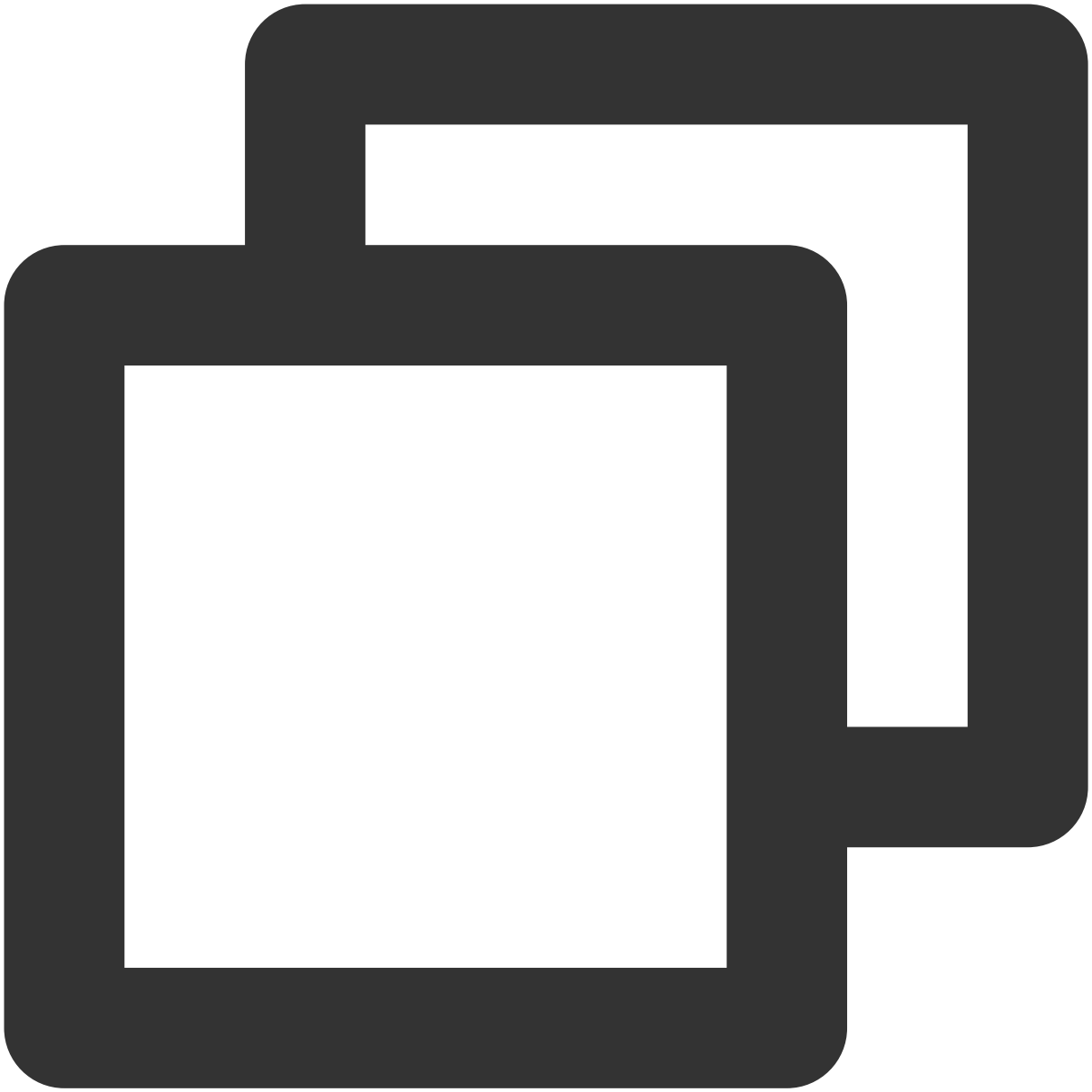
Writing Data

Last updated：2024-08-07 22:07:03

Overview

Similar to the scenario of data reporting by Flink jobs, you need to write data directly to Prometheus through APIs, as the lifecycle of these jobs may be very short, and it would be too late to wait for Prometheus to pull the data. To write data, you can directly use the remote write protocol or Pushgateway.

Remote Write



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

Remote write is a standard protocol of Prometheus. For more information, please see REMOTE WRITE TUNING.

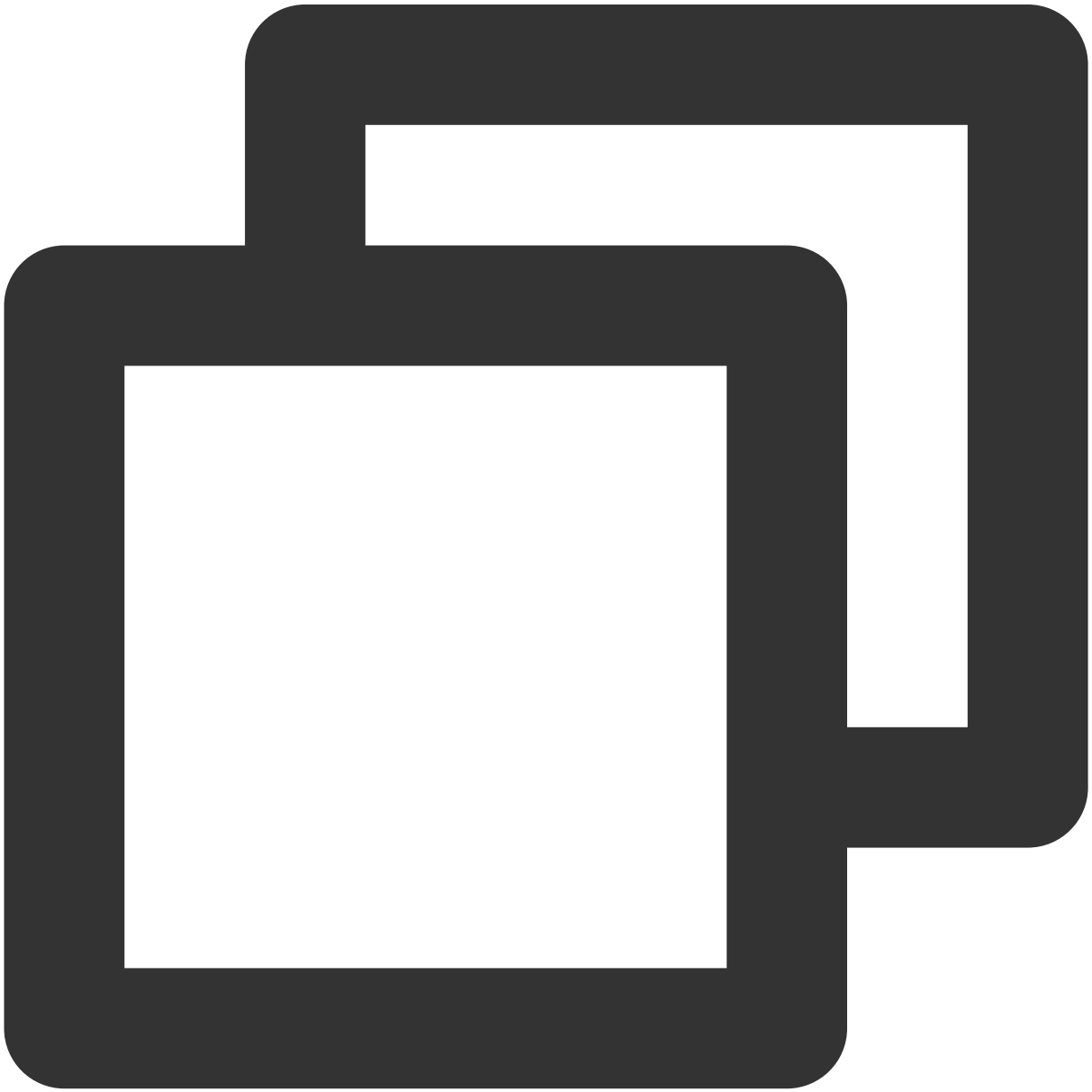
With remote write, you can write other Prometheus data in the VPC to TMP, which helps improve the data stability and make migration easier.

Pushgateway

Although the pull method is recommended in Prometheus, you may still need to use the push method in some scenarios. For more information, please see WHEN TO USE THE PUSHGATEWAY.
TMP is natively integrated with the Pushgateway module, which can directly push data to it. Below is a simple example of pushing data in Go. You need to change the variables of `$IP`, `$PORT`, `$APPID`, and `$TOKEN` to the authentication information of your own instance, which can be queried in the console.

Note:

We strongly recommend you replace `$INSTANCE` with an identifier of the current machine, such as `IP/Hostname`. If this `groupingKey` is not set, when multiple machines report data together, the data entries will overwrite each other.



```
package main

import (
    "fmt"
    "time"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/push"
    "github.com/prometheus/common/expfmt"
)

var completionTime = prometheus.NewGauge(prometheus.GaugeOpts{
    Name: "db_backup_last_completion_timestamp_seconds",
    Help: "The timestamp of the last successful completion of a DB backup.",
})

func do() {
    completionTime.SetToCurrentTime()
}

func ExamplePusher_Push() {
    if err := push.New("http://$IP:$PORT", "db_backup").
        BasicAuth("$APPID", "$TOKEN").
        Collector(completionTime).
        Grouping("instance", "$INSTANCE").
        Grouping("db", "customers").
        Format(expfmt.FmtText).
        Push(); err != nil {
        fmt.Println("Could not push completion time to Pushgateway:", err)
    }
}

func main() {
    do()
    ticker := time.NewTicker(2 * time.Second)
    done := make(chan bool)
    for {
        select {
        case <-done:
            return
        case <-ticker.C:
            ExamplePusher_Push()
        }
    }
}
```

Note:

You can customize the HTTP Client through the `Client` method for the object generated by `push.New`. We recommend you set an appropriate timeout period. In addition, if data is pushed, we recommend you use an async method for calls so as to avoid blocking the primary business process.

For other reporting scenarios, please see Custom Monitoring.

Querying Monitoring Data

Last updated：2024-08-07 22:07:09

Overview

When you need to query monitoring data, you can request data through query APIs.

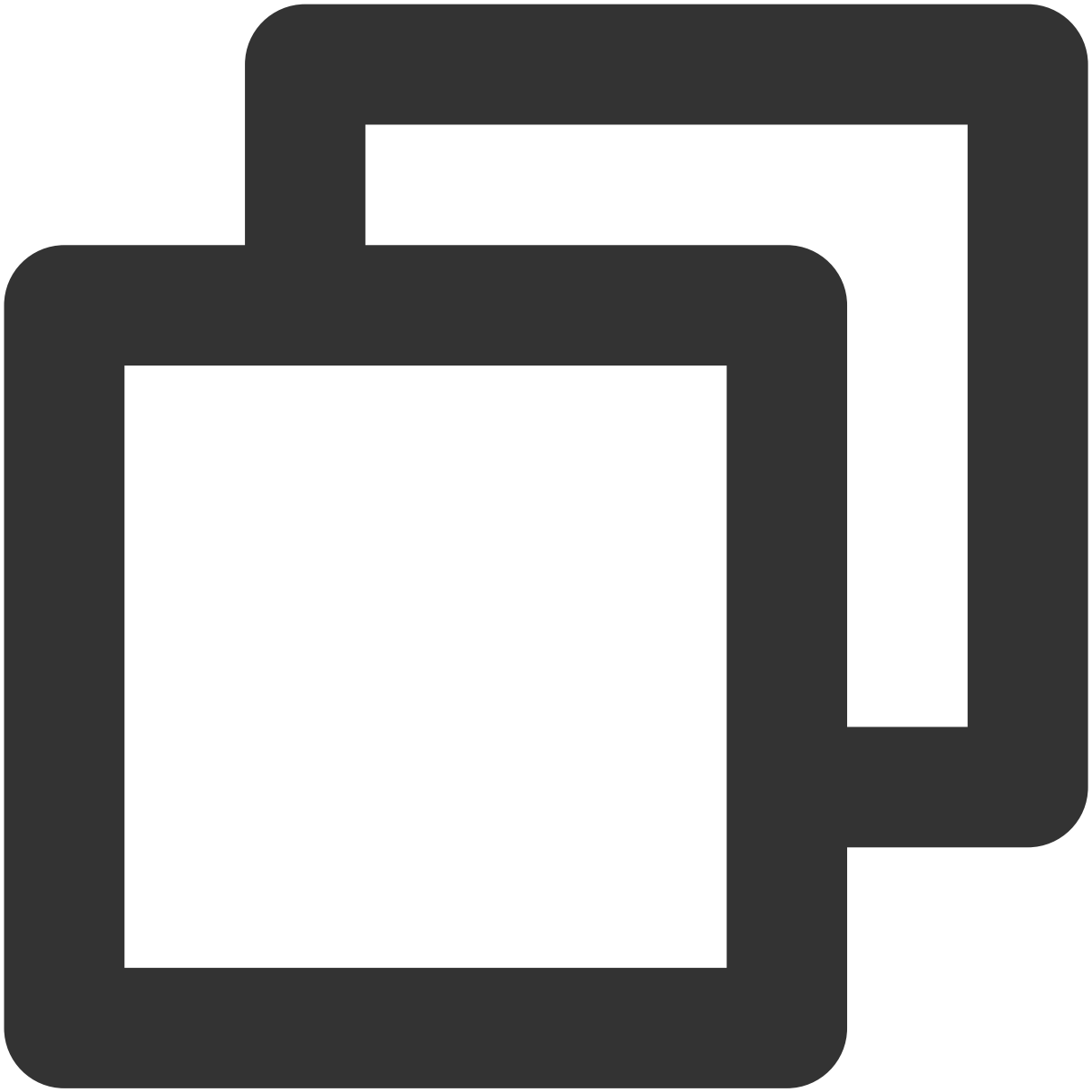
APPID/Token Acquisition Method

TMP uses `APPID` + `Token` to authenticate the access.

`APPID` can be obtained here.

`Token` can be obtained from the basic information of the corresponding TMP instance.

Query APIs



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

Query Parameters

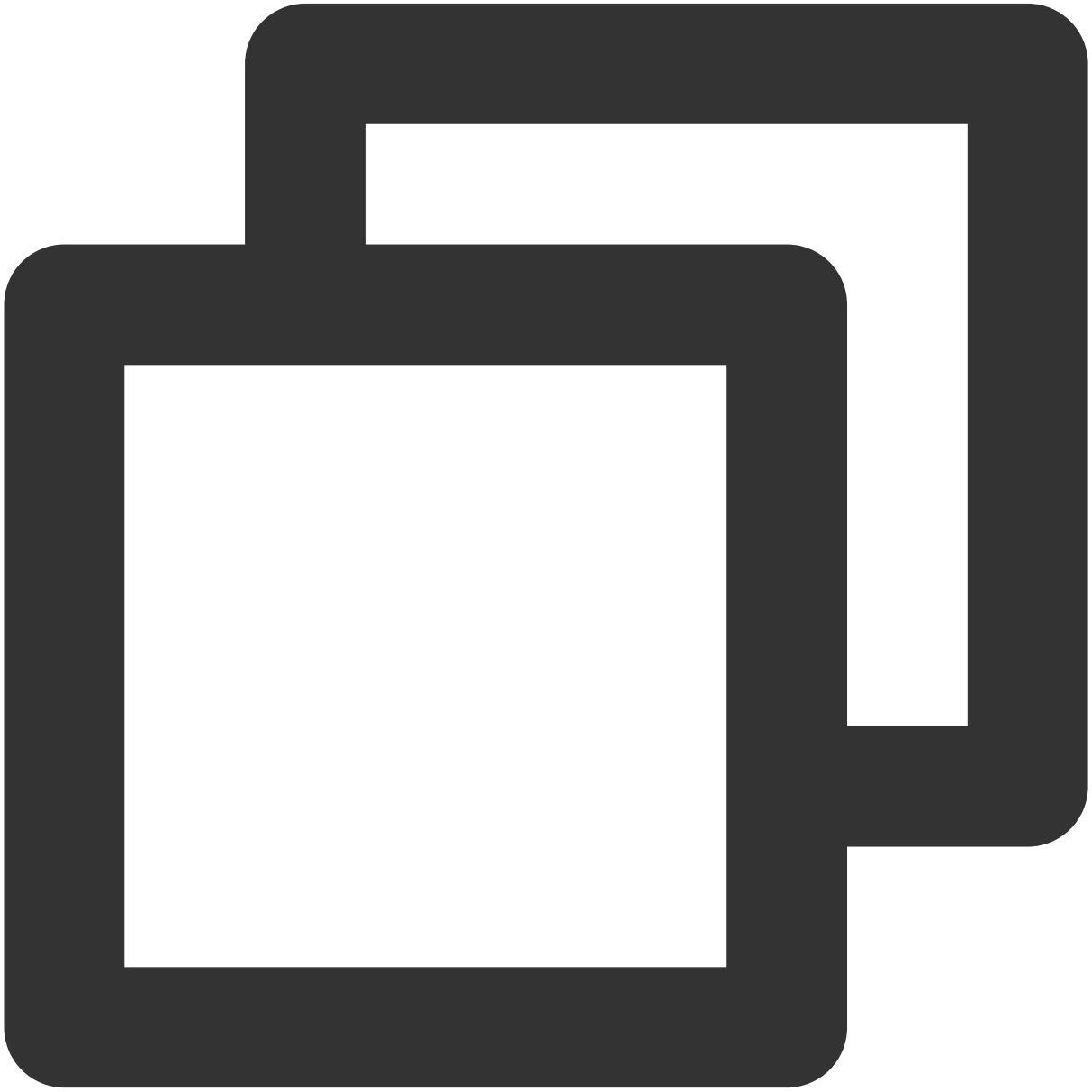
query=<string>: Prometheus: query expression.

time=<rfc3339 | unix_timestamp>: timestamp, which is optional.

timeout=<duration>: detection timeout period, which is optional and specified by the `-query.timeout` parameter by default.

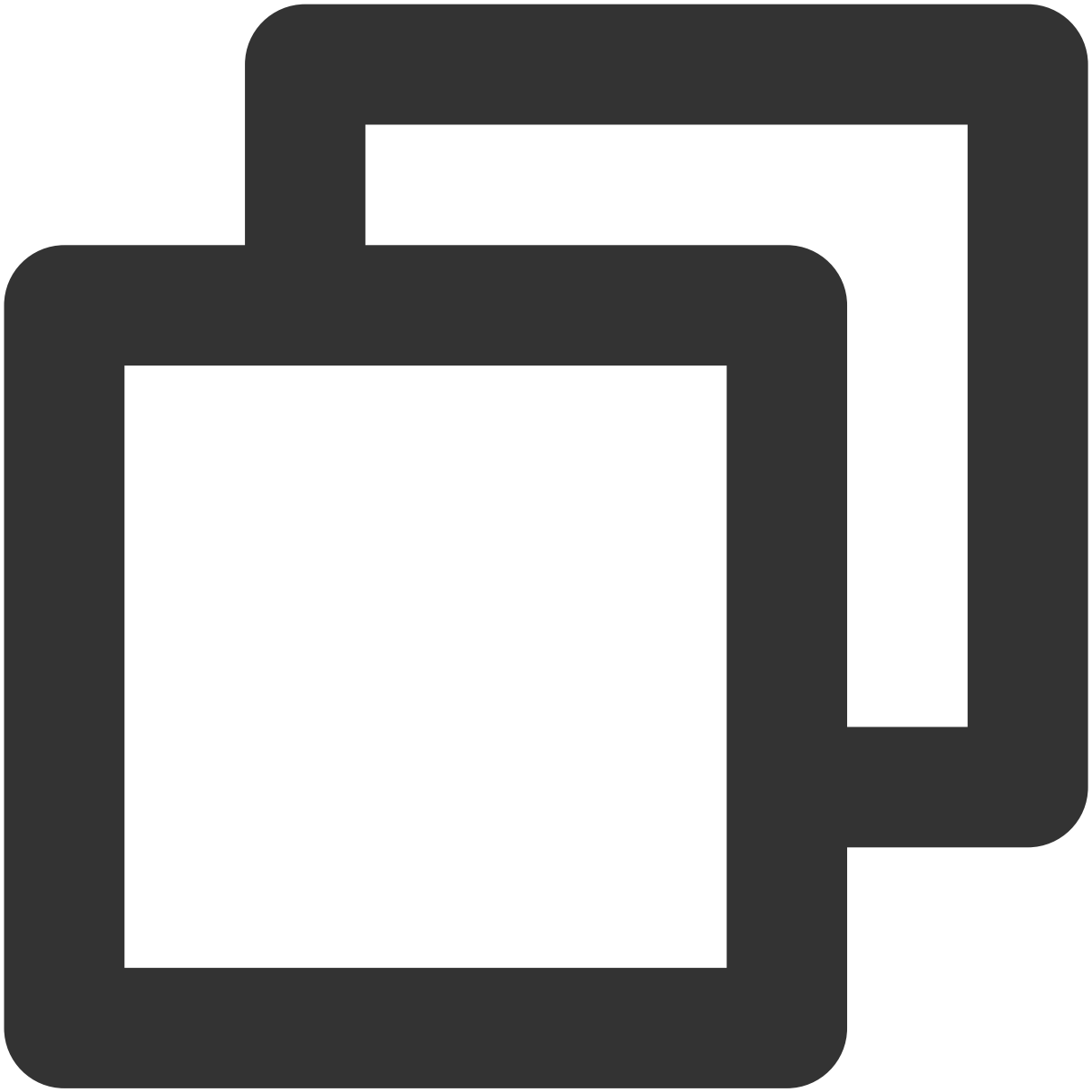
Sample Simple Query

You can use the following sample to query data through an API. The query service address and authentication information can be viewed in the corresponding instance's information in the console:



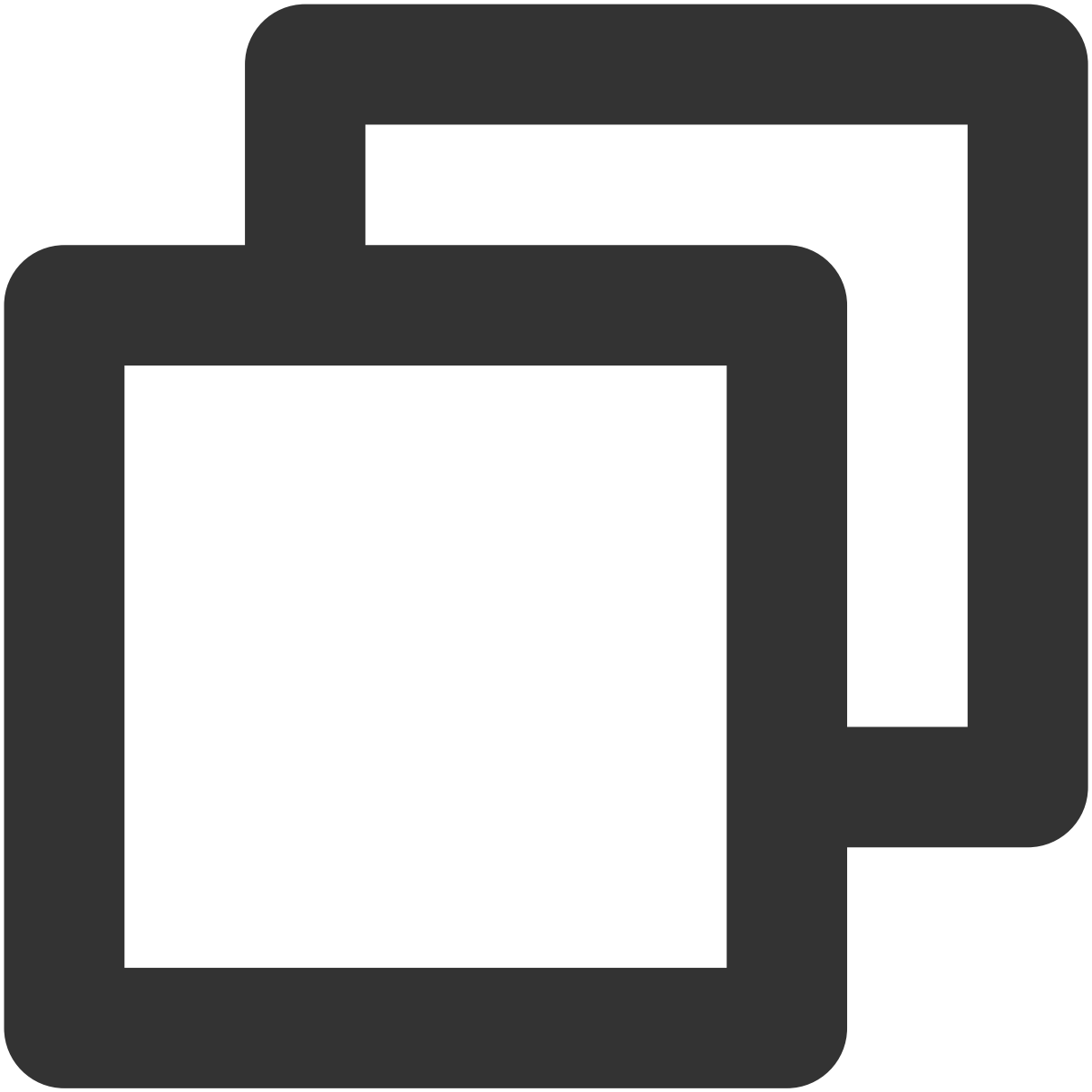
```
curl -u "appid:token" 'http://IP:PORT/api/v1/query?query=up'
```

If the returned status code is 401, please check whether the authentication information is correct.



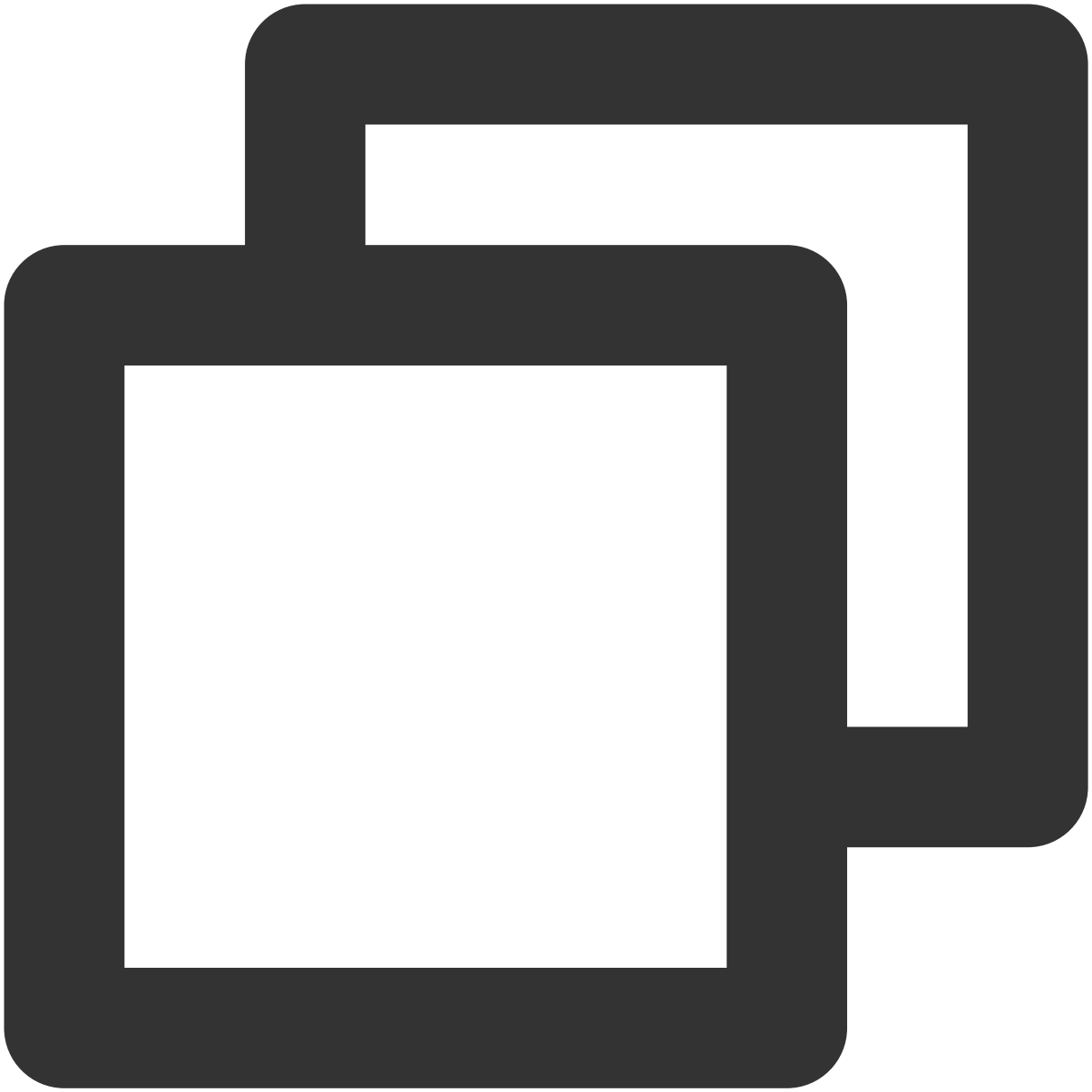
```
< HTTP/1.1 401 Unauthorized
< Content-Length: 0
```

Range Query



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

Querying data by time range is the most common query scenario. To do so, you can use the `/api/v1/query_range` API as shown below:



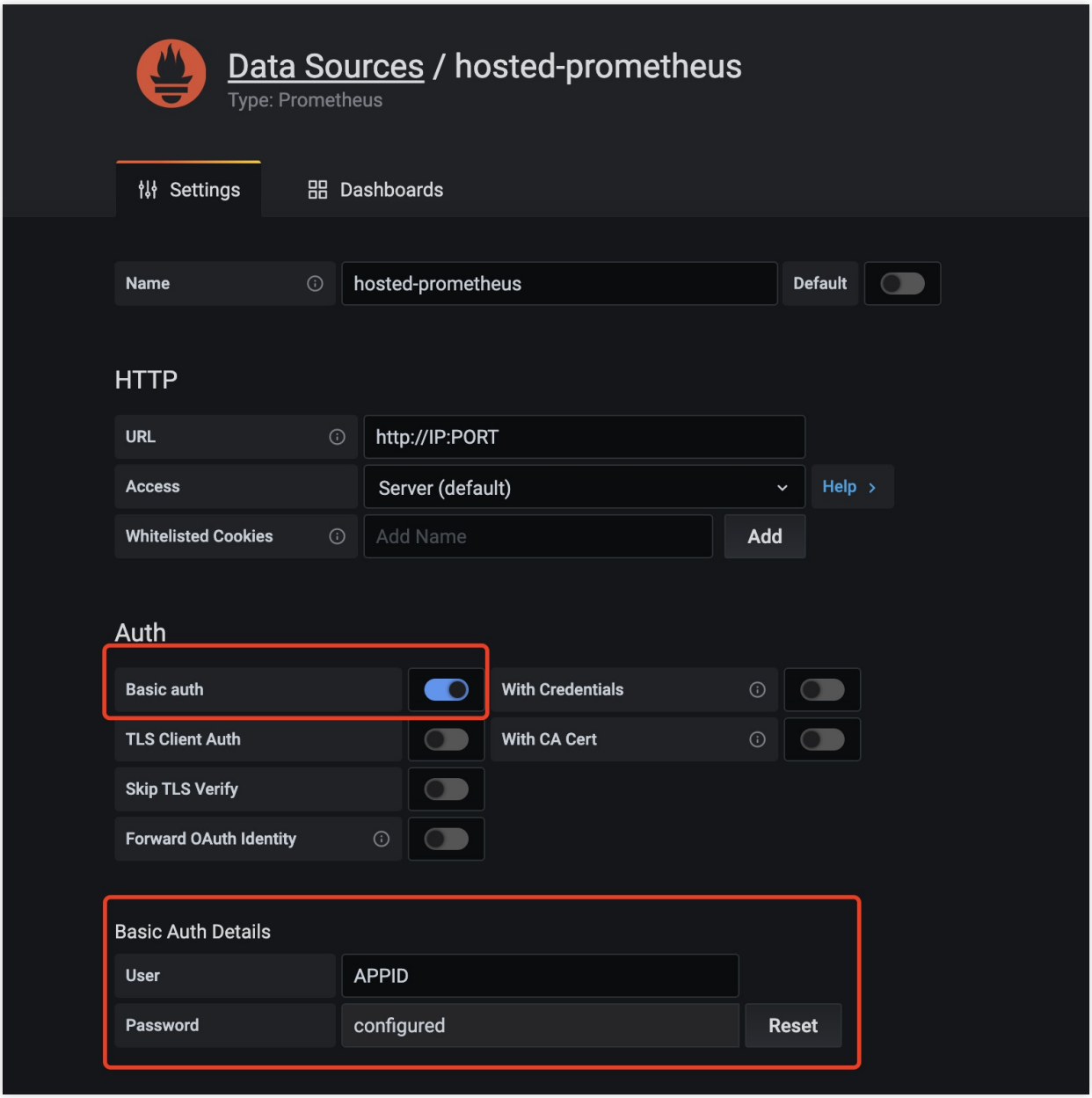
```
$ curl -u "appid:token" 'http://IP:PORT/api/v1/query_range?query=up&start=2015-07-0
{
   "status" : "success",
   "data" : {
      "resultType" : "matrix",
      "result" : [
         {
            "metric" : {
               "__name__" : "up",
               "job" : "prometheus",
               "instance" : "localhost:9090"
```

```
            },
            "values" : [
               [ 1435781430.781, "1" ],
               [ 1435781445.781, "1" ],
               [ 1435781460.781, "1" ]
            ]
         },
         {
            "metric" : {
               "__name__" : "up",
               "job" : "node",
               "instance" : "localhost:9091"
            },
            "values" : [
               [ 1435781430.781, "0" ],
               [ 1435781445.781, "0" ],
               [ 1435781460.781, "1" ]
            ]
         }
      ]
   }
}
```

Adding Data Source to Self-Built Grafana

You can add TMP as a data source in your self-built Grafana, and then you can view data in Grafana, provided that they are in the same VPC and can access each other over the network.

Enable the `BasicAuth` authentication method and enter the corresponding authentication information as shown below:



TKE Metrics

Free Metrics in Pay-as-You-Go Mode

Last updated：2024-08-07 22:07:22

For pay-as-you-go instances with a storage period of more than 15 days, storage fees for their free metrics will be charged based on the excessive storage period.

Configuration File	Metric Name
node-exporter	node_boot_time_seconds
node-exporter	node_context_switches_total
node-exporter	node_cpu_seconds_total
node-exporter	node_disk_io_now
node-exporter	node_disk_io_time_seconds_total
node-exporter	node_disk_io_time_weighted_seconds_total
node-exporter	node_disk_read_bytes_total
node-exporter	node_disk_read_time_seconds_total
node-exporter	node_disk_reads_completed_total
node-exporter	node_disk_write_time_seconds_total
node-exporter	node_disk_writes_completed_total
node-exporter	node_disk_written_bytes_total
node-exporter	node_filefd_allocated
node-exporter	node_filesystem_avail_bytes
node-exporter	node_filesystem_free_bytes
node-exporter	node_filesystem_size_bytes
node-exporter	node_load1
node-exporter	node_load15
node-exporter	node_load5

node-exporter	node_memory_Buffers_bytes
node-exporter	node_memory_Cached_bytes
node-exporter	node_memory_MemAvailable_bytes
node-exporter	node_memory_MemFree_bytes
node-exporter	node_memory_MemTotal_bytes
node-exporter	node_netstat_TcpExt_ListenDrops
node-exporter	node_netstat_Tcp_ActiveOpens
node-exporter	node_netstat_Tcp_CurrEstab
node-exporter	node_netstat_Tcp_InSegs
node-exporter	node_netstat_Tcp_OutSegs
node-exporter	node_netstat_Tcp_PassiveOpens
node-exporter	node_network_receive_bytes_total
node-exporter	node_network_transmit_bytes_total
node-exporter	node_sockstat_TCP_alloc
node-exporter	node_sockstat_TCP_inuse
node-exporter	node_sockstat_TCP_tw
node-exporter	node_sockstat_UDP_inuse
node-exporter	node_sockstat_sockets_used
node-exporter	node_uname_info
cadvisor	container_cpu_usage_seconds_total
cadvisor	container_fs_limit_bytes
cadvisor	container_fs_reads_bytes_total
cadvisor	container_fs_usage_bytes
cadvisor	container_fs_writes_bytes_total
cadvisor	container_memory_working_set_bytes

cadvisor	container_network_receive_bytes_total
cadvisor	container_network_receive_packets_dropped_total
cadvisor	container_network_receive_packets_total
cadvisor	container_network_transmit_bytes_total
cadvisor	container_network_transmit_packets_dropped_total
cadvisor	container_network_transmit_packets_total
cadvisor	machine_cpu_cores
cadvisor	machine_memory_bytes
kubelet	kubelet_cgroup_manager_duration_seconds_count
kubelet	kubelet_node_config_error
kubelet	kubelet_node_name
kubelet	kubelet_pleg_relist_duration_seconds_bucket
kubelet	kubelet_pleg_relist_duration_seconds_count
kubelet	kubelet_pleg_relist_interval_seconds_bucket
kubelet	kubelet_pod_start_duration_seconds_count
kubelet	kubelet_pod_worker_duration_seconds_count
kubelet	kubelet_running_containers
kubelet	kubelet_running_pods
kubelet	kubelet_runtime_operations_duration_seconds_bucket
kubelet	kubelet_runtime_operations_errors_total
kubelet	kubelet_runtime_operations_total
kubelet	process_cpu_seconds_total
kubelet	process_resident_memory_bytes
kubelet	rest_client_request_duration_seconds_bucket
kubelet	rest_client_requests_total

kubelet	storage_operation_duration_seconds_bucket
kubelet	storage_operation_duration_seconds_count
kubelet	storage_operation_errors_total
kubelet	volume_manager_total_volumes
kube-state-metrics	kube_job_status_succeeded
kube-state-metrics	kube_job_status_failed
kube-state-metrics	kube_job_status_active
kube-state-metrics	kube_node_status_capacity_cpu_cores
kube-state-metrics	kube_node_status_capacity_memory_bytes
kube-state-metrics	kube_node_status_allocatable_cpu_cores
kube-state-metrics	kube_node_status_allocatable_memory_bytes
kube-state-metrics	kube_pod_info
kube-state-metrics	kube_pod_owner
kube-state-metrics	kube_pod_status_phase
kube-state-metrics	kube_pod_container_status_waiting
kube-state-metrics	kube_pod_container_status_running
kube-state-metrics	kube_pod_container_status_terminated
kube-state-metrics	kube_pod_container_status_restarts_total
kube-state-metrics	kube_pod_container_resource_requests_cpu_cores
kube-state-metrics	kube_pod_container_resource_requests_memory_bytes
kube-state-metrics	kube_pod_container_resource_limits_cpu_cores
kube-state-metrics	kube_pod_container_resource_limits_memory_bytes
kube-state-metrics	kube_replicaset_owner
kube-state-metrics	kube_statefulset_status_replicas
kube-controller-manager	rest_client_request_duration_seconds_bucket

kube-controller-manager	rest_client_requests_total
kube-controller-manager	workqueue_adds_total
kube-controller-manager	workqueue_depth
kube-controller-manager	workqueue_queue_duration_seconds_bucket
kube-apiserver	apiserver_current_inflight_requests
kube-apiserver	apiserver_current_inqueue_requests
kube-apiserver	apiserver_init_events_total
kube-apiserver	apiserver_longrunning_gauge
kube-apiserver	apiserver_registered_watchers
kube-apiserver	apiserver_request_duration_seconds_bucket
kube-apiserver	apiserver_request_duration_seconds_sum
kube-apiserver	apiserver_request_duration_seconds_count
kube-apiserver	apiserver_request_filter_duration_seconds_bucket
kube-apiserver	apiserver_request_filter_duration_seconds_sum
kube-apiserver	apiserver_request_filter_duration_seconds_count
kube-apiserver	apiserver_request_total
kube-apiserver	apiserver_requested_deprecated_apis
kube-apiserver	apiserver_response_sizes_bucket
kube-apiserver	apiserver_response_sizes_sum
kube-apiserver	apiserver_response_sizes_count
kube-apiserver	apiserver_selfrequest_total
kube-apiserver	apiserver_tls_handshake_errors_total
kube-apiserver	apiserver_watch_events_sizes
kube-apiserver	apiserver_watch_events_sizes_bucket
kube-apiserver	apiserver_watch_events_sizes_sum

kube-apiserver	apiserver_watch_events_sizes_count
kube-apiserver	apiserver_watch_events_total

Recommended Common Metrics for TKE

Last updated：2024-08-07 22:07:28

Based on the usage of most users, it is recommend that you configure the following commonly used TKE metrics.

Note

The following are paid metrics. For the billing method of metrics, see Billing Mode and Resource Usage.

Configuration File	Metric Name	Metric Description
kubelet	kubelet_running_container_count	kubelet_running_container Number of containers curre running
kubelet	kubelet_running_pod_count	kubelet_running_pod_cour Number of pods currently r
kube-state-metrics	kube_pod_container_info	Information about a contair pod.
kube-state-metrics	kube_deployment_status_replicas	The number of replicas per deployment.
kube-state-metrics	kube_deployment_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_pod_start_time	Start time in unix timestamp pod.
kube-state-metrics	kube_pod_status_ready	Describes whether the pod ready to serve requests.
kube-state-metrics	kube_node_info	Information about a cluster
kube-state-metrics	kube_node_status_condition	The condition of a cluster n
kube-state-metrics	kube_deployment_status_replicas_updated	The number of updated rep per deployment.
kube-state-metrics	kube_deployment_status_replicas_available	The number of available re per deployment.
kube-state-metrics	kube_node_status_capacity_pods	The total pod resources of node.

kube-state-metrics	kube_pod_container_status_ready	Describes whether the cont readiness check succeede
kube-state-metrics	kube_deployment_spec_replicas	Number of desired pods for deployment.
kube-state-metrics	kube_pod_status_scheduled_time	Unix timestamp when pod r into scheduled status
kube-state-metrics	kube_node_status_allocatable_pods	The pod resources of a noc are available for scheduling
kube-state-metrics	kube_pod_container_resource_limits	The number of requested li resource by a container.
kube-state-metrics	node_filefd_maximum	File descriptor statistics: maximum.
kube-state-metrics	kube_pod_container_resource_requests	The number of requested r resource by a container.
kube-state-metrics	kube_namespace_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_deployment_status_replicas_unavailable	The number of unavailable replicas per deployment.
kube-state-metrics	kube_pod_created	Unix creation timestamp
kube-state-metrics	kube_pod_container_status_waiting_reason	Describes the reason the container is currently in wa state.
kube-state-metrics	kube_daemonset_status_desired_number_scheduled	The number of nodes that s be running the daemon poc
kube-state-metrics	kube_pod_restart_policy	Describes the restart policy use by this pod.
kube-state-metrics	kube_deployment_metadata_generation	Sequence number represe specific generation of the d state.
kube-state-metrics	kube_statefulset_status_update_revision	Indicates the version of the StatefulSet used to genera Pods in the sequence [repli updatedReplicas].

kube-state-metrics	kube_node_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_statefulset_replicas	Number of desired pods fo StatefulSet.
kube-state-metrics	kube_statefulset_status_observed_generation	The generation observed b StatefulSet controller.
kube-state-metrics	kube_pod_container_status_last_terminated_reason	Describes the last reason t container was in terminate
kube-state-metrics	kube_replicaset_spec_replicas	Number of desired pods fo ReplicaSet.
kube-state-metrics	kube_statefulset_created	Unix creation timestamp
kube-state-metrics	kube_statefulset_status_replicas_current	The number of current repl per StatefulSet.
kube-state-metrics	kube_statefulset_status_current_revision	Indicates the version of the StatefulSet used to genera Pods in the sequence [0].
kube-state-metrics	kube_statefulset_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_deployment_created	Unix creation timestamp
kube-state-metrics	kube_namespace_created	Unix creation timestamp
kube-state-metrics	kube_daemonset_status_number_ready	The number of nodes that s be running the daemon poc have one or more of the da pod running and ready.
kube-state-metrics	kube_deployment_status_observed_generation	The generation observed b deployment controller.
kube-state-metrics	kube_endpoint_info	Information about endpoint
kube-state-metrics	kube_statefulset_status_replicas_updated	The number of updated rep per StatefulSet.

kube-state-metrics	kube_statefulset_metadata_generation	Sequence number represe specific generation of the d state for the StatefulSet.
kube-state-metrics	kube_secret_created	Unix creation timestamp
kube-state-metrics	kube_endpoint_address_not_ready	Number of addresses not r in endpoint
kube-state-metrics	kube_secret_type	Type about secret.
kube-state-metrics	kube_deployment_spec_paused	Whether the deployment is paused and will not be proc by the deployment controlle
kube-state-metrics	kube_pod_container_status_terminated_reason	Describes the reason the container is currently in terminated state.
kube-state-metrics	kube_statefulset_status_replicas_ready	The number of ready replic StatefulSet.
kube-state-metrics	kube_endpoint_address_available	Number of addresses avail endpoint.
kube-state-metrics	kube_secret_info	Information about secret.
kube-state-metrics	kube_service_info	Information about service.
kube-state-metrics	kube_node_status_allocatable	The allocatable for differen resources of a node that ar available for scheduling.
kube-state-metrics	kube_endpoint_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_deployment_status_condition	The current status conditio deployment.
kube-state-metrics	kube_endpoint_created	Unix creation timestamp
kube-state-metrics	kube_replicaset_labels	Kubernetes labels converte Prometheus labels.

kube-state-metrics	kube_replicaset_metadata_generation	Sequence number represe specific generation of the d state.
kube-state-metrics	kube_namespace_status_phase	kubernetes namespace sta phase.
kube-state-metrics	kube_service_created	Unix creation timestamp
kube-state-metrics	kube_configmap_created	Unix creation timestamp
kube-state-metrics	kube_secret_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_deployment_spec_strategy_rollingupdate_max_surge	Maximum number of replic can be scheduled above th desired number of replicas a rolling update of a deploy
kube-state-metrics	kube_configmap_metadata_resource_version	Resource version represen specific version of the confi
kube-state-metrics	kube_pod_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_replicaset_status_replicas	The number of replicas per ReplicaSet.
kube-state-metrics	kube_node_created	Unix creation timestamp
kube-state-metrics	kube_service_spec_type	Type about service.
kube-state-metrics	kube_secret_metadata_resource_version	Resource version represen specific version of secret.
kube-state-metrics	kube_configmap_info	Information about configma
kube-state-metrics	kube_replicaset_status_observed_generation	The generation observed b ReplicaSet controller.
kube-state-metrics	kube_service_labels	Kubernetes labels converte Prometheus labels.

kube-state-metrics	kube_replicaset_created	Unix creation timestamp
kube-state-metrics	kube_deployment_spec_strategy_rollingupdate_max_unavailable	Maximum number of unava replicas during a rolling upd a deployment.
kube-state-metrics	kube_replicaset_status_ready_replicas	The number of ready replic ReplicaSet.
kube-state-metrics	kube_replicaset_status_fully_labeled_replicas	The number of fully labeled replicas per ReplicaSet.
kube-state-metrics	kube_pod_status_scheduled	Describes the status of the scheduling process for the
kube-state-metrics	kube_storageclass_created	Unix creation timestamp
kube-state-metrics	kube_daemonset_status_number_misscheduled	The number of nodes runni daemon pod but are not supposed to.
kube-state-metrics	kube_storageclass_labels	Kubernetes labels converte Prometheus labels.
kube-state-metrics	kube_node_status_capacity	The capacity for different resources of a node.
kube-state-metrics	kube_daemonset_status_current_number_scheduled	The number of nodes runni least one daemon pod and supposed to.
kube-state-metrics	kube_storageclass_info	Information about storagec
kube-state-metrics	kube_node_spec_unschedulable	Whether a node can sched new pods.
kube-state-metrics	kube_daemonset_status_number_available	The number of nodes that s be running the daemon poc have one or more of the da pod running and available.
kube-state-metrics	kube_daemonset_labels	Kubernetes labels converte Prometheus labels.
kube-state-	kube_daemonset_created	Unix creation timestamp

metrics		
kube-state-metrics	kube_daemonset_status_number_unavailable	The number of nodes that s be running the daemon poc have none of the daemon p running and available
kube-state-metrics	kube_daemonset_metadata_generation	Sequence number represe specific generation of the d state.
kube-state-metrics	kube_mutatingwebhookconfiguration_info	Information about the MutatingWebhookConfigur
kube-state-metrics	kube_mutatingwebhookconfiguration_created	Unix creation timestamp.
kube-state-metrics	kube_mutatingwebhookconfiguration_metadata_resource_version	Resource version represen specific version of the MutatingWebhookConfigur
kube-state-metrics	kube_daemonset_updated_number_scheduled	The total number of nodes are running updated daem
node-exporter	node_filesystem_files_free	Filesystem total free file no
node-exporter	node_filesystem_files	Filesystem total file nodes.
node-exporter	node_sockstat_UDP_mem_bytes	Number of UDP sockets in mem_bytes.
node-exporter	node_nf_conntrack_entries_limit	Maximum size of connectic tracking table.
node-exporter	node_memory_Shmem_bytes	Memory information field Shmem_bytes.
node-exporter	node_netstat_Tcp_RetransSegs	Statistic TcpRetransSegs.
node-exporter	node_sockstat_TCP_mem_bytes	Number of TCP sockets in mem_bytes.
node-exporter	node_network_info	Non-numeric data from /sys/class/net/<iface>

node-exporter	node_filesystem_readonly	Filesystem read-only status
node-exporter	node_exporter_build_info	A metric with a constant '1' labeled by version
node-exporter	node_network_iface_link_mode	iface_link_mode value of /sys/class/net/<iface>.
node-exporter	node_network_receive_packets_total	Network device statistic receive_packets.
node-exporter	node_network_transmit_packets_total	Network device statistic transmit_packets.
node-exporter	node_memory_Mlocked_bytes	Memory information field Mlocked_bytes.
node-exporter	node_network_iface_id	iface_id value of /sys/class/net/<iface>.
node-exporter	node_memory_WritebackTmp_bytes	Memory information field WritebackTmp_bytes.
node-exporter	kube_service_status_load_balancer_ingress	Service load balancer ingre status
node-exporter	node_vmstat_pgpgout	/proc/vmstat information fie pgpgout.
node-exporter	node_nf_conntrack_entries	Number of currently allocat flow entries for connection tracking.
node-exporter	node_memory_Inactive_file_bytes	Memory information field Inactive_file_bytes.
node-exporter	node_memory_SwapFree_bytes	Memory information field SwapFree_bytes.
node-exporter	node_sockstat_TCP_mem	Number of TCP sockets in mem.
node-exporter	node_memory_Slab_bytes	Memory information field Slab_bytes.
node-exporter	node_network_transmit_errs_total	Network device statistic transmit_errs.

node-exporter	node_memory_Active_bytes	Memory information field Active_bytes.
node-exporter	node_procs_blocked	Number of processes block waiting for I/O to complete.
node-exporter	node_sockstat_UDP_mem	Number of UDP sockets in mem.
node-exporter	node_timex_maxerror_seconds	Maximum error in seconds.
node-exporter	node_memory_Inactive_bytes	Memory information field Inactive_bytes.
node-exporter	node_network_receive_errs_total	Network device statistic receive_errs.
node-exporter	node_memory_Unevictable_bytes	Memory information field Unevictable_bytes.
node-exporter	node_memory_KernelStack_bytes	Memory information field KernelStack_bytes.
node-exporter	node_procs_running	Number of processes in ru state.
node-exporter	node_memory_SwapTotal_bytes	Memory information field SwapTotal_bytes.
node-exporter	node_netstat_IpExt_OutOctets	Statistic IpExtOutOctets.
node-exporter	node_memory_Active_file_bytes	Memory information field Active_file_bytes.
node-exporter	node_memory_SwapCached_bytes	Memory information field SwapCached_bytes.
node-exporter	node_netstat_Icmp_InMsgs	Statistic IcmpInMsgs.
node-exporter	node_forks_total	Total number of forks.
node-exporter	node_sockstat_RAW_inuse	Number of RAW sockets in inuse.

node-exporter	node_time_seconds	System time in seconds sir epoch (1970).
node-exporter	node_vmstat_pgpgin	/proc/vmstat information fie pgpgin.
node-exporter	node_memory_Mapped_bytes	Memory information field Mapped_bytes.
node-exporter	node_memory_SUnreclaim_bytes	Memory information field SUnreclaim_bytes.
node-exporter	node_memory_HardwareCorrupted_bytes	Memory information field HardwareCorrupted_bytes
node-exporter	node_memory_PageTables_bytes	Memory information field PageTables_bytes.
node-exporter	node_netstat_Udp6_InDatagrams	Statistic Udp6InDatagrams
node-exporter	node_netstat_Icmp_OutMsgs	Statistic IcmpOutMsgs.
node-exporter	node_netstat_Udp6_NoPorts	Statistic Udp6NoPorts.
node-exporter	node_memory_AnonPages_bytes	Memory information field AnonPages_bytes.
node-exporter	node_memory_Committed_AS_bytes	Memory information field Committed_AS_bytes.
node-exporter	node_netstat_TcpExt_ListenOverflows	Statistic TcpExtListenOver
node-exporter	node_netstat_UdpLite_InErrors	Statistic UdpLiteInErrors.
node-exporter	node_entropy_available_bits	Bits of available entropy.
node-exporter	node_memory_Inactive_anon_bytes	Memory information field Inactive_anon_bytes.
node-exporter	node_vmstat_pswpin	/proc/vmstat information fie pswpin.

node-exporter	node_memory_AnonHugePages_bytes	Memory information field AnonHugePages_bytes.
node-exporter	node_memory_SReclaimable_bytes	Memory information field SReclaimable_bytes.
node-exporter	node_netstat_IpExt_InOctets	Statistic IpExtInOctets.
node-exporter	node_netstat_Udp_NoPorts	Statistic UdpNoPorts.
node-exporter	node_timex_sync_status	Is clock synchronized to a r server (1 = yes).
node-exporter	node_memory_CommitLimit_bytes	Memory information field CommitLimit_bytes.
node-exporter	node_memory_VmallocChunk_bytes	Memory information field VmallocChunk_bytes.
node-exporter	node_netstat_Udp_InDatagrams	Statistic UdpInDatagrams.
node-exporter	node_netstat_Icmp6_InErrors	Statistic Icmp6InErrors.
node-exporter	node_netstat_Icmp6_OutMsgs	Statistic Icmp6OutMsgs.
node-exporter	node_netstat_UdpLite6_InErrors	Statistic UdpLite6InErrors.
node-exporter	node_netstat_TcpExt_SyncookiesSent	Statistic TcpExtSyncookies
node-exporter	node_netstat_Tcp_InErrs	Statistic TcpInErrs.
node-exporter	node_intr_total	Total number of interrupts serviced.
node-exporter	node_timex_offset_seconds	Time offset in between loca system and reference clock
node-exporter	node_memory_Bounce_bytes	Memory information field Bounce_bytes.

node-exporter	node_memory_Writeback_bytes	Memory information field Writeback_bytes.
node-exporter	node_netstat_Udp_OutDatagrams	Statistic UdpOutDatagram
node-exporter	node_netstat_Icmp6_InMsgs	Statistic Icmp6InMsgs.
node-exporter	node_netstat_Ip6_OutOctets	Statistic Ip6OutOctets.
node-exporter	node_netstat_Ip_Forwarding	Statistic IpForwarding.
node-exporter	node_sockstat_TCP_orphan	Number of TCP sockets in orphan.
node-exporter	node_netstat_Ip6_InOctets	Statistic Ip6InOctets.
node-exporter	node_netstat_TcpExt_SyncookiesFailed	Statistic TcpExtSyncookiesFailed.
node-exporter	node_netstat_Udp_InErrors	Statistic UdpInErrors.
node-exporter	node_vmstat_pgmajfault	/proc/vmstat information fie pgmajfault.
node-exporter	node_network_transmit_drop_total	Network device statistic transmit_drop.
node-exporter	node_vmstat_pswpout	/proc/vmstat information fie pswpout.
node-exporter	node_network_up	Value is 1 if operstate is 'up
node-exporter	node_memory_NFS_Unstable_bytes	Memory information field NFS_Unstable_bytes.
node-exporter	node_memory_VmallocTotal_bytes	Memory information field VmallocTotal_bytes.
node-exporter	node_sockstat_FRAG_inuse	Number of FRAG sockets i inuse.

node-exporter	node_memory_Dirty_bytes	Memory information field Dirty_bytes.
node-exporter	node_netstat_Udp6_InErrors	Statistic Udp6InErrors.
node-exporter	node_netstat_TcpExt_SyncookiesRecv	Statistic TcpExtSyncookies
node-exporter	node_netstat_Udp6_OutDatagrams	Statistic Udp6OutDatagran
node-exporter	node_memory_HugePages_Rsvd	Memory information field HugePages_Rsvd.
node-exporter	node_arp_entries	ARP entries by device
node-exporter	node_network_carrier	carrier value of /sys/class/net/<iface>.
node-exporter	node_timex_pps_stability_exceeded_total	Pulse per second count of stability limit exceeded eve
node-exporter	node_network_receive_compressed_total	Network device statistic receive_compressed.
node-exporter	node_network_transmit_carrier_total	Network device statistic transmit_carrier.
node-exporter	node_memory_DirectMap2M_bytes	Memory information field DirectMap2M_bytes.
node-exporter	node_memory_Hugepagesize_bytes	Memory information field Hugepagesize_bytes.
node-exporter	node_network_address_assign_type	address_assign_type value /sys/class/net/<iface>.
node-exporter	node_network_receive_multicast_total	Network device statistic receive_multicast.
node-exporter	node_network_transmit_compressed_total	Network device statistic transmit_compressed.
node-exporter	node_memory_DirectMap4k_bytes	Memory information field DirectMap4k_bytes.

node-exporter	node_network_transmit_queue_length	transmit_queue_length valu /sys/class/net/<iface>.
node-exporter	node_memory_HugePages_Free	Memory information field HugePages_Free.
node-exporter	node_network_receive_frame_total	Network device statistic receive_frame.
node-exporter	node_memory_HugePages_Total	Memory information field HugePages_Total.
node-exporter	node_network_flags	flags value of /sys/class/net/<iface>.
node-exporter	node_network_receive_fifo_total	Network device statistic receive_fifo.
node-exporter	node_scrape_collector_duration_seconds	node_exporter: Duration of collector scrape.
node-exporter	node_network_speed_bytes	speed_bytes value of /sys/class/net/<iface>.
node-exporter	node_sockstat_UDPLITE_inuse	Number of UDPLITE socke state inuse.
node-exporter	node_cpu_guest_seconds_total	Seconds the cpus spent in (VMs) for each mode.
node-exporter	node_filesystem_device_error	Whether an error occurred getting statistics for the giv device.
node-exporter	node_scrape_collector_success	node_exporter: Whether a collector succeeded.
node-exporter	node_network_transmit_fifo_total	Network device statistic transmit_fifo.
node-exporter	node_vmstat_pgfault	/proc/vmstat information fie pgfault.
node-exporter	node_network_device_id	device_id value of /sys/class/net/<iface>.
node-exporter	node_network_protocol_type	protocol_type value of /sys/class/net/<iface>.

node-exporter	node_network_receive_drop_total	Network device statistic receive_drop.
node-exporter	node_timex_estimated_error_seconds	Estimated error in seconds
node-exporter	node_disk_writes_merged_total	The number of writes merg
node-exporter	node_network_transmit_colls_total	Network device statistic transmit_colls.
node-exporter	node_timex_tick_seconds	Seconds between clock tic
node-exporter	node_textfile_scrape_error	1 if there was an error oper reading a file
node-exporter	node_network_iface_link	iface_link value of /sys/class/net/<iface>.
node-exporter	node_disk_reads_merged_total	The total number of reads merged.
node-exporter	node_timex_status	Value of the status array bi
node-exporter	node_netstat_Icmp_InErrors	Statistic IcmpInErrors.
node-exporter	node_memory_Active_anon_bytes	Memory information field Active_anon_bytes.
node-exporter	node_timex_pps_frequency_hertz	Pulse per second frequenc
node-exporter	node_network_mtu_bytes	mtu_bytes value of /sys/class/net/<iface>.
node-exporter	node_timex_tai_offset_seconds	International Atomic Time (offset.
node-exporter	node_timex_pps_jitter_total	Pulse per second count of j limit exceeded events.
node-exporter	node_timex_pps_jitter_seconds	Pulse per second jitter.

node-exporter	node_network_net_dev_group	net_dev_group value of /sys/class/net/<iface>.
node-exporter	node_network_dormant	dormant value of /sys/class/net/<iface>.
node-exporter	node_timex_pps_calibration_total	Pulse per second count of calibration intervals.
node-exporter	node_timex_pps_shift_seconds	Pulse per second interval duration.
node-exporter	node_timex_pps_error_total	Pulse per second count of calibration errors.
node-exporter	node_memory_VmallocUsed_bytes	Memory information field VmallocUsed_bytes.
node-exporter	node_timex_frequency_adjustment_ratio	Local clock frequency adjustment.
node-exporter	node_sockstat_FRAG_memory	Number of FRAG sockets i memory.
node-exporter	node_memory_HugePages_Surp	Memory information field HugePages_Surp.
node-exporter	node_timex_loop_time_constant	Phase-locked loop time co
node-exporter	node_timex_pps_stability_hertz	Pulse per second stability

Recommended Common Metrics for TKE

Last updated：2024-08-07 22:07:38

Cluster Monitoring Overview

Chart Name	Query Statement
CPU Requests Commitment	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"}) / sum(kube_
CPU Limits Commitment	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster"}) / sum(kube_nod
Memory Utilisation	1 - sum(:node_memory_MemAvailable_bytes:sum{cluster="$cluster"}) / sum(node_memory
Memory Requests Commitment	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) / sum(ku
Memory Limits Commitment	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster"}) / sum(kube_
Node Count	count(kube_node_info{cluster="$cluster"})
Pod Count	count(kube_pod_info{cluster="$cluster"})

Node Request CPU Average Percent	avg(sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"})by (node
Node Request Memory Average Percent	avg(sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"})by (node)/sum(kube_node_status_capacity_memory_bytes{cluster="$cluster"})by(node))
API Server Success Request Percent	sum(irate(apiserver_request_total{cluster="$cluster",code=~"20.*",verb=~"GET|LIST"}[5m] [5m]))
Namespace Overview	count(kube_pod_info{cluster="$cluster"}) by (namespace)
	count(kube_service_info{cluster="$cluster"}) by(namespace)
	count(kube_pod_container_info{cluster="$cluster"}) by(namespace)
	count(kube_configmap_info{cluster="$cluster"}) by(namespace)
	count(kube_secret_info{cluster="$cluster"}) by(namespace)
	count(kube_deployment_created{cluster="$cluster"}) by (namespace)
	count(kube_statefulset_created{cluster="$cluster"}) by (namespace)
	count(kube_job_created{cluster="$cluster"}) by (namespace)
	count(kube_cronjob_created{cluster="$cluster"}) by (namespace)
	count(kube_pod_status_ready{cluster="$cluster",condition="false"}==1) by(namespace) - (by(namespace) or vector(0)) or count(kube_pod_status_ready{cluster="$cluster",condition

	count(kube_deployment_status_replicas_ready{cluster="$cluster"}<kube_deployment_spe
	count(kube_statefulset_status_replicas_ready{cluster="$cluster"}<kube_statefulset_replica
	count(kube_daemonset_status_number_unavailable{cluster="$cluster"}>0)by(namespace)
	count(kube_job_status_failed{cluster="$cluster"} == 1) by (namespace)
	count(kube_daemonset_created{cluster="$cluster"}) by (namespace)
	count(kube_persistentvolumeclaim_info{cluster="$cluster"}) by (namespace)
CPU Usage	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{clus
CPU Quota	sum(kube_pod_owner{cluster="$cluster"}) by (namespace)
	count(avg(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster"}) by (wo
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{clus
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"}) by (namespa

	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{clus sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster"}) by (namespa
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster"}) by (namespace)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{clus sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster"}) by (namespace)
Memory Usage (working_set)	sum(container_memory_working_set_bytes{cluster="$cluster", container!="", container!="F
Memory Requests	sum(kube_pod_owner{cluster="$cluster"}) by (namespace)
	count(avg(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster"}) by (wor
	sum(container_memory_rss{cluster="$cluster", container!="", container!="POD"}) by (name
	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) by (name
	sum(container_memory_rss{cluster="$cluster", container!="", container!="POD"}) by (name sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster"}) by (name
	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster"}) by (namespa
	sum(container_memory_rss{cluster="$cluster", container!="", container!="POD"}) by (name sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster"}) by (namespa
Node Memory Usage (Top 10)	sum(label_replace(topk(10, 1-(node_memory_MemAvailable_bytes{cluster="$cluster"} / no "instance", "(.*)"))by(node_ip)

Node CPU Usage (Top 10)	topk(10, sum(label_replace(1 - sum(rate(node_cpu_seconds_total{cluster="$cluster",mode sum(rate(node_cpu_seconds_total{cluster="$cluster"}[1m])) by (instance),"host_ip","$1","in
Node Disk Usage (Top 10)	topk(10, sum(label_replace(1- node_filesystem_free_bytes{cluster="$cluster",mountpoint="/"}/node_filesystem_size_bytes (.*)"))by(host_ip))
Node Network In (Top 10)	topk(10, sum(label_replace(max(irate(node_network_receive_bytes_total{cluster="$cluster
Node Network Out (Top 10)	topk(10, sum(label_replace(max(irate(node_network_transmit_bytes_total{cluster="$cluste
Node Sockets Count(Top 10)	topk(10, sum(label_replace(max(node_sockstat_TCP_alloc{cluster="$cluster"}) by (instanc
Container Memory Usage(Top10)	topk(10, sum (container_memory_working_set_bytes{cluster="$cluster",container !="",cont
Container Memory Usage/Limit(Top10)	topk(10, avg(container_memory_working_set_bytes{cluster="$cluster",container!=""}/(cont namespace))
Container CPU Usage(Top10)	topk(10, sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",container !="",co
Container Network	topk(10, sum(irate(container_network_receive_bytes_total{cluster="$cluster",image!="",co
	-topk(10, sum(irate(container_network_transmit_bytes_total{cluster="$cluster",image!="",c
Container Memory Usage/Limit (Top 10)	topk(10, avg(container_memory_working_set_bytes{cluster="$cluster",container!=""}/(cont namespace))
Container CPU Usage (Top 10)	topk(10, sum(irate(container_cpu_usage_seconds_total{cluster="$cluster",container!="",co
Container Socket Count(Top 10)	topk(10, sum(container_sockets{cluster="$cluster",container!=""}) by (container,pod,names

Cluster Namespace Dashboard

Chart Name	Query Statement
CPU Usage	sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$name
CPU Usage/Request(%)	sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$name [2m]))/sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",nam
CPU Usage/Limit(%)	sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$name [2m]))/sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namesp
CPU Request	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace
CPU Limit	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namespace=~'
Cluster Available	sum(sum(kube_node_status_capacity{resource="cpu",cluster="$cluster",namespace=~
StatefulSet Created	count(kube_statefulset_created{cluster="$cluster",namespace="$namespace"}) or on()
Pod Created	count(kube_pod_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0)
Containers	count(kube_pod_container_info{cluster="$cluster",namespace="$namespace"}) or on()
DaemonSet Created	count(kube_daemonset_created{cluster="$cluster",namespace="$namespace"}) or on()
Job Created	count(kube_job_info{cluster="$cluster",namespace="$namespace"})or on() vector(0)
Job Active	count(kube_job_status_active{cluster="$cluster",namespace="$namespace"}==1)or on

Cron Job Created	count(kube_cronjob_created{cluster="$cluster",namespace="$namespace"}) or on() vec
Cron Job Active	count(kube_cronjob_status_active{cluster="$cluster",namespace="$namespace"}==1) (
Unbound PVC	count(kube_persistentvolumeclaim_status_phase{phase!="Bound", cluster="$cluster",n
PersistentVolumeClaim Created	count(kube_persistentvolumeclaim_info{cluster="$cluster",namespace="$namespace"})
Service Created	count(kube_service_info{cluster="$cluster",namespace="$namespace"}) or on() vector(
LoadBalancer Created	count(kube_service_spec_type{type="LoadBalancer", cluster="$cluster",namespace="$
Ingress Created	count(kube_ingress_info{cluster="$cluster",namespace="$namespace"})or on() vector((
ConfigMap Created	count(kube_configmap_info{cluster="$cluster",namespace="$namespace"})
Secret Created	count(kube_secret_info{cluster="$cluster",namespace="$namespace"}) or on() vector(0
PVC Storage Requests Total	sum(kube_persistentvolumeclaim_resource_requests_storage_bytes{cluster="$cluster"
Pod NotReady	count(kube_pod_status_ready{condition="false", cluster="$cluster",namespace="$name by(namespace) or vector(0)) or count(kube_pod_status_ready{condition="false", cluster
Pod UnSchedulable	count(kube_pod_status_unschedulable{cluster="$cluster",namespace="$namespace"})
Deployment NotReady	count(sum(kube_deployment_status_replicas_ready{cluster="$cluster",namespace="$r vector(0)

Daemonset NotReady	count(kube_daemonset_status_number_unavailable{cluster="$cluster",namespace="$r
Job Failed	count(kube_job_status_failed{cluster="$cluster",namespace="$namespace"} == 1)
CPU Usage	sum(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace=~"$name
CPU Quota	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{c
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespac
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{c sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespac
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{c sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="
Memory Usage	sum(container_memory_working_set_bytes{cluster="$cluster",namespace=~"$namespa
Memory Usage/Request(%)	sum(container_memory_working_set_bytes{cluster="$cluster",namespace=~"$namespa unit="byte", resource="memory"}) or on() vector(0)
Memory Usage/Limit(%)	sum(container_memory_working_set_bytes{cluster="$cluster",namespace=~"$namespa unit="byte", resource="memory"}) or on() vector(0)
Memory Request	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",names
Memory Limit	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespac

Cluster Available	sum(sum(kube_node_status_capacity{resource="memory"}) by (node) + sum(kube_noc
Memory Usage (w/o cache)	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespa
	scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard"
	scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard"
Memory Quota	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespa
	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", names
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespa sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",names
	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespac
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespa
	sum(container_memory_rss{cluster="$cluster", namespace="$namespace",container!='
	sum(container_memory_cache{cluster="$cluster", namespace="$namespace",containe
	sum(container_memory_swap{cluster="$cluster", namespace="$namespace",container
Containers	group by (image, container, container_id, pod) (kube_pod_container_info{cluster="$clus
	group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (kube_pod_container_status_running{cluster="$cluster",namespace="$namespace"})

group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (kube_pod_container_status_restarts_total{cluster="$cluster",namespace="$namespac
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar max(irate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$names
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (max(irate(container_cpu_usage_seconds_total{container!="",container!="POD",cluster (max(container_spec_cpu_quota{container!="",container!="POD",cluster="$cluster",nar
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (max(irate(container_cpu_usage_seconds_total{container!="",container!="POD",cluster (kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (kube_pod_container_resource_limits{resource="cpu",cluster="$cluster",namespace="$
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar !="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,contain
group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (max(container_memory_working_set_bytes{container!="",container!="POD",cluster="$!="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod, contair

	group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar (kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace
	group by (container, container_id, pod) (kube_pod_container_info{cluster="$cluster",nar !="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,contain

API Server (Independent Cluster)

Chart Name	Query Statement
Availability > 99.000%	1 - ((sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{cluster="$cluster"}[5m])) - sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_bucket{le="1", cluster="$cluster"}[5m]))) + sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",code=~"5.. cluster="$cluster"}[5m]) or vector(0))) / sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube- apiserver",cluster="$cluster"}[5m]))
ErrorBudget > 99.000%	100 * (1 - ((sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{cluster="$cluster"}[5m])) - sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_bucket{le="1", cluster="$cluster"}[5m])))

	+ sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",code=~"5..cluster="$cluster"}[5m]) or vector(0))) / sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",cluster="$cluster"}[5m])) -0.990000)
Read Availability	1 - ((sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{verb=~"LIST|GET", cluster="$cluster"}[5m]) - sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_bucket{verb=~"LIST\|GET",le="1", cluster="$cluste[5m]))) + sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",verb=~"LIST|GET",code=~"5..", cluster="$cluster"}[5m]) or vector(0))) / sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET", cluster="$cluster"}[5m]))
Read SLI - Requests	sum by (code) (rate(apiserver_request_total{job="kube-apiserver",verb=~"LIST|GET",cluster="$cluster"}[5m]))
Read SLI - Errors	sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"LIST|GET",code=~"5..",cluster="$cluster"}[5m]))/ sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"LIST\|GET",cluster="$cluster"}[5m]))
Read SLI - Duration	histogram_quantile(0.99, sum by (le, resource,cluster,cluster_type) (rate(apiserver_request_duration_seconds_bucket{job="kube-apiserver",verb=~"LIST|GET",cluster="$cluster"}[5m]))) > 0
Write Availability	1 - ((sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_count{verb=~"POST|PUT|PATCH|DELETE", cluster="$cluster"}[5m])) - sum by (cluster,cluster_type) (increase(apiserver_request_duration_seconds_bucket{verb=~"POST|PUT|PATCH|DELETE",le="

	cluster="$cluster"}[5m]))) + sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",code=~"5..", cluster="$cluster"}[5m]) or vector(0))) / sum by (cluster,cluster_type) (increase(apiserver_request_total{job="kube-apiserver",verb=~"POST|PUT|PATCH|DELETE", cluster="$cluster"}[5m]))
Write SLI - Requests	sum by (code) (rate(apiserver_request_total{job="kube-apiserver",verb=~"POST|PUT|PATCH|DELETE",cluster="$cluster"}[5m]))
Write SLI - Errors	sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"POST|PUT|PATCH|DELETE",code=~"5..",cluster="$cluster"}[5m]))/ sum by (resource) (rate(apiserver_request_total{job="kube-apiserver",verb=~"POST\|PUT\|PATCH\|DELETE",cluster="$cluster"}[5m]))
Write SLI - Duration	histogram_quantile(0.99, sum by (le, resource,cluster,cluster_type) (rate(apiserver_request_duration_seconds_bucket{job="kube-apiserver",verb=~"POST|PUT|PATCH|DELETE",cluster="$cluster"}[5m]))) > 0
Work Queue Add Rate	sum(rate(workqueue_adds_total{job="kube-apiserver", instance=~"$instance", cluster=~"$cluster"}[5m])) by (instance, name)
Work Queue Depth	sum(rate(workqueue_depth{job="kube-apiserver", instance=~"$instance", cluster=~"$cluster"}[5m])) by (instance, name)
Work Queue Latency	histogram_quantile(0.99, sum(rate(workqueue_queue_duration_seconds_bucket{job="kube-apiserver", instance=~"$instance", cluster=~"$cluster"}[5m])) by (instance, name, le))
Memory	process_resident_memory_bytes{job="kube-apiserver",instance=~"$instance", cluster=~"$cluster"}
CPU usage	rate(process_cpu_seconds_total{job="kube-apiserver",instance=~"$instance", cluster=~"$cluster"}[5m])

Controller Manager (Independent Cluster)

Chart Name	Query Statement	Metrics Us

Up	sum(up{cluster=~"$cluster",job="kube-controller-manager"})	up
Work Queue Add Rate	sum(rate(workqueue_adds_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name)	workqueue
Work Queue Depth	sum(rate(workqueue_depth{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name)	workqueue
Work Queue Latency	histogram_quantile(0.99, sum(rate(workqueue_queue_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance"}[5m])) by (instance, name, le))	workqueue
Kube API Request Rate	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"2.."}[5m]))	rest_client
	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"3.."}[5m]))	rest_client
	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"4.."}[5m]))	rest_client
	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance",code=~"5.."}[5m]))	rest_client
Post Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance", verb="POST"}[5m])) by (verb, url, le))	rest_client
Get Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-controller-manager", instance=~"$instance", verb="GET"}[5m])) by (verb, url, le))	rest_client
Memory	process_resident_memory_bytes{cluster=~"$cluster",job="kube-controller-manager",instance=~"$instance"}	process_re
CPU usage	rate(process_cpu_seconds_total{cluster=~"$cluster",job="kube-controller-manager",instance=~"$instance"}[5m])	process_c

Kubelet

Chart Name	Query Statement
Up	sum(up{cluster="$cluster", job="kubelet"})
Running Pods	sum(kubelet_running_pods{cluster="$cluster", job="kubelet", instance=~"$instance"})
Running Container	sum(kubelet_running_containers{cluster="$cluster", job="kubelet", instance=~"$instance"})
Actual Volume Count	sum(volume_manager_total_volumes{cluster="$cluster", job="kubelet", instance=~"$instance", state="actual_state_of_world"})
Desired Volume Count	sum(volume_manager_total_volumes{cluster="$cluster", job="kubelet", instance=~"$instance",state="desired_state_of_world"})
Config Error Count	sum(rate(kubelet_node_config_error{cluster="$cluster", job="kubelet", instance=~"$instance"}[5m]))
Operation Rate	sum(rate(kubelet_runtime_operations_total{cluster="$cluster",job="kubelet",instance=~"$instance"}[(operation_type, instance)
Operation Error Rate	sum(rate(kubelet_runtime_operations_errors_total{cluster="$cluster",job="kubelet",instance=~"$inst (instance, operation_type)
Operation duration 99th quantile	histogram_quantile(0.99, sum(rate(kubelet_runtime_operations_duration_seconds_bucket{cluster="$cluster",job="kubelet",ins [5m])) by (instance, operation_type, le))
Pod Start Rate	sum(rate(kubelet_pod_start_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$ (instance)
	sum(rate(kubelet_pod_worker_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~ by (instance)
Pod Start Duration	histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$ (instance, le))
	histogram_quantile(0.99,

	sum(rate(kubelet_pod_worker_duration_seconds_bucket{cluster="$cluster",job="kubelet",instance= by (instance, le))
Storage Operation Rate	sum(rate(storage_operation_duration_seconds_count{cluster="$cluster",job="kubelet",instance=~"$ (instance, operation_name, volume_plugin)
Storage Operation Error Rate	sum(rate(storage_operation_errors_total{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m operation_name, volume_plugin)
Storage Operation Duration 99th quantile	histogram_quantile(0.99, sum(rate(storage_operation_duration_seconds_bucket{cluster="$cluster", instance=~"$instance"}[5m])) by (instance, operation_name, volume_plugin, le))
Cgroup manager operation rate	sum(rate(kubelet_cgroup_manager_duration_seconds_count{cluster="$cluster", job="kubelet", insta [5m])) by (instance, operation_type)
Cgroup manager 99th quantile	histogram_quantile(0.99, sum(rate(kubelet_cgroup_manager_duration_seconds_bucket{cluster="$c job="kubelet", instance=~"$instance"}[5m])) by (instance, operation_type, le))
PLEG relist rate	sum(rate(kubelet_pleg_relist_duration_seconds_count{cluster="$cluster", job="kubelet", instance=~' (instance)
PLEG relist interval	histogram_quantile(0.99, sum(rate(kubelet_pleg_relist_interval_seconds_bucket{cluster="$cluster",job="kubelet",instance=~" (instance, le))
PLEG relist duration	histogram_quantile(0.99, sum(rate(kubelet_pleg_relist_duration_seconds_bucket{cluster="$cluster",job="kubelet",instance=~ (instance, le))
RPC Rate	sum(rate(rest_client_requests_total{cluster="$cluster",job="kubelet", instance=~"$instance",code=~'
	sum(rate(rest_client_requests_total{cluster="$cluster",job="kubelet", instance=~"$instance",code=~'
Request duration 99th quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster="$cluster" instance=~"$instance"}[5m])) by (instance, verb, url, le))
Memory	process_resident_memory_bytes{cluster="$cluster",job="kubelet",instance=~"$instance"}

CPU usage	rate(process_cpu_seconds_total{cluster="$cluster",job="kubelet",instance=~"$instance"}[5m])
Goroutines	go_goroutines{cluster="$cluster",job="kubelet",instance=~"$instance"}

Proxy(Non-default Installation Component)

Chart Name	Query Statement
Up	sum(up{job="kube-proxy"})
Rules Sync Rate	sum(rate(kubeproxy_sync_proxy_rules_duration_seconds_count{job="kube-proxy", instance=~"$instance"}[5m]))
Rule Sync Latency 99th Quantile	histogram_quantile(0.99,rate(kubeproxy_sync_proxy_rules_duration_seconds_bucket{job="kube proxy", instance=~"$instance"}[5m]))
Network Programming Rate	sum(rate(kubeproxy_network_programming_duration_seconds_count{job="kube-proxy", instance=~"$instance"}[5m]))
Network Programming Latency 99th Quantile	histogram_quantile(0.99, sum(rate(kubeproxy_network_programming_duration_seconds_bucket{job="kube-proxy", instance=~"$instance"}[5m])) by (instance, le))
Kube API Request Rate	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"2.."}[5m]))
	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"3.."}[5m]))
	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"4.."}[5m]))
	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"5.."}[5m]))
Post Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{job="kube-proxy",instance=~"$instance",verb="POST"}[5m])) by (verb, url, le))
Kube API	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"2.."}[5m]))

Request Rate	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"3.."}[5m]))
	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"4.."}[5m]))
	sum(rate(rest_client_requests_total{job="kube-proxy", instance=~"$instance",code=~"5.."}[5m]))
Post Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{job="kube-proxy",instance=~"$instance",verb="POST"}[5m])) by (verb, url, le))
Get Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{job="kube-proxy", instance=~"$instance", verb="GET"}[5m])) by (verb, url, le))
Memory	process_resident_memory_bytes{job="kube-proxy",instance=~"$instance"}
CPU usage	rate(process_cpu_seconds_total{job="kube-proxy",instance=~"$instance"}[5m])

Scheduler(Independent Cluster)

Chart Name	Query Statement	Metrics Us
Up	sum(up{cluster=~"$cluster", job="kube-scheduler"})	up
Kube API Request Rate	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"2.."}[5m]))	rest_client
	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"3.."}[5m]))	rest_client
	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"4.."}[5m]))	rest_client
	sum(rate(rest_client_requests_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance",code=~"5.."}[5m]))	rest_client
Post Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="POST"}[5m])) by (verb, url, le))	rest_client

Get Request Latency 99th Quantile	histogram_quantile(0.99, sum(rate(rest_client_request_duration_seconds_bucket{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance", verb="GET"}[5m])) by (verb, url, le))	rest_client
Memory	process_resident_memory_bytes{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance"}	process_re
CPU usage	rate(process_cpu_seconds_total{cluster=~"$cluster",job="kube-scheduler", instance=~"$instance"}[5m])	process_c

Cluster Node Monitoring Details

Chart Name	Query Statement
Server resource overview table	node_uname_info{job=~"$job", cluster=~"$cluster"} - 0
	node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"} - 0
	count(node_cpu_seconds_total{job=~"$job",mode='system',cluster=~"$cluster"}) by (instance)
	sum(time() - node_boot_time_seconds{job=~"$job",cluster=~"$cluster"})by(instance)
	max((node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?|xfs"}- node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?|xfs"}) *100/(node_filesy {job=~"$job",cluster=~"$cluster",fstype=~"ext.?|xfs"}+ (node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?|xfs"}- node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"ext.?|xfs"}))))by(instance)
	(1 - avg(irate(node_cpu_seconds_total{job=~"$job",mode="idle",cluster=~"$cluster"}[5m])) by (insta
	(1 - (node_memory_MemAvailable_bytes{job=~"$job",cluster=~"$cluster"} / (node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"})))* 100

	node_load5{job=~"$job",cluster=~"$cluster"}
	max(irate(node_disk_written_bytes_total{job=~"$job",cluster=~"$cluster"}[5m])) by (instance)
	max(irate(node_network_receive_bytes_total{job=~"$job",cluster=~"$cluster"}[5m])*8) by (instance
	max(irate(node_network_transmit_bytes_total{job=~"$job",cluster=~"$cluster"}[5m])*8) by (instanc
	node_load5{job=~"$job",cluster=~"$cluster"}
Overall total load and average CPU utilization	count(node_cpu_seconds_total{job=~"$job",cluster=~"$cluster", mode='system'})
	sum(node_load5{job=~"$job",cluster=~"$cluster"})
	avg(1 - avg(irate(node_cpu_seconds_total{job=~"$job",mode="idle",cluster=~"$cluster"}[5m])) by (i
Overall total memory and average memory utilization	sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"})
	sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"} - node_memory_MemAvailable_bytes{job=~"$job",cluster=~"$cluster"})
	(sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"} - node_memory_MemAvailable_bytes{job=~"$job",cluster=~"$cluster"}) / sum(node_memory_MemTotal_bytes{job=~"$job",cluster=~"$cluster"}))*100
Overall total disk and average disk utilization	sum(avg(node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by(device
	sum(avg(node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by(device sum(avg(node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by(device

	(sum(avg(node_filesystem_size_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by(devic sum(avg(node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by(device *100/(sum(avg(node_filesystem_avail_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by (sum(avg(node_filesystem_size_bytes{job=~"$job",fstype=~"xfs\|ext.*"})by(device,instance)) - sum(avg(node_filesystem_free_bytes{job=~"$job",cluster=~"$cluster",fstype=~"xfs|ext.*"})by(device
Running time	avg(time() - node_boot_time_seconds{instance=~"$node",cluster=~"$cluster"}) 75
CPU cores	count(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node", mode='system'})
Total memory	sum(node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"})
Total CPU utilization	100 - (avg(irate(node_cpu_seconds_total{instance=~"$node",mode="idle",cluster=~"$cluster"}[5m]
Memory utilization	(1 - (node_memory_MemAvailable_bytes{instance=~"$node",cluster=~"$cluster"} / (node_memory_MemTotal_bytes{instance=~"$node",cluster=~"$cluster"})))* 100
Maximum partition utilization	(node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint= node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint=' /(node_filesystem_avail_bytes {cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoir (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint= node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint='
CPU iowait	avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="iowait"}[5m])) * 1
Available space for each partition	node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint !
	node_filesystem_avail_bytes {cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint
	(node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint ! *100/(node_filesystem_avail_bytes {cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",moun

	(node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint !=
CPU utilization	avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="system"}[5m])) b
	avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="user"}[5m])) by (
	avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="iowait"}[5m])) by
	(1 - avg(irate(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node",mode="idle"}[5m])) b
Memory information	node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"}
	node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"} - node_memory_MemAvailable_bytes{cluster=~"$cluster",instance=~"$node"}
	node_memory_MemAvailable_bytes{cluster=~"$cluster",instance=~"$node"}
	(1 - (node_memory_MemAvailable_bytes{cluster=~"$cluster",instance=~"$node"} / (node_memory_MemTotal_bytes{cluster=~"$cluster",instance=~"$node"}))) * 100
Network bandwidth usage per second	irate(node_network_receive_bytes_total{cluster=~"$cluster",instance=~'$node',device=~"$device"}
	irate(node_network_transmit_bytes_total{cluster=~"$cluster",instance=~'$node',device=~"$device"
System average load	node_load1{cluster=~"$cluster",instance=~"$node"}
	node_load5{cluster=~"$cluster",instance=~"$node"}
	node_load15{cluster=~"$cluster",instance=~"$node"}

	sum(count(node_cpu_seconds_total{cluster=~"$cluster",instance=~"$node", mode='system'}) by (by(instance)
Disk read/write capacity per second	irate(node_disk_read_bytes_total{cluster=~"$cluster",instance=~"$node"}[5m])
	irate(node_disk_written_bytes_total{cluster=~"$cluster",instance=~"$node"}[5m])
Disk utilization	(node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint ! *100/(node_filesystem_avail_bytes {cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mour (node_filesystem_size_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint node_filesystem_free_bytes{cluster=~"$cluster",instance=~'$node',fstype=~"ext.*|xfs",mountpoint !
	node_filesystem_files_free{cluster=~"$cluster",instance=~'$node',fstype=~"ext.?|xfs"} / node_filesystem_files{cluster=~"$cluster",instance=~'$node',fstype=~"ext.?|xfs"}
Disk read/write rate (IOPS)	irate(node_disk_reads_completed_total{cluster=~"$cluster",instance=~"$node"}[5m])
	irate(node_disk_writes_completed_total{cluster=~"$cluster",instance=~"$node"}[5m])
	node_disk_io_now{cluster=~"$cluster",instance=~"$node"}
I/O operation time percentage per second	irate(node_disk_io_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m])
Time taken for each IO read/write	irate(node_disk_read_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) / irate(node_disk_reads_completed_total{instance=~"$node"}[5m])
	irate(node_disk_write_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m]) / irate(node_disk_writes_completed_total{cluster=~"$cluster",instance=~"$node"}[5m])

	irate(node_disk_io_time_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m])
	irate(node_disk_io_time_weighted_seconds_total{cluster=~"$cluster",instance=~"$node"}[5m])
Network socket connection information	node_netstat_Tcp_CurrEstab{cluster=~"$cluster",instance=~'$node'}
	node_sockstat_TCP_tw{cluster=~"$cluster",instance=~'$node'}
	node_sockstat_sockets_used{cluster=~"$cluster",instance=~'$node'}
	node_sockstat_UDP_inuse{cluster=~"$cluster",instance=~'$node'}
	node_sockstat_TCP_alloc{cluster=~"$cluster",instance=~'$node'}
	irate(node_netstat_Tcp_PassiveOpens{cluster=~"$cluster",instance=~'$node'}[5m])
	irate(node_netstat_Tcp_ActiveOpens{cluster=~"$cluster",instance=~'$node'}[5m])
	irate(node_netstat_Tcp_InSegs{cluster=~"$cluster",instance=~'$node'}[5m])
	irate(node_netstat_Tcp_OutSegs{cluster=~"$cluster",instance=~'$node'}[5m])
	irate(node_netstat_Tcp_RetransSegs{cluster=~"$cluster",instance=~'$node'}[5m])
Open file descriptors (left)/context switches per second (right)	node_filefd_allocated{cluster=~"$cluster",instance=~"$node"}
	irate(node_context_switches_total{cluster=~"$cluster",instance=~"$node"}[5m])
	(node_filefd_allocated{cluster=~"$cluster",instance=~"$node"}/node_filefd_maximum{cluster=~"$cl *100

Node Pod Monitoring

Chart Name	Query Statement
Pods	count(kube_pod_info{node=~"$node"})
Pod Request Memory	sum(kube_pod_container_resource_requests_memory_bytes{node=~"$node"})by(node)
Pod Request CPU Cores	sum(kube_pod_container_resource_requests_cpu_cores{node=~"$node"})by(node)
CPU Usage	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluste node=~"$node", container!="POD", container!=""}) by (pod)
CPU Quota	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluste node=~"$node", container!="POD", container!=""}) by (pod)
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluste node=~"$node", container!="POD", container!=""}) by (pod) / sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod)
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluste node=~"$node", container!="POD", container!=""}) by (pod) / sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", node=~"$node"}) by (pod)

Memory Usage	sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node", container!="", container!="POD"}) by (pod)
Memory Quota	sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node",container!="", container!="POD"}) by (pod)
	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", node=~"$node"}) by (pod)
	sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node",container!="", container!="POD"}) by (pod) / sum(kube_pod_container_resource_requests_memory_bytes{node=~"$node"}) by (pod)
	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", node=~"$node"}) by (poc
	sum(node_namespace_pod_container:container_memory_working_set_bytes{cluster="$cluster", node=~"$node",container!="", container!="POD"}) by (pod) / sum(kube_pod_container_resource_limits_memory_bytes{node=~"$node"}) by (pod)
Pod List	group (kube_pod_info{host_ip="$node"})by(created_by_kind, created_by_name,host_network,pod_ip,pod,priority_class,namespace)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(kube_pod_status_phase{}==1) by (pod, phase)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() sum(container_memory_working_set_bytes) by (pod)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() sum(rate(container_cpu_usage_seconds_total{image!=""}[5m])) by (pod)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(time()-kube_pod_start_time by (pod)

	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) max(kube_pod_status_ready{condition="true" by (pod) or on() vector(0)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_network_receive_bytes_total{image!=""}[5m])) by (pod) or on() vector(0)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_network_transmit_bytes_total{image!=""}[5m])) by (pod) or on() vector(0)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_fs_reads_bytes_total{container!="POD", container!=""}[5m])) by (pod) or on() vector(0)
	min(kube_pod_info{host_ip="$node"})by(pod) * on(pod) group_right() max(rate(container_fs_writes_bytes_total{container!="POD", container!=""}[5m])) by (pod) or on() vector(0)

Workload Monitoring Overview

Chart Name	Query Statement
CPU Usage	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="requests.cpu"})

	scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="limits.cpu"})
CPU Quota	count(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type) /sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",

	workload_type="$type"}) by (workload, workload_type)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type) /sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
Memory Usage	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="requests.memory"})
	scalar(kube_resourcequota{cluster="$cluster", namespace="$namespace", type="hard",resource="limits.memory"})
Memory Quota	count(namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}

) by (workload, workload_type)
sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type) /sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)
sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}

	) by (workload, workload_type) /sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload_type="$type"}) by (workload, workload_type)

Deployment

Chart Name	Query Statement
Age	time() - max(kube_deployment_created{cluster="$cluster",namespace="$namespace",deplo
Replicas(Pods)-Request	max(kube_deployment_spec_replicas{deployment="$workload",cluster="$cluster",namespa
Replicas(Pods)-Ready	max(kube_deployment_status_replicas_ready{deployment="$workload",cluster="$cluster",r
Replica Trend	max(kube_deployment_spec_replicas{deployment="$workload",cluster="$cluster",namespa
	max(kube_deployment_status_replicas{deployment="$workload",cluster="$cluster",namesp
	min(kube_deployment_status_replicas_ready{deployment="$workload",cluster="$cluster",na pod)
	min(kube_deployment_status_replicas_available{deployment="$workload",cluster="$cluste (instance, pod)
	min(kube_deployment_status_replicas_updated{deployment="$workload",cluster="$cluster" (instance, pod)
	min(kube_deployment_status_replicas_unavailable{deployment="$workload",cluster="$clus (instance, pod)
CPU Usage	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster= container!="POD", container!=""}

	* on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
CPU Quota	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster= container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$na * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster= container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod) /sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$na * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$name * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster= container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe

	namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod) /sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$name * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
CPU Limit-Total	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(kube_pod_container_resource_limits_cpu_cores{resource="cp cluster="$cluster",namespace="$namespace"}) by (pod))
CPU Request-Total	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(kube_pod_container_resource_requests_cpu_cores{resource= cluster="$cluster",namespace="$namespace"}) by (pod))
CPU Info	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset)

	* on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace (pod, container)
	max(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$ by (pod, container))by(container)
	max(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster",namespace="$nam (pod, container))by(container)
CPU Usage/Limit (%)	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace (pod, container) / max by(container, pod) (kube_pod_container_resource_limits_cpu_cores{ cluster="$cluster",namespace="$namespace"})

CPU Usage/Request(%)	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace (pod, container) / max by(container, pod) (kube_pod_container_resource_requests_cpu_co cluster="$cluster",namespace="$namespace"})
CPU User Time(%)	avg(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() (max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace' (pod,container) / max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace" [5m])+rate(container_cpu_system_seconds_total{cluster="$cluster",namespace="$namespa (pod,container))) by (pod,container)
Memory Usage	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)
Memory Quota	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="deployment"}) by (pod)

	sum(
	  kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace=
	* on(namespace,pod)
	  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
	namespace="$namespace", workload="$workload", workload_type="deployment"}
	) by (pod)

	sum(
	  container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co
	* on(namespace,pod)
	  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
	namespace="$namespace", workload="$workload", workload_type="deployment"}
	) by (pod)
	/sum(
	  kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace=
	* on(namespace,pod)
	  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
	namespace="$namespace", workload="$workload", workload_type="deployment"}
	) by (pod)

	sum(
	  kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$na
	* on(namespace,pod)
	  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
	namespace="$namespace", workload="$workload", workload_type="deployment"}
	) by (pod)

	sum(
	  container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co
	* on(namespace,pod)
	  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
	namespace="$namespace", workload="$workload", workload_type="deployment"}
	) by (pod)
	/sum(
	  kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$na
	* on(namespace,pod)
	  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
	namespace="$namespace", workload="$workload", workload_type="deployment"}
	) by (pod)

Memory Limit-Total	sum(label_replace(
	  max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl
	uid, pod, pod_ip, node),
	  "replicaset",
	  "$1",
	  "created_by_name",

	"(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(container_spec_memory_limit_bytes{cluster="$cluster",namesp
Memory Request-Total	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(kube_pod_container_resource_requests_memory_bytes{resou cluster="$cluster",namespace="$namespace"}) by (pod))
Memory Info	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{clus container!="", image!="", container!="POD"})
	max(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max by(container, pod) (kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$
	max(label_replace(

	max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max by(container, pod) (kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespace="$nam
Memory Usage/Limit(%)	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{clus container!="", image!="", container!="POD"})/max by(container, pod) (kube_pod_container_resource_limits_memory_bytes{resource="memory", cluster="$cluste
Memory Usage/Request(%)	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{clus container!="", image!="", container!="POD"})/max by(container, pod) (kube_pod_container_resource_requests_memory_bytes{resource="memory", cluster="$clu
Sockets	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name",

	"(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(container_sockets{cluster="$cluster",namespace="$namespac
Network In	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(rate(container_network_receive_bytes_total{cluster="$cluster",
Network Out	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() sum(rate(container_network_transmit_bytes_total{cluster="$cluster"
Network Errors	sum(label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() (sum(container_network_receive_errors_total{cluster="$cluster",nan sum(container_network_transmit_errors_total{cluster="$cluster",namespace="$namespace'
Network IO	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node),

	"replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(rate(container_network_receive_bytes_total{cluster="$cluster",
	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(rate(container_network_transmit_bytes_total{cluster="$cluster"
File System Read	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(rate(container_fs_reads_bytes_total{cluster="$cluster",namesp container!=""}[5m])) by (pod,container)
File System Write	label_replace(max(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Repl uid, pod, pod_ip, node), "replicaset", "$1", "created_by_name", "(.+)") * on(replicaset) group_left() max(kube_replicaset_owner{cluster="$cluster",namespace="$namespace",owner_kind="De (replicaset) * on(pod) group_right() max(rate(container_fs_writes_bytes_total{cluster="$cluster",namesp container!=""}[5m])) by (pod,container)

StatefulSet

Chart Name	Query Statement
Generation	max(kube_statefulset_metadata_generation{cluster="$cluster",namespace="$namespace",
Replicas(Pods)-Request	max(kube_statefulset_replicas{statefulset="$workload",cluster="$cluster",namespace="$na
Replicas(Pods)-Ready	max(kube_statefulset_status_replicas_ready{statefulset="$workload",cluster="$cluster",nan
Age	time() - max(kube_statefulset_created{cluster="$cluster",namespace="$namespace",statefu
Replica Trend	max(kube_statefulset_replicas{statefulset="$workload",cluster="$cluster",namespace="$na
	max(kube_statefulset_status_replicas{statefulset="$workload",cluster="$cluster",namespac pod)
	min(kube_statefulset_status_replicas_ready{statefulset="$workload",cluster="$cluster",nam (instance, pod)
	min(kube_statefulset_status_replicas_available{statefulset="$workload",cluster="$cluster",r (instance, pod)
	min(kube_statefulset_status_replicas_updated{statefulset="$workload",cluster="$cluster",n (instance, pod)
CPU Usage	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster= namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
CPU Quota	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster= namespace="$namespace", container!="POD", container!=""} * on(namespace,pod)

```
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
```

```
sum(
  kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$na
 * on(namespace,pod)
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
```

```
sum(
  node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster=
namespace="$namespace", container!="POD", container!=""}
 * on(namespace,pod)
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
/sum(
  kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$na
 * on(namespace,pod)
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
```

```
sum(
  kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$name
 * on(namespace,pod)
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
```

```
sum(
  node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster=
namespace="$namespace", container!="POD", container!=""}
 * on(namespace,pod)
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
/sum(
  kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$name
 * on(namespace,pod)
  group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe
namespace="$namespace", workload="$workload", workload_type="statefulset"}
) by (pod)
```

CPU Limit-Total	sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind= created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(kube_pod_container_resource_limits_cpu_cores{resource="cp cluster="$cluster",namespace="$namespace"}) by (pod))
CPU Request-Total	sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind= created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(kube_pod_container_resource_requests_cpu_cores{resource= cluster="$cluster",namespace="$namespace"}) by (pod))
CPU Info	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace [5m])) by (pod, container,image)
	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat created_by_name="$workload"}) by (pod) * on(pod) group_right() max(kube_pod_container_resource_limits_cpu_cores{cluster="$clus (pod, container,image)
	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat created_by_name="$workload"}) by (pod) * on(pod) group_right() max(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",namespace="$ container,image)
CPU Usage/Limit (%)	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace [5m])) by (pod, container) / max by(container, pod) (kube_pod_container_resource_limits_cp cluster="$cluster",namespace="$namespace"})
CPU Usage/Request(%)	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_cpu_usage_seconds_total{cluster="$cluster",namespace="$namespace [5m])) by (pod, container) / max by(container, pod) (kube_pod_container_resource_requests cluster="$cluster",namespace="$namespace"})
CPU User Time(%)	avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind=' created_by_name="$workload"}) by (pod)

	* on(pod) group_right() (max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace' [5m])) by (pod, container,image) / max(rate(container_cpu_user_seconds_total{cluster="$cluster",namespace="$namespace" [5m])+rate(container_cpu_system_seconds_total{cluster="$cluster",namespace="$namespa [5m])) by (pod,container,image))) by (pod,container,image)
Memory Usage	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
Memory Quota	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace= * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) /sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace= * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$na * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe

	namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", co * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod) /sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$na * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabe namespace="$namespace", workload="$workload", workload_type="statefulset"}) by (pod)
Memory Limit-Total	sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind= created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(container_spec_memory_limit_bytes{cluster="$cluster",namespace="$namespace",co (pod))
Memory Request-Total	sum(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind= created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(kube_pod_container_resource_requests_memory_bytes{resou cluster="$cluster",namespace="$namespace"}) by (pod))
Memory Info	avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind=' created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod, image) (container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", cont container!="POD"}))by (container, pod, image)
	max(avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_k created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod, image) (kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",namespace="$ pod))by(container)
	max(avg(group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_k created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod) (kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",namespace="$nam pod))by(container)
Memory	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat

Usage/Limit(%)	created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", cont container!="POD"})/max by(container, pod) (kube_pod_container_resource_limits_memory_ cluster="$cluster",namespace="$namespace"})
Memory Usage/Request(%)	group(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Stat created_by_name="$workload"}) by (pod) * on(pod) group_right() max by(container, pod) (container_memory_working_set_bytes{cluster="$cluster",namespace="$namespace", cont container!="POD"})/max by(container, pod) (kube_pod_container_resource_requests_mem cluster="$cluster",namespace="$namespace"})
Sockets	sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="$ created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(container_sockets{cluster="$cluster",namespace="$namespac
Network In	sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="$ created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(rate(container_network_receive_bytes_total{cluster="$cluster", by (pod))
Network Out	sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="$ created_by_name="$workload"}) by (pod) * on(pod) group_right() sum(rate(container_network_transmit_bytes_total{cluster="$cluster" by (pod))
Network Errors	sum(sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="$ created_by_name="$workload"}) by (pod) * on(pod) group_right() (sum(container_network_receive_errors_total{cluster="$cluster",nan sum(container_network_transmit_errors_total{cluster="$cluster",namespace="$namespace"
Network IO	sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="Statef created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_network_receive_bytes_total{cluster="$cluster", by (pod)
	-sum(kube_pod_info{cluster="$cluster",namespace="$namespace",created_by_kind="State created_by_name="$workload"}) by (pod) * on(pod) group_right() max(rate(container_network_transmit_bytes_total{cluster="$cluster" by (pod)

DaemonSet

Chart Name	Query Statement
CPU Usage	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
CPU Quota	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod) /sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",

	namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cluster", namespace="$namespace", container!="POD", container!=""} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod) /sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
Memory Usage	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)
Memory Quota	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", container!="", container!="POD"} * on(namespace,pod) group_left(workload, workload_type)

```
namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",
workload="$workload", workload_type="daemonset"}
) by (pod)
```

```
sum(
kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",
namespace="$namespace"}
  * on(namespace,pod)
    group_left(workload, workload_type)
namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",
workload="$workload", workload_type="daemonset"}
) by (pod)
```

```
sum(
    container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace",
container!="", container!="POD"}
  * on(namespace,pod)
    group_left(workload, workload_type)
namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",
workload="$workload", workload_type="daemonset"}
) by (pod)
/sum(

kube_pod_container_resource_requests_memory_bytes{cluster="$cluster",
namespace="$namespace"}
  * on(namespace,pod)
    group_left(workload, workload_type)
namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",
workload="$workload", workload_type="daemonset"}
) by (pod)
```

```
sum(
kube_pod_container_resource_limits_memory_bytes{cluster="$cluster",
namespace="$namespace"}
  * on(namespace,pod)
    group_left(workload, workload_type)
namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",
workload="$workload", workload_type="daemonset"}
) by (pod)
```

```
sum(
    container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace",
container!="", container!="POD"}
  * on(namespace,pod)
    group_left(workload, workload_type)
namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace",
```

	workload="$workload", workload_type="daemonset"}) by (pod) /sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$namespace"} * on(namespace,pod) group_left(workload, workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster="$cluster", namespace="$namespace", workload="$workload", workload_type="daemonset"}) by (pod)

Cluster Pod Monitoring

Chart Name	Query Statement
Age	time() - max(kube_pod_created{pod=~"$pod",cluster="$cluster",namespace="$namespace"})
Restart Count-Last 1 Hour	ceil(sum (increase(kube_pod_container_status_restarts_total{pod=~"$pod",cluster="$cluster",nam
Requests-CPU	sum(kube_pod_container_resource_requests_cpu_cores{pod=~"$pod"}) or vector(0)
Requests-Memory	sum(kube_pod_container_resource_requests_memory_bytes{pod=~"$pod"}) or vector(0)
Limits-CPU	sum(kube_pod_container_resource_limits_cpu_cores{pod=~"$pod"}) or vector(0)
Limits-Memory	sum(kube_pod_container_resource_limits_memory_bytes{pod=~"$pod"}) or vector(0)
Containers	group by (image, container,pod) (kube_pod_container_info{cluster="$cluster",namespace="$name
	sum by (container,pod)(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster",nan
	sum by (container,pod)(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster" pod=~"$pod"})

	max by (container,pod)(kube_pod_container_status_running{cluster="$cluster",namespace="$nam
	sum by (container,pod)(kube_pod_container_resource_limits{resource="cpu",cluster="$cluster",na
	sum by (container,pod)(kube_pod_container_resource_limits{resource="memory",cluster="$cluster pod=~"$pod"})
	max by (container,pod)(kube_pod_container_status_restarts_total{cluster="$cluster",namespace='
CPU Usage (%)	max(irate(container_cpu_usage_seconds_total{pod=~"$pod",container!="",container!="POD",cluste [1m])) by (container,namespace,pod) / max(container_spec_cpu_quota{pod=~"$pod",container!="",container!="POD",cluster="$cluster",na (container,namespace,pod) or on() vector(0)
CPU Usage By Cores	max(irate(container_cpu_usage_seconds_total{pod=~"$pod",container!="",container!="POD",cluste [1m])) by (pod,container,namespace)or on() vector(0)
CPU Load (10s)	max(container_cpu_load_average_10s{namespace=~"$namespace", pod=~"$pod", container!="", 1000)by(pod,container)
CPU Throttled Percent	max (rate (container_cpu_cfs_throttled_seconds_total{image!="", container!="", cluster="$cluster",r pod=~"$pod"}[1m])) by (container,pod) / max (rate (container_cpu_cfs_periods_total{image!="", cor cluster="$cluster",namespace=~"$namespace", pod=~"$pod"}[1m])) by (container,pod) or on() vect
CPU Quota	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cl pod="$pod", container!="POD", container!=""}) by (container)
	sum(kube_pod_container_resource_requests_cpu_cores{cluster="$cluster", namespace="$names
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cl pod="$pod", container!="POD", container!=""}) by (container) / sum(kube_pod_container_resource_ namespace="$namespace", pod="$pod"}) by (container)
	sum(kube_pod_container_resource_limits_cpu_cores{cluster="$cluster", namespace="$namespac
	sum(node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate{cluster="$cl pod="$pod", container!="POD", container!=""}) by (container) / sum(kube_pod_container_resource_ namespace="$namespace", pod="$pod"}) by (container)

Memory Usage (WSS)	max(container_memory_working_set_bytes{pod=~"$pod",container !="",container!="POD",cluster= by (pod,namespace,container)
Memory Usage	max(container_memory_usage_bytes{pod=~"$pod",container !="",container!="POD",cluster="$clus (pod,namespace,container)
Memory Usage (RSS)	max(container_memory_rss{pod=~"$pod",container !="",container!="POD",cluster="$cluster",name (pod,namespace,container) or on() vector(0)
Memory Cache	max(container_memory_cache{pod=~"$pod",container !="",container!="POD",cluster="$cluster",na (pod,namespace,container)
Usage WSS/Limit (%)	(max(container_memory_working_set_bytes{pod=~"$pod",container !="",container!="POD",cluster= by (pod,namespace,container)/ max(container_spec_memory_limit_bytes{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,namespace, contai
Usage/Limit (%)	(max(container_memory_usage_bytes{pod=~"$pod",container !="",container!="POD",cluster="$clu (pod,namespace,container)/ max(container_spec_memory_limit_bytes{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,namespace, contai
Usage RSS/Limit (%)	(max(container_memory_rss{pod=~"$pod",container !="",container!="POD",cluster="$cluster",nam (pod,namespace,container)/ sum(container_spec_memory_limit_bytes{pod=~"$pod",container !="",container!="POD",cluster="$cluster",namespace=~"$namespace"}) by (pod,namespace, contai
Memory Failcnt	max (increase(container_memory_failcnt{cluster="$cluster",namespace=~"$namespace", pod=~"$p (pod,container)
Memory Quota	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", pod="$ by (container)
	sum(kube_pod_container_resource_requests_memory_bytes{cluster="$cluster", namespace="$na
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", pod="$ by (container) / sum(kube_pod_container_resource_requests_memory_bytes{namespace="$name
	sum(kube_pod_container_resource_limits_memory_bytes{cluster="$cluster", namespace="$name (container)
	sum(container_memory_working_set_bytes{cluster="$cluster", namespace="$namespace", pod="$ by (container) / sum(kube_pod_container_resource_limits_memory_bytes{namespace="$namespa

Network Input	max (rate (container_network_receive_bytes_total{image!="",cluster="$cluster",namespace=~"$nar by(pod)
Network Output	max (rate (container_network_transmit_bytes_total{image!="",cluster="$cluster",namespace=~"$na [1m]))by(pod)
Network Input Error (%)	max (increase (container_network_receive_packets_dropped_total{id!="/", cluster="$cluster",name [1m])) by (pod,interface) / max (increase (container_network_receive_packets_total{id!="/", cluster= pod=~"$pod"}[1m])) by (pod,interface)
	max (increase (container_network_receive_errors_total{id!="/", cluster="$cluster",namespace=~"$r (pod,interface) / max (increase (container_network_receive_packets_total{id!="/", cluster="$cluster pod=~"$pod"}[1m])) by (pod,interface)
Network Output Error (%)	max (increase (container_network_transmit_packets_dropped_total{id!="/", cluster="$cluster",nam [1m])) by (pod,interface) / max (increase (container_network_transmit_packets_total{id!="/", cluster pod=~"$pod"}[1m])) by (pod,interface)
	max (increase (container_network_transmit_errors_total{id!="/", cluster="$cluster",namespace=~"$ (pod,interface) / max (increase (container_network_receive_packets_total{id!="/", cluster="$cluster pod=~"$pod"}[1m])) by (pod,interface)
File System Read	max (rate(container_fs_reads_bytes_total{cluster="$cluster",namespace=~"$namespace", pod=~"$ (container,pod)
File System Write	max (rate(container_fs_writes_bytes_total{cluster="$cluster",namespace=~"$namespace", pod=~"$ (container,pod)
Network Socket	max(container_sockets{cluster="$cluster",namespace=~"$namespace", pod=~"$pod", container!="
Process Number	count(container_processes{cluster="$cluster",namespace=~"$namespace", pod=~"$pod", containe

Cluster Network Monitoring

Chart Name	Query Statement
Current Rate of	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"} (namespace))

Bytes Received	
Current Rate of Bytes Transmitted	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+" (namespace))
Current Status	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"} (namespace))
	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+" (namespace))
	sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[(namespace))
	sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"} (namespace))
	sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~". (namespace))
	sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~" (namespace))
	sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~" (namespace))
	sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",name [5m])) by (namespace))
Average Rate of Bytes Received	sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"}[(namespace))
Average Rate of Bytes Transmitted	sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+"} (namespace))
Receive Bandwidth	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~".+"} (namespace))
Transmit Bandwidth	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~".+" (namespace))

Rate of Received Packets	sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~". (namespace))
Rate of Transmitted Packets	sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~" (namespace))
Rate of Received Packets Dropped	sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",names [5m])) by (namespace))
Rate of Transmitted Packets Dropped	sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",name [5m])) by (namespace))
Rate of TCP Retransmits out of all sent segments	sort_desc(sum(rate(node_netstat_Tcp_RetransSegs{cluster=~"$cluster"}[5m]) / rate(node_netstat_Tcp_OutSegs{cluster=~"$cluster"}[$interval:$resolution])) by (instance))
Rate of TCP SYN Retransmits out of all retransmits	sort_desc(sum(rate(node_netstat_TcpExt_TCPSynRetrans{cluster=~"$cluster"}[$interval:$resolutio rate(node_netstat_Tcp_RetransSegs{cluster=~"$cluster"}[$interval:$resolution])) by (instance))

Namespace Pods Network Monitoring

Chart Name	Query Statement
Current Rate of Bytes Received	sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"}
Current Rate of	sum(irate(container_network_transmit_bytes_total{namespace=~"$namespace"}[5m]))

Bytes Transmitted	
Current Status	sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"} (pod)
	sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace' (pod)
	sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespac (pod)
	sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespa (pod)
	sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$r [5m])) by (pod)
	sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$ [5m])) by (pod)
Receive Bandwidth	sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace"} (pod)
Transmit Bandwidth	sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace' (pod)
Rate of Received Packets	sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespac (pod)
Rate of Transmitted Packets	sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespa (pod)
Rate of Received Packets Dropped	sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$r [5m])) by (pod)
Rate of Transmitted Packets Dropped	sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster", namespace=~"$namespace"}[5m])) by (pod)

Namespace Workload Network Monitoring

Chart Name	Query Statement
Current Rate of Bytes Received	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Current Rate of Bytes Transmitted	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$n * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Current Status	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$n * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$ * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"

	* on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",names [5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
	sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",name [5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Average Rate of Bytes Received	sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Average Rate of Bytes Transmitted	sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Receive Bandwidth	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Transmit Bandwidth	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$n * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Rate of Received Packets	sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$ * on (namespace,pod) group_left(workload,workload_type)

	namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Rate of Transmitted Packets	sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~" * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Rate of Received Packets Dropped	sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",names [5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))
Rate of Transmitted Packets Dropped	sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",name [5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload_type="$type"}) by (workload))

Pod Network Monitoring

Chart Name	Query Statement
Current Rate of Bytes Received	sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace", pod=~"$pod"}[5m]))
Current Rate of Bytes Transmitted	sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace" pod=~"$pod"}[5m]))
Receive Bandwidth	sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$namespace", pod=~"$pod"}[5m])) by (pod)
Transmit Bandwidth	sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$namespace" pod=~"$pod"}[5m])) by (pod)
Rate of	sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$namespac

Received Packets	pod=~"$pod"}[5m])) by (pod)
Rate of Transmitted Packets	sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~"$namespa pod=~"$pod"}[5m])) by (pod)
Rate of Received Packets Dropped	sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",namespace=~"$r pod=~"$pod"}[5m])) by (pod)
Rate of Transmitted Packets Dropped	sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",namespace=~"$ pod=~"$pod"}[5m])) by (pod)

Workload Network Monitoring

Chart Name	Query Statement
Current Rate of Bytes Received	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Current Rate of Bytes Transmitted	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$n * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Average Rate of Bytes Received	sort_desc(avg(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Average Rate of Bytes Transmitted	sort_desc(avg(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type)

	namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Receive Bandwidth	sort_desc(sum(irate(container_network_receive_bytes_total{cluster=~"$cluster",namespace=~"$na * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Transmit Bandwidth	sort_desc(sum(irate(container_network_transmit_bytes_total{cluster=~"$cluster",namespace=~"$n * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Rate of Received Packets	sort_desc(sum(irate(container_network_receive_packets_total{cluster=~"$cluster",namespace=~"$ * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Rate of Transmitted Packets	sort_desc(sum(irate(container_network_transmit_packets_total{cluster=~"$cluster",namespace=~" * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Rate of Received Packets Dropped	sort_desc(sum(irate(container_network_receive_packets_dropped_total{cluster=~"$cluster",names [5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))
Rate of Transmitted Packets Dropped	sort_desc(sum(irate(container_network_transmit_packets_dropped_total{cluster=~"$cluster",name [5m]) * on (namespace,pod) group_left(workload,workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=~"$cluster",namespace=~"$namespac workload=~"$workload", workload_type="$type"}) by (pod))

PVC Storage Monitoring

Chart	Query Statement	Metrics Used

Name		
Volume Space Usage	(sum without(instance, node) (kubelet_volume_stats_capacity_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}) - sum without(instance, node) (kubelet_volume_stats_available_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}))	kubelet_volume_stats_capacity_bytes kubelet_volume_stats_available_bytes
	sum without(instance, node) (kubelet_volume_stats_available_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"})	kubelet_volume_stats_available_bytes
Volume Space Usage	(kubelet_volume_stats_capacity_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"} - kubelet_volume_stats_available_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}) / kubelet_volume_stats_capacity_bytes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"} * 100	kubelet_volume_stats_capacity_bytes kubelet_volume_stats_available_bytes kubelet_volume_stats_capacity_bytes
Volume inodes Usage	sum without(instance, node) (kubelet_volume_stats_inodes_used{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"})	kubelet_volume_stats_inodes_used
	(sum without(instance, node) (kubelet_volume_stats_inodes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}) -	kubelet_volume_stats_inodes

	sum without(instance, node) (kubelet_volume_stats_inodes_used{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"}))	kubelet_volume_stats_inodes_used
Volume inodes Usage	kubelet_volume_stats_inodes_used{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"} / kubelet_volume_stats_inodes{cluster="$cluster", job="kubelet", namespace="$namespace", persistentvolumeclaim="$volume"} * 100	kubelet_volume_stats_inodes_used kubelet_volume_stats_inodes

Data Collection Configuration

Last updated：2024-08-07 22:07:45

Overview

This document describes how to configure monitoring collection items for the associated cluster.

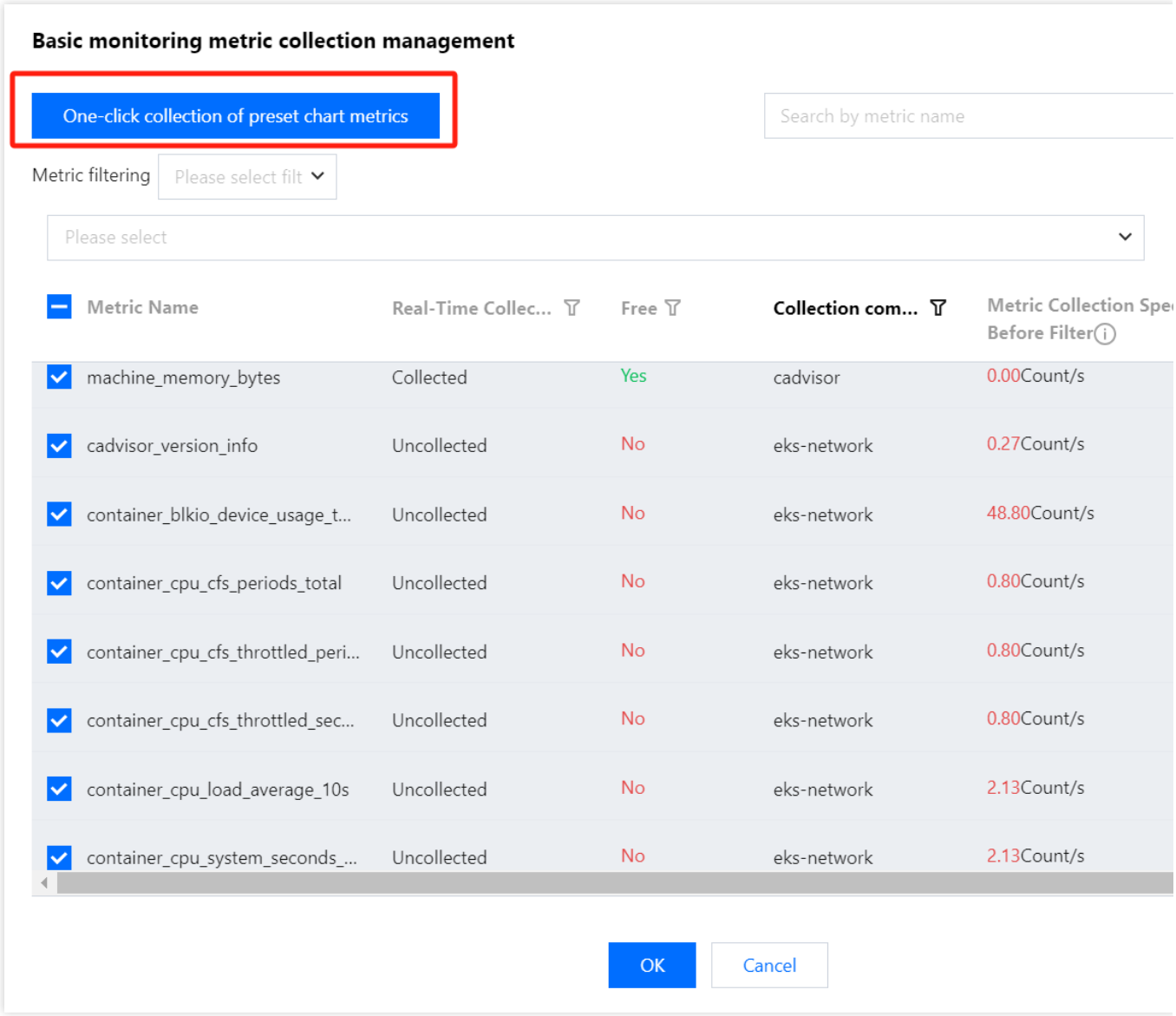
Prerequisites

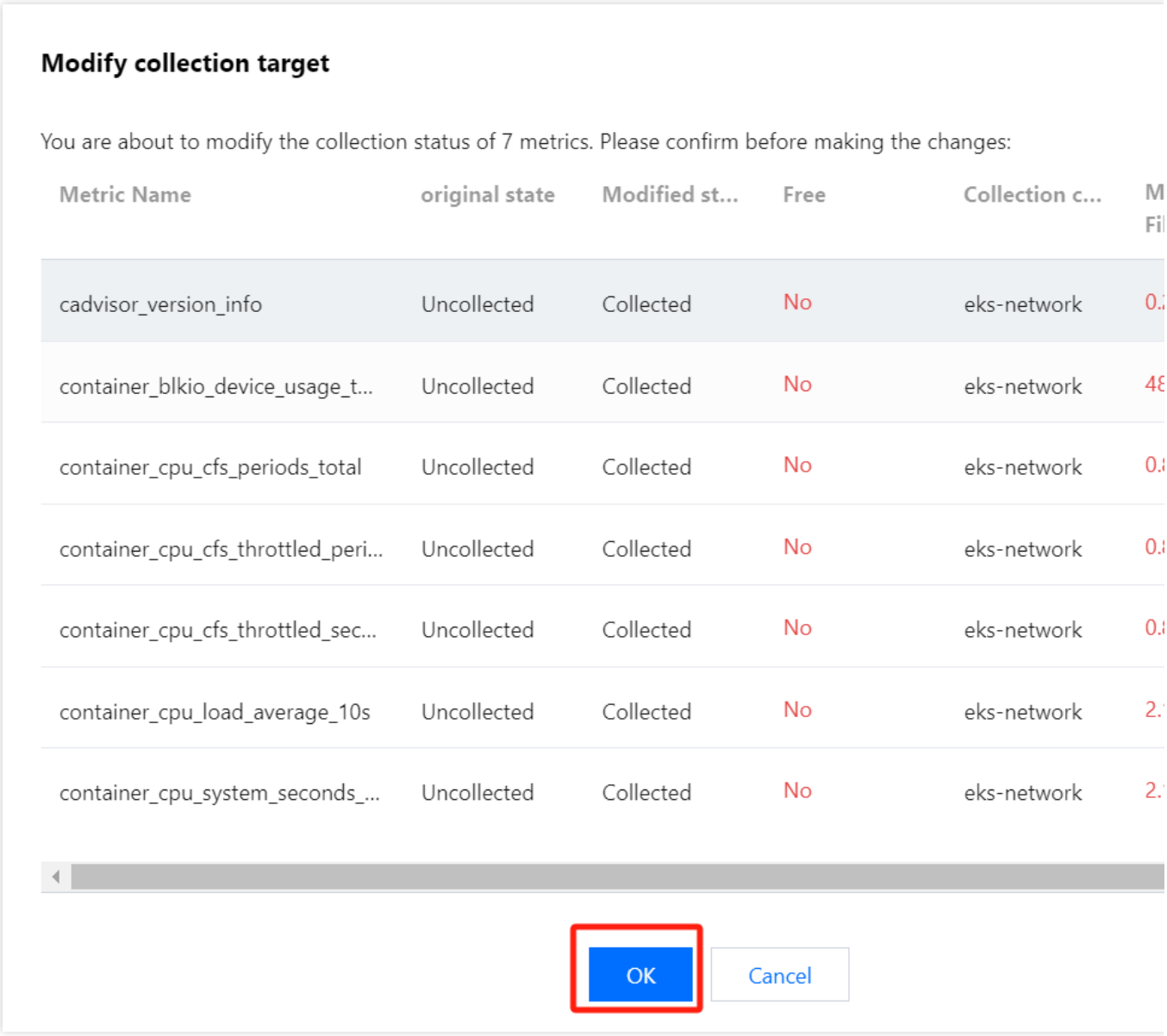
Before configuring monitoring collection items, you need to perform the following operations:
The Prometheus monitoring instance has been successfully created.
The cluster to be monitored has been associated with the corresponding instance.

Directions

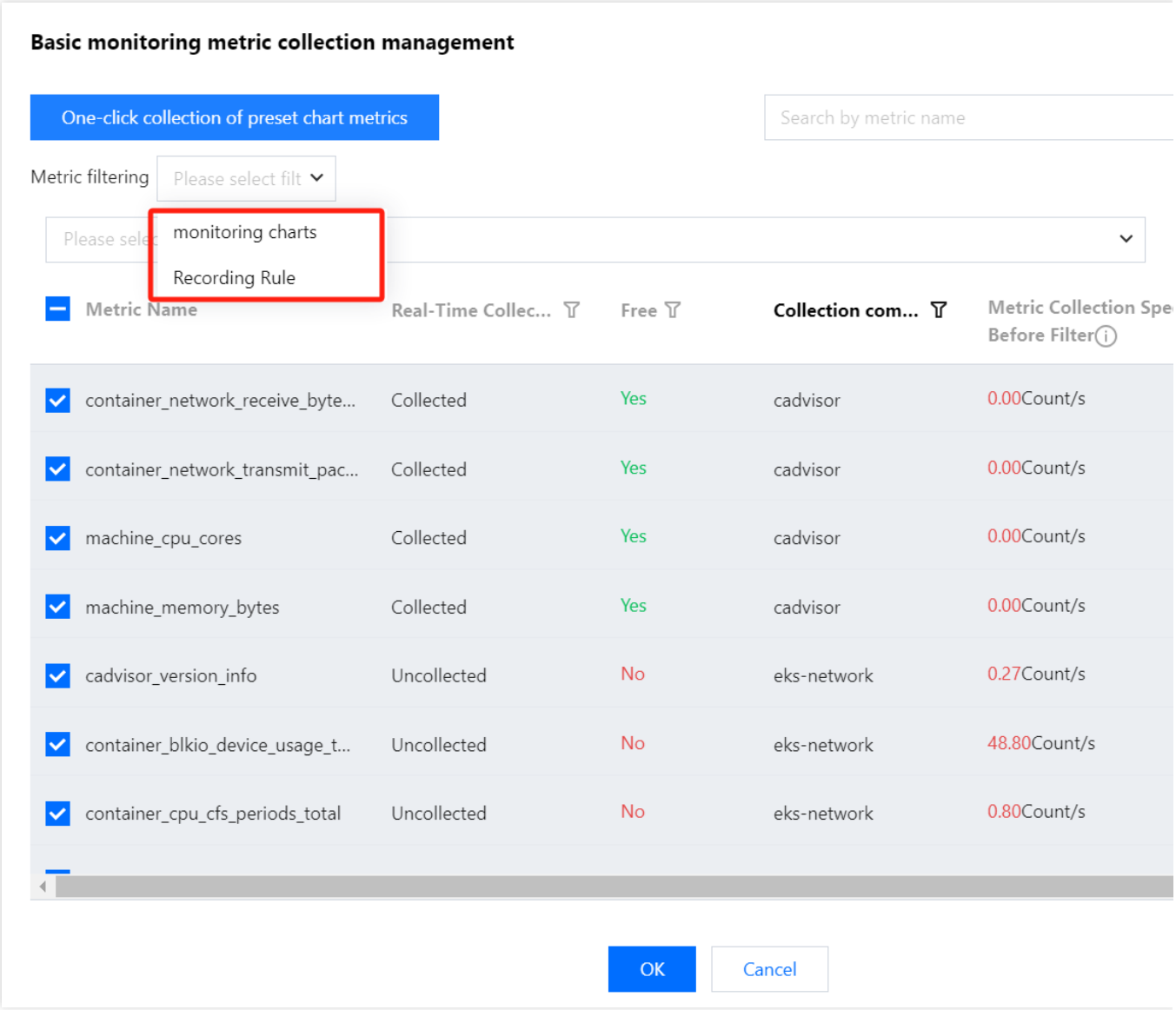
Configuring Data Collection

1. Log in to TMP Console.
2. On the instance list page, select the name of the instance that requires configuring data collection rules and enter its details page.
3. On the **Data Collection > Integration with TKE** page, click **Data Collection Configuration** on the right of the instance to enter the collection configuration list page.
4. Click **Metrics collection management** at the top of the page to pop up the Basic Monitoring Metrics Collection Management page.
If you need to collect all container chart metrics, you can click **One-click collection of preset chart metrics** in the pop-up window, and then click **OK** in the pop-up confirmation window to collect all preset chart metrics.





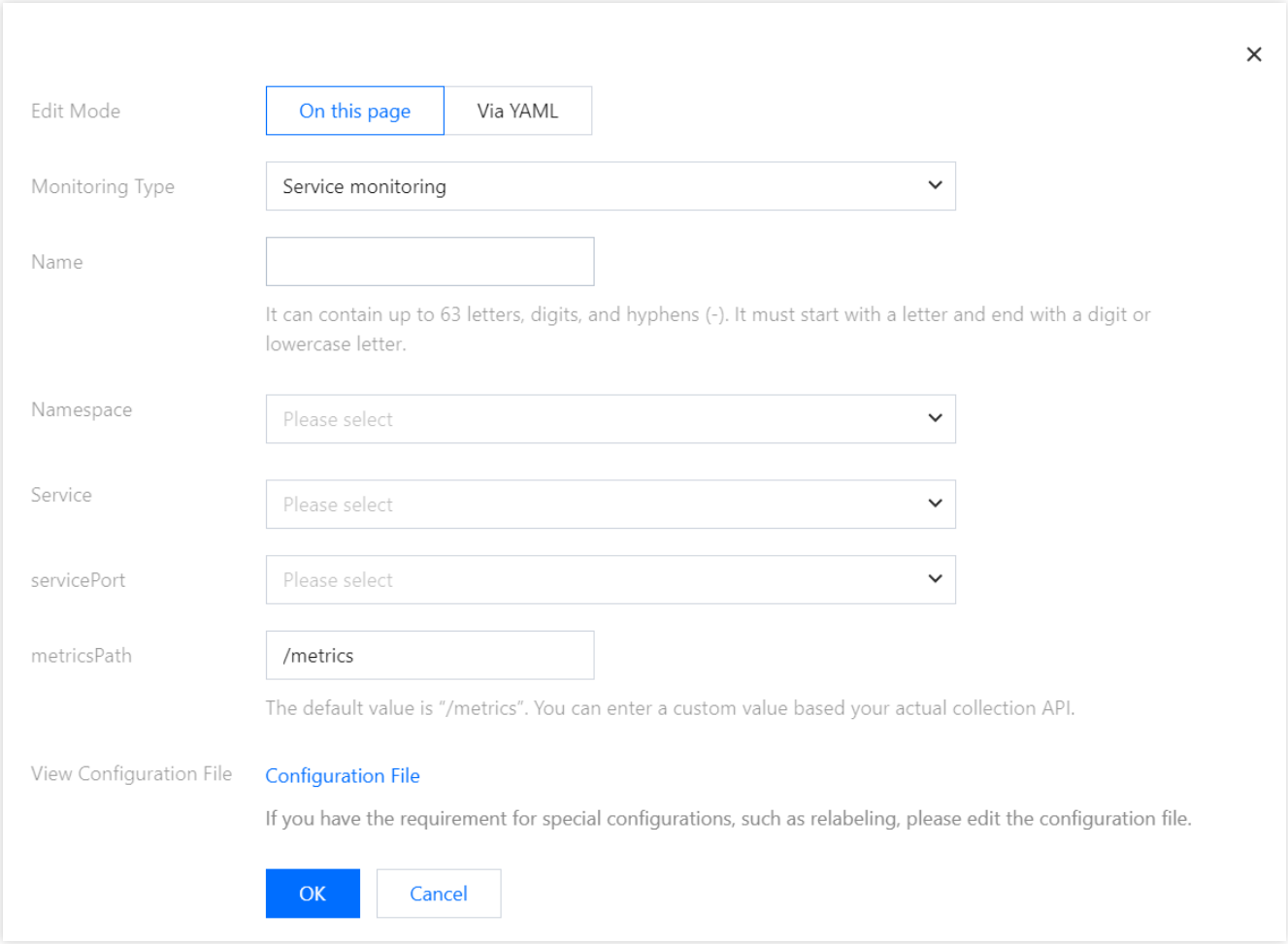
If only part of the metrics need to be collected, you can filter the metrics through, **monitoring charts** and **Recording Rule**, select the metrics you need, and then click **OK**. For details, see the following figure.



5. On the "Data Collection Configuration" page, click **Customize Monitoring Configuration** to add a new data collection configuration. TKE has preset some collection configuration files for collecting standard monitoring data. You can configure new data collection rules to monitor your business data in the following two ways:
Adding Configuration in the Console
Adding Configuration via YAML File

Monitoring Service

1. Click **On this Page**.
2. In the "Create Collection Configuration" pop-up window, fill in the configuration information. For details, see the following figure.



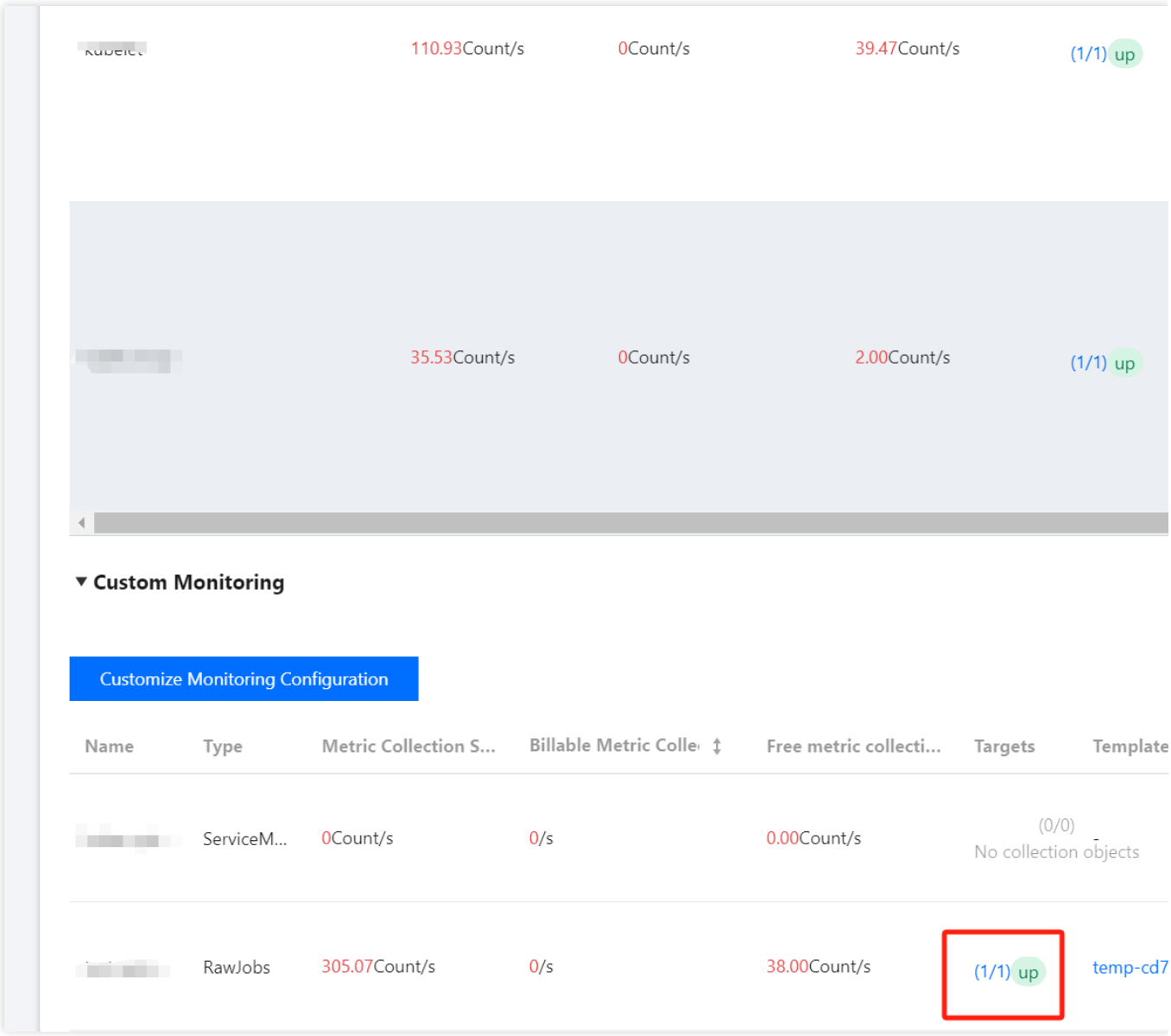
1. Click **Via YAML**.
2. In the pop-up window, select the monitoring type and fill in the corresponding configuration. You can complete the data collection configuration by submitting the corresponding YAML as per community usage.

Workload Monitoring: Corresponding configuration is PodMonitors.

Service Monitoring: Corresponding configuration is ServiceMonitors.

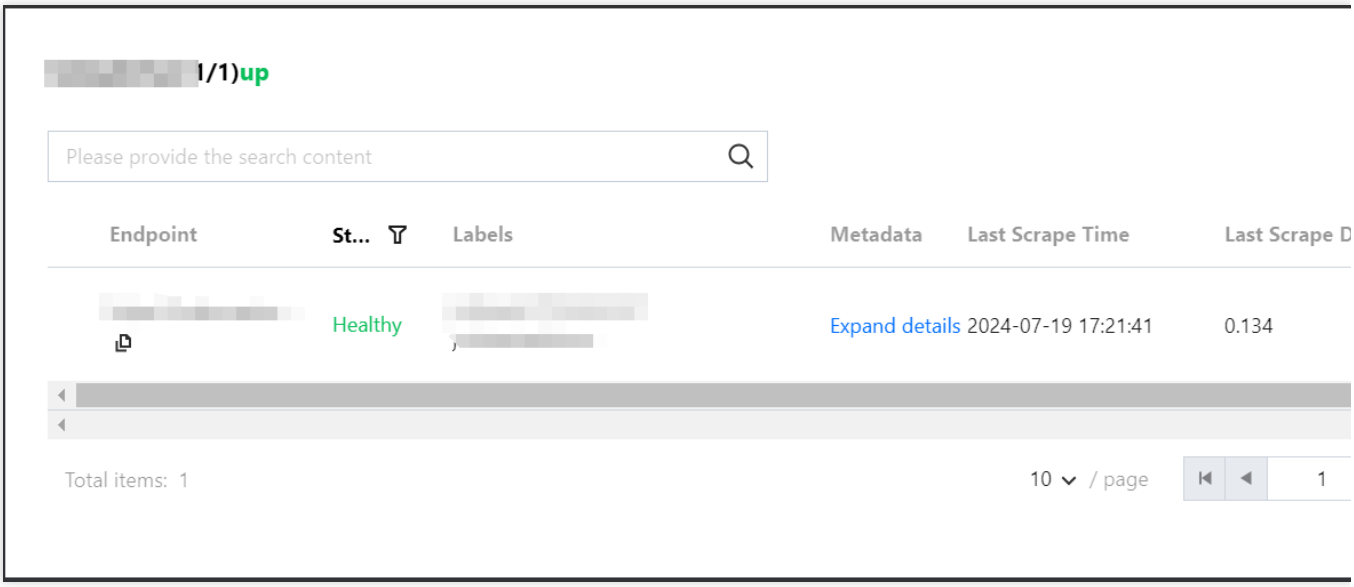
RawJobs Monitoring: Corresponding configuration is RawJobs.

6. Click **OK** to complete the configuration.
7. On the Data Collection Configuration page of the instance, view the status of the collection target. For details, see the following figure.



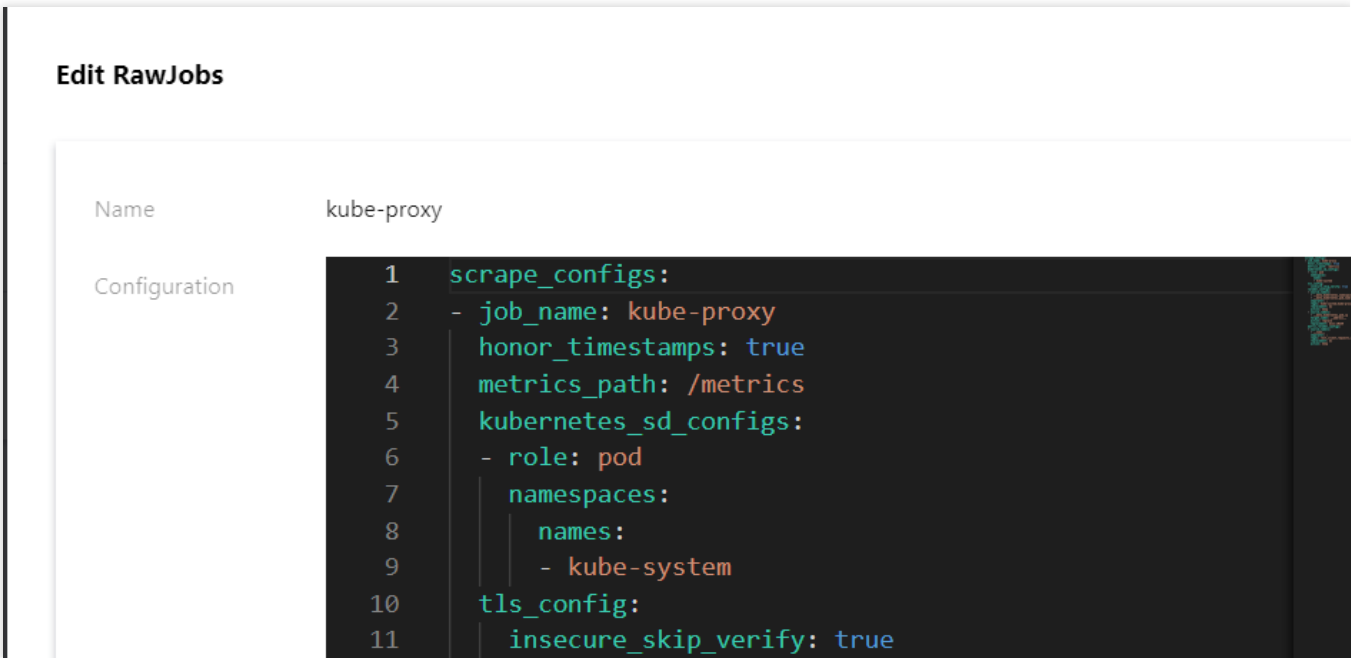
targets(1/1) means (the actual number of targets scraped is 1 / the number of detected collection targets is 1). When the actual number of scraped targets equals the number of detected targets, it is displayed as up, indicating that the current scrape is normal. When the actual number of scraped targets is less than the number of detected targets, it is displayed as down, indicating that some endpoints failed to scrape.

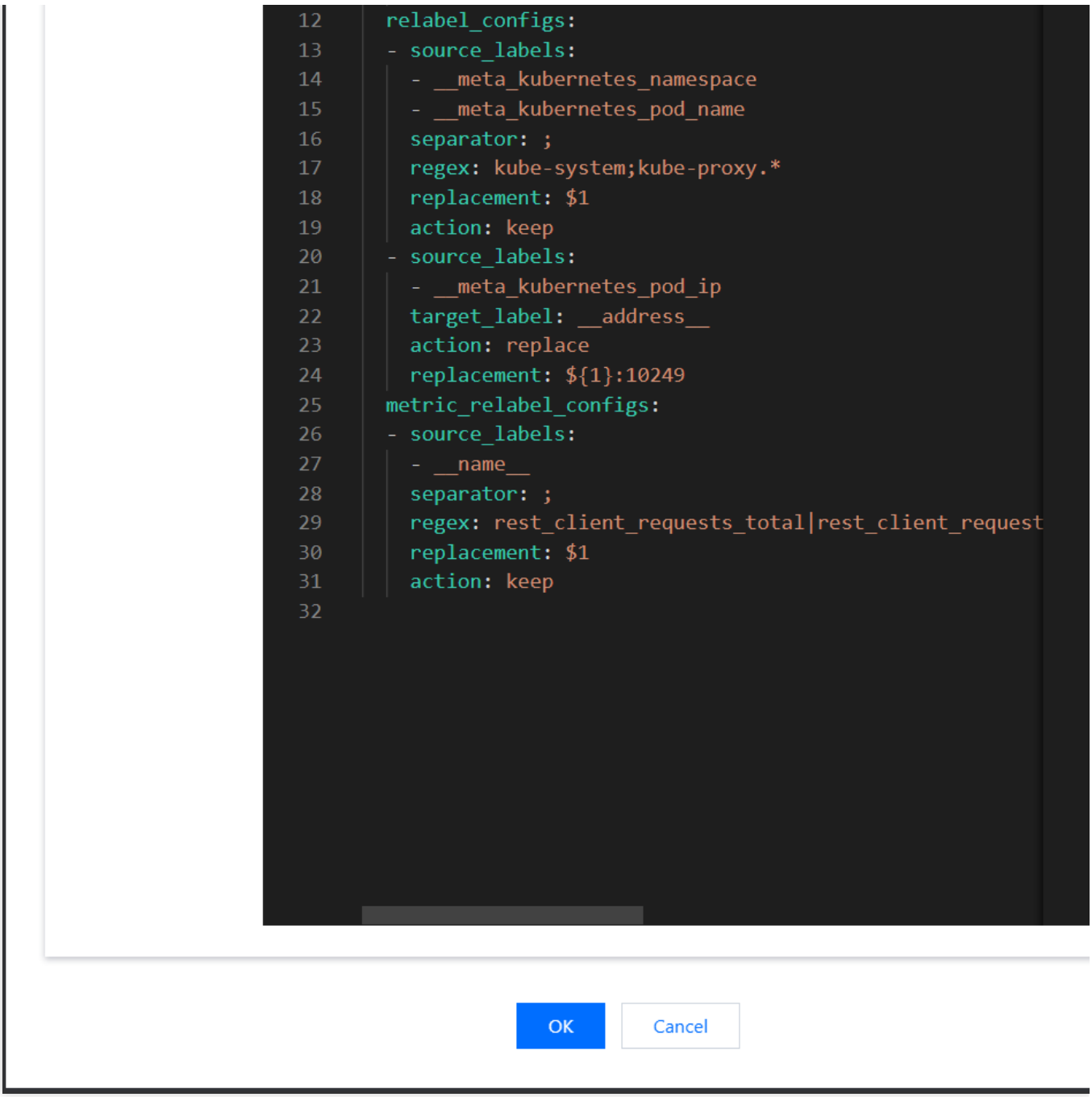
Click the field value **(1/1)** in the above figure to view more details about the collection target. For details, see the following figure.



View Existing Configurations

1. Log in to TMP Console.
2. On the instance list page, select the name of the instance that requires configuring data collection rules and enter its details page.
3. On the **Data Collection > Integration TKE** page, click **Data Collection Configuration** on the right side of the instance to enter the collection configuration list page. Select **Basic Monitoring** or **Custom Monitoring**, then click **Edit** on the right.
4. In the pop-up edit window, view all monitoring objects currently configured in the YAML file. For details, see the following figure.



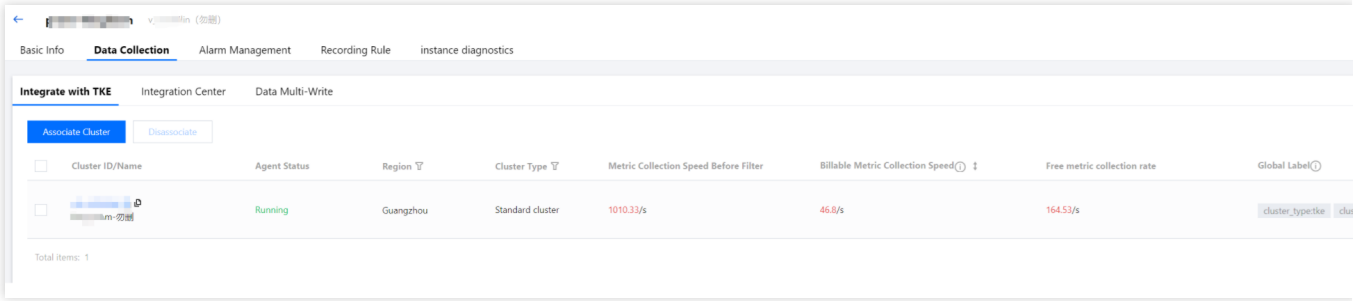


View Collection Targets

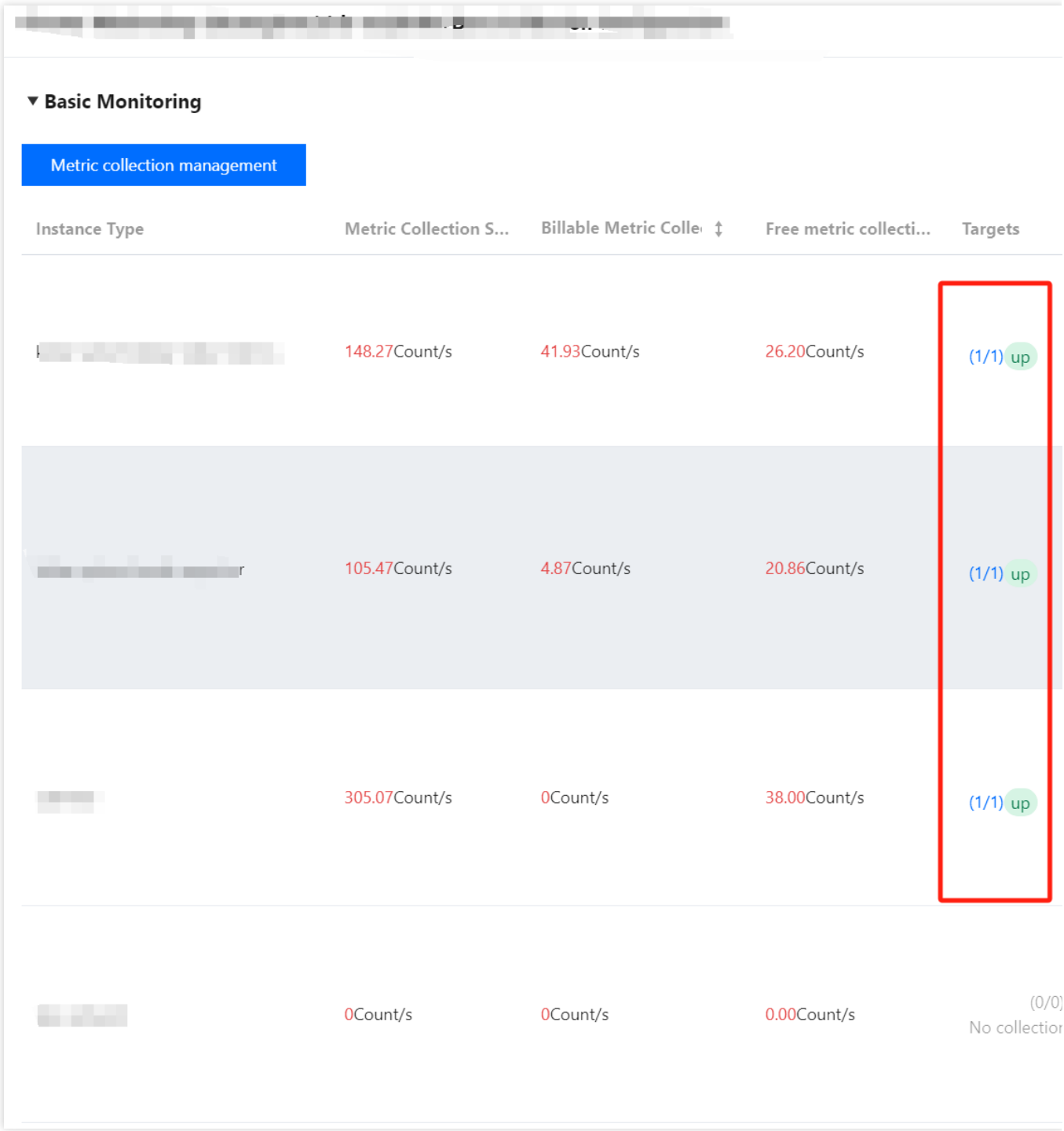
1. Log in to TMP Console.

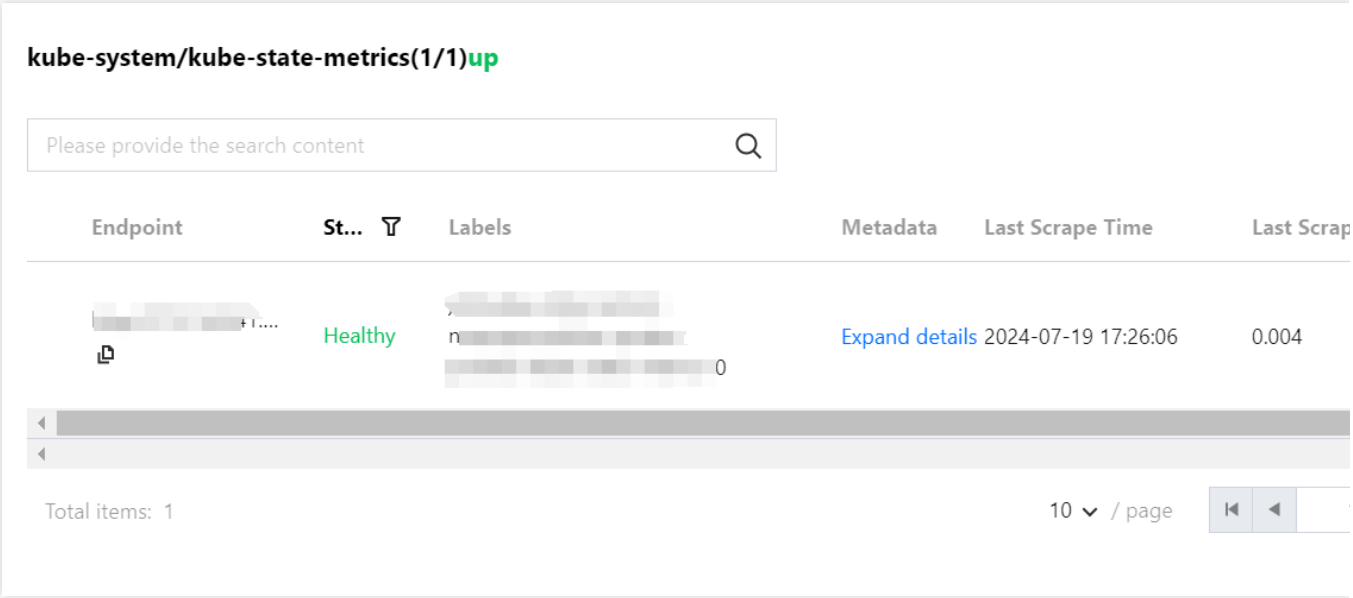
2. On the monitoring instance list page, select the instance name whose targets you want to view and enter the details page of the instance.

3. On the **Data Collection > Integration with TKE** page, click **Data Collection Configuration** on the right side of the instance.



4. In the pop-up window, click the status under targets to jump to the details page of the current data.





Related Operations

Mounting Files to a Collector

When configuring the collection item, if you need to provide some files for the configuration, such as a certificate, you can mount the files to the collector in the following way. The file update will be synchronized to the collector in real time.

prometheus.tke.tencent.cloud.com/scrape-mount = "true"

Add the above label to the configmap under the prom-xxx namespace. All keys will be mounted to the collector's path `/etc/prometheus/configmaps/[configmap-name]/` .

prometheus.tke.tencent.cloud.com/scrape-mount = "true"

Add the above label to the secret under the prom-xxx namespace. All keys will be mounted to the collector's path `/etc/prometheus/secrets/[secret-name]/` .

Configuring Necessary Monitoring Metrics

Last updated：2024-08-08 10:00:44

This document describes how to configure necessary **collection metrics** of TMP to avoid unnecessary expenses.

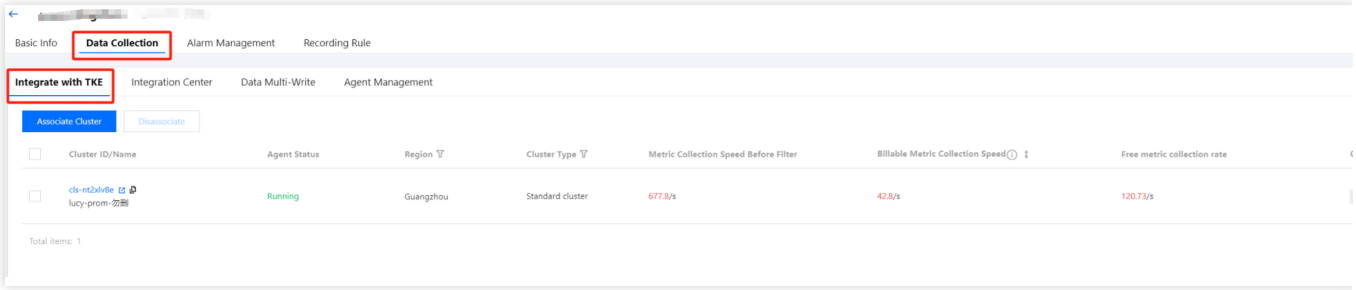
Prerequisites

Before configuring monitoring collection items, you need to perform the following operations:
Create a Prometheus monitoring instance.
Associate the cluster to be monitored with the instance.

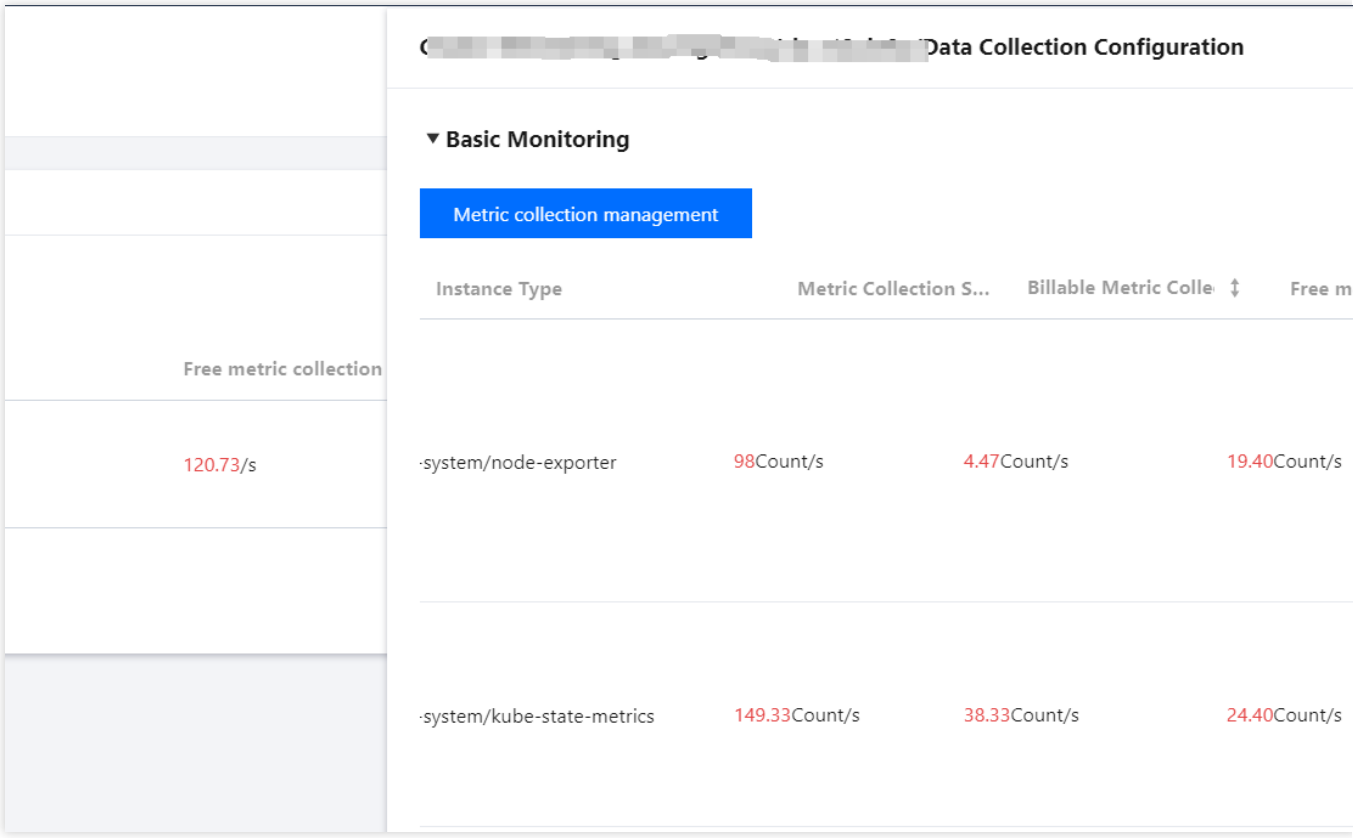
Configuring Necessary Metrics

Configuring Necessary Metrics via Console

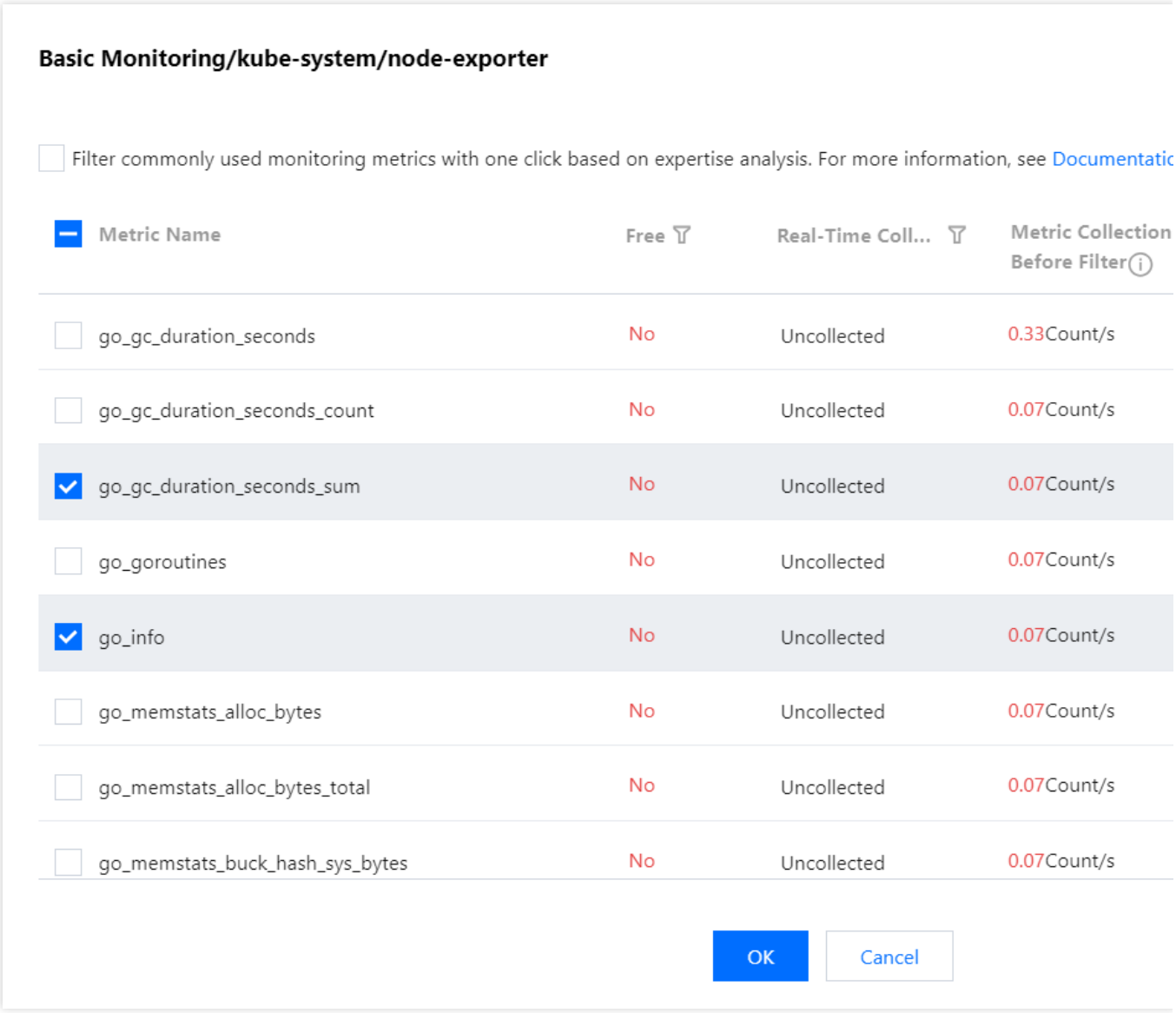
TMP provides over 100 free basic monitoring metrics. For the complete metric list, see Free Metrics in Pay-as-You-Go Mode.
1. Log in to the TCOP console, and click Prometheus Monitoring in the left sidebar.
2. On the instance list page, select the name of the instance that requires configuring data collection rules and enter its details page.
3. On the **Data Collection** page, click **Integration with TKE**, find the target cluster, and click **Data Collection Configuration** in the operation bar, as shown below:



4. In the data collection configuration window, click **Metric Details** in the operation bar to deselect unnecessary metrics, as shown below:



5. On the page below, you can view free and billed metrics. If a metric is selected, corresponding data will be collected. You can select metrics as needed. Only basic monitoring provides free metrics. For all free metrics, see Free Metrics in Pay-as-You-Go Mode. For billing of paid metrics, see TMP Pay-as-You-Go.



Configuring Necessary Metrics via YAML File

The billing of TMP currently is based on the number of monitored data points. To minimize unnecessary expenses, it is recommended that you optimize the collection configuration to collect data of only necessary metrics, thereby reducing the overall data amount. For billing details and the usage of related cloud resources, see Resource Usage and Billing Overview.

The following part describes how to add filters to ServiceMonitor, PodMonitor, and native Job to configure necessary metrics.

1. Log in to the TCOP console, and click Prometheus Monitoring in the left sidebar.
2. On the instance list page, select the name of the instance that requires configuring data collection rules and enter its details page.
3. On the **Data Collection** page, click **Integration with TKE**, find the target cluster, and click **Data Collection Configuration** in the operation bar.

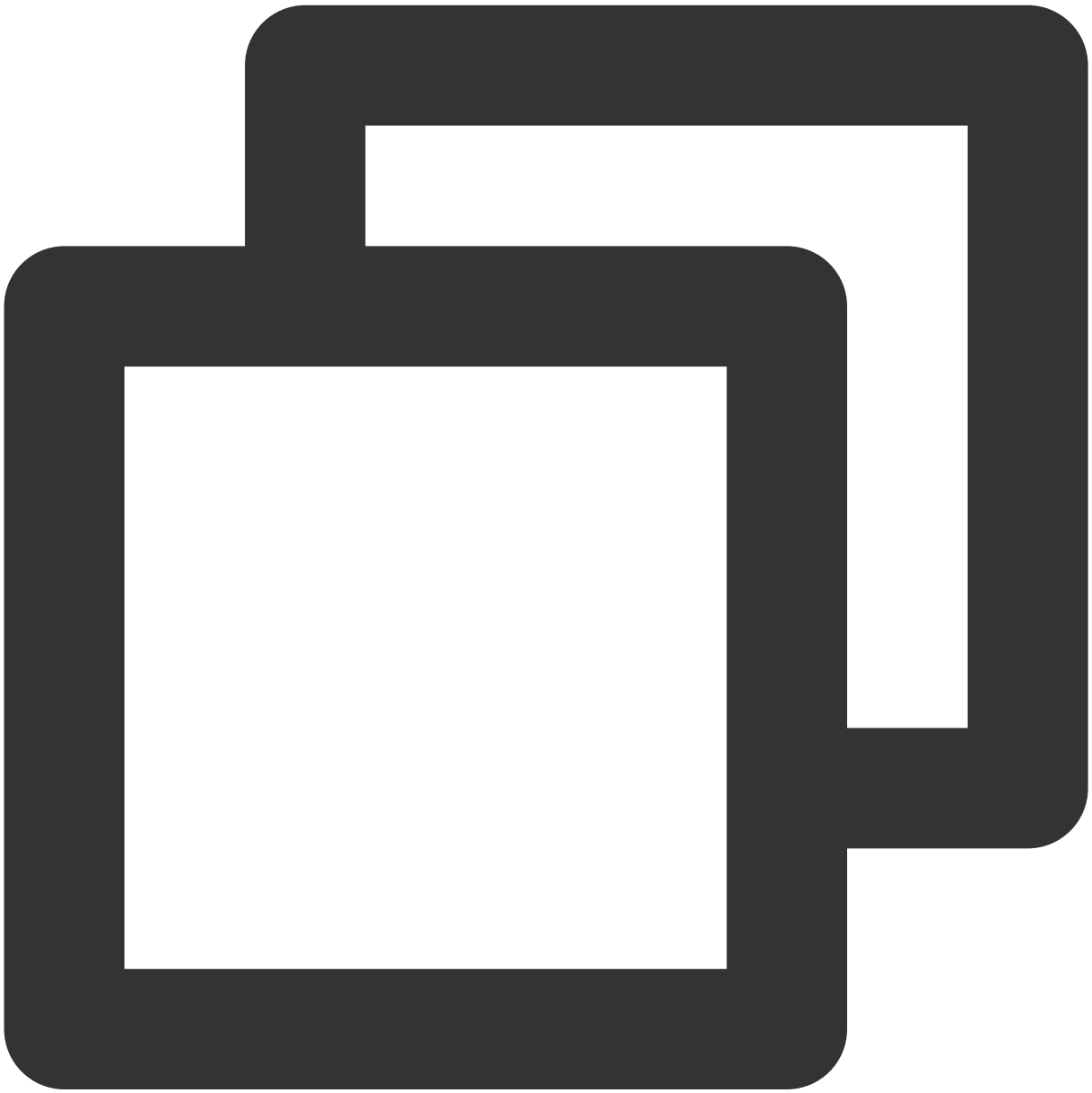
4. Find the target instance, and click **Editing** in the operation bar to view metric details.

ServiceMonitor and PodMonitor

Native Job

The filter configurations of ServiceMonitor and PodMonitor are the same. Take ServiceMonitor as an example.

ServiceMonitor example:

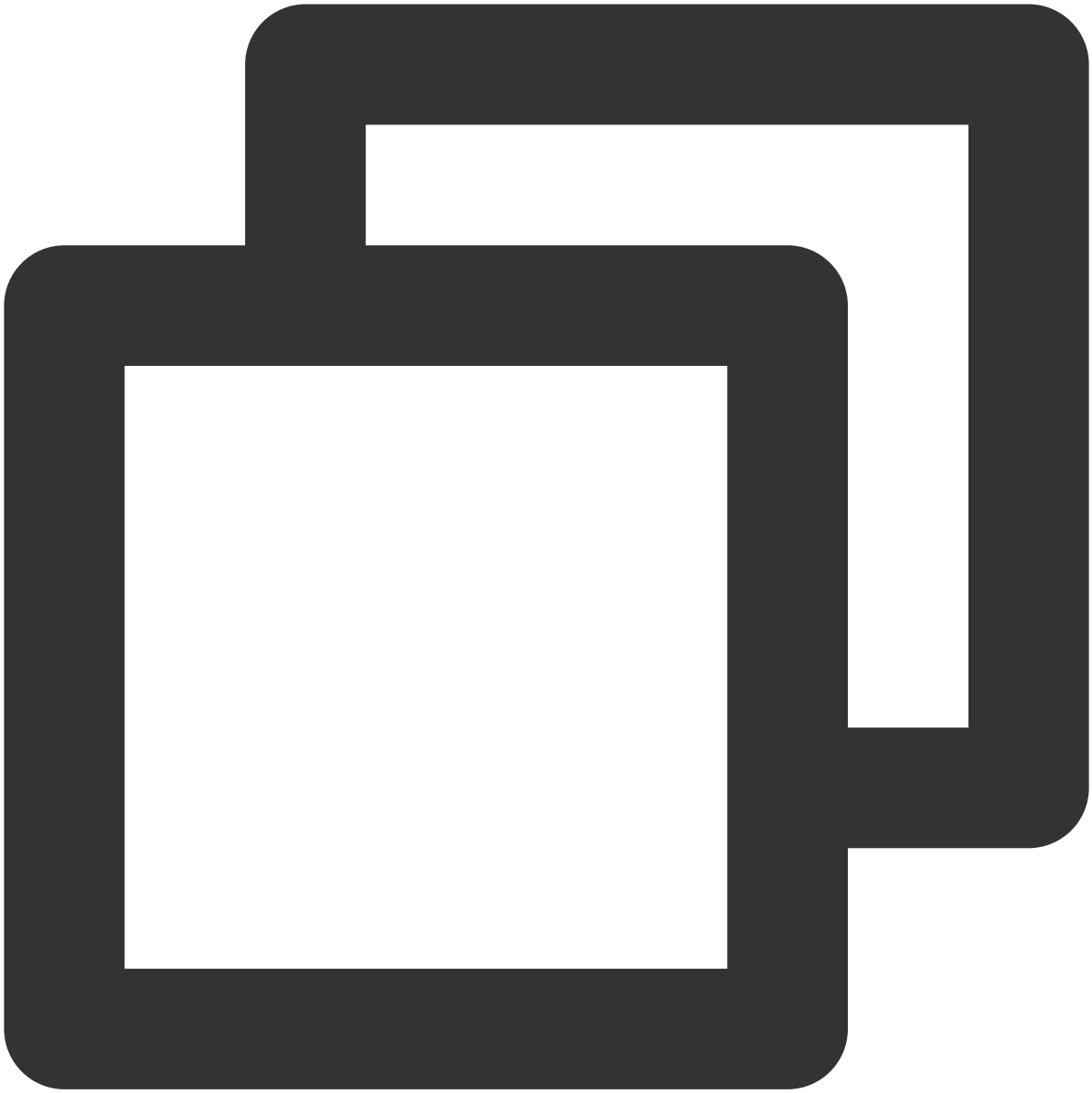


```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
```

```
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 1.9.7
  name: kube-state-metrics
  namespace: kube-system
spec:
  endpoints:
  - bearerTokenSecret:
      key: ""
    interval: 15s # This parameter indicates the collection interval. You can incre
    port: http-metrics
    scrapeTimeout: 15s # This parameter indicates the collection timeout. TMP requi
  jobLabel: app.kubernetes.io/name
  namespaceSelector: {}
  selector:
    matchLabels:
      app.kubernetes.io/name: kube-state-metrics
```

For example, if you want to collect data of metrics `kube_node_info` and `kube_node_role`, add the configuration with the `metricRelabelings` field to the endpoints list of ServiceMonitor. Note that the field is **`metricRelabelings`**, not `relabelings`.

Example of adding `metricRelabelings`:

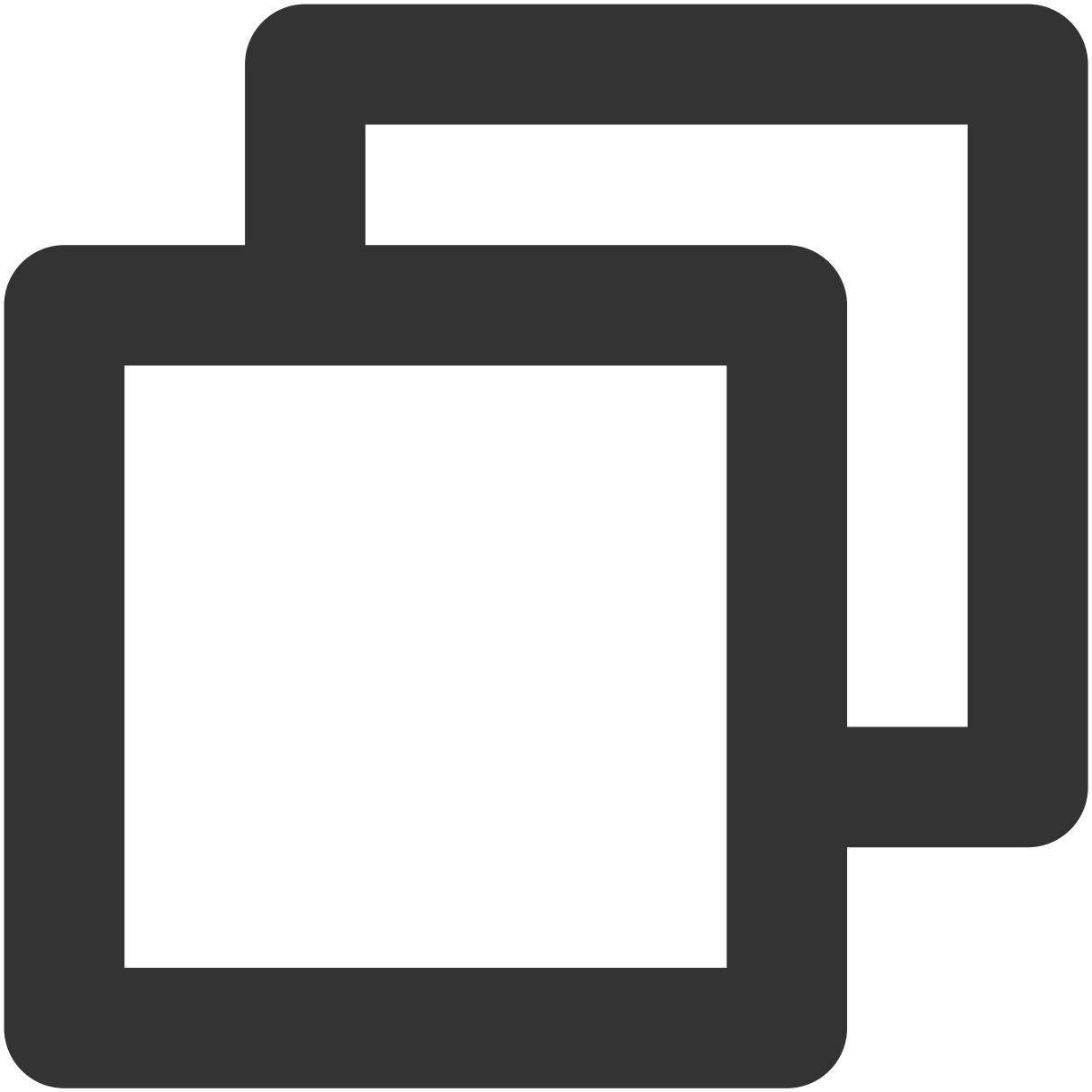


```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 1.9.7
  name: kube-state-metrics
  namespace: kube-system
spec:
  endpoints:
  - bearerTokenSecret:
```

```
      key: ""
    interval: 15s # This parameter indicates the collection interval. You can incre
    port: http-metrics
    scrapeTimeout: 15s
    # The following four lines are added:
    metricRelabelings: # Each metric will undergo the following process.
    - sourceLabels: ["__name__"] # The label name to be checked. __name__ refers to
      regex: kube_node_info|kube_node_role # Check whether the label matches this r
      action:  keep # If the metric name meets the above conditions, the metric is
  jobLabel: app.kubernetes.io/name
  namespaceSelector: {}
  selector:
```

If native Jobs of Prometheus are used, filter metrics using the method below.

Job example:

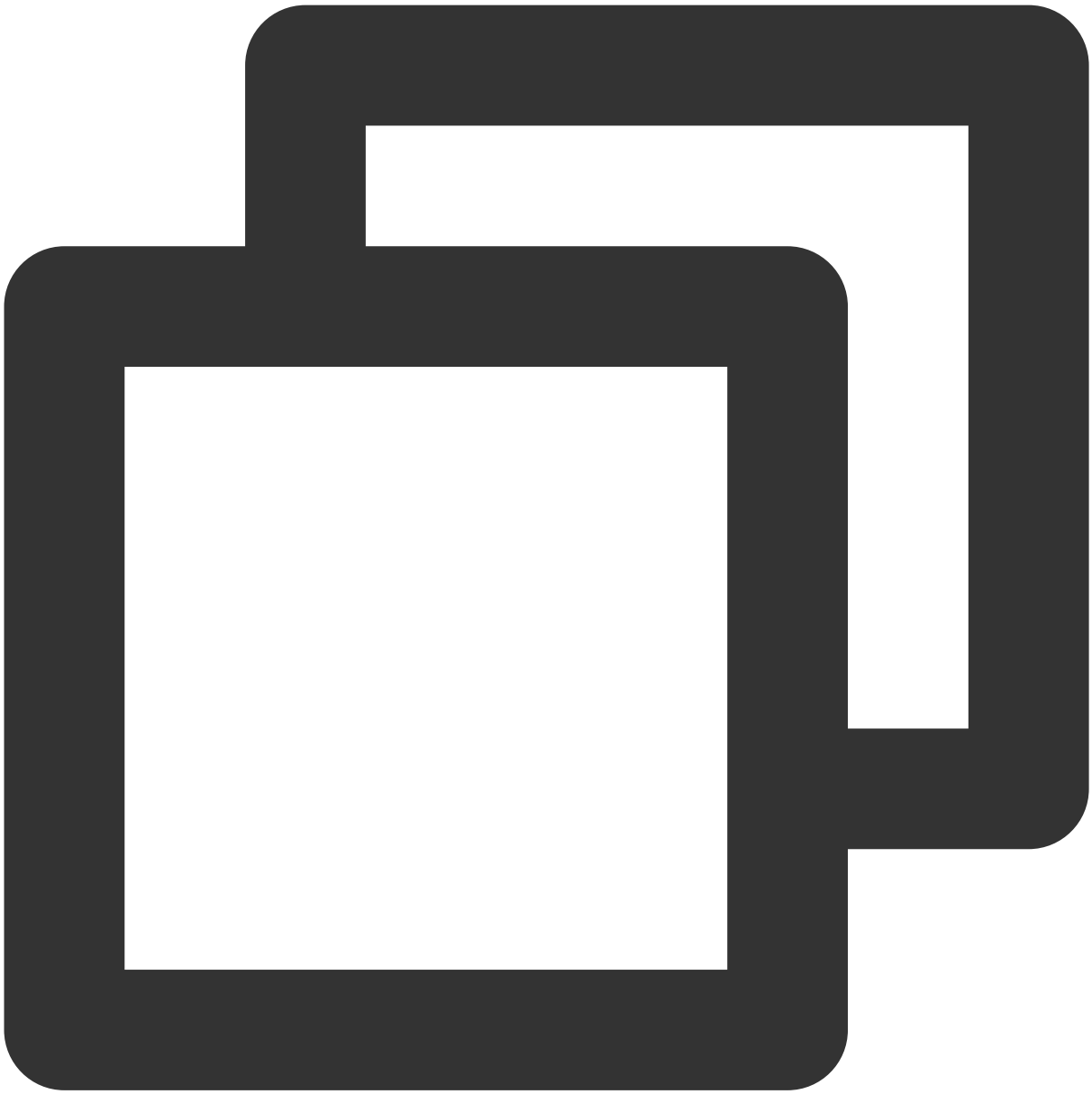


```
scrape_configs:
  - job_name: job1
    scrape_interval: 15s # This parameter indicates the collection interval. You ca
    static_configs:
      - targets:
          - '1.1.1.1'
```

For example, if you want to collect data of metrics `kube_node_info` and `kube_node_role` only, add the configuration with the field `metric_relabel_configs` . Note: The field is **`metric_relabel_configs`** , not

`relabel_configs`.

Example of adding `metric_relabel_configs`:



```
scrape_configs:
  - job_name: job1
    scrape_interval: 15s # This parameter indicates the collection interval. You ca
    static_configs:
    - targets:
      - '1.1.1.1'
    # The following four lines are added:
    metric_relabel_configs: # Each metric will undergo the following process.
```

```
    - source_labels: ["__name__"] # The label name to be checked. __name__ refers t
      regex: kube_node_info|kube_node_role # Check whether the label matches this r
      action:  keep # If the metric name meets the above conditions, the metric is
```

5. Click **Confirm**.

Skipping Data Collection

Skipping Data Collection from Entire Namespace

TMP will manage all ServiceMonitors and PodMonitors in a cluster by default after the cluster is associated. If you want to skip data collection from a namespace, you can add the label `tps-skip-monitor: "true"` to the specified namespace. For label details, see Labels and Selectors.

Skipping Data Collection

TMP collects monitoring data by creating ServiceMonitor and PodMonitor types of CRD resources in a cluster. If you want to skip data collection from specified ServiceMonitor and PodMonitor resources, you can add the label `tps-skip-monitor: "true"` to these CRD resources. For label details, see Labels and Selectors.

Adjusting Collection Interval

Last updated：2024-08-08 10:02:32

This document describes how to adjust the **sampling interval** of TMP to avoid unnecessary expenses.

Prerequisites

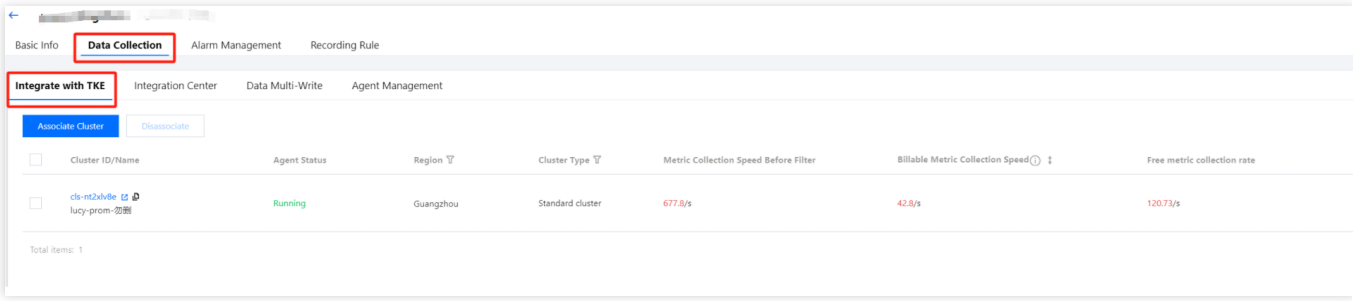
Before configuring monitoring collection items, you need to perform the following operations:

Create a Prometheus monitoring instance.

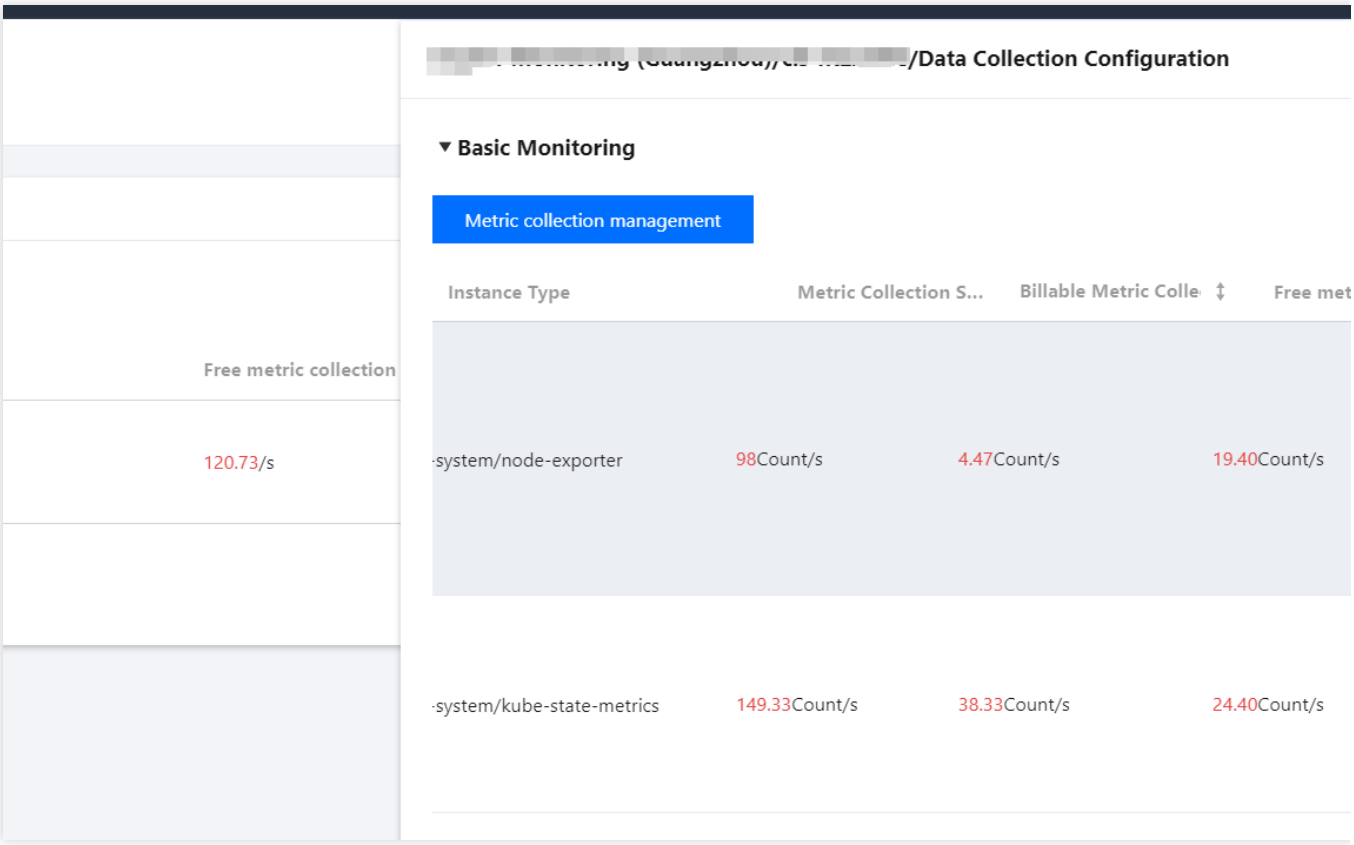
Associate the cluster to be monitored with the instance.

Adjusting Collection Interval via TCOP Console

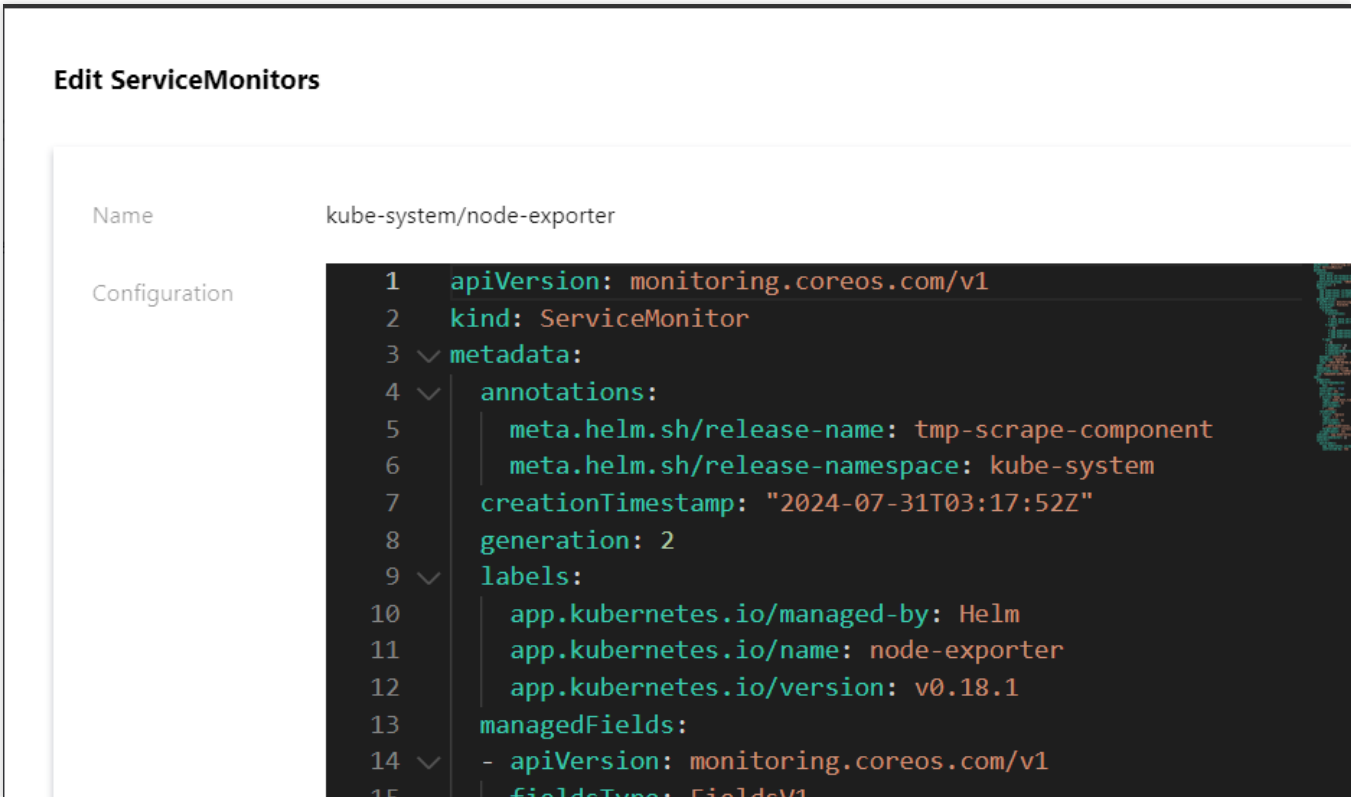
1. Log in to the TCOP console, and click Prometheus Monitoring in the left sidebar.
2. On the instance list page, select the name of the instance that requires configuring data collection rules and enter its details page.
3. On the **Data Collection** page, select **Integrate with TKE** , click **Data Collection Configuration** on the right side of the cluster, as shown in the following figure:

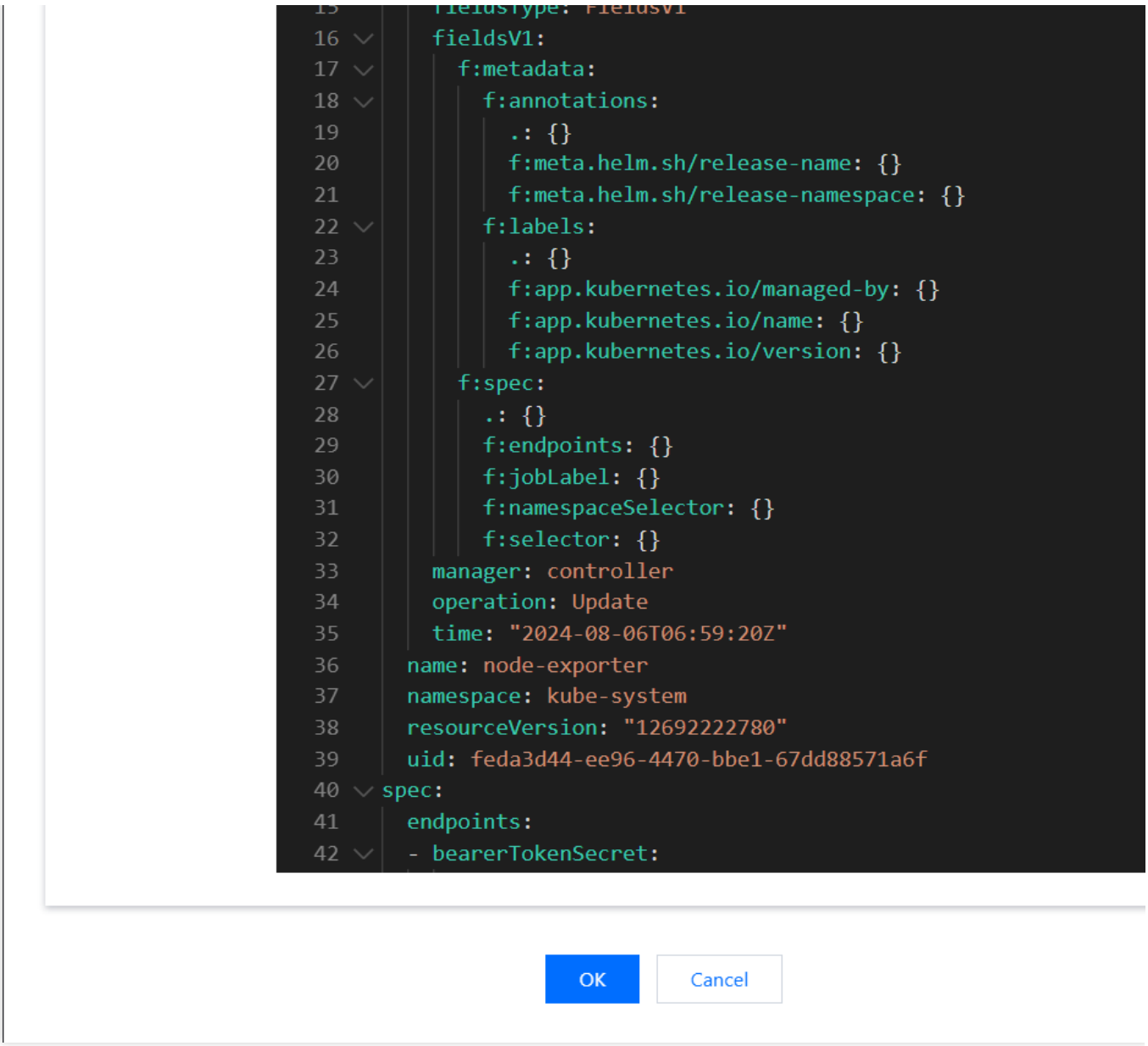


4. In the data collection configuration window, click **Edit** in the operation bar, as shown below:



5. Find the **interval** field, which represents the collection interval. You can increase its value to reduce data storage costs, as shown below:





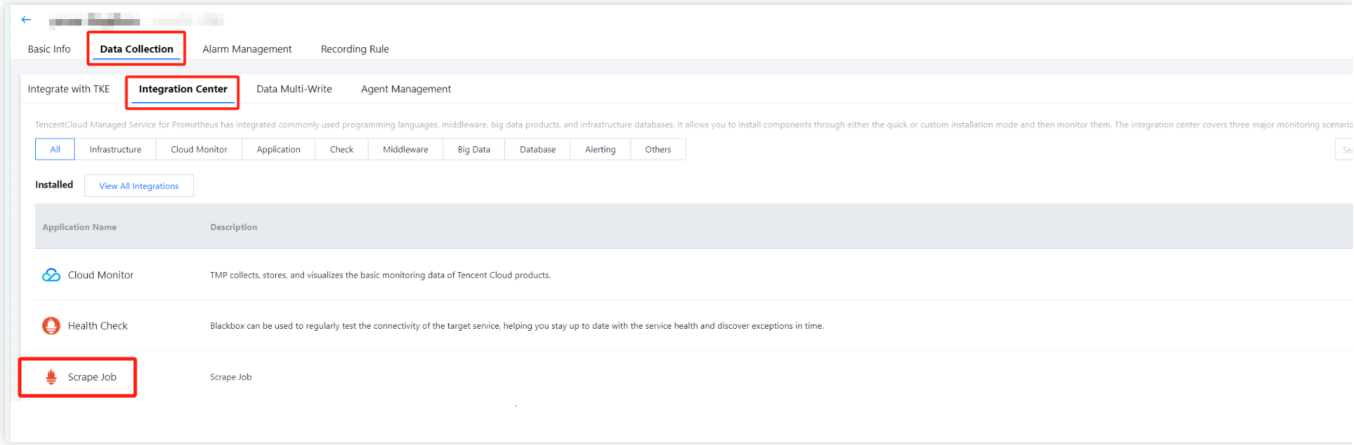
6. Click **Confirm** after completing the configuration.

Adjusting Collection Interval via Integration Center

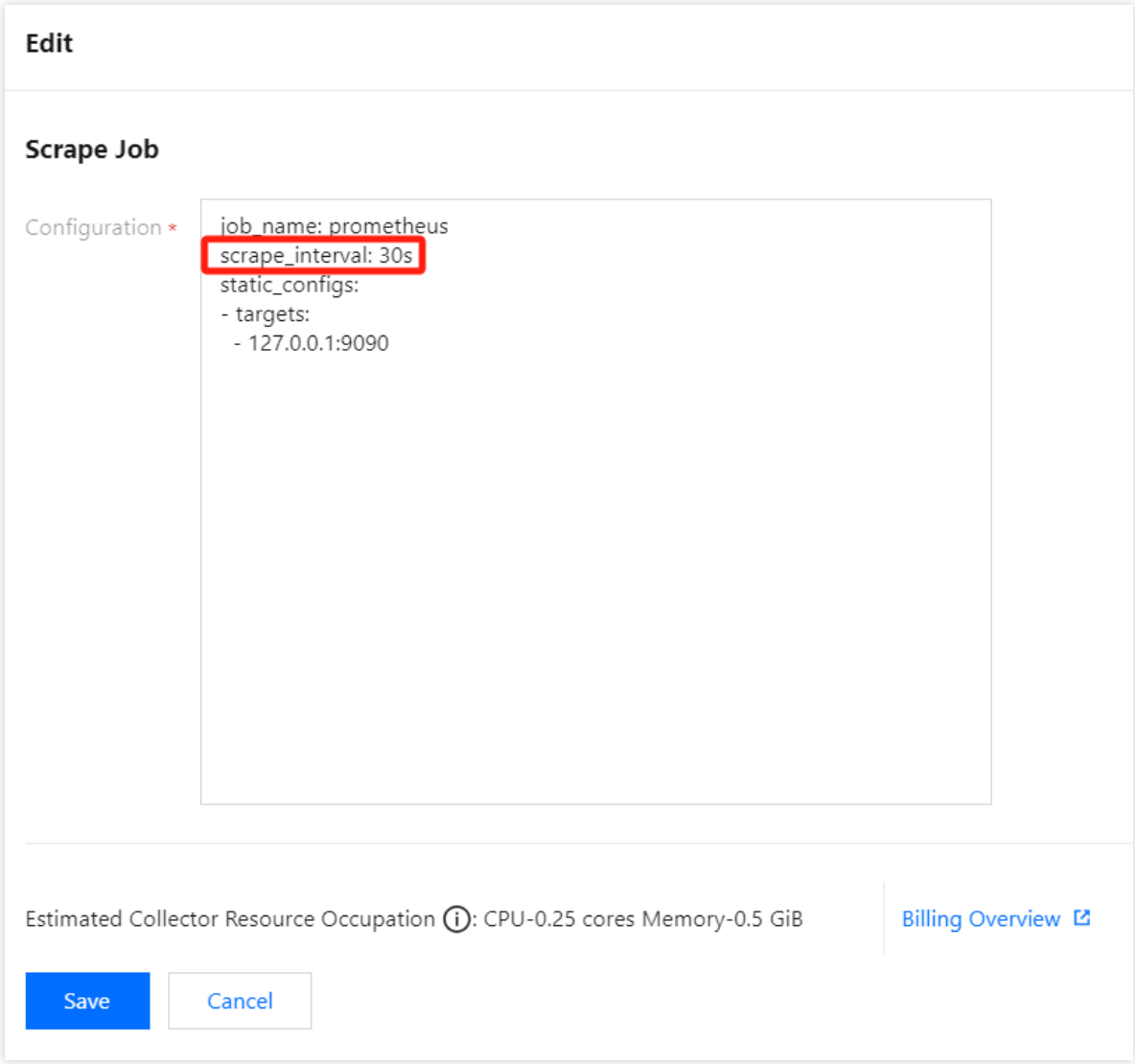
Note:

The Scraping Task application should have been installed and integrated. If it is not installed, click **One-click installation** on the right before you proceed with the following steps.

1. Log in to the TCOP console, and click Prometheus Monitoring in the left sidebar.
2. On the instance list page, select the name of the instance that requires configuring data collection rules and enter its details page.
3. On the **Data Collection** page, click **Integration Center**, find **Scrape Job**, and click **View Integrations** on the right, as shown below:



4. On the page that appears, click any name to enter the collection tasks configuration page. Then, find the **scrape_interval** field, which represents the collection interval. You can increase its value to reduce data storage costs, as shown below:



5. Click **Save** after completing the configuration.

Resource Usage and Billing Overview

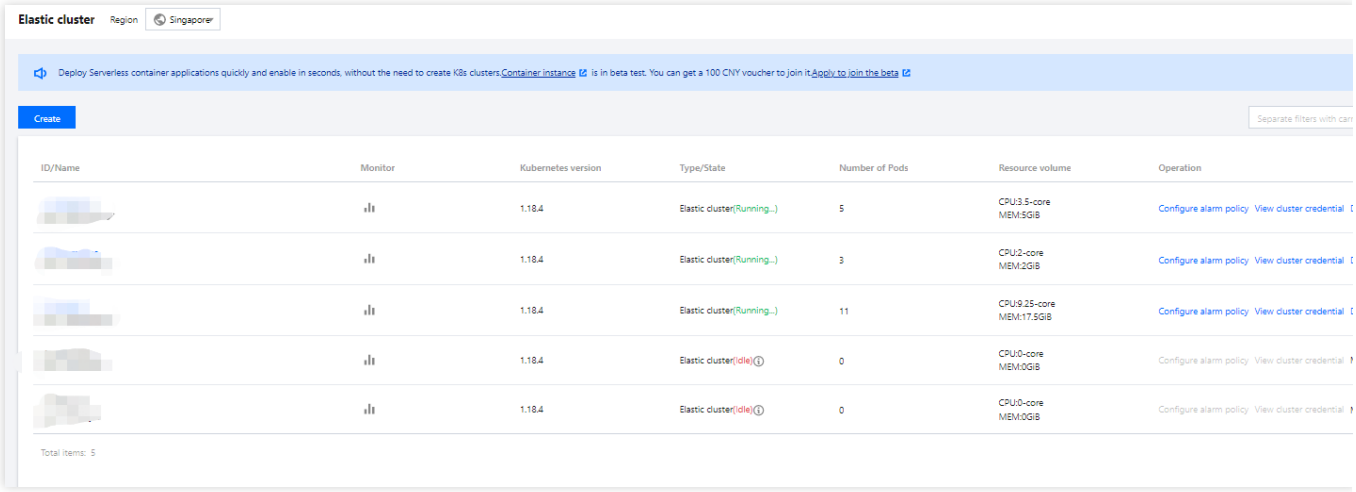
Last updated：2024-08-07 22:07:52

When using the Tencent Managed Service for Prometheus (TMP) service, you may use resources such as **Tencent Kubernetes Engine (TKE) serverless clusters, TencentCloud Managed Service for Grafana (TCMG), and Cloud Load Balancer (CLB)**. This document describes the use cases and billing rules of these resources.

TKE Serverless Cluster

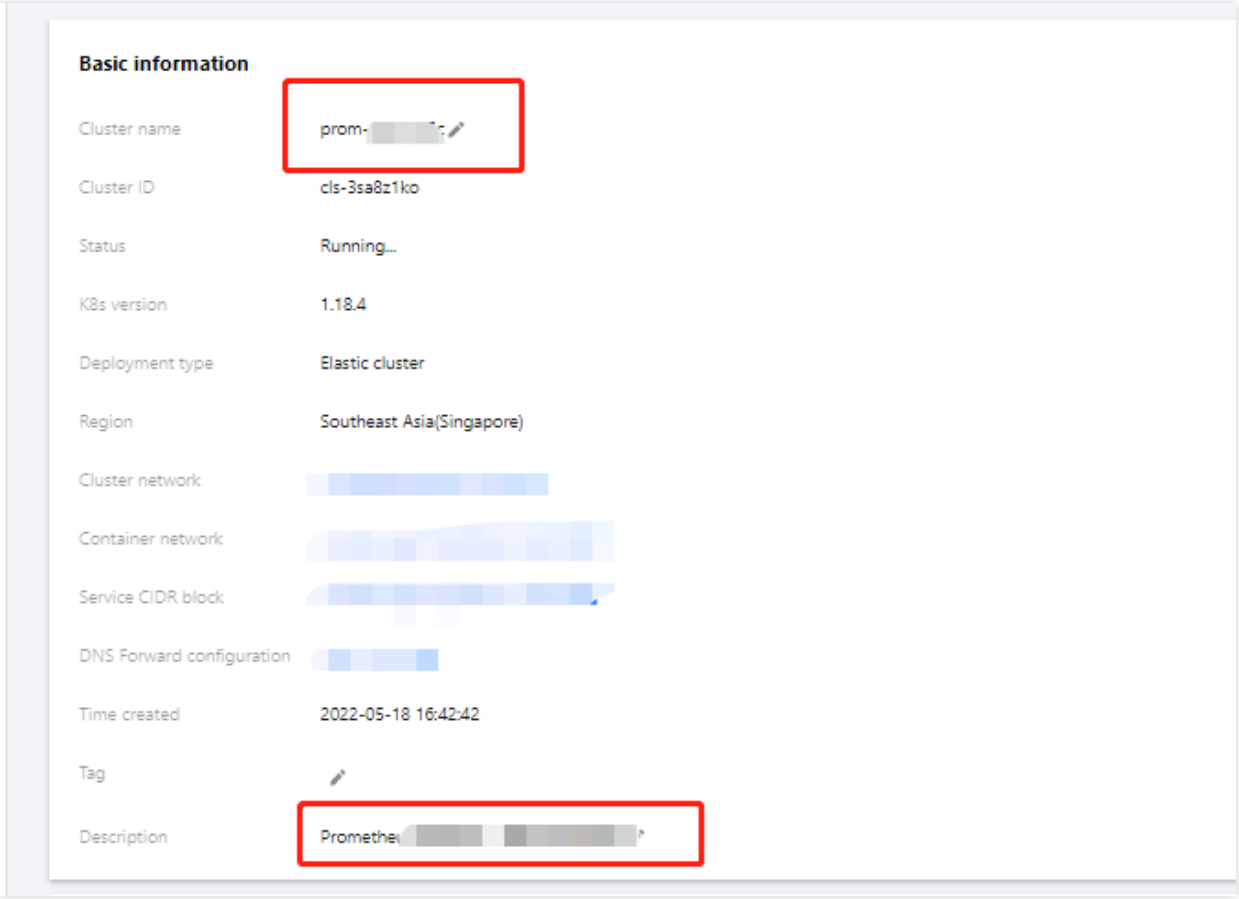
Use cases

You need to create a TKE serverless cluster if you use a TMP-associated cluster to monitor TKE.
A TKE serverless cluster will be automatically created for data collection when you install an integration plugin in the TMP integration center.
If both use cases are required for you, only one TKE serverless cluster will be created and shared. You can view the created clusters on the cluster list page.



Note

The name of the TKE serverless cluster is the TMP **instance ID**, and the cluster description states that **For TMP use only. Do not modify or delete**.



Billing overview

The billing mode is **pay-as-you-go**. For more information, see Product Pricing.

The TKE serverless cluster automatically scales according to the monitoring size. The relationship between the monitoring size and the TKE serverless cluster cost is shown below:

Reported Instantaneous Series	Estimated TKE Serverless Cluster Resources Required	List Price/Day
< 500,000	1.25 cores, 1.6 GiB	0.35 USD
1 million	0.5 cores, 1.5 GiB*2	1.46 USD
5 million	1 core, 3 GiB*3	2.93 USD
20 million	1 core, 6 GiB*5	7.98 USD
30 million	1 core, 6 GiB*8	12.77 USD

Sample TKE serverless cluster costs are as follows:

If the TKE serverless cluster used for a newly initialized TMP instance consumes 1.25 CPU cores and 1.5 GiB memory, then the estimated list price per day will be 0.0319 *24 + 0.0132* 24 = 1.0824 USD.

TCMG

Use cases

When creating a TMP instance, you need to associate it with a TCMG instance in the same region for the visual display of monitoring data collected by TMP. For billing information, see Billing Overview.

CLB

When you use a TMP-associated cluster to monitor TKE, a private network CLB instance will be created under your account for network connectivity between the collector and the cluster.
If you associate an edge cluster or a cluster with no network connection, a public network CLB instance will be created for network connectivity.
To access the TCMG service over the public network, you need to create a public network CLB instance.
These CLB resources will be charged. You can view the resource information of the created public network CLB instances in the CLB console.
Resources are billed based on the actual usage. For billing details, see Network Pricing.

Resource Termination

If you terminate TMP instances in the TMP console, all relevant resources will also be terminated. Tencent Cloud does not repossess TMP instances proactively. If you no longer use TMP, you need to delete the instances promptly to avoid extra charges. For instance termination directions, see Terminating Instance.

Practical Tutorial
Migration from Self-Built Prometheus

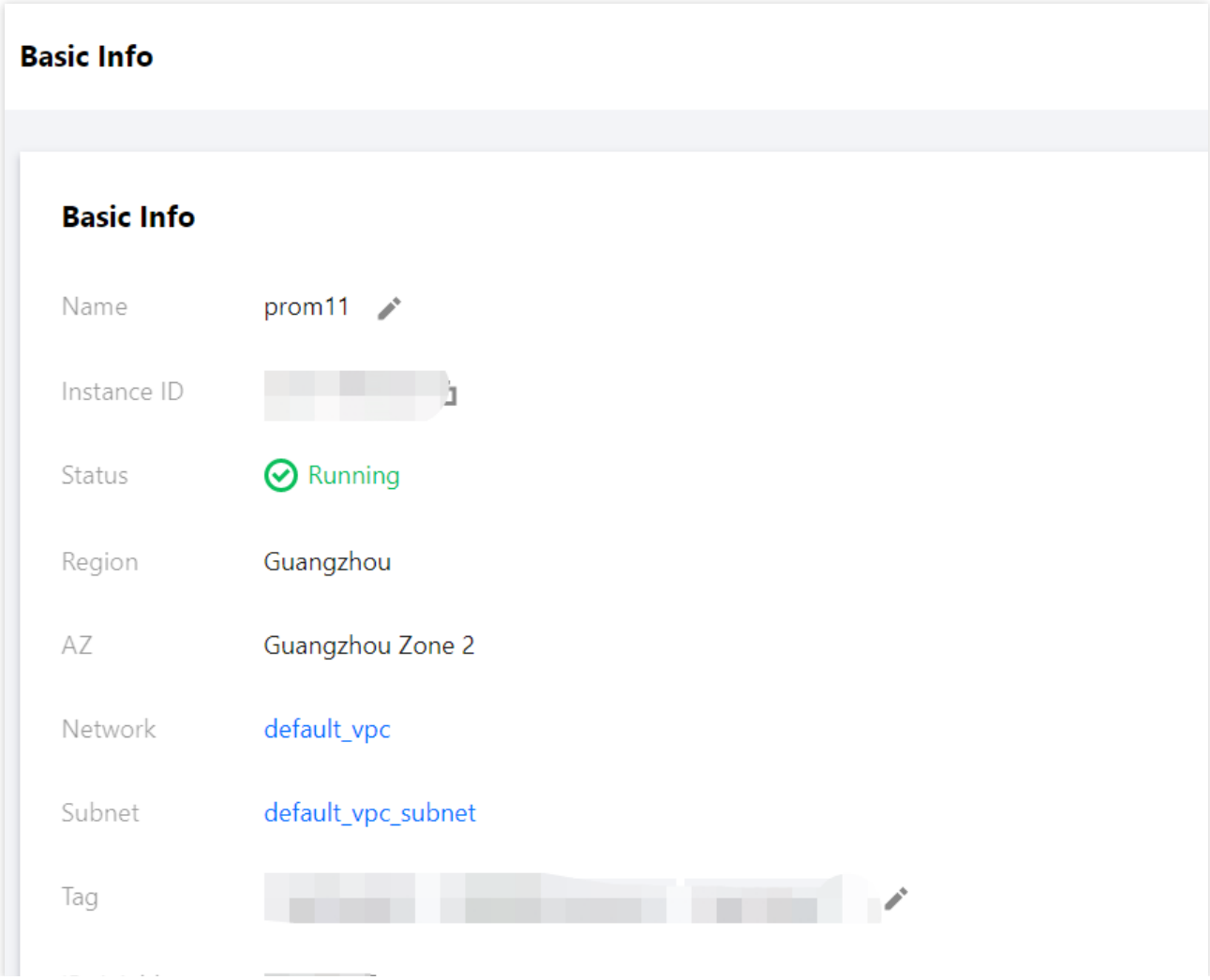
Last updated：2024-08-07 22:08:03

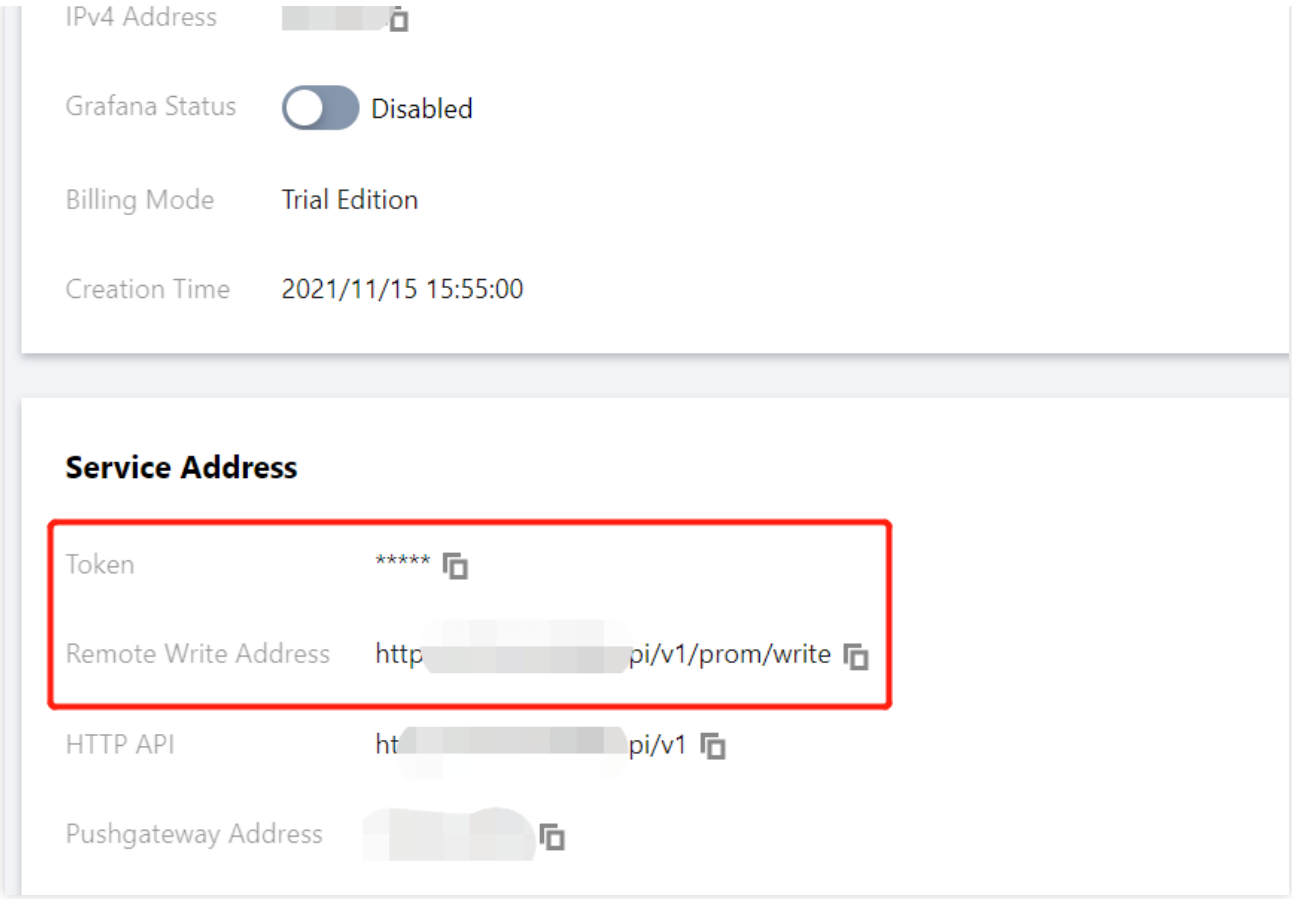
Overview

You can quickly migrate from your self-built Prometheus service to TMP.

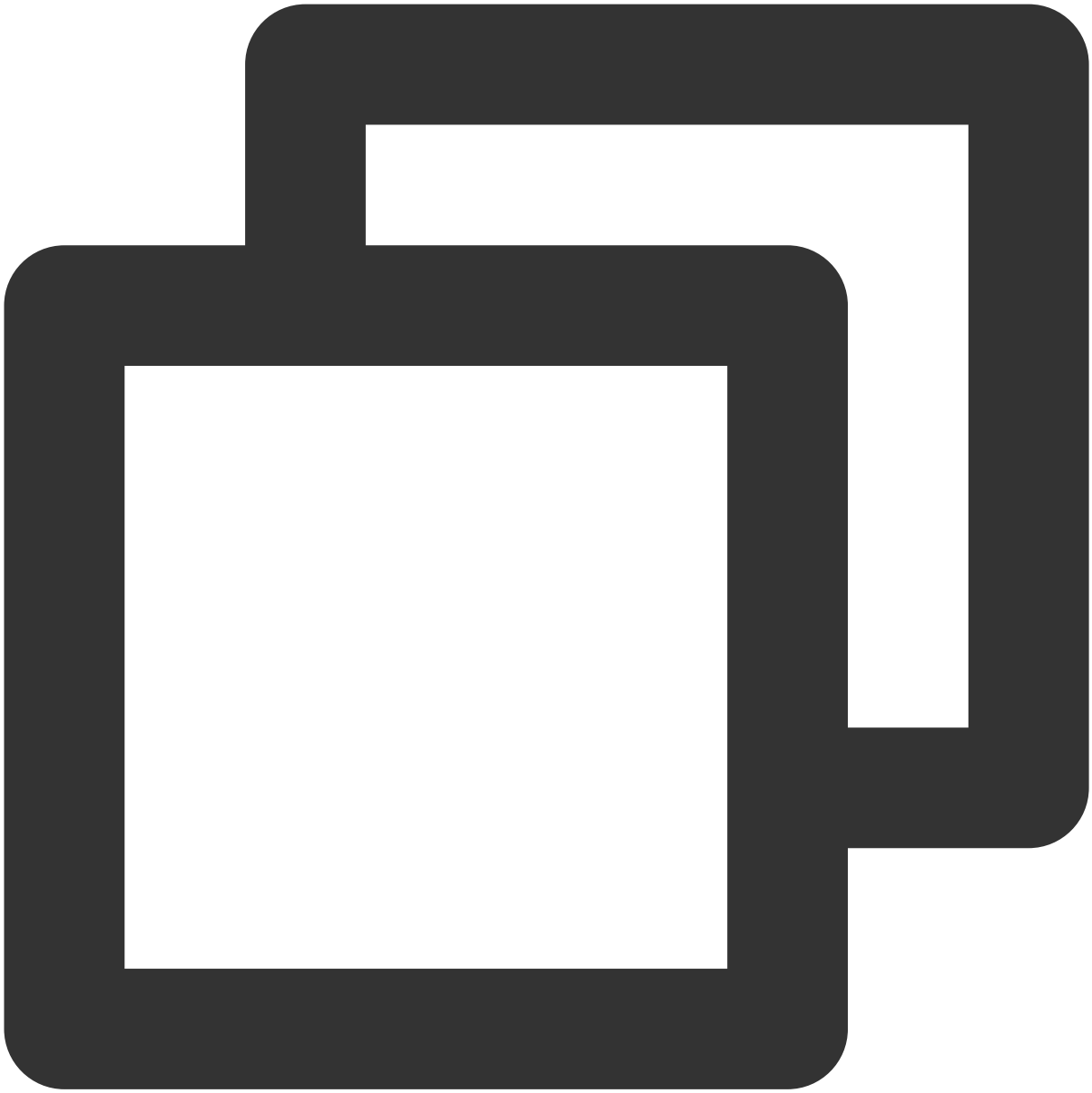
Directions

Prometheus itself supports remote write to an external storage; therefore, you can add a remote write configuration pointing to TMP in the configuration file of your self-built Prometheus. The steps are as follows:
1. Get the remote write address and token of TMP through the basic information of the instance as follows:



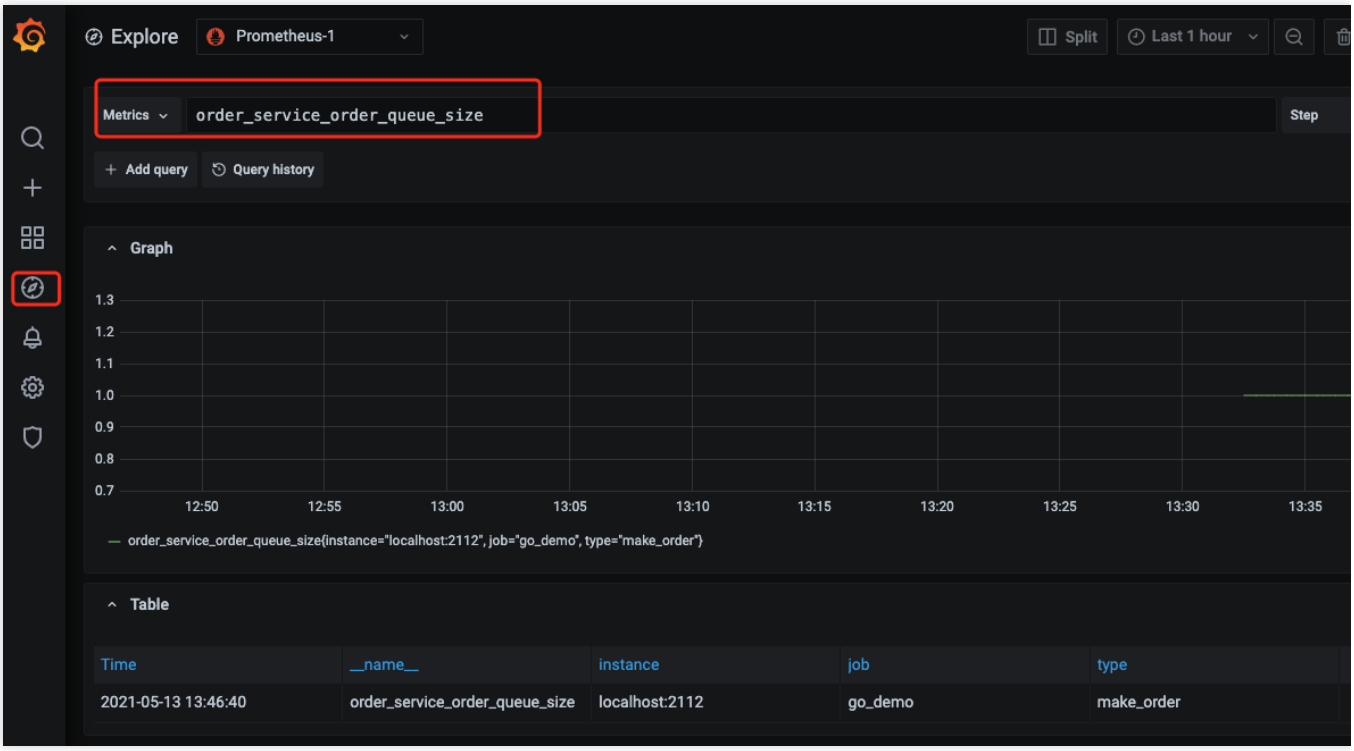


2. Modify `prometheus.yml` and restart Prometheus. The specific configuration is as follows. For more information on the remote write configuration parameters, please see remote_write.



```
remote_write:

  - name: cm_prometheus # Remote write name
    url: http://ip:port/api/v1/prom/write  # Get the remote write address from the
    remote_timeout: 30s # Set according to the actual situation
    bearer_token: k32*****trR # Get the token information from the basic informatio
```

3. Open the Grafana service that comes with TMP and use `Explore` to verify whether the data is written successfully as shown below. You can also customize Grafana monitoring dashboards.



4. You can also use Prometheus APIs for self-built visualization. For more information, please see Monitoring Data Query.

Custom Integration with CVM

Last updated：2024-08-07 22:08:08

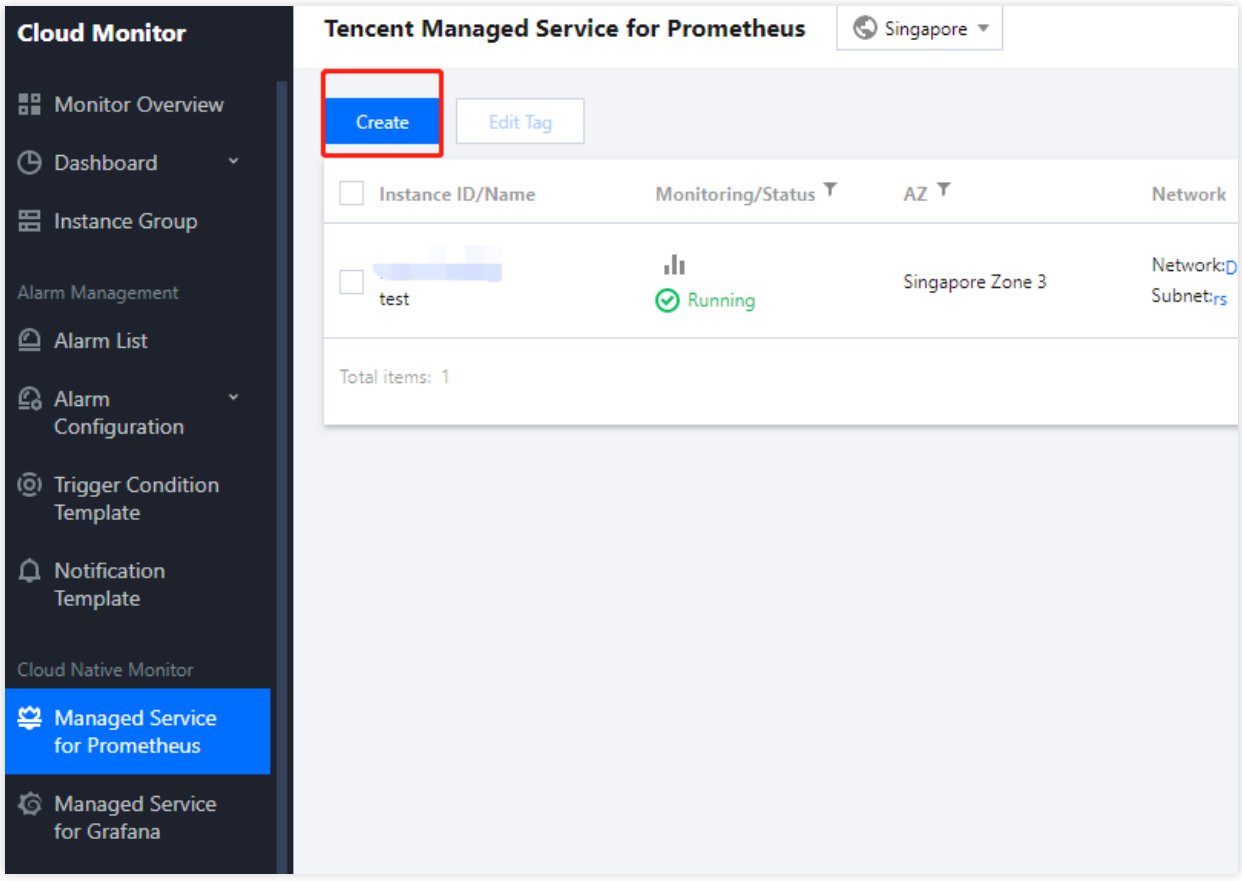
This document describes how to integrate CVM with TMP.

Purchasing a TMP Instance

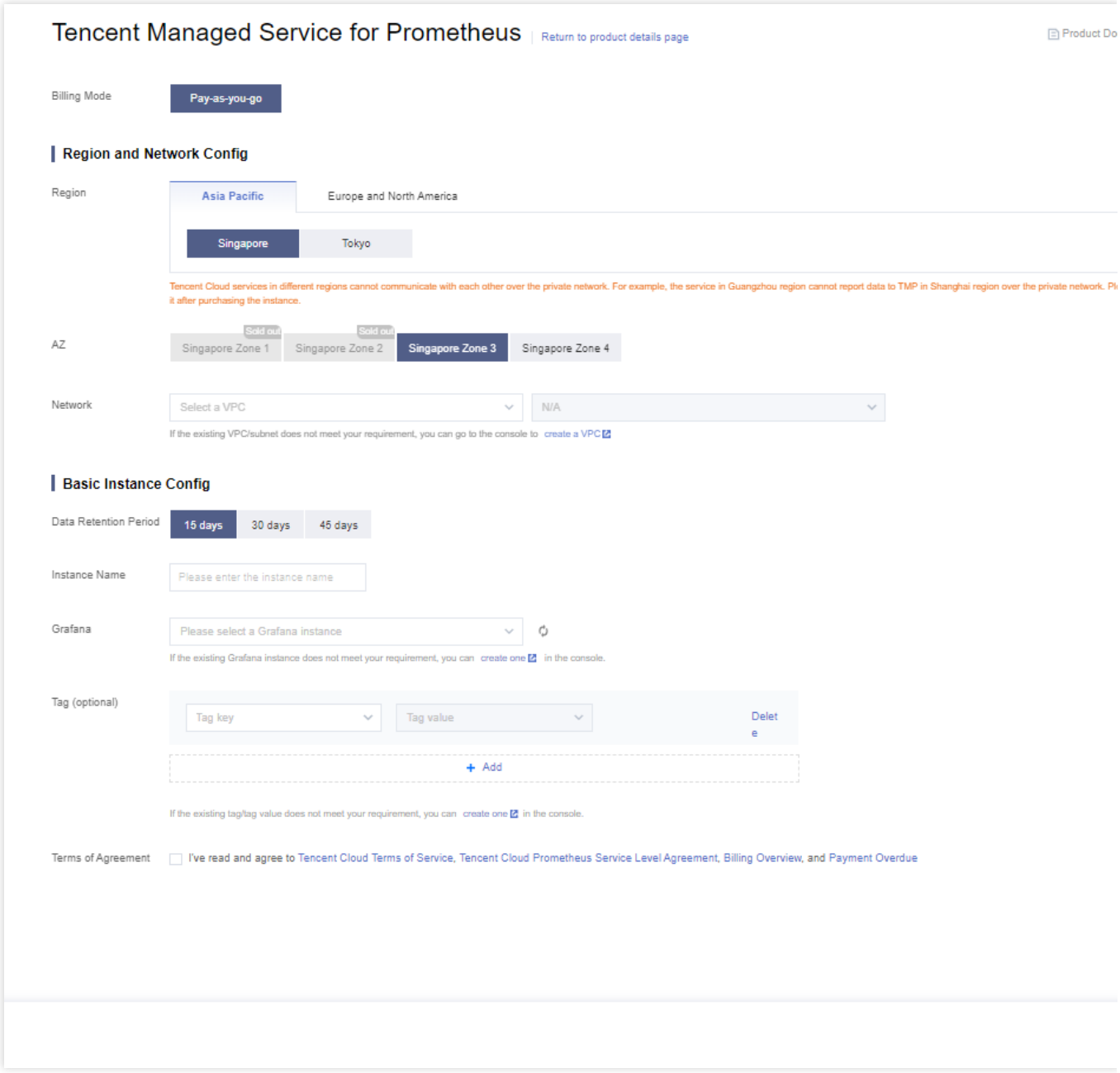
Note:

The purchased TMP instance must be in the same VPC as the monitored CVM instance for network connectivity.

1. Log in to the TMP console and click **Create** to purchase a TMP instance.



2. On the purchase page, select the target instance specification and network. Make sure that the TMP and CVM instances have the same VPC IP range so that data can be collected. Select the instance specification based on your reported data volume.

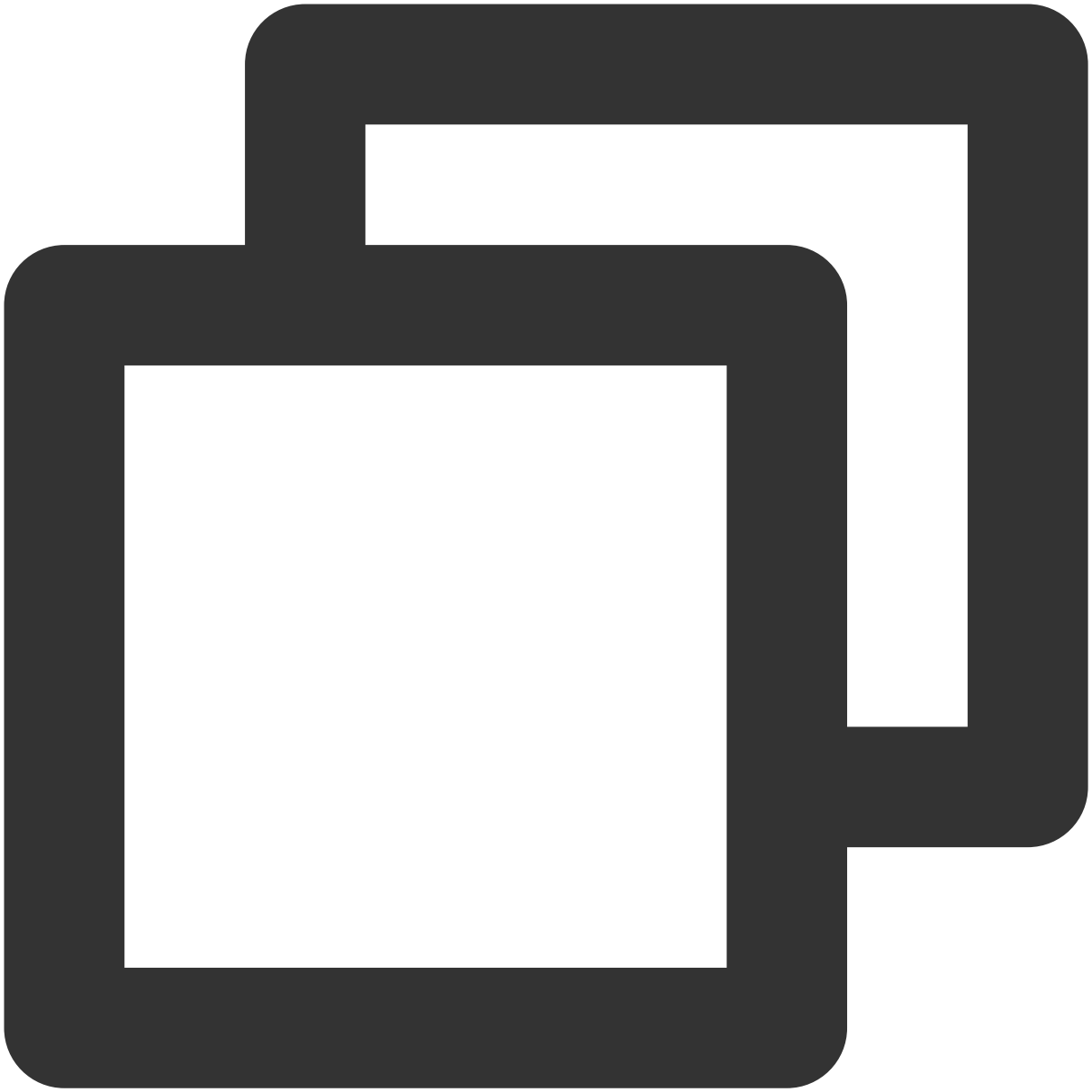


3. Click **Buy Now** and make the payment.

Integrating CVM Basic Metrics

1. Download and install Node Exporter.

Download and install Node Exporter (used to collect basic metric data) in the target CVM instance. Click here or run the following command for download:



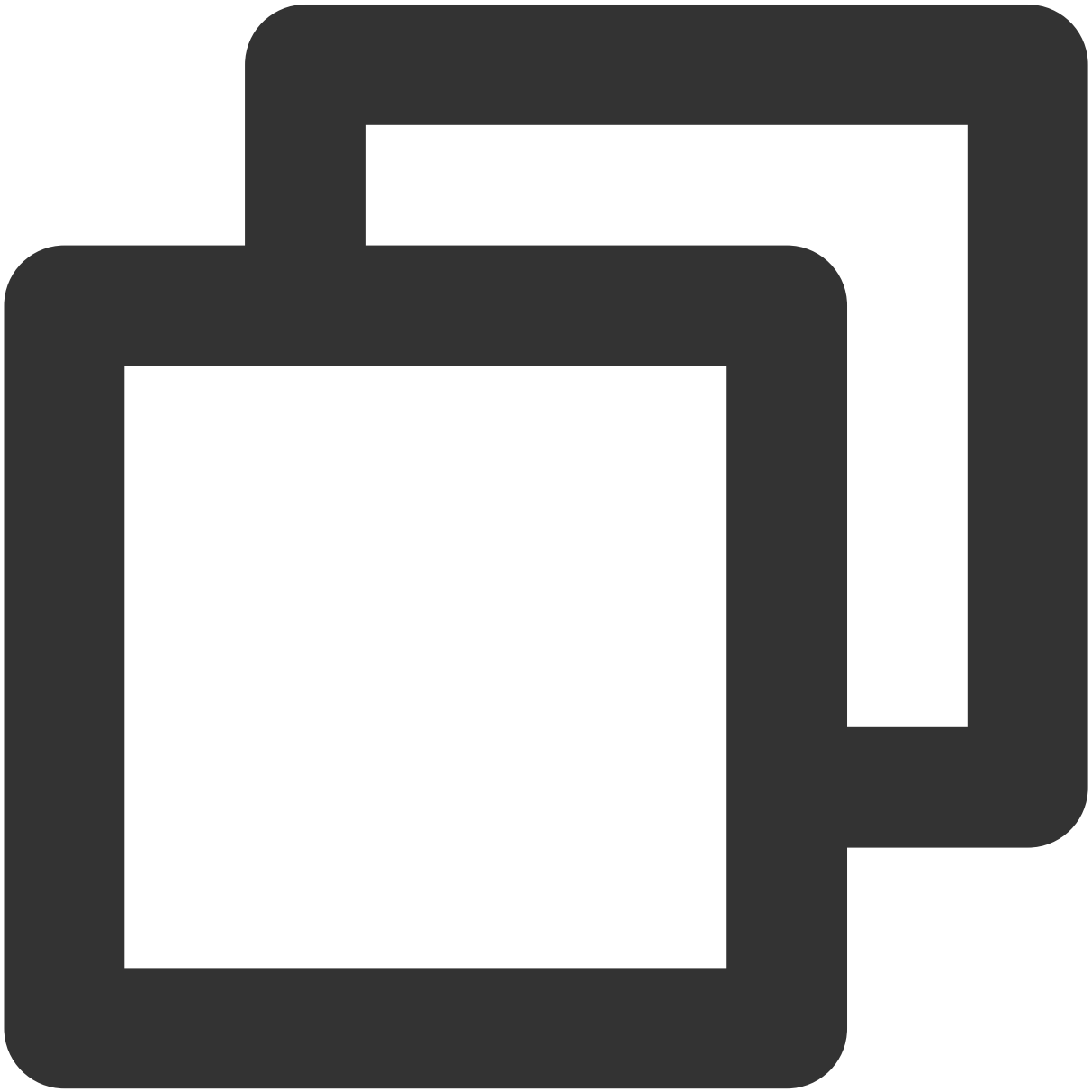
```
wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_expo
```

The file directory is as follows:

```
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ll
total 18080
-rw-r--r-- 1 3434 3434    11357 Aug  6  2021 LICENSE
-rwxr-xr-x 1 3434 3434 18494215 Aug  6  2021 node_exporter
-rw-r--r-- 1 3434 3434      463 Aug  6  2021 NOTICE
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ./node_exporter
```

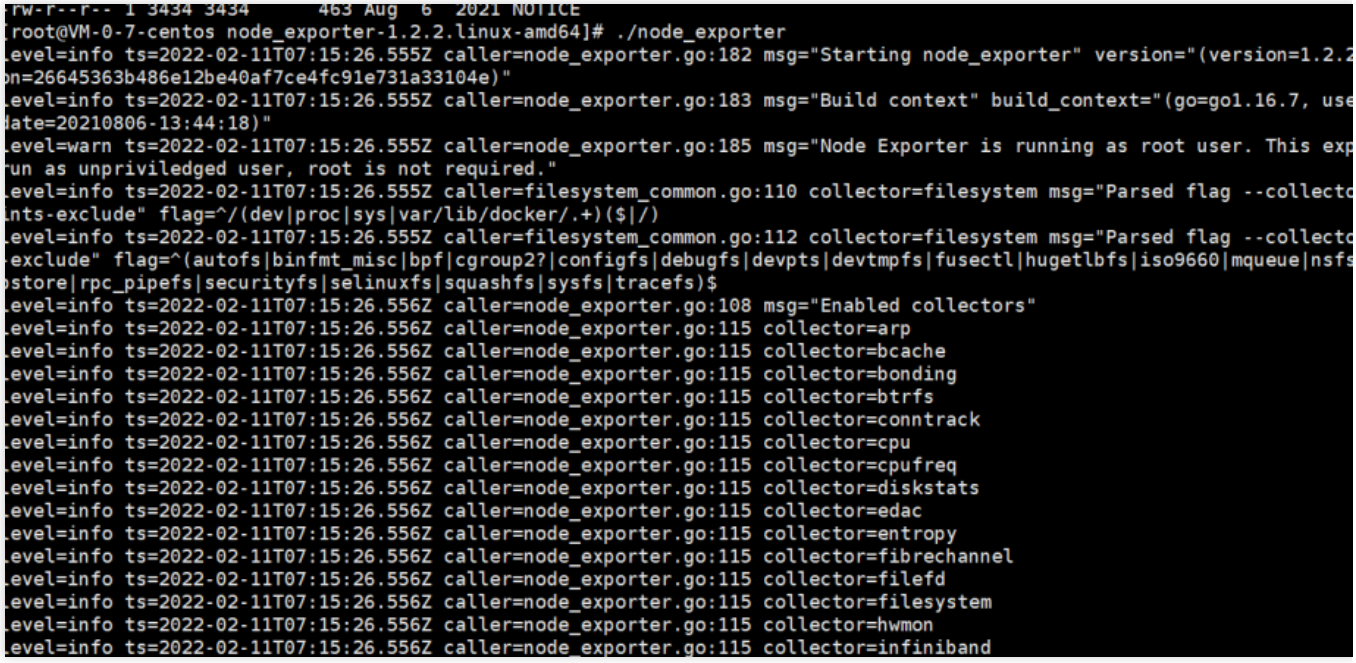
2. Run Node Exporter to collect basic monitoring data.

2.1 Go to the target folder and run Node Exporter.

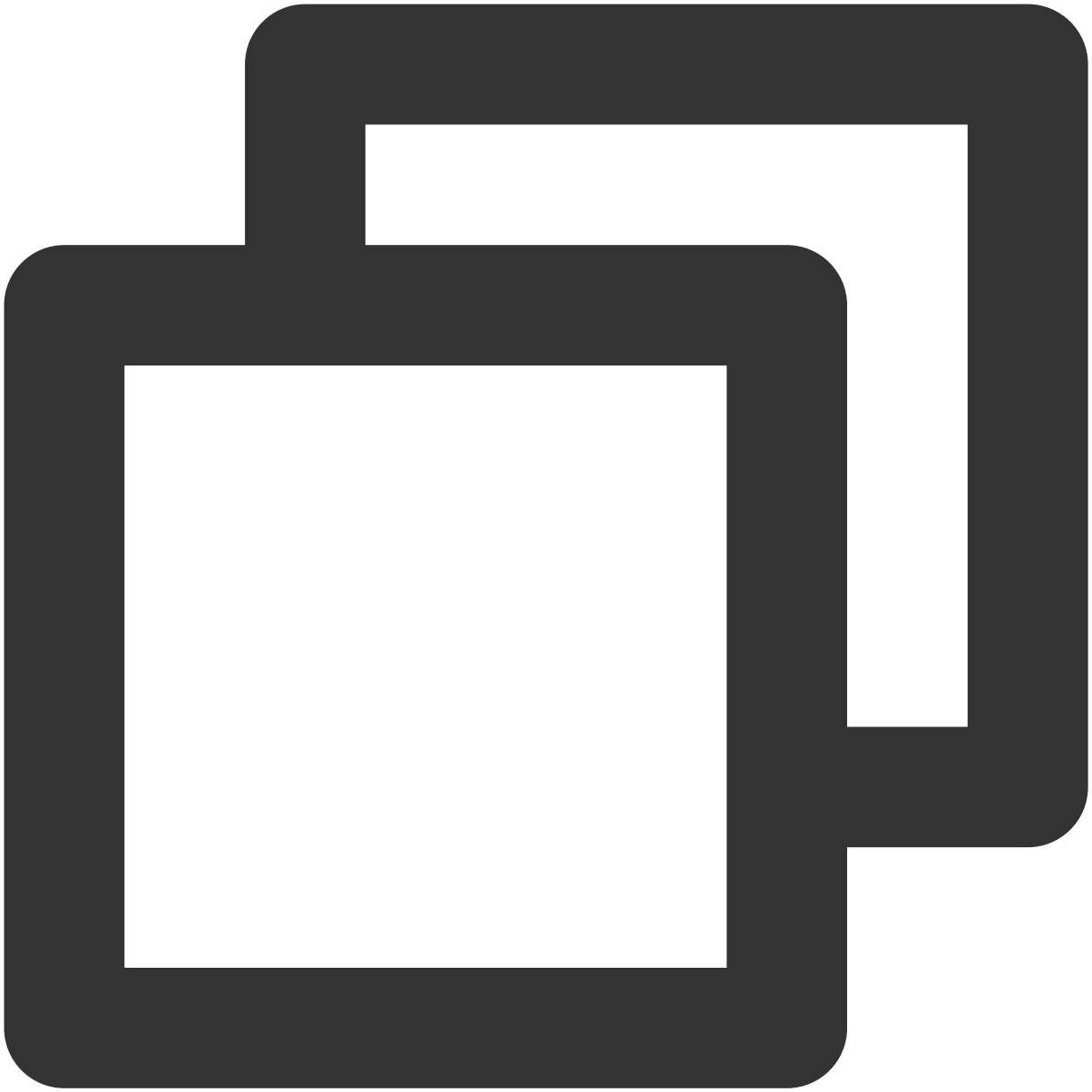


```
cd node_exporter-1.3.1.linux-amd64
./node_exporter
```

If the following result is displayed, basic monitoring data has been collected successfully.

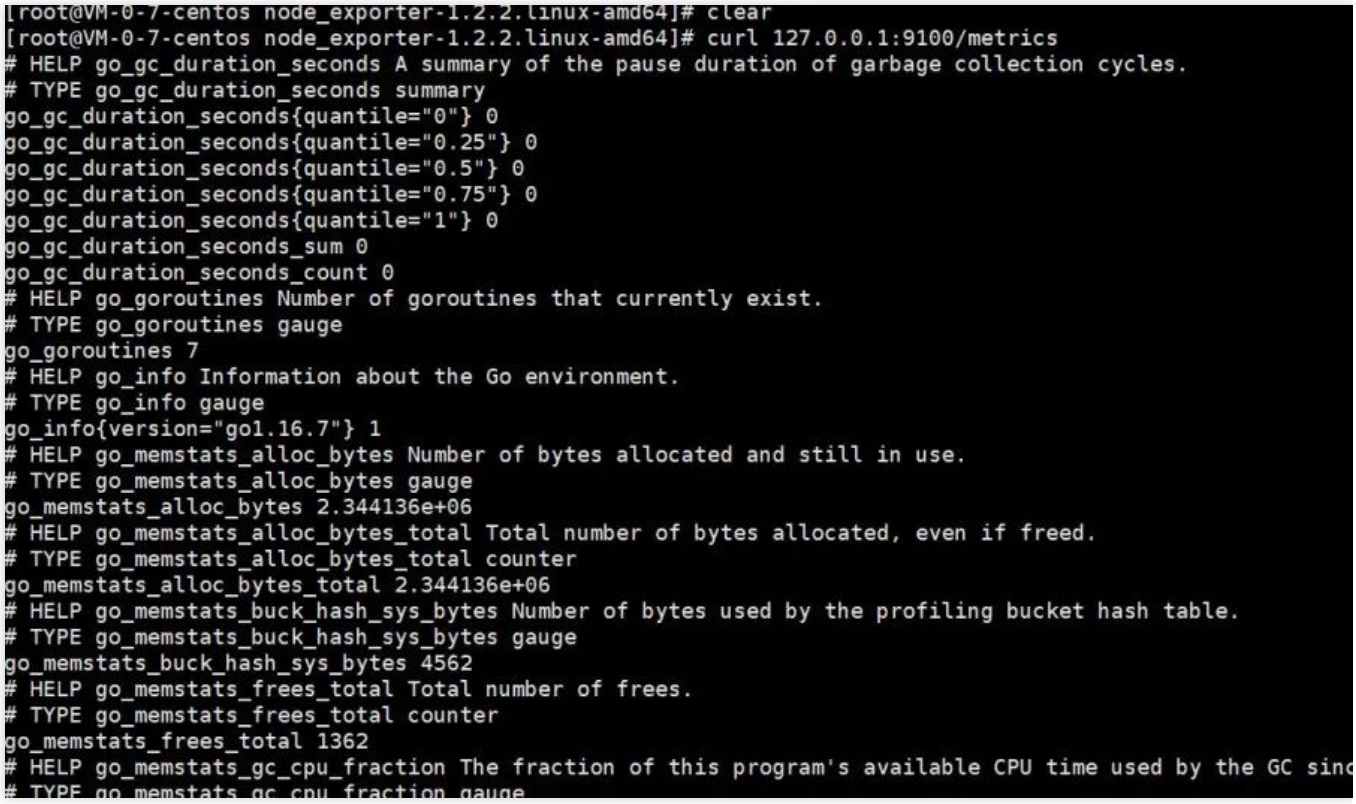


2.2 Run the following command to expose the basic monitoring data to port 9100:



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

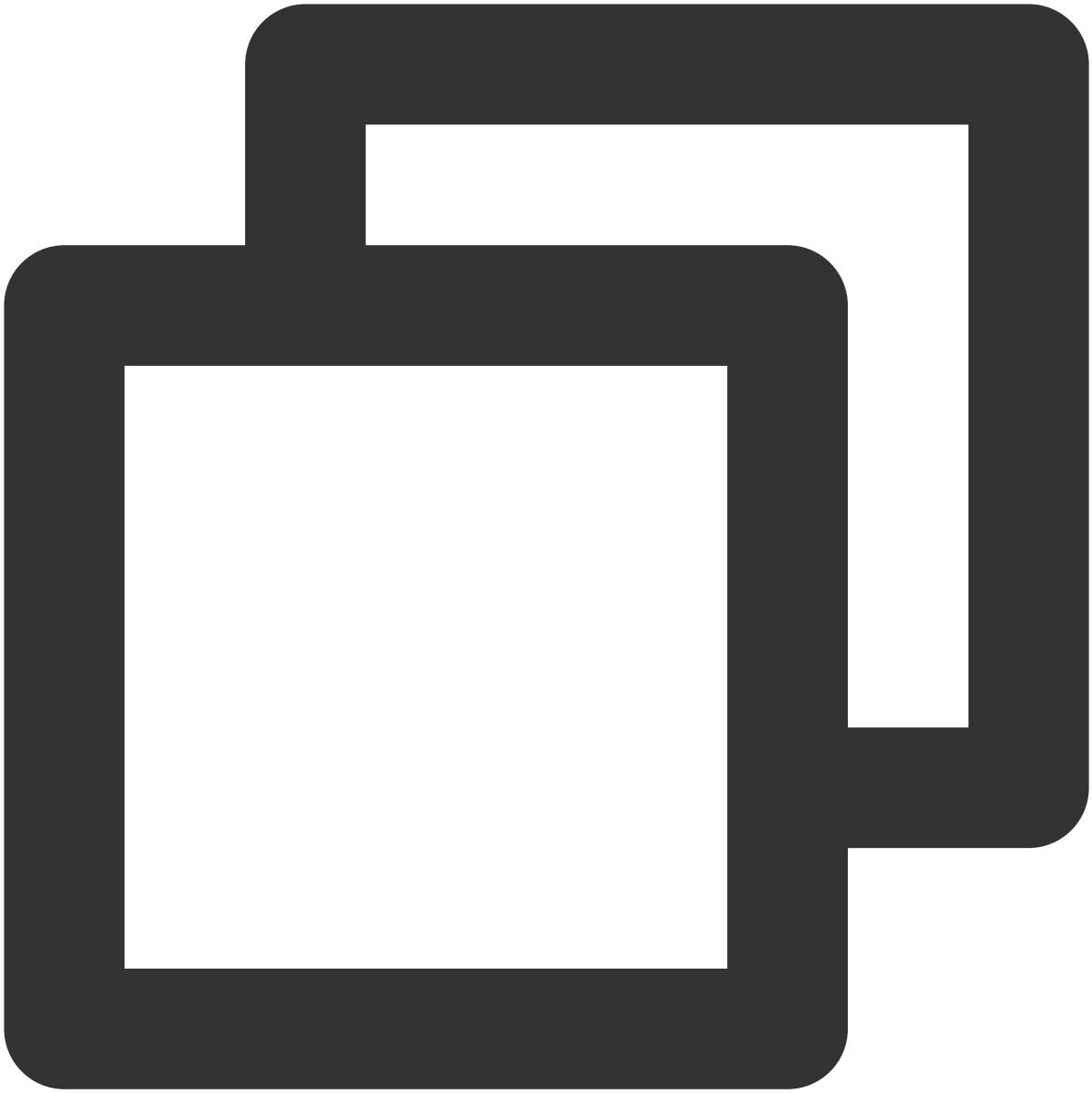
You can see the following metric monitoring data that is exposed after the command is executed.



3. Add a scrape task.

Log in to the TMP console, select **Integration Center** > **CVM**, and configure the information in **Task Configuration** as prompted.

Below is a sample configuration of a scrape task:



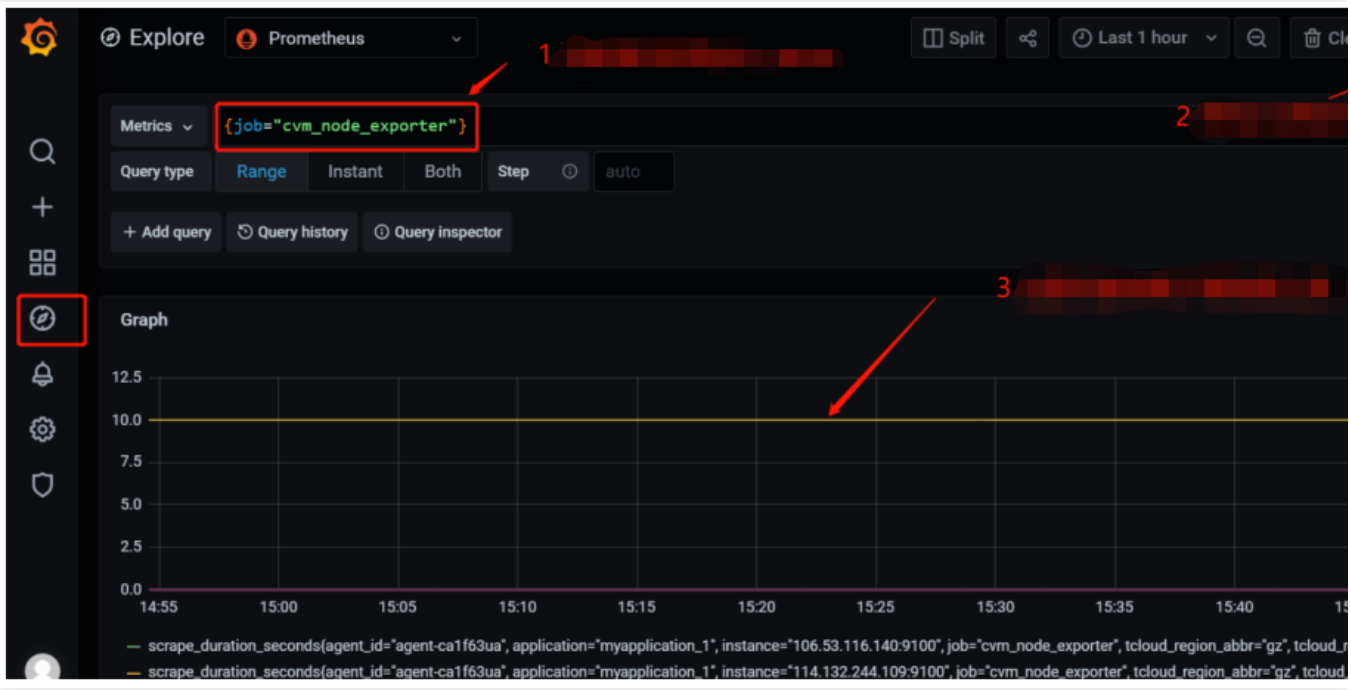
```
job_name: example-job-name
metrics_path: /metrics
cvm_sd_configs:
- region: ap-guangzhou
  ports:
  - 9100
  filters:
  - name: tag:Sample tag key
    values:
    - Sample tag value
relabel_configs:
```

```
- 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
```

4. Check whether data is reported successfully.

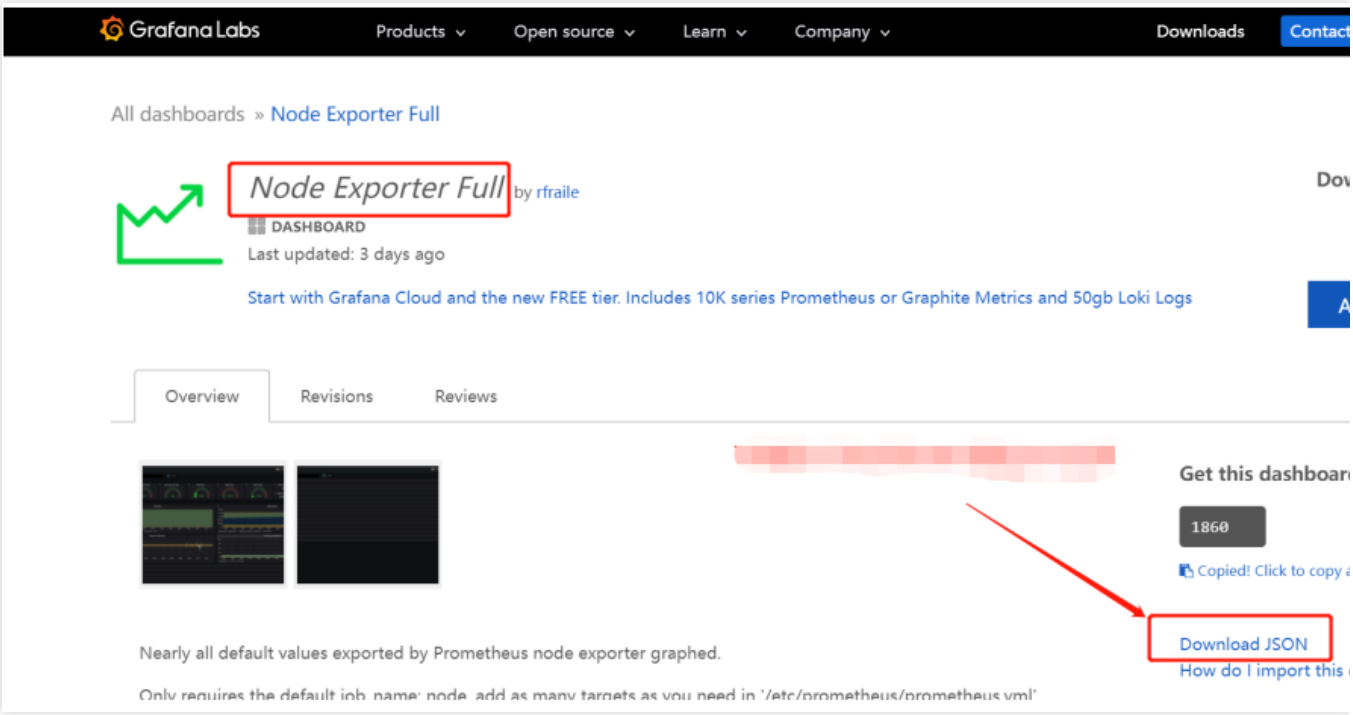
Log in to the TMP console and click the Grafana icon to enter Grafana.

Search for `{job="cvm_node_exporter"}` in **Explore** to see whether there is data, and if so, data is reported successfully.

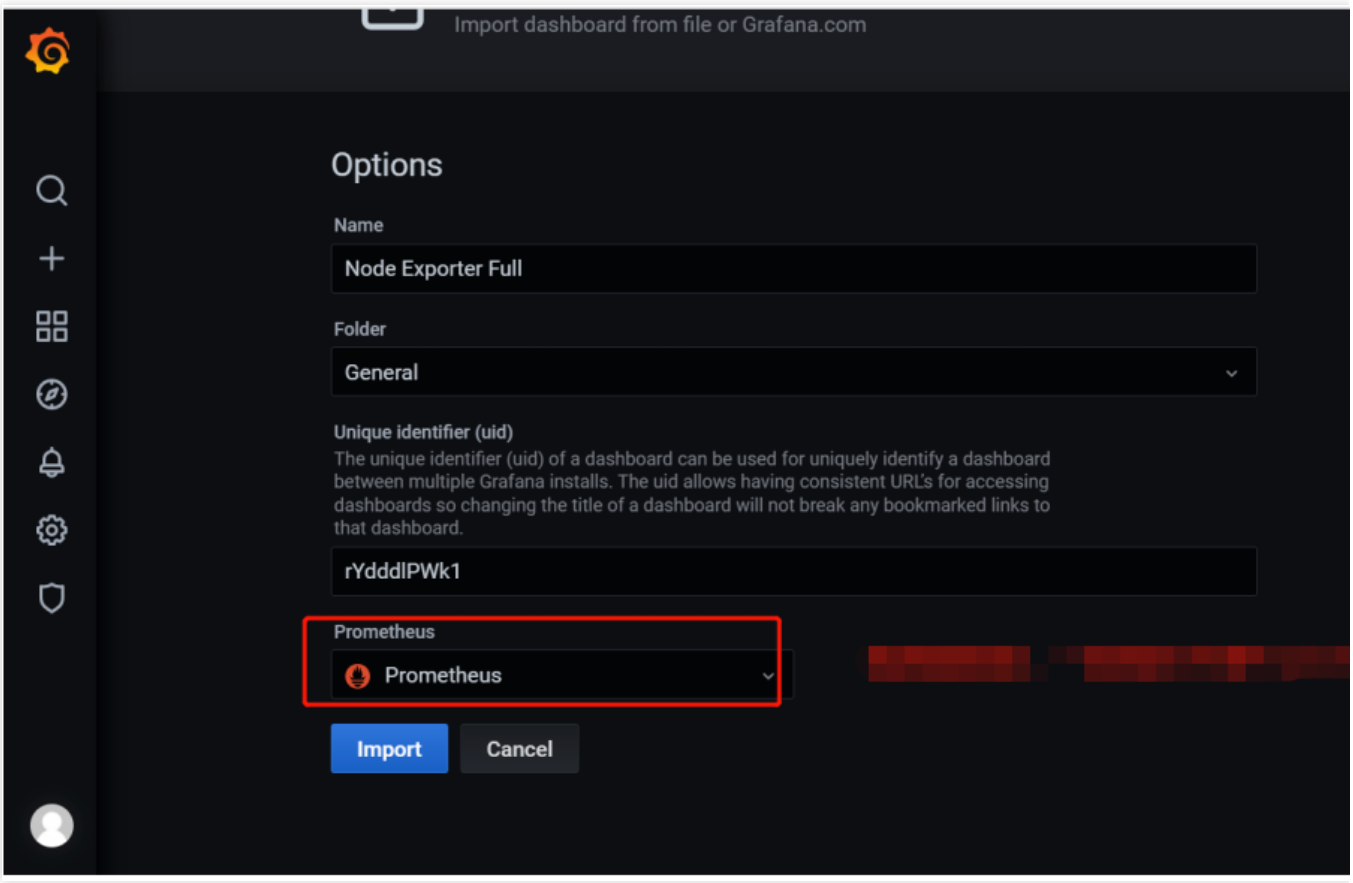


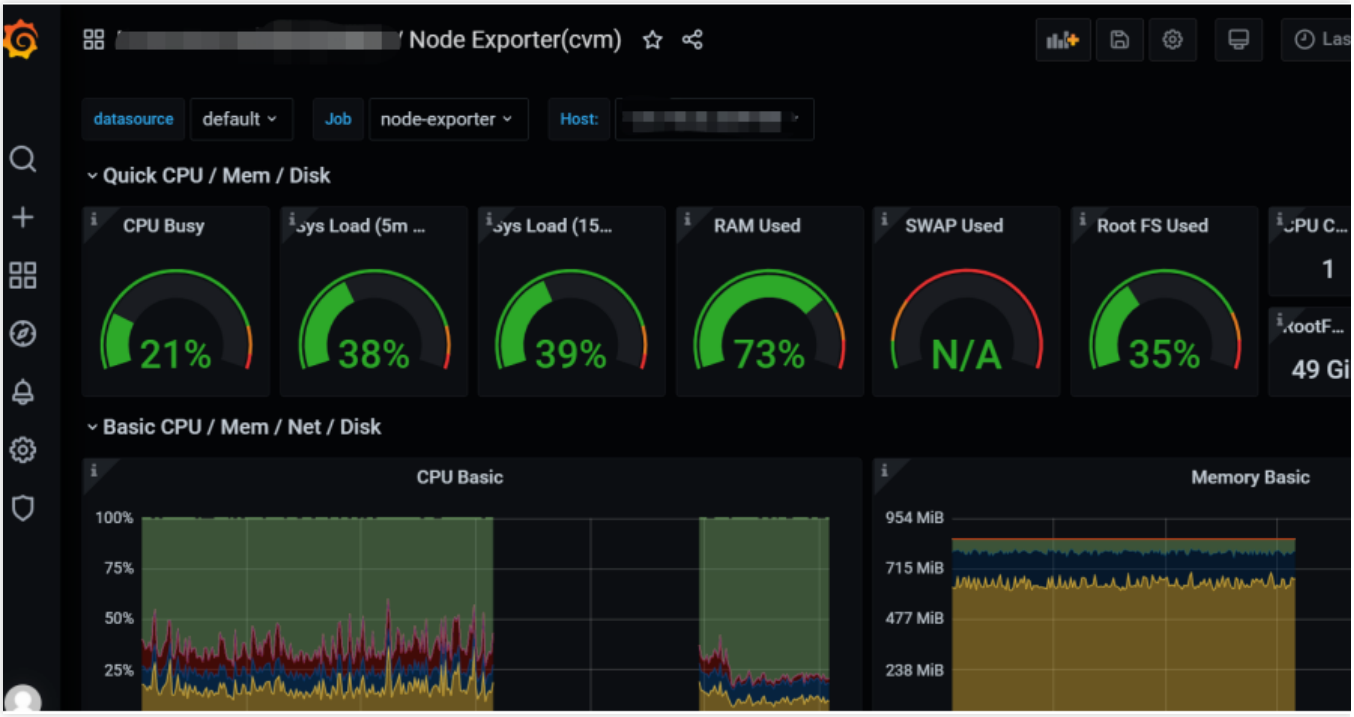
5. Configure the dashboard page: Every product has some existing JSON files that can be directly imported into the dashboard.

5.1 **Download a dashboard file**: Go to the **Dashboard** page, search for `node_exporter`, and select the latest dashboard for download.



5.2 **Import a JSON file into the dashboard**: Log in to the TMP console, select **Basic Info** > **Grafana Address** to enter Grafana. In the Grafana console, select **Create** > **Import** and upload the dashboard file in **Upload JSON file**.





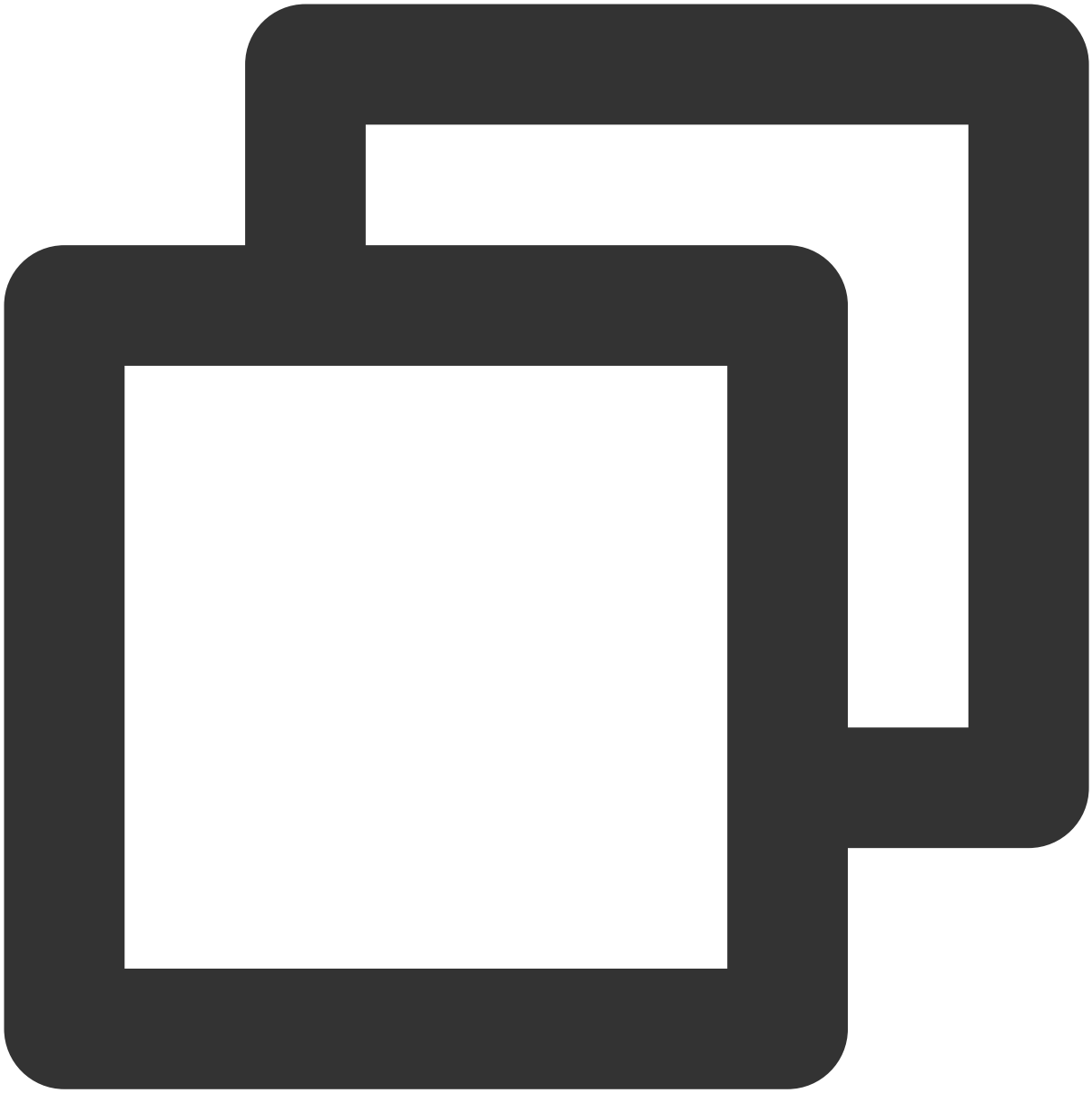
Integrating CVM Metrics at the Business Layer

Prometheus provides four metric types for different monitoring scenarios: Counter, Gauge, Histogram, and Summary. The Prometheus community provides SDKs for multiple programing languages, which are basically similar in usage and mainly differ in the syntax. The following uses Go as an example to describe how to report custom monitoring metrics. For detailed directions of other metric types, see Custom Monitoring.

Counter

A metric in Counter type increases monotonically and will be reset after service restart. You can use counters to monitor the numbers of requests, exceptions, user logins, orders, etc.

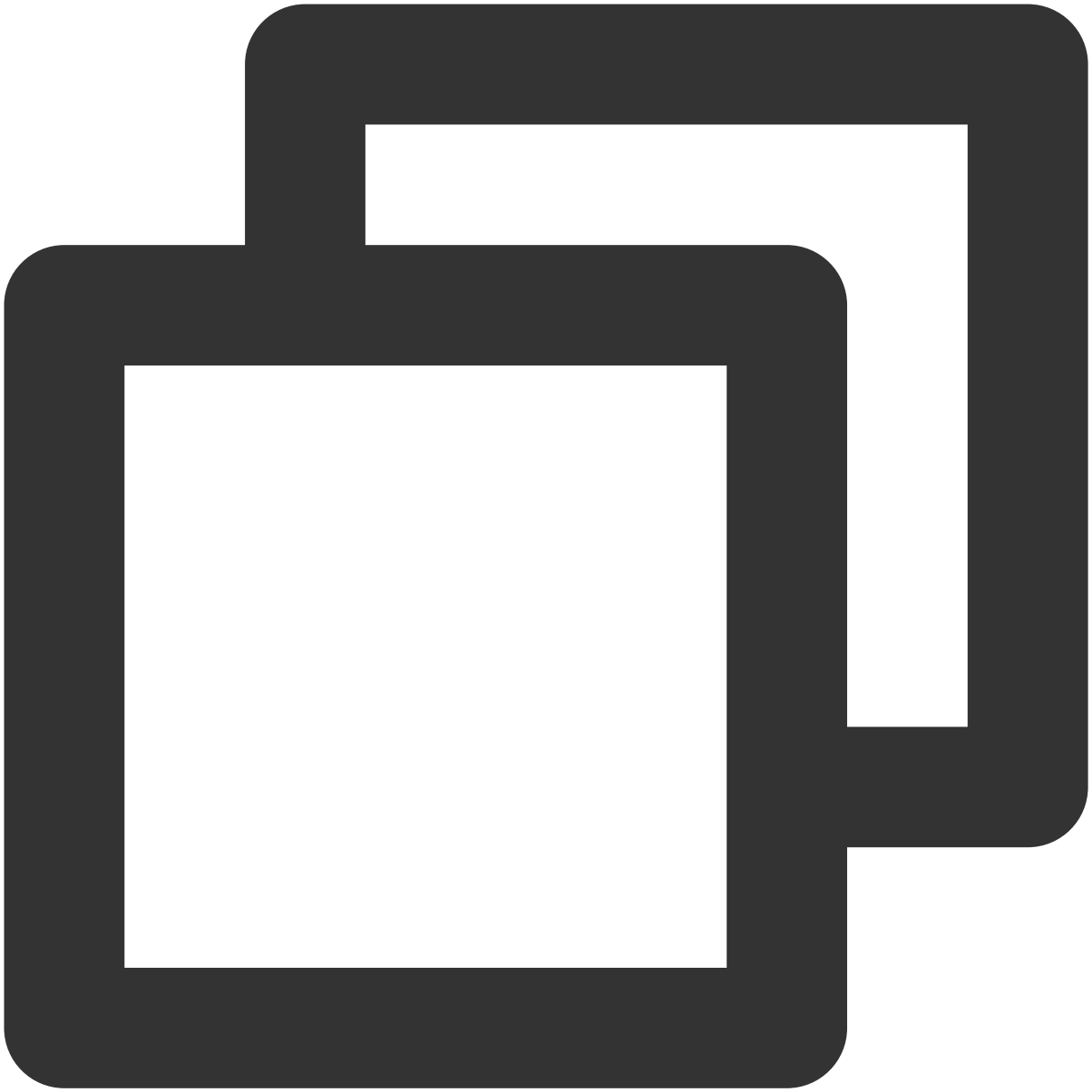
1. You can use a counter to monitor the number of orders as follows:



```
package order

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

// Define the counter object to be monitored
var (
 opsProcessed = promauto.NewCounterVec(prometheus.CounterOpts{
     Name: "order_service_processed_orders_total",
```

```
        Help: "The total number of processed orders",
    }, []string{"status"}) // Processing status
)

// Process the order
func makeOrder() {
    opsProcessed.WithLabelValues("success").Inc() // Success
    // opsProcessed.WithLabelValues("fail").Inc() // Failure

    // Order placement business logic
}
```

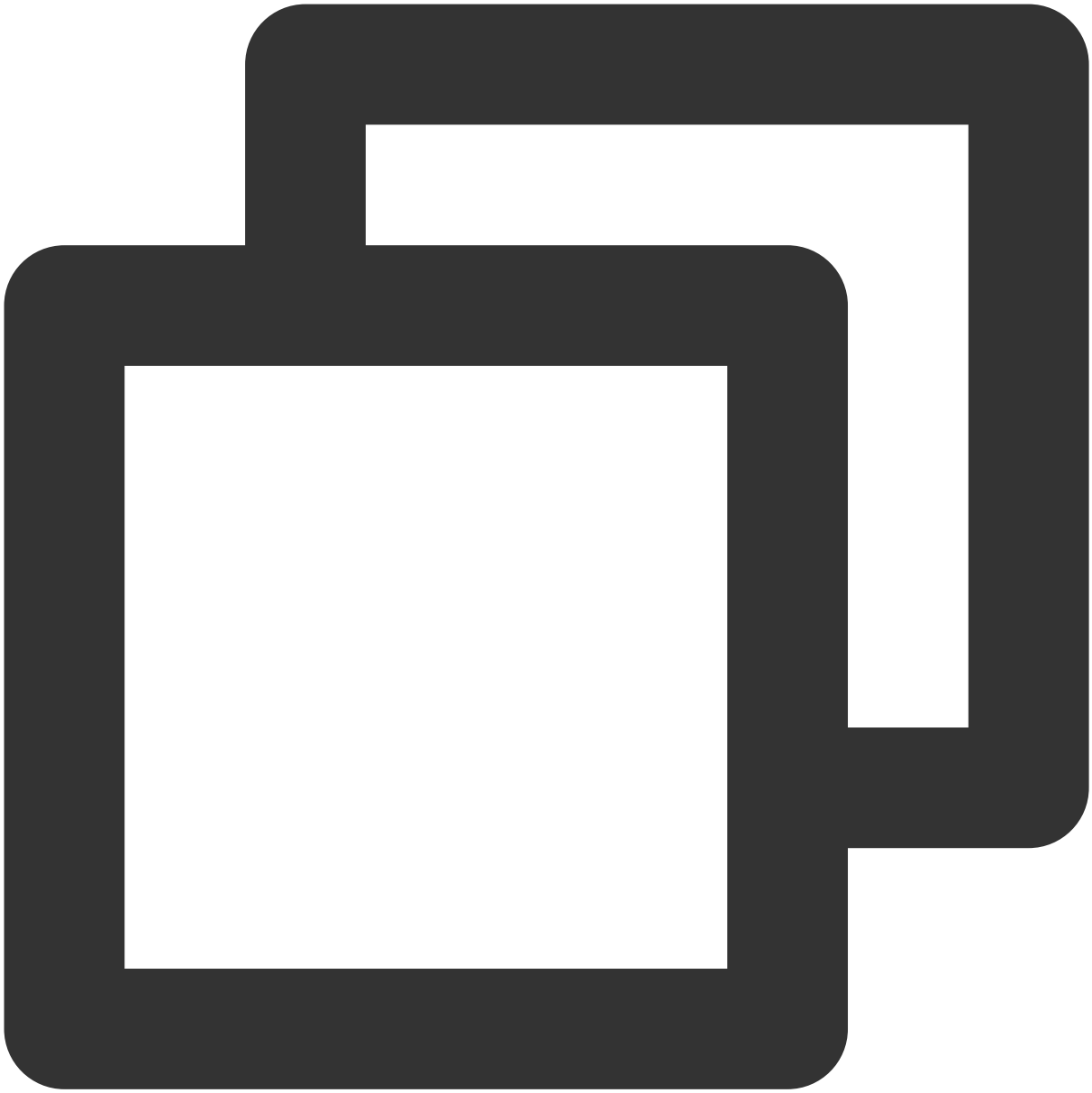
For example, you can use the `rate()` function to get the order increase rate:



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

2. Expose Prometheus metrics:

Use `promhttp.Handler()` to expose the metric tracking data to the HTTP service.

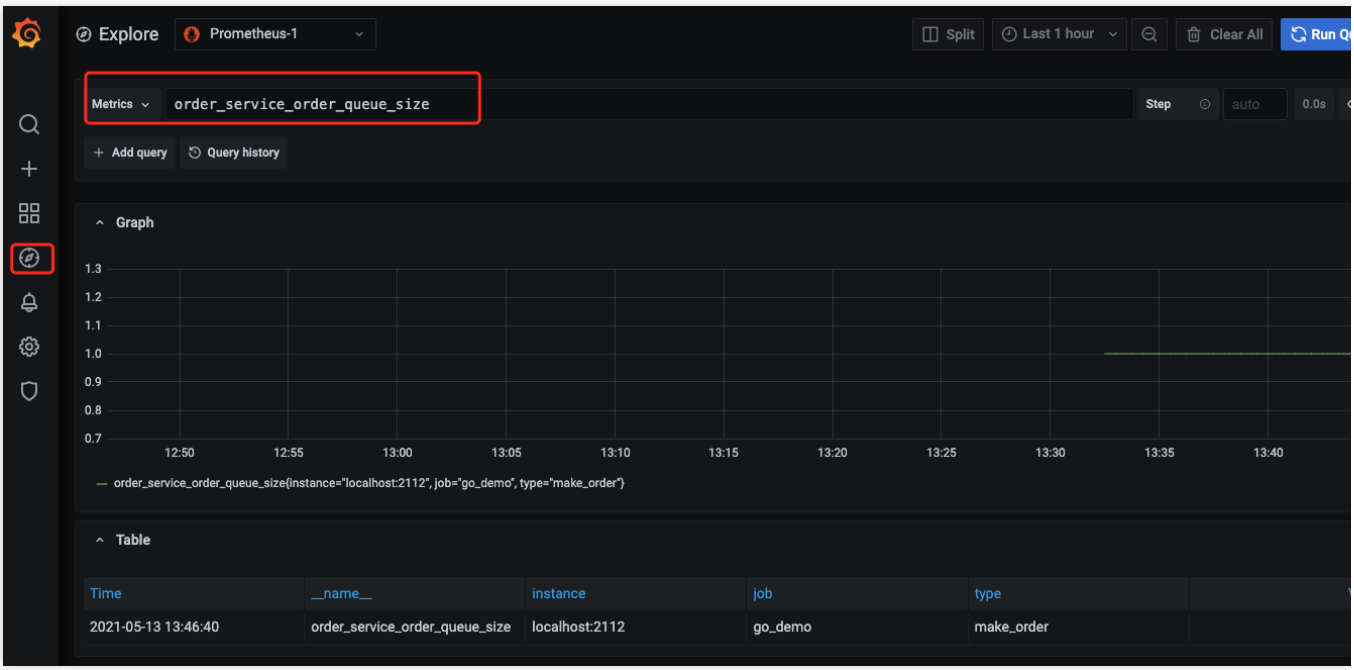


```
package main

import (
 "net/http"

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

func main() {
     // Business code
```

```
        // Expose Prometheus metrics in the HTTP service
        http.Handle("/metrics", promhttp.Handler())

        // Business code
    }
```

3. Collect data:

After the tracking of custom metrics for your business is completed and the application is released, you can use Prometheus to collect the monitoring metric data. After the collection is completed, wait a few minutes and then you can view the business metric monitoring data in Grafana integrated in TMP.



TKE Monitoring

Last updated：2024-08-07 22:08:14

Background

As we all know, Prometheus is the best monitoring tool for container scenarios. However, self-building Prometheus is too expensive for small and medium-sized enterprises with limited Ops manpower, and it is likely to hit performance bottlenecks for large enterprises with rapid business development. Therefore, using Prometheus managed on the cloud has become the first choice for more and more cloud companies. For how to use the managed Prometheus to monitor TKE, see Tencent Kubernetes Engine (TKE).

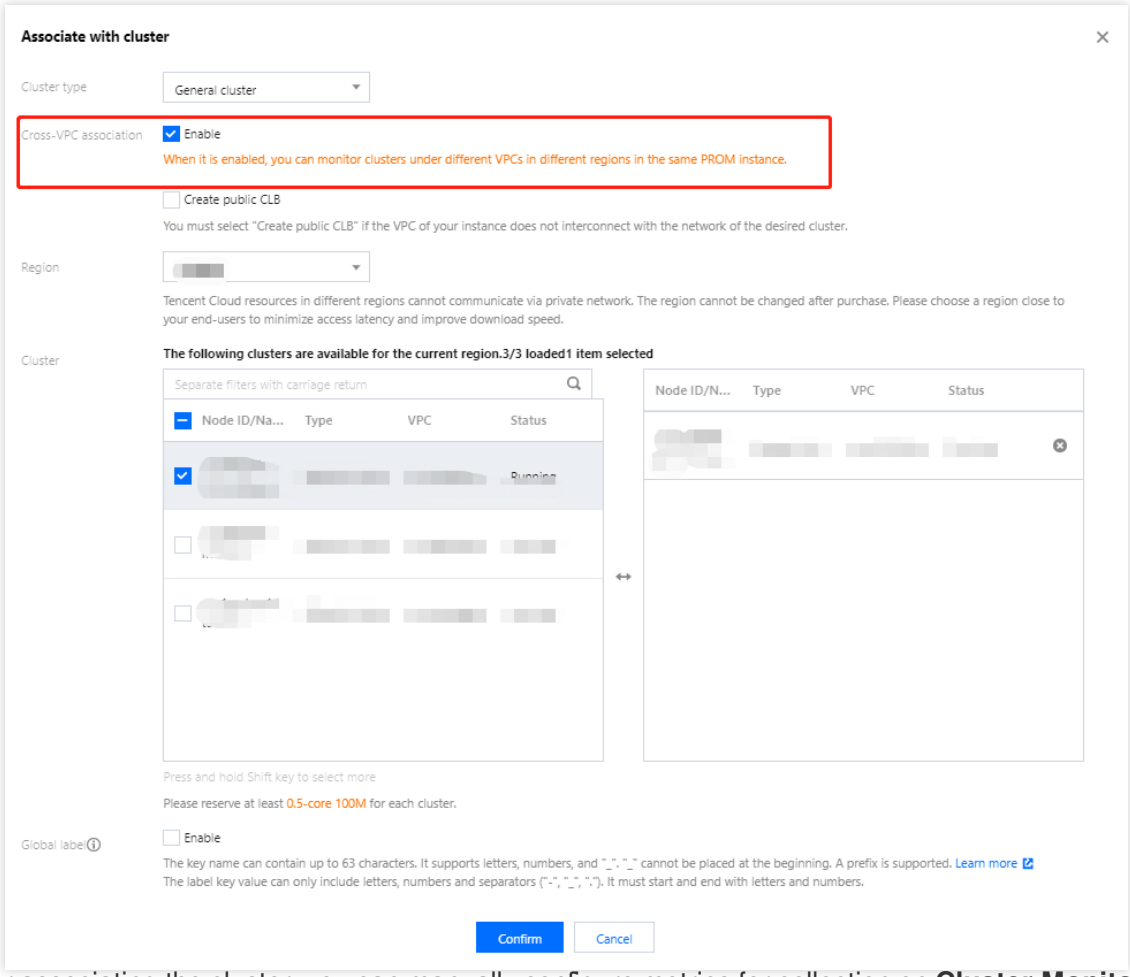
Directions

Step 1. Purchasing an instance

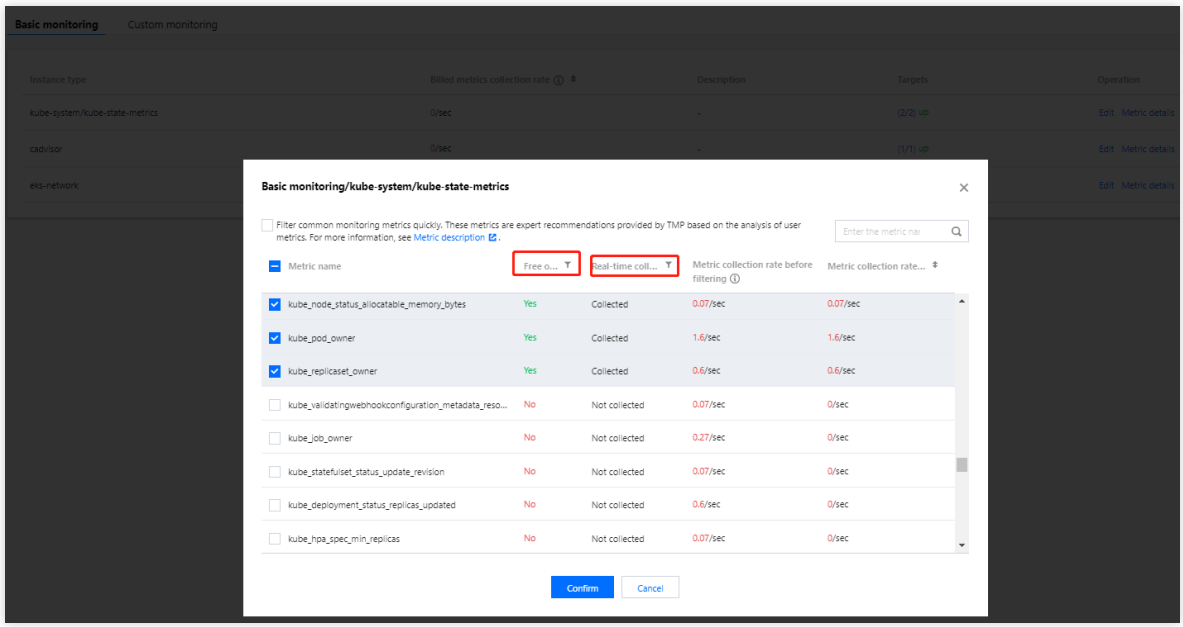
1. Log in to the TMP console.
2. Click **Create**, select the purchase region, storage duration and select the Grafana instance to be associated based on your needs. If there is no Grafana instance, see Creating Instance to create one. You need to create an instance and complete the purchase.
3. After completing the configuration, click **Buy Now**. For more information on billing rules, see Pay-as-You-Go Description.

Step 2. Integrate with TKE

1. After creating the instance, click the **ID**/**Name** of the target instance in the instance list to enter the instance details page.
2. On the left sidebar, click **Integrate with TKE** > **Associate Cluster**.
3. Select the cluster that needs to be associated in the pop-up window. A total of 4 types of clusters are supported: standard cluster, elastic cluster, registered cluster, and edge cluster. The clusters can be across VPCs. If different VPCs are not interconnected, you need to create a public network CLB instance.

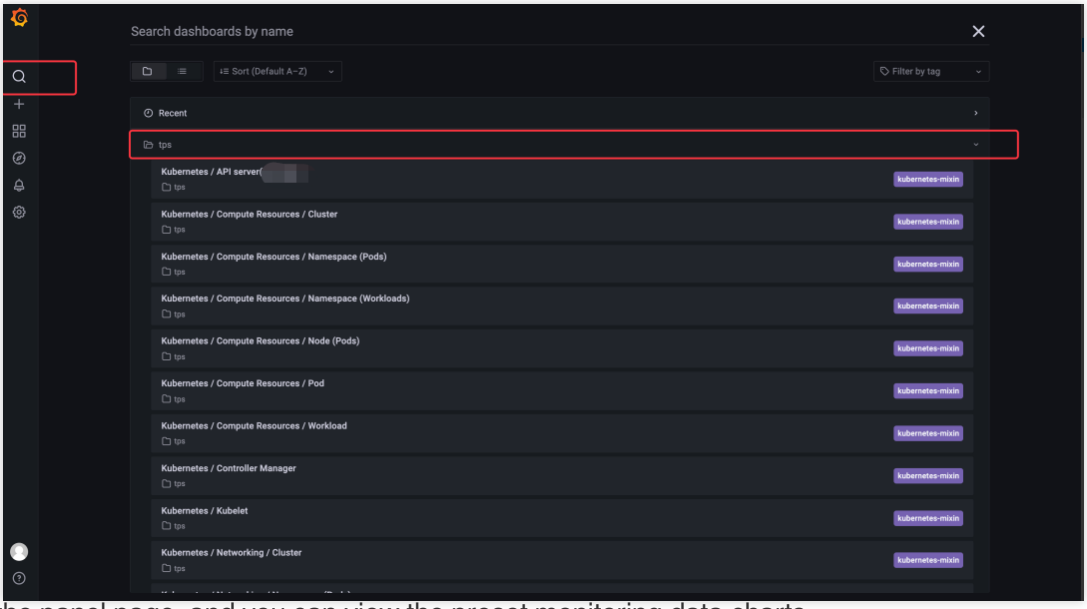


4. After associating the cluster, you can manually configure metrics for collection on **Cluster Monitoring** > **Data Collection Configuration**, view the default free basic collection metrics, and add or reduce the metrics as needed.

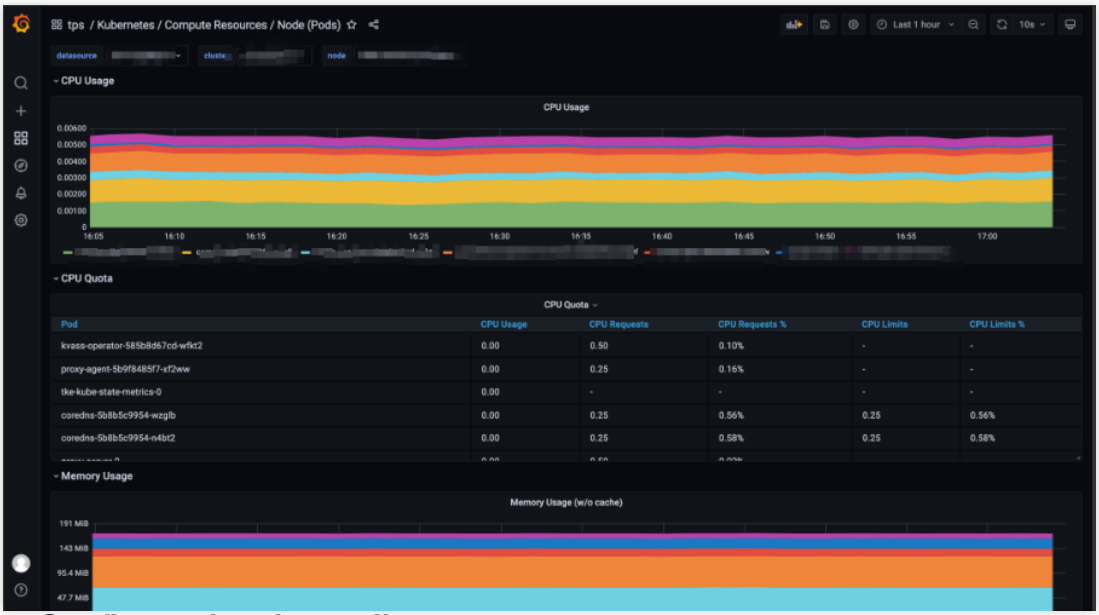


Step 3. View monitoring data in Grafana

1. Click the Grafana icon to the right of the instance in the instance list to enter the Grafana service platform.
2. In the dashboard search list, TKE-related monitoring panels are preset by default, and click a panel name.

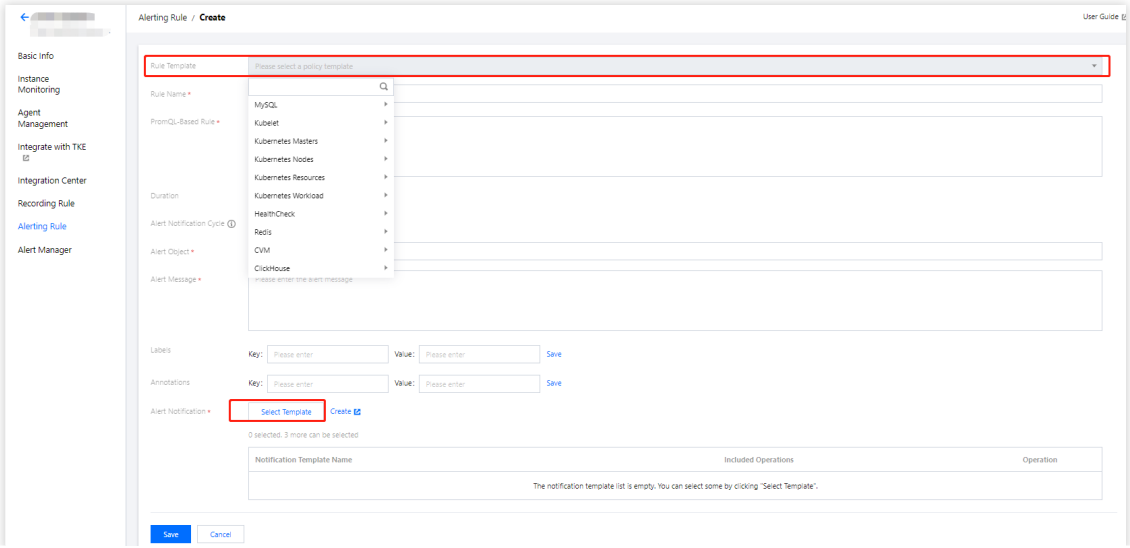


Enter the panel page, and you can view the preset monitoring data charts.



Step 4. Configure the alert policy

On the Prometheus instance details page, click **Alerting Rule**, and you can select a preset template type without manual configuration. For alert notifications, you can select existing notification templates on the TCOP to quickly configure alerts.



Enabling Public Network Access for TKE Serverless Cluster

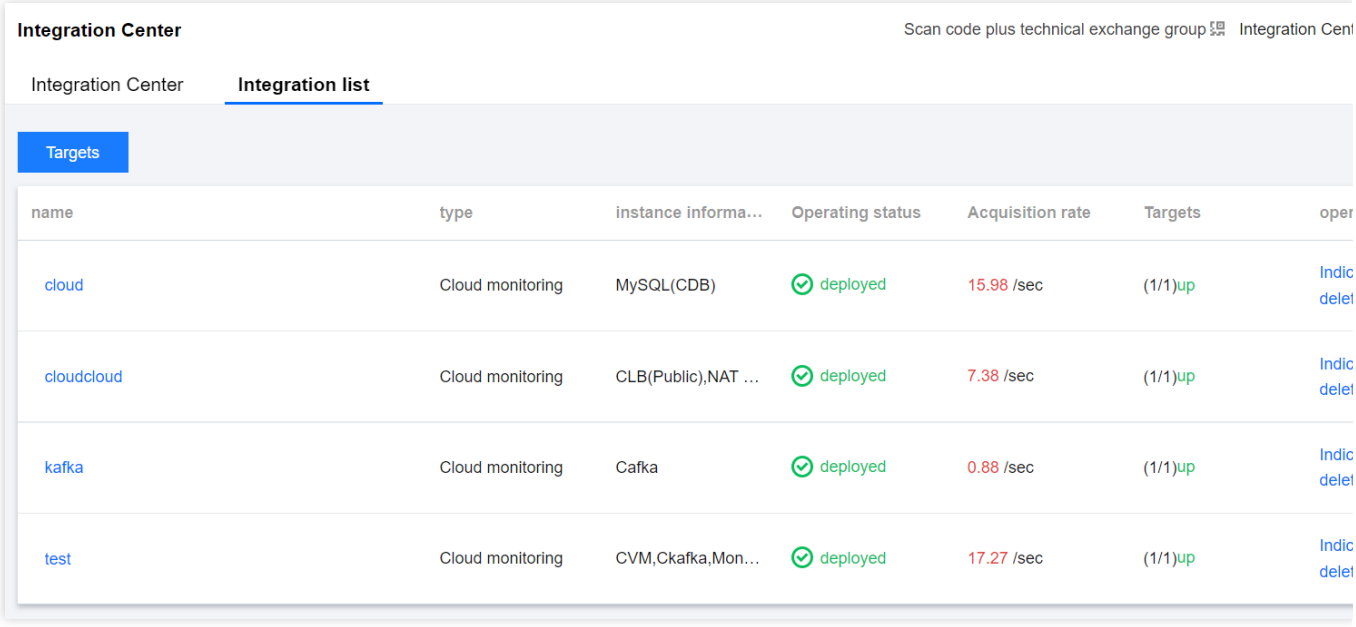
Last updated：2024-08-07 22:08:22

Overview

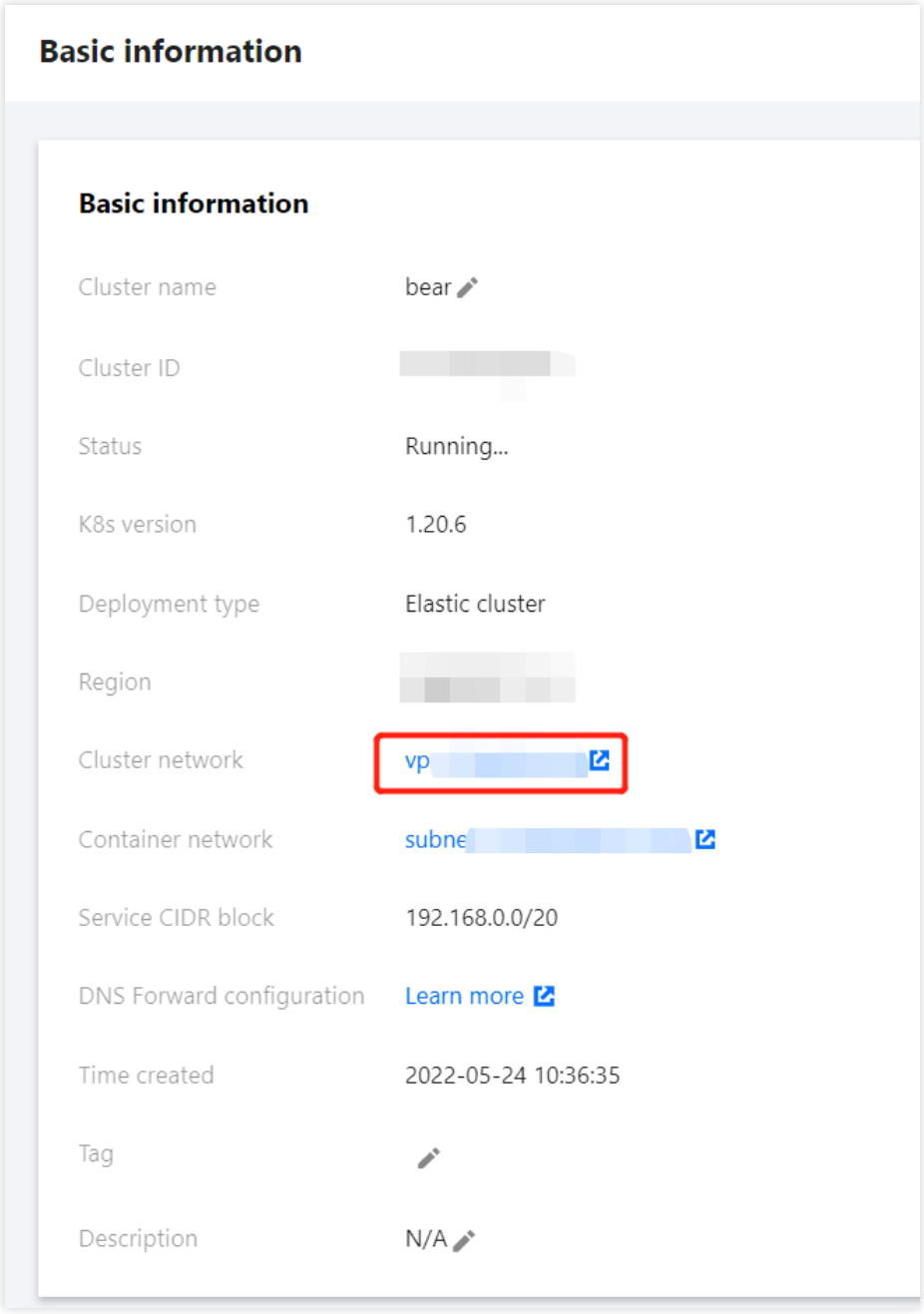
TMP is integrated with CM. When creating an integration, if you select COS, you need to enable public network access for the TKE Serverless cluster of the target CM exporter, as COS doesn't support private network access.

Directions

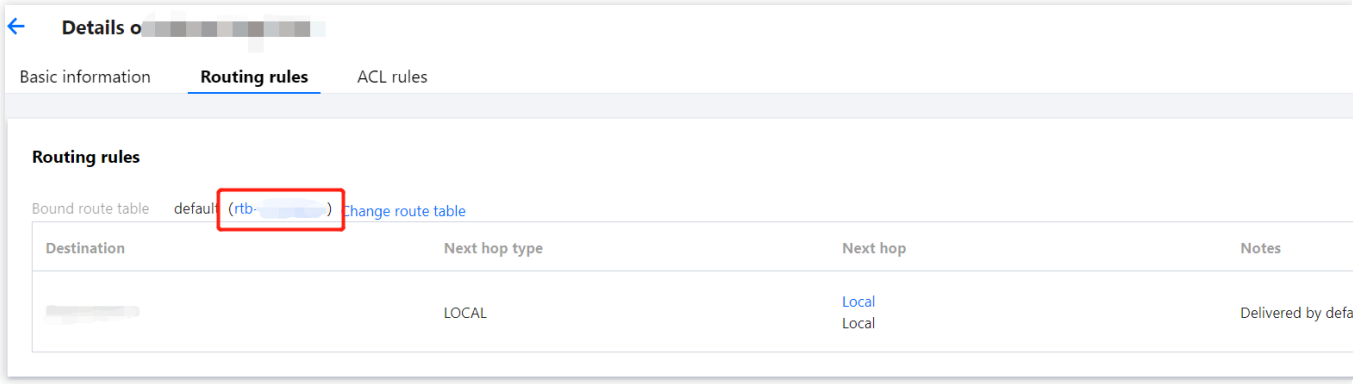
1. Log in to the TMP console.
2. Click the target instance to enter the instance management page. Then, click **Integration Center** > **Integration List**.
3. Click **Log** in the **Operation** column of the line where **Type** is **CM**.



4. On the topbar, switch to the Pod management page. Click the instance name to enter the cluster page.
5. On the **Basic Info** page, click **Container Network**.



6. On the topbar of the container network page, switch to the **Routing Policy** tab. Click the route table link (`rtb-xxx` in the list) to enter the route table page.

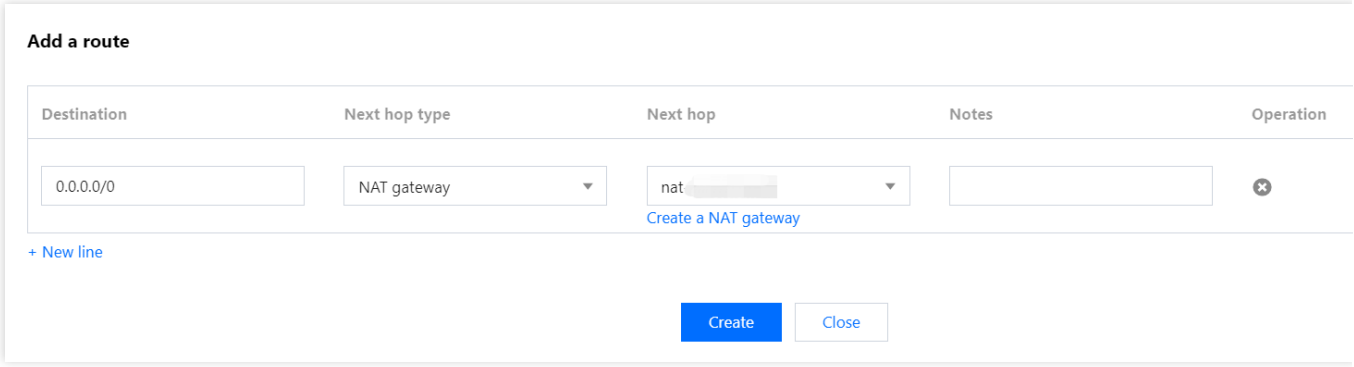


7. On the route table page, click **Create Routing Policy**.

Destination: Enter `0.0.0.0/0` .

Next Hop Type: Select **NAT Gateway**.

Next Hop: Select the target gateway. If there is no gateway, create one as instructed in Getting Started.



8. Click **Create**. After the creation is successful, public network access is enabled for TKE Serverless.

Connecting TMP to Local Grafana

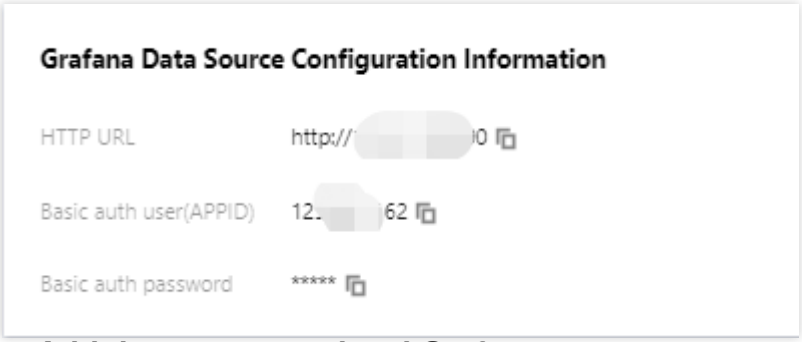
Last updated：2024-08-07 22:08:28

If you need to view the relevant data of TMP in the local Grafana system, you can use the HTTP API provided by TMP to do so. This document describes how to connect TMP data to local Grafana.

Directions

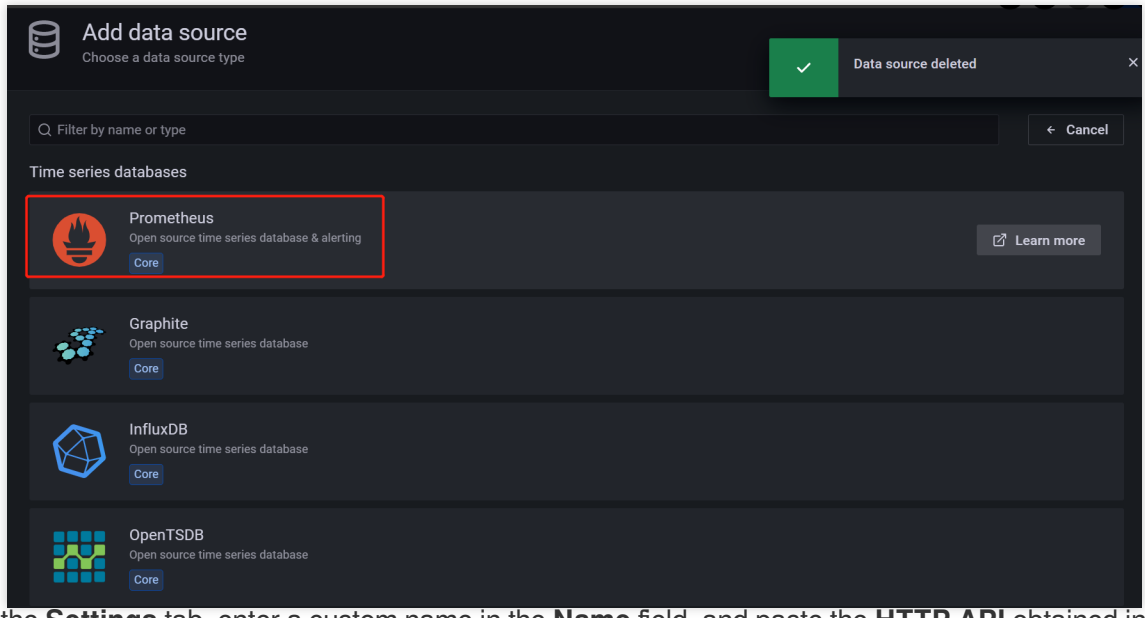
Step 1: Obtain the HTTP API provided by TMP

1. Log in to the TMP console.
2. Click the corresponding pay-as-you-go instance to enter the basic information page of TMP.
3. Get the HTTP API address in the service address module. If you need to improve the security of Grafana data read, you can obtain the authentication token of the TMP instance and fill it in as instructed in step 2.



Step 2. Add data source to local Grafana

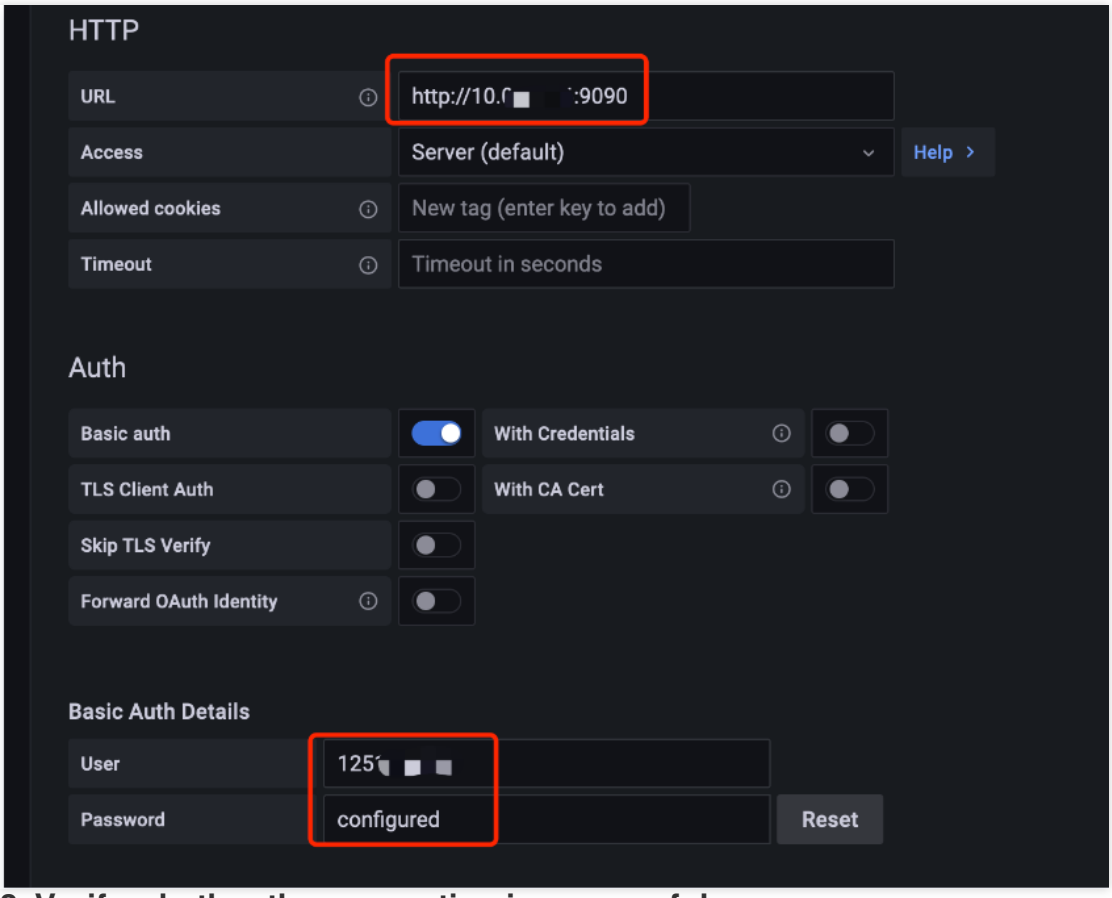
1. Log in to the local Grafana system with **admin account**.
2. Select **Configuration > Data Sources** on the left sidebar (non-admins cannot view this menu).
3. On the **Data Sources** page, click **Add data source**.
4. On the **Add data source** page, select **Prometheus**.



5. On the **Settings** tab, enter a custom name in the **Name** field, and paste the **HTTP API** obtained in step 1 in the URL field.

6. Toggle on **Basic Auth** in the **Auth** module. In the **Basic Auth Details** module, set **User** same as **Basic auth user** and **Password** as **Basic auth password** obtained in the step 1.

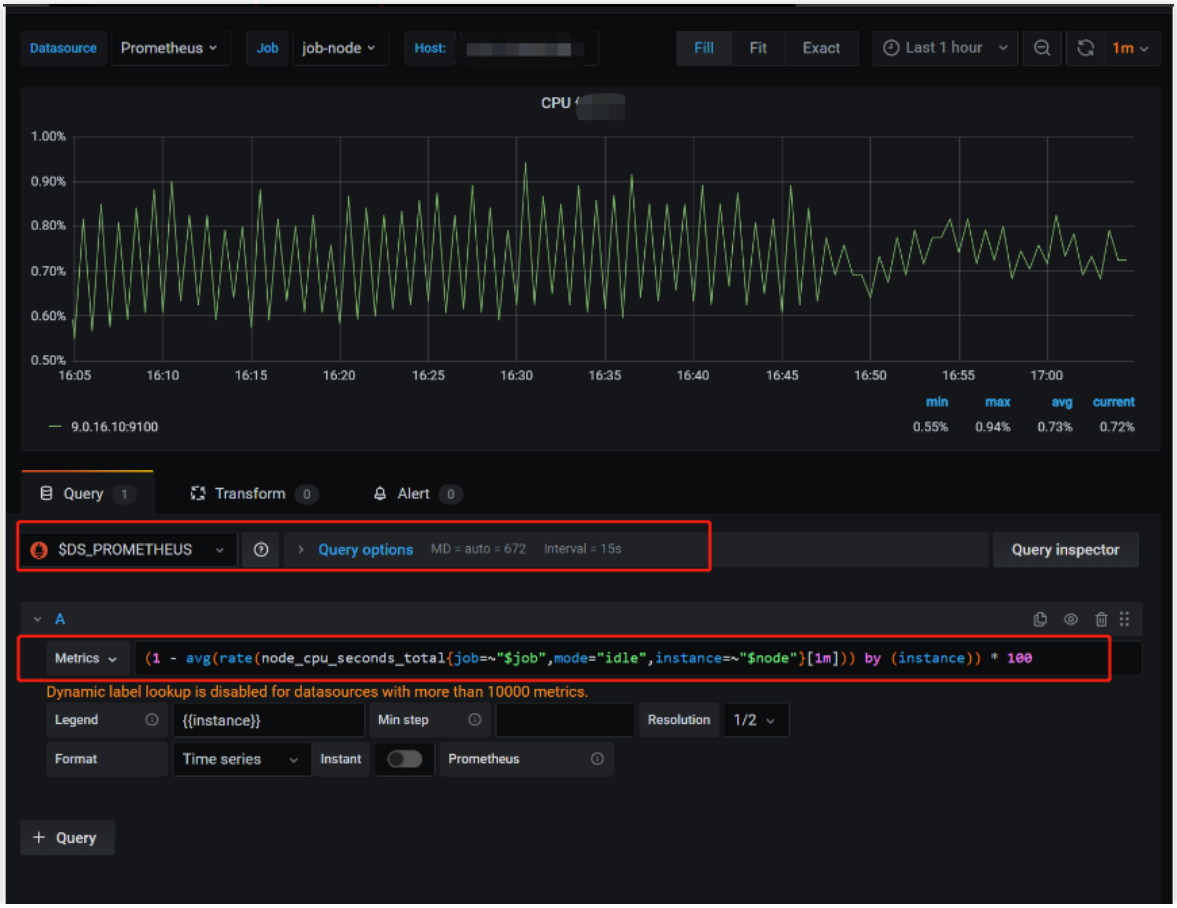
7. Click **Save & test** to complete the settings.



Step 3. Verify whether the connection is successful

Follow the steps below to verify whether TMP is successfully connected to the local Grafana:

1. Log in to your local Grafana system.
2. On the left sidebar, select **+ > Create**.
3. On the **New dashboard** page, click **Add a new panel**.
4. On the **Edit Panel** page, select the data source added in step 2 in the drop-down box on the **Query** page, enter the metric name in the **Metrics** field in the A module and press Enter.
5. If the chart of the corresponding metric can be displayed, the operation is successful. Otherwise, check whether the API address or token entered is correct, and whether the data source has TMP data.



Terraform

Terraform Overview

Last updated：2023-12-29 16:06:50

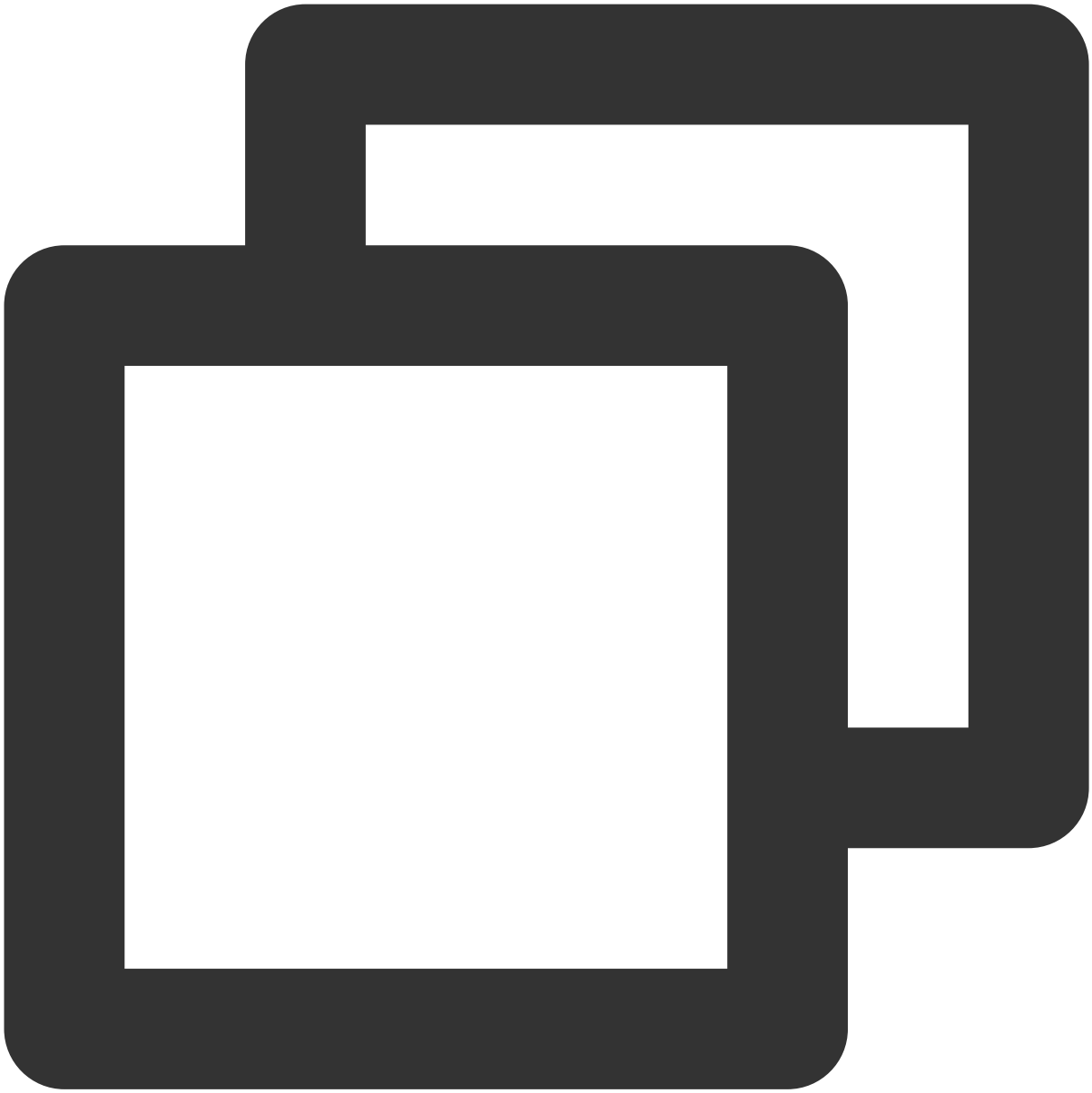
Terraform is an Infrastructure as Code (IaC) tool that allows creation of reusable, maintainable infrastructure code. You can manage and update your infrastructure using codes, thereby simplifying the deployment and management of cloud infrastructure. With abundant resource types, powerful modules, and flexible customization options, Terraform makes it easy to create and manage a variety of resources, such as CVM and CLB.

Terraform Features and Advantages

Terraform is a powerful open-source tool for automatically deploying and orchestrating cloud infrastructure.
Terraform enables you to operate, construct, define, deploy, and preview cloud infrastructure on Tencent Cloud using simple template language commands.
Terraform is remarkably user-friendly. It uses the simple HCL or JSON configuration language to define infrastructure, requiring virtually no code writing.
With variables, modules, and data source features, Terraform's configuration files are easy to use, reusable, and maintainable.
Terraform can provide plan reports at different stages, so users can get to know the actual operations executions.
Terraform tracks the actual configuration and status information of infrastructure through state files and supports rollback operations.
Note: For details about the application scenarios of Terraform, see Use Cases.

Terraform Resources

In Terraform, resources mainly fall into two categories: Resource and Data Source.
Resource
Resource enables the creation and management of new cloud infrastructure components. Through resource definitions, Terraform can be used for creating, modifying, and deleting a diverse range of resources supported by cloud service providers, such as creating a new CVM instance.

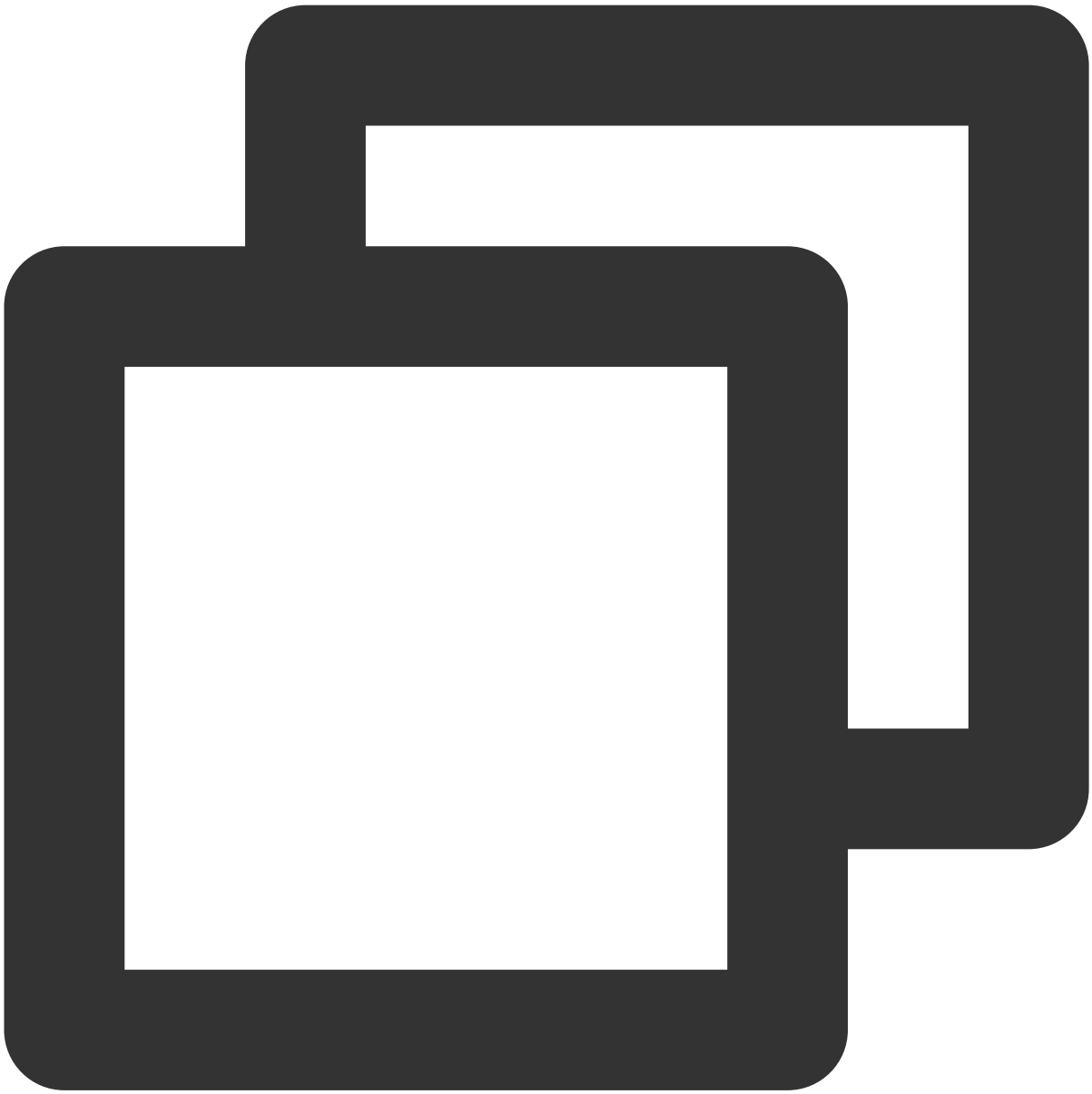


```
#Resource of cvm chc_assist_vpc.
# Reference document: https://registry.terraform.io/providers/tencentcloudstack/ten
resource "tencentcloud_cvm_chc_assist_vpc" "chc_assist_vpc" {
  chc_id = "chc-xxxxx"
}
```

Data Source

A data source is used to access and retrieve information of already existing resources and subsequently implementing it as an input or attribute within the configuration. A data source can be used to retrieve the configuration and attribute information from an existing cloud infrastructure, making it available for reference and usage within a Terraform

configuration file. The following example shows that it is used to query an instance of a created cvm chc_denied_actions:



```
#Using this data source to extract detailed information from cvm chc_denied_actions
# Reference document: https://registry.terraform.io/providers/tencentcloudstack/ten
data "tencentcloud_cvm_chc_denied_actions" "chc_denied_actions" {
  chc_ids = ["chc-xxxxx"]
}
```

For information about the Resource of the TCOP Prometheus version, see Usage with Terraform.

Terraform Tools

Terraform Command Line Interface (CLI):

Terraform CLI is a command line interface designed to create, manage, and manage infrastructure. It brings a suite of commands and options, enabling users to perform various operations, including initializing configuration, creating plans, and applying changes. Terraform CLI also incorporates several sub-commands associated with additional Terraform functionality, such as configuration validation and code formatting. Employing Terraform CLI enables users to interact with Terraform configuration files and manage the infrastructure effectively.

Terraform Provider:

Each cloud provider provides its proprietary Terraform Provider, which is a plugin used to incorporate the features of the cloud provider into Terraform. Each Terraform Provider is tasked with interacting with the APIs of the corresponding cloud service provider, defining available resources and data sources, and performing operations of creating, modifying, or deleting these resources. By deploying the appropriate provider, you can directly use the resources and features of the cloud provider within Terraform configurations.

Note:

The Terraform CLI serves as the command line interface to manage Terraform, while Terraform Provider is the plugin incorporating the features of diverse cloud providers into Terraform. These two components work collaboratively, enabling users to use Terraform to create and manage infrastructure. For more information about Terraform, see the usage document of Terraform.

Usage Advantages

Multi-Cloud Solution: Applicable to multi-cloud solutions, Terraform can deploy analogous infrastructure to Tencent Cloud and local data centers. Due to Terraform's portability, developers can manage resources from distinct cloud providers simultaneously using the same tools and configuration files. Such flexibility allows users to switch to other cloud platforms at any time without the need to modify their configuration files.

Automated Management: Terraform enables the creation of configuration file templates, defining, provisioning, and configuring resources in a repeatable and predictable manner, reducing deployment and management errors due to human factors. Using Terraform allows users to deploy the same template multiple times, creating identical development, test, and production environments. Additionally, Terraform automates the process of creating and managing infrastructure, thereby enhancing deployment speed and efficiency.

Infrastructure as Code: The Infrastructure as Code (IaC) method of Terraform allows for the management of resources through code, providing convenience in version control, collaboration, and code reuse. Using Terraform, you can save the state of your infrastructure to track system changes and share these configurations with others. This code-based management approach not only simplifies deployment but also ensures consistency, security, and maintainability.

Visual Interface: Terraform's graphical interface allows for the viewing and management of infrastructure resources. This enables users to view and understand their infrastructures and interdependencies of infrastructures more directly. Through this visual interface, users can locate and rectify potential configuration problems faster and easier.

Community Support: Developed by HashiCorp, Terraform is backed by a large community. This community comprises professionals across various industries and experience levels capable of sharing best practices and solutions, offering supports like forums and email lists.

Using TCOP Prometheus Managed Service via Terraform

Prometheus supports managing the following resources via Terraform:

Name	Description
tencentcloud_monitor_tmp_instance	Prometheus Instances
tencentcloud_monitor_tmp_alert_rule	Prometheus Alarm Rules
tencentcloud_monitor_tmp_recording_rule	Prometheus Recording Rules
tencentcloud_monitor_tmp_cvm_agent	Prometheus Cvm Agent
tencentcloud_monitor_tmp_scrape_job	Prometheus cvm_agent Capture Tasks
tencentcloud_monitor_tmp_exporter_integration	Prometheus Integration Center Resources
tencentcloud_monitor_tmp_manage_grafana_attachment	Prometheus Association with Grafana
tencentcloud_monitor_tmp_tke_cluster_agent	Prometheus Association with Containers
tencentcloud_monitor_tmp_tke_config	Prometheus Cluster Collection Configuration

Managing Prometheus Instances Using Terraform

Last updated：2023-12-29 16:08:36

Prerequisites

Installing Terraform

For detailed instructions on installing Terraform, see Installing Terraform.

Note:

The installed version of Terraform must not be below v1.6.3. You can verify the version of Terraform installed by using the command terraform --version.

Configuring Tencent Cloud Account Information

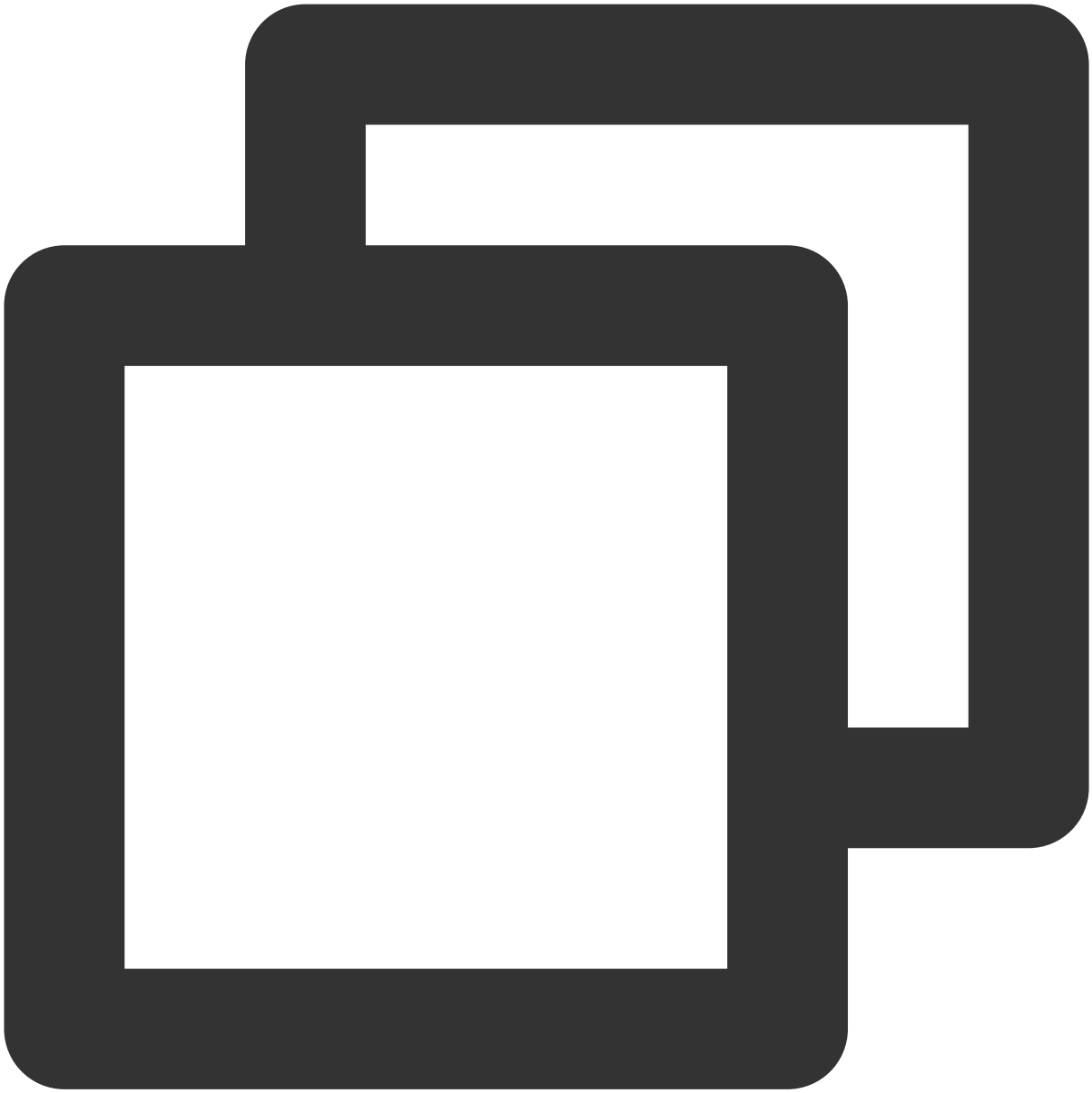
Before using Terraform for the first time, go to API Key to apply for a SecretID and SecretKey of the security credentials. If you already have valid credentials, skip this step.

1. Log in to the CAM Console and choose **Access Key > API Keys** from the left sidebar.
2. On the API Keys page, click **Create Key** to create a SecretId/SecretKey pair.

There are two methods for configuring Tencent Cloud account information:

Authentication by Static Credentials

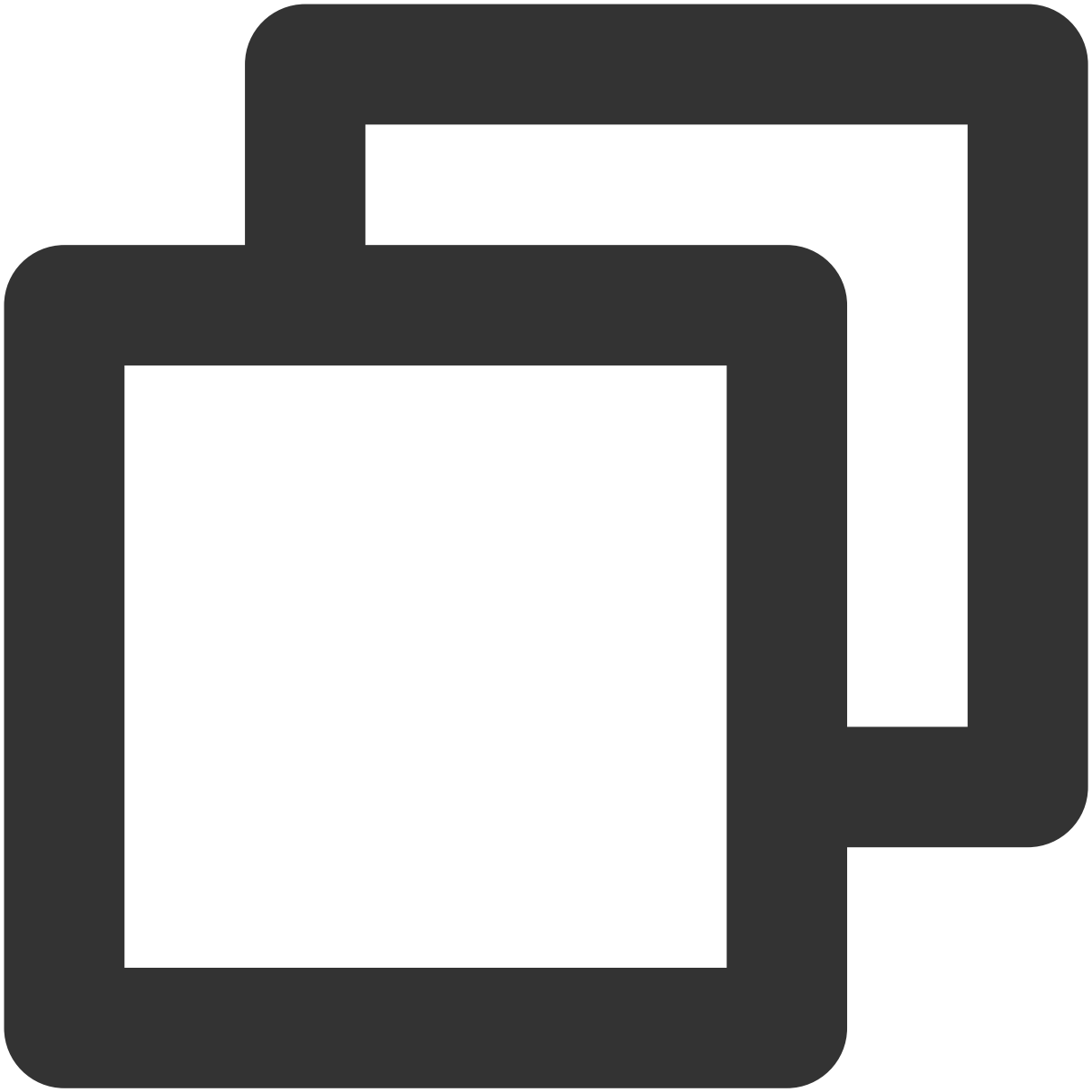
Create a `provider.tf` file in the user directory with the following content. Replace `my-secret-id` and `my-secret-key` with your SecretId and SecretKey, respectively.



```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

Authentication by Environment Variables

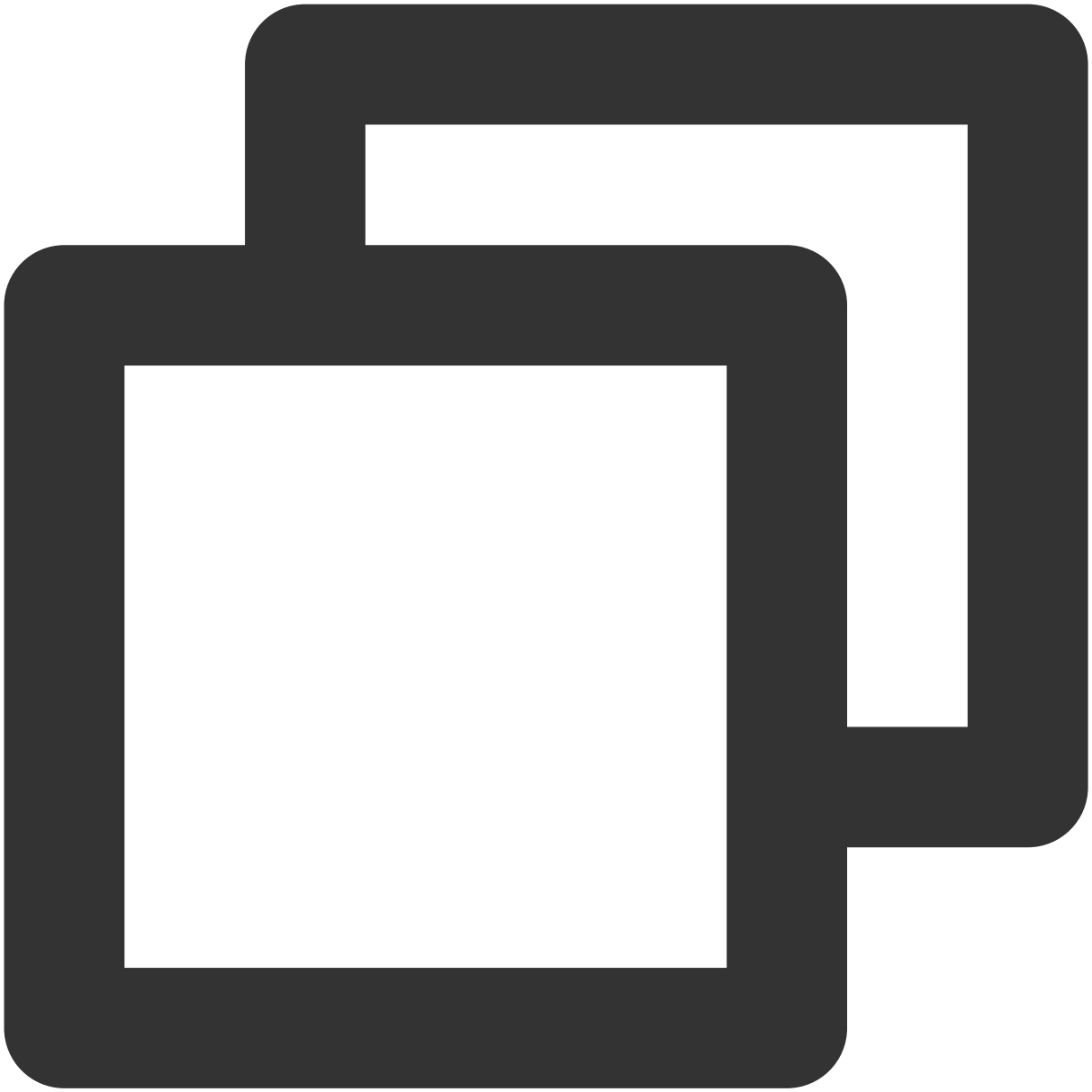
Configure the computer environment variables or cloud environment variables by running the following command. Replace `YOUR_SECRET_ID` and `YOUR_SECRET_KEY` with your SecretId and SecretKey, respectively.



```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

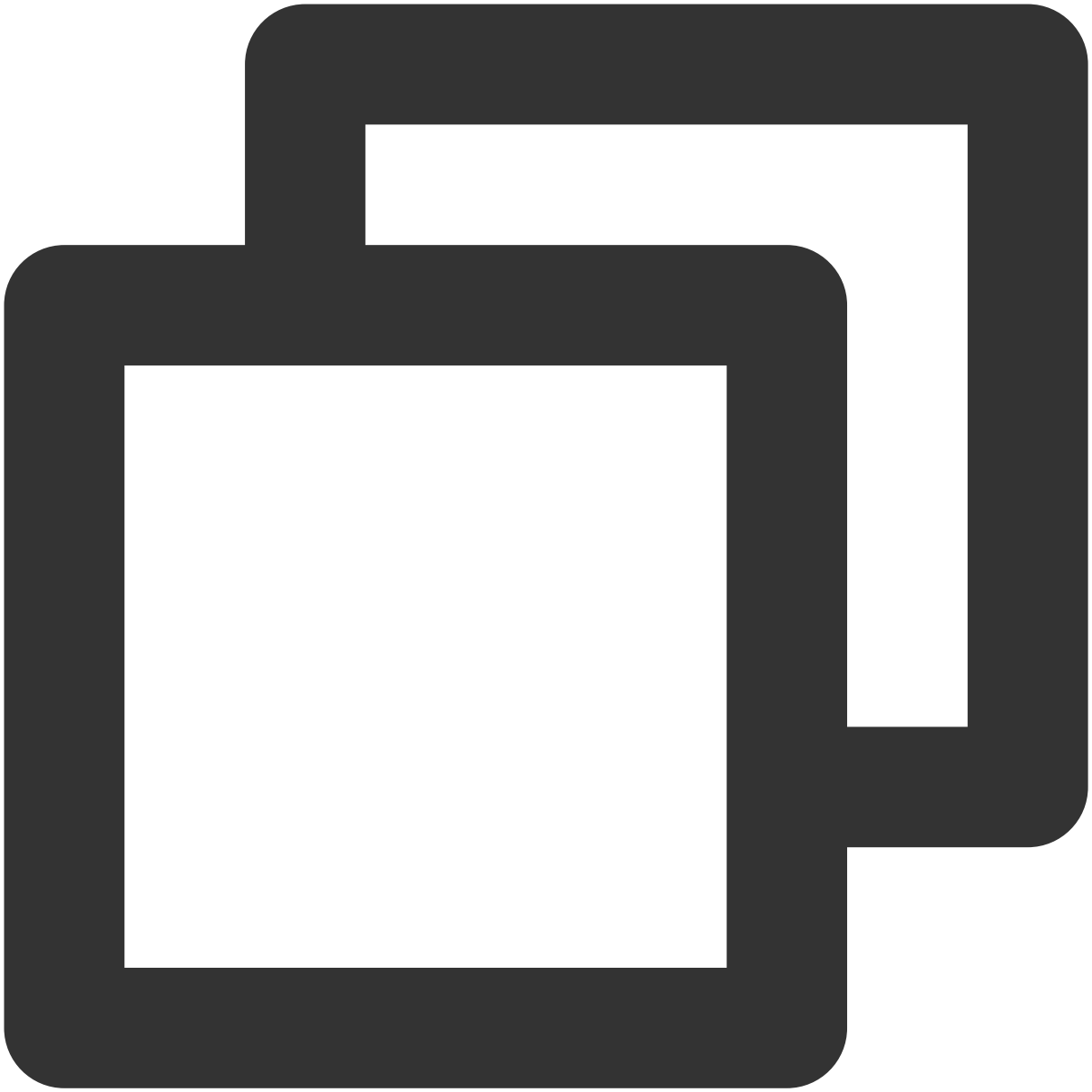
Creating Prometheus Instances

1. Create the configuration file for Terraform. Create a file named main.tf in the project directory and open it with an appropriate editor. Add the Tencent Cloud provider configuration to the main.tf or provider.tf file.



```
provider "tencentcloud" {
  region     = "your_region"
  secret_id  = "your_secret_id"
  secret_key = "your_secret_key"
}
```

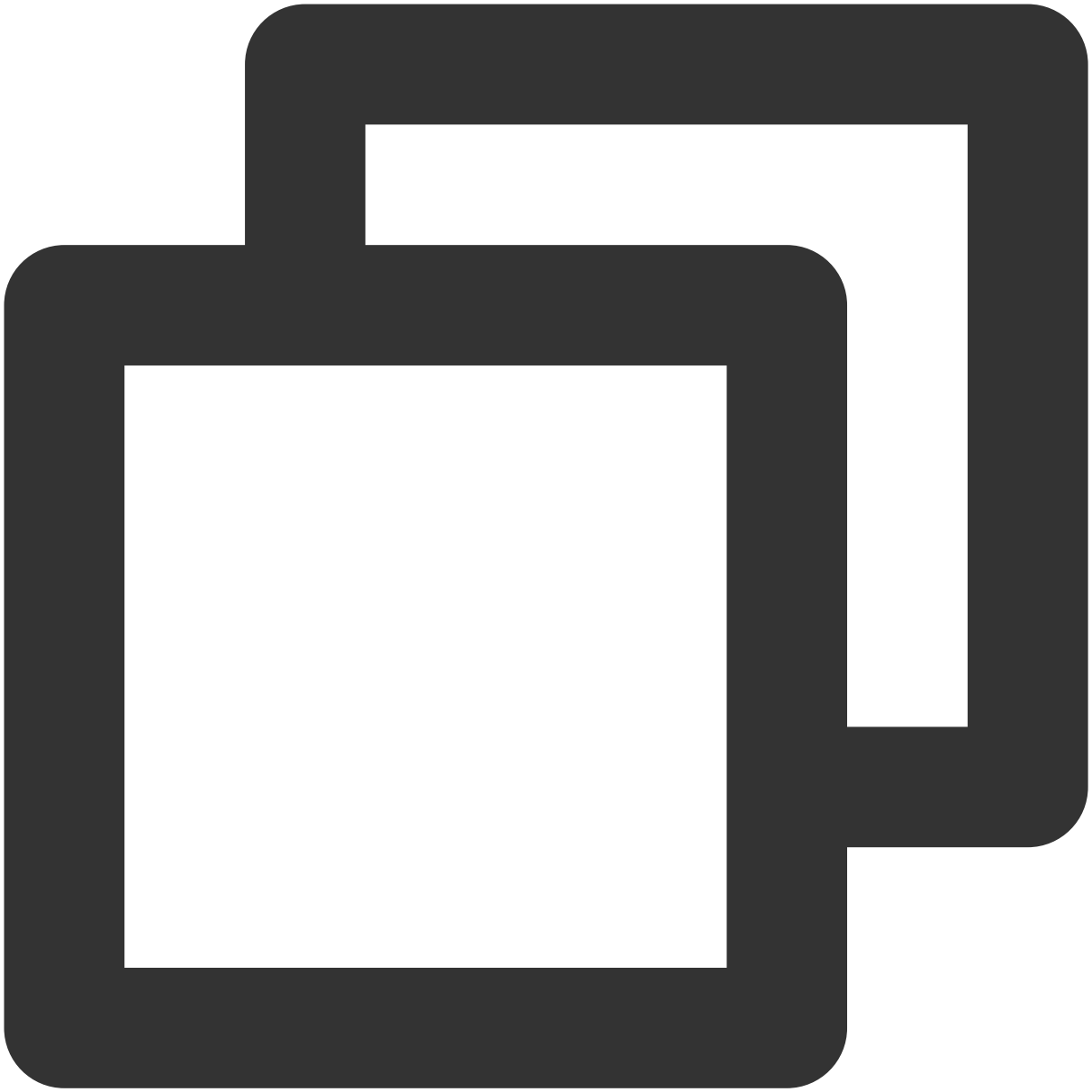
2. Defining Tencent Cloud Resources: Use the resources provided by Tencent Cloud in the main.tf file to define the resources you want to create and manage.



```
 #Specifying Provider Configuration Information

 terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}

 #Creating a Prometheus Instance in the Mumbai Region
```

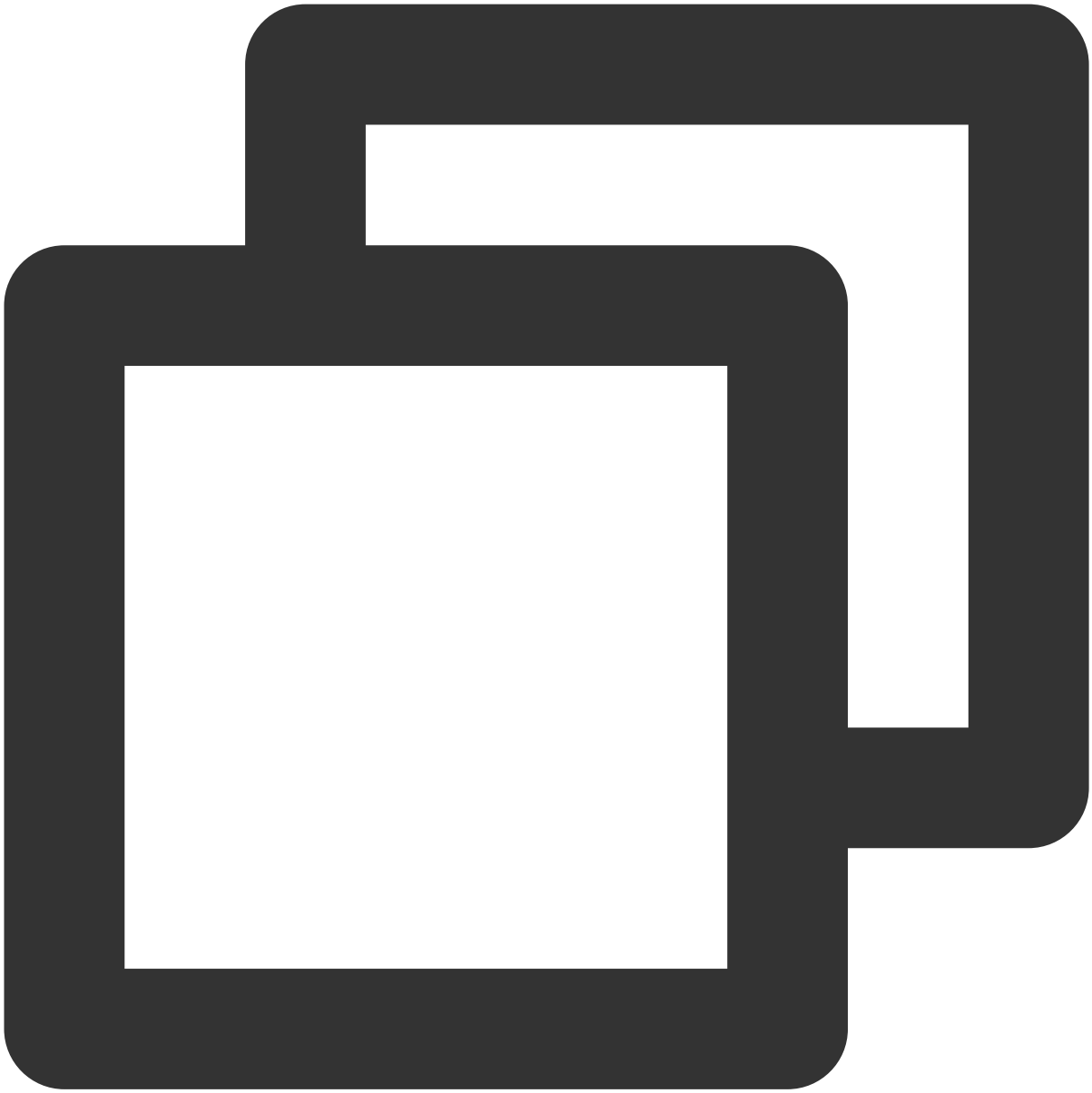
```
resource "tencentcloud_monitor_tmp_instance" "foo" {
  instance_name       = "tf-tmp-instance-sjtest"
  vpc_id              = "vpc-0n42dxzs"
  subnet_id           = "subnet-es8rv1kx"
  data_retention_time = 30
  zone                = "ap-mumbai-1"
  tags = {
    "createdBy" = "terraform"
  }
}

#Creating a Grafana Instance in the Mumbai Region (Optional)

resource "tencentcloud_monitor_grafana_instance" "foo" {
  instance_name         = "tf-grfana-cstest"
  vpc_id                = "vpc-0n42dxzs"
  subnet_ids            = ["subnet-es8rv1kx"]
  grafana_init_password = "1234567890"
  enable_internet       = false
  is_destroy            = true

  tags = {
    "createdBy" = "test"
  }
}

#Binding Grafana and Prometheus Instances (Optional)

resource "tencentcloud_monitor_tmp_manage_grafana_attachment" "foo" {
  grafana_id  = tencentcloud_monitor_grafana_instance.foo.id
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
}
```

3. Initialize the Terraform runtime environment by running the following command:



```
terraform init
```

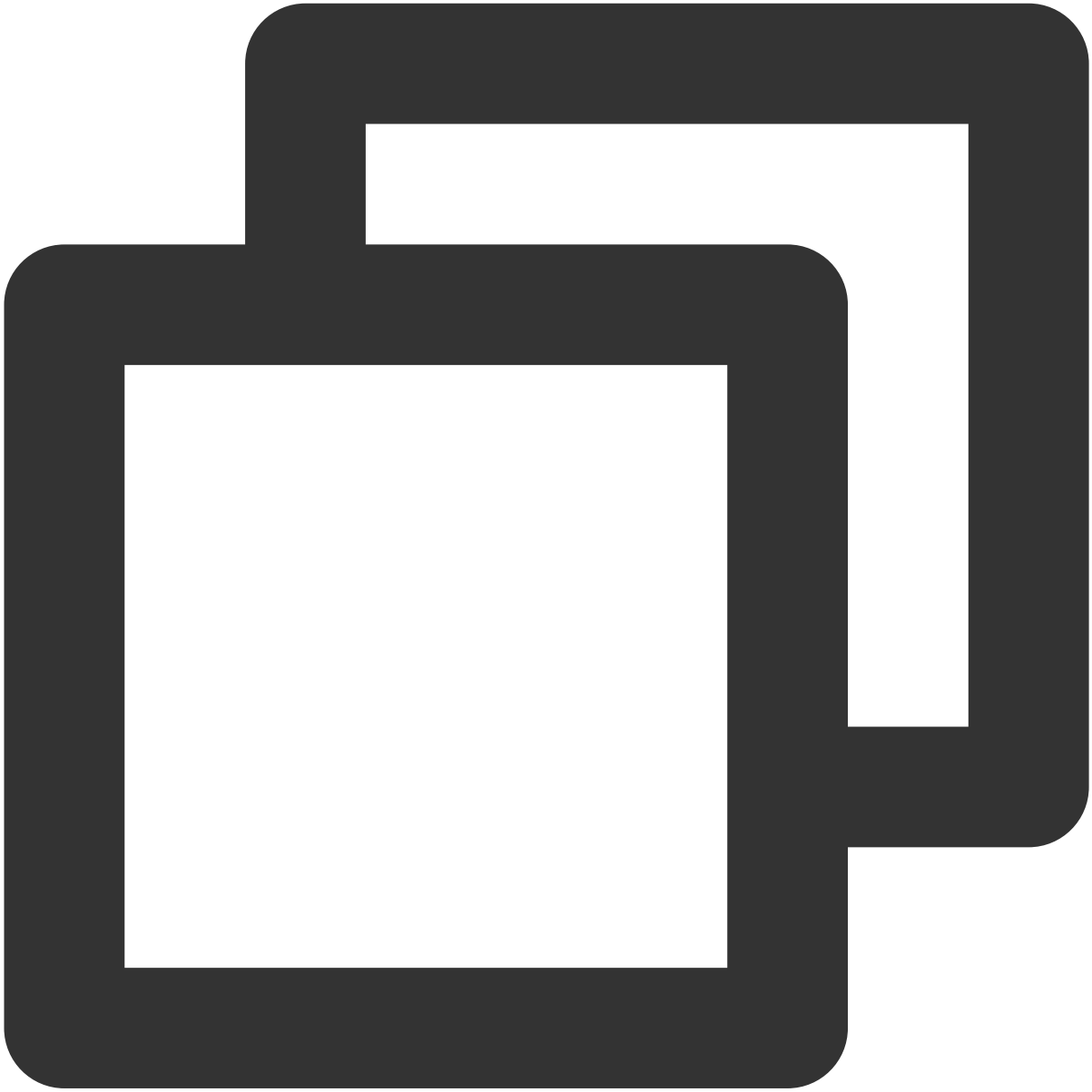
Expected output information:



```
  Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lo
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

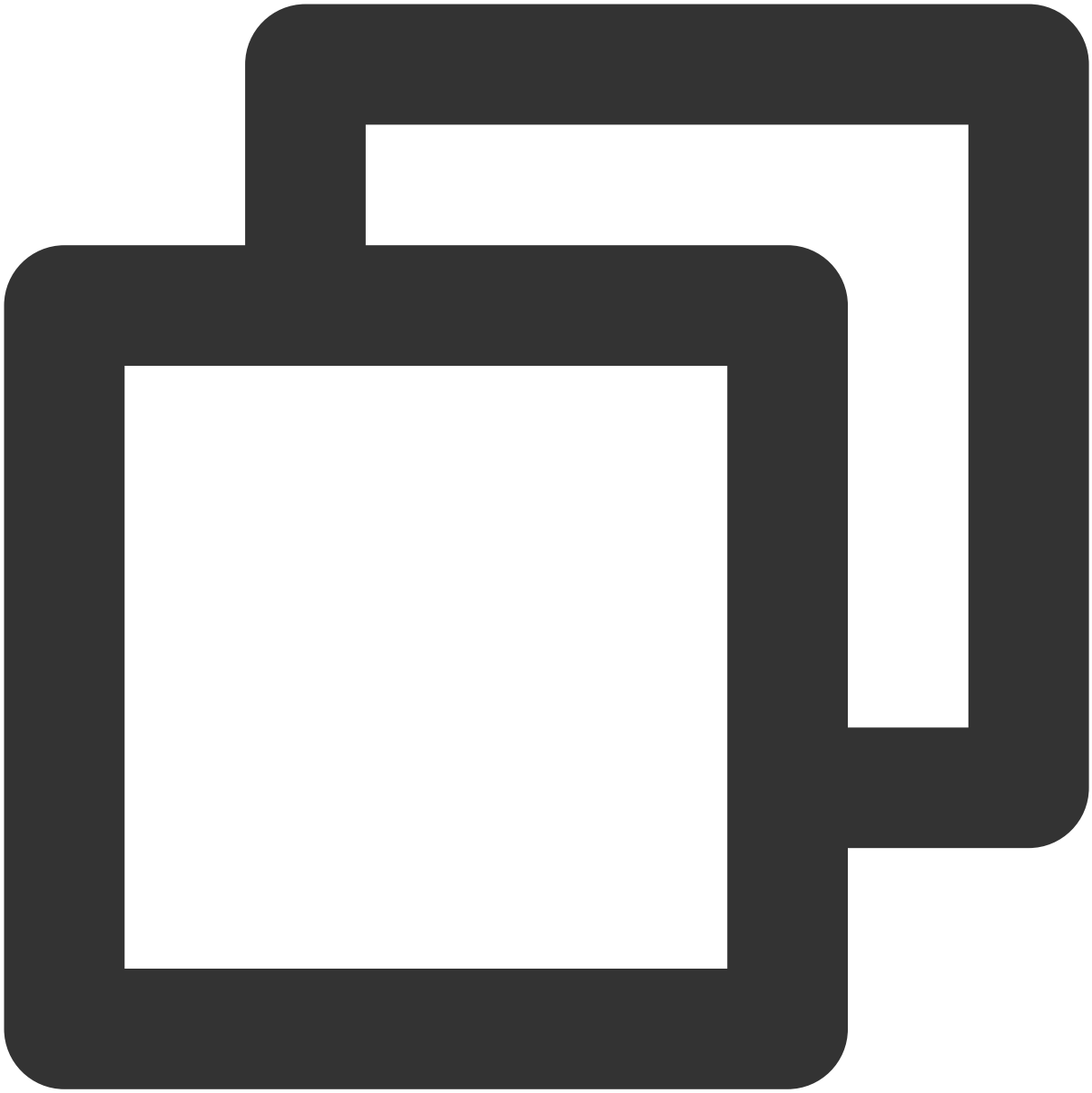
```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

4. To generate resource planning, run the following command:



```
terraform plan
```

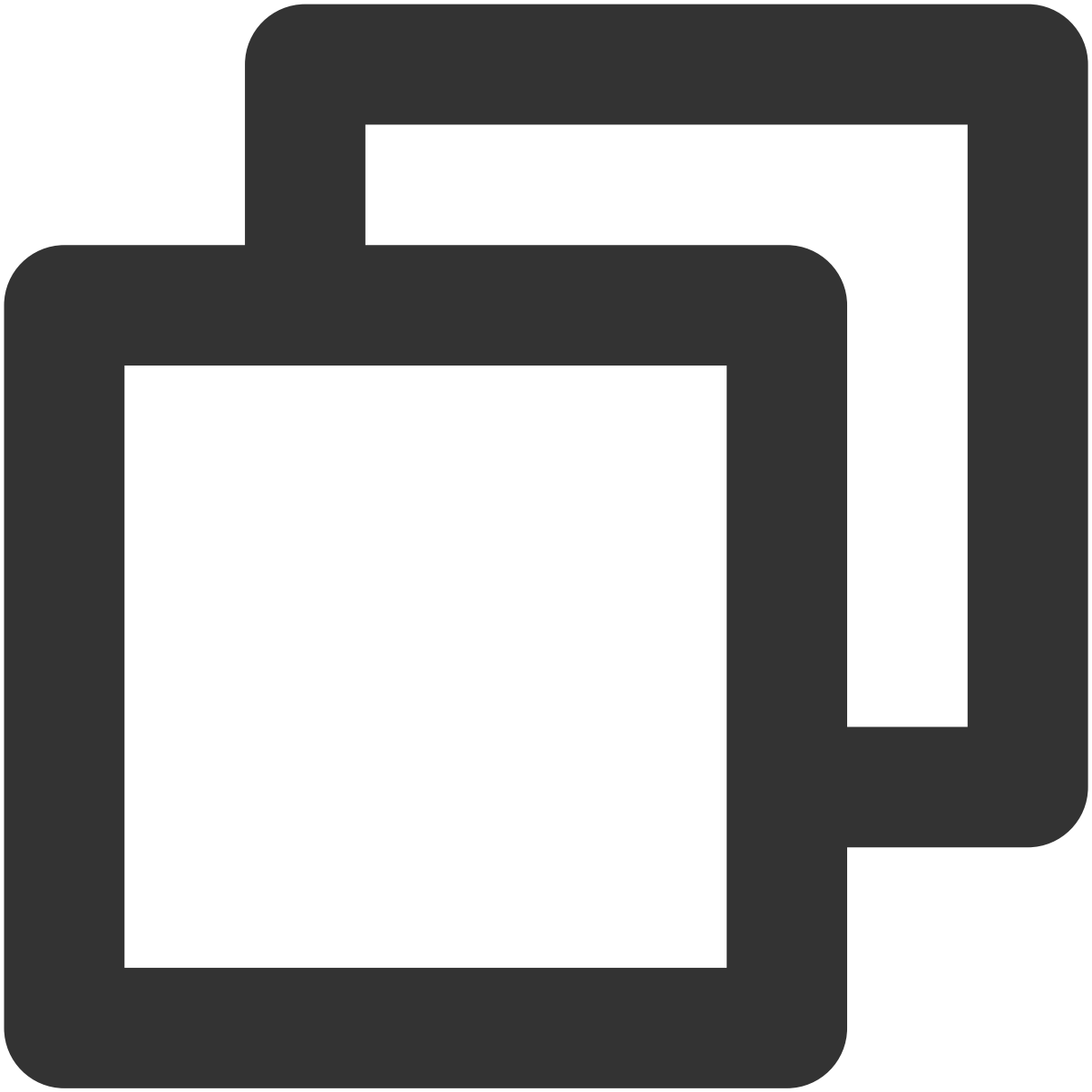
Expected output information:



```
  tencentcloud_vpc.vpc: Refreshing state... [id=vpc-3csjp7k8]
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-4wcvt7p1]

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_instance.foo will be created
  + resource "tencentcloud_monitor_tmp_instance" "foo" {
```

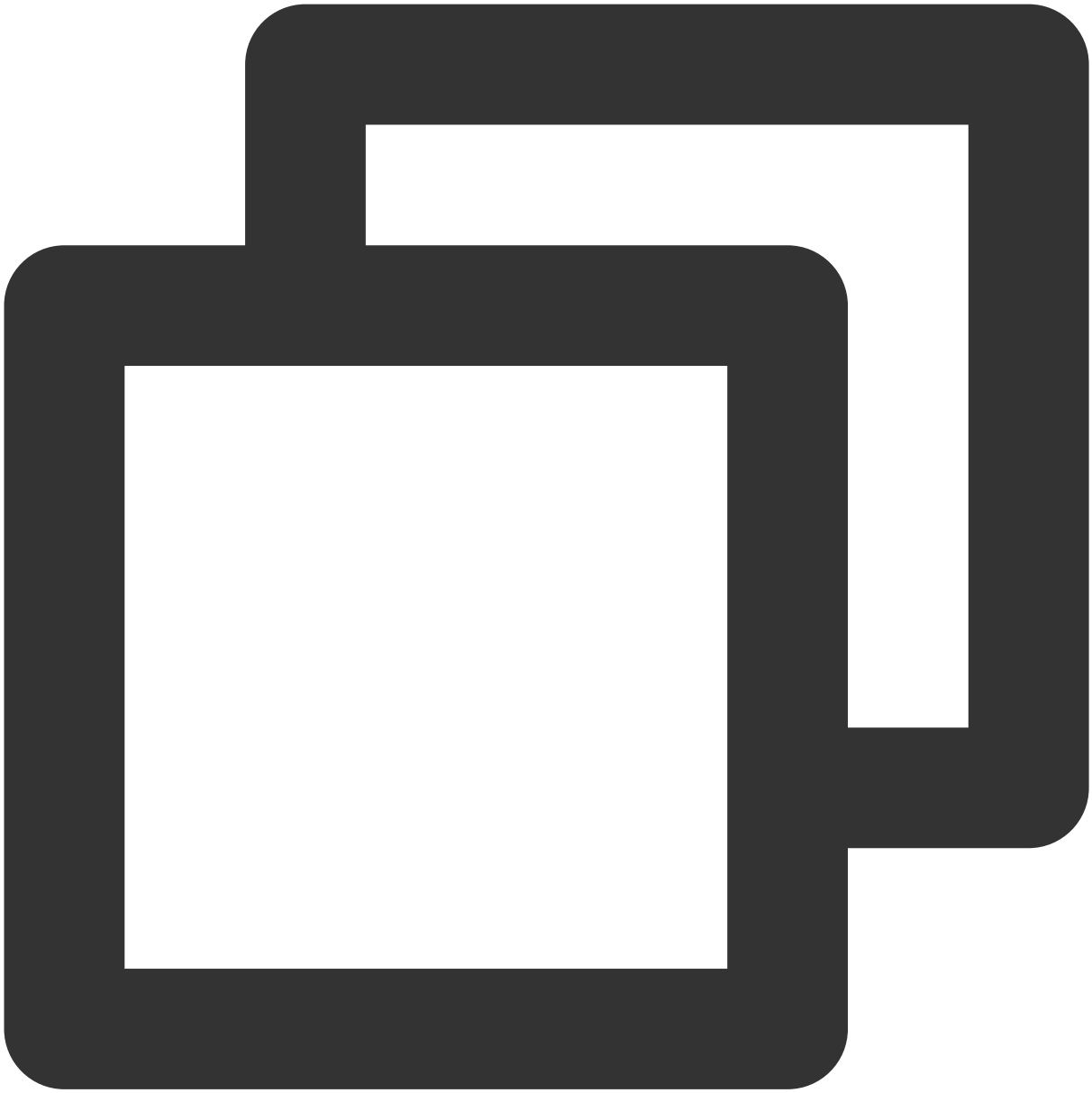
```
      + api_root_path       = (known after apply)
      + data_retention_time = 30
      + id                  = (known after apply)
      + instance_name       = "tf-tmp-instance-sjtest"
      + ipv4_address        = (known after apply)
      + proxy_address       = (known after apply)
      + remote_write        = (known after apply)
      + subnet_id           = "subnet-es8rv1kx"
      + tags                = {
          + "createdBy" = "terraform"
        }
      + vpc_id              = "vpc-0n42dxzs"
      + zone                = "ap-mumbai-1"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

5. To create an instance, run the following command:



```
terraform apply
```

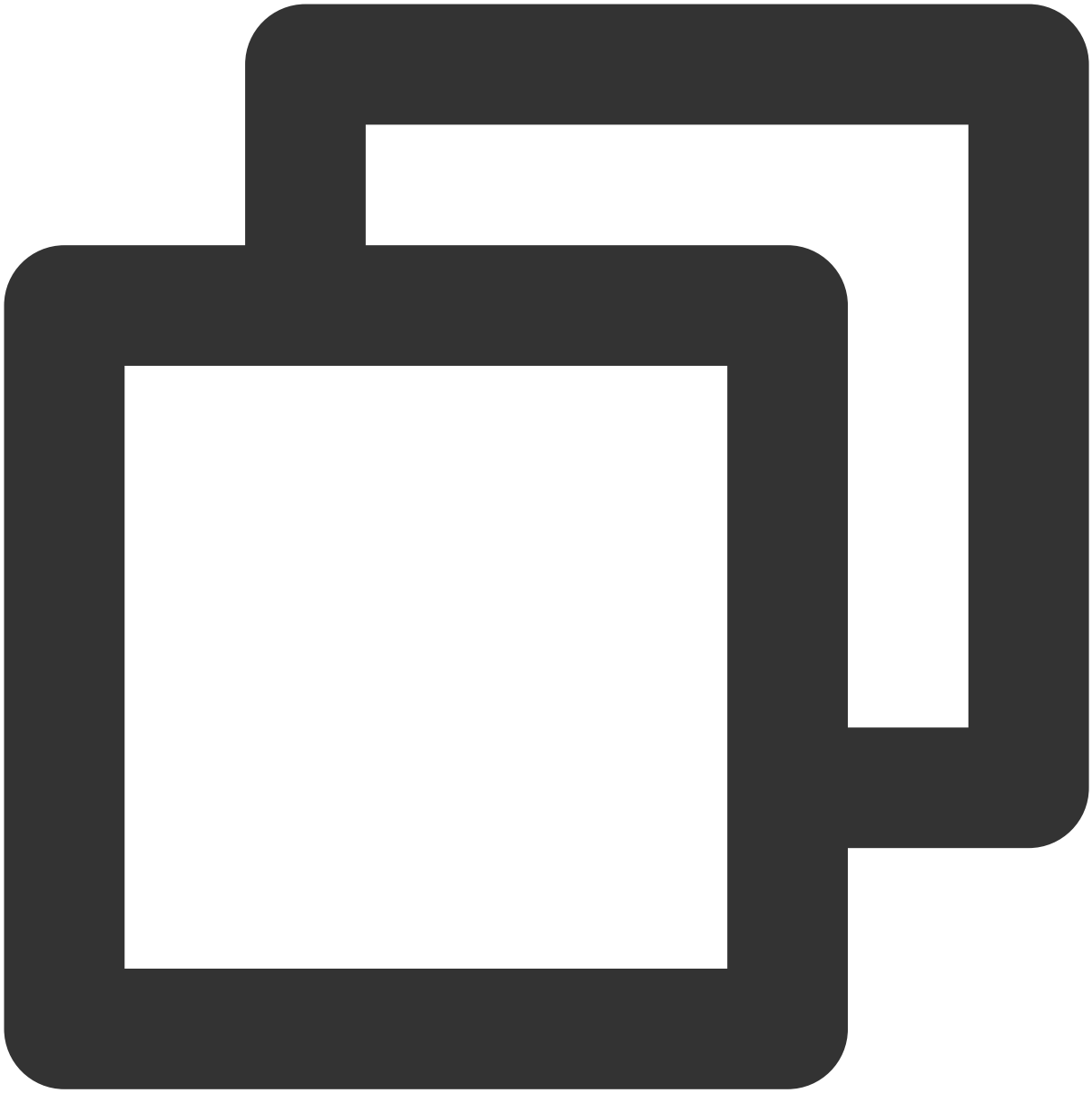
Expected output information:



```
tencentcloud_vpc.vpc: Refreshing state... [id=vpc-3csjp7k8]
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-4wcvt7p1]

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_instance.foo will be created
  + resource "tencentcloud_monitor_tmp_instance" "foo" {
```

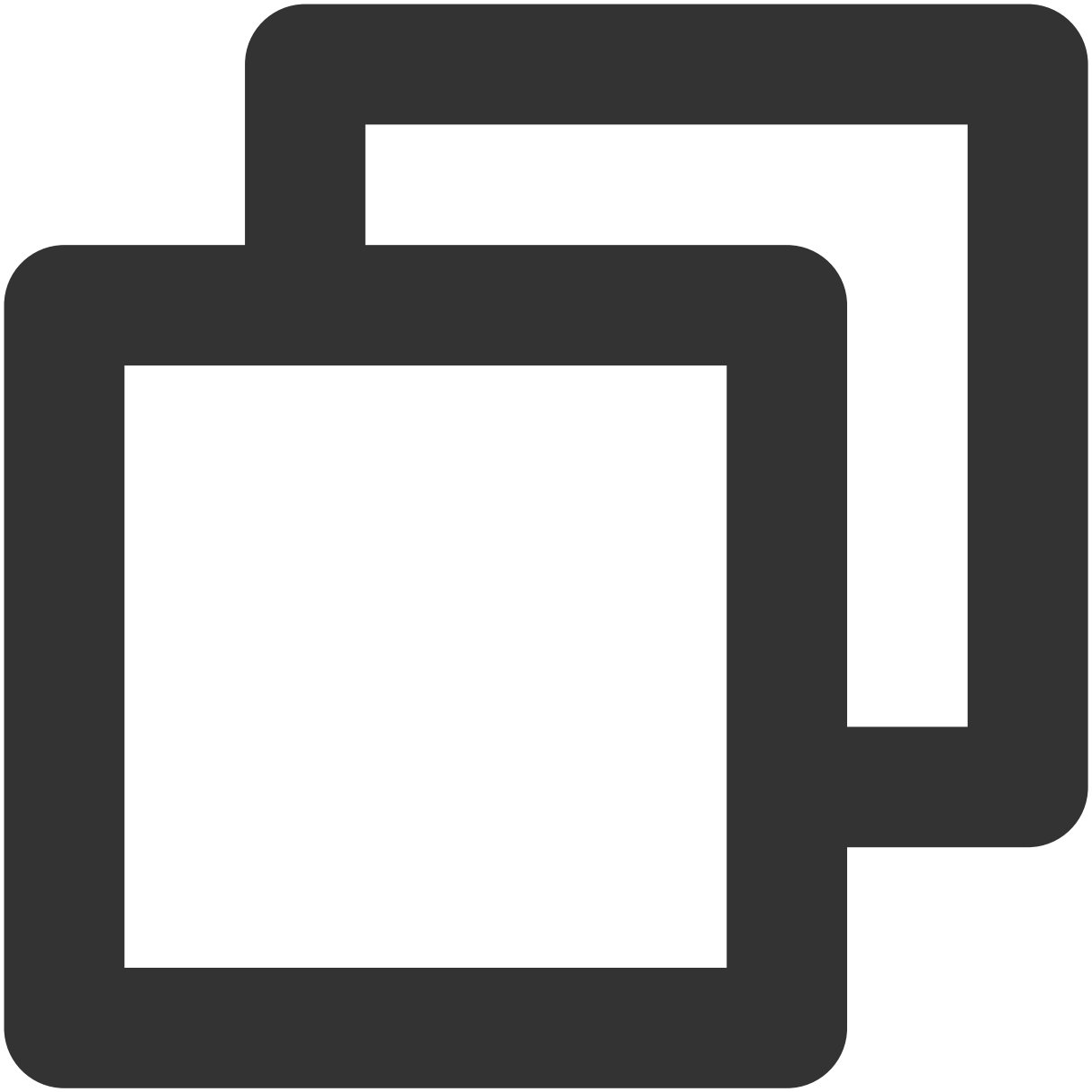
```
      + api_root_path       = (known after apply)
      + data_retention_time = 30
      + id                  = (known after apply)
      + instance_name       = "tf-tmp-instance-sjtest"
      + ipv4_address        = (known after apply)
      + proxy_address       = (known after apply)
      + remote_write        = (known after apply)
      + subnet_id           = "subnet-es8rv1kx"
      + tags                = {
          + "createdBy" = "terraform"
        }
      + vpc_id              = "vpc-0n42dxzs"
      + zone                = "ap-mumbai-1"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Enter "yes" to the following information to proceed:



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

If the following information appear, the creation is successful:



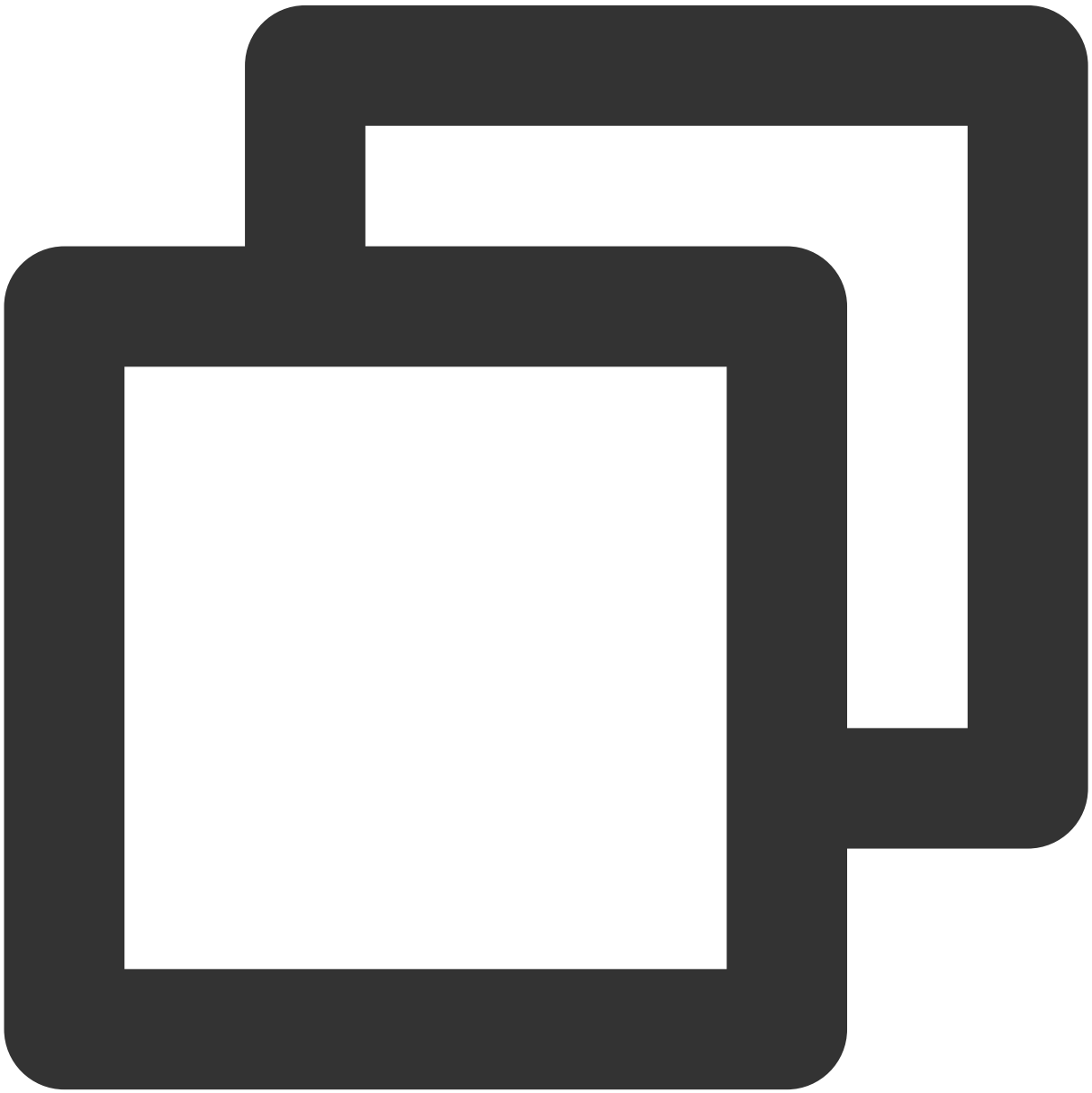
```
tencentcloud_monitor_tmp_instance.foo: Creating...
tencentcloud_monitor_tmp_instance.foo: Still creating... [10s elapsed]
tencentcloud_monitor_tmp_instance.foo: Creation complete after 12s [id=prom-8dyb6in

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Viewing the Status of Prometheus Instances

Log in to the TCOP, and choose **Managed Service for Prometheus** from the left-hand navigation bar. Existing instances are displayed in the Prometheus instance list.

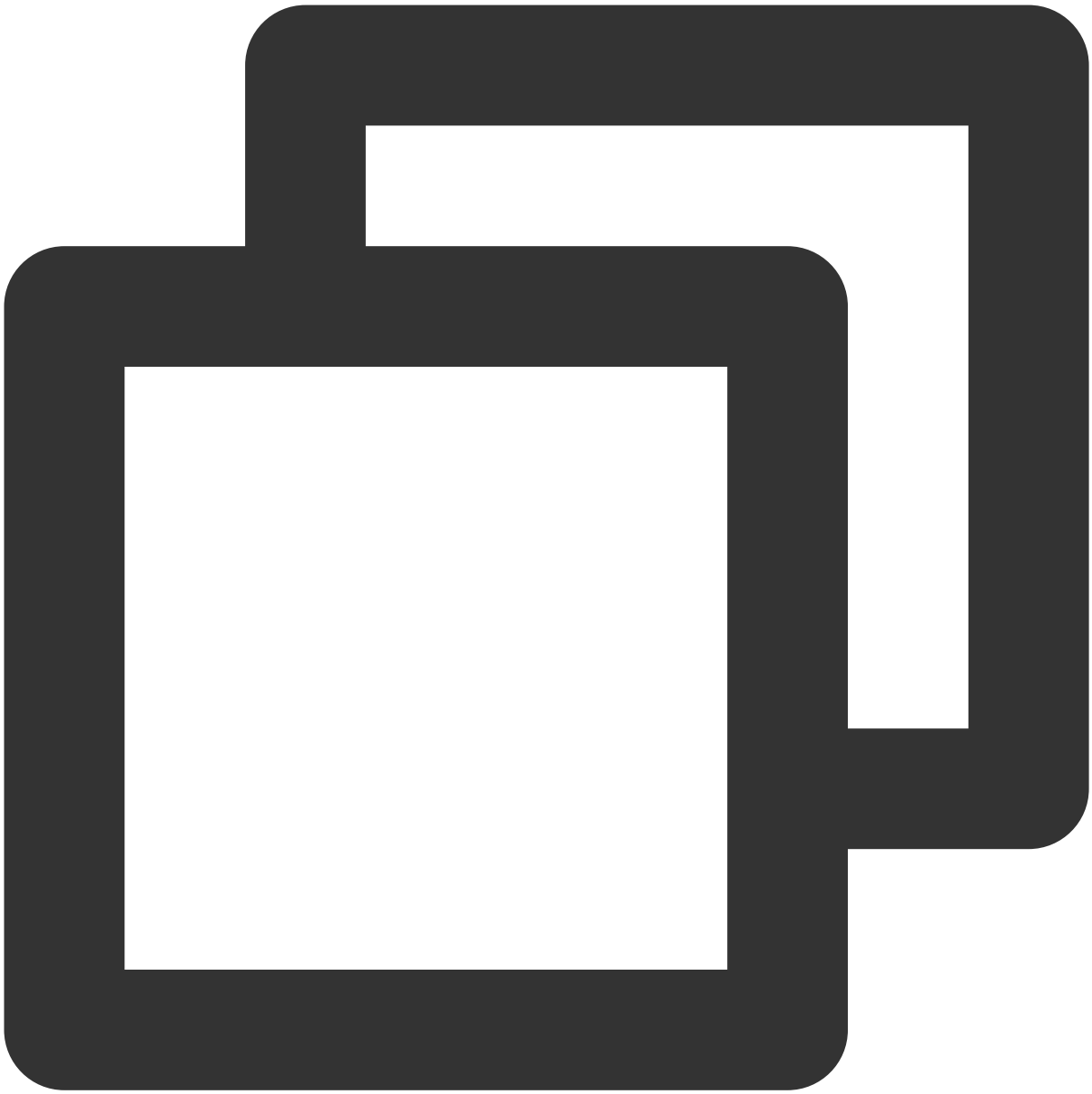
Deleting Prometheus Instances

To delete a Prometheus instance, run the following command:



```
terraform destroy
```

If the following information appears, you are prompted to enter "yes" for confirmation:



```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  - destroy

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_instance.foo will be destroyed
```

```
  - resource "tencentcloud_monitor_tmp_instance" "foo" {
      - api_root_path       = "http://10.0.0.34:9090/api/v1" -> null
      - data_retention_time = 30 -> null
      - id                  = "prom-8dyb6iny" -> null
      - instance_name       = "tf-tmp-instance-sjtest" -> null
      - ipv4_address        = "10.0.0.34" -> null
      - proxy_address       = "10.0.0.34:9090" -> null
      - remote_write        = "http://10.0.0.34:9090/api/v1/prom/write" -> null
      - subnet_id           = "subnet-es8rv1kx" -> null
      - tags                = {
          - "createdBy" = "terraform"
        } -> null
      - vpc_id              = "vpc-0n42dxzs" -> null
      - zone                = "ap-mumbai-1" -> null
    }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

Note:

When `Destroy complete! Resources: 1 destroyed.` is displayed, the instance has been successfully deleted.

Managing the Integration Center of Prometheus Instances Using Terraform

Last updated：2023-12-29 16:09:44

Prerequisites

Installing Terraform

For detailed instructions on installing Terraform, see Installing Terraform.

Note:

The installed version of Terraform must not be below v1.6.3. You can verify the version of Terraform installed by using the command terraform --version.

Configuring Tencent Cloud Account Information

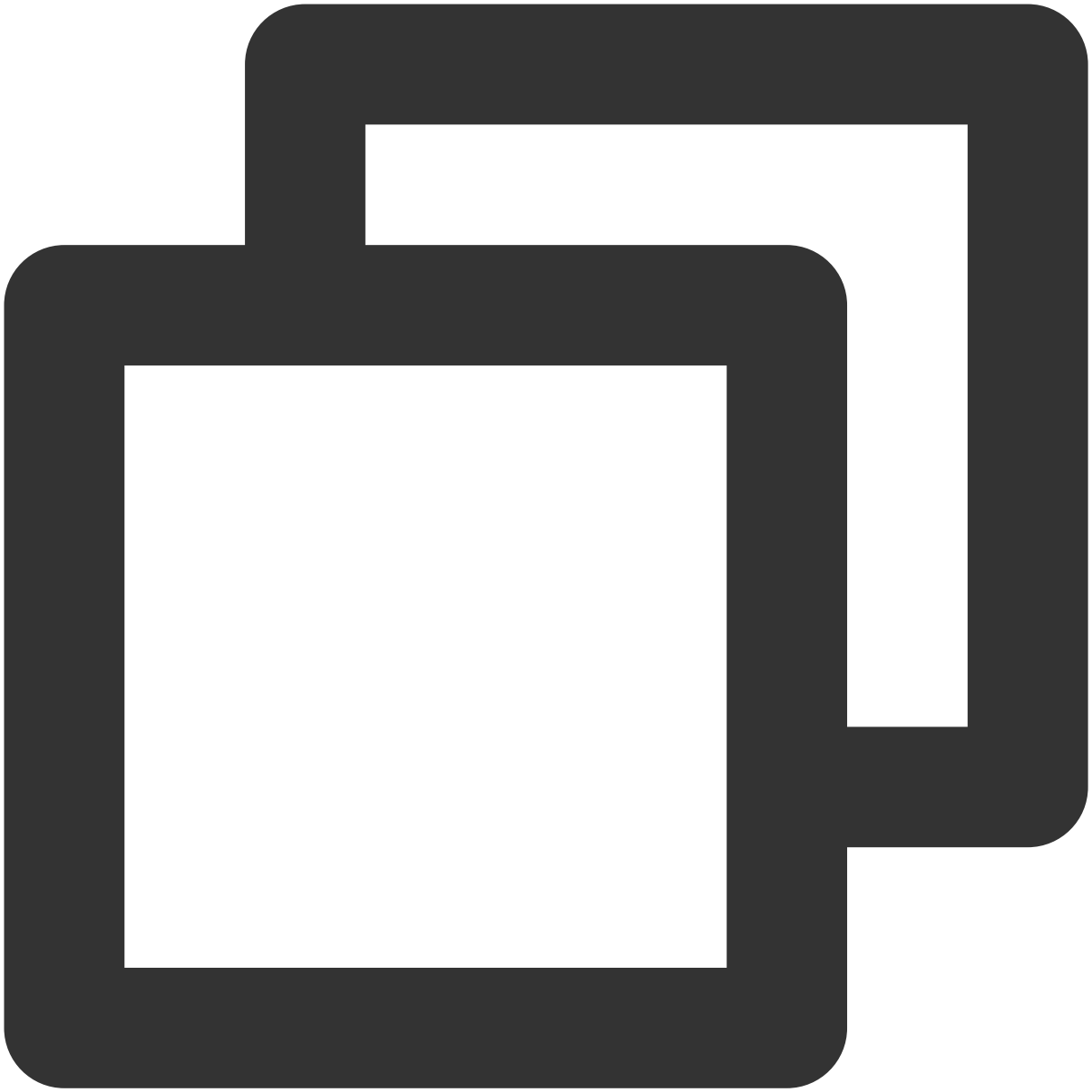
Before using Terraform for the first time, go to API Key to apply for a SecretID and SecretKey of the security credentials. If you already have valid credentials, skip this step.

1. Log in to the CAM Console and choose **Access Key > API Keys** from the left sidebar.
2. On the API Keys page, click **Create Key** to create a SecretId/SecretKey pair.

There are two methods for configuring Tencent Cloud account information:

Authentication by Static Credentials

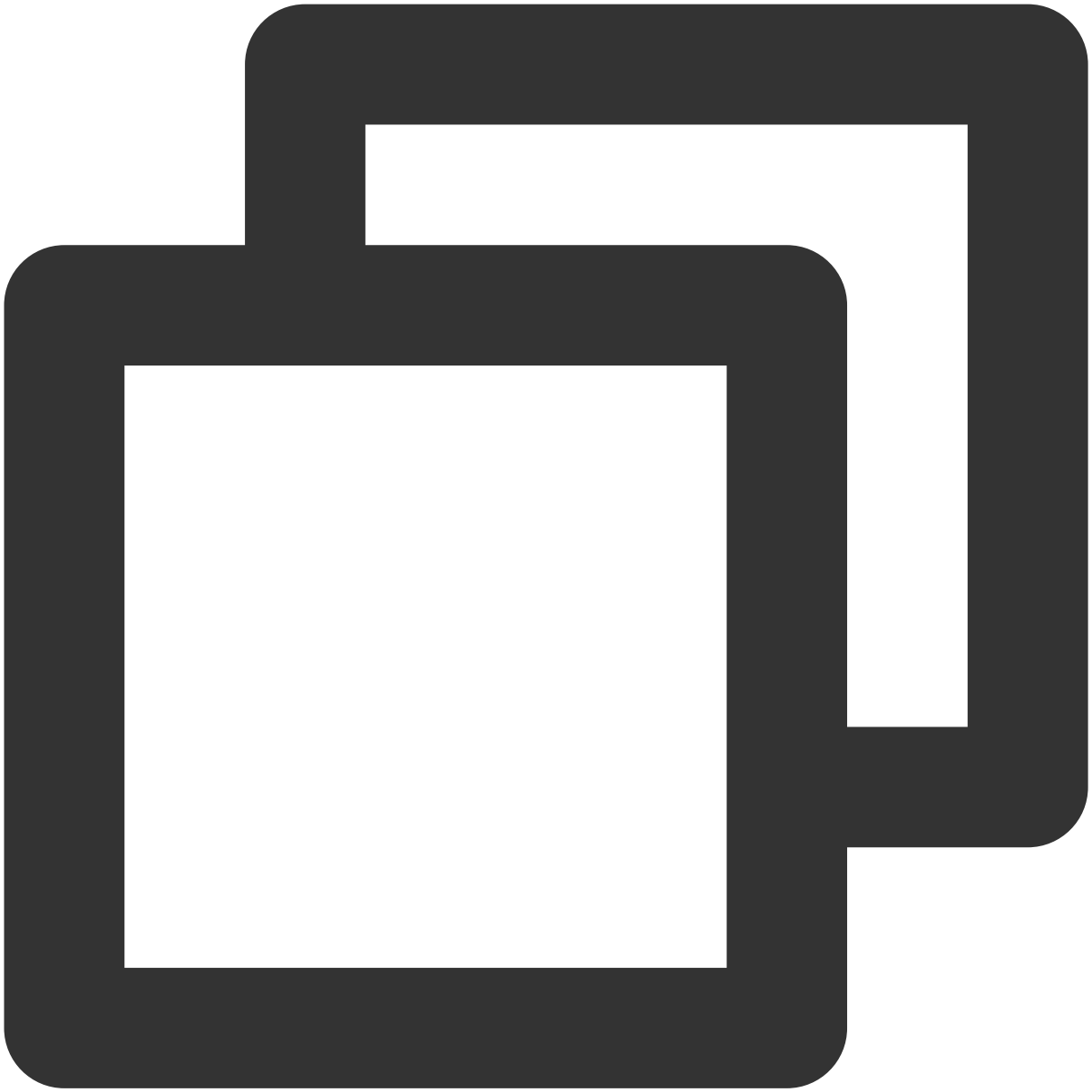
Create a `provider.tf` file in the user directory with the following content. Replace `my-secret-id` and `my-secret-key` with your SecretId and SecretKey, respectively.



```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

Authentication by Environment Variables

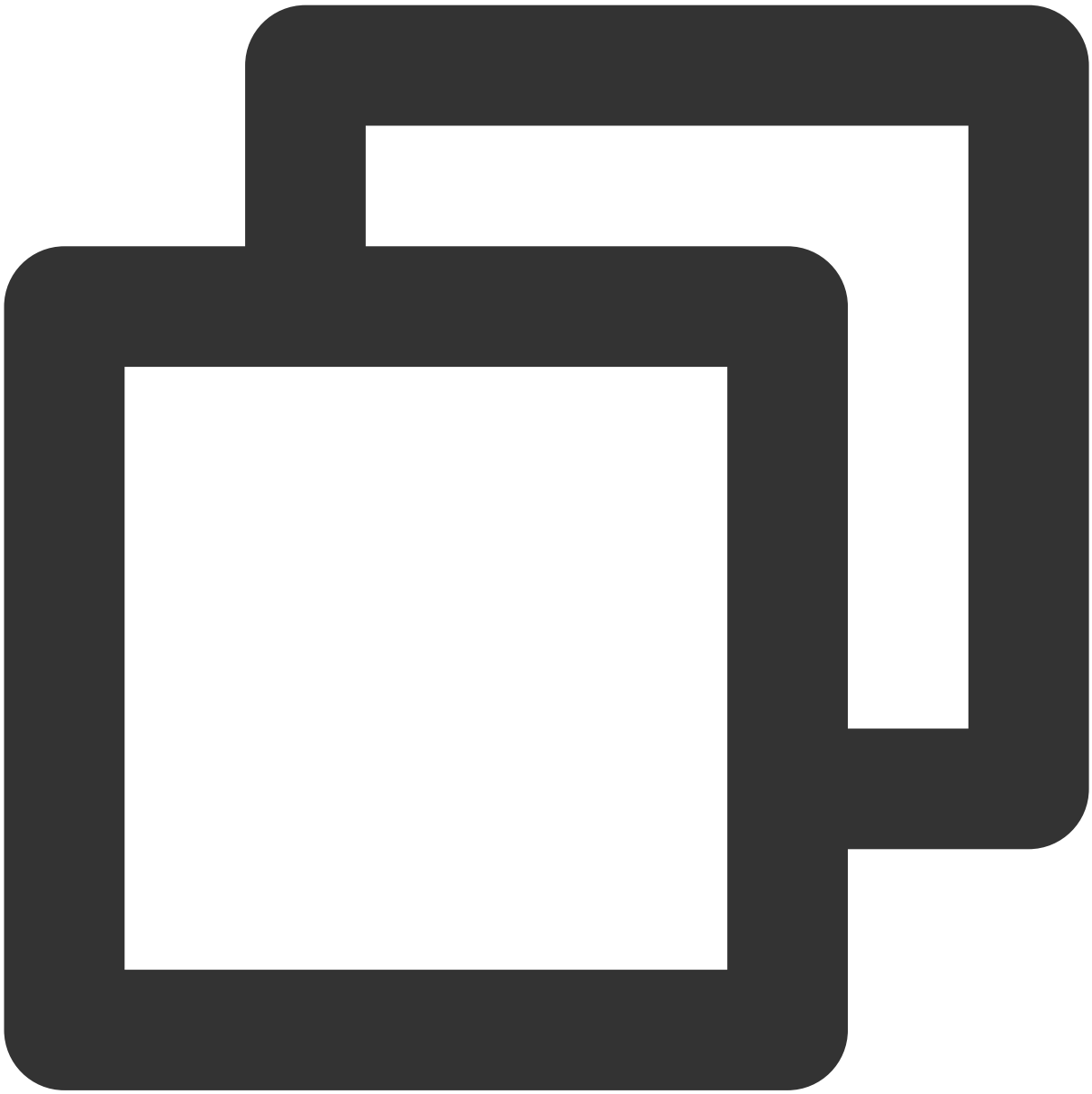
Configure the computer environment variables or cloud environment variables by running the following command. Replace `YOUR_SECRET_ID` and `YOUR_SECRET_KEY` with your SecretId and SecretKey, respectively.



```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

Incorporating Component Integration from the Integration Center of a Prometheus Instance

1. Creating a new Terraform configuration file: Create a new directory and a file named main.tf in the directory. The following is the configuration information:



```
#Specifying Provider Configuration Information

terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
```

```
  }
}

#Prometheus Managing Cloud Monitoring Integration

##black-box Integration
resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterIntegration" {
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  kind        = "blackbox-exporter"
  content     = "{\\"name\\":\\"balck-box-tf-test\\",\\"kind\\":\\"blackbox-exporte
  kube_type   = 1
  cluster_id  = ""
}

##Cloud Monitoring Plugin Integration
resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterMointor" {
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  kind        = "qcloud-exporter"
  content     = "{\\"name\\":\\"tf-test-cjtest\\",\\"kind\\":\\"qcloud-exporter\\",
  kube_type   = 1
  cluster_id  = ""
}
```

Note:

The fields to be configured when you create the Integration Center component of the Prometheus instance are as follows:

cluster_id: (Required, string) Cluster ID.

content: (Required, string) Integration configuration (internal parameters within content can be replaced as needed).

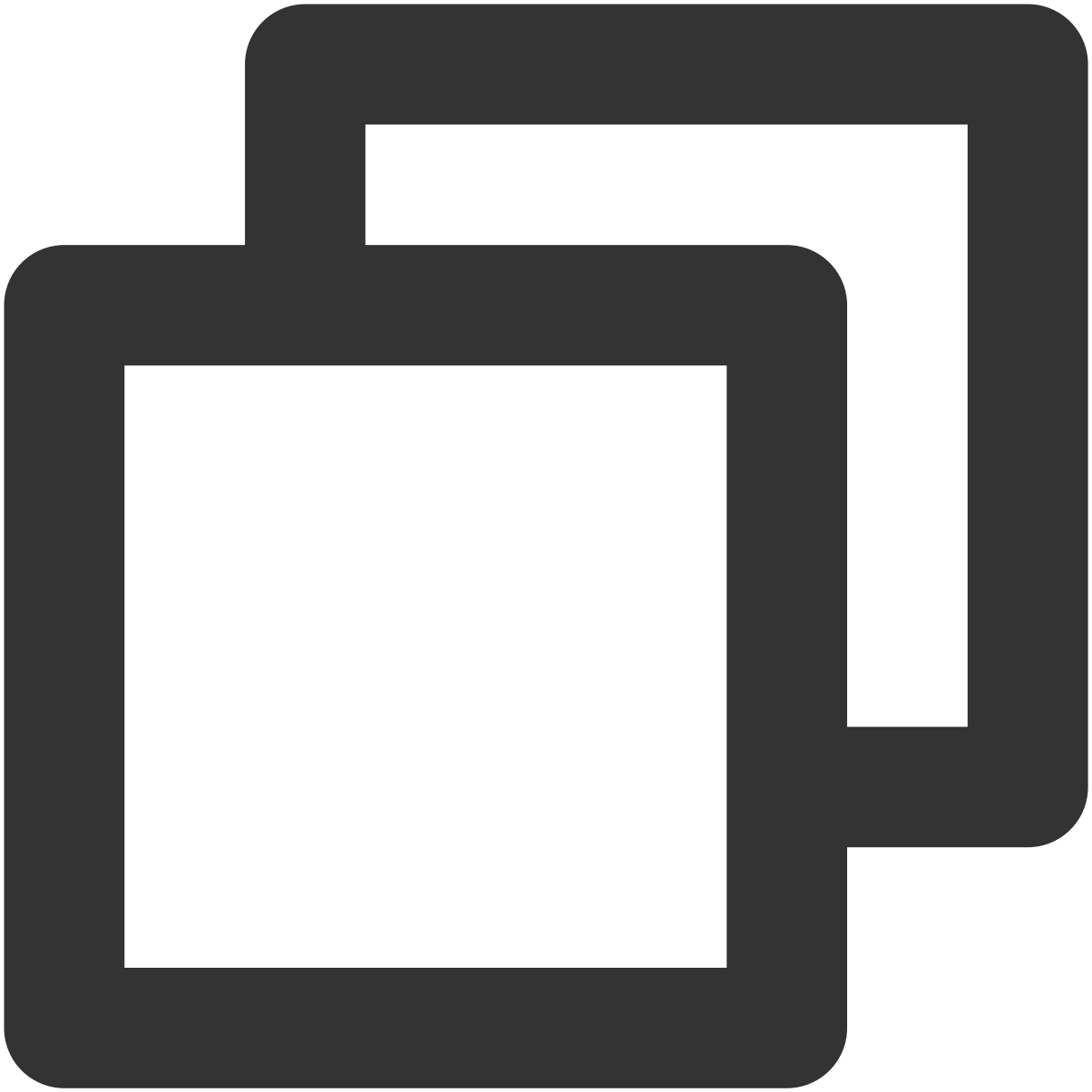
instance_id: (Required, string) Instance ID.

kind: (Required, string) Type.

kube_type: (Required, integer) Integration configuration.

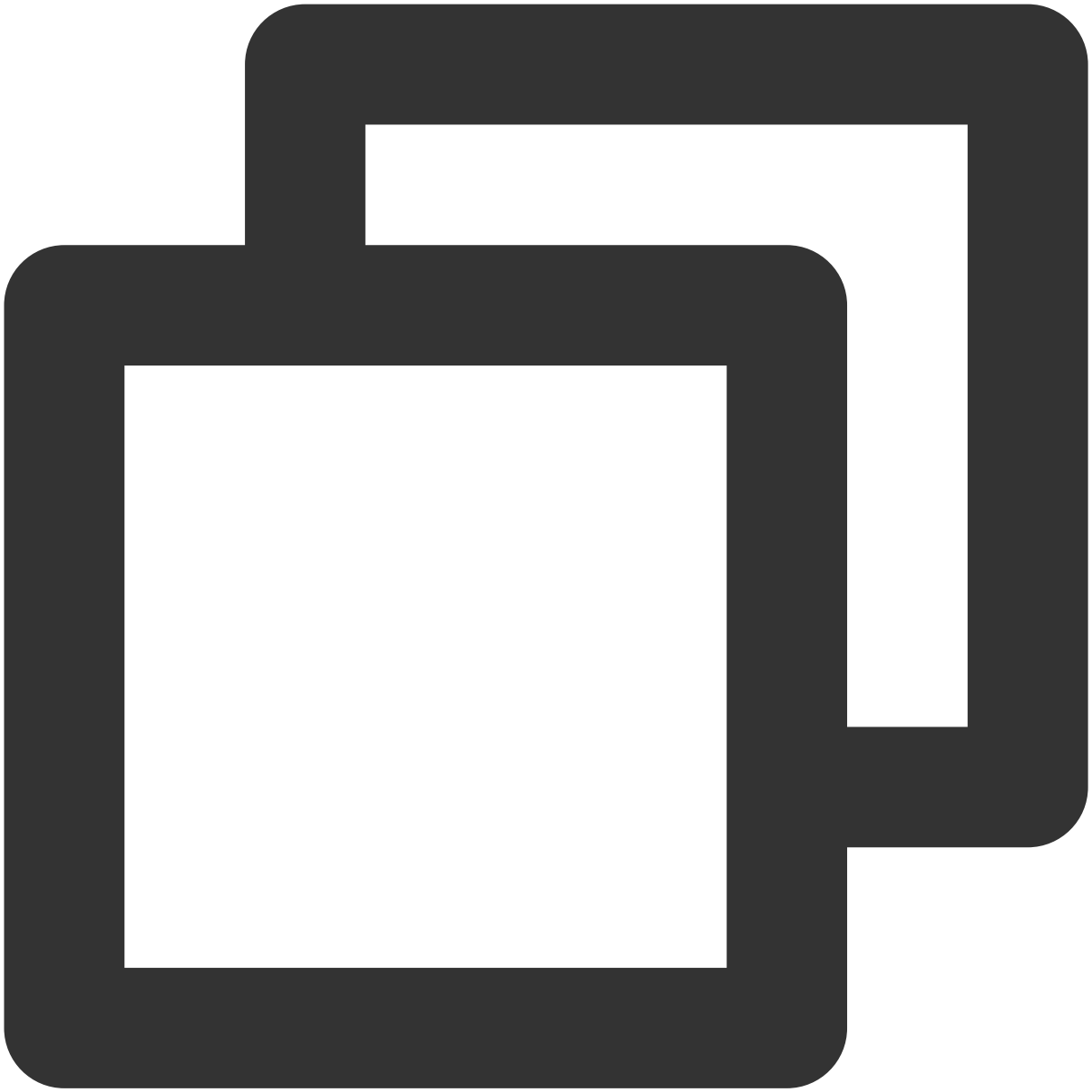
For more detailed parameters, see Tencent Cloud Integration with GitHub or Terraform Tencent Cloud Provider.

2. To initialize the Terraform runtime environment, run the following command:



```
terraform init
```

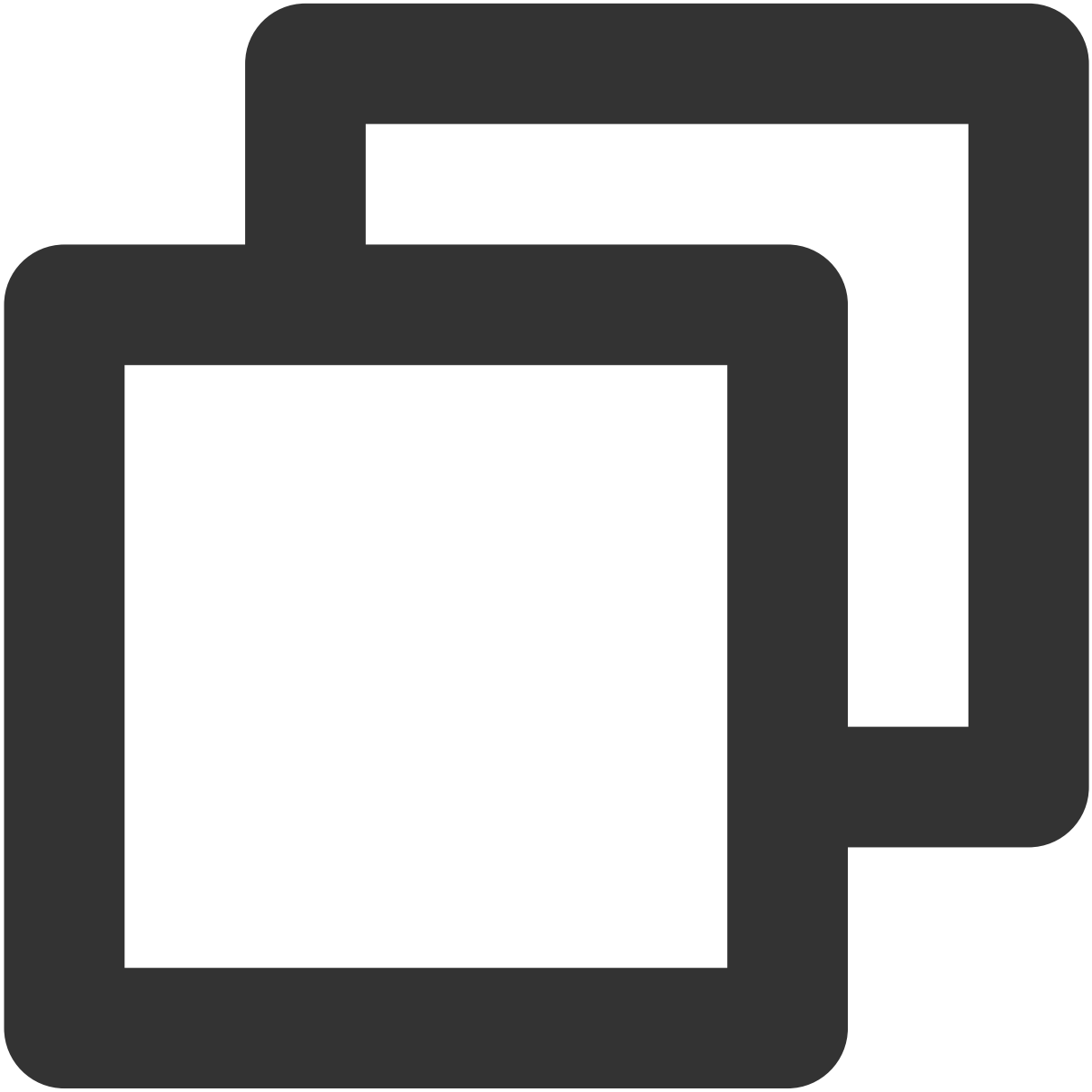
Expected output:



```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lo
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

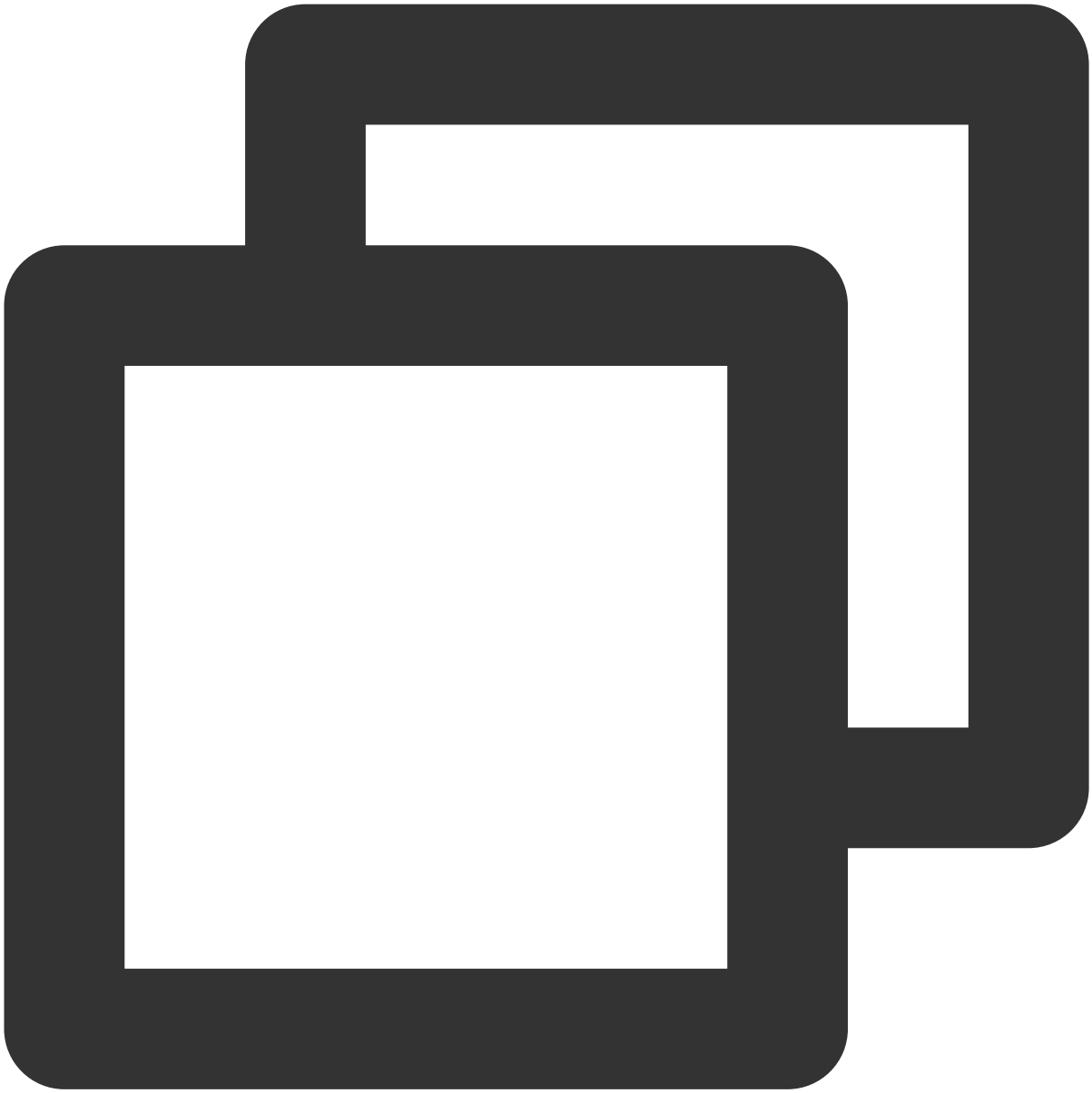
```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

3. To generate a resource plan, run the following command:



```
terraform plan
```

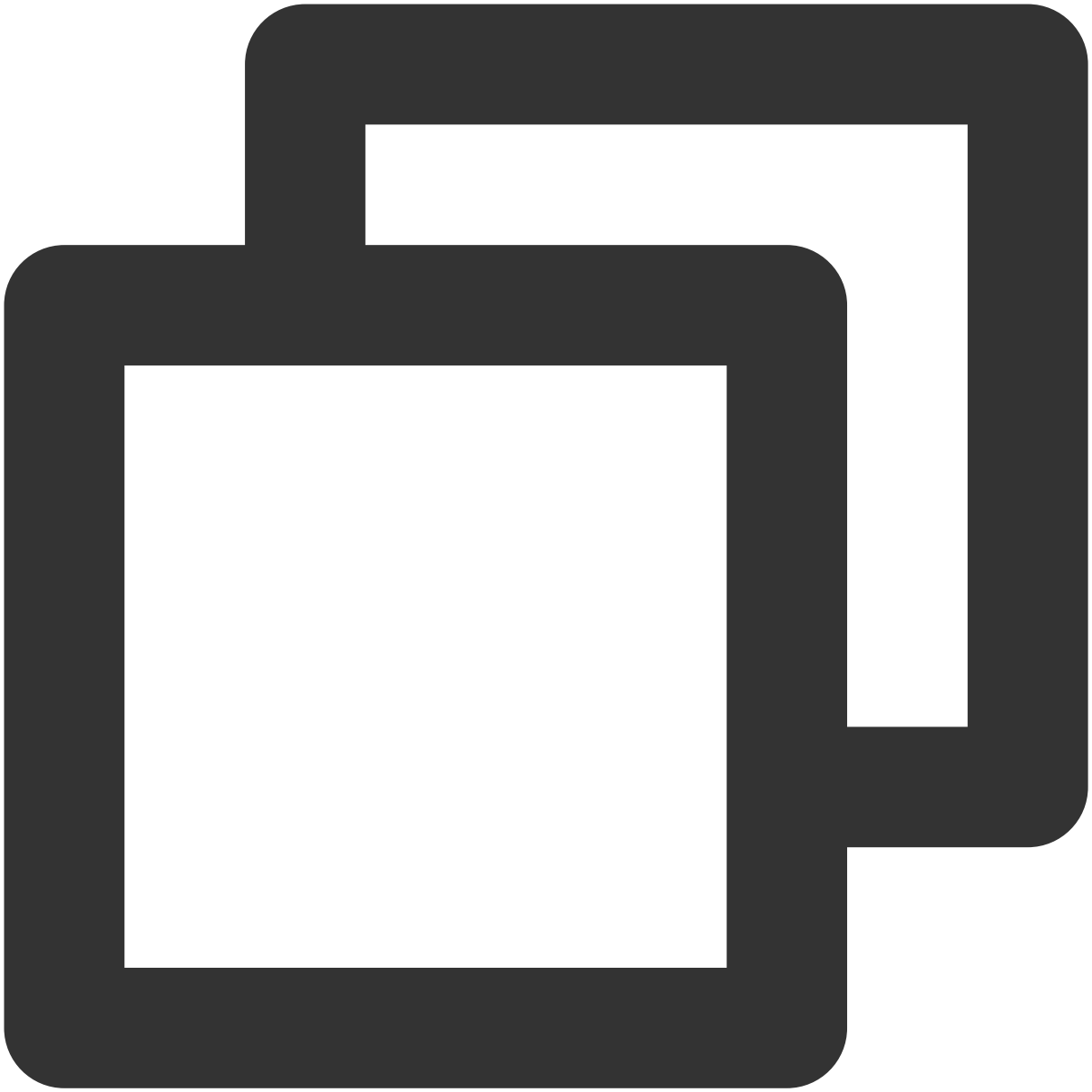
Expected output:



```
Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration will be cr
  + resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterIntegratio
      + content     = jsonencode(
            XXX...
        )
```

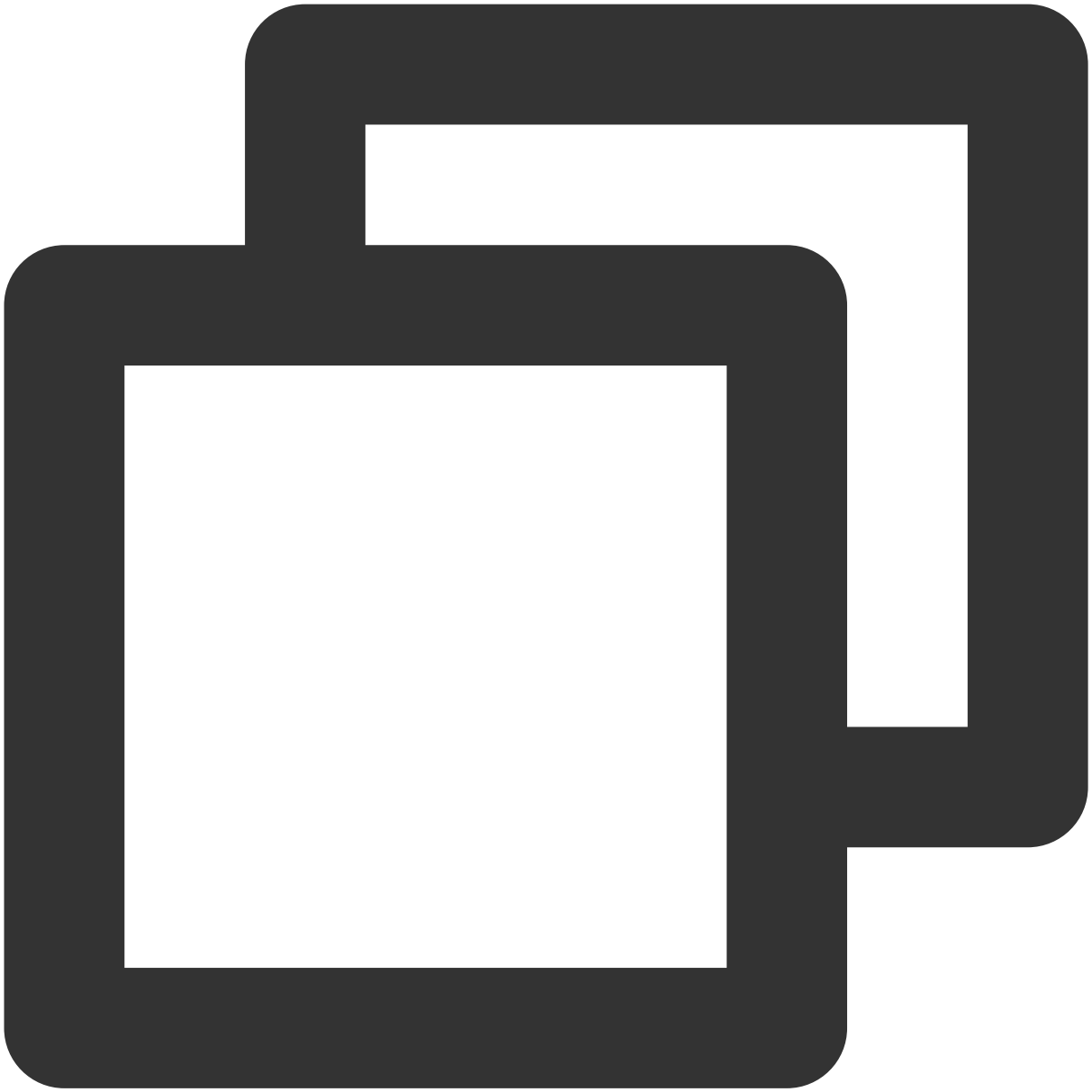
```
      + id          = (known after apply)
      + instance_id = (known after apply)
      + kind        = "blackbox-exporter"
      + kube_type   = 1
    }

  # tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor will be create
  + resource "tencentcloud_monitor_tmp_exporter_integration" "tmpExporterMointor" {
      + content     = jsonencode(
            XXX...
        )
      + id          = (known after apply)
      + instance_id = (known after apply)
      + kind        = "qcloud-exporter"
      + kube_type   = 1
    }

  # tencentcloud_monitor_tmp_instance.foo will be created
  + resource "tencentcloud_monitor_tmp_instance" "foo" {
      + api_root_path       = (known after apply)
      + data_retention_time = 30
      + id                  = (known after apply)
      + instance_name       = "tf-tmp-instance-sjtest"
      + ipv4_address        = (known after apply)
      + proxy_address       = (known after apply)
      + remote_write        = (known after apply)
      + subnet_id           = "subnet-es8rv1kx"
      + tags                = {
          + "createdBy" = "terraform"
        }
      + vpc_id              = "vpc-0n42dxzs"
      + zone                = "ap-mumbai-1"
    }

Plan: 3 to add, 0 to change, 0 to destroy.

──────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarante
you run "terraform apply" now.
```

4. To create component integration of the Integration Center, run the following command:



```
terraform apply
```

Expected output:



```
Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:
XXX...


Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
```

```
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Creating...
tencentcloud_monitor_tmp_instance.foo: Still creating... [10s elapsed]
...

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

Note:

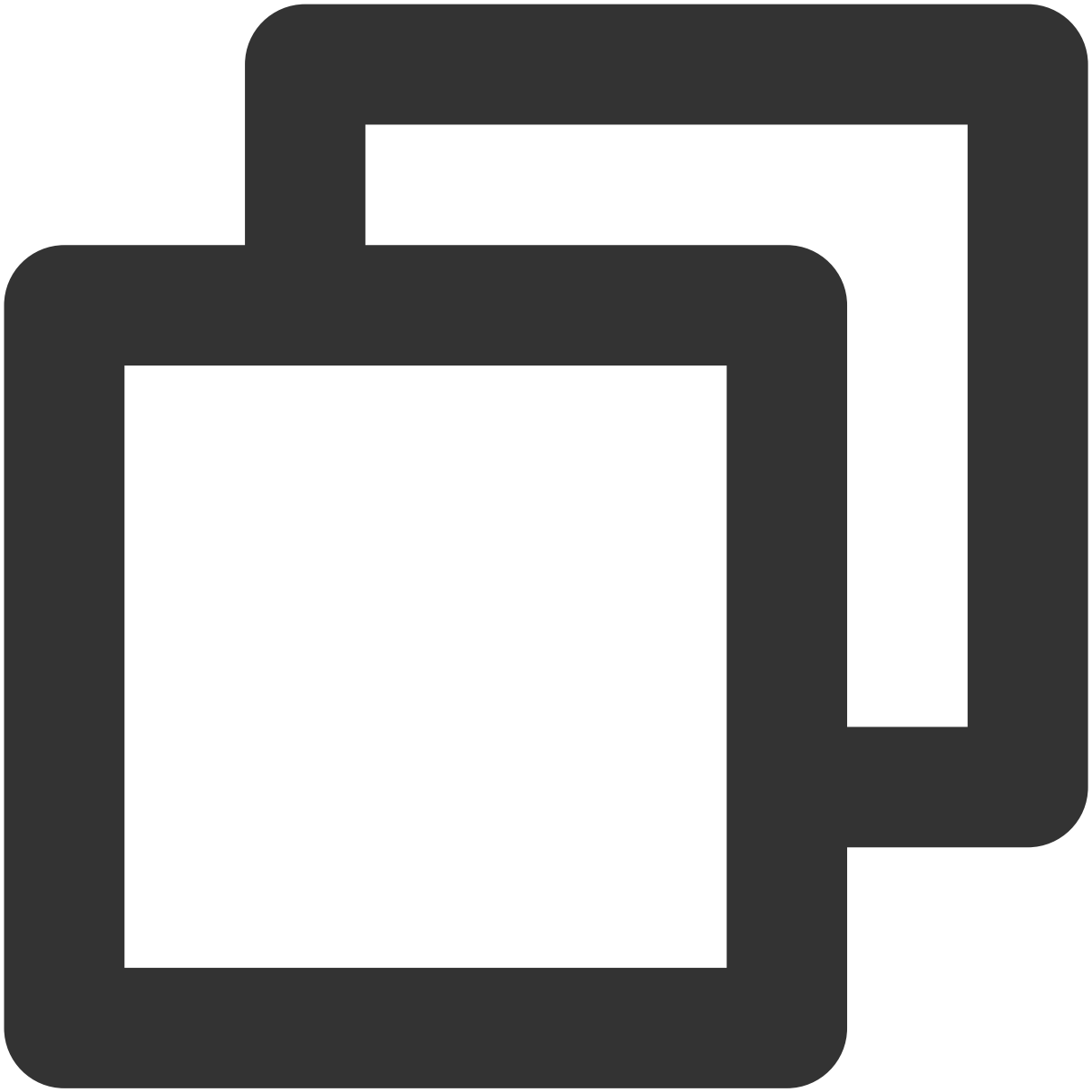
When the message "Apply complete! Resources: 3 added, 0 changed, 0 destroyed." is displayed, the creation is successful.

Viewing the Status of Prometheus Instances

Log in to the TCOP, and choose **Managed Service for Prometheus** from the left-hand navigation bar. Existing instances are displayed in the Prometheus instance list.

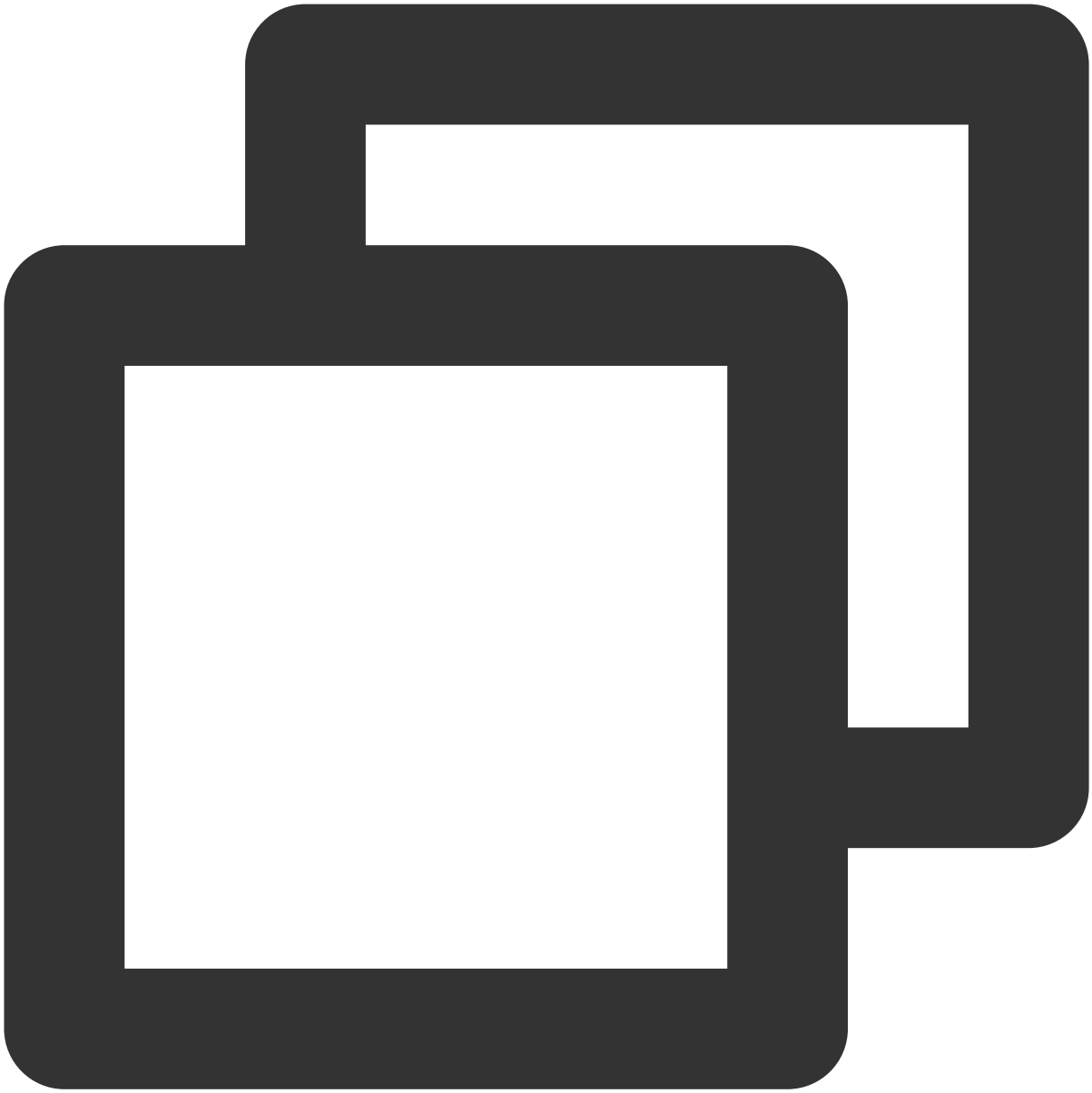
Deleting the Component Integration in the Prometheus Instance Integration Center

To terminate the resources, run the following command:



```
terraform destroy
```

Expected output:



```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  - destroy

Terraform will perform the following actions:
XXX...

Plan: 0 to add, 0 to change, 1 to destroy.
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

Note:

When `Destroy complete! Resources: 1 destroyed.` is displayed, the instance has been successfully deleted.

Collecting Container Monitoring Data Using Terraform

Last updated：2023-12-29 16:10:38

Prerequisites

Installing Terraform

For detailed instructions on installing Terraform, see Installing Terraform.

Note:

The installed version of Terraform must not be below v1.6.3. You can verify the version of Terraform installed by using the command terraform --version.

Configuring Tencent Cloud Account Information

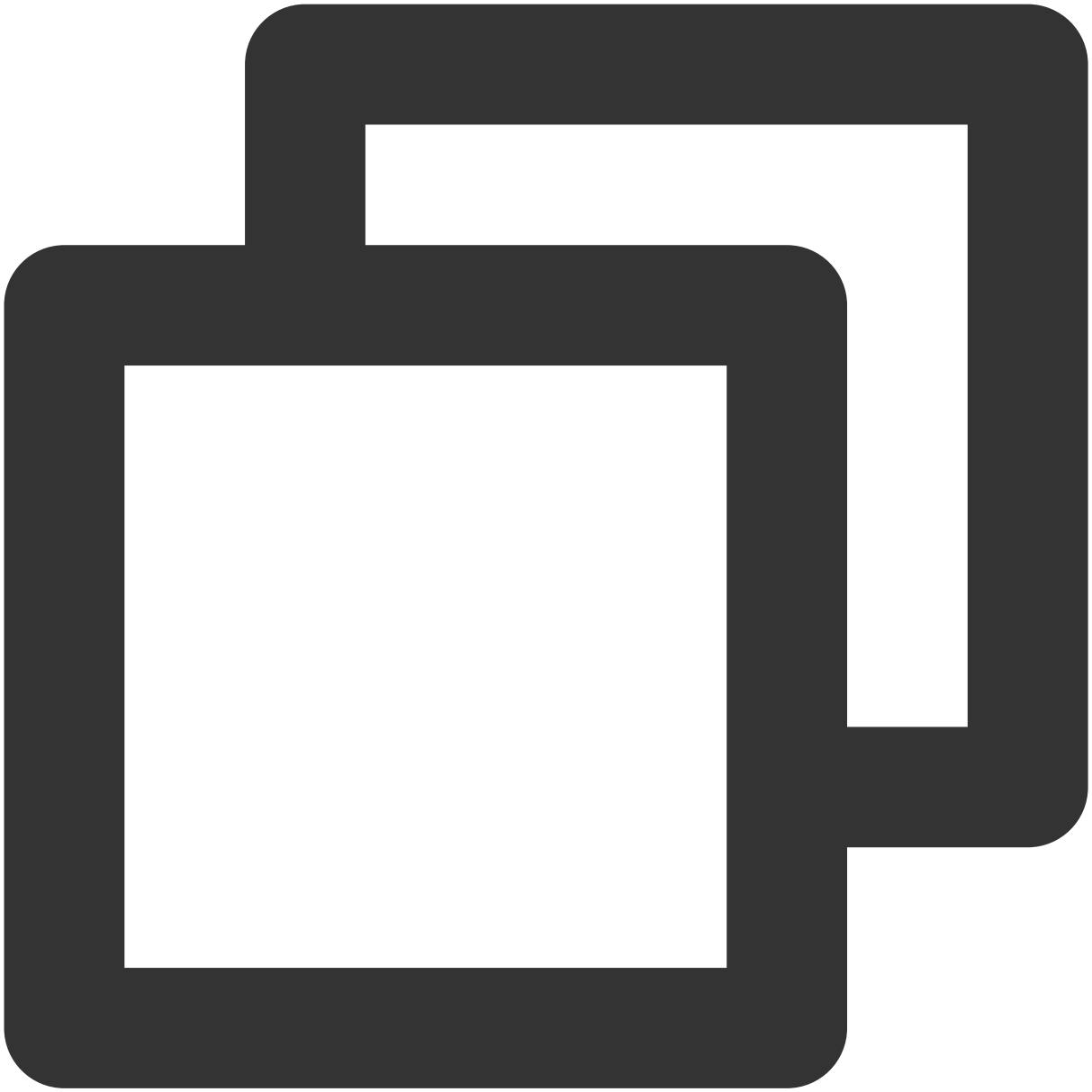
Before using Terraform for the first time, go to API Key to apply for a SecretID and SecretKey of the security credentials. If you already have valid credentials, skip this step.

1. Log in to the CAM Console and choose **Access Key > API Keys** from the left sidebar.
2. On the API Keys page, click **Create Key** to create a SecretId/SecretKey pair.

There are two methods for configuring Tencent Cloud account information:

Authentication by Static Credentials

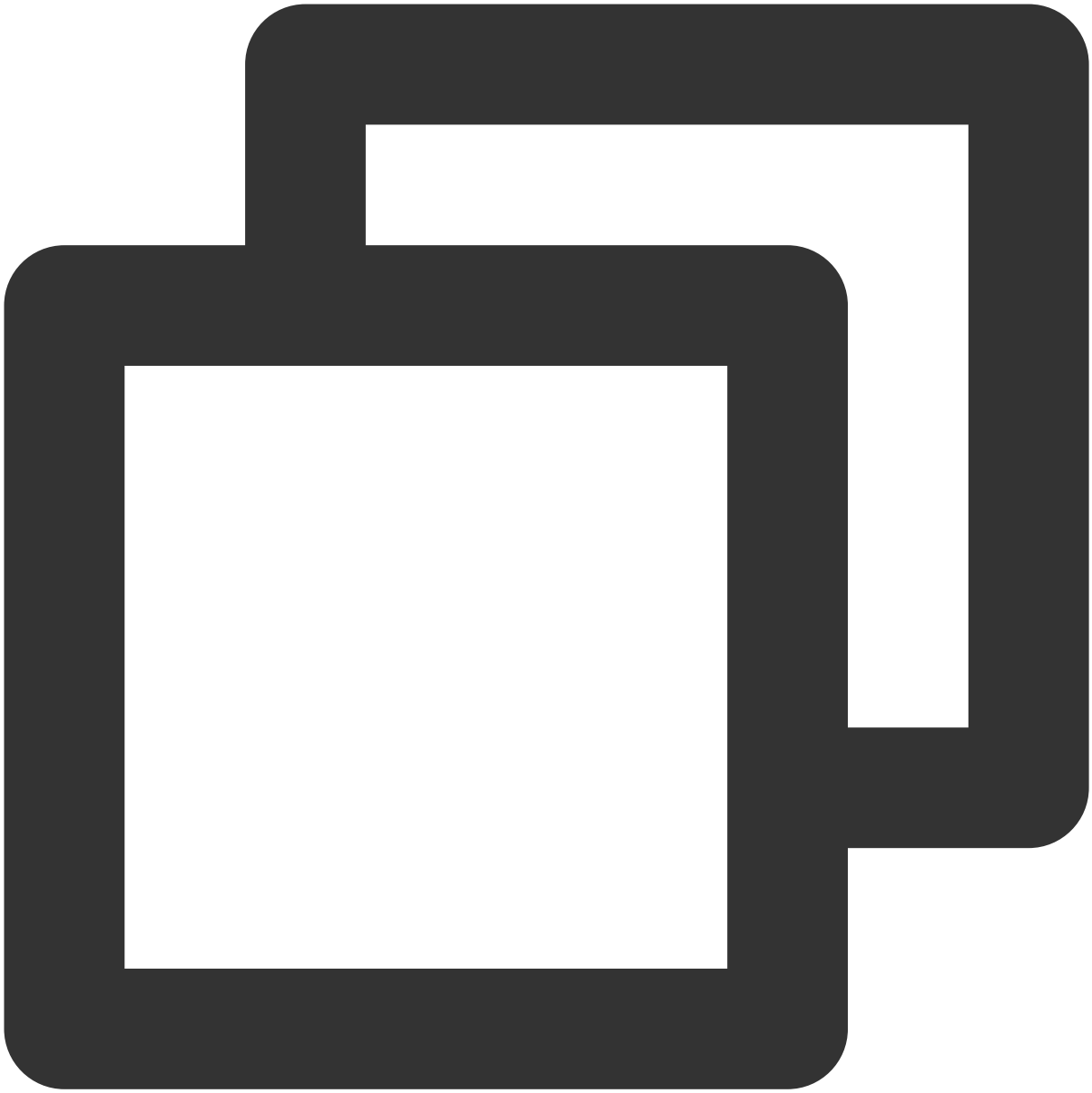
Create a `provider.tf` file in the user directory with the following content. Replace `my-secret-id` and `my-secret-key` with your SecretId and SecretKey, respectively.



```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

Authentication by Environment Variables

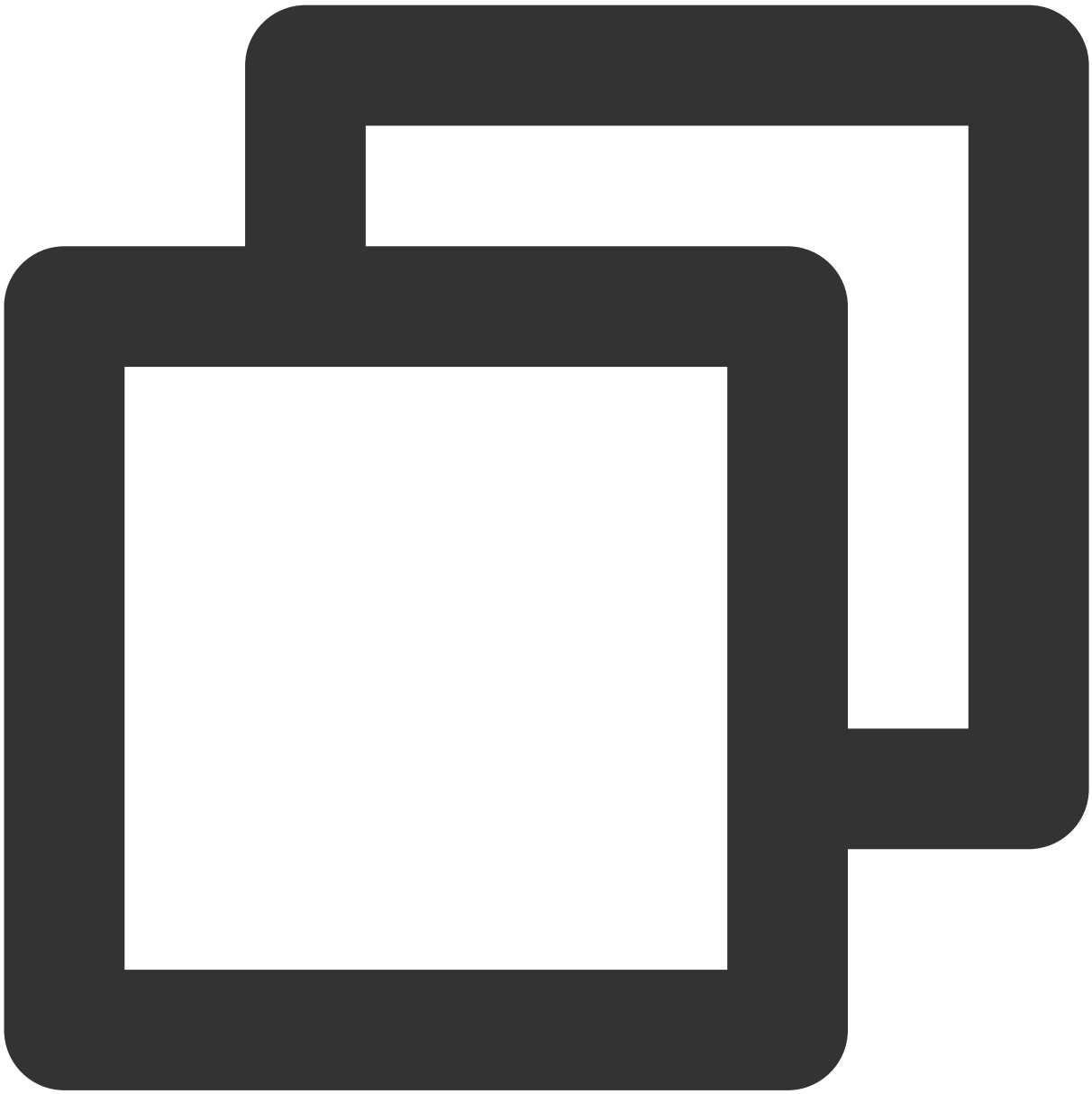
Set the computer variables or cloud environment variables by running the command below. Replace `YOUR_SECRET_ID` and `YOUR_SECRET_KEY` with your SecretId and SecretKey, respectively.



```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

Adding Container Monitoring Data Collection to Prometheus Instances

1. Create a new Terraform configuration file: Create a new directory and a file named main.tf in the directory. The following shows the configuration:



```
#Specifying Provider Configuration Information

terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
```

```
  }
}
#Prometheus Managing Cloud Monitoring Integration

#Prometheus Management of Container Clusters (Collection Containers)

resource "tencentcloud_monitor_tmp_tke_cluster_agent" "foo" {
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  agents {
    region          = "ap-mumbai"
    cluster_type    = "eks"
    cluster_id      = "cls-1uary7z2" #ID of the cluster that needs to be associated
    enable_external = false
  }
}
```

Note:

To create the Integration Center component of a Prometheus instance, the corresponding fields must be configured as follows:

instance_id: (Required, string, ForceNew) Instance ID.

agents: (Required, list) List of agents.

cluster_id: (Required, string) Identifier for the cluster ID.

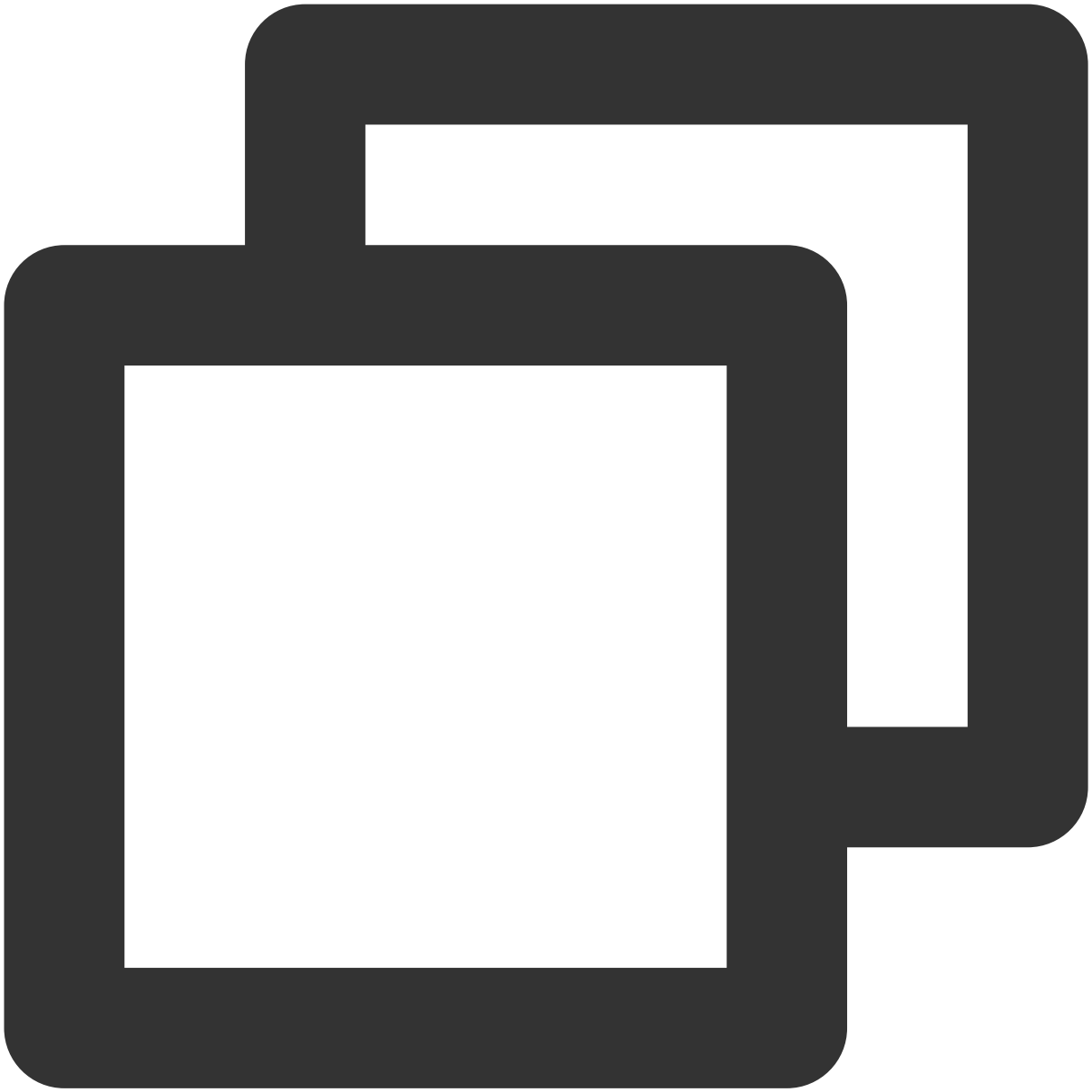
region: (Required, string) Region restrictions.

enable_external: (Required, Bool) Whether to enable public network CLB.

cluster_type: (Required, string) Cluster type.

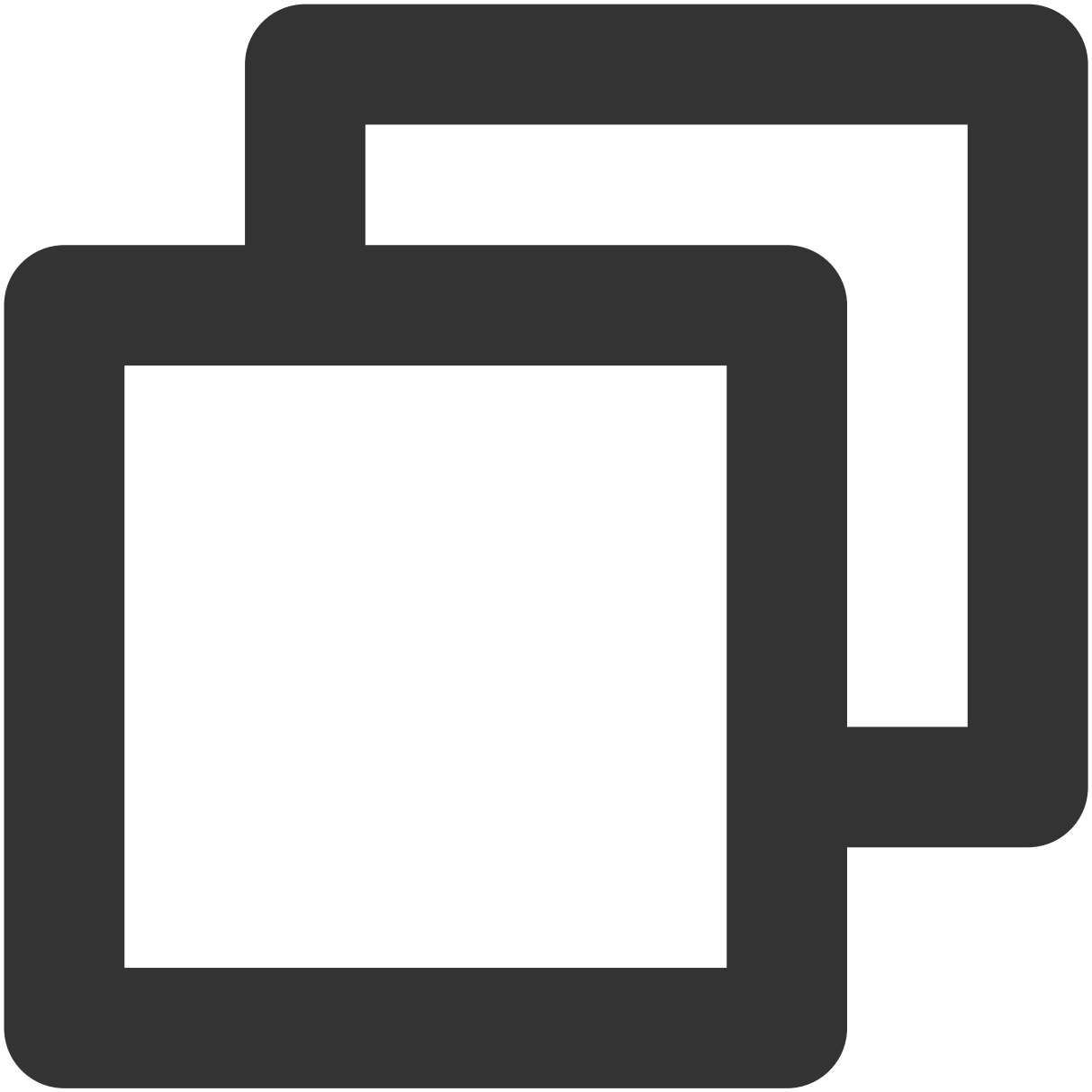
For more details about parameters, see **Detailed Parameters**. You can also see Tencent Cloud Provider on Github and Terraform Tencent Cloud Provider.

2. To initialize the Terraform runtime environment, run the following command:



```
terraform init
```

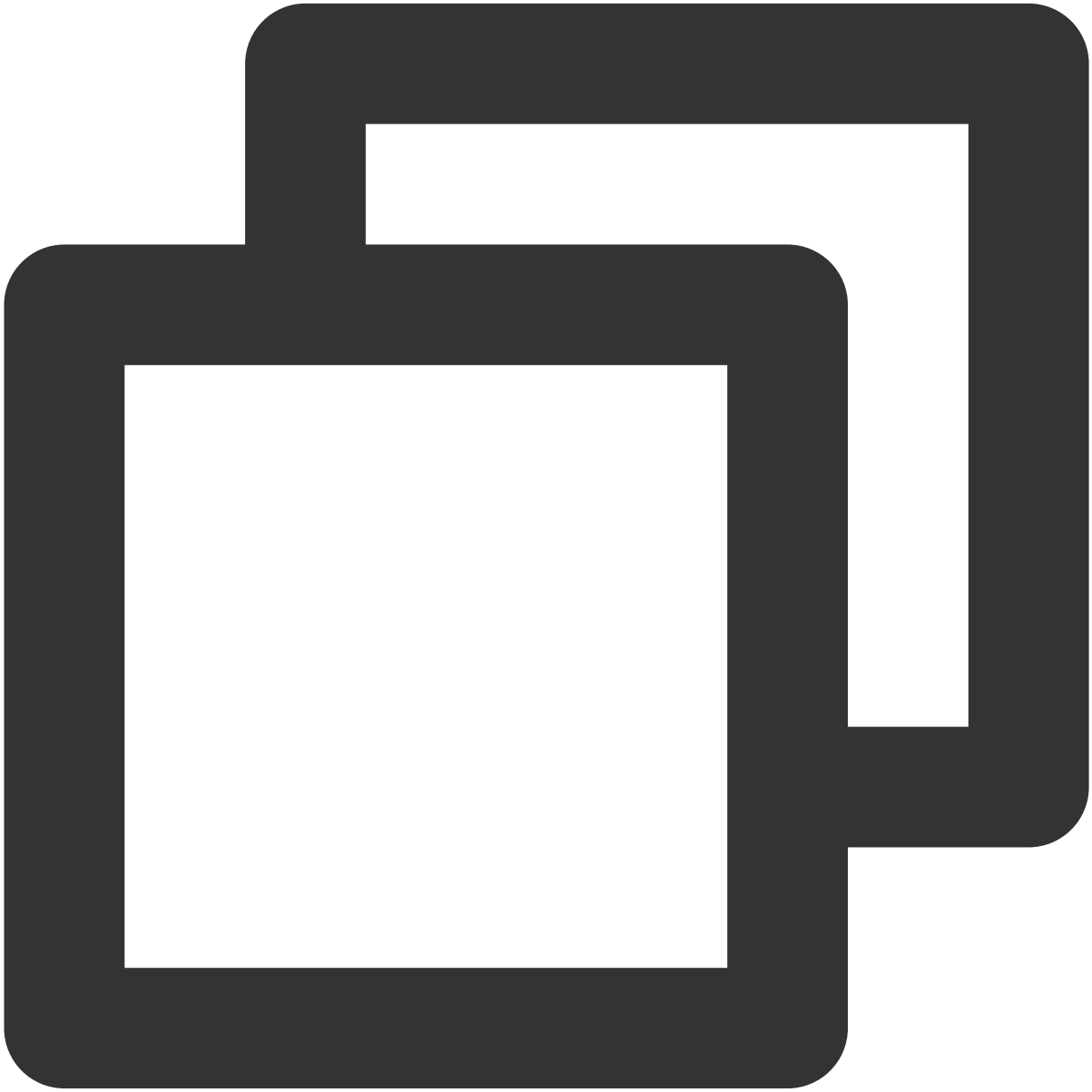
Expected output:



```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lo
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

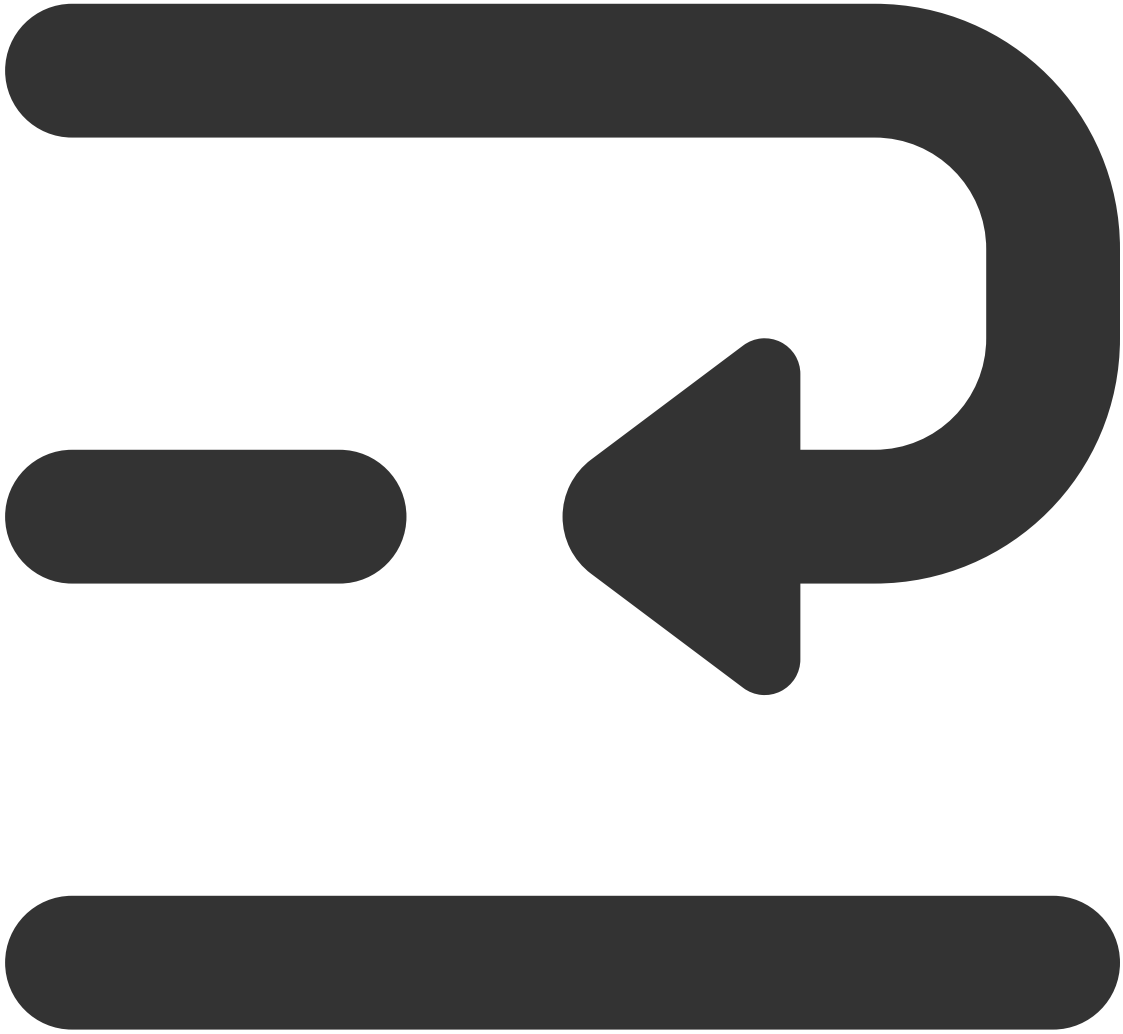
```
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

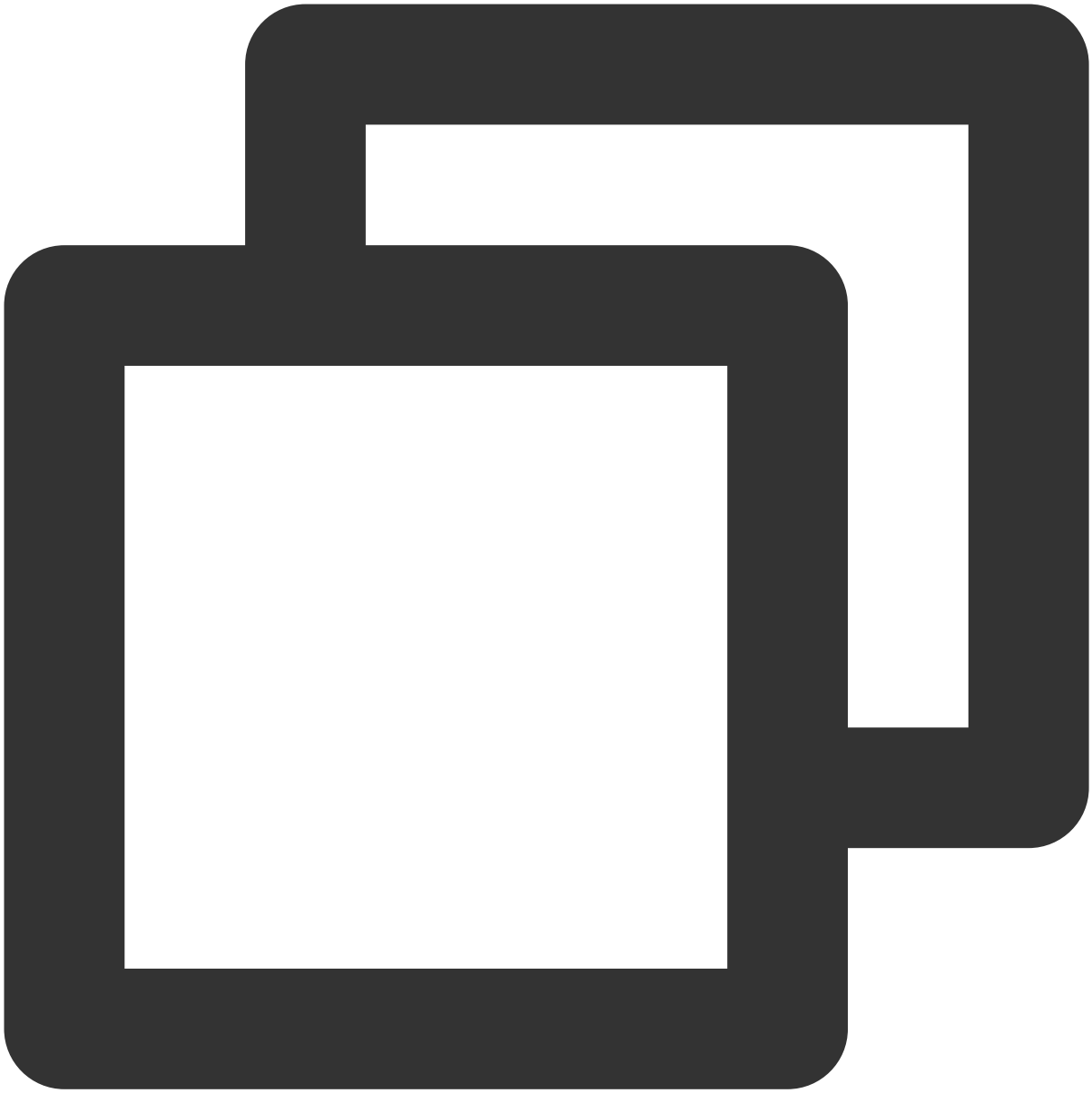
3. To generate a resource plan, run the following command:



```
terraform plan
```

Expected output:

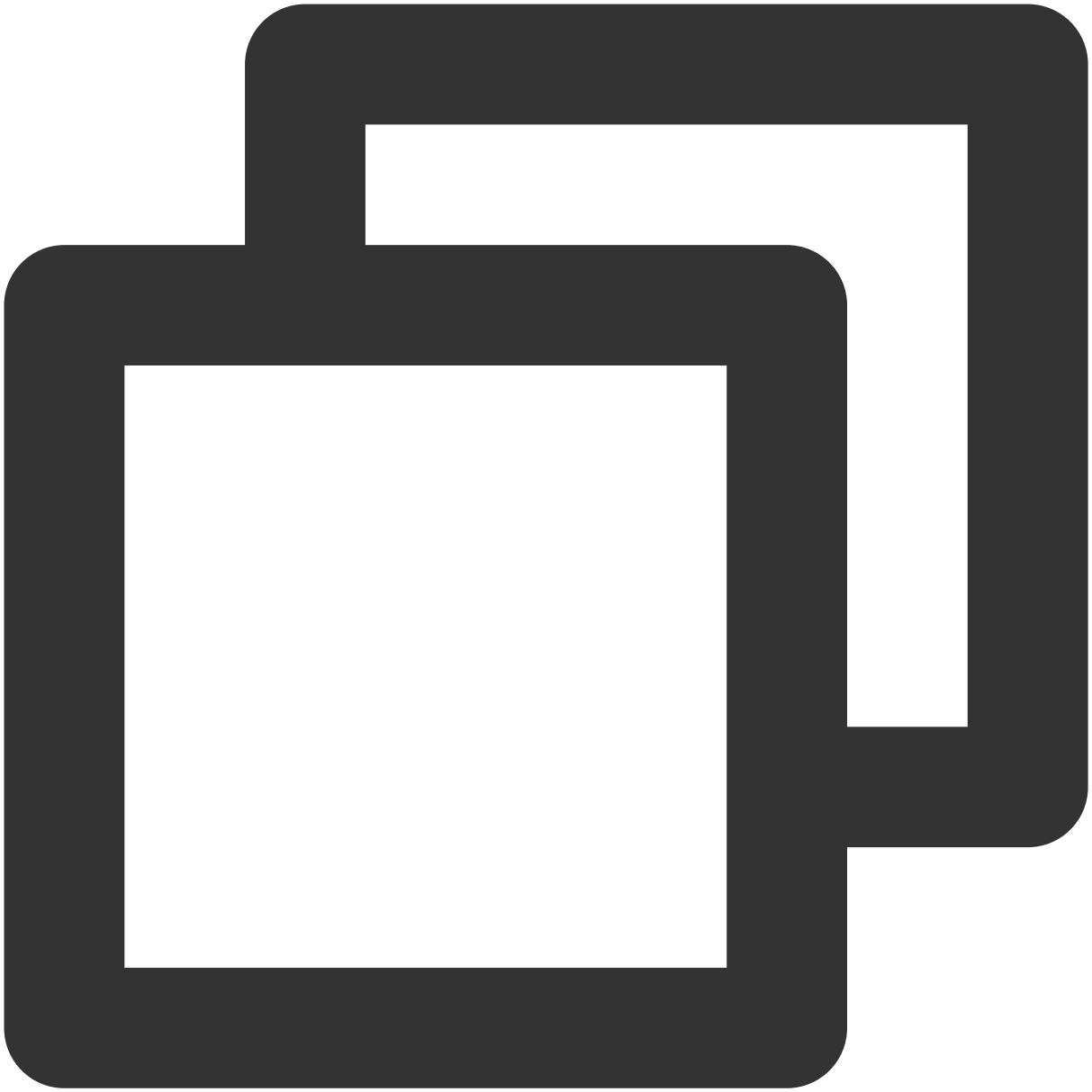




```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing st
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state.

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_tke_cluster_agent.foo will be created
```

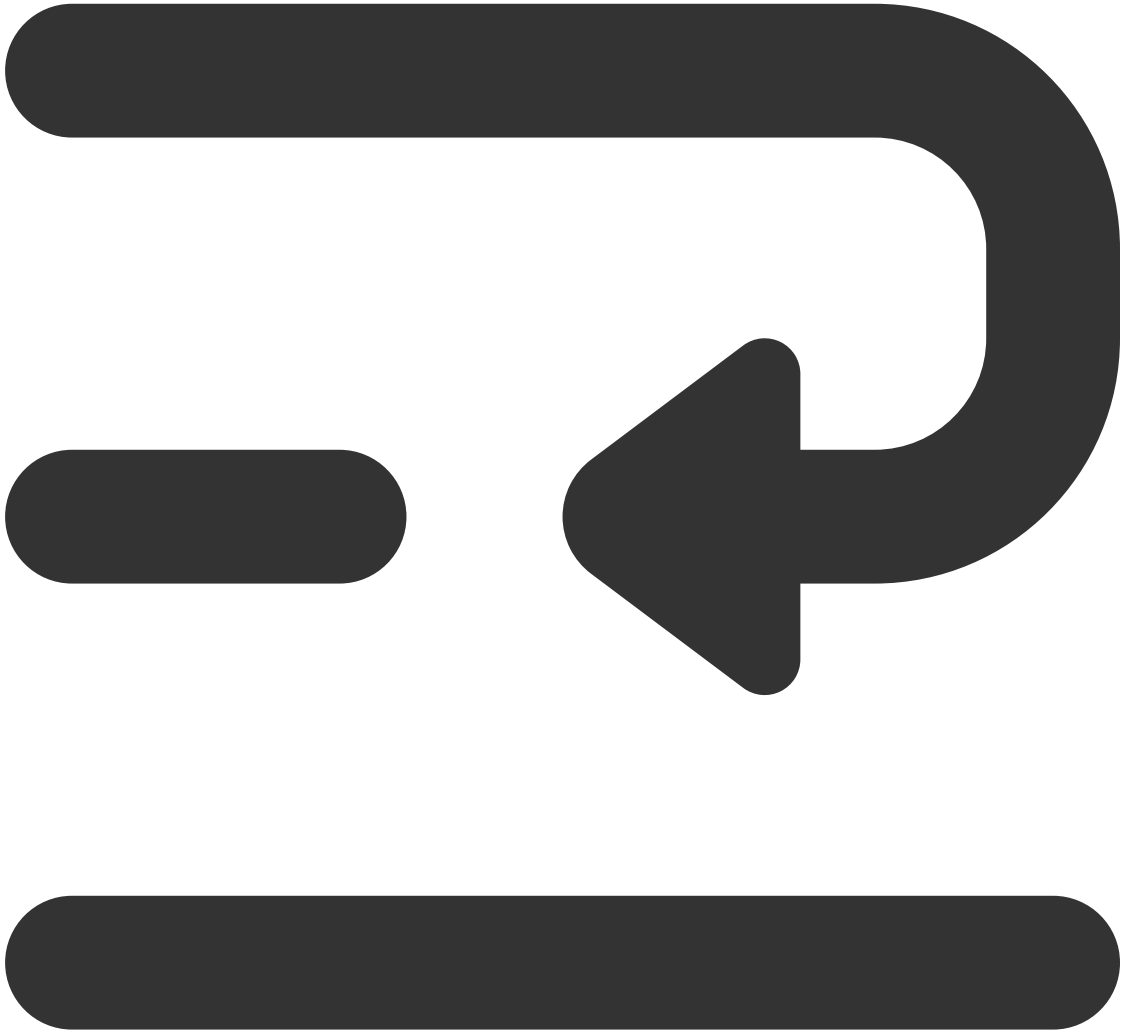
```
  + resource "tencentcloud_monitor_tmp_tke_cluster_agent" "foo" {
      + id          = (known after apply)
      + instance_id = "prom-jh0zntj2"

      + agents {
          + cluster_id      = "cls-1uary7z2"
          + cluster_name    = (known after apply)
          + cluster_type    = "eks"
          + enable_external = false
          + region          = "ap-mumbai"
          + status          = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

─────────────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarante
you run "terraform apply" now.
```

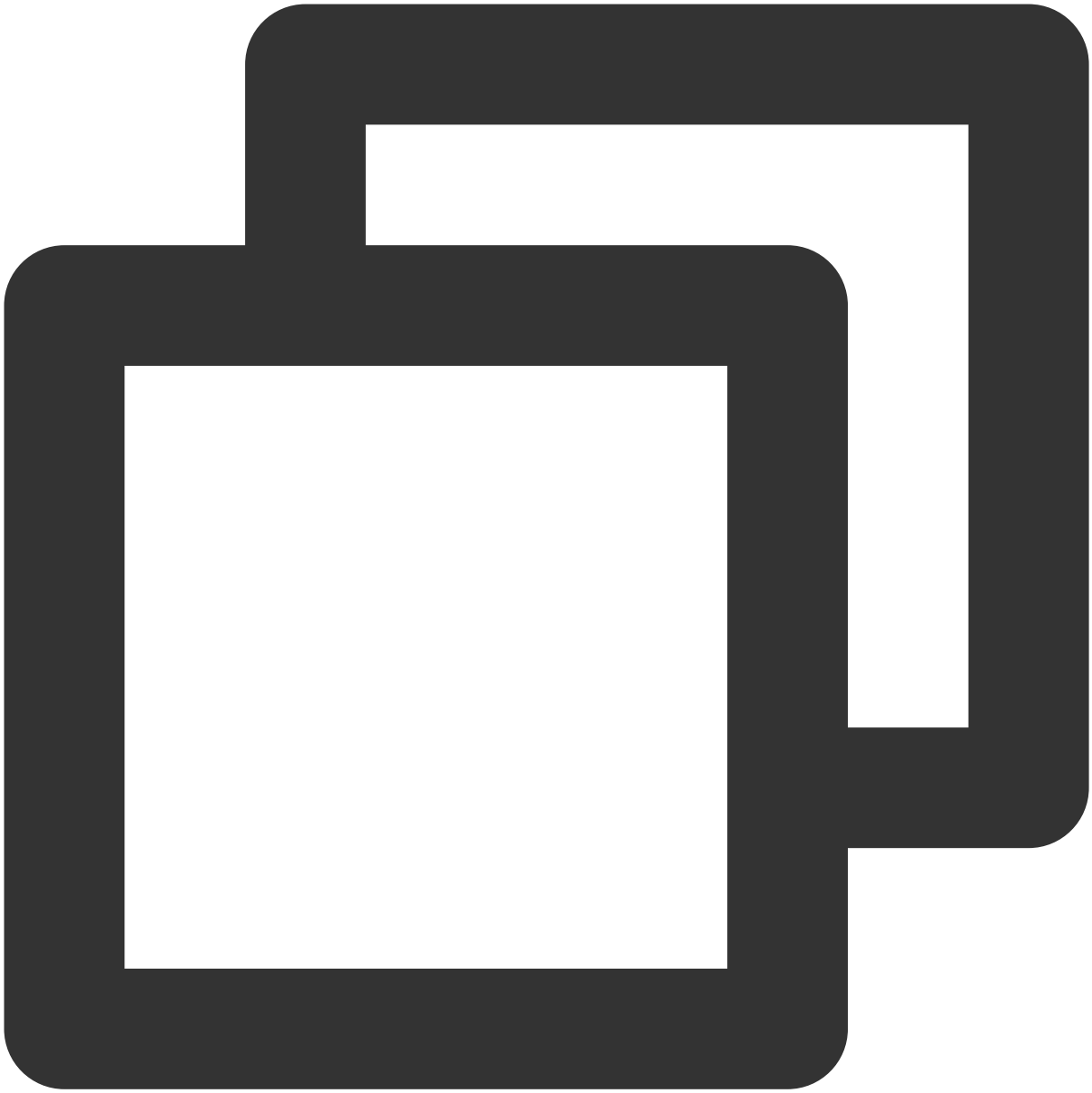
4. To create component integration of the Integration Center, run the following command:



```
terraform apply
```

Expected output:





```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing st
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state.

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:

  # tencentcloud_monitor_tmp_tke_cluster_agent.foo will be created
```

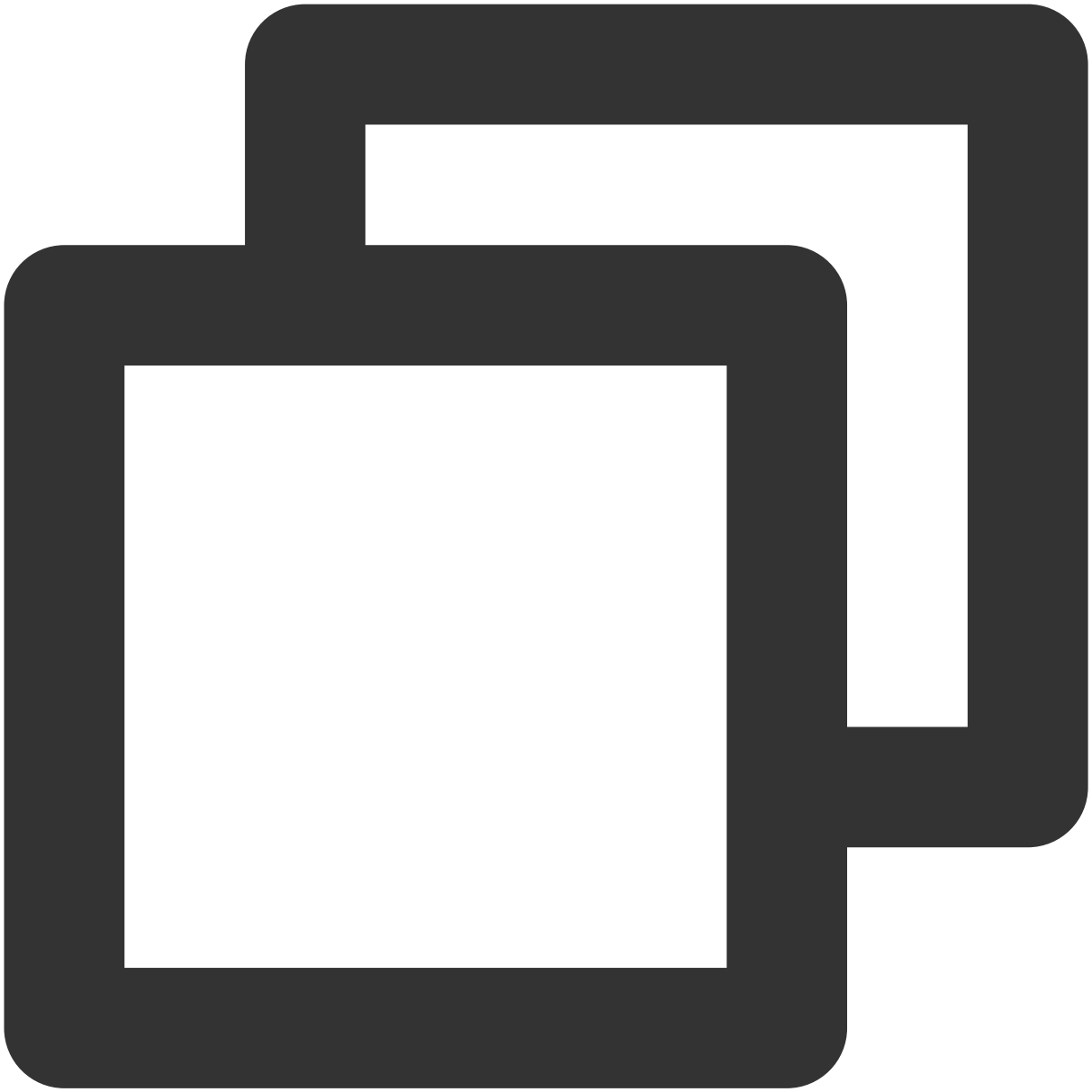
```
  + resource "tencentcloud_monitor_tmp_tke_cluster_agent" "foo" {
      + id          = (known after apply)
      + instance_id = "prom-jh0zntj2"

      + agents {
          + cluster_id      = "cls-1uary7z2"
          + cluster_name    = (known after apply)
          + cluster_type    = "eks"
          + enable_external = false
          + region          = "ap-mumbai"
          + status          = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tencentcloud_monitor_tmp_tke_cluster_agent.foo: Creating...
Instance content...

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Viewing the Status of the Prometheus Instance

Log in to the TCOP, and choose **Managed Service for Prometheus** from the left-hand navigation bar. Existing instances are displayed in the Prometheus instance list.

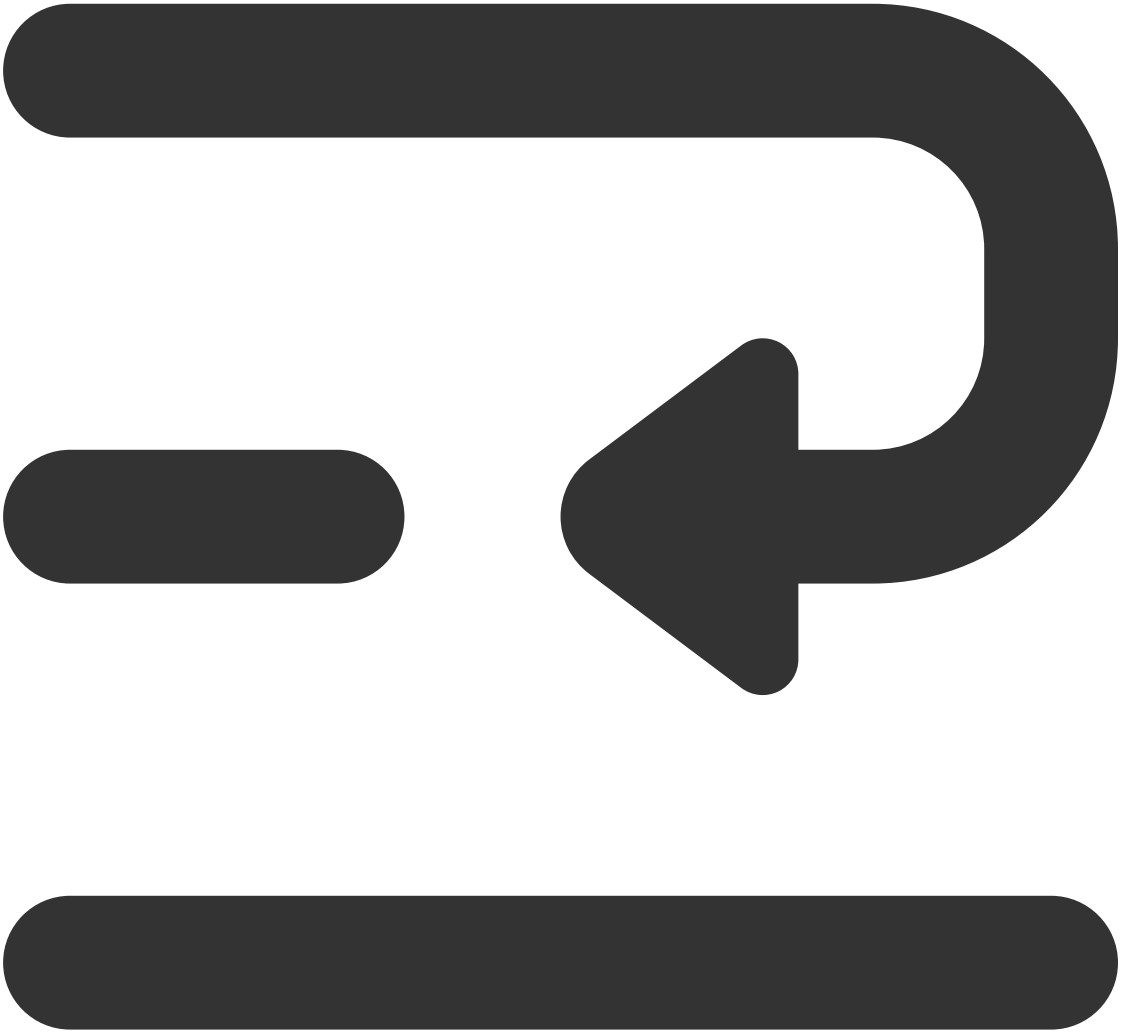
Deleting the Component Integration in the Prometheus Instance Integration Center

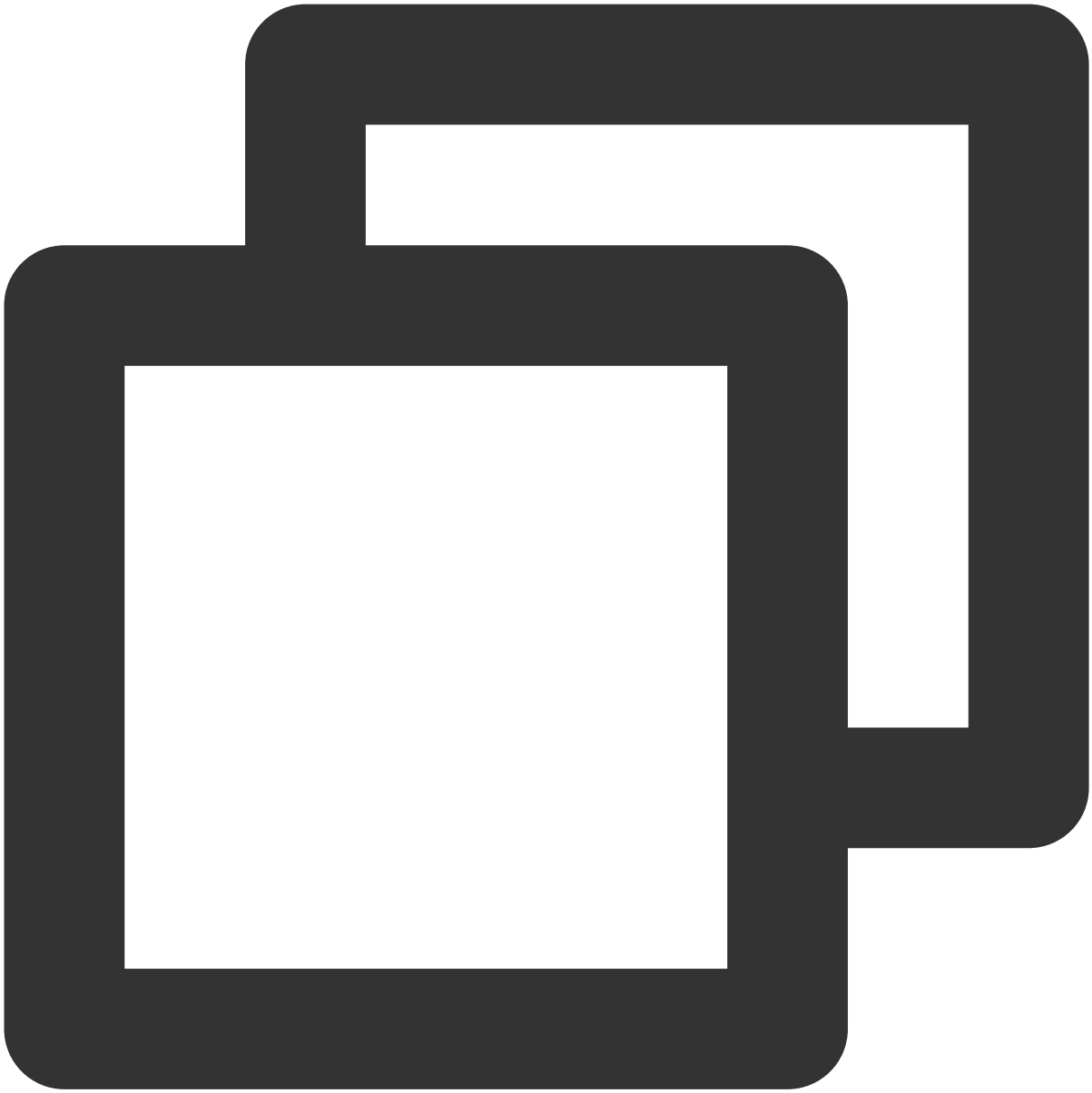
To terminate the resources, run the following command:



```
terraform destroy
```

Expected output:





```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  - destroy

Terraform will perform the following actions:
Instance content...

Plan: 0 to add, 0 to change, 1 to destroy.
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

Note:

If `Destroy complete! Resources: (numbers of existing instances) destroyed.` is displayed, the instance has been deleted.

Configuring Alarm Policies Using Terraform

Last updated：2023-12-29 16:11:24

Prerequisites

Installing Terraform

For detailed instructions on installing Terraform, see Installing Terraform.

Note:

The installed version of Terraform must not be below v1.6.3. You can verify the version of Terraform installed by using the command terraform --version.

Configuring Tencent Cloud Account Information

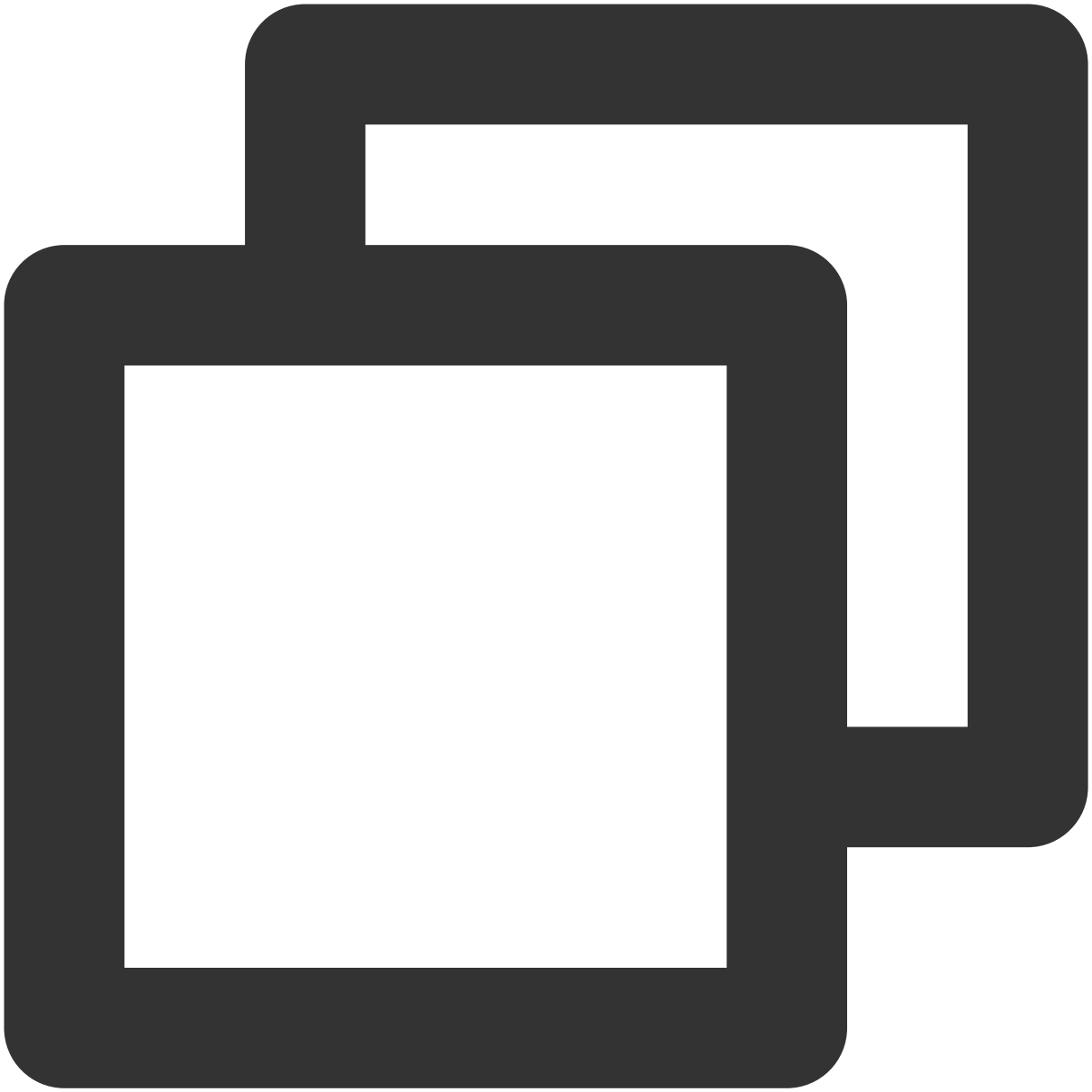
Before using Terraform for the first time, go to API Key to apply for a SecretID and SecretKey of the security credentials. If you already have valid credentials, skip this step.

1. Log in to the CAM Console and choose **Access Key > API Keys** from the left sidebar.
2. On the API Keys page, click **Create Key** to create a SecretId/SecretKey pair.

There are two methods for configuring Tencent Cloud account information:

Authentication by Static Credentials

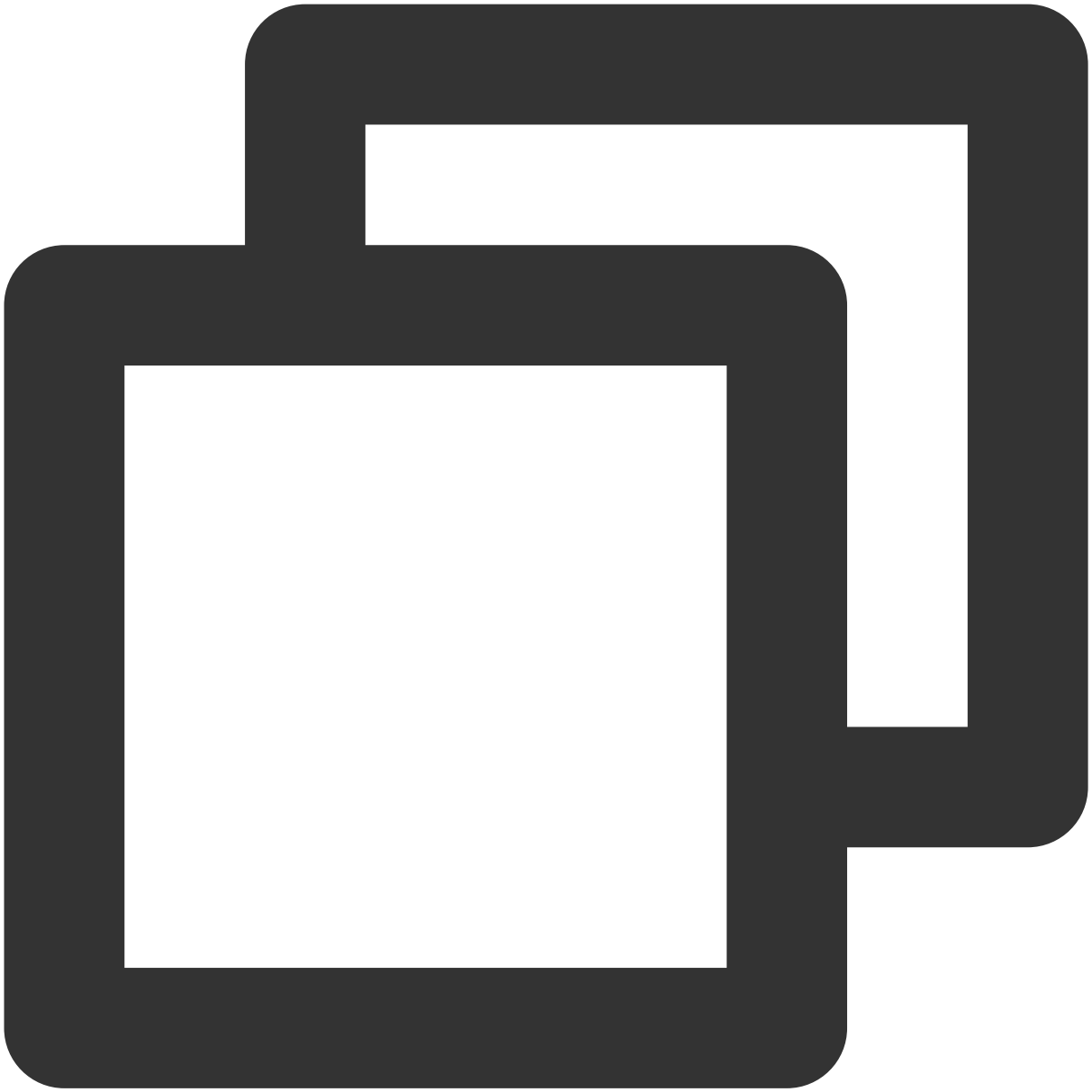
Create a `provider.tf` file in the user directory with the following content. Replace `my-secret-id` and `my-secret-key` with your SecretId and SecretKey, respectively.



```
provider "tencentcloud" {
  secret_id = "my-secret-id"
  secret_key = "my-secret-key"
}
```

Authentication by Environment Variables

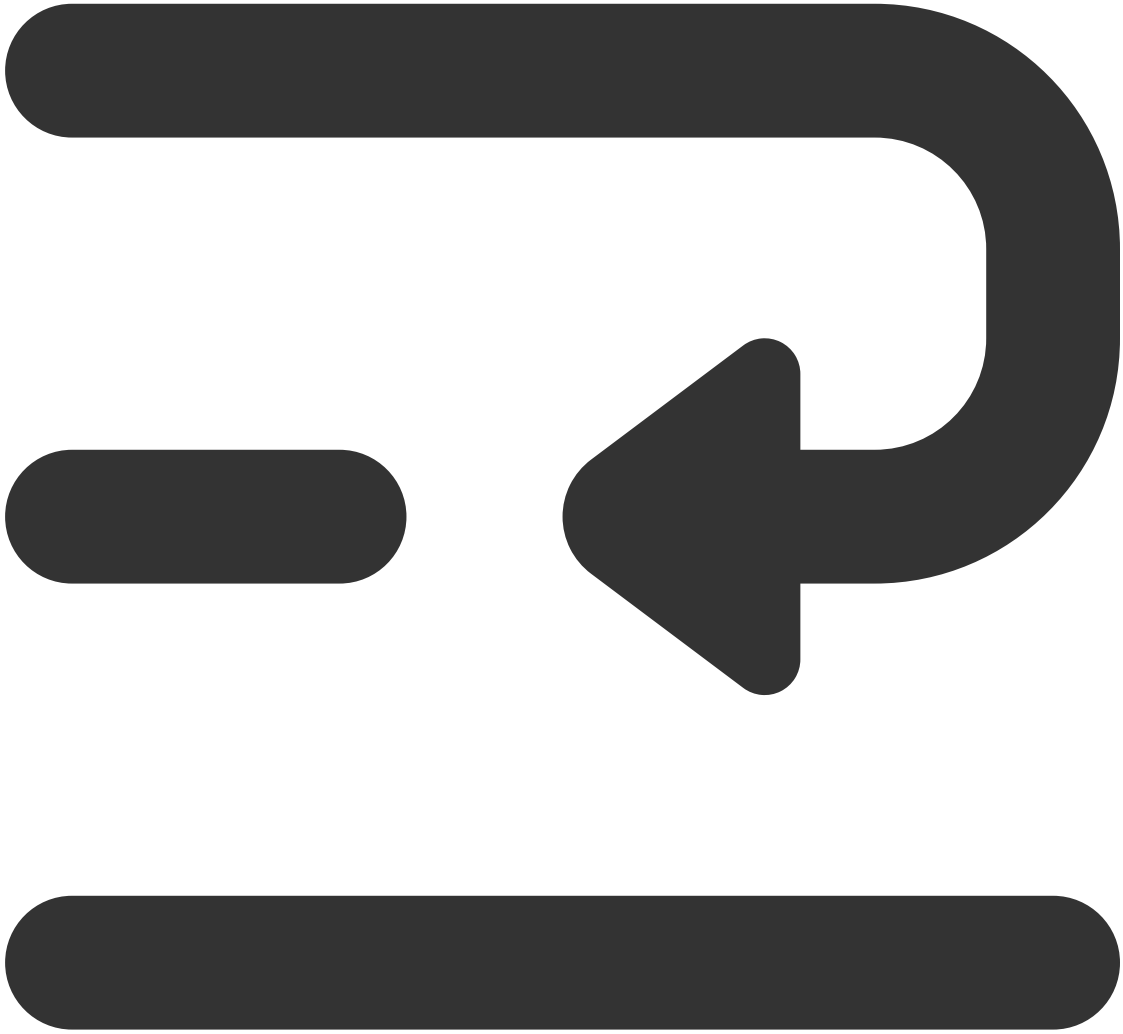
Configure the computer environment variables or cloud environment variables by running the following command. Replace `YOUR_SECRET_ID` and `YOUR_SECRET_KEY` with your SecretId and SecretKey, respectively.

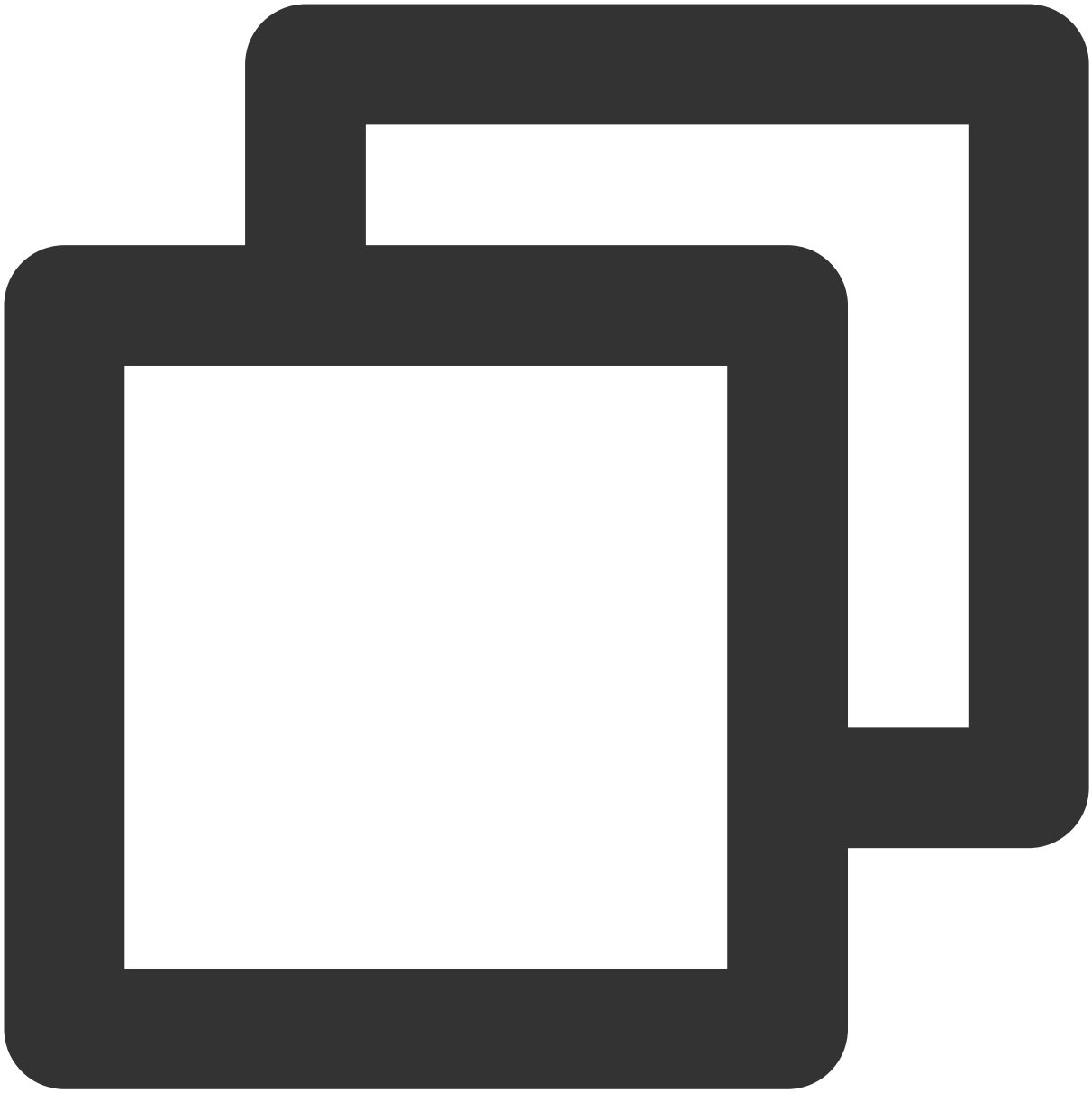


```
export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID
export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY
```

Adding a Terraform Alarm Policy to a Prometheus Instance

1. Create a new Terraform configuration file. Create a new directory and a file named main.tf in the directory. The following is the configuration information:





```
#Specifying Provider Configuration Information

terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}

#Configuring Alarm for Prometheus (Cloud Monitoring Side Configuration)
```

```
resource "tencentcloud_monitor_tmp_alert_rule" "foo" {
  duration    = "2m"
  expr        = "avg by (instance) (mysql_global_status_threads_connected) / avg by
  instance_id = tencentcloud_monitor_tmp_instance.foo.id
  receivers = ["notice-zmjsavnp"] # Notifications templates can be created here usi
  rule_name = "MySQL Excessive Connections--tf-Cloud Monitor Test"
  rule_state  = 2
  type = "MySQL/Excessive MySQL Connections"

  annotations {
    key   = "description"
    value = "Too many MySQL connections, instance: {{$labels.instance}}, current va
  }
  annotations {
    key   = "summary"
    value = "MySQL has too many connections(>80%)"
  }

  labels {
    key   = "severity"
    value = "warning"
  }
}
```

Note:

The fields of the configuration are as follows:

duration: Rule persistence duration.

expr: Alarm expression.

instance_id: Instance ID.

receivers: List of alarm recipients.

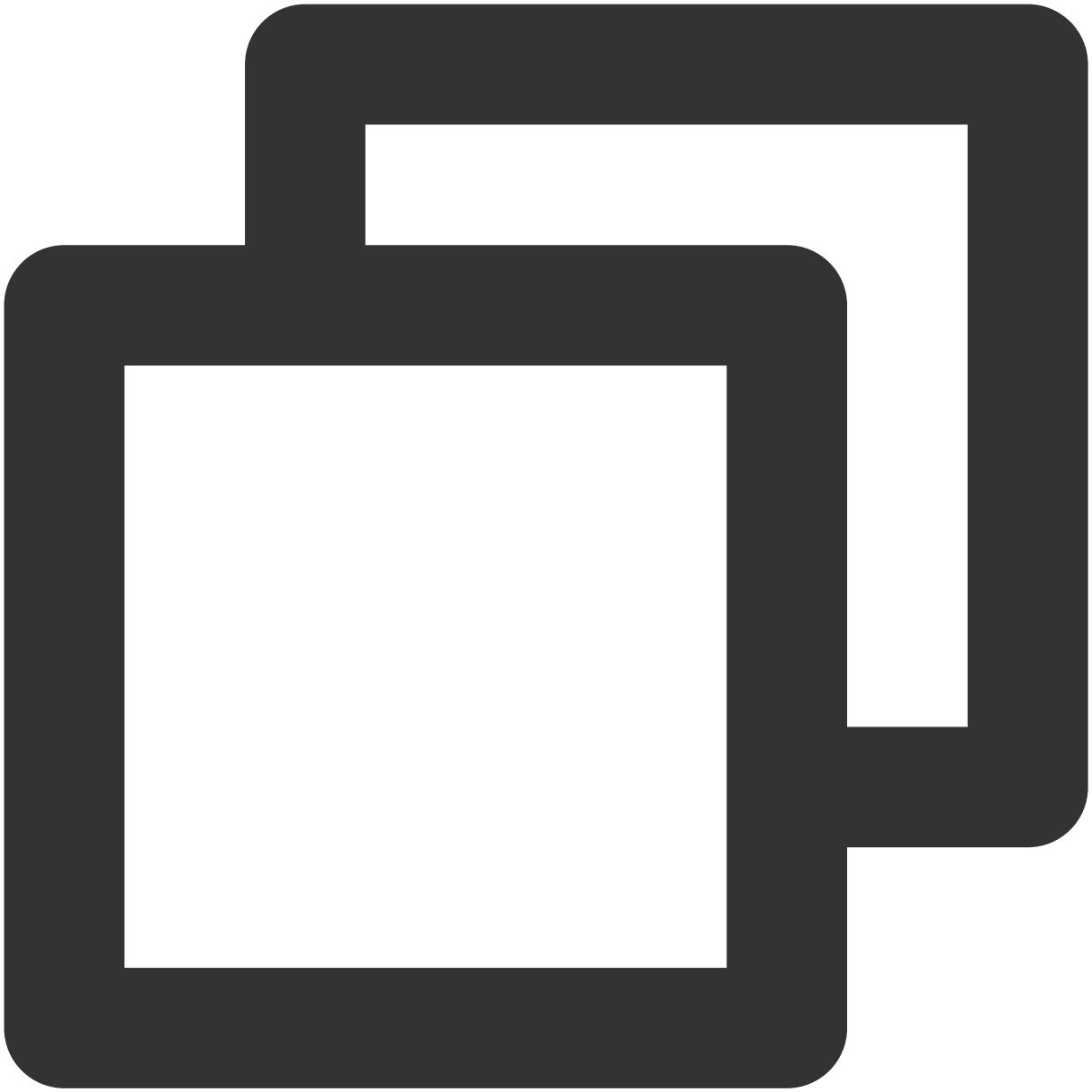
rule_name: Alarm rule name.

rule_state: Alarm rule status.

type: Alarm rule type.

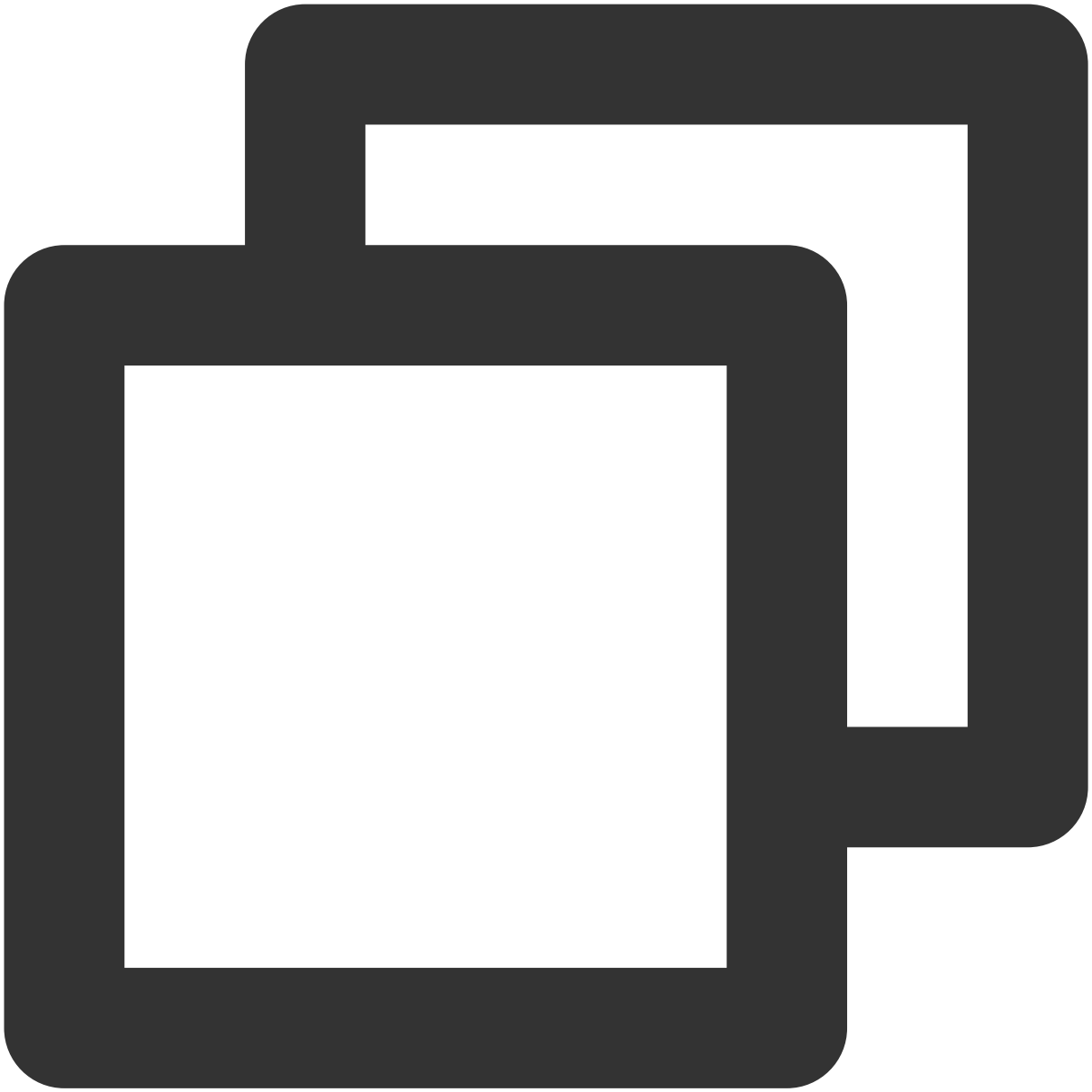
For more detailed parameters, see Tencent Cloud Integration to GitHub or Terraform Tencent Cloud Provider.

2. To initialize the Terraform runtime environment, run the following command:



```
terraform init
```

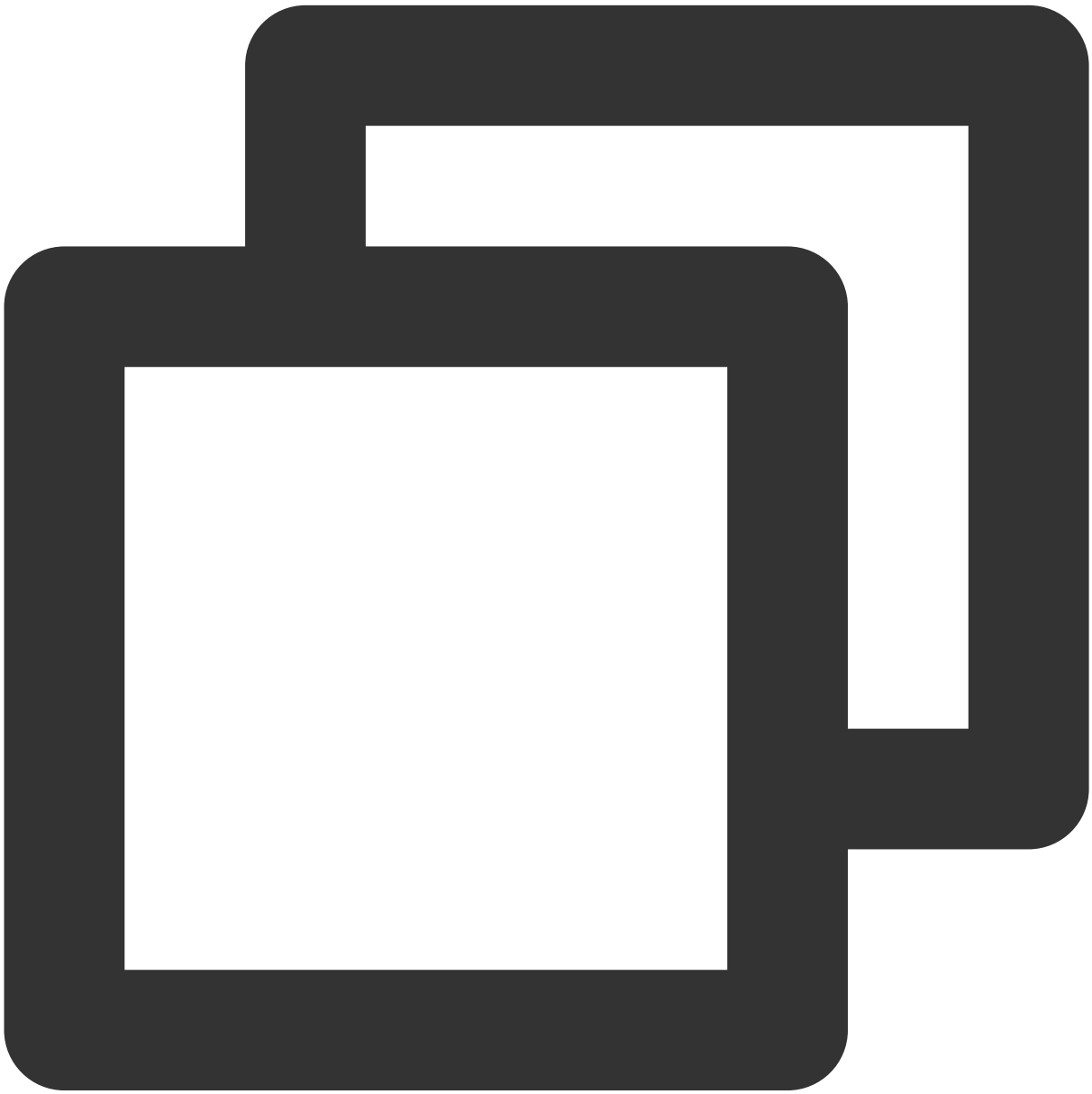
Expected output:



```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of tencentcloudstack/tencentcloud from the dependency lo
- Using previously-installed tencentcloudstack/tencentcloud v1.81.32

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
```

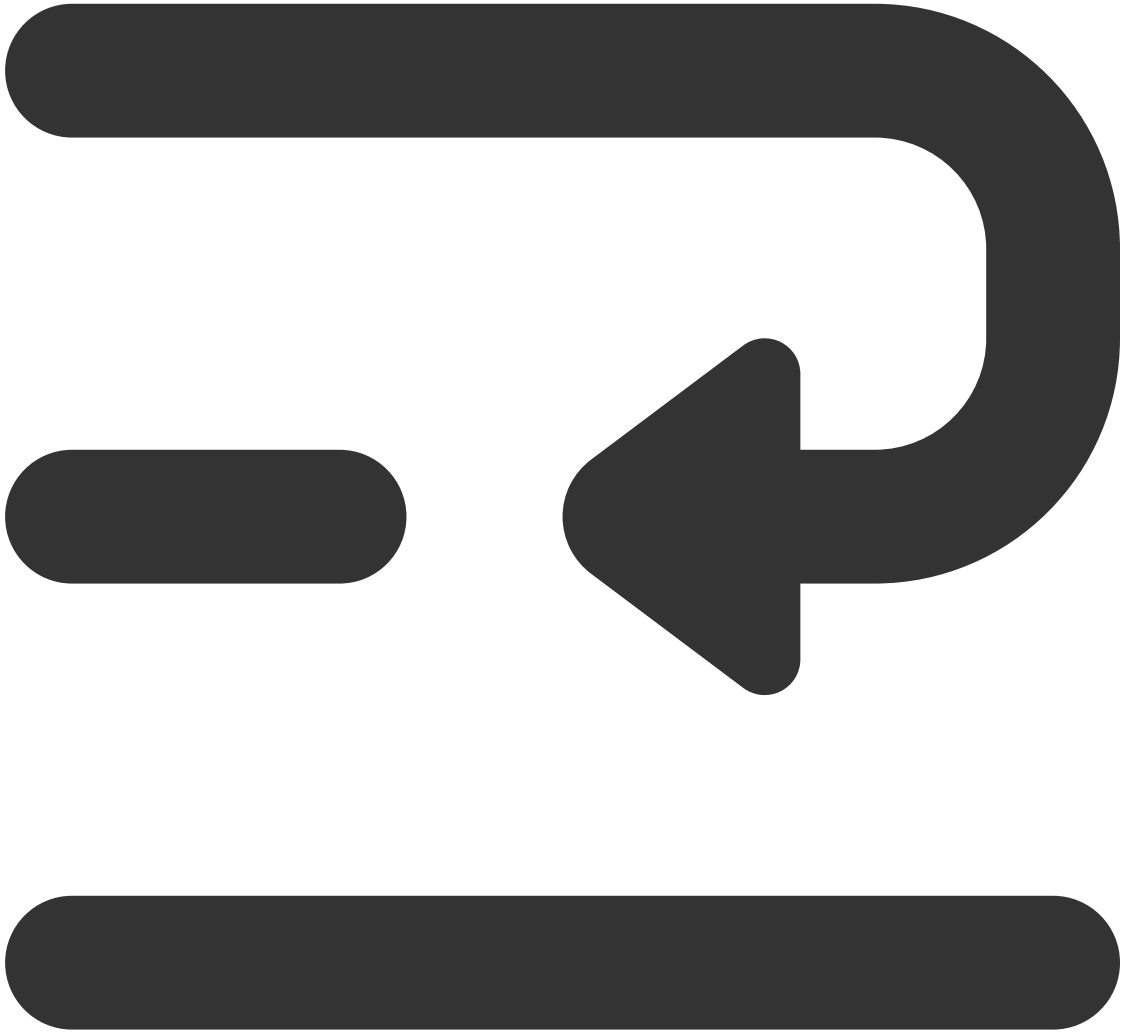
```
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

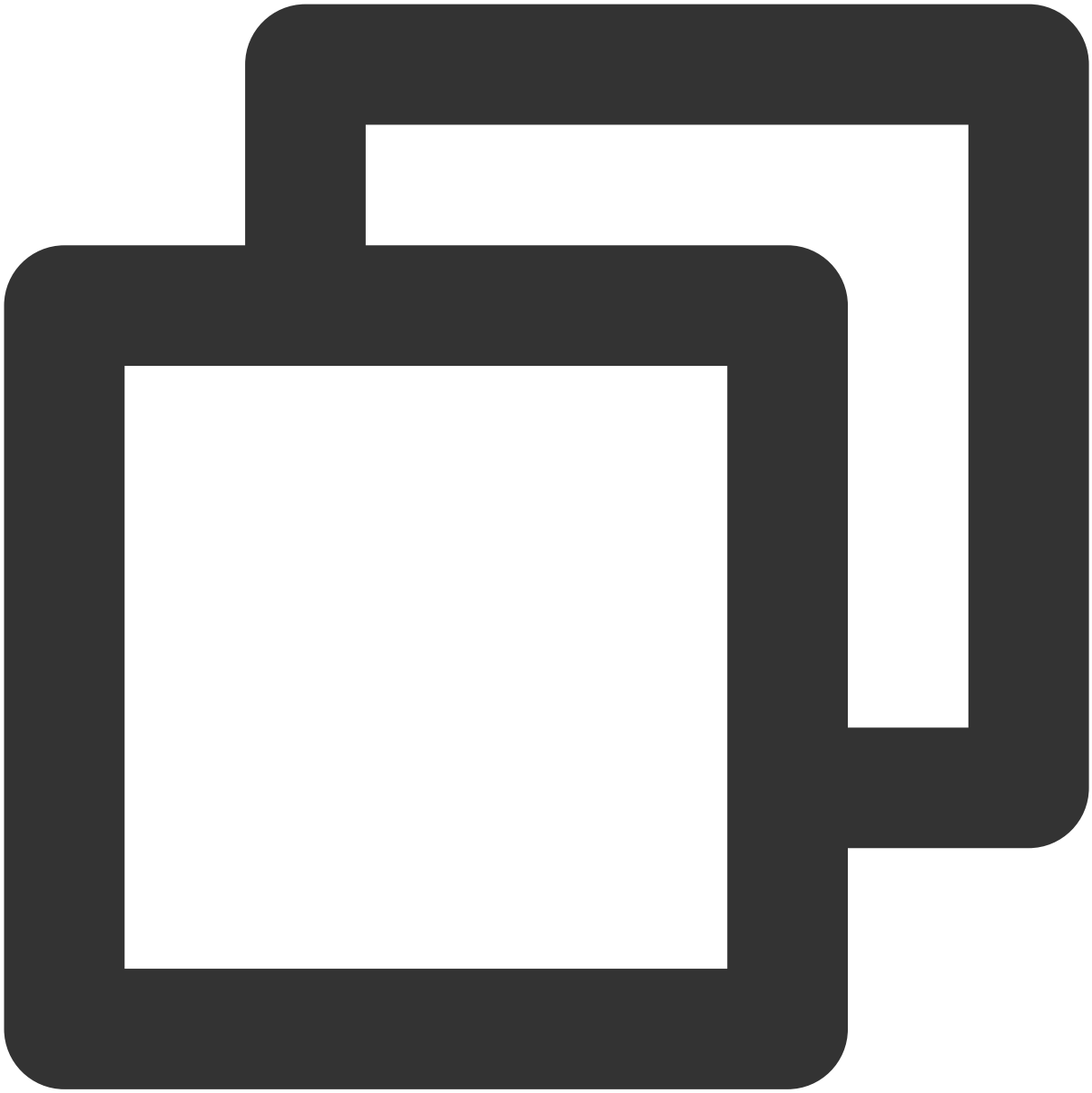
3. To generate a resource plan, run the following command:



```
terraform plan
```

Expected output:

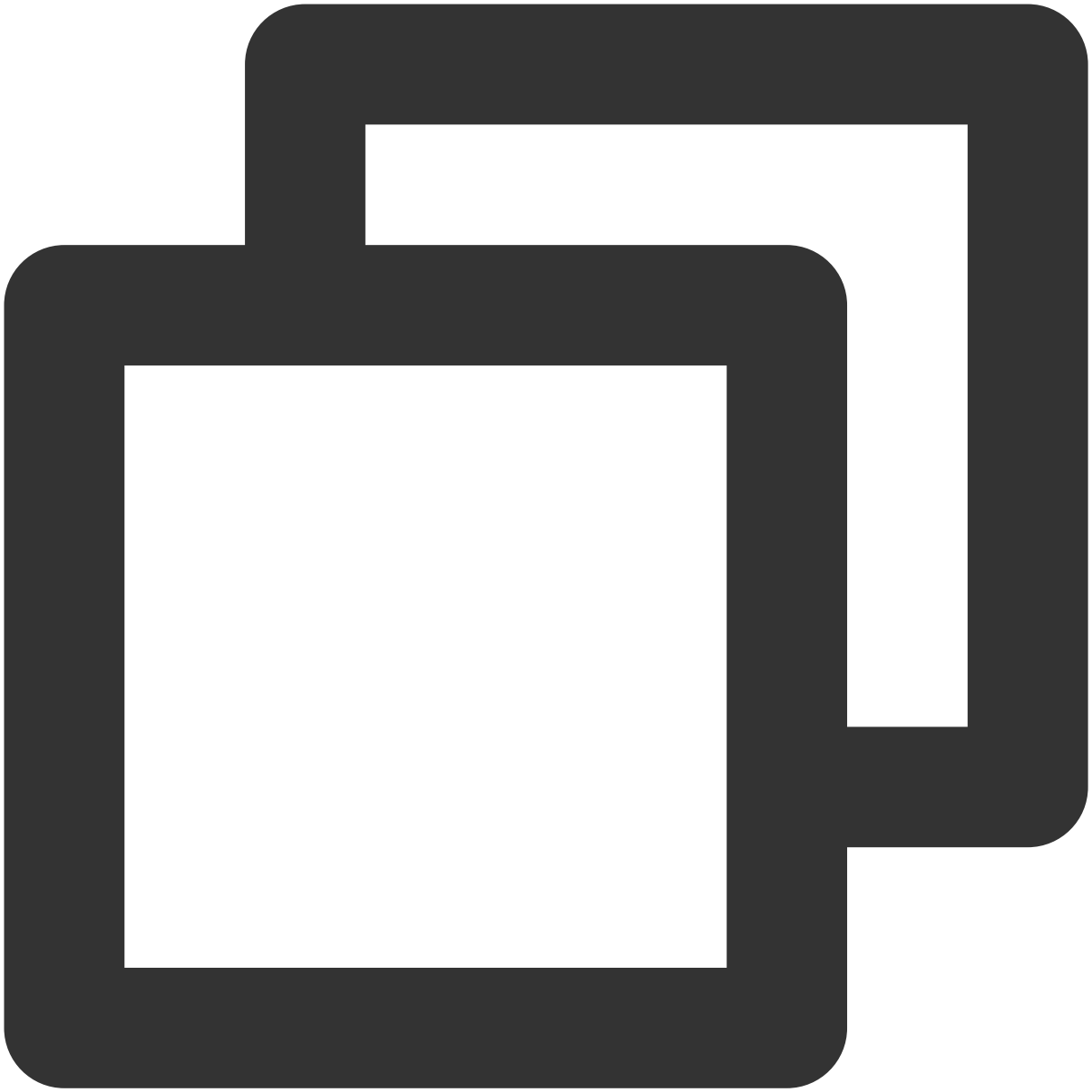




```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing st
tencentcloud_monitor_tmp_tke_cluster_agent.foo: Refreshing state... [id=prom-jh0znt
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state.

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:
```

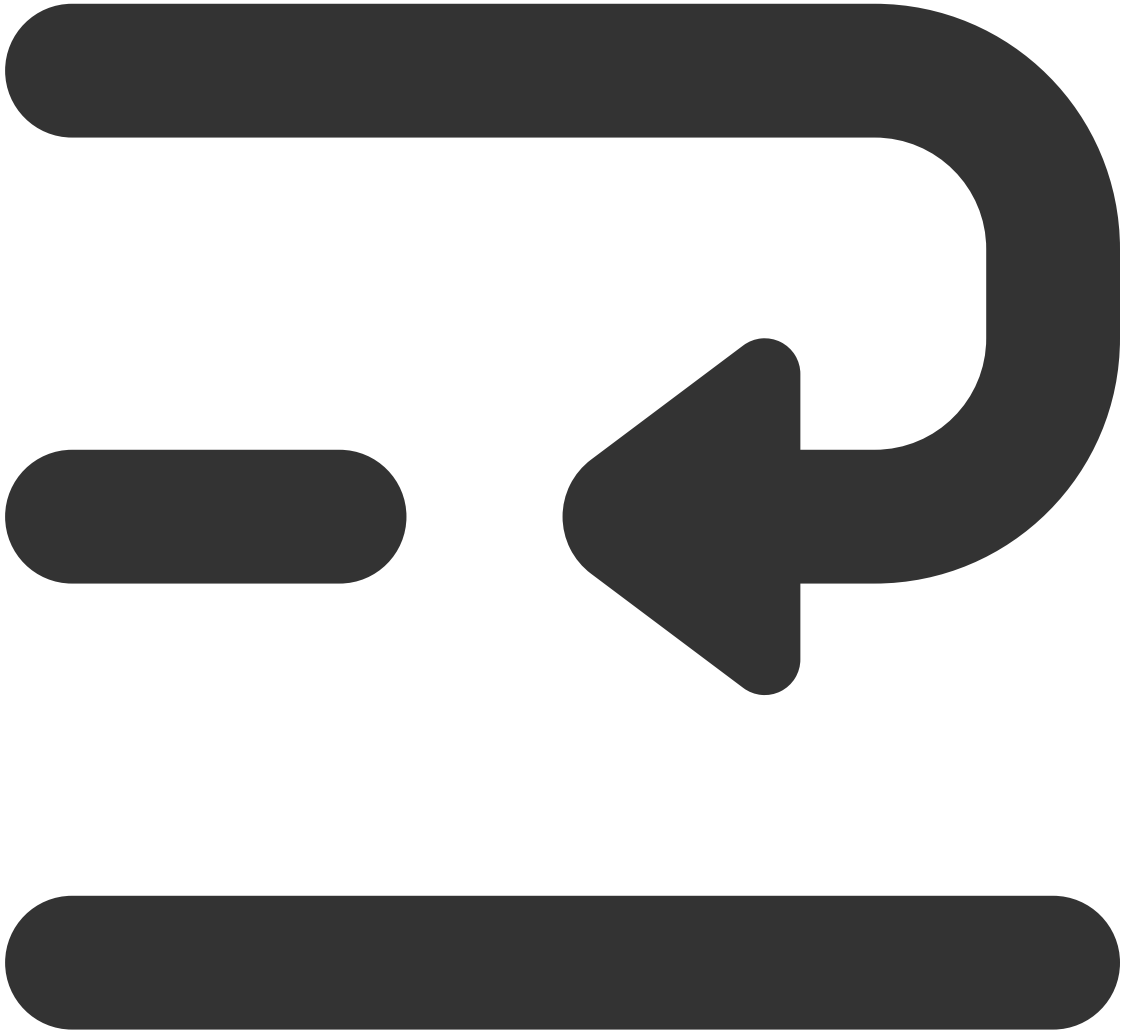
```
  # tencentcloud_monitor_tmp_alert_rule.foo will be created
  + resource "tencentcloud_monitor_tmp_alert_rule" "foo" {
      + duration    = "2m"
      + expr        = "avg by (instance) (mysql_global_status_threads_connected) /
      + id          = (known after apply)
      + instance_id = "prom-jh0zntj2"
      + receivers   = [
          + "notice-zmjsavnp",
        ]
      + rule_name   = "Excessive MySQL connections--tf-Cloud monitoring test"
      + rule_state  = 2
      + type        = "MySQL/Excessive MySQL Connections"

      + annotations {
          + key   = "description"
          + value = "Excessive MySQL connections, instance: {{$labels.instance}}, C
        }
      + annotations {
          + key   = "summary"
          + value = "Excessive MySQL connections (>80%)"
        }

      + labels {
          + key   = "severity"
          + value = "warning"
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------------

Note: You didn't use the -out option to save this plan, so Terraform can't guarante
you run "terraform apply" now.
```

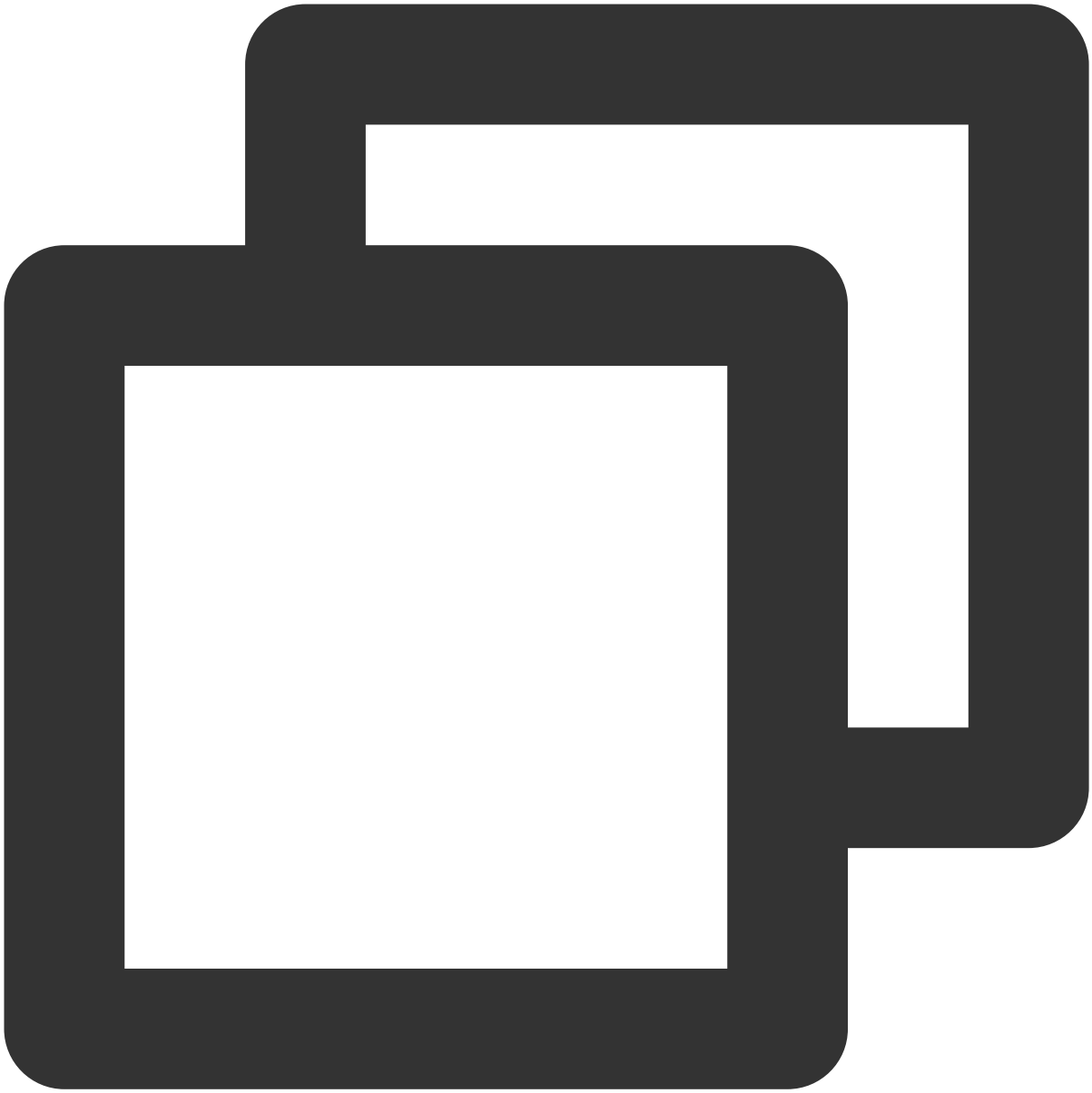
4. To create component integration of the Integration Center, run the following command:



```
terraform apply
```

Expected output:





```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-jh0zntj2]
tencentcloud_monitor_tmp_exporter_integration.tmpExporterIntegration: Refreshing st
tencentcloud_monitor_tmp_exporter_integration.tmpExporterMointor: Refreshing state.
tencentcloud_monitor_tmp_tke_cluster_agent.foo: Refreshing state... [id=prom-jh0znt

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  + create

Terraform will perform the following actions:
 Instance content...
```

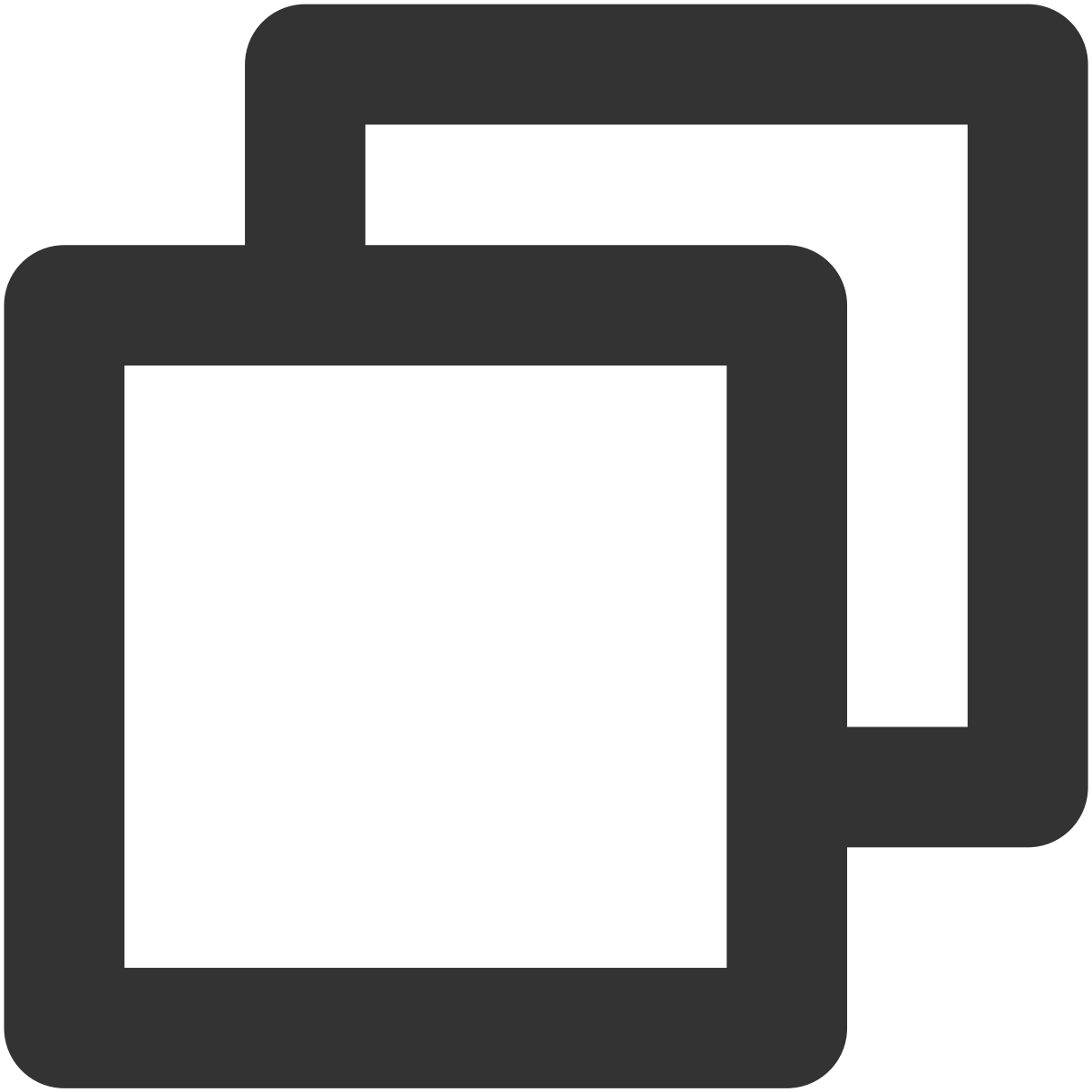
```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tencentcloud_monitor_tmp_alert_rule.foo: Creating...
tencentcloud_monitor_tmp_alert_rule.foo: Creation complete after 2s [id=prom-jh0znt

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Viewing the Status of the Prometheus Instance

Log in to the TCOP, and choose **Managed Service for Prometheus** from the left-hand navigation bar. Existing instances are displayed in the Prometheus instance list.

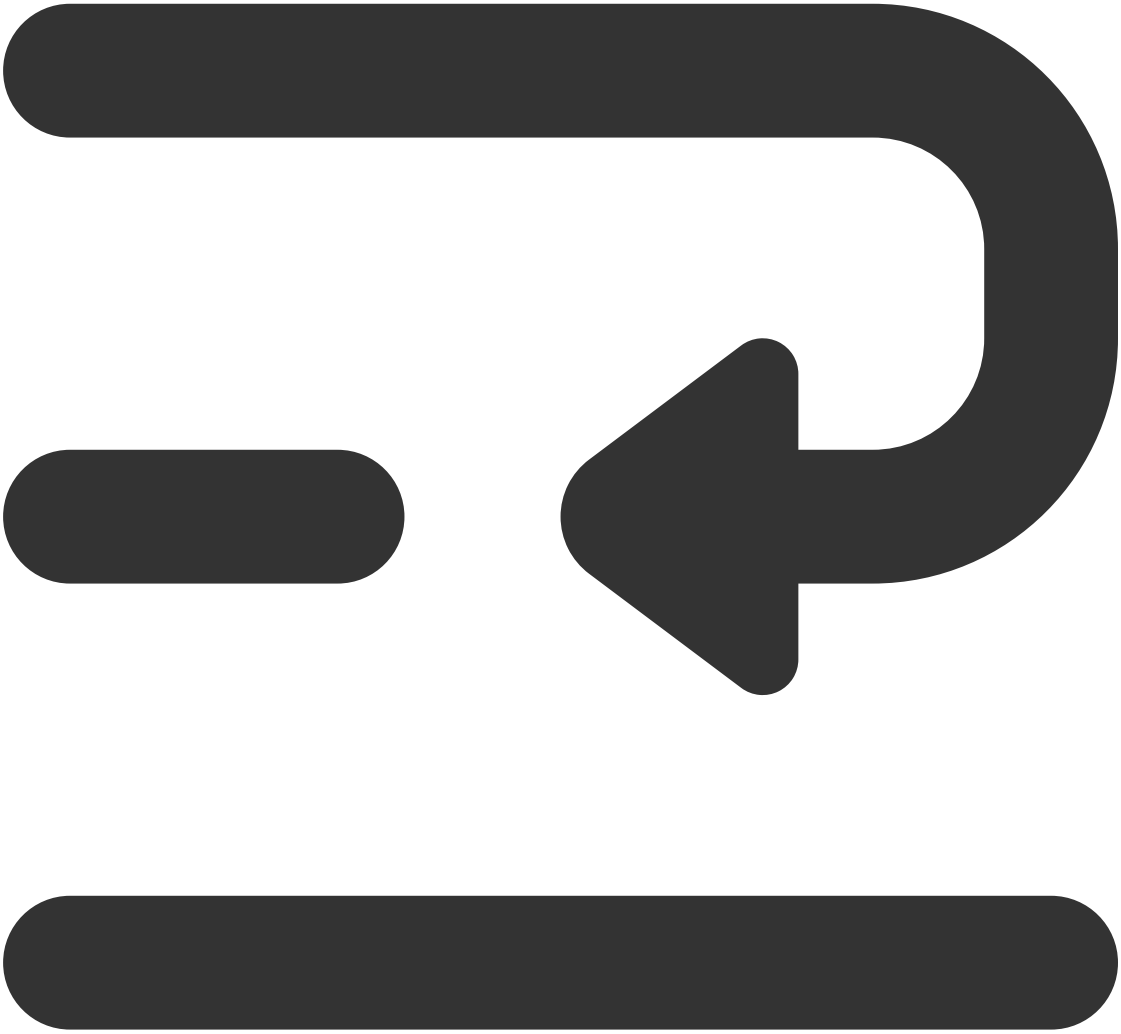
Deleting the Component Integration in the Prometheus Instance Integration Center

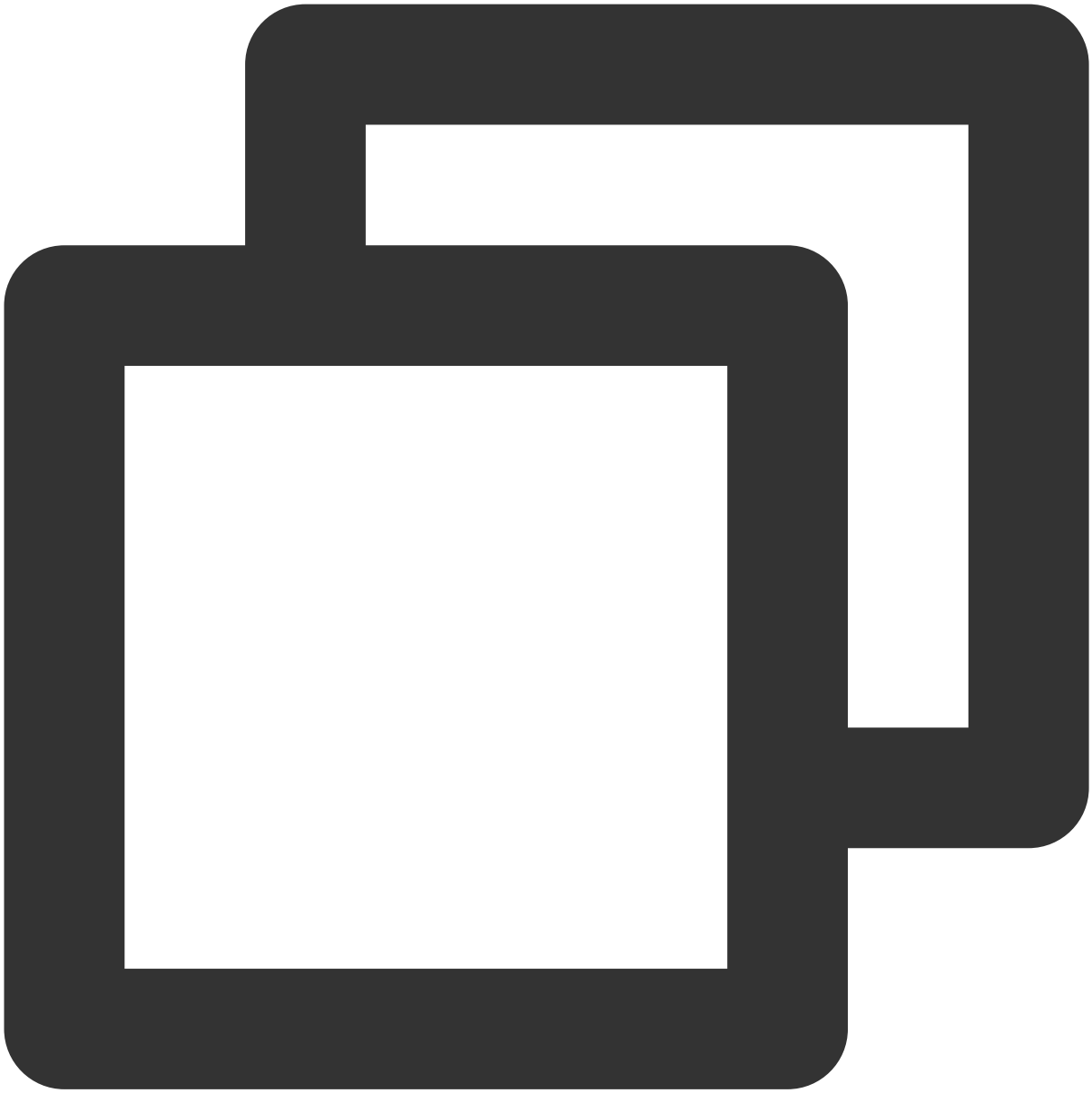
To terminate the resources, run the following command:



```
terraform destroy
```

Expected output:





```
tencentcloud_monitor_tmp_instance.foo: Refreshing state... [id=prom-8dyb6iny]

Terraform used the selected providers to generate the following execution plan. Res
following symbols:
  - destroy

Terraform will perform the following actions:
Instance content...

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
```

```
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

tencentcloud_monitor_tmp_instance.foo: Destroying... [id=prom-8dyb6iny]
tencentcloud_monitor_tmp_instance.foo: Destruction complete after 6s

Destroy complete! Resources: 1 destroyed.
```

Note:

If `Destroy complete! Resources: (numbers of existing instances) destroyed.` is displayed, the instance has been deleted.

FAQs
Basic Questions

Last updated：2024-08-15 17:56:35

How to use Prometheus's native default UI and other features?

TencentCloud Managed Service for Prometheus (TMP) is different from the open source stand-alone Prometheus. TMP is of structure with separated collection and storage and does not provide native default UI features. Use Tencent Cloud Managed Service for Grafana as an alternative for querying features.

When the alarm is recovered, the $value in the notification template is incorrect. How to handle it?

The $value during the alarm recovering is the value that meets the alarm expression condition for the last time. The value that does not meet the condition cannot be obtained. Regarding the design, the alarm expression is a whole. The alarm expression has no difference from the PromQL query in the normal scene. If the series in the query result meets the duration, the alarm will be triggered. When the next query result does not contain a series, the alarm corresponding to the series will be recovered. Prometheus cannot deconstruct and interpret the alarm expression by itself, because some expressions themselves do not contain comparison relationships such as thresholds, for example: `a and b` and `123456789` .

How to view the alarm records?

Prometheus itself does not have the concept of alarm records. Due to special reasons, it cannot fully support the feature of alarm records, but the related alarms and status can be viewed through the ALERTS or ALERTS_FOR_STATE metrics. In addition, all Prometheus alarms are currently delivered to the Tencent Cloud Observability Platform Alarm Service, and then sent to the user side through various notification channels of the Platform. The alarm records feature provided by the Tencent Cloud Observability Platform can make up for this defect to a certain extent. Due to conceptual differences in design, it cannot be used as a complete alternative but can be used for troubleshooting and other requirements. You can view records through the Alarm Records Feature.

Is the alarm repeat interval inconsistent with the configured duration?

Prometheus alarm repeat interval (repeat_interval) is not set for a single alarm but for the entire group. When there is a new alarm or alarms are recovered in the group, the entire group will be notified, and then notifications will be sent at a repeated interval. Currently, TencentCloud Managed Service for Prometheus is grouped by a single alarm policy. For example, a single alarm policy may be for the restart of instance A/B/C. When A/B restarts, the alarm is triggered and then notifications will be sent at a repeated interval. After a period of time, C also restarts, and the interval will be interrupted and recovered. At this time, it will be notified to the user taking A/B/C as a Group. Currently, due to implementation limitations, the alarms delivered based on the Tencent Cloud Observability Platform cannot be

grouped and aggregated as a single message to notify users. If necessary, you can use a custom Alertmanager or Tencent Cloud Managed Service for Grafana.

Why doesn't the rate/irate function produce any data when the original metrics exist?

The rate/irate function requires at least two data points to calculate, so the time range calculated by rate/irate must cover at least two data points. Considering the possibility of data point loss due to network exception, the official recommendation for the time range is **four times the collection interval**.

Why does the rate/irate function calculate a very large outlier?

rate/irate functions can only be used for Counter type metrics, which are defined as strictly increasing numbers. Querying through Prometheus will process the problem of Counter being reset to 0, such as server restart. Normally, this will not affect the calculation results unless there is a problem of data point disorder. For example, the disorder of two second-level data points 9999 and 10000 results in an outlier of (10000+9999)-10000 = 9999 (normally it should be 1). The typical scene of this situation is as follows:
There are multiple collection components collecting the same metric and repeatedly reporting it to the same Prometheus instance, which may cause disorder problems, resulting in a large outlier calculated after the reset processing logic of Prometheus. Solution: If the collection component collects the same metric, it does not meet the expectations. It is recommended to troubleshoot and solve the problem of repeated collection. If the collection solution is designed to ensure the high availability of the collection component, different collection components need to add replica or other tags to distinguish all collected metrics.
Report directly to TencentCloud Managed Service for Prometheus through pushgateway and other methods and no timestamp information is included. TencentCloud Managed Service for Prometheus side will store the current time as the timestamp of the metric. If the network jitters, the arrival time of the data points will become disordered or the timestamp information granted by different processes/threads processing different data points will be disordered, resulting in calculation errors. The solution is to include the timestamp information when reporting. TencentCloud Managed Service for Prometheus built-in pushgateway reporting method is only a supplement to remotewrite and does not realize the complete feature. It is recommended to use Agent(remotewrite) for collection and reporting when it is not necessary.

Why is the interval of data points returned by the query different from that of capturing?

The interval of data points returned by querying through Prometheus is determined by the query parameter interval/step. Each data point is filled in and aligned strictly according to the interval/step, and there is no one-to-one correspondence with the capturing interval. If there is too much data loss or the capturing interval is too large, data will not be filled in, and the storage side will not store any information related to the capturing configuration. Users need to process their capturing configuration.

Why does the query return extra data for the last five minutes?

By default, Prometheus will fill in data for certain queries. Even if there is only one data point in the last five minutes, it may return multiple data points in the past five minutes (according to the step/interval parameters of the query). This is the default behavior of open source Prometheus and cannot be adjusted currently. Generally, it does not affect normal use.

How do time-related functions in PromQL handle local time zones?

All time in Prometheus is in UTC time. There are no exceptions, and the concept of time zone is not considered in the design, which may cause `day_of_*` series of functions to be inconvenient to use, and the official support for them can not be realized in the short term. The temporary solution for the domestic time zone can be as follows: `day_of_week(vector(time() + 8*3600))` .

Can Prometheus-related use limits be adjusted?

The use limits related to Prometheus are adjustable. In most cases, exceeding these use limits may lead to degradation of user experience and performance. Therefore, adjusting the limits upward cannot guarantee the query and write performance benchmarks before adjustments. The service level agreement may no longer apply. Users need to have certain psychological expectations for potential related risks.

From the chart, it seems that the alarm meets the duration but is not actually triggered?

The basic principle of Prometheus alarm detection is to use Instant Query to query data every minute, and trigger alarms if the alarm condition is (continuously) met. In some cases, Prometheus's Range Query will fill in the missing data, and the continuous timeline on the chart may be discontinuous under the execution logic of the alarm component. In addition, due to the characteristics of alarm scheduled inspection, the time when the alarm is detected may be slightly delayed. You can check the status of the alarm by querying the ALERTS or ALERTS_FOR_STATE metrics.

Integration with TKE Cluster

Last updated：2024-08-07 22:08:34

What should I do if I can't access the TKE cluster over the private network?

When installing an agent, you need to access TKE over the private network. If **private network access** is not enabled for the corresponding TKE cluster, the installation will fail. You can solve this problem by following the steps below:

1. Log in to the TKE console and select a container cluster in the corresponding region.
2. Enable **Private Network Access** under **Basic Information** > **Cluster APIServer Information**.

What should I do if the status of all kube-proxy collection targets is "Down"?

In TKE, the launch parameter `--metrics-bind-address` is not specified for kube-proxy, and the default listening address of the metrics service is 127.0.0.1; therefore, the agent cannot pull metrics by Pod IP. You can solve this problem by following the steps below:

1. Log in to the TKE console and select a container cluster in the corresponding region.
2. Go to **Basic Information** > **Cluster APIServer Information** > **Connecting to Kubernetes cluster through Kubectl** to configure Kubectl as instructed.
3. Run `kubectl edit ds kube-proxy -n kube-system` and add the launch parameter `--metrics-bind-address=0.0.0.0` in `spec.template.spec.containers.args` .

What should I do if the status of all component collection targets on the master node in a dedicated TKE cluster is "Down"?

The inbound rule of the default security group of the master node in a dedicated TKE cluster does not allow access to the metrics ports of some components. You can solve this problem by following the steps below:

1. Log in to the Security Group console and select the corresponding region.
2. Enter `tke-master-security-for-<tke cluster id>` in the security group search box. For example, if the cluster ID is `cls-xxx` , then enter `tke-master-security-for-cls-xxx` .
3. Click the returned security group ID to enter the **Edit Inbound Rule** window.
4. The protocol port column of the rule to be edited should include `TCP:60001,60002` . Select the rules one by one and add ports 10249, 10252, 10251, 9100, and 9153 for the following purposes respectively:

10249: kube-proxy metrics port
10252: kube-controller-manager metrics port
10251: kube-scheduler metrics port
9100: node-exporter metrics port
9153: core-dns metrics port

Product Consulting

Last updated：2024-08-07 22:08:40

Does TMP support customizing the reported data?

Yes. TMP supports customizing the reported metric monitoring data in multiple languages and displaying such data in the integrated Grafana dashboards.

How does TMP collect monitoring data?

TMP can pull data through the Prometheus agent or write data through Pushgateway. It is fully compatible with open-source Prometheus in terms of collection method.

Is there a separate exporter for each instance?

Each MySQL or Kafka instance has one exporter, while multiple Redis instances can share the same exporter.

What is the difference between TMP and a self-built Prometheus service?

TMP is fully compatible with the open-source Prometheus ecosystem and is integrated with CM to help you quickly set up a monitoring system (for custom monitoring, component monitoring, basic monitoring, etc.). It supports Grafana, with preset common monitoring dashboards. It also supports diverse exporters, with preset common alarm templates. It well solves various problems with open-source Prometheus, such as complicated HA setup, poor performance, and labor-intensive Ops.

Which Tencent Cloud products does TMP support?

TMP supports CVM, TencentDB for MongoDB, TencentDB for MySQL, TencentDB for PostgreSQL, TencentDB for Redis, ES, and TKE. For more information, see Integration Center.

Can I integrate other data sources on the Grafana page of TMP?

No. We recommend you integrate them by using Grafana.

Can I pull monitoring data in Prometheus pull mode?

No, you can only pull data through APIs. For detailed directions, see Querying Monitoring Data.

What is the series storage limit on pay-as-you-go TMP instances?

Each instance can have up to 4.5 million series. If you need to adjust it, submit a ticket for application.

Does TMP support integration with EKS?

Yes. Pay-as-you-go TMP instances support general clusters, EKS elastic clusters, edge clusters, and registered clusters.

Use and Technology

Last updated：2024-08-07 22:08:46

How do I migrate from self-built Prometheus to TMP?

In the configuration file of self-built Prometheus, add a remote write configuration pointing to TMP for migration. For more information, see Migration from Self-Built Prometheus.

Can I batch import/export dashboards into/from Grafana?

You can do so through APIs as instructed in HTTP API reference.

What should I do if one exporter has too much data?

You can use Prometheus in this scenario, but we recommend you not expose too many metrics in the exporter, as the exporter itself and the Prometheus agent consume a lot of memory. Only expose key metrics, and do not use parameters such as user ID and URL in metric labels.

How do I use TMP to monitor clusters in two different VPCs?

1. You can interconnect the VPCs of the two clusters through CCN and then integrate the clusters into Prometheus .
2. Install the agent in one of the clusters, and then use the Nginx proxy to remotely write the target address to the TMP instance.

How do I integrate native container services with TMP?

You can't integrate them directly. However, you can create data and add a remote write configuration pointing to TMP in the configuration file of your self-built Prometheus. For detailed directions, see Migration from Self-Built Prometheus.

How do I set the parameters of the alarm cycle when creating an alarm rule through Prometheus APIs?

Add `key=_interval_  value=1m/5m/10m/15m/30m/60m` to the new `Labels` parameter.

Will data get lost during instance rebooting?

Data will not get lost during instance rebooting, as it is stored in TencentDB for CTSDB. However, because data cannot be reported normally during rebooting, data breakpoints may occur.

Is it normal to see a rise in generated data after the TMP instance is rebooted?

Retries will be performed after instance rebooting, so it is normal if the data volume seems to fluctuate greatly in a short period of time.

Can I batch add MySQL instances when integrating MySQL with Prometheus in the integration center?

No. You can only add instances one by one.

Where can I view the security group of the EKS cluster created by Prometheus?

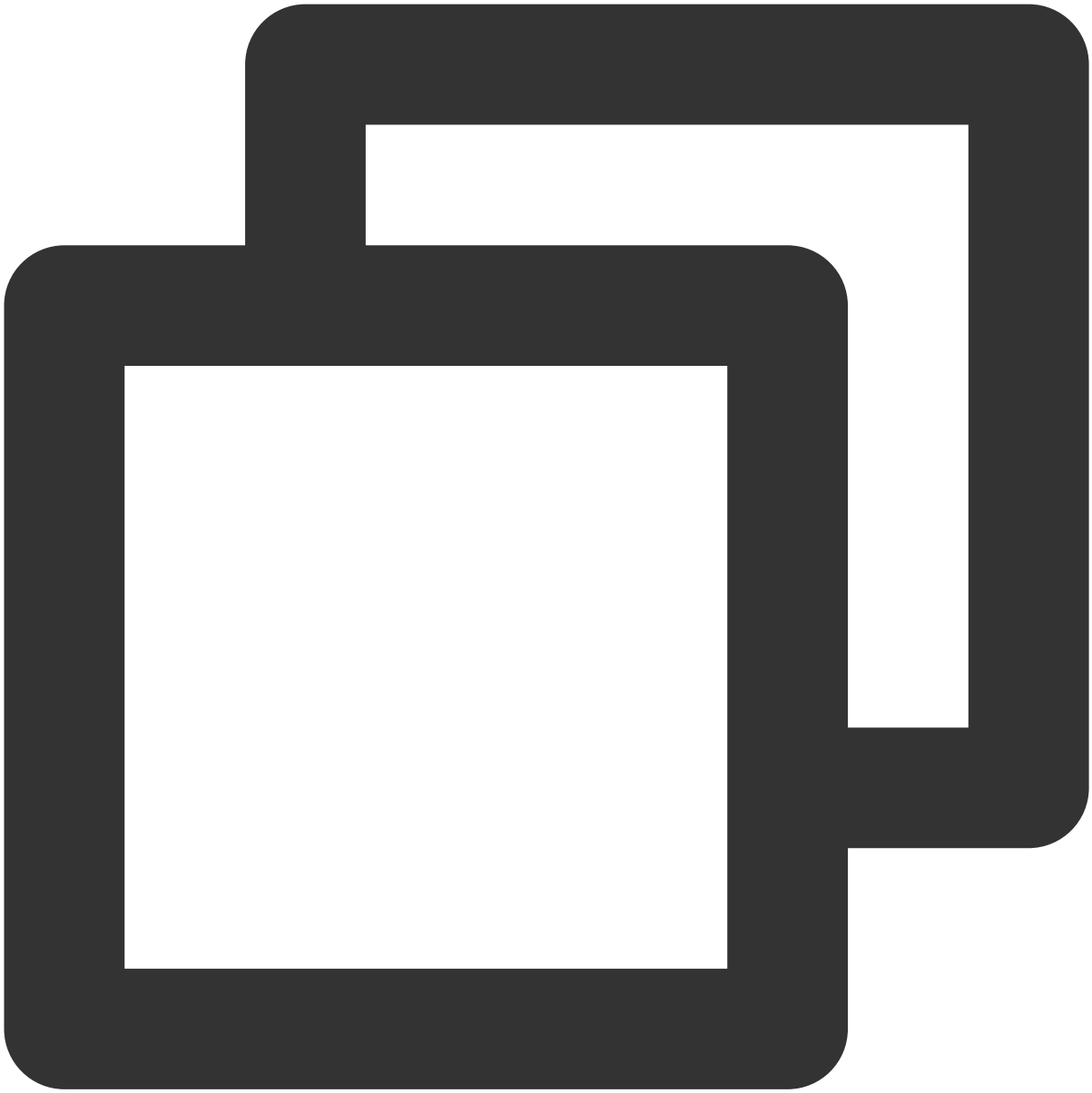
Go to **Security** > **Security Group** in the VPC console and search by the current TMP instance ID.

Can I configure multiple scrape tasks for a PodMonitor?

Yes, but you should ensure that the tasks have the same port name and label.

(TMP Pushgateway usage) If the clients of multiple instances push the same job with different label metrics, metrics will be overwritten. What should I do?

You can use `groupingkey` to solve this problem. Below is the sample code:



```
if err := push.New("http://$IP:$PORT", "db_backup").
        BasicAuth("$APPID", "$TOKEN").
        Collector(completionTime).
        Grouping("instance", "$INSTANCE").
```