TencentCloud Managed Service for Prometheus

Integration Guide

Product Documentation

Contents

Integration Guide
Scrape Configuration Description

Last updated：2024-01-29 15:55:08

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：2024-01-29 15:55:07

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.

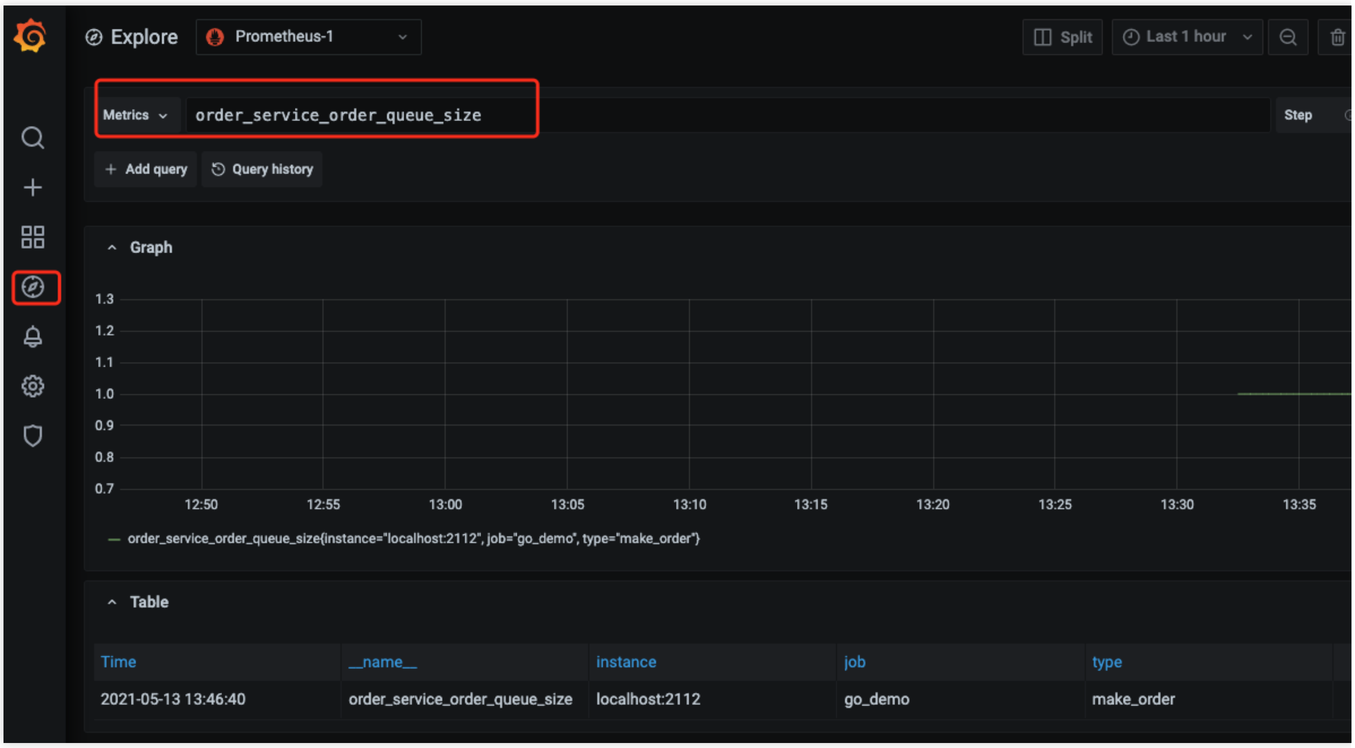
```
package main

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.



order_service_order_queue_size

order_service_order_queue_size{instance="localhost:2112", job="go_demo", type="make_order"}

Time	__name__	instance	job	type
2021-05-13 13:46:40	order_service_order_queue_size	localhost:2112	go_demo	make_order

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：2024-01-29 15:55:07

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

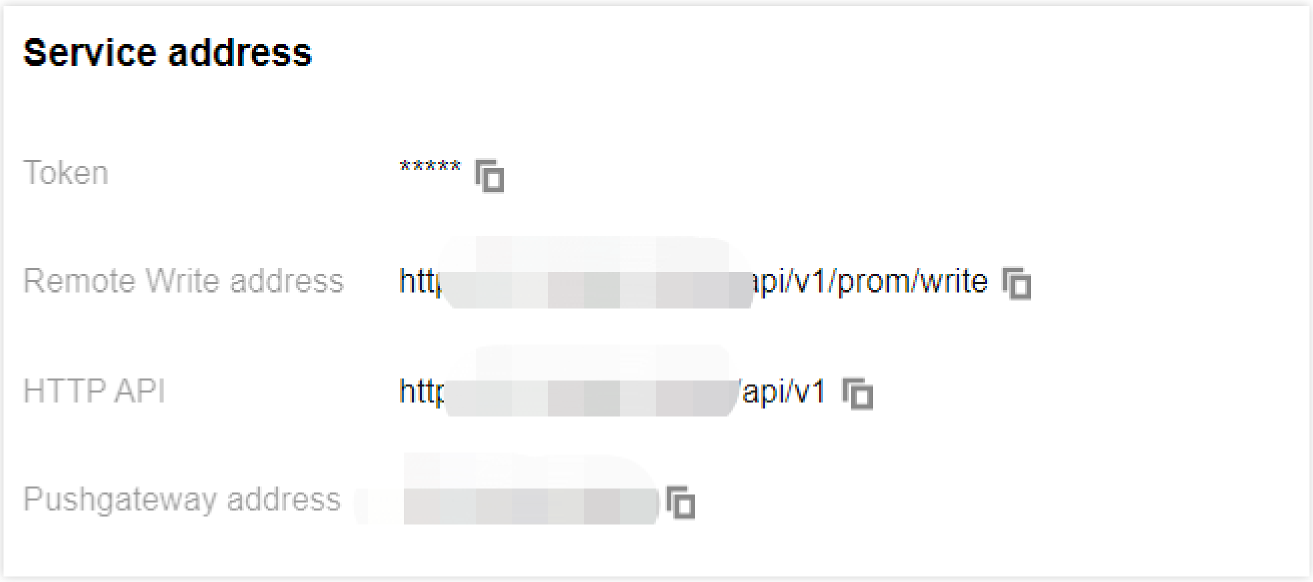
1. The EMR product you purchased includes the Flink component, and a Flink task is running in your instance.
2. You have created a TKE cluster in the region and VPC of your TMP instance.

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.



Service address

Token	*****
Remote Write address	htt[redacted]pi/v1/prom/write
HTTP API	htt[redacted]/api/v1
Pushgateway address	[redacted]

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:

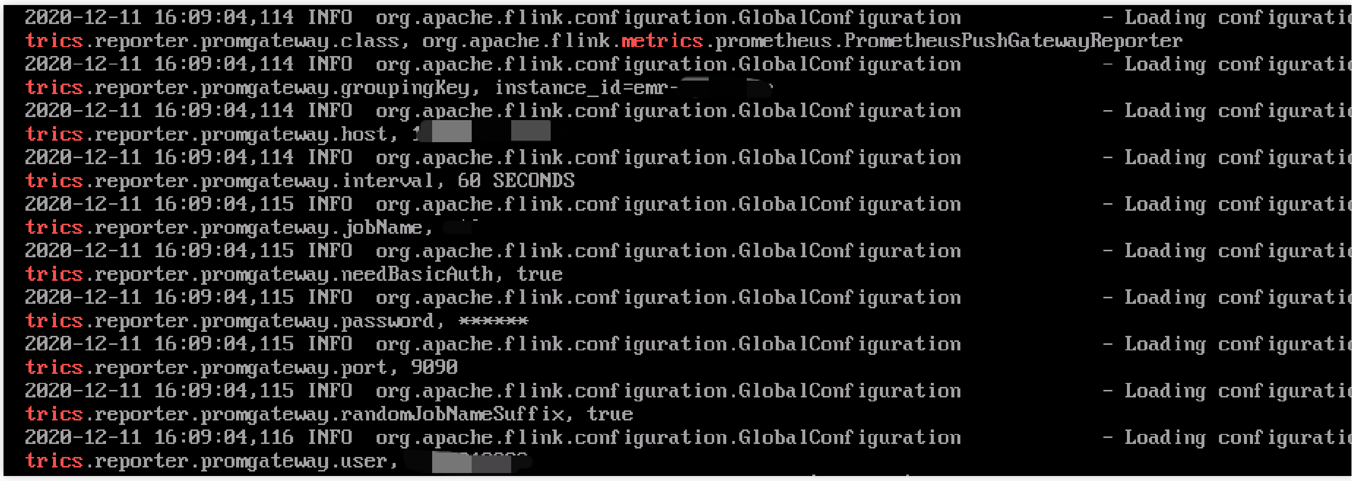
```
cd /usr/local/service/flink/lib
wget https://rig-1258344699.cos.ap-guangzhou.myqcloud.com/flink/flink-metrics-prome
```

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:



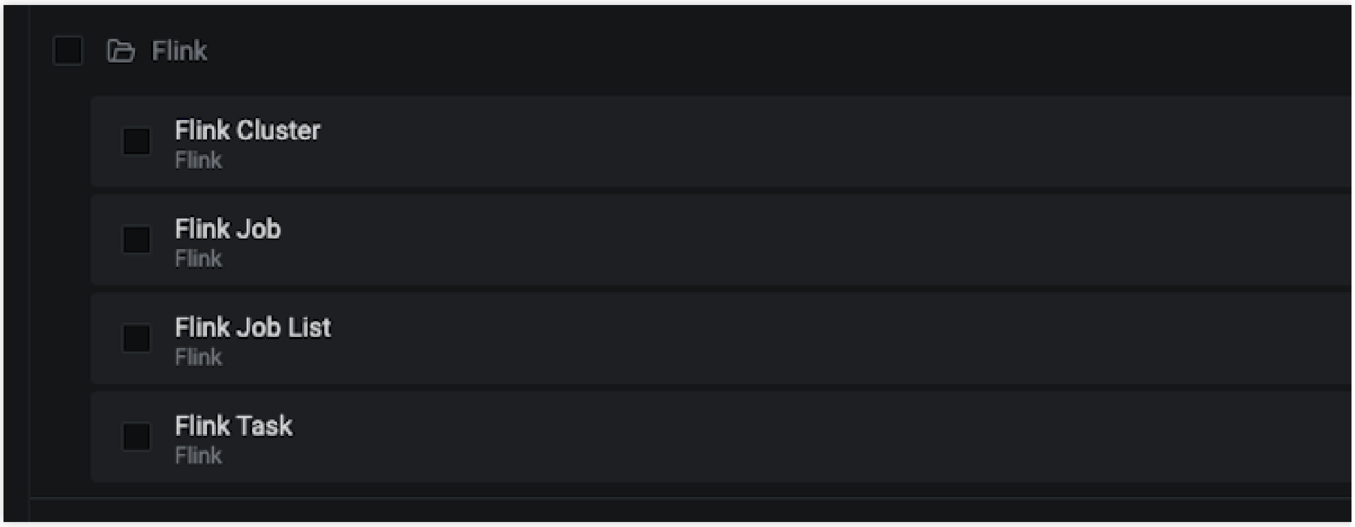
```
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.class, org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporter
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.groupingKey, instance_id=emr-[redacted]
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.host, 1[redacted]
2020-12-11 16:09:04,114 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.interval, 60 SECONDS
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.jobName, [redacted]
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.needBasicAuth, true
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.password, ******
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.port, 9090
2020-12-11 16:09:04,115 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.randomJobNameSuffix, true
2020-12-11 16:09:04,116 INFO  org.apache.flink.configuration.GlobalConfiguration            - Loading configurati
trics.reporter.promgateway.user, [redacted]
```

Note:

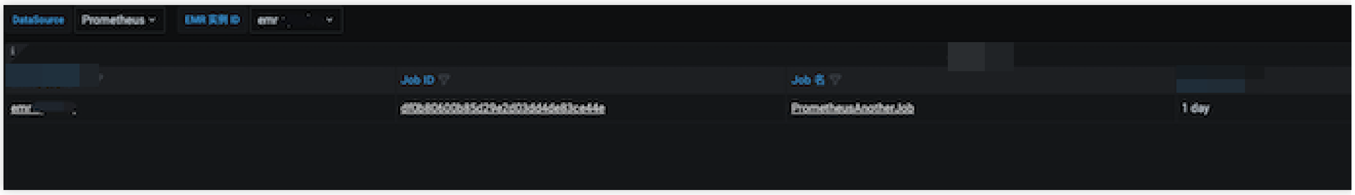
As tasks previously submitted in the cluster use the old configuration file, their metrics are not reported.

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.

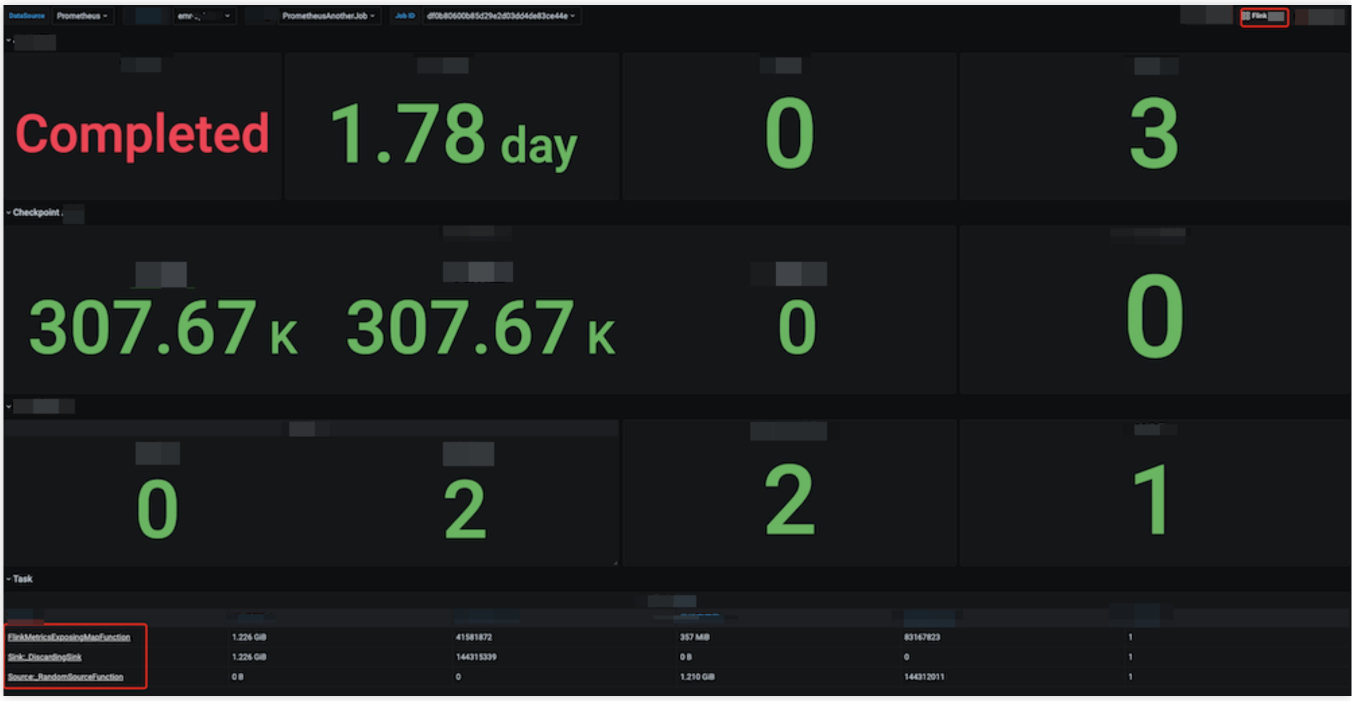


3. Click **Flink Job List** to view the monitoring information.



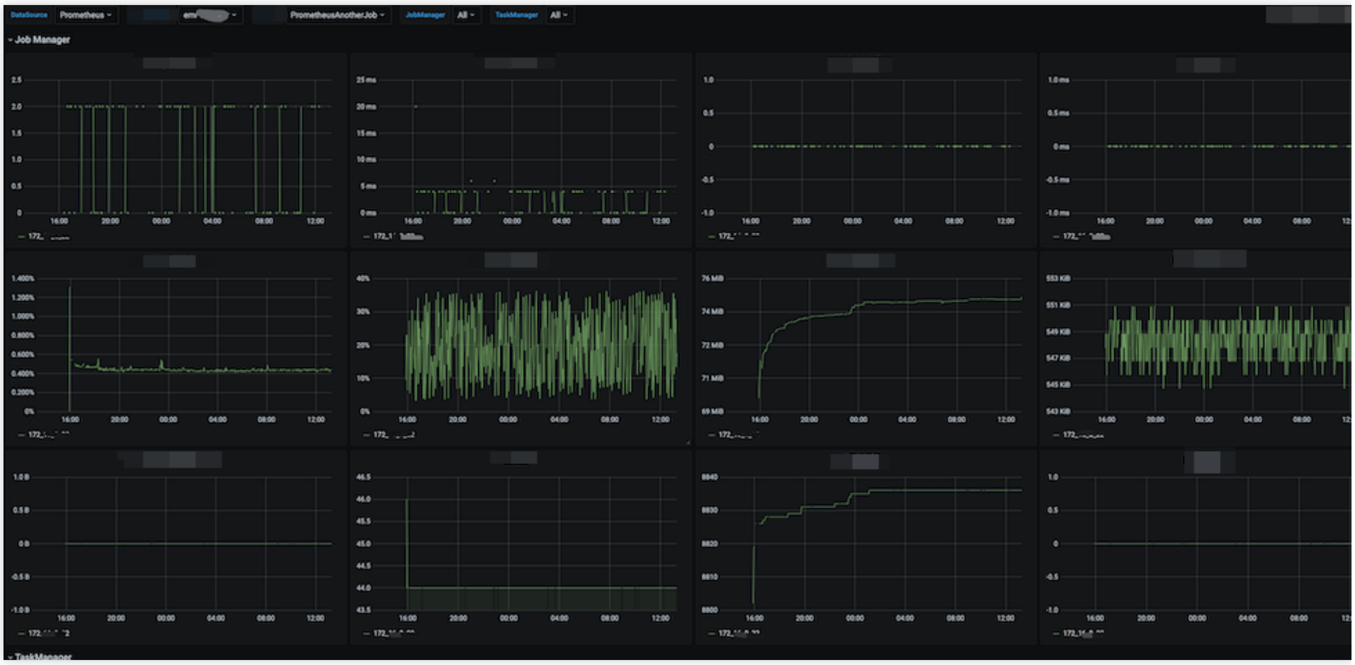
[redacted]	Job ID	Job 名	[redacted]
emr [redacted]	df0b80600b85d29e2d03dd4de83ce44e	PrometheusAnotherJob	1 day

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

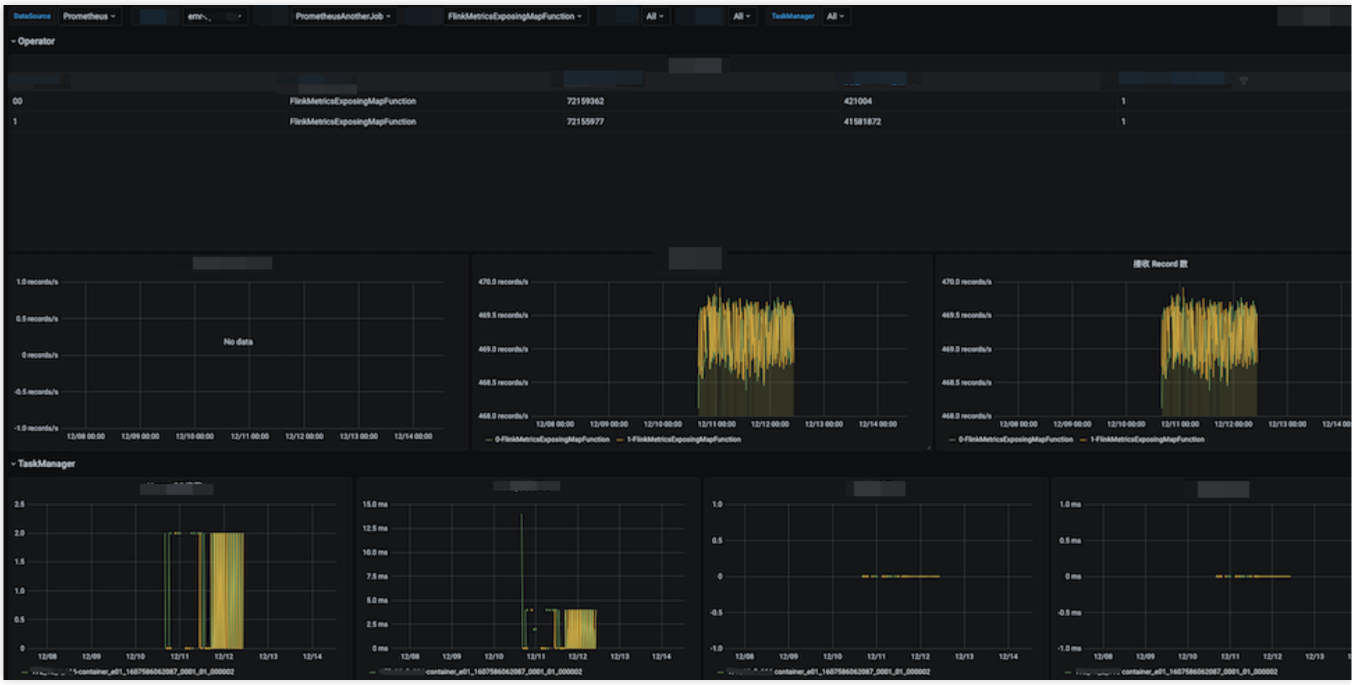


[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FlinkMetricsExposingMapFunction	1.226 GiB	41581872	357 MiB	83167823	1
Sink: DiscardingSink	1.226 GiB	144315339	0 B	0	1
Source: RandomSourceFunction	0 B	0	1.210 GiB	144312011	1

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：2024-01-29 15:29:42

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.

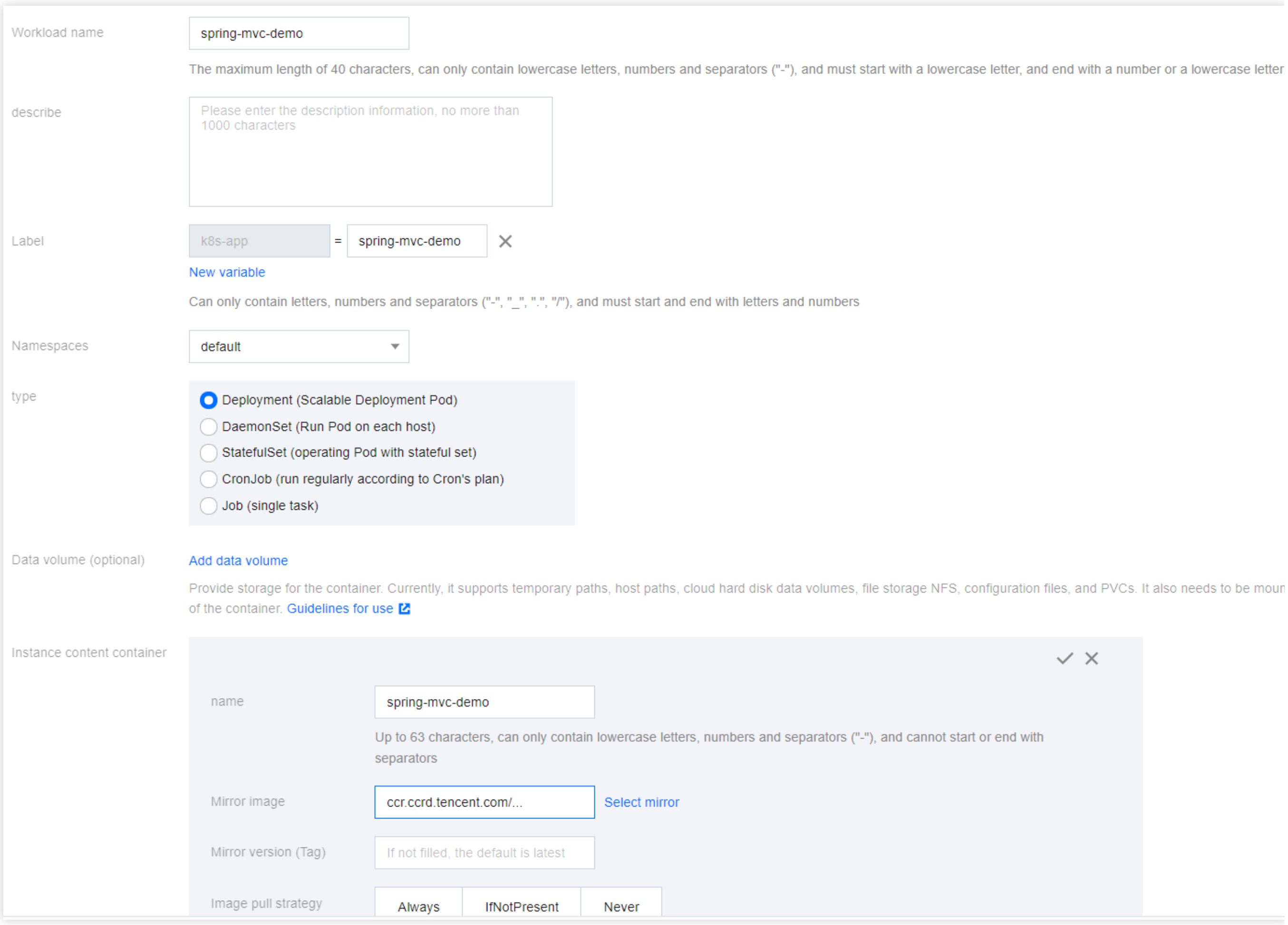
```
mvn clean package
docker build . -t ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[image tag]
```

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.



Workload name: spring-mvc-demo

The maximum length of 40 characters, can only contain lowercase letters, numbers and separators ("-"), and must start with a lowercase letter, and end with a number or a lowercase letter

describe: Please enter the description information, no more than 1000 characters

Label: k8s-app = spring-mvc-demo

New variable

Can only contain letters, numbers and separators ("-", "_", ".", "/"), and must start and end with letters and numbers

Namespaces: default

type:
- [✓ Deployment (Scalable Deployment Pod)]
- DaemonSet (Run Pod on each host)
- StatefulSet (operating Pod with stateful set)
- CronJob (run regularly according to Cron's plan)
- Job (single task)

Data volume (optional): Add data volume

Provide storage for the container. Currently, it supports temporary paths, host paths, cloud hard disk data volumes, file storage NFS, configuration files, and PVCs. It also needs to be mour of the container. Guidelines for use

Instance content container

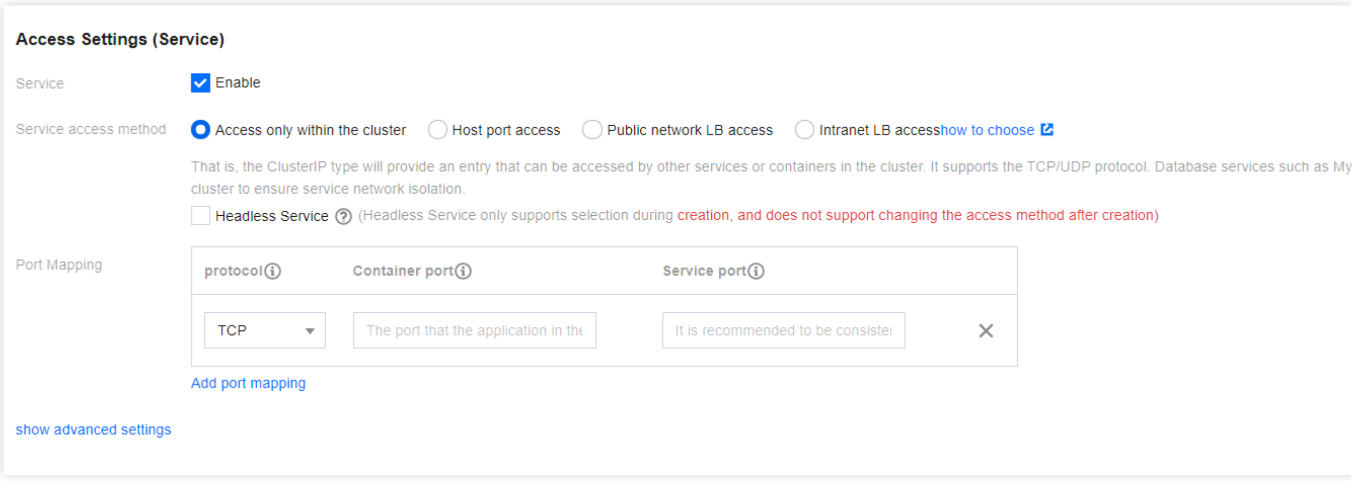
name: spring-mvc-demo

Up to 63 characters, can only contain lowercase letters, numbers and separators ("-"), and cannot start or end with separators

Mirror image: ccr.ccrd.tencent.com/... Select mirror

Mirror version (Tag): If not filled, the default is latest

Image pull strategy: Always IfNotPresent Never



Access Settings (Service)

Service: [✓ Enable]

Service access method: [✓ Access only within the cluster] Host port access Public network LB access Intranet LB accesshow to choose

That is, the ClusterIP type will provide an entry that can be accessed by other services or containers in the cluster. It supports the TCP/UDP protocol. Database services such as My cluster to ensure service network isolation.

Headless Service (Headless Service only supports selection during creation, and does not support changing the access method after creation)

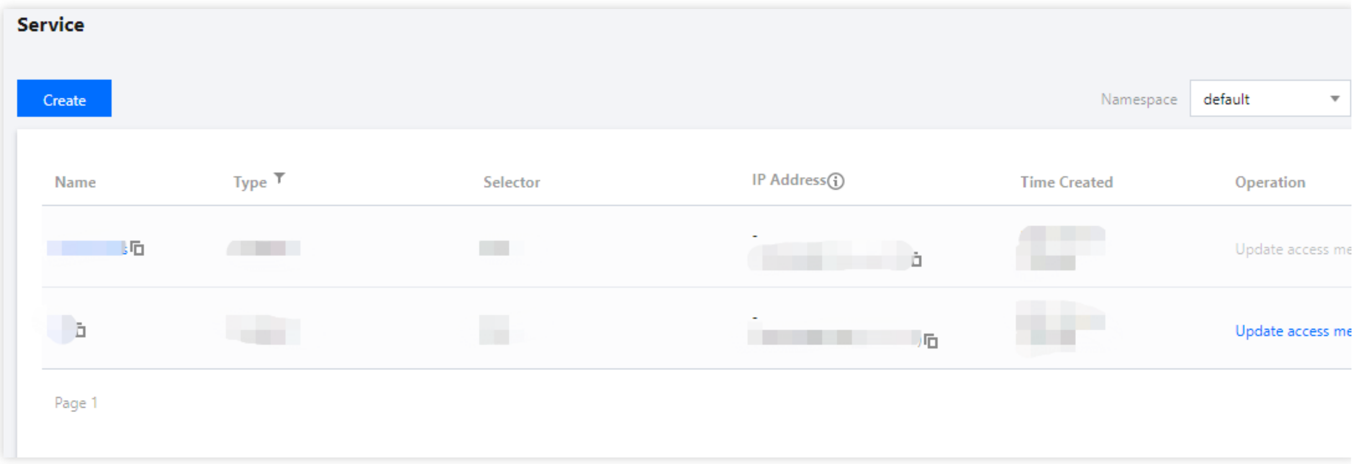
Port Mapping

protocol	Container port	Service port
TCP	The port that the application in the	It is recommended to be consister

Add port mapping

show advanced settings

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:


Service
Create
Namespace
default
Name
Type
Selector
IP Address
Time Created
Operation
Update access me
Update access me
Page 1

```
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:

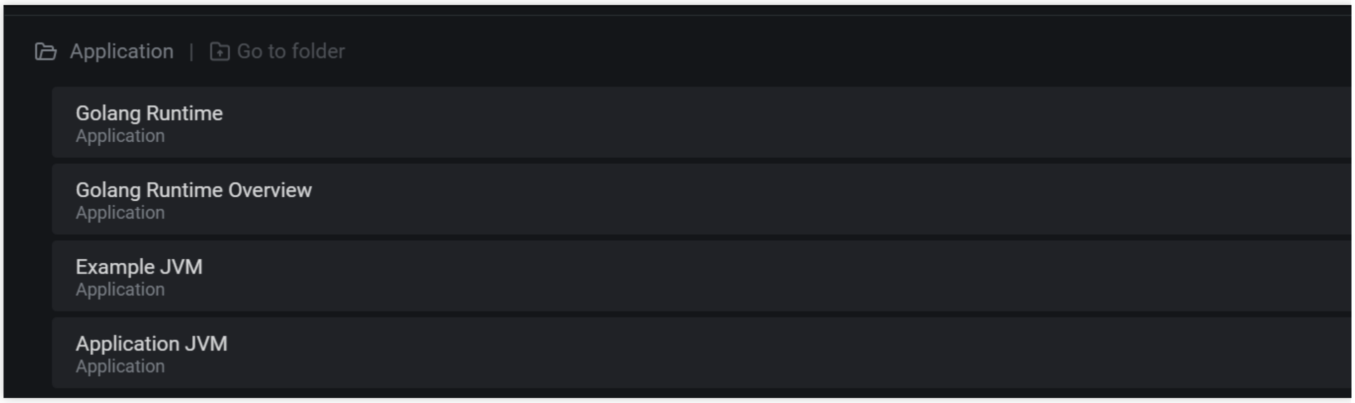
Here, note that the `port` value is the `spec/ports/name` value in the Service YAML configuration file.

```
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：2024-01-29 15:55:08

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]
```

<b>Below is a sample:</b>

```
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

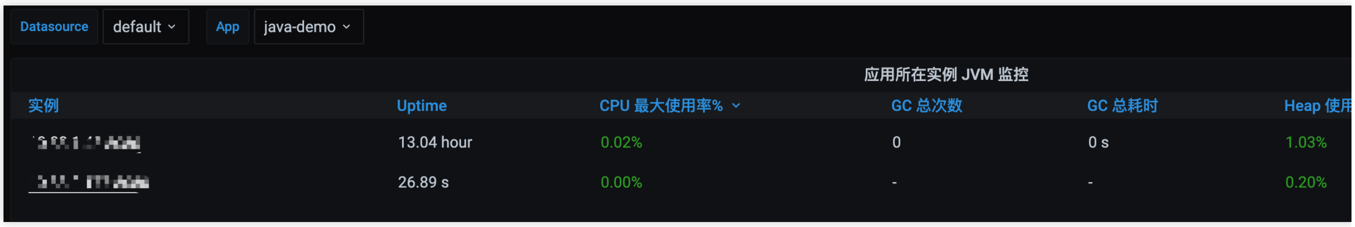
1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:

```
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
```

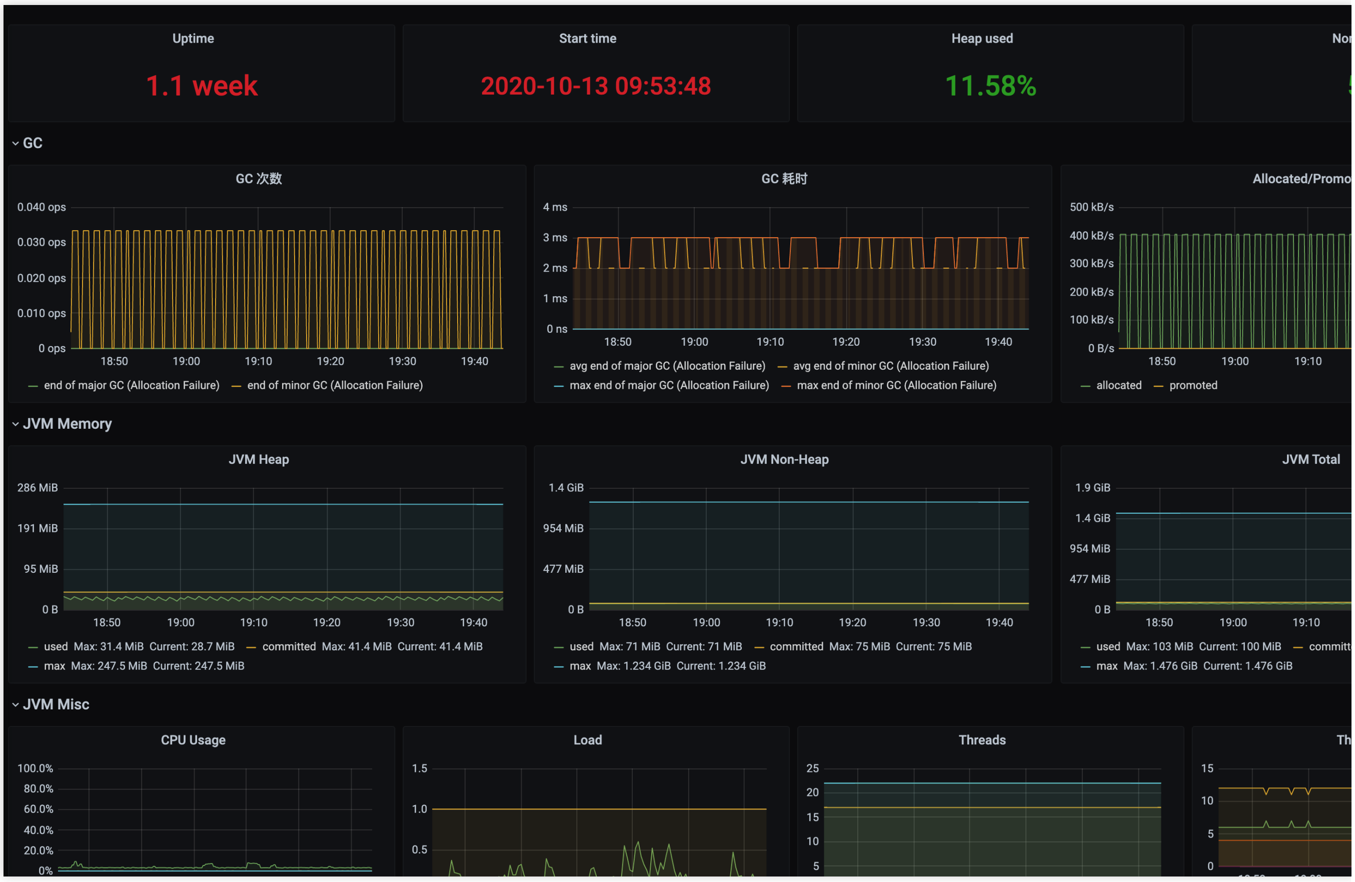
Step 5. View the monitoring information

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.



应用所在实例 JVM 监控

实例	Uptime	CPU 最大使用率%	GC 总次数	GC 总耗时	Heap 使用
[redacted]	13.04 hour	0.02%	0	0 s	1.03%
[redacted]	26.89 s	0.00%	-	-	0.20%


Uptime
1.1 week
Start time
2020-10-13 09:53:48
Heap used
11.58%
GC
GC 次数
end of major GC (Allocation Failure)
end of minor GC (Allocation Failure)
GC 耗时
avg end of major GC (Allocation Failure)
avg end of minor GC (Allocation Failure)
max end of major GC (Allocation Failure)
max end of minor GC (Allocation Failure)
allocated
promoted
JVM Memory
JVM Heap
used Max: 31.4 MiB Current: 28.7 MiB
committed Max: 41.4 MiB Current: 41.4 MiB
max Max: 247.5 MiB Current: 247.5 MiB
JVM Non-Heap
used Max: 71 MiB Current: 71 MiB
committed Max: 75 MiB Current: 75 MiB
max Max: 1.234 GiB Current: 1.234 GiB
JVM Total
used Max: 103 MiB Current: 100 MiB
max Max: 1.476 GiB Current: 1.476 GiB
JVM Misc
CPU Usage
Load
Threads

Go Application Integration

Last updated：2024-01-29 15:55:07

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.

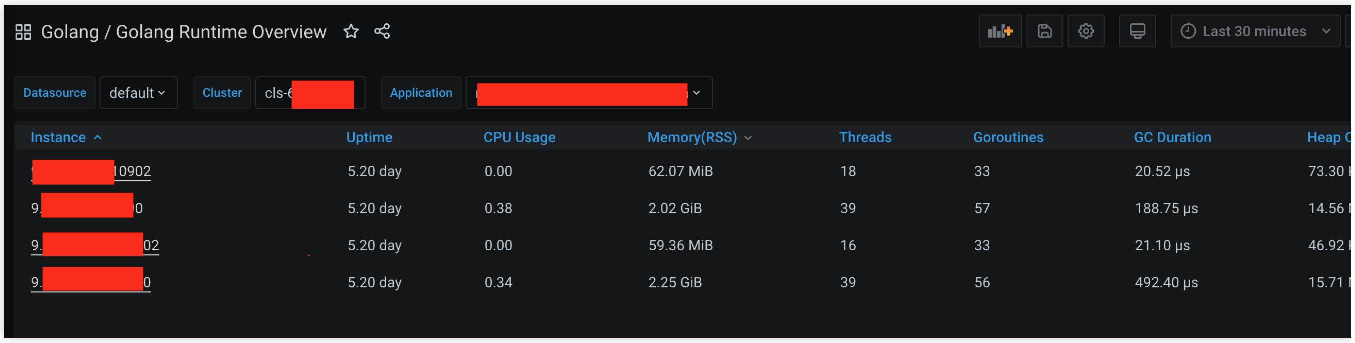
Viewing monitoring information

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

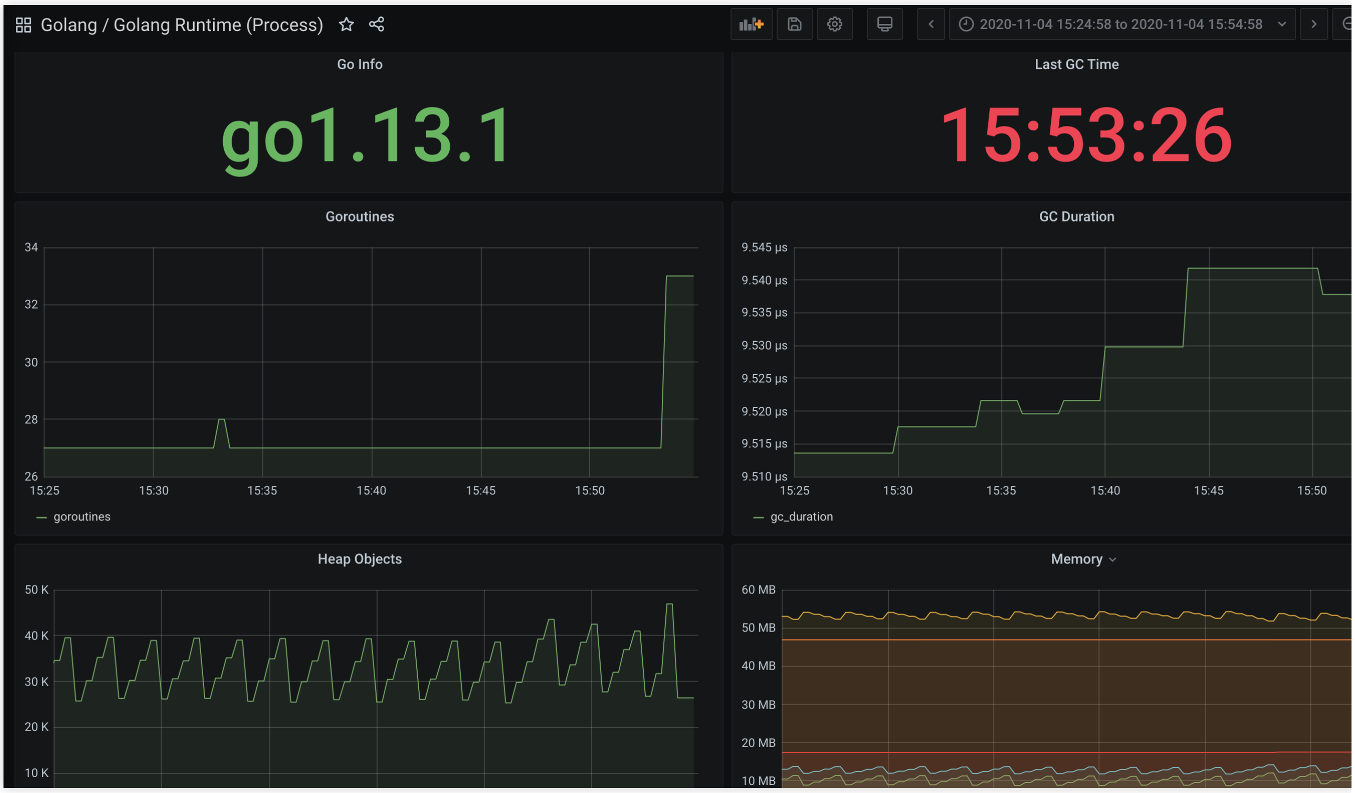
icon to expand the monitoring dashboard, and click the name of the corresponding monitoring chart to view the monitoring data.



Golang / Golang Runtime Overview

Datasource default · Cluster cls-[redacted] · Application [redacted] · Last 30 minutes

Instance	Uptime	CPU Usage	Memory(RSS)	Threads	Goroutines	GC Duration	Heap C
[redacted]10902	5.20 day	0.00	62.07 MiB	18	33	20.52 µs	73.30
9.[redacted]0	5.20 day	0.38	2.02 GiB	39	57	188.75 µs	14.56
9.[redacted]02	5.20 day	0.00	59.36 MiB	16	33	21.10 µs	46.92
9.[redacted]0	5.20 day	0.34	2.25 GiB	39	56	492.40 µs	15.71



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：2024-01-29 15:55:07

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:

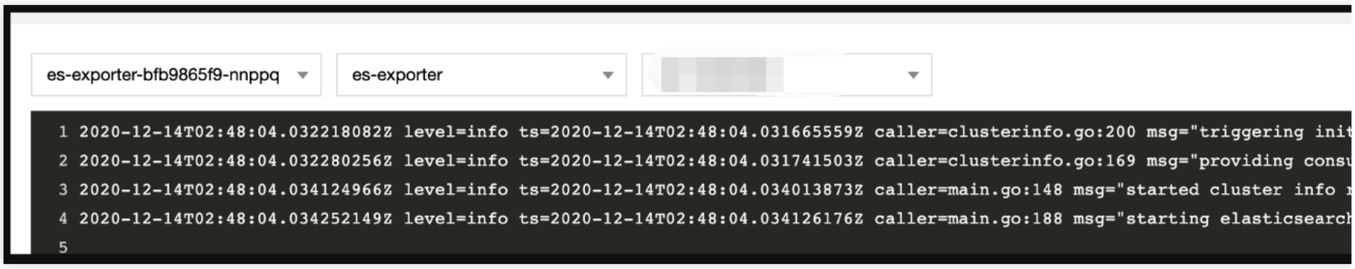
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: es-exporter
  name: es-exporter
  namespace: es-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: es-exporter
  template:
    metadata:
      labels:
        k8s-app: es-exporter
    spec:
      containers:
      - 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:

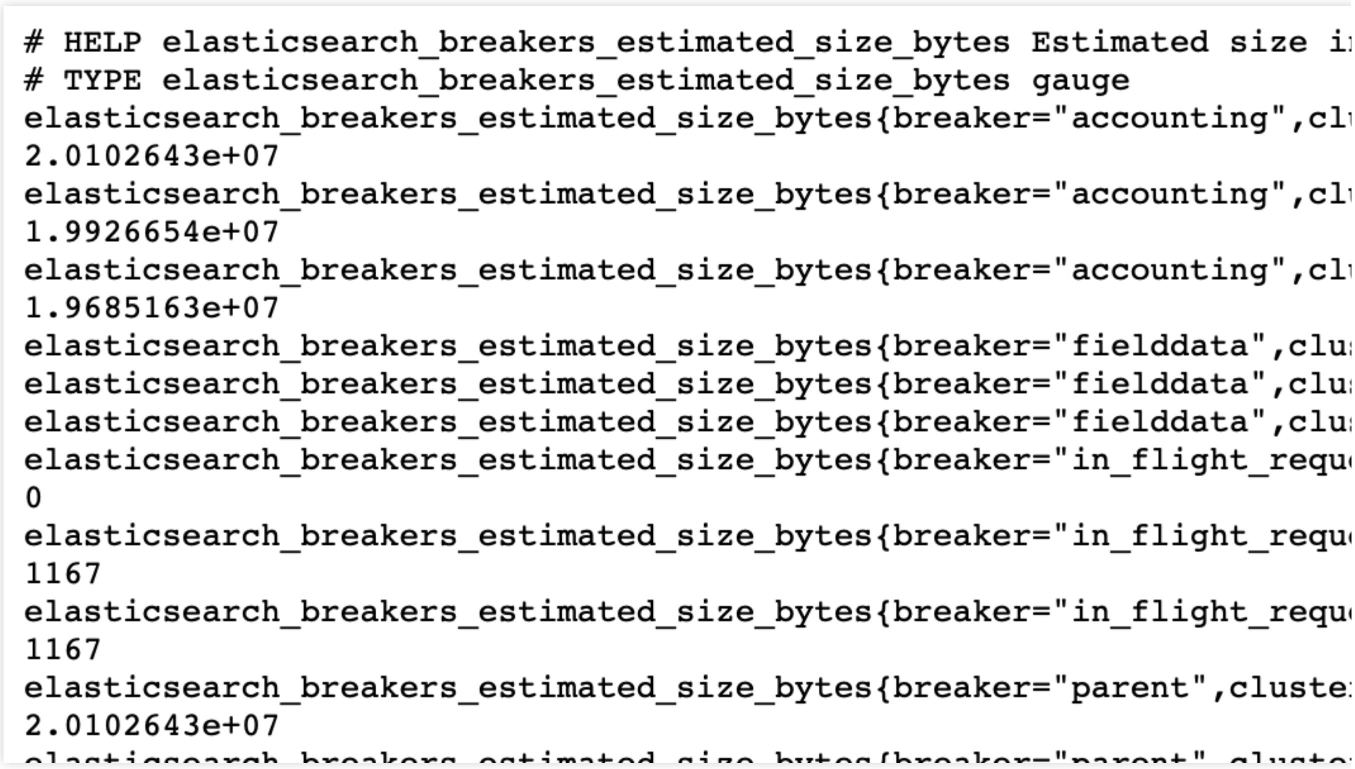


```
2020-12-14T02:48:04.032218082Z level=info ts=2020-12-14T02:48:04.031665559Z caller=clusterinfo.go:200 msg="triggering init
2020-12-14T02:48:04.032280256Z level=info ts=2020-12-14T02:48:04.031741503Z caller=clusterinfo.go:169 msg="providing consu
2020-12-14T02:48:04.034124966Z level=info ts=2020-12-14T02:48:04.034013873Z caller=main.go:148 msg="started cluster info r
2020-12-14T02:48:04.034252149Z level=info ts=2020-12-14T02:48:04.034126176Z caller=main.go:188 msg="starting elasticsearch
```

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
elasticsearch_breakers_estimated_size_bytes{breaker="parent",cluste
```

Adding scrape task

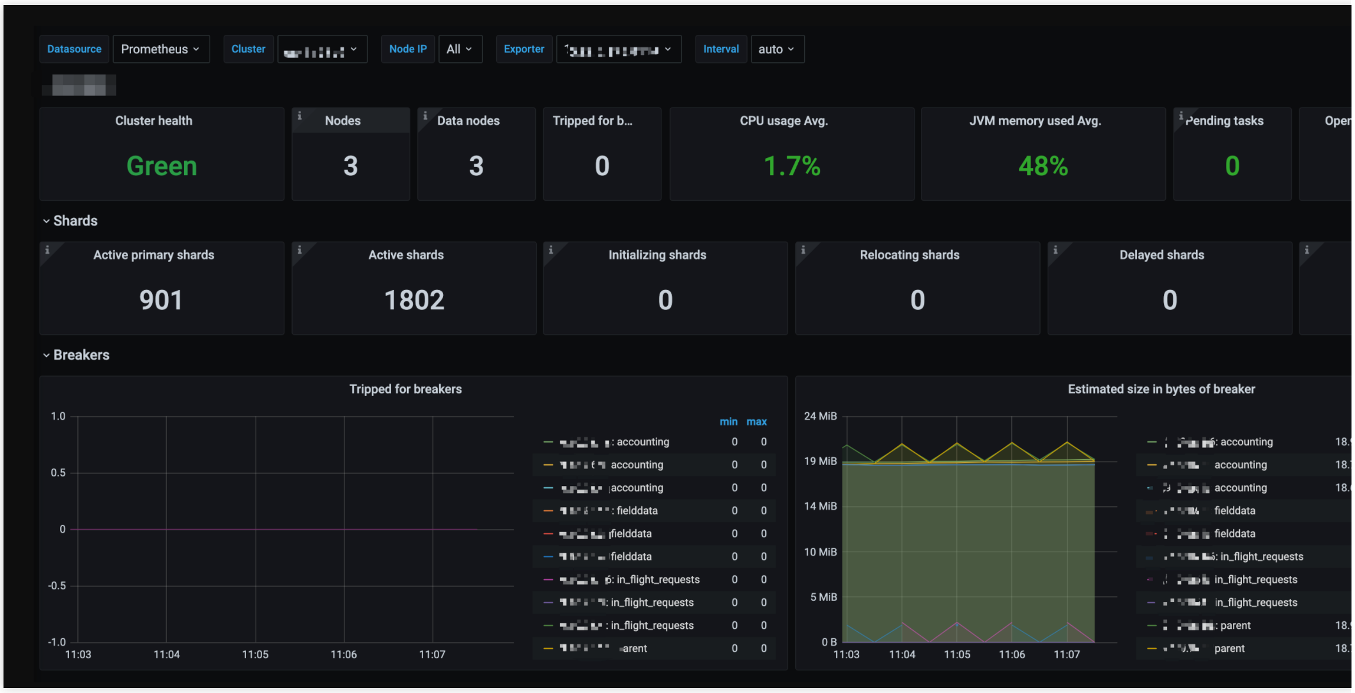
1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:

```
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:



Datasource: Prometheus | Cluster: [redacted] | Node IP: All | Exporter: [redacted] | Interval: auto

Cluster health	Nodes	Data nodes	Tripped for b...	CPU usage Avg.	JVM memory used Avg.	Pending tasks
Green	3	3	0	1.7%	48%	0

Shards

Active primary shards	Active shards	Initializing shards	Relocating shards	Delayed shards
901	1802	0	0	0

Breakers

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：2024-01-29 15:55:07

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:

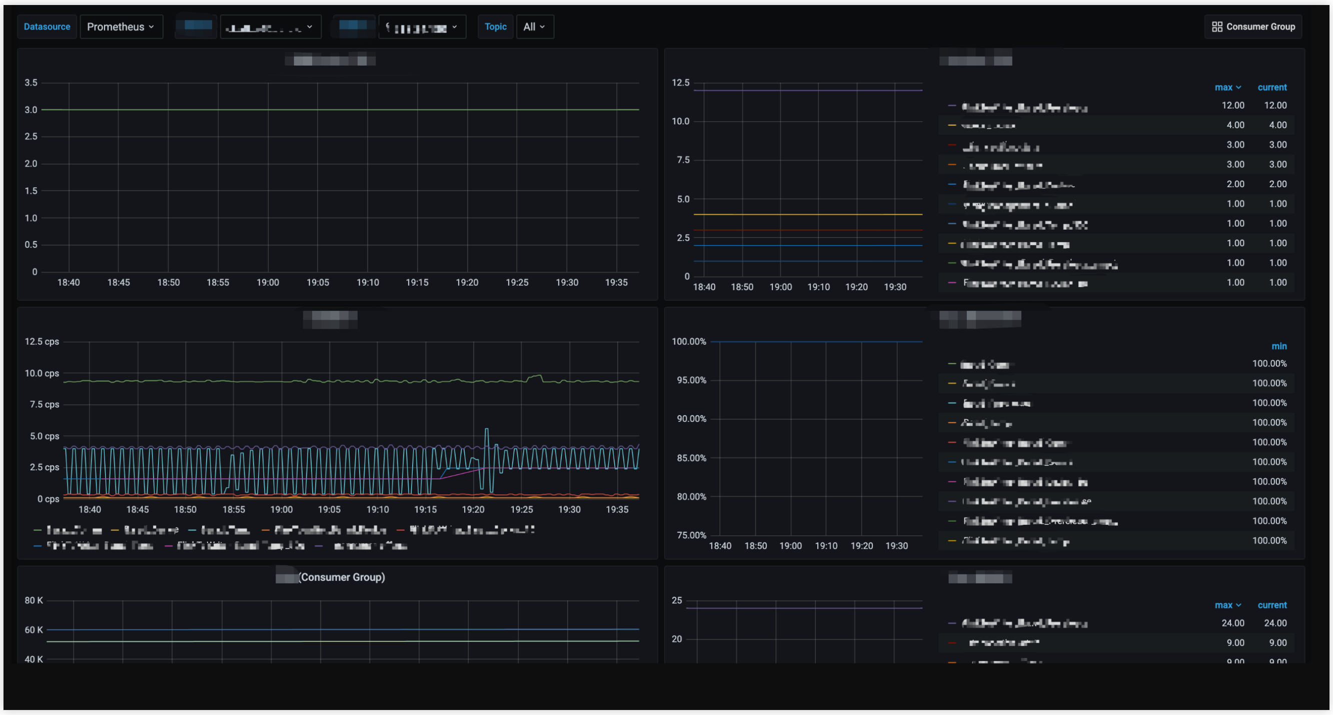
```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: kafka-exporter  # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port # Enter the name of the corresponding port of the 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：2024-01-29 15:55:08

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
```

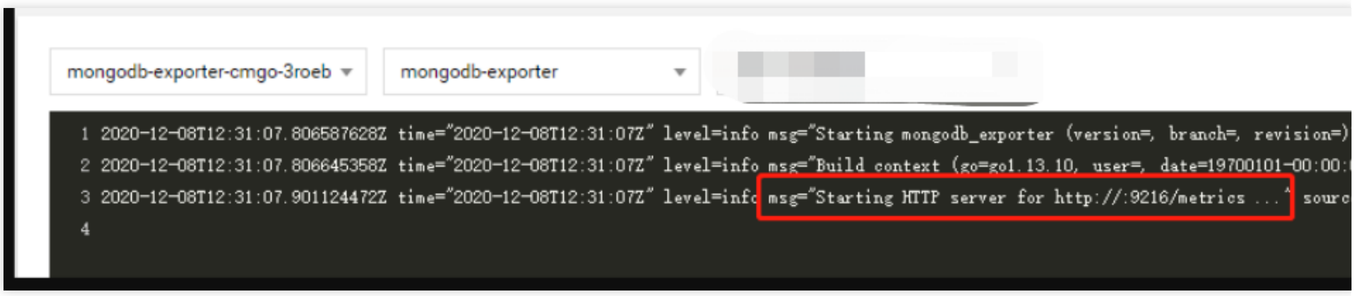
Note:

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:

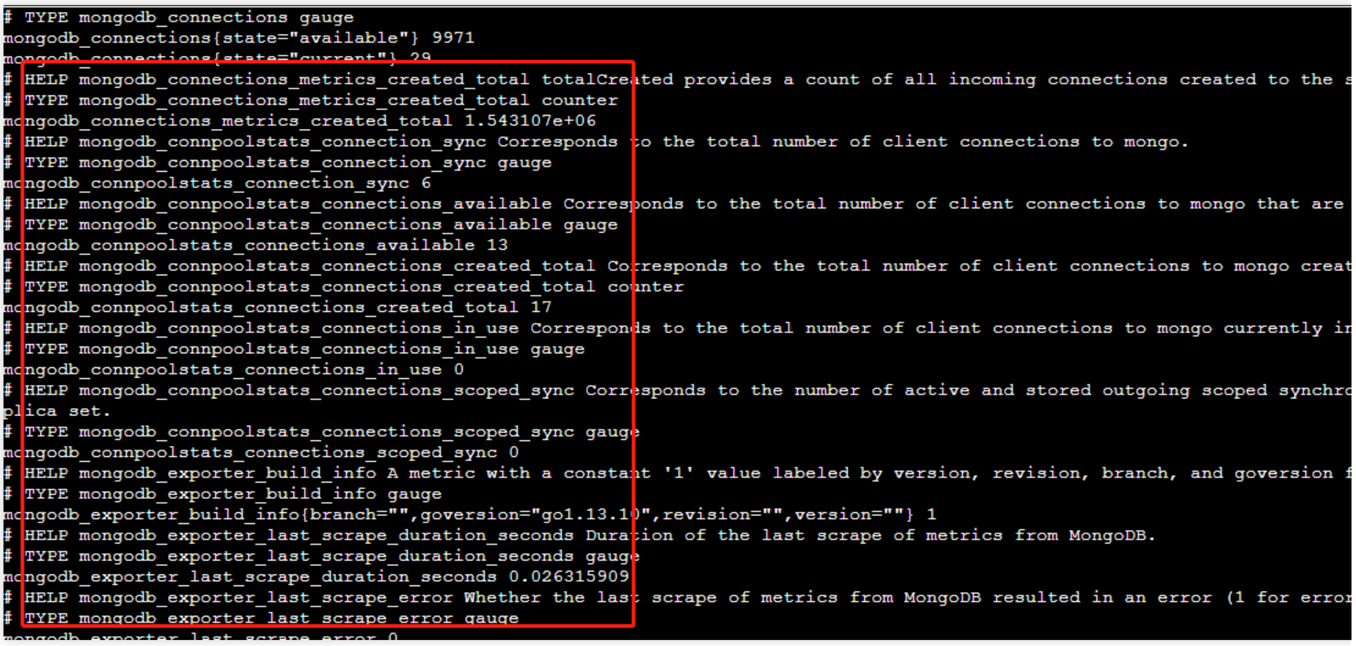


```
2020-12-08T12:31:07.806587628Z time="2020-12-08T12:31:07Z" level=info msg="Starting mongodb_exporter (version=, branch=, revision=)
2020-12-08T12:31:07.806645358Z time="2020-12-08T12:31:07Z" level=info msg="Build context (go=go1.13.10, user=, date=19700101-00:00:
2020-12-08T12:31:07.901124472Z time="2020-12-08T12:31:07Z" level=info msg="Starting HTTP server for http://:9216/metrics ..." sourc
```

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:



```
# TYPE mongodb_connections gauge
mongodb_connections{state="available"} 9971
mongodb_connections{state="current"} 29
# HELP mongodb_connections_metrics_created_total totalCreated provides a count of all incoming connections created to the s
# TYPE mongodb_connections_metrics_created_total counter
mongodb_connections_metrics_created_total 1.543107e+06
# HELP mongodb_connpoolstats_connection_sync Corresponds to the total number of client connections to mongo.
# TYPE mongodb_connpoolstats_connection_sync gauge
mongodb_connpoolstats_connection_sync 6
# HELP mongodb_connpoolstats_connections_available Corresponds to the total number of client connections to mongo that are
# TYPE mongodb_connpoolstats_connections_available gauge
mongodb_connpoolstats_connections_available 13
# HELP mongodb_connpoolstats_connections_created_total Corresponds to the total number of client connections to mongo creat
# TYPE mongodb_connpoolstats_connections_created_total counter
mongodb_connpoolstats_connections_created_total 17
# HELP mongodb_connpoolstats_connections_in_use Corresponds to the total number of client connections to mongo currently in
# TYPE mongodb_connpoolstats_connections_in_use gauge
mongodb_connpoolstats_connections_in_use 0
# HELP mongodb_connpoolstats_connections_scoped_sync Corresponds to the number of active and stored outgoing scoped synchro
plica set.
# TYPE mongodb_connpoolstats_connections_scoped_sync gauge
mongodb_connpoolstats_connections_scoped_sync 0
# HELP mongodb_exporter_build_info A metric with a constant '1' value labeled by version, revision, branch, and goversion f
# TYPE mongodb_exporter_build_info gauge
mongodb_exporter_build_info{branch="",goversion="go1.13.10",revision="",version=""} 1
# HELP mongodb_exporter_last_scrape_duration_seconds Duration of the last scrape of metrics from MongoDB.
# TYPE mongodb_exporter_last_scrape_duration_seconds gauge
mongodb_exporter_last_scrape_duration_seconds 0.026315909
# HELP mongodb_exporter_last_scrape_error Whether the last scrape of metrics from MongoDB resulted in an error (1 for error
# TYPE mongodb_exporter_last_scrape_error gauge
mongodb_exporter_last_scrape_error 0
```

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:

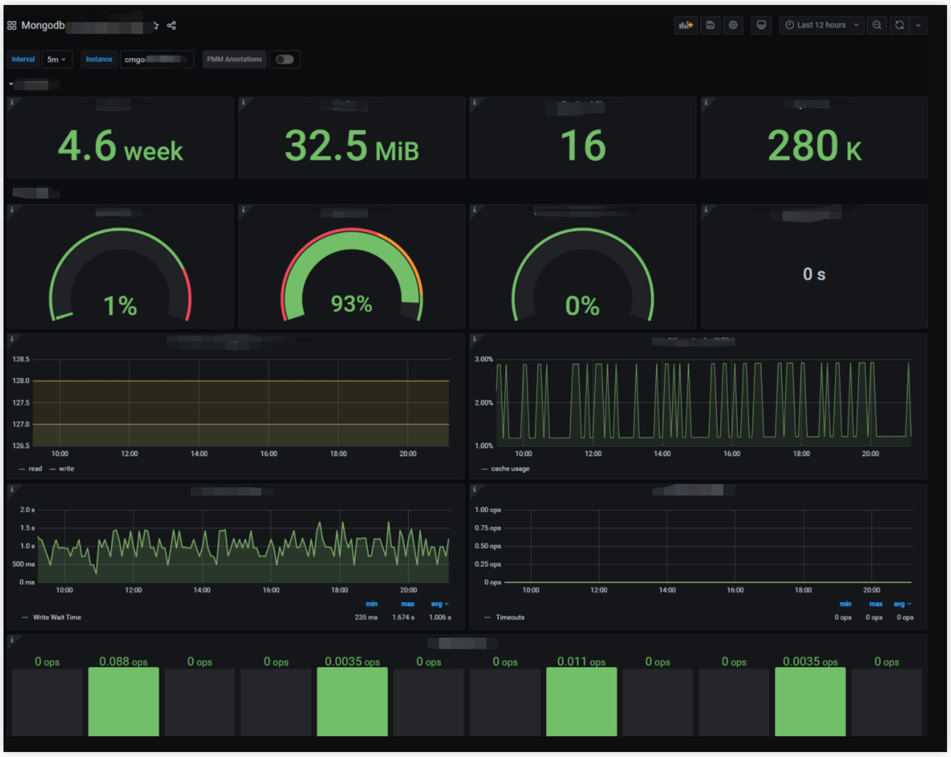
As the exporter and MongoDB are deployed on different servers, we recommend you use the Prometheus relabeling mechanism to add the MongoDB instance information to the monitoring metrics so as to locate problems more easily.

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.

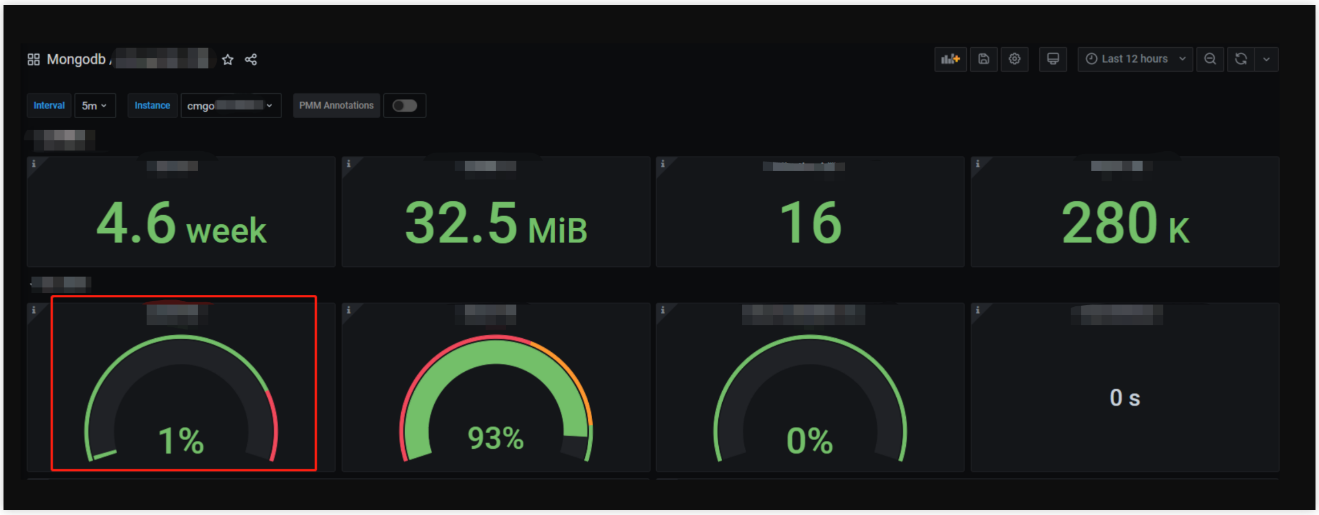
Integrating with alert feature

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

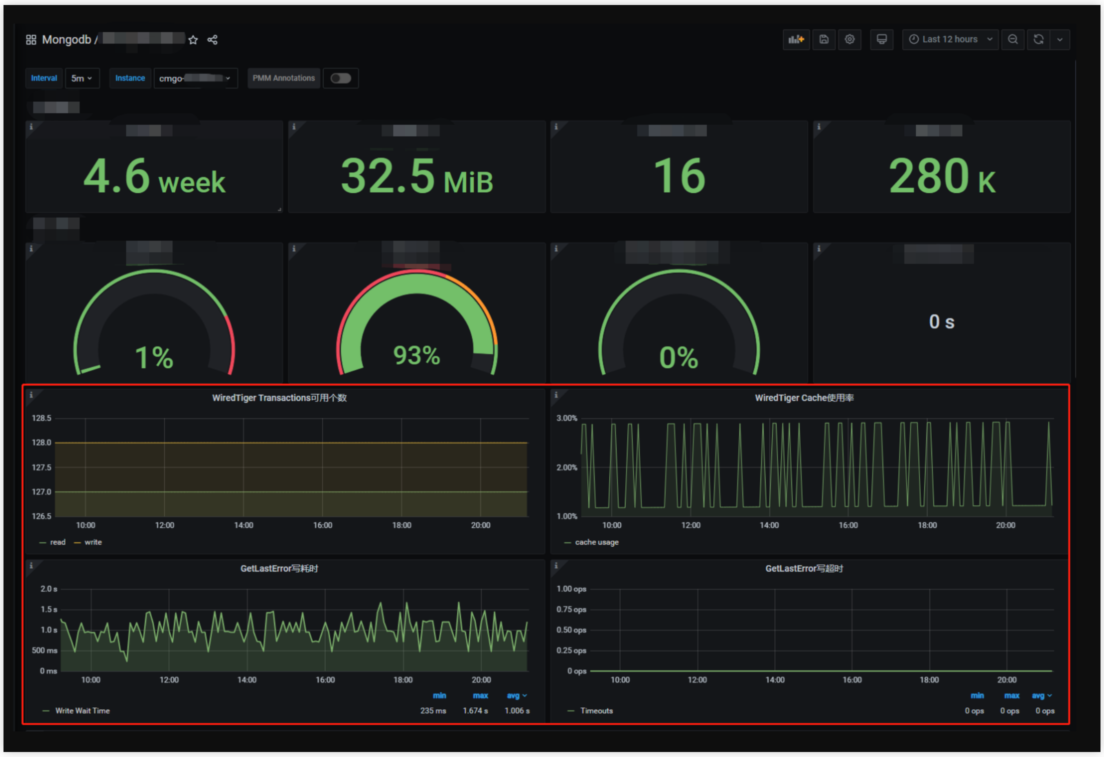
The client reported an error "client checkout connect timeout". What should I do?

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：2024-01-29 15:55:07

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 > Deploying PostgreSQL exporter.

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):

```yaml
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
sslkey	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:

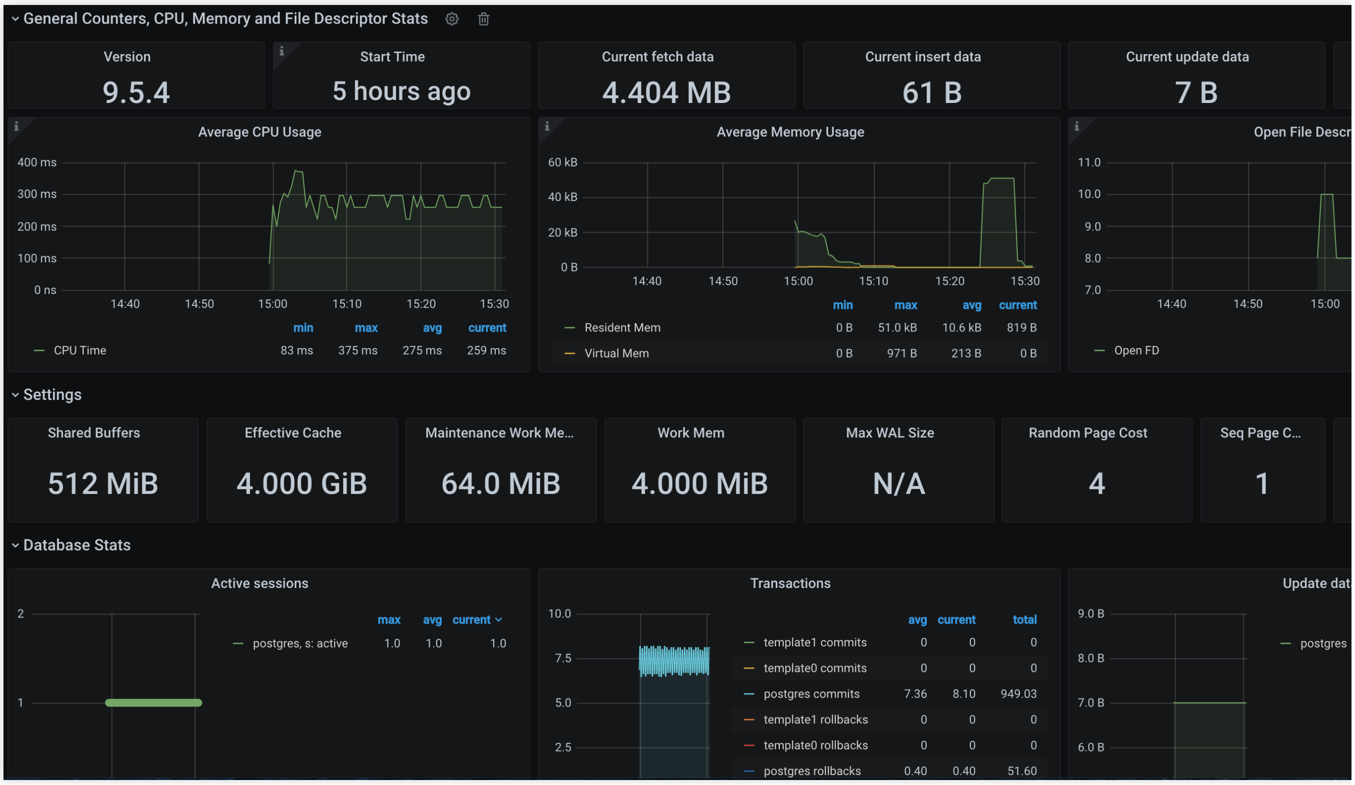
For more advanced usage, please see ServiceMonitor and PodMonitor.

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.



General Counters, CPU, Memory and File Descriptor Stats

Version	Start Time	Current fetch data	Current insert data	Current update data
9.5.4	5 hours ago	4.404 MB	61 B	7 B

Average CPU Usage

	min	max	avg	current
CPU Time	83 ms	375 ms	275 ms	259 ms

Average Memory Usage

	min	max	avg	current
Resident Mem	0 B	51.0 kB	10.6 kB	819 B
Virtual Mem	0 B	971 B	213 B	0 B

Open File Descr

Open FD

Settings

Shared Buffers	Effective Cache	Maintenance Work Me...	Work Mem	Max WAL Size	Random Page Cost	Seq Page C...
512 MiB	4.000 GiB	64.0 MiB	4.000 MiB	N/A	4	1

Database Stats

Active sessions

	max	avg	current
postgres, s: active	1.0	1.0	1.0

Transactions

	avg	current	total
template1 commits	0	0	0
template0 commits	0	0	0
postgres commits	7.36	8.10	949.03
template1 rollbacks	0	0	0
template0 rollbacks	0	0	0
postgres rollbacks	0.40	0.40	51.60

Update dat

postgres

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：2024-01-29 15:55:08

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:

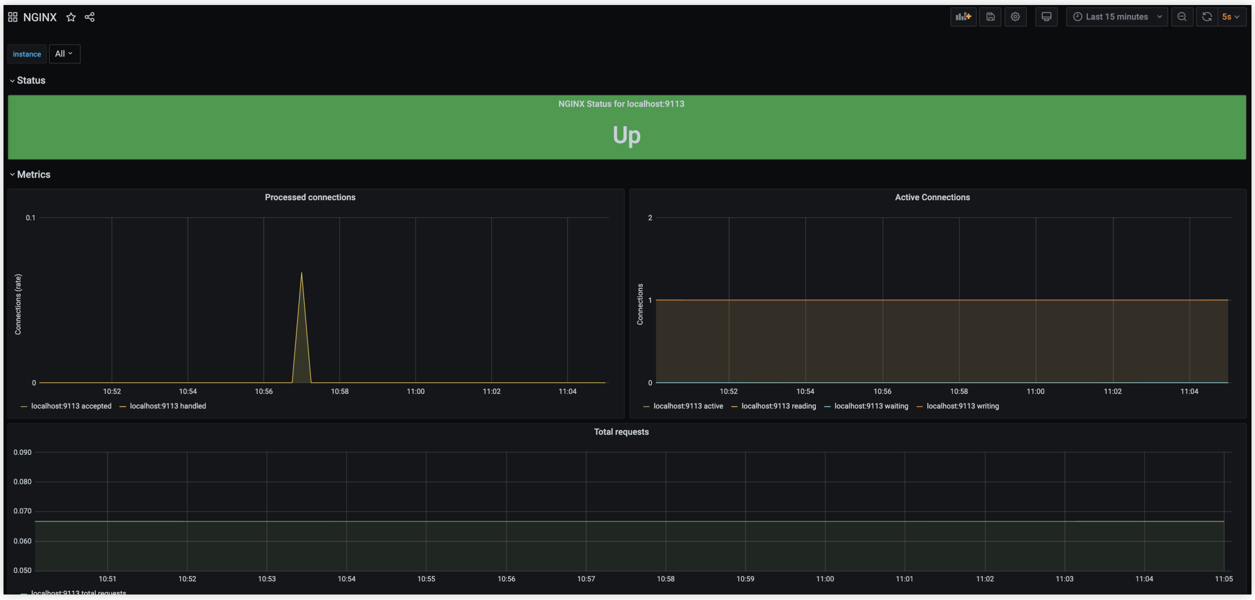
```
...
- job_name: 'mysqld_exporter'
static_configs:
- targets: ['your_exporter:port']
relabel_configs:
 - source_labels: [__address__]
   regex: '.*'
   target_label: instance
   replacement: '10.0.0.1:80'
```

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：2024-01-29 15:55:07

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:

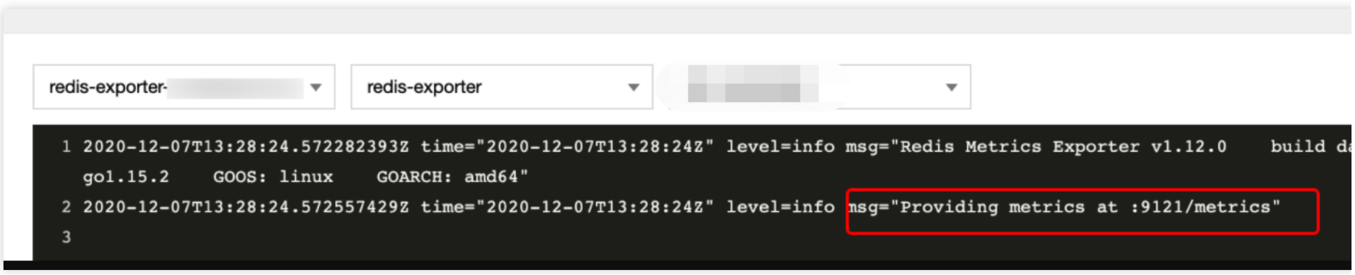
For more information on the detailed exporter parameters, please see redis_exporter.

```
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:

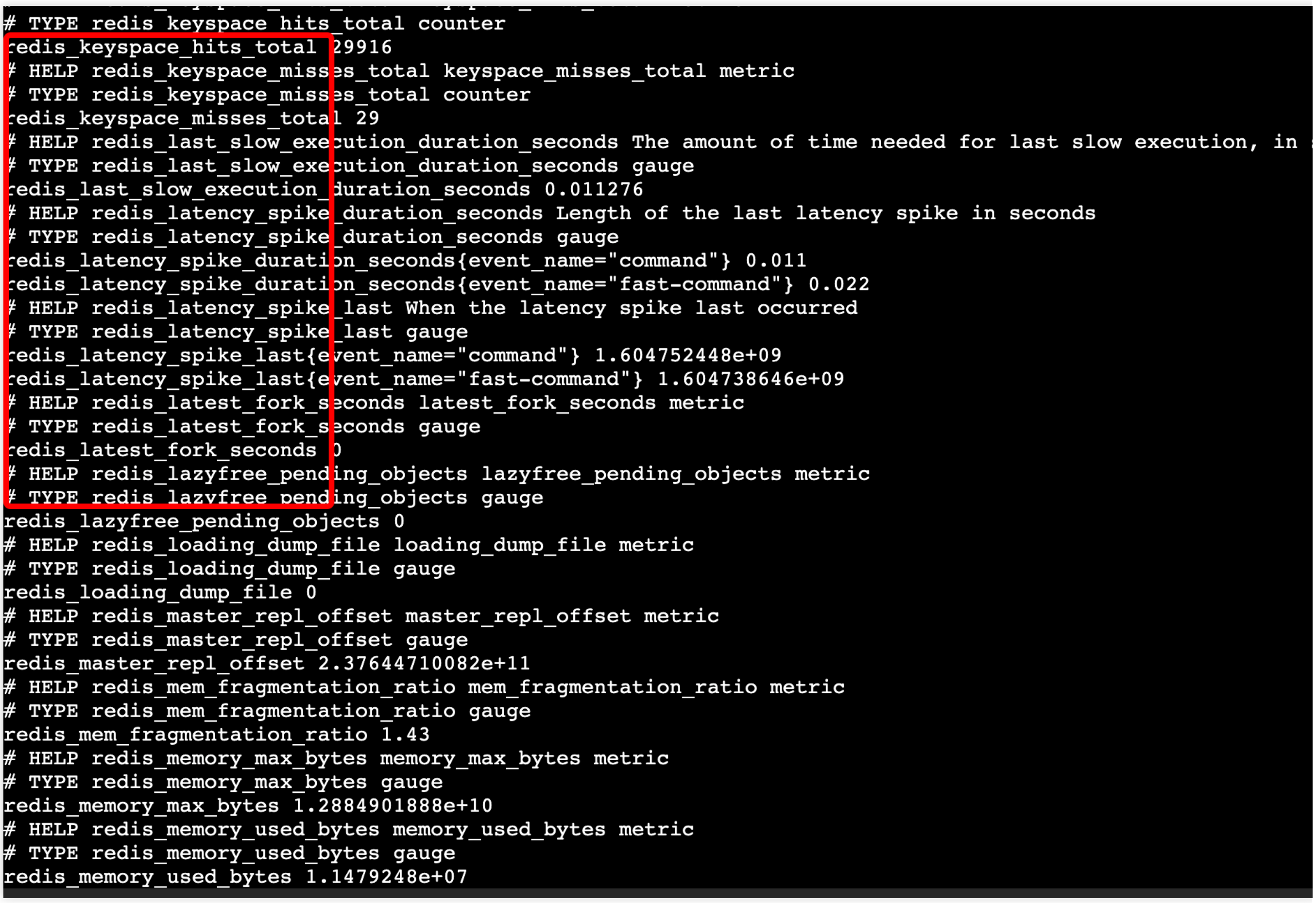


```
2020-12-07T13:28:24.572282393Z time="2020-12-07T13:28:24Z" level=info msg="Redis Metrics Exporter v1.12.0    build da
go1.15.2    GOOS: linux    GOARCH: amd64"
2020-12-07T13:28:24.572557429Z time="2020-12-07T13:28:24Z" level=info msg="Providing metrics at :9121/metrics"
```

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:

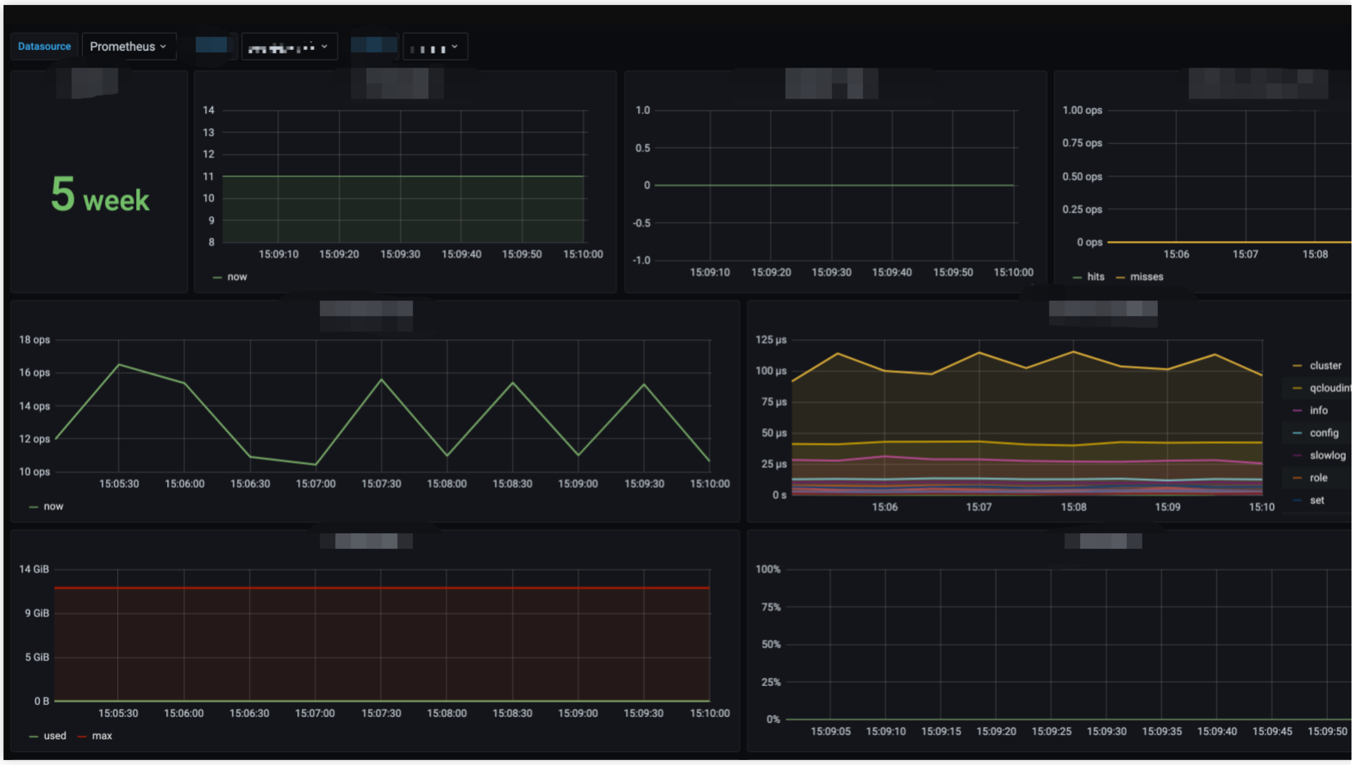
```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter # Enter a unique name
  namespace: cm-prometheus  # The namespace is fixed. Do not change it
spec:
  podMetricsEndpoints:
  - interval: 30s
    port: metric-port  # Enter the name of the corresponding port of the 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：2024-01-29 15:55:08

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.

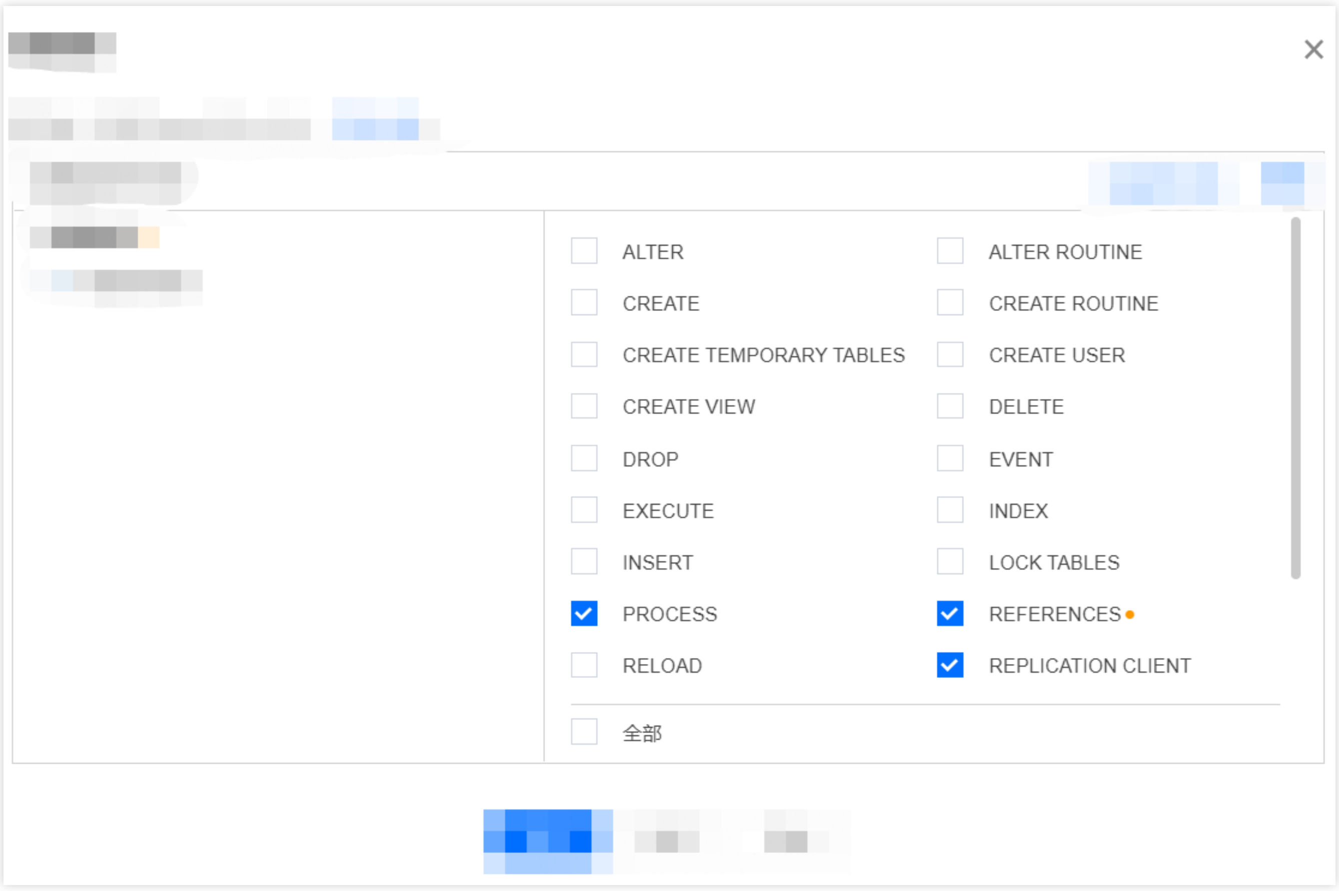
Prerequisites

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:



☐ ALTER	☐ ALTER ROUTINE
☐ CREATE	☐ CREATE ROUTINE
☐ CREATE TEMPORARY TABLES	☐ CREATE USER
☐ CREATE VIEW	☐ DELETE
☐ DROP	☐ EVENT
☐ EXECUTE	☐ INDEX
☐ INSERT	☐ LOCK TABLES
✓ PROCESS	✓ REFERENCES●
☐ RELOAD	✓ REPLICATION CLIENT
☐ 全部	

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
```

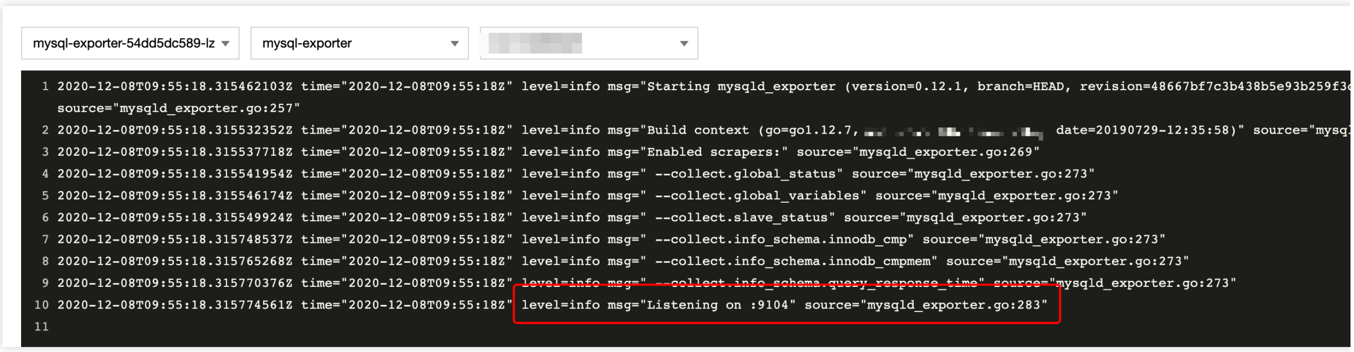
Deploying MySQL exporter

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:



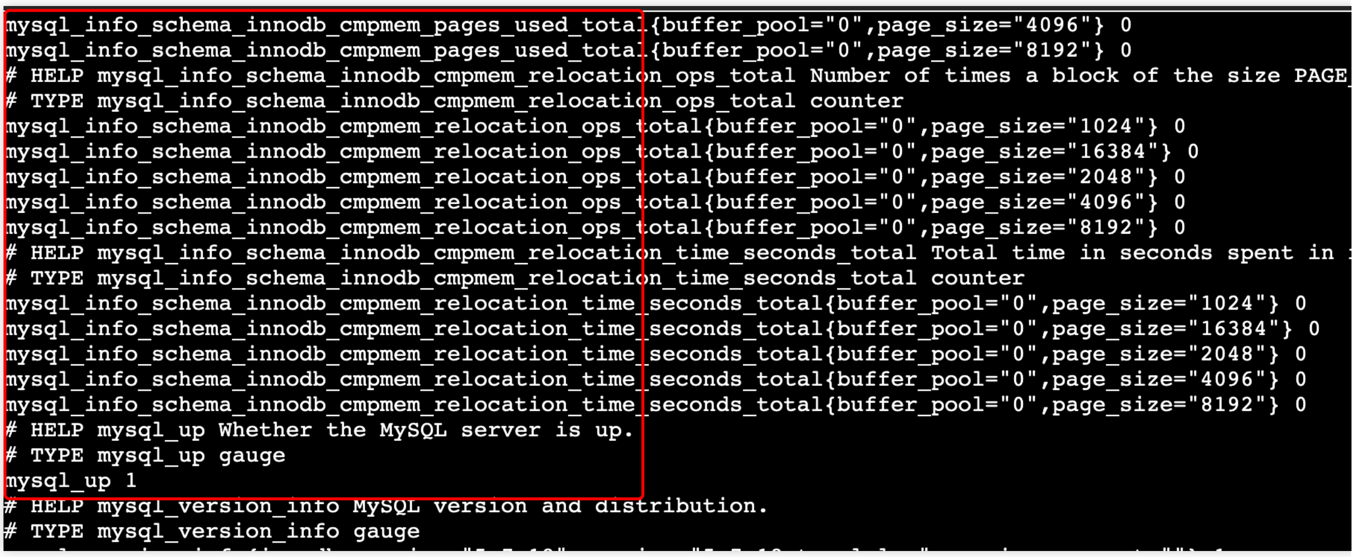
mysql-exporter-54dd5dc589-lz ▾ | mysql-exporter ▾ | [redacted] ▾

```
2020-12-08T09:55:18.315462103Z time="2020-12-08T09:55:18Z" level=info msg="Starting mysqld_exporter (version=0.12.1, branch=HEAD, revision=48667bf7c3b438b5e93b259f3c
source="mysqld_exporter.go:257"
2020-12-08T09:55:18.315532352Z time="2020-12-08T09:55:18Z" level=info msg="Build context (go=go1.12.7, [redacted] date=20190729-12:35:58)" source="mysql
2020-12-08T09:55:18.315537718Z time="2020-12-08T09:55:18Z" level=info msg="Enabled scrapers:" source="mysqld_exporter.go:269"
2020-12-08T09:55:18.315541954Z time="2020-12-08T09:55:18Z" level=info msg=" --collect.global_status" source="mysqld_exporter.go:273"
2020-12-08T09:55:18.315546174Z time="2020-12-08T09:55:18Z" level=info msg=" --collect.global_variables" source="mysqld_exporter.go:273"
2020-12-08T09:55:18.315549924Z time="2020-12-08T09:55:18Z" level=info msg=" --collect.slave_status" source="mysqld_exporter.go:273"
2020-12-08T09:55:18.315748537Z time="2020-12-08T09:55:18Z" level=info msg=" --collect.info_schema.innodb_cmp" source="mysqld_exporter.go:273"
2020-12-08T09:55:18.315765268Z time="2020-12-08T09:55:18Z" level=info msg=" --collect.info_schema.innodb_cmpmem" source="mysqld_exporter.go:273"
2020-12-08T09:55:18.315770376Z time="2020-12-08T09:55:18Z" level=info msg=" --collect.info_schema.query_response_time" source="mysqld_exporter.go:273"
2020-12-08T09:55:18.315774561Z time="2020-12-08T09:55:18Z" level=info msg="Listening on :9104" source="mysqld_exporter.go:283"
```

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:



```
mysql_info_schema_innodb_cmpmem_pages_used_total{buffer_pool="0",page_size="4096"} 0
mysql_info_schema_innodb_cmpmem_pages_used_total{buffer_pool="0",page_size="8192"} 0
# HELP mysql_info_schema_innodb_cmpmem_relocation_ops_total Number of times a block of the size PAGE_
# TYPE mysql_info_schema_innodb_cmpmem_relocation_ops_total counter
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="1024"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="16384"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="2048"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="4096"} 0
mysql_info_schema_innodb_cmpmem_relocation_ops_total{buffer_pool="0",page_size="8192"} 0
# HELP mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total Total time in seconds spent in
# TYPE mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total counter
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="1024"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="16384"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="2048"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="4096"} 0
mysql_info_schema_innodb_cmpmem_relocation_time_seconds_total{buffer_pool="0",page_size="8192"} 0
# HELP mysql_up Whether the MySQL server is up.
# TYPE mysql_up gauge
mysql_up 1
# HELP mysql_version_info MySQL version and distribution.
# TYPE mysql_version_info gauge
```

Adding scrape task

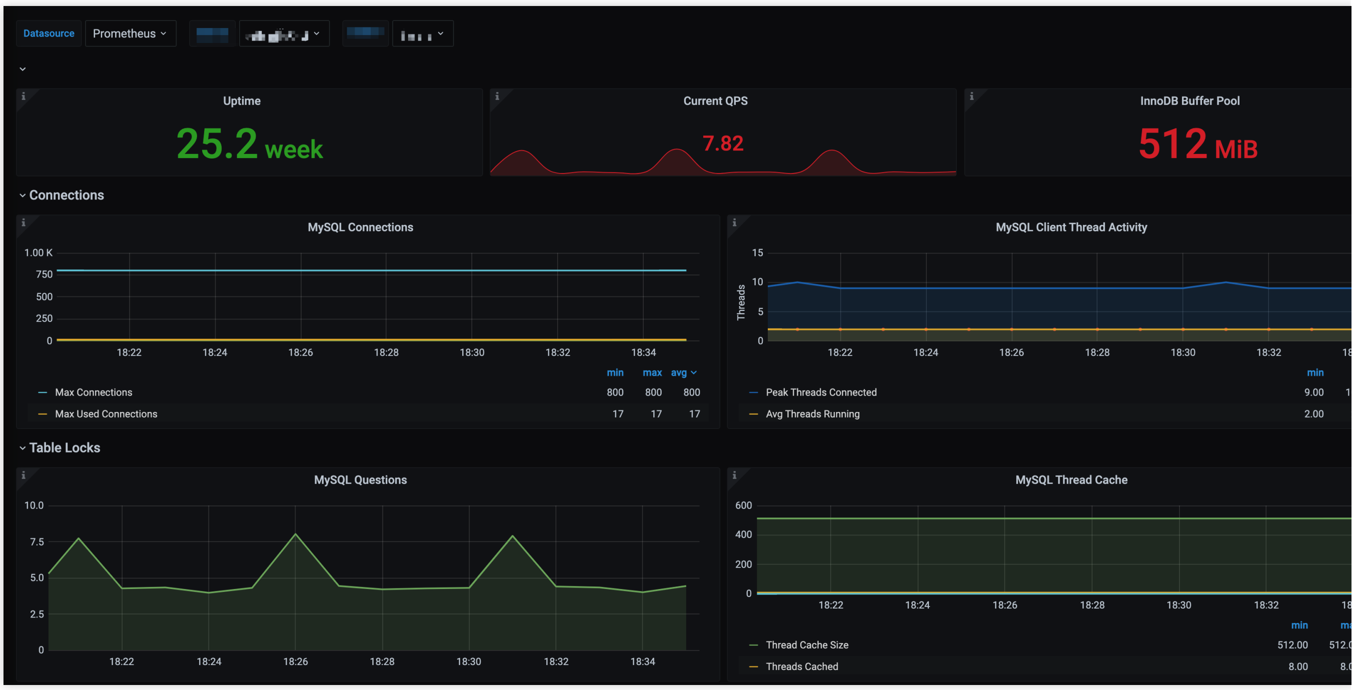
1. Log in to the TMP console and select the target TMP instance to enter the management page.
2. Click a **cluster ID** in the TKE cluster list to enter the **Integrate with TKE** page.
3. In **Scrape Configuration**, add `Pod Monitor` to define a Prometheus scrape task. Below is a sample YAML configuration:

```
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
```

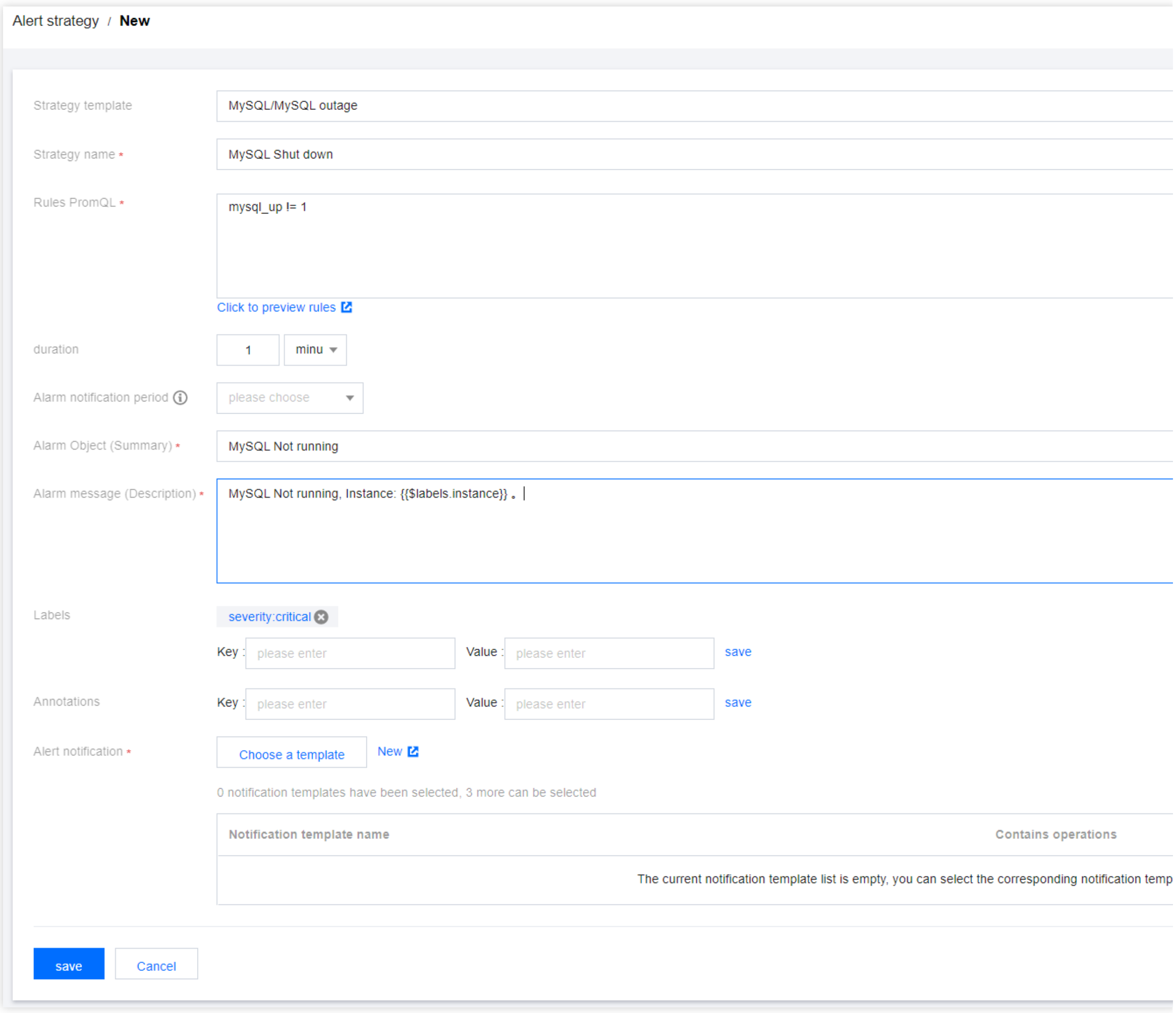
Viewing monitoring information

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.


Alert strategy / **New**

Strategy template	MySQL/MySQL outage
Strategy name *	MySQL Shut down
Rules PromQL *	mysql_up != 1
duration	1 minu
Alarm notification period	please choose
Alarm Object (Summary) *	MySQL Not running
Alarm message (Description) *	MySQL Not running, Instance: {{$labels.instance}} 。
Labels	severity:critical
Annotations	
Alert notification *	0 notification templates have been selected, 3 more can be selected

Notification template name	Contains operations
The current notification template list is empty, you can select the corresponding notification temp	

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 threa `information_schema.processli`
collect.info_schema.processlist.min_time	5.1	Minimum time a thread must be in each
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：2024-01-29 15:55:08

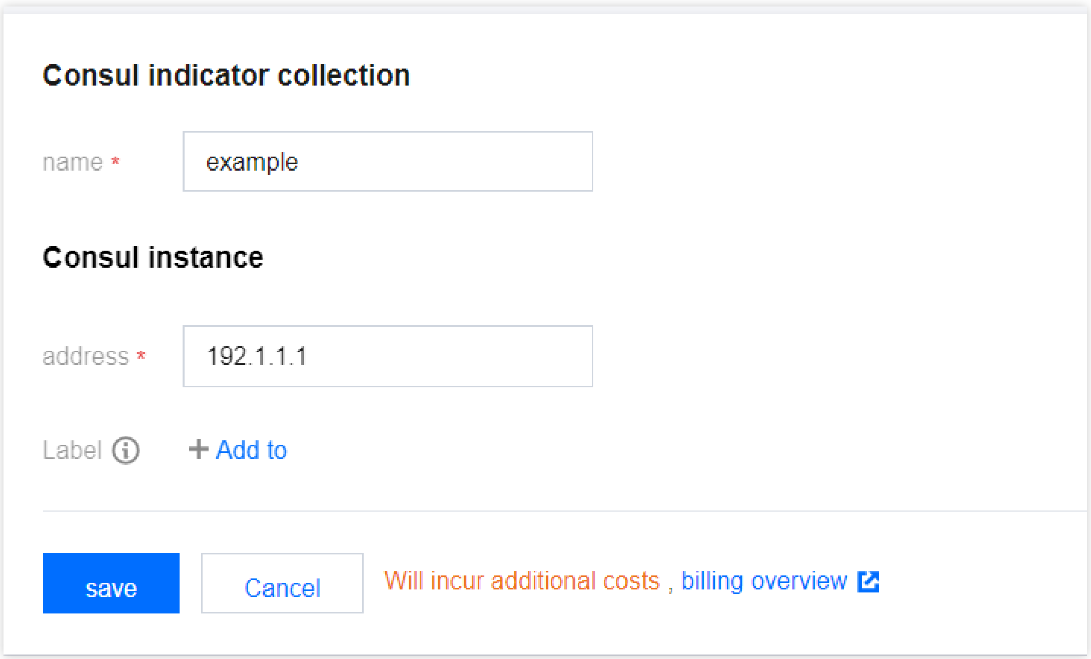
Overview

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



Consul indicator collection

name * example

Consul instance

address * 192.1.1.1

Label + Add to

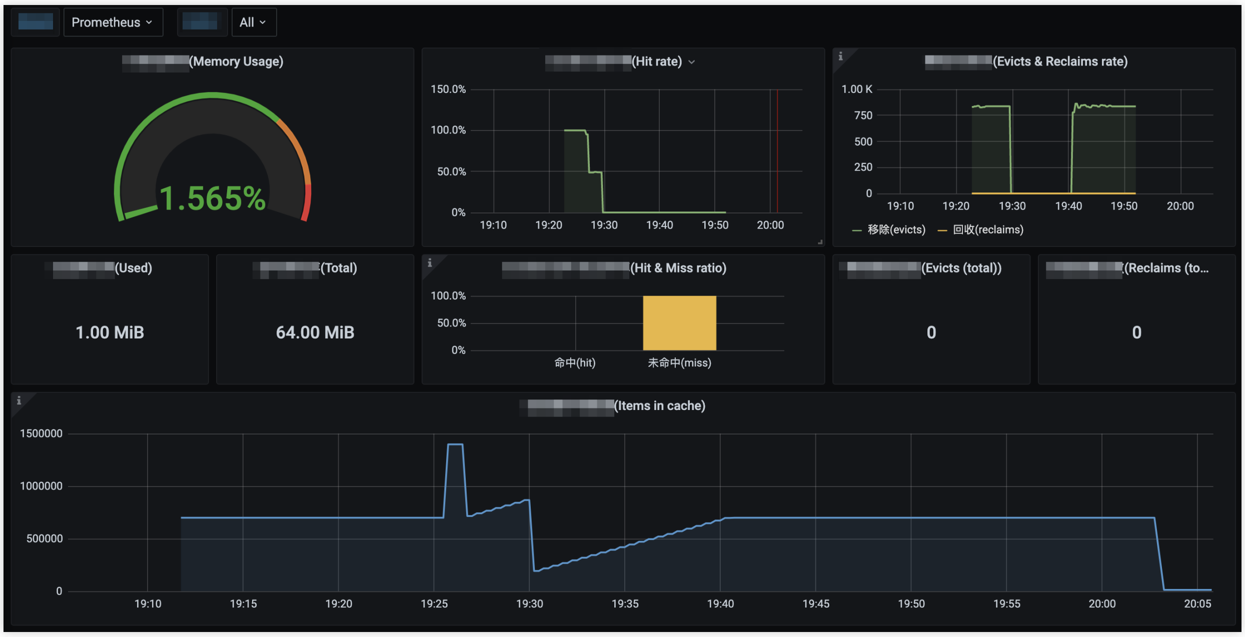
Item	Description
Name	Unique integration name

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：2024-01-29 15:55:08

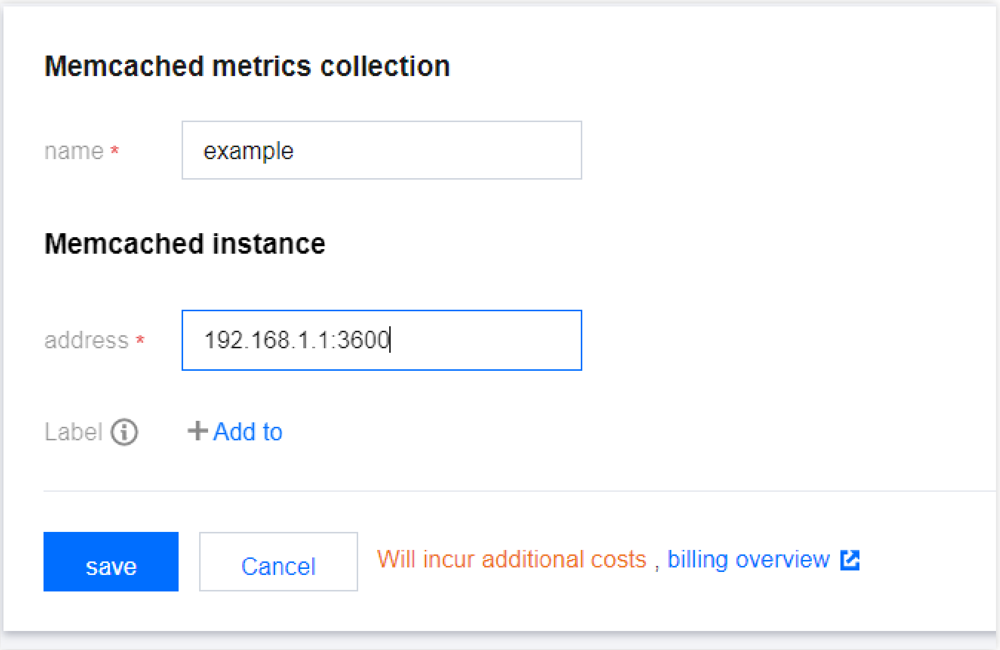
Overview

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



Memcached metrics collection

name * example

Memcached instance

address * 192.168.1.1:3600

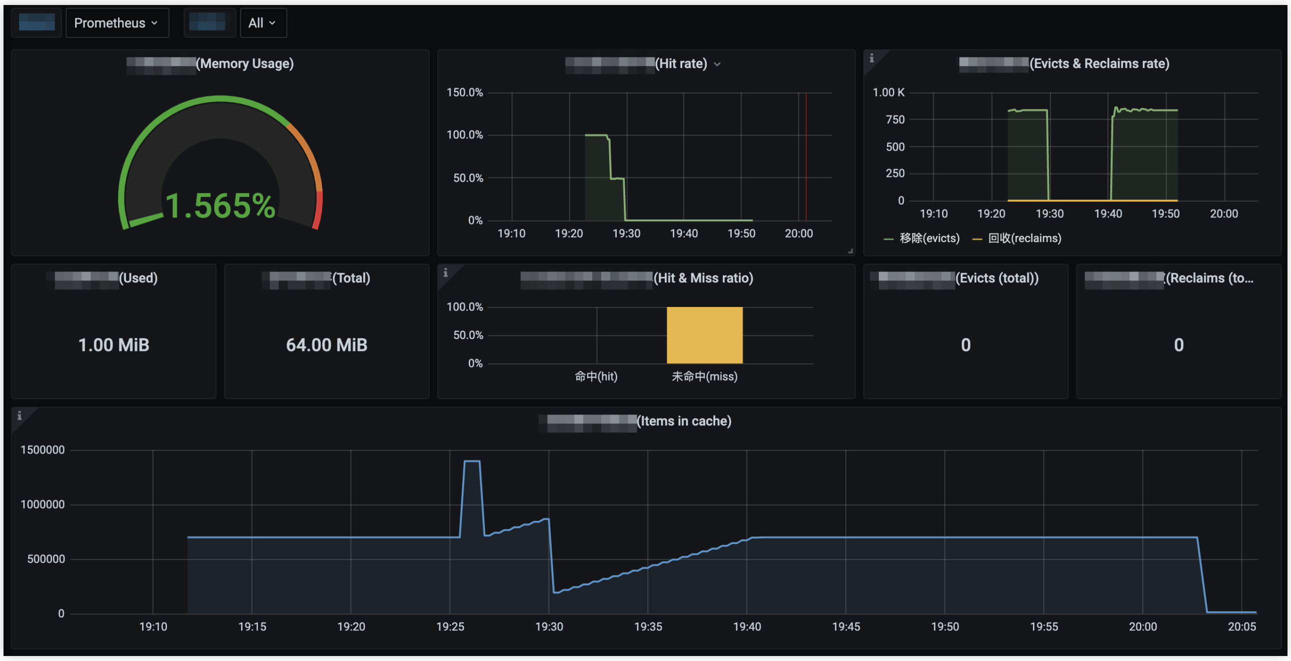
Item	Description
Name	Unique integration name

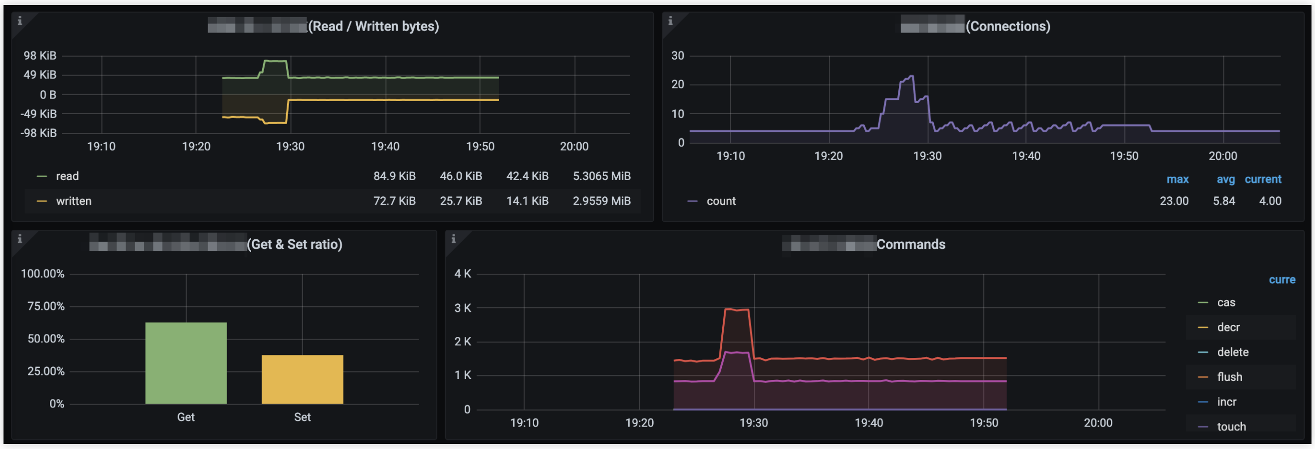
Address	Address and port of the Memcached instance to be collected
Label	Label with business meaning, which will be automatically added to Prometheus labels

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.




[redacted](Read / Written bytes)
98 KiB
49 KiB
0 B
-49 KiB
-98 KiB
19:10
19:20
19:30
19:40
19:50
20:00
read 84.9 KiB 46.0 KiB 42.4 KiB 5.3065 MiB
written 72.7 KiB 25.7 KiB 14.1 KiB 2.9559 MiB
[redacted](Connections)
30
20
10
0
19:10
19:20
19:30
19:40
19:50
20:00
max avg current
count 23.00 5.84 4.00
[redacted](Get & Set ratio)
100.00%
75.00%
50.00%
25.00%
0%
Get
Set
[redacted]Commands
4 K
3 K
2 K
1 K
0
19:10
19:20
19:30
19:40
19:50
20:00
curre
cas
decr
delete
flush
incr
touch

Integration with Other Exporters

Last updated：2024-01-29 15:55:07

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：2024-01-29 15:55:08

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

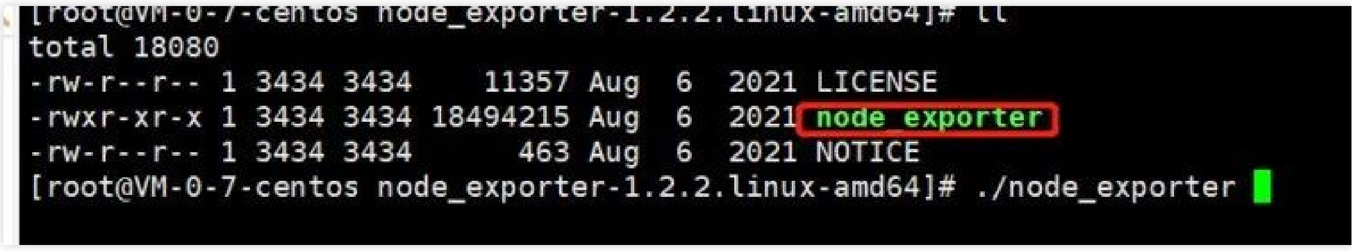
Directions

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:



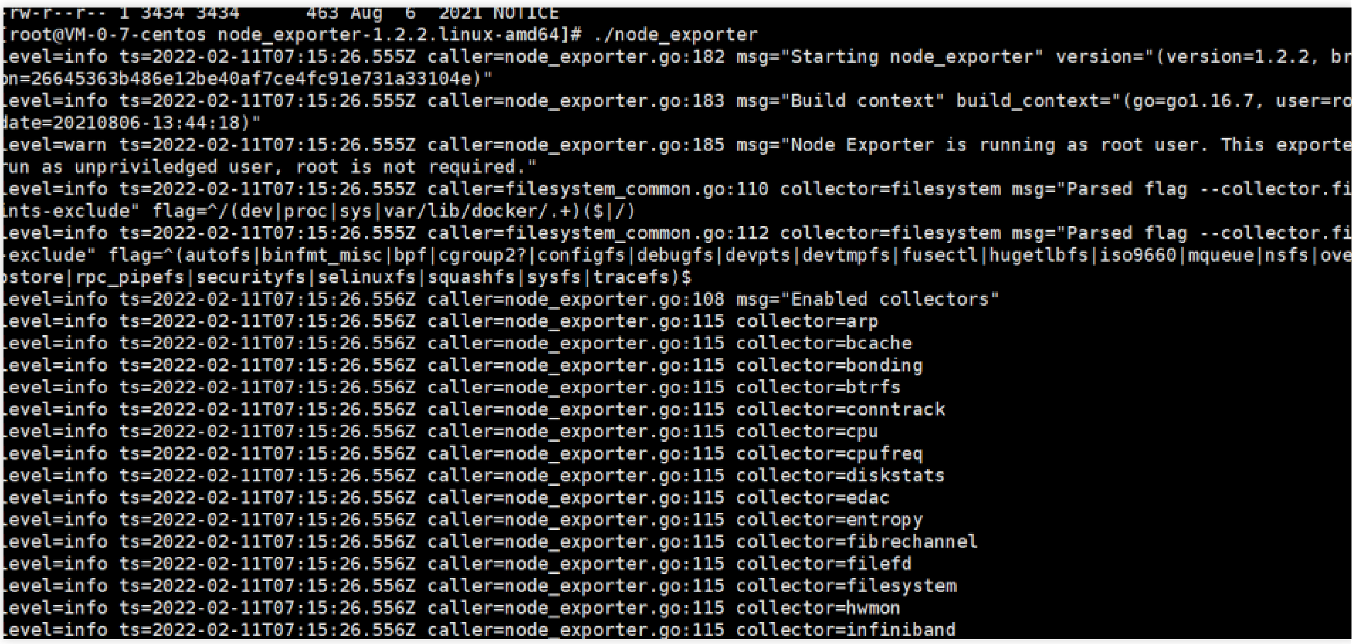
```
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ll
total 18080
-rw-r--r-- 1 3434 3434    11357 Aug  6  2021 LICENSE
-rwxr-xr-x 1 3434 3434 18494215 Aug  6  2021 node_exporter
-rw-r--r-- 1 3434 3434      463 Aug  6  2021 NOTICE
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ./node_exporter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

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.

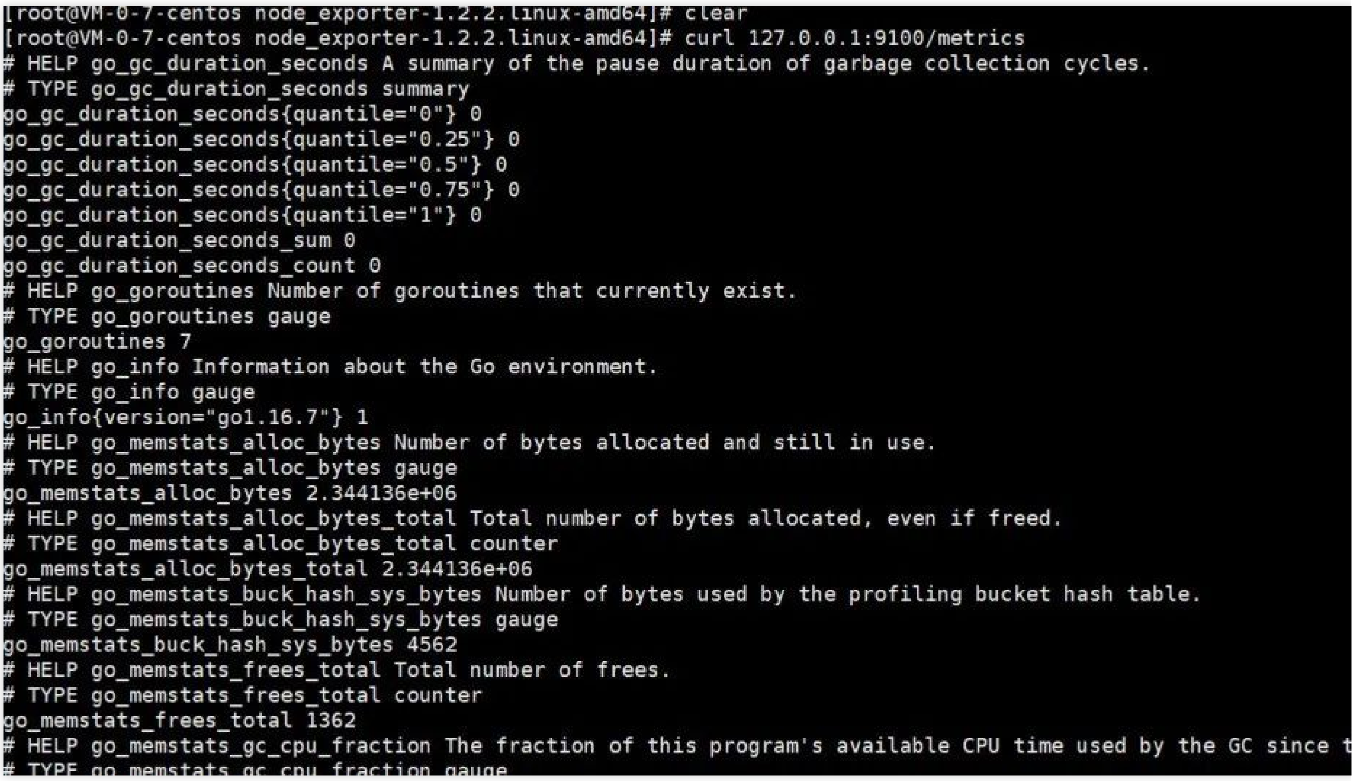


```
-rw-r--r-- 1 3434 3434      463 Aug  6  2021 NOTICE
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# ./node_exporter
level=info ts=2022-02-11T07:15:26.555Z caller=node_exporter.go:182 msg="Starting node_exporter" version="(version=1.2.2, br
on=26645363b486e12be40af7ce4fc91e731a33104e)"
level=info ts=2022-02-11T07:15:26.555Z caller=node_exporter.go:183 msg="Build context" build_context="(go=go1.16.7, user=ro
date=20210806-13:44:18)"
level=warn ts=2022-02-11T07:15:26.555Z caller=node_exporter.go:185 msg="Node Exporter is running as root user. This exporte
run as unpriviledged user, root is not required."
level=info ts=2022-02-11T07:15:26.555Z caller=filesystem_common.go:110 collector=filesystem msg="Parsed flag --collector.fi
ints-exclude" flag=^/(dev|proc|sys|var/lib/docker/.+)($|/)
level=info ts=2022-02-11T07:15:26.555Z caller=filesystem_common.go:112 collector=filesystem msg="Parsed flag --collector.fi
-exclude" flag=^(autofs|binfmt_misc|bpf|cgroup2?|configfs|debugfs|devpts|devtmpfs|fusectl|hugetlbfs|iso9660|mqueue|nsfs|ove
pstore|rpc_pipefs|securityfs|selinuxfs|squashfs|sysfs|tracefs)$
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:108 msg="Enabled collectors"
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=arp
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=bcache
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=bonding
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=btrfs
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=conntrack
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=cpu
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=cpufreq
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=diskstats
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=edac
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=entropy
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=fibrechannel
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=filefd
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=filesystem
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=hwmon
level=info ts=2022-02-11T07:15:26.556Z caller=node_exporter.go:115 collector=infiniband
```

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.



```
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# clear
[root@VM-0-7-centos node_exporter-1.2.2.linux-amd64]# curl 127.0.0.1:9100/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.16.7"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.344136e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 2.344136e+06
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 4562
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 1362
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since t
# TYPE go_memstats_gc_cpu_fraction gauge
```

Step 3. Configure the collection

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

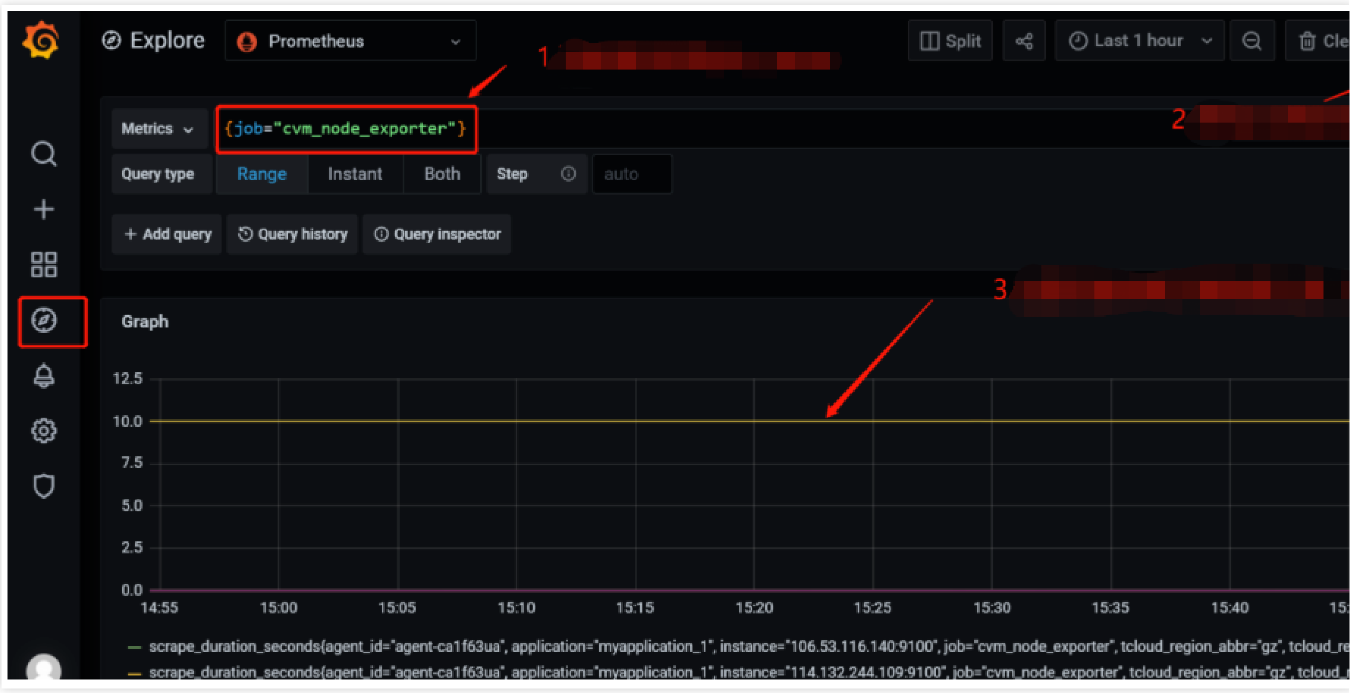
Below is a sample configuration of a scrape task:

```
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.

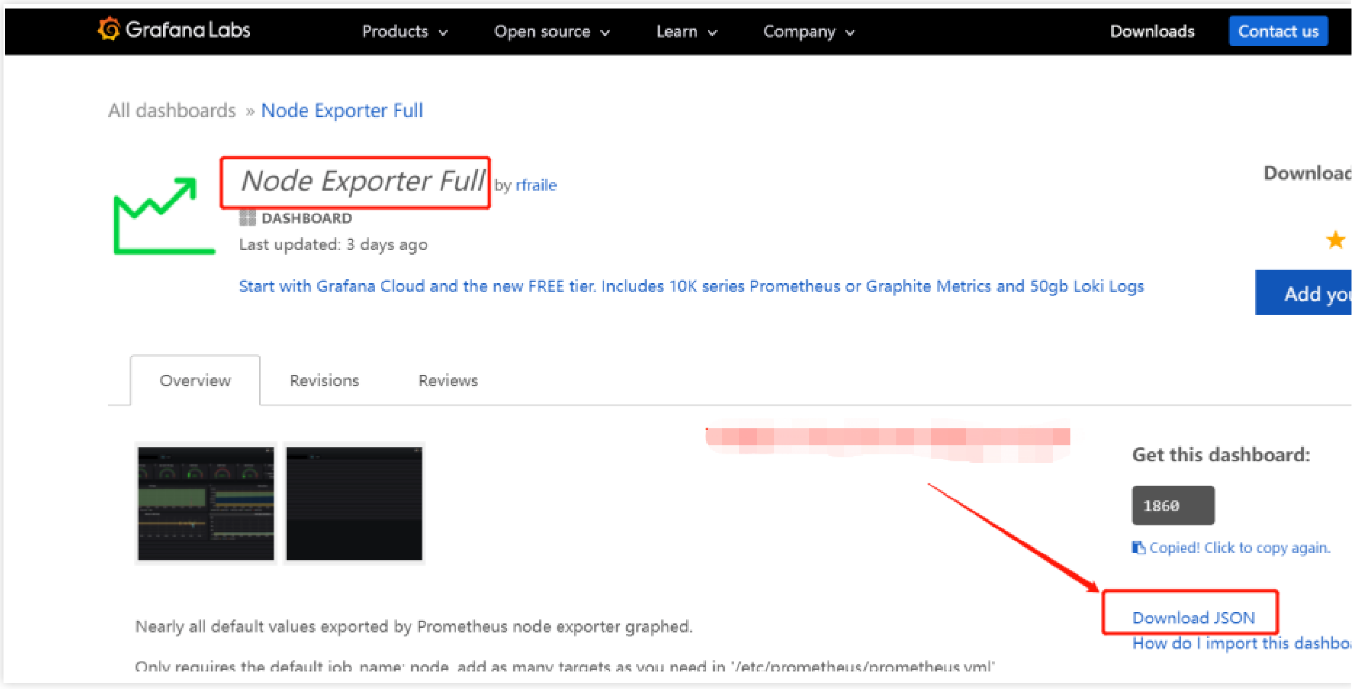


```
— scrape_duration_seconds{agent_id="agent-ca1f63ua", application="myapplication_1", instance="106.53.116.140:9100", job="cvm_node_exporter", tcloud_region_abbr="gz", tcloud_re
— scrape_duration_seconds{agent_id="agent-ca1f63ua", application="myapplication_1", instance="114.132.244.109:9100", job="cvm_node_exporter", tcloud_region_abbr="gz", tcloud_
```

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.



All dashboards » Node Exporter Full

Node Exporter Full by rfraile

DASHBOARD

Last updated: 3 days ago

Start with Grafana Cloud and the new FREE tier. Includes 10K series Prometheus or Graphite Metrics and 50gb Loki Logs

[redacted]

Get this dashboard:

1860

Copied! Click to copy again.

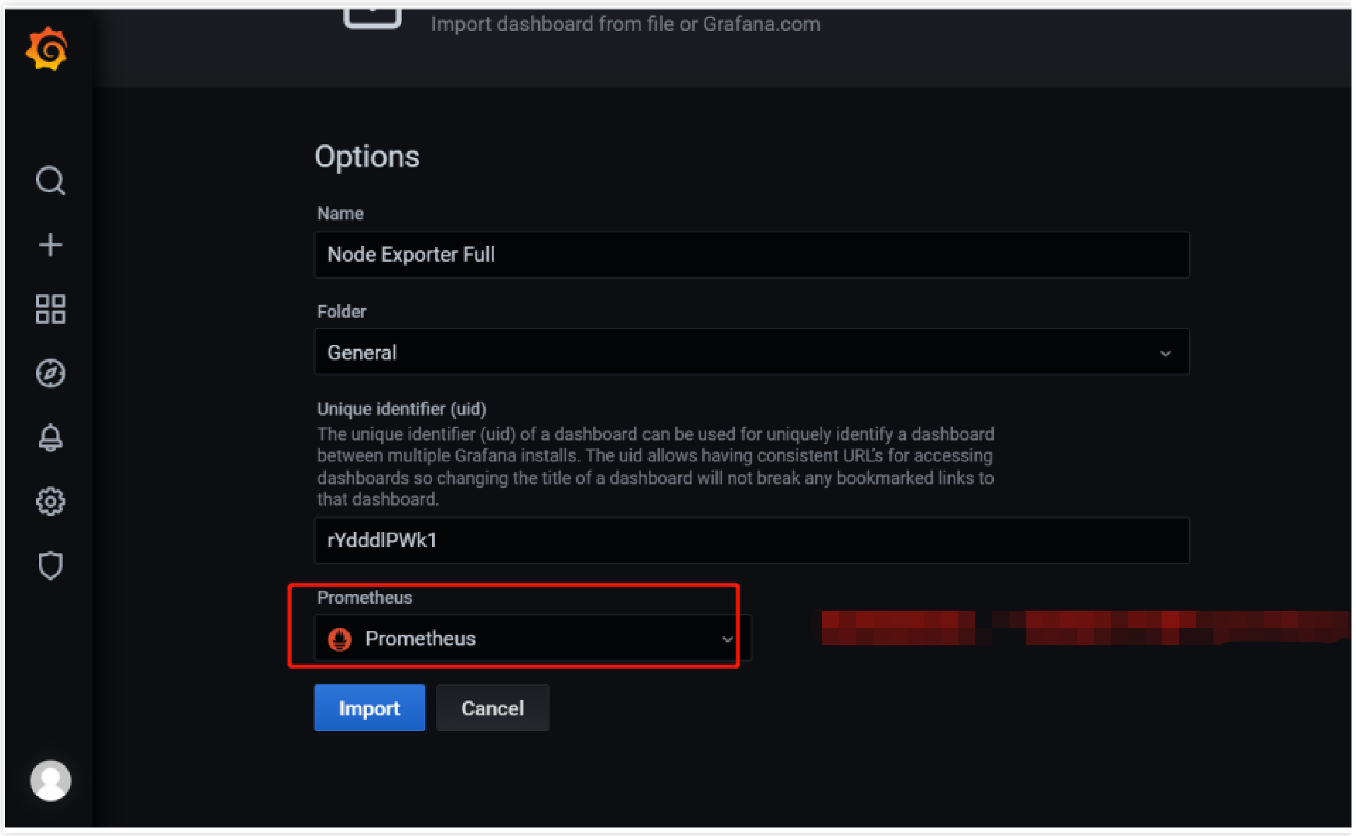
Download JSON

How do I import this dashbo

Nearly all default values exported by Prometheus node exporter graphed.

Only requires the default job_name: node, add as many targets as you need in '/etc/prometheus/prometheus.yml'.

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**.


Import dashboard from file or Grafana.com

Options

Name

Node Exporter Full

Folder

General

Unique identifier (uid)

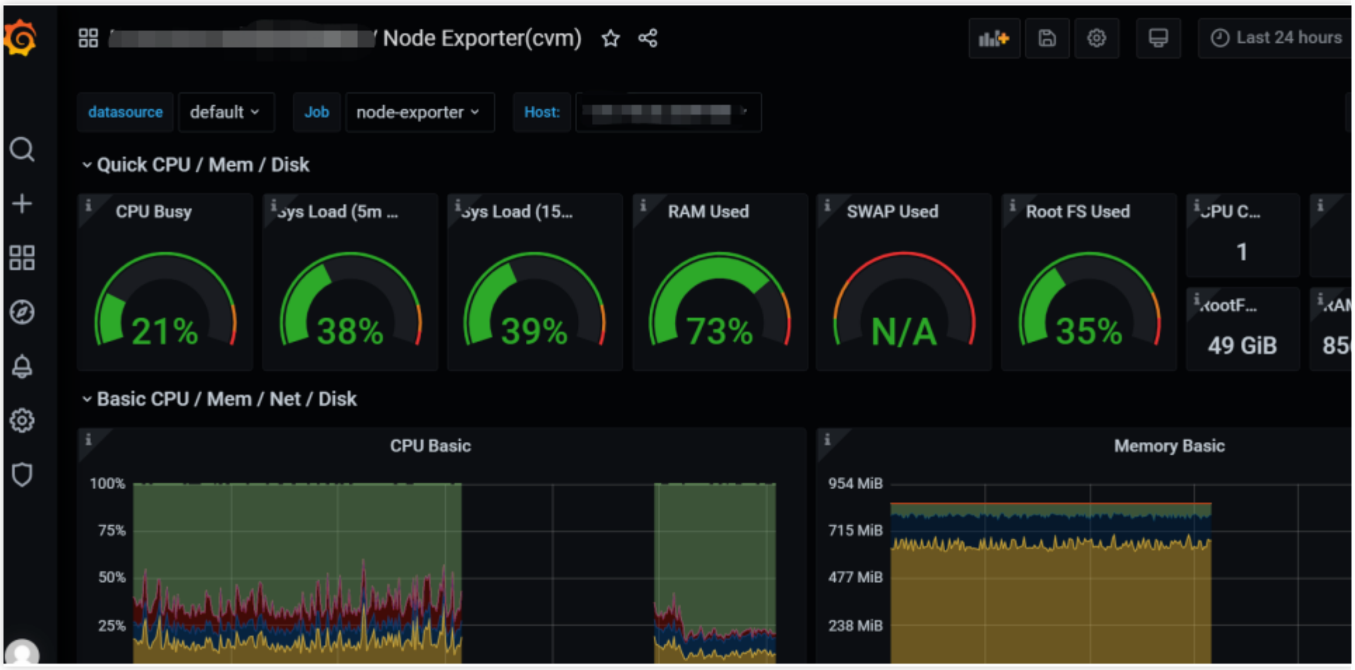
The unique identifier (uid) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The uid allows having consistent URL's for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

rYdddlPWk1

Prometheus

Prometheus

[redacted]


[redacted] / Node Exporter(cvm)
Last 24 hours
datasource
default
Job
node-exporter
Host: [redacted]
Quick CPU / Mem / Disk
CPU Busy
21%
Sys Load (5m ...
38%
Sys Load (15...
39%
RAM Used
73%
SWAP Used
N/A
Root FS Used
35%
CPU C...
1
RootF...
49 GiB
RAM
85
Basic CPU / Mem / Net / Disk
CPU Basic
100%
75%
50%
25%
Memory Basic
954 MiB
715 MiB
477 MiB
238 MiB

Health Check

Last updated：2024-01-29 15:55:08

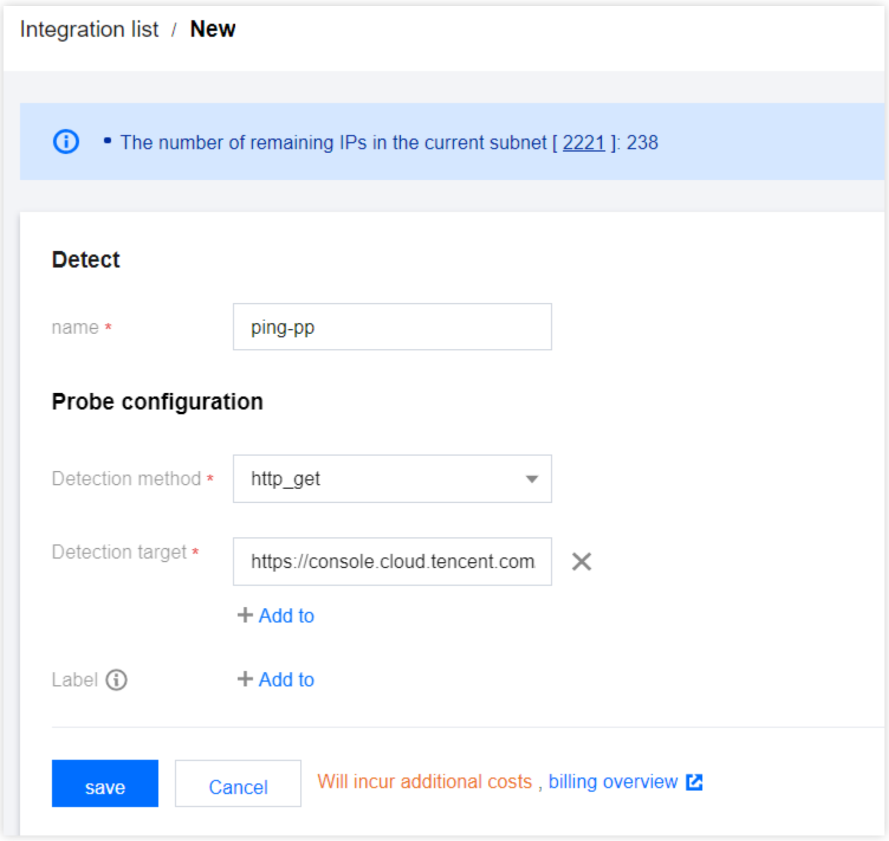
Overview

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


Integration list / **New**

• The number of remaining IPs in the current subnet [2221]: 238

Detect

name * ping-pp

Probe configuration

Detection method * http_get

Detection target * https://console.cloud.tencent.com

+ Add to

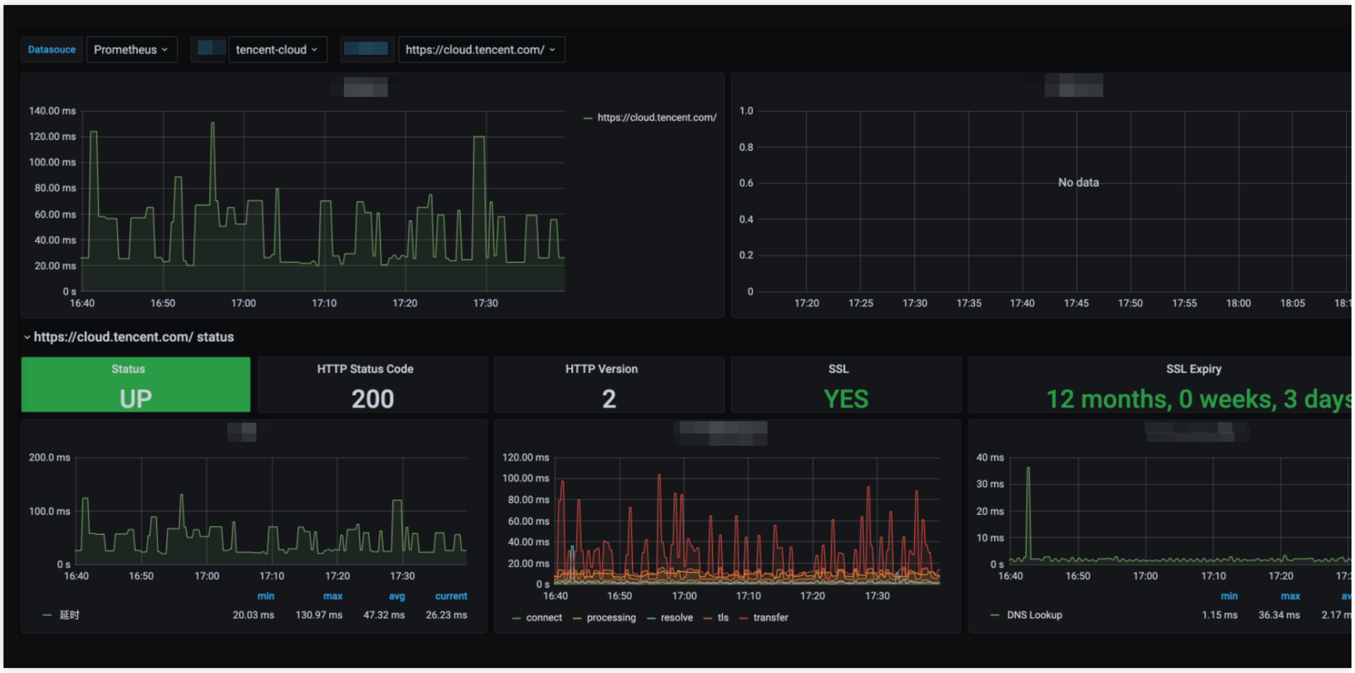
Label + Add to

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
tenant-cloud
https://cloud.tencent.com/
https://cloud.tencent.com/
No data
https://cloud.tencent.com/ status
connect
processing
resolve
tls
transfer
DNS Lookup

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
DNS Lookup	1.15 ms	36.34 ms	2.17 m	

Instructions for Installing Components in the TKE Cluster

Last updated：2024-07-23 17:53:35

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.

Cloud Monitoring

Last updated：2024-01-29 15:55:08

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：2024-01-29 15:55:08

Overview

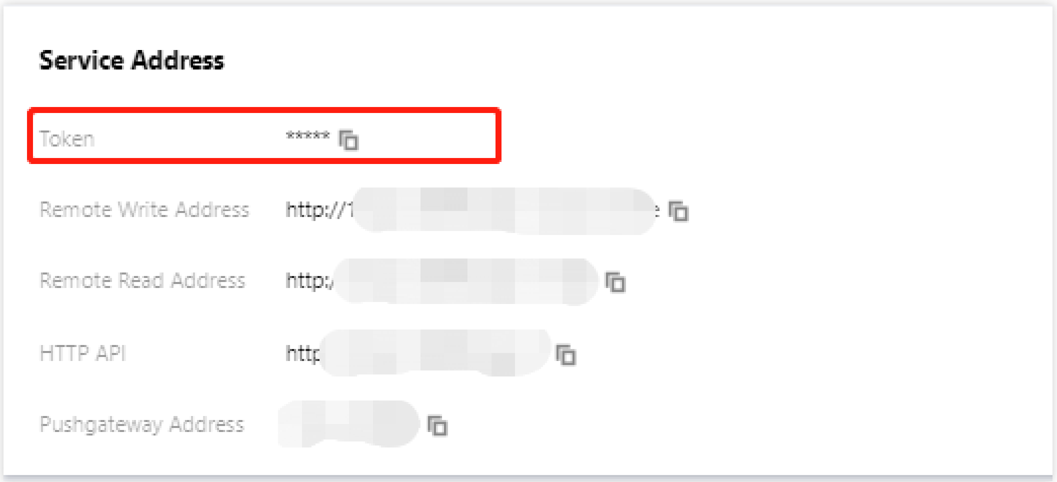
TMP provides the remote read API, which supports organizing a series of data sources of the Prometheus protocol into a single data source for query. This document describes how to use self-built Prometheus to read data from a cloud-managed TMP instance through the remote read API.

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

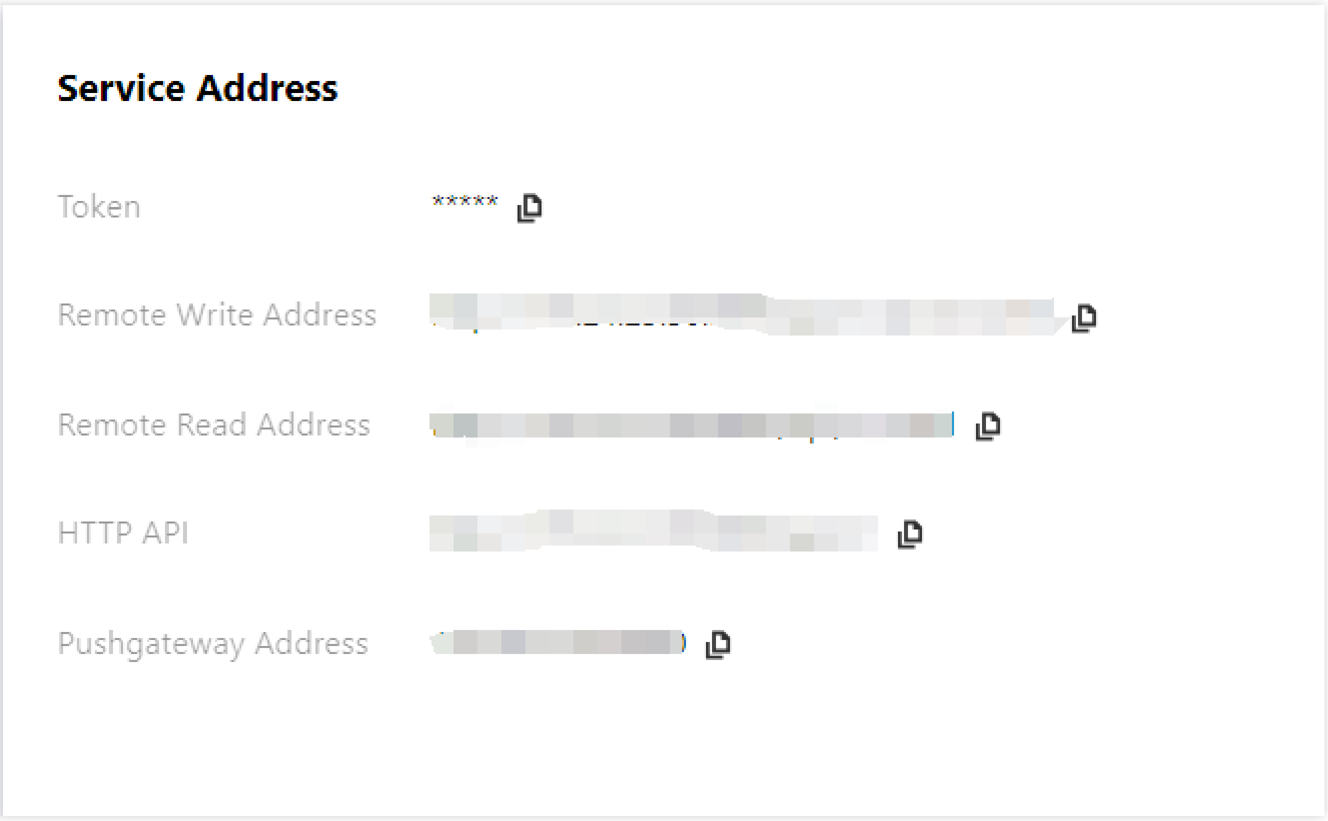
Last updated：2024-08-15 17:08:56

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.



Service Address

Token	*****
Remote Write Address	
Remote Read Address	
HTTP API	
Pushgateway Address	

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
```

Security Group Open Description

Last updated：2024-08-15 17:08:56

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

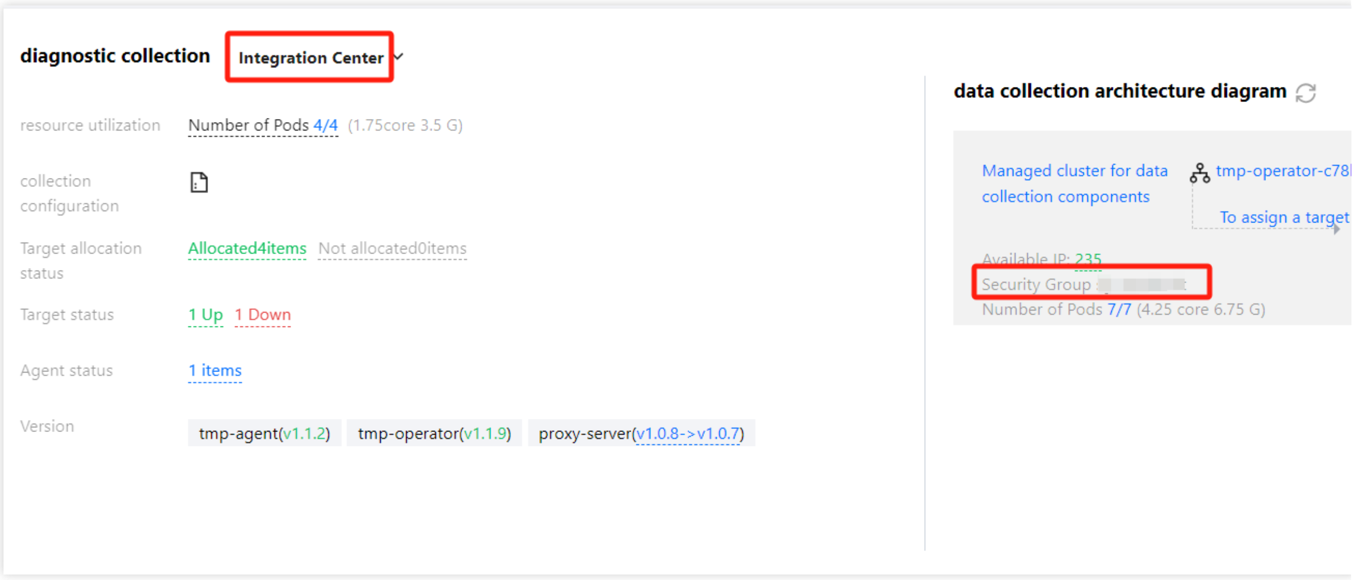
Port Description

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.


diagnostic collection Integration Center

resource utilization	Number of Pods 4/4 (1.75core 3.5 G)
collection configuration	
Target allocation status	Allocated4items Not allocated0items
Target status	1 Up 1 Down
Agent status	1 items
Version	tmp-agent(v1.1.2) tmp-operator(v1.1.9) proxy-server(v1.0.8->v1.0.7)

data collection architecture diagram

Managed cluster for data collection components
tmp-operator-c78
To assign a target
Available IP: 235
Security Group [redacted]
Number of Pods 7/7 (4.25 core 6.75 G)

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

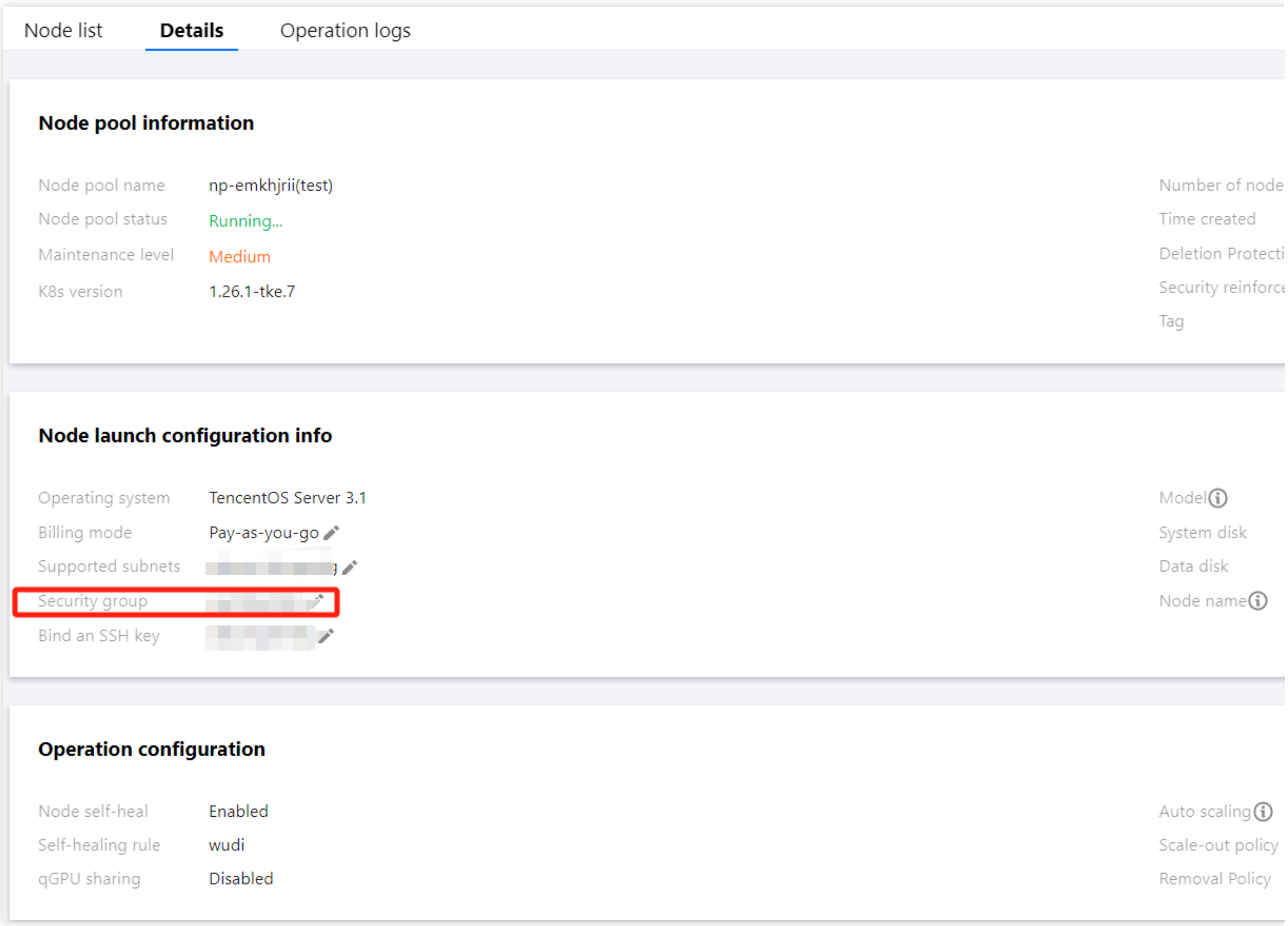
rule	cluster			opening ports.
Inbound rule	Independent cluster	TCP: 9092, 8180, 443, 10249, 9100, 60002, 10252, 10257, 10259, and 10251	Allow	The independent cluster needs to open additional master node-related ports to ensure proxy-agent can pull master node-related monitoring data

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.



Node pool information

Node pool name	np-emkhjrii(test)	Number of node	
Node pool status	Running...	Time created	
Maintenance level	Medium	Deletion Protecti	
K8s version	1.26.1-tke.7	Security reinforc	
		Tag	

Node launch configuration info

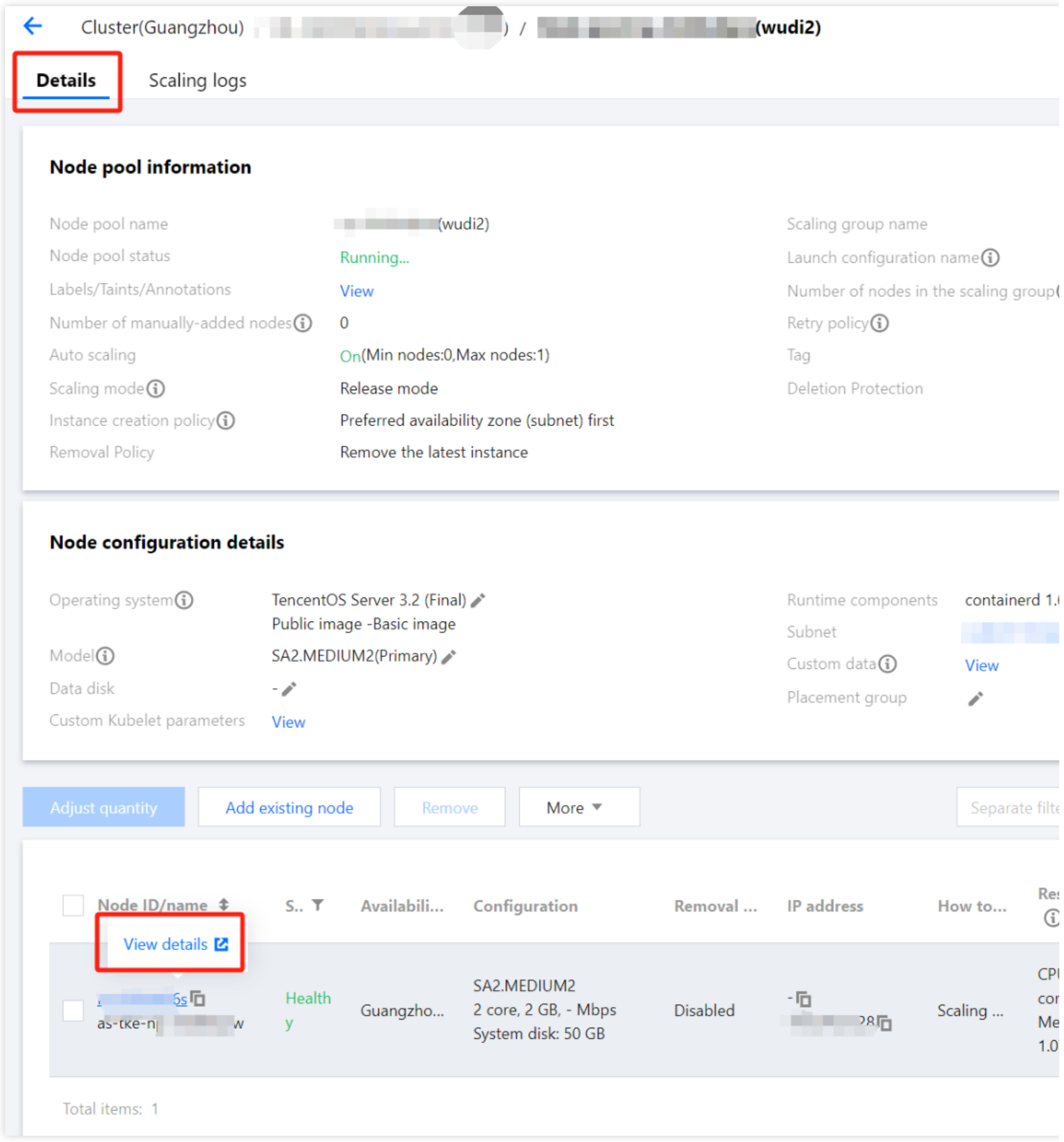
Operating system	TencentOS Server 3.1	Model	
Billing mode	Pay-as-you-go	System disk	
Supported subnets	[redacted]	Data disk	
Security group	[redacted]	Node name	
Bind an SSH key	[redacted]		

Operation configuration

Node self-heal	Enabled	Auto scaling	
Self-healing rule	wudi	Scale-out policy	
qGPU sharing	Disabled	Removal Policy	

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**:



Cluster(Guangzhou) [redacted]) / [redacted](wudi2)

Details   Scaling logs

Node pool information

Node pool name	[redacted](wudi2)	Scaling group name	
Node pool status	Running...	Launch configuration nameⓘ	
Labels/Taints/Annotations	View	Number of nodes in the scaling group	
Number of manually-added nodesⓘ	0	Retry policyⓘ	
Auto scaling	On(Min nodes:0,Max nodes:1)	Tag	
Scaling modeⓘ	Release mode	Deletion Protection	
Instance creation policyⓘ	Preferred availability zone (subnet) first		
Removal Policy	Remove the latest instance		

Node configuration details

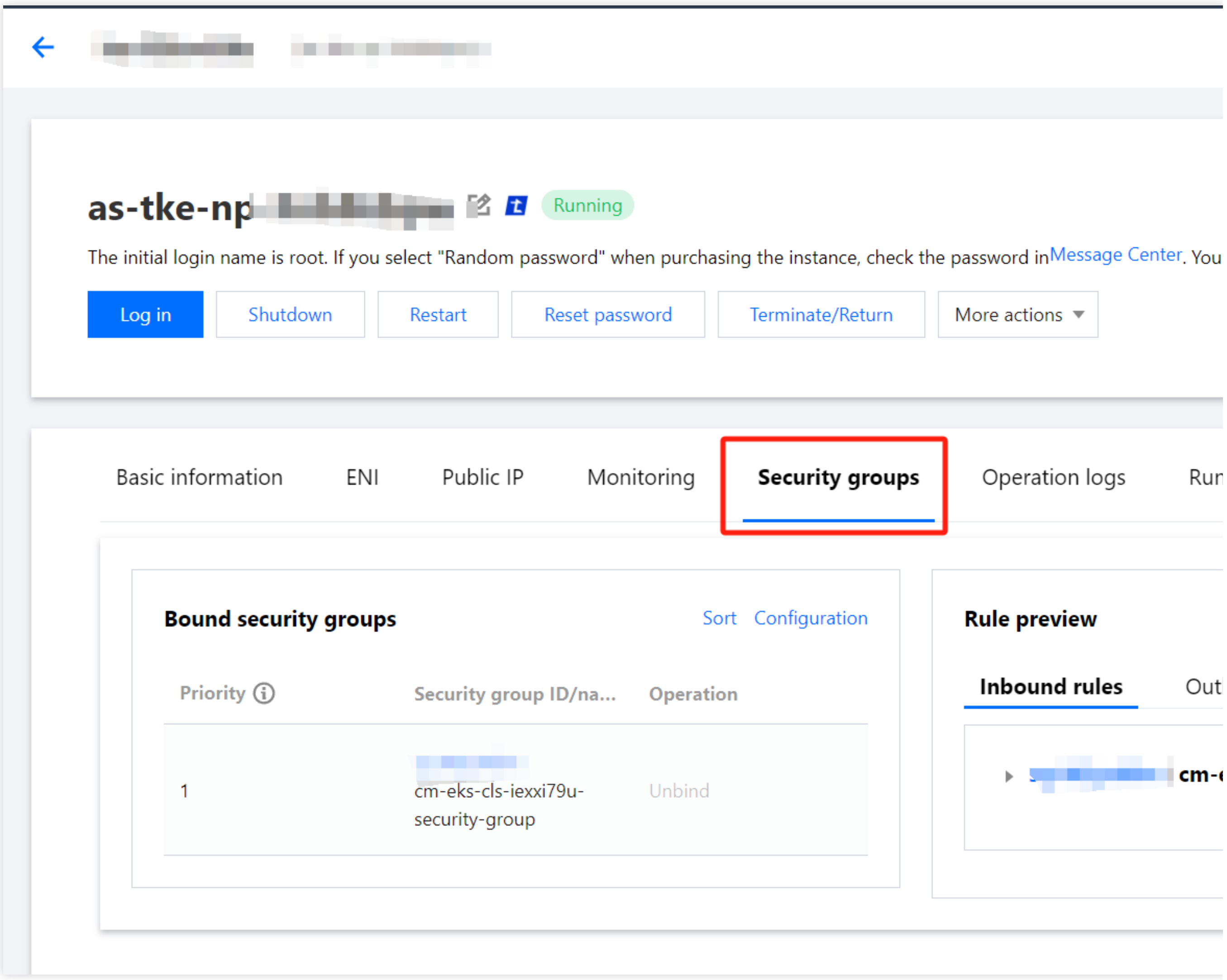
Operating systemⓘ	TencentOS Server 3.2 (Final) Public image -Basic image	Runtime components	containerd 1.
Modelⓘ	SA2.MEDIUM2(Primary)	Subnet	[redacted]
Data disk	-	Custom dataⓘ	View
Custom Kubelet parameters	View	Placement group	

Adjust quantity   Add existing node   Remove   More ▾   Separate filte

Node ID/name	S..	Availabili...	Configuration	Removal ...	IP address	How to...	Res
View details [redacted]6s as-tke-n[redacted]w	Healthy	Guangzho...	SA2.MEDIUM2 2 core, 2 GB, - Mbps System disk: 50 GB	Disabled	- [redacted]28	Scaling ...	CPI cor Me 1.0

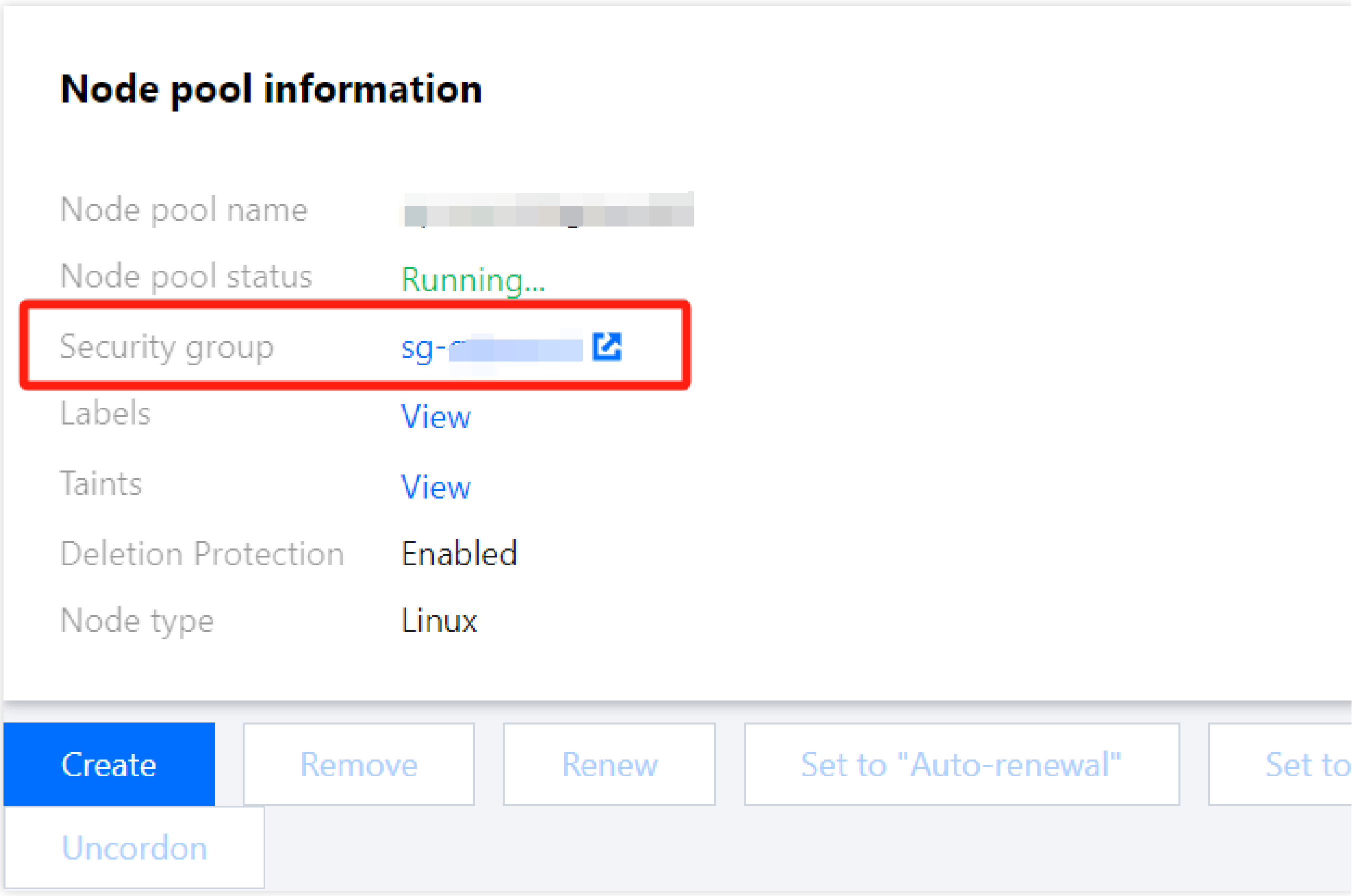
Total items: 1

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:



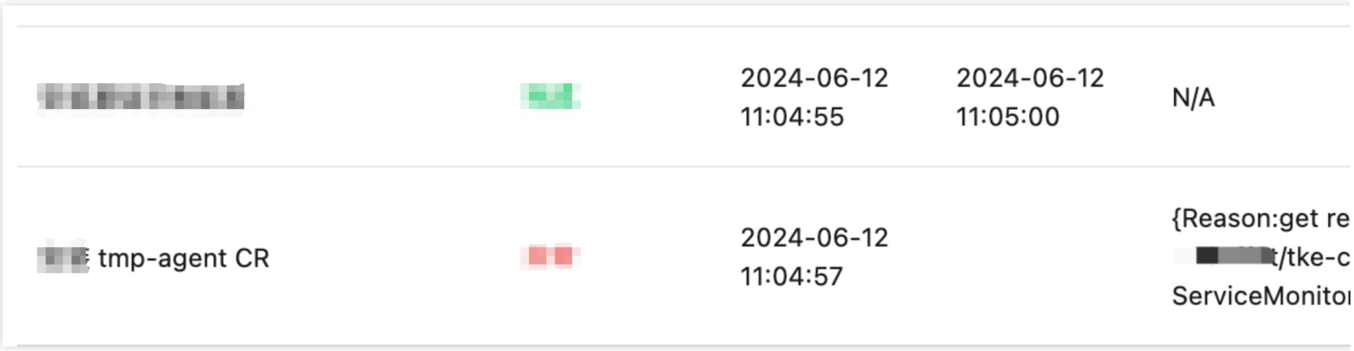
Node pool information

Node pool name	[redacted]
Node pool status	Running...
Security group	sg-[redacted]
Labels	View
Taints	View
Deletion Protection	Enabled
Node type	Linux

Related Issues

Issue Description

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

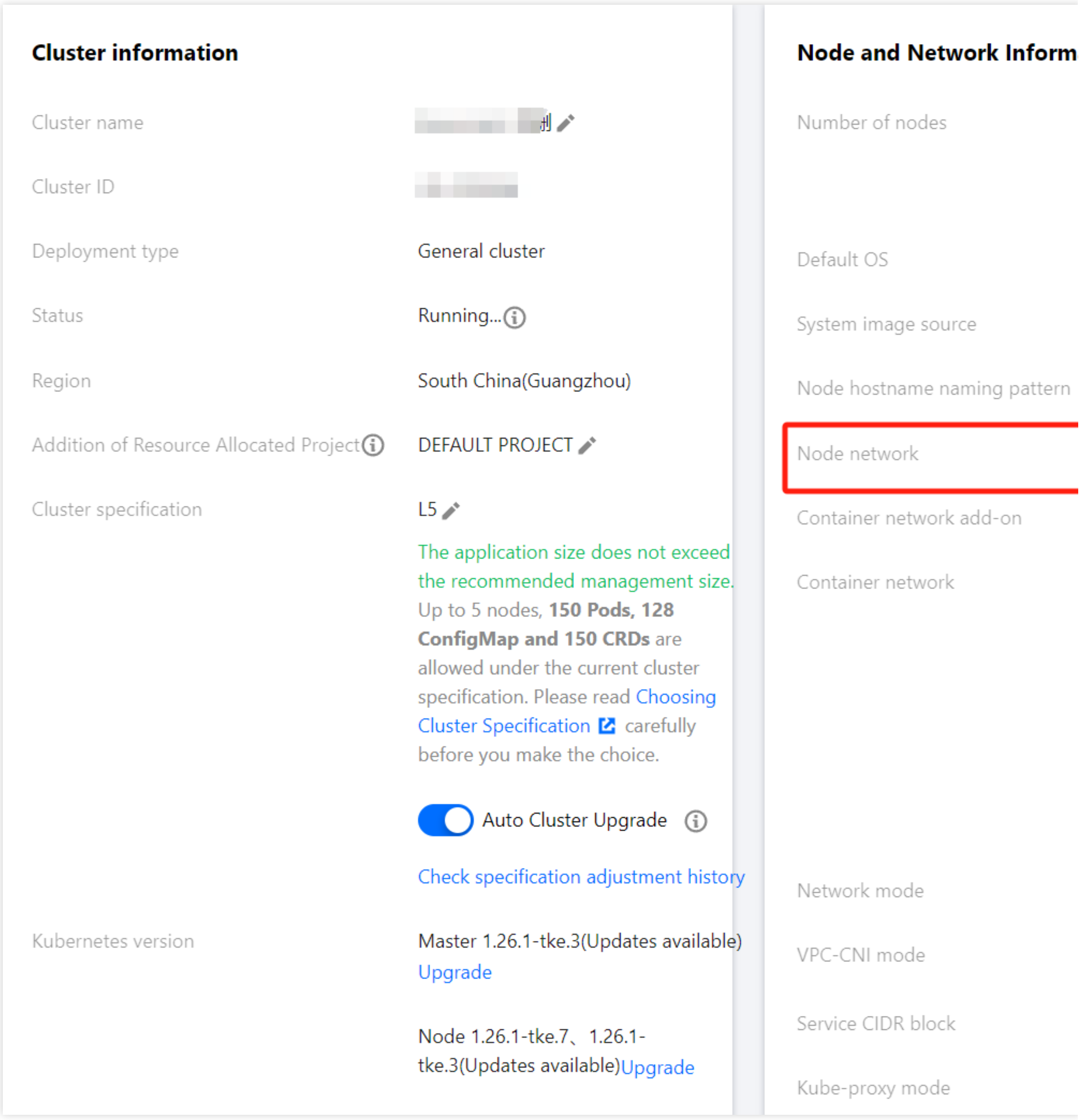


[redacted]	[redacted]	2024-06-12 11:04:55	2024-06-12 11:05:00	N/A
[redacted] tmp-agent CR	[redacted]	2024-06-12 11:04:57		{Reason:get re [redacted]t/tke-c ServiceMonitoı

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):



Cluster information

Cluster name	[redacted]
Cluster ID	[redacted]
Deployment type	General cluster
Status	Running...
Region	South China(Guangzhou)
Addition of Resource Allocated Project	DEFAULT PROJECT
Cluster specification	L5 The application size does not exceed the recommended management size. Up to 5 nodes, **150 Pods, 128 ConfigMap and 150 CRDs** are allowed under the current cluster specification. Please read Choosing Cluster Specification carefully before you make the choice. Auto Cluster Upgrade Check specification adjustment history
Kubernetes version	Master 1.26.1-tke.3(Updates available) Upgrade Node 1.26.1-tke.7、1.26.1-tke.3(Updates available)Upgrade

Node and Network Inform

Number of nodes

Default OS

System image source

Node hostname naming pattern

Node network

Container network add-on

Container network

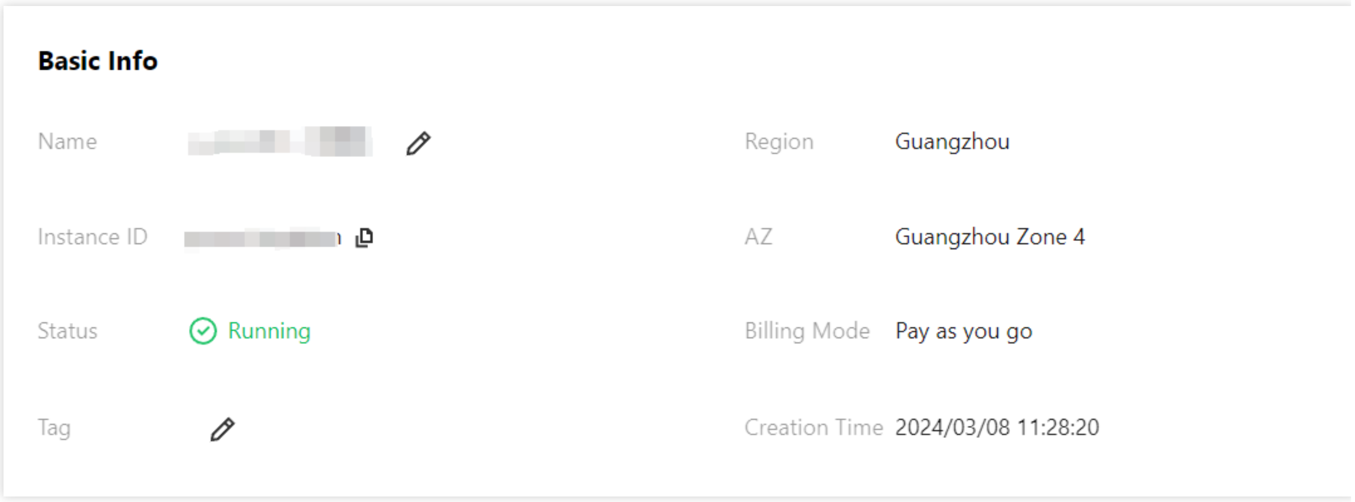
Network mode

VPC-CNI mode

Service CIDR block

Kube-proxy mode

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

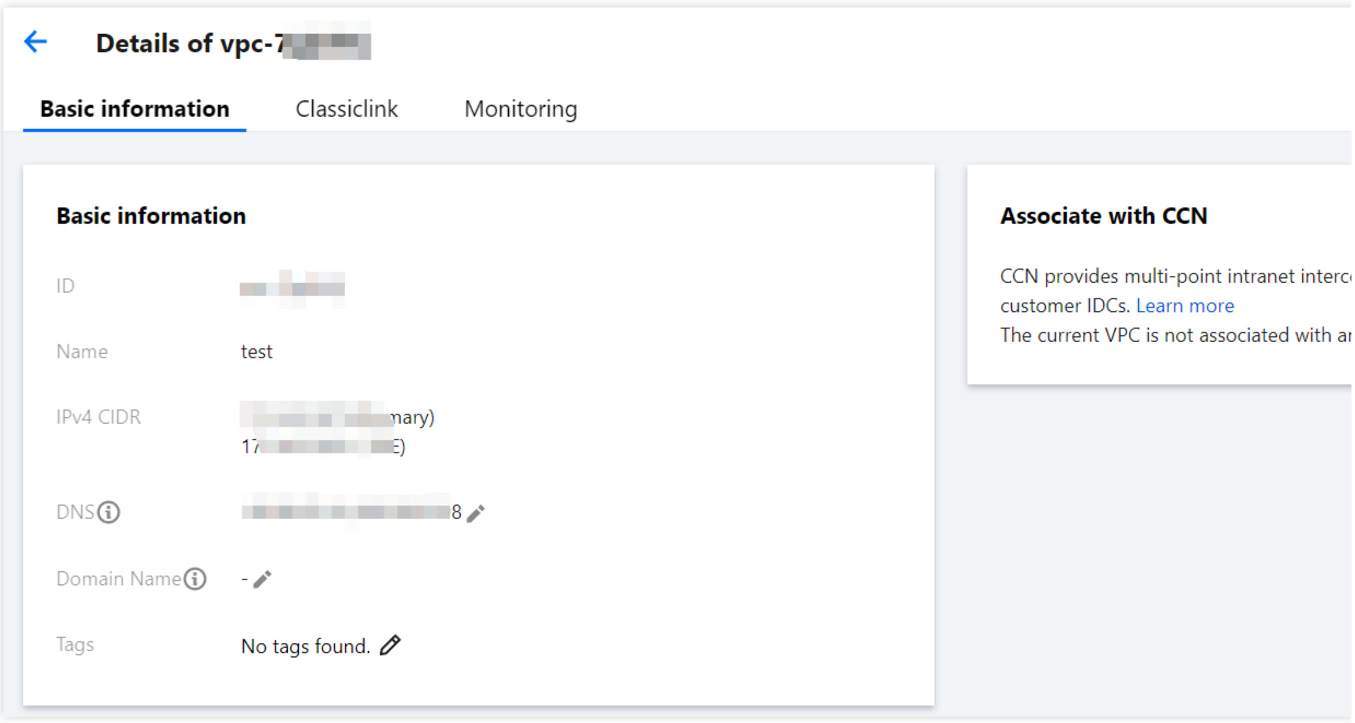


Basic Info

Name		Region	Guangzhou
Instance ID		AZ	Guangzhou Zone 4
Status	Running	Billing Mode	Pay as you go
Tag		Creation Time	2024/03/08 11:28:20

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:



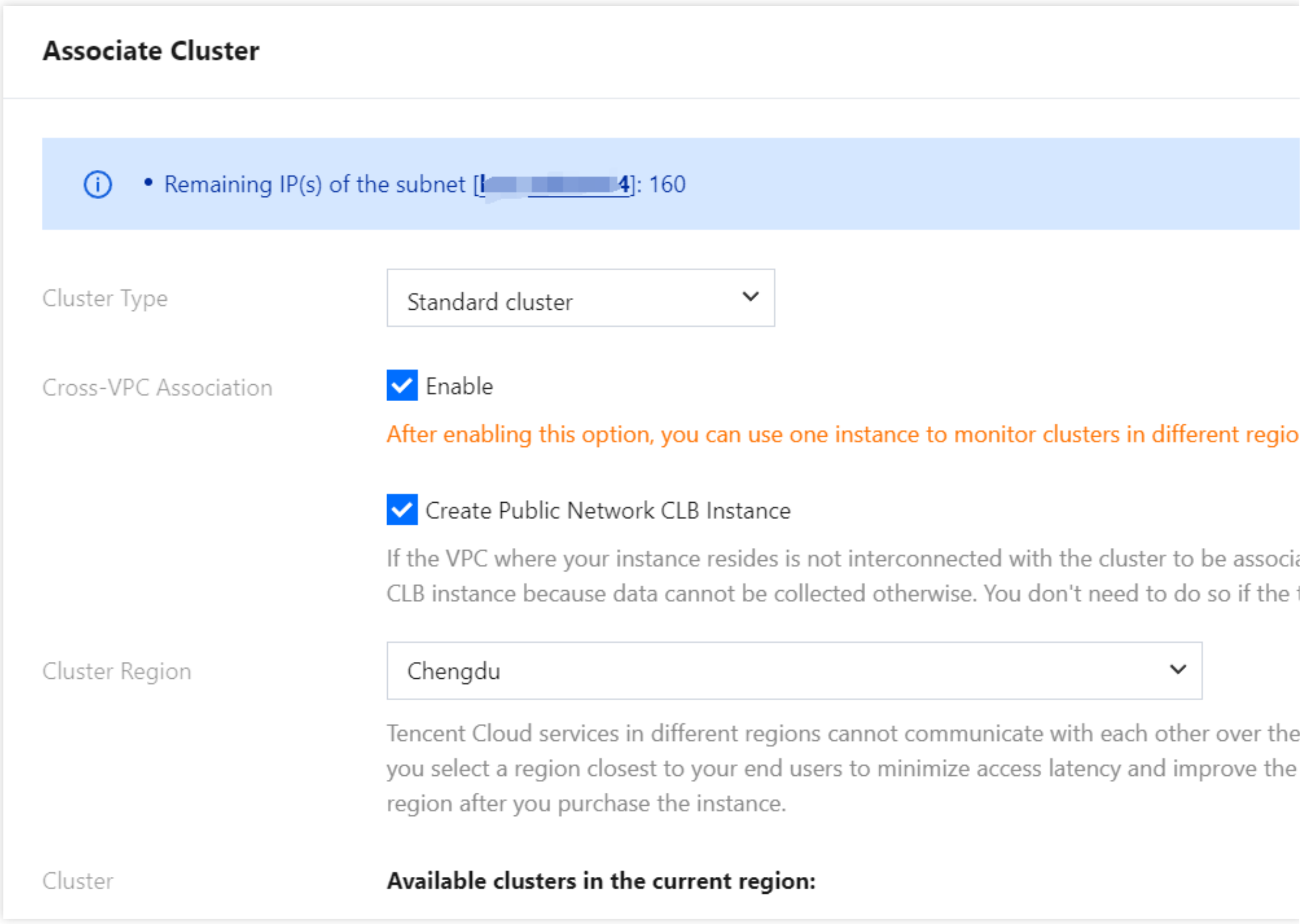
Basic information

ID	
Name	test
IPv4 CIDR	mary) 17 E)
DNS	8
Domain Name	-
Tags	No tags found.

Associate with CCN

CCN provides multi-point intranet interc
customer IDCs. Learn more
The current VPC is not associated with ar

Select Create Public Network CLB Instance:



Associate Cluster

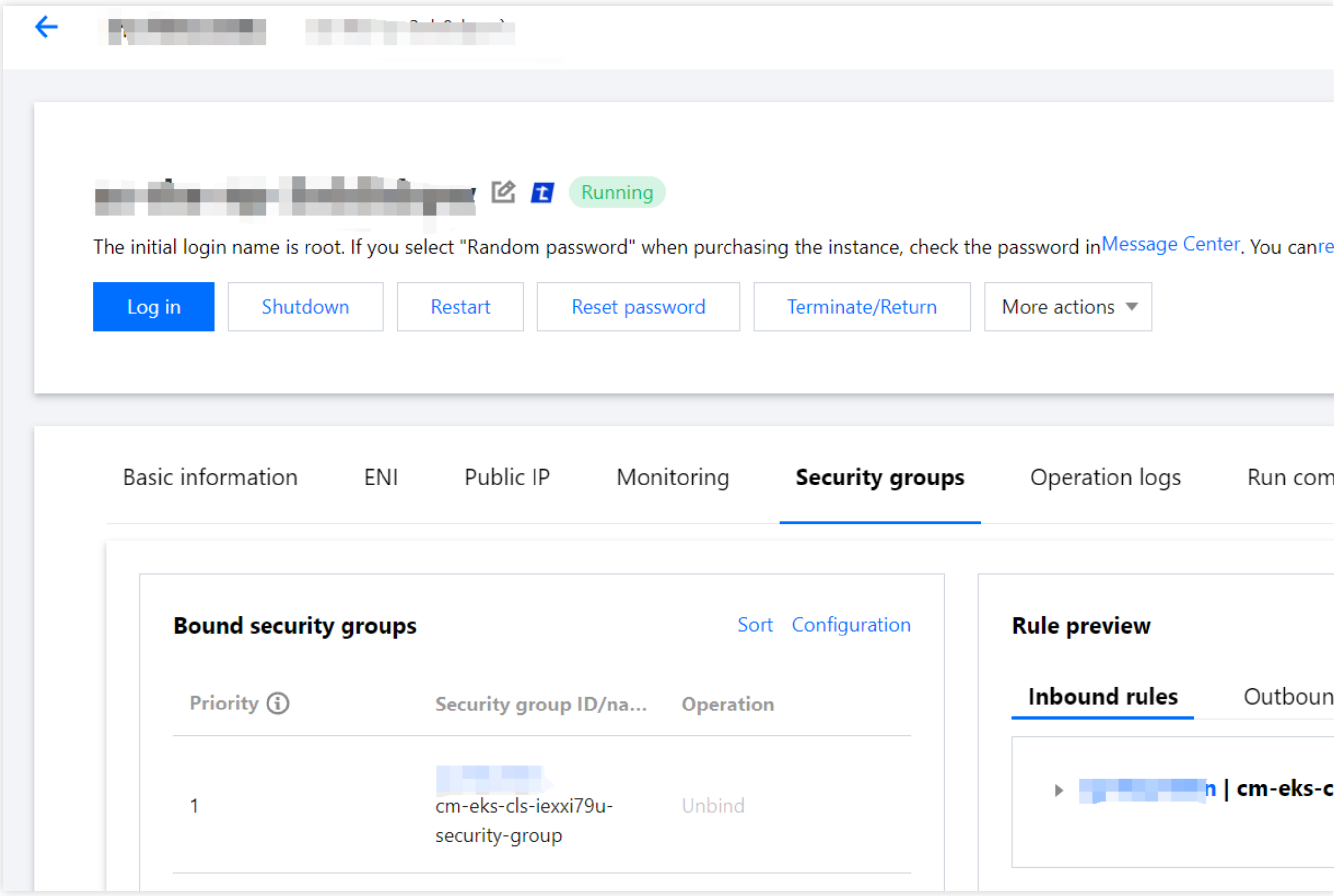
• Remaining IP(s) of the subnet [[redacted]4]: 160

Cluster Type	Standard cluster
Cross-VPC Association	✓ Enable After enabling this option, you can use one instance to monitor clusters in different regio ✓ Create Public Network CLB Instance If the VPC where your instance resides is not interconnected with the cluster to be associ CLB instance because data cannot be collected otherwise. You don't need to do so if the t
Cluster Region	Chengdu Tencent Cloud services in different regions cannot communicate with each other over the you select a region closest to your end users to minimize access latency and improve the region after you purchase the instance.
Cluster	**Available clusters in the current region:**

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:



The initial login name is root. If you select "Random password" when purchasing the instance, check the password inMessage Center. You canre

Bound security groups

Priority	Security group ID/na...	Operation
1	[redacted] cm-eks-cls-iexxi79u-security-group	Unbind

[redacted] | cm-eks-c

Check if the security group rules meet the requirements.