

Application Operations Management

Best Practices

Issue 01
Date 2025-01-07



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: <https://www.huaweicloud.com/intl/en-us/>

Contents

1 Overview.....	1
2 Alarm Noise Reduction.....	2
3 Customizing OS Images to Automatically Connect UniAgent.....	7
4 Connecting Self-Built Middleware in the CCE Container Scenario.....	9
4.1 Connecting PostgreSQL Exporter.....	9
4.2 Connecting MySQL Exporter.....	14
4.3 Connecting Kafka Exporter.....	18
4.4 Connecting Memcached Exporter.....	22
4.5 Connecting MongoDB Exporter.....	26
4.6 Connecting Elasticsearch Exporter.....	30
4.7 Connecting Redis Exporter.....	34
4.8 Connecting Other Exporters.....	39

1 Overview

This chapter lists the best practices about Application Operations Management (AOM).

- [2 Alarm Noise Reduction](#)
- [3 Customizing OS Images to Automatically Connect UniAgent](#)
- [4 Connecting Self-Built Middleware in the CCE Container Scenario](#)
 - [4.1 Connecting PostgreSQL Exporter](#)
 - [4.2 Connecting MySQL Exporter](#)
 - [4.3 Connecting Kafka Exporter](#)
 - [4.4 Connecting Memcached Exporter](#)
 - [4.5 Connecting MongoDB Exporter](#)
 - [4.6 Connecting Elasticsearch Exporter](#)
 - [4.7 Connecting Redis Exporter](#)
 - [4.8 Connecting Other Exporters](#)

2 Alarm Noise Reduction

This section describes how to set alarm noise reduction. Before sending an alarm notification, AOM processes alarms based on noise reduction rules to prevent alarm storms.

Scenario

When analyzing applications, resources, and businesses, e-commerce O&M personnel find that the number of alarms is too large and there are too many identical alarms. They need to detect faults based on the alarms and monitor applications comprehensively.

Solution

Use AOM to set alarm rules to monitor the usage of resources (such as hosts and components) in the environment in real time. When AOM or an external service is abnormal, an alarm is triggered immediately. AOM also provides the alarm noise reduction function. Before sending an alarm notification, AOM processes alarms based on noise reduction rules. This helps you identify critical problems and avoid alarm storms.

Alarm noise reduction consists of four parts: grouping, deduplication, suppression, and silence.

- You can filter different subnets of alarms and then group them according to certain conditions. Alarms in the same group are aggregated to trigger one notification.
- By using suppression rules, you can suppress or block notifications related to specific alarms. For example, when a major alarm is generated, less severe alarms can be suppressed. Another example, when a node is faulty, all other alarms of the processes or containers on this node can be suppressed.
- You can create a silence rule to shield alarm notifications in a specified period. The rule takes effect immediately after it is created.
- AOM has built-in deduplication rules. The service backend automatically deduplicates alarms. You do not need to manually create rules.

Monitoring ELB metrics at the business layer is used as an example here.

Prerequisite

An alarm action rule has been created.

Step 1: Create a Grouping Rule

When a critical or major alarm is generated, the **Monitor_host** action rule is triggered, and alarms are grouped by alarm source. To create a grouping rule, do as follows:

- Step 1** Log in to the AOM 2.0 console.
- Step 2** In the navigation pane, choose **Alarm Management > Alarm Noise Reduction**.
- Step 3** On the **Grouping Rules** tab page, click **Create** and set the rule name and grouping condition.

Figure 2-1 Creating a grouping rule

* Rule Name

* Enterprise Project

Description

Grouping Rule

Grouping Condition

Alarm Severity	event_severity	Equals To	Critic... x Ma... x	<input type="button" value="x"/>
Alarm Source	resource_provider	Equals To	A... x	<input type="button" value="x"/>

Action Rule

Combination Rule

* Combine Notifications

* Initial Wait Time Range: 0s to 10 mins.

* Batch Processing Interval Range: 5s to 30 mins.

* Repeat Interval Range: 1 min to 15 days.

Note: If Repeat Interval is set to 0, identical notifications will not be sent again.

Table 2-1 Alarm combination rule

Combine Notifications	<p>Combines grouped alarms based on specified fields. Alarms in the same group are aggregated for sending one notification.</p> <p>Notifications can be combined:</p> <ul style="list-style-type: none"> • By alarm source: Alarms triggered by the same alarm source are combined into one group for sending notifications. • By alarm source + severity: Alarms triggered by the same alarm source and of the same severity are combined into one group for sending notifications. • By alarm source + all tags: Alarms triggered by the same alarm source and with the same tag are combined into one group for sending notifications.
Initial Wait Time	<p>Interval for sending an alarm notification after alarms are combined for the first time. It is recommended that the time be set to seconds to prevent alarm storms.</p> <p>Value range: 0s to 10 minutes. Recommended: 15s.</p>
Batch Processing Interval	<p>Waiting time for sending an alarm notification after the combined alarm data changes. It is recommended that the time be set to minutes. If you want to receive alarm notifications as soon as possible, set the time to seconds.</p> <p>The change here refers to a new alarm or an alarm status change.</p> <p>Value range: 5s to 30 minutes. Recommended: 60s.</p>
Repeat Interval	<p>Waiting time for sending an alarm notification after the combined alarm data becomes duplicate. It is recommended that the time be set to hours.</p> <p>Duplication means that no new alarm is generated and no alarm status is changed while other attributes (such as titles and content) are changed.</p> <p>Value range: 0 minutes to 15 days. Recommended: 1 hour.</p>

----End

Step 2: Create a Metric Alarm Rule (Configuration Mode Set to Select from all metrics)

You can set threshold conditions in metric alarm rules for resource metrics. If a metric value meets the threshold condition, a threshold alarm will be generated. If no metric data is reported, an insufficient data event will be generated.

Metric alarm rules can be created in the following modes: **Select from all metrics** and **PromQL**. The following describes how to create an alarm rule for monitoring all metrics at the ELB business layer.

Step 1 Log in to the AOM 2.0 console.

Step 2 In the navigation pane, choose **Alarm Management > Alarm Rules**.

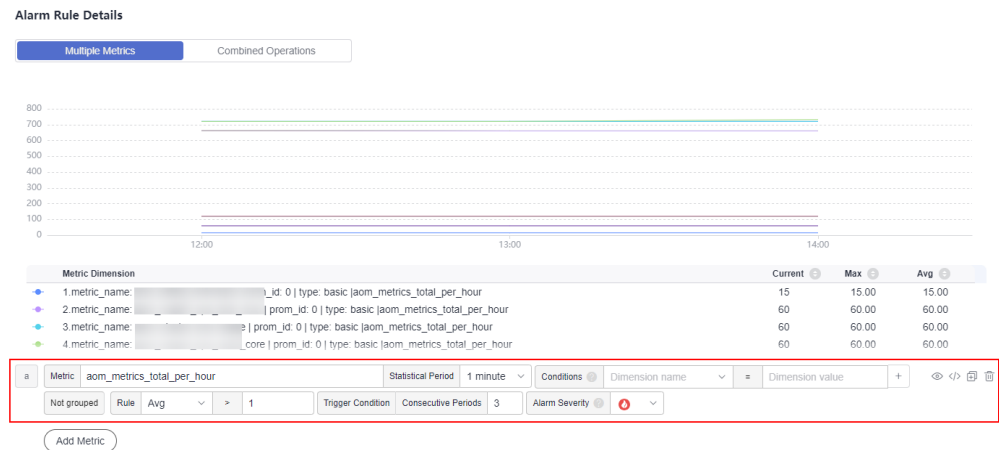
Step 3 On the **Metric/Event Alarm Rules** tab page, click **Create**.

Step 4 Set basic information about the alarm rule, such as the rule name.

Step 5 Set detailed information about the alarm rule.

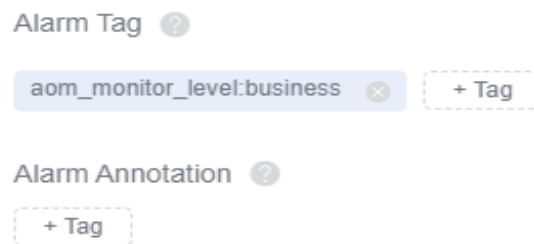
1. Set **Rule Type** to **Metric alarm rule** and **Configuration Mode** to **Select from all metrics**.
2. Set parameters such as the metric, environment, and check interval.

Figure 2-2 Setting the detailed information about the alarm rule



3. Set alarm tags and annotations to group alarms. They can be associated with alarm noise reduction policies for sending notifications. As a business-layer metric is selected in [Step 5.2](#), set **Alarm Tag** to **aom_monitor_level:business**.

Figure 2-3 Customizing tag information



NOTE

The tag of full metrics is in the format of "key:value". Generally, **key** is set to **aom_monitor_level**. **value** varies depending on the layer of metrics:

- Infrastructure metrics: **infrastructure**
- Middleware metrics: **middleware**
- Application metrics: **application**
- Business metrics: **business**

Step 6 Set an alarm notification policy. There are two alarm notification modes. In this example, the alarm noise reduction mode is selected.

Alarm noise reduction: Alarms are sent only after being processed based on noise reduction rules, preventing alarm storms.

Figure 2-4 Selecting the alarm noise reduction mode

Alarm Notification

Notify When

Alarm triggered Alarm cleared

Alarm Mode

Direct alarm reporting **Alarm noise reduction**

Grouping Rule

aom1 ↕ ↻ 📄

Step 7 Click **Confirm**. Then, click **Back to Alarm Rule List** to view the created alarm rule.

As shown in the following figure, a metric alarm rule is created. Click **↕** in front of the rule name to view its details.

Figure 2-5 Creating a metric alarm rule

<input type="checkbox"/>	Rule Name/Type	Rule Status	Monitored Object	Alarm Condition	Action Rule	Bound Prometheus L...	Status	Operation
<input checked="" type="checkbox"/>	Metric alarm	Enabled	--	Monitored Object. For 3 consecutive ...	--	Prometheus_AO...	<input checked="" type="checkbox"/>	↕ 📄 🗑️

Basic Info		Monitored Object	Alarm Condition	Alarms
Alarm Condition		Monitored Object. For 3 consecutive periods Avg>1		Alarm Severity 🔴
Check Interval	Custom interval, every 1 minute			
Alarm Clearance	If the monitored object does not meet the trigger condition for 1 monitoring period, the alarm will be automatically cleared.			
Action Taken for Insufficient Data	N/A			

In the expanded list, if a metric value meets the configured alarm condition, a metric alarm is generated on the alarm page. To view the alarm, choose **Alarm Management > Alarm List** in the navigation pane.

If the preset notification policy is met, the system sends an alarm notification to the specified personnel by email, SMS, or WeCom.

----End

3 Customizing OS Images to Automatically Connect UniAgent

This section describes how to package images for connecting UniAgent in the Linux and Windows environments.

Overview

An image is an Elastic Compute Server (ECS) or Bare Metal Server (BMS) template that contains OS or service data and may also contain proprietary software and application software, such as database software. Images are classified into public, private, Marketplace, and shared images.

Image Management Service (IMS) provides easy-to-use, self-service image management functions. You can use a public, private, or shared image to apply for ECSs. You can also create private images from existing ECSs or using external image files.

Packaging an Image in the Linux Environment

In the Linux environment, you can package an image according to the following procedure:

Prerequisites

Ensure that no UniAgent has been installed on the Linux host where the image is to be packaged.

Procedure

Step 1 Create an ECS by referring to [ECS Getting Started](#).

Step 2 For example, in the **CN North-Beijing4** region, download the **install_uniagentd_self_OS.sh** script to the **/root** directory of the ECS:

```
wget https://aom-uniagent-cn-north-4.obs.cn-north-4.myhuaweicloud.com/install_uniagentd_self_OS.sh
      {region_id}=cn-north-4
      {obs_domain}=obs.cn-north-4.myhuaweicloud.com
```

NOTE

Download command: **wget https://aom-uniagent-{region_id}.{obs_domain}/install_uniagentd_self_OS.sh**

Step 3 In the `/etc/init.d/` directory, set the `install_uniagentd_self_OS.sh` script to automatically start upon power-on:

```
bash /root/install_uniagentd_self_OS.sh config
```

If the **AOMInstall** startup script exists in the `/etc/init.d/` directory, your setting is successful.

Step 4 Delete the configuration script:

```
rm -f /root/install_uniagentd_self_OS.sh
```

 **NOTE**

After the preceding steps are complete, you can create an image. Do not restart the Linux host before you create an image.

Step 5 Locate the target ECS and click **Create Image** in the **Operation** column to create a private image. For details, see [Creating an Image](#).

Step 6 Configure image information as required.

----End

Packaging an Image in the Windows Environment

In the Windows environment, you can only install the UniAgent, delete some files, and then package your private image.

Step 1 Create an ECS by referring to [ECS Getting Started](#).

Step 2 On the ECS, manually install the UniAgent by referring to [Installing a UniAgent](#). Then check the UniAgent status on the UI.

Step 3 Run the following command on the ECS after the UniAgent is installed:

```
sc stop uniagentdservice
&& del /s/q C:\uniagentd\uniagentd.sn && rd /s/q C:\uniagentd\tmp C:\uniagentd\log C:\uniagentd\libexec
&& echo -e "${ak_info}\n${sk_info}\n${master_info}" > C:\uniagentd\conf\uniagentd.conf
```

 **NOTE**

Obtain the values of `${ak_info}`, `${sk_info}`, and `${master_info}` from the manual installation page and replace them based on site requirements. Each AK/SK pair corresponds to a specific project.

Step 4 Locate the target ECS and click **Create Image** in the **Operation** column to create a private image. For details, see [Creating an Image](#).

Step 5 Configure image information as required.

----End

4 Connecting Self-Built Middleware in the CCE Container Scenario

4.1 Connecting PostgreSQL Exporter

Application Scenario

When using PostgreSQL, you need to monitor their status and locate their faults in a timely manner. The Prometheus monitoring function monitors PostgreSQL running using Exporter in the CCE container scenario. This section describes how to deploy PostgreSQL Exporter and implement alarm access.

Prerequisites

- A CCE cluster has been created and PostgreSQL has been installed.
- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the [postgres_exporter](#) image to SoftWare Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Deploying PostgreSQL Exporter

Step 1 Log in to the CCE console.

Step 2 Click the connected cluster. The cluster management page is displayed.

Step 3 Perform the following operations to deploy Exporter:

1. Use **Secret** to manage PostgreSQL passwords.

In the navigation pane, choose **Workloads**. In the upper right corner, click **Create from YAML** to configure a YAML file. In the YAML file, use Kubernetes **Secret** to manage and encrypt passwords. When starting PostgreSQL Exporter, the secret key can be directly used but the corresponding password needs to be changed as required.

YAML configuration example:

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: postgres-test
type: Opaque
stringData:
  username: postgres
  password: you-guess # PostgreSQL password.
```

2. Deploy PostgreSQL Exporter.

In the navigation pane, choose **Workloads**. In the upper right corner, click **Create from YAML** to deploy Exporter.

YAML configuration example (Change the parameters if needed):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test # Change the name based on requirements. You are advised to add the
  PostgreSQL instance information.
  namespace: default # Must be the same as the namespace of the PostgreSQL service.
  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: swr.cn-north-4.myhuaweicloud.com/aom-exporter/postgres-exporter:v0.8.0 # postgres-
          exporter image uploaded to SWR.
          args:
            - "--web.listen-address=:9187" # Enabled port of Exporter.
            - "--log.level=debug" # Log level.
          env:
            - name: DATA_SOURCE_USER
              valueFrom:
                secretKeyRef:
                  name: postgres-test # Secret name specified in the previous step.
                  key: username # Secret key specified in the previous step.
            - name: DATA_SOURCE_PASS
              valueFrom:
                secretKeyRef:
                  name: postgres-test # Secret name specified in the previous step.
                  key: password # Secret key specified in the previous step.
            - name: DATA_SOURCE_URI
              value: "x.x.x.x:5432/postgres?sslmode=disable" # Connection information.
          ports:
            - name: http-metrics
              containerPort: 9187
```

3. Obtain metrics.

The running time of the Postgres instance cannot be obtained by running the **curl http://exporter:9187/metrics** command. To obtain this metric, customize a **queries.yaml** file.

- a. Create a configuration that contains **queries.yaml**.
- b. Mount the configuration as a volume to a directory of Exporter.
- c. Use the configuration through **extend.query-path**. The following shows **Secret** and **Deployment**:

```
# The following shows the queries.yaml file that contains custom metrics:
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-test-configmap
  namespace: default
data:
  queries.yaml: |
    pg_postmaster:
      query: "SELECT pg_postmaster_start_time as start_time_seconds from
pg_postmaster_start_time()"
      master: true
      metrics:
        - start_time_seconds:
            usage: "GAUGE"
            description: "Time at which postmaster started"

# The following shows the mounted Secret and ConfigMap, and defines Exporter deployment
parameters (such as the image):
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-test
  namespace: default
  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
```

```

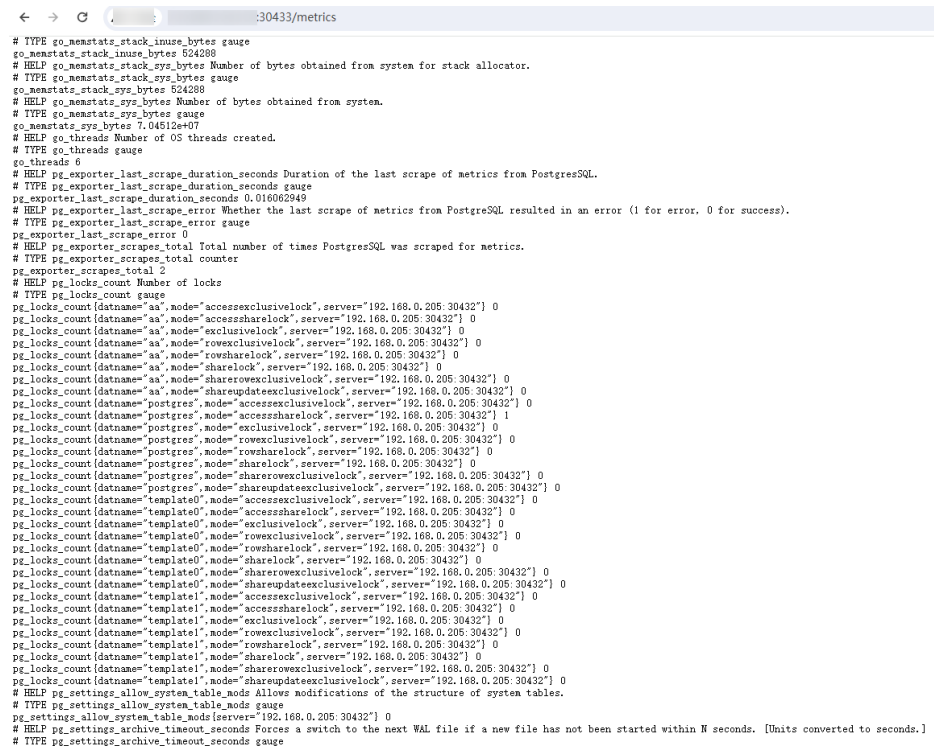
configMap:
  name: postgres-test-configmap
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  type: NodePort
  selector:
    app: postgres
  app.kubernetes.io/name: postgresql
  ports:
    - protocol: TCP
      nodePort: 30433
      port: 9187
      targetPort: 9187

```

d. Access the following address:

<http://{Public IP address of any node in the cluster}:30433/metrics>. You can then use the custom **queries.yaml** file to query the Postgres instance startup time.

Figure 4-1 Accessing a cluster node



```

← → ↻ :30433/metrics
# TYPE go_memstats_stack_inuse_bytes gauge
go_memstats_stack_inuse_bytes 624288
# HELP go_memstats_stack_sys_bytes Number of bytes obtained from system for stack allocator.
# TYPE go_memstats_stack_sys_bytes gauge
go_memstats_stack_sys_bytes 824288
# HELP go_memstats_sys_bytes Number of bytes obtained from system.
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 7.04512e+07
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 6
# HELP pg_exporter_last_scrape_duration_seconds Duration of the last scrape of metrics from PostgreSQL.
# TYPE pg_exporter_last_scrape_duration_seconds gauge
pg_exporter_last_scrape_duration_seconds 0.016062949
# HELP pg_exporter_last_scrape_error Whether the last scrape of metrics from PostgreSQL resulted in an error (1 for error, 0 for success).
# TYPE pg_exporter_last_scrape_error gauge
pg_exporter_last_scrape_error 0
# HELP pg_exporter_scrapes_total Total number of times PostgreSQL was scraped for metrics.
# TYPE pg_exporter_scrapes_total counter
pg_exporter_scrapes_total 2
# HELP pg_locks_count Number of locks
# TYPE pg_locks_count gauge
pg_locks_count{datname="aa",mode="accessexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="accessshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="exclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="rowexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="rowshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="share",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="shareexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="aa",mode="shareupdateexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="accessexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="accessshare",server="192.168.0.205:30432"} 1
pg_locks_count{datname="postgres",mode="exclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="rowexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="rowshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="share",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="shareexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="postgres",mode="shareupdateexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="accessexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="accessshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="exclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="rowexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="rowshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="share",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="shareexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template0",mode="shareupdateexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="accessexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="accessshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="exclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="rowexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="rowshare",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="share",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="shareexclusive",server="192.168.0.205:30432"} 0
pg_locks_count{datname="template1",mode="shareupdateexclusive",server="192.168.0.205:30432"} 0
# HELP pg_settings_allow_system_table_mods Allows modifications of the structure of system tables.
# TYPE pg_settings_allow_system_table_mods gauge
pg_settings_allow_system_table_mods{server="192.168.0.205:30432"} 0
# HELP pg_settings_archive_timeout_seconds Forces a switch to the next WAL file if a new file has not been started within N seconds. [Units converted to seconds.]
# TYPE pg_settings_archive_timeout_seconds gauge

```

----End

Adding a Collection Task

Add PodMonitor to configure a collection rule for monitoring the service data of applications deployed in the CCE cluster.

NOTE

In the following example, metrics are collected every 30s. Therefore, you can check the reported metrics on the AOM page about 30s later.

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: postgres-exporter
  namespace: default
spec:
  namespaceSelector:
    matchNames:
      - default # Namespace where Exporter is located.
  podMetricsEndpoints:
    - interval: 30s
      path: /metrics
      port: http-metrics
  selector:
    matchLabels:
      app: postgres
```

Verifying that Metrics Can Be Reported to AOM

- Step 1** Log in to the AOM 2.0 console.
- Step 2** In the navigation pane on the left, choose **Prometheus Monitoring > Instances**.
- Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
- Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
- Step 5** Select job `{namespace}/postgres-exporter` to query metrics starting with **pg**.
----End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.2 Connecting MySQL Exporter

Application Scenario

MySQL Exporter collects MySQL database metrics. Core database metrics collected through Exporter are used for alarm reporting and dashboard display. Currently, Exporter supports MySQL 5.6 or later. If the MySQL version is earlier than 5.6, some metrics may fail to be collected.

NOTE

You are advised to use CCE for unified Exporter management.

Prerequisites

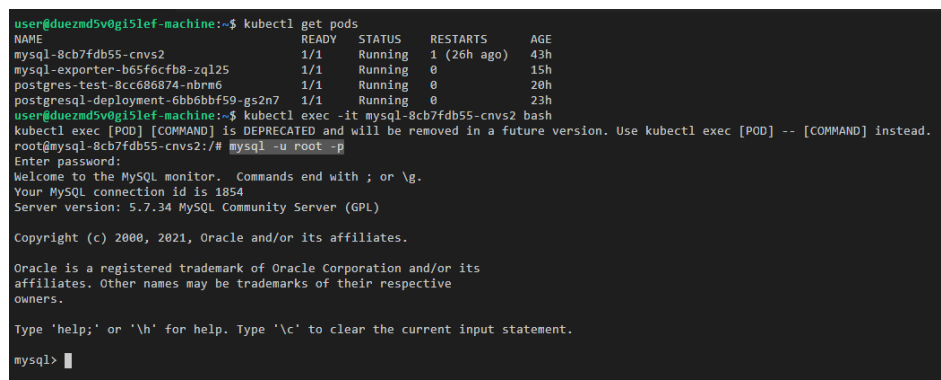
- A CCE cluster has been created and MySQL has been installed.
- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the [mysql_exporter](#) image to SoftWare Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Database Authorization

Step 1 Log in to the cluster and run the following command:

```
kubectl exec -it ${mysql_podname} bash
mysql -u root -p
```

Figure 4-2 Executing the command



```
user@dueznd5v0gi51ef-machine:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mysql-8cb7fdb55-cnvs2               1/1     Running  1 (26h ago) 43h
mysql-exporter-b65f6cfb8-zql25      1/1     Running  0           15h
postgres-test-8cc686874-nbrm6      1/1     Running  0           20h
postgresl-deployment-6bb6bbf59-gs2n7 1/1     Running  0           23h
user@dueznd5v0gi51ef-machine:~$ kubectl exec -it mysql-8cb7fdb55-cnvs2 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@mysql-8cb7fdb55-cnvs2:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1854
Server version: 5.7.34 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Step 2 Log in to the database and run the following command:

```
CREATE USER 'exporter'@'x.x.x.x(hostip)' IDENTIFIED BY 'xxxx(password)' WITH MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter'@'x.x.x.x(hostip)';
```

Step 3 Check whether the authorization is successful.

Enter the following SQL statement to check whether there is any Exporter data. *host* indicates the IP address of the node where the MySQL database is located.

```
select user,host from mysql.user;
```

Figure 4-3 SQL statement

```
mysql> select user,host from mysql.user;
+-----+-----+
| user      | host      |
+-----+-----+
| root      | %         |
| exporter  | 192.168.0.205 |
| mysql.session | localhost |
| mysql.sys | localhost |
| root      | localhost |
+-----+-----+
5 rows in set (0.00 sec)

mysql> █
```

----End

Deploying MySQL Exporter

- Step 1** Log in to the CCE console.
- Step 2** Click the connected cluster. The cluster management page is displayed.
- Step 3** Perform the following operations to deploy Exporter:

1. Use **Secret** to manage MySQL connection strings.

In the navigation pane, choose **ConfigMaps and Secrets**. In the upper right corner, click **Create from YAML** and enter the following **.yaml** file. The password is encrypted based on Opaque requirements.

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: default
type: Opaque
stringData:
  datasource: "user:password@tcp(ip:port)/" # MySQL connection string, which needs to be encrypted.
```

NOTE

For details about how to configure a secret, see [Creating a Secret](#).

2. Deploy MySQL Exporter.

In the navigation pane, choose **Workloads**. In the upper right corner, click **Create Workload**. Then select the **Deployment** workload and select a desired namespace to deploy MySQL Exporter. YAML configuration example for deploying Exporter:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mysql-exporter # Change the name based on service requirements. You are advised to add the MySQL instance information, for example, ckafka-2vrgx9fd-mysql-exporter.
  name: mysql-exporter # Change the name based on service requirements. You are advised to add the MySQL instance information, for example, ckafka-2vrgx9fd-mysql-exporter.
  namespace: default # Must be the same as the namespace of MySQL.
spec:
```

```
replicas: 1
selector:
  matchLabels:
    k8s-app: mysql-exporter # Change the name based on service requirements. You are advised to
add the MySQL instance information, for example, ckafka-2vrgx9fd-mysql-exporter.
template:
  metadata:
    labels:
      k8s-app: mysql-exporter # Change the name based on service requirements. You are advised to
add the MySQL instance information, for example, ckafka-2vrgx9fd-mysql-exporter.
spec:
  containers:
  - env:
    - name: DATA_SOURCE_NAME
      valueFrom:
        secretKeyRef:
          name: mysql-secret
          key: datasource
      image: swr.cn-north-4.myhuaweicloud.com/aom-exporter/mysql-d-exporter:v0.12.1
      imagePullPolicy: IfNotPresent
      name: mysql-exporter
      ports:
      - containerPort: 9104
        name: metric-port
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: default-secret
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-exporter
spec:
  type: NodePort
  selector:
    k8s-app: mysql-exporter
  ports:
  - protocol: TCP
    nodePort: 30337
    port: 9104
    targetPort: 9104
```

NOTE

For details about Exporter parameters, see [mysql-exporter](#).

3. Check whether MySQL Exporter is successfully deployed.
 - a. On the **Deployments** tab page, click the Deployment created in [Step 3.2](#). In the pod list, choose **More > View Logs** in the **Operation** column. The Exporter is successfully started and its access address is exposed.
 - b. Perform verification using one of the following methods:
 - Log in to a cluster node and run either of the following commands:
`curl http://{Cluster IP address}:9104/metrics`
`curl http://{Private IP address of any node in the cluster}:30337/metrics`
 - In the instance list, choose **More > Remote Login** in the **Operation** column and run the following command:
`curl http://localhost:9104/metric`

 NOTE

In this example, metrics are collected every 30s. Therefore, you can check the reported metrics on the AOM page about 30s later.

Verifying that Metrics Can Be Reported to AOM

- Step 1** Log in to the AOM 2.0 console.
- Step 2** In the navigation pane on the left, choose **Prometheus Monitoring > Instances**.
- Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
- Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
- Step 5** Select job `{namespace}/mysql-exporter` to query custom metrics starting with **mysql**.

----End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.3 Connecting Kafka Exporter

Application Scenario

When using Kafka, you need to monitor their running, for example, checking the cluster status and whether messages are stacked. The Prometheus monitoring function monitors Kafka running using Exporter in the CCE container scenario. This section describes how to deploy Kafka Exporter and implement alarm access.

 NOTE

You are advised to use CCE for unified Exporter management.

Prerequisites

- A CCE cluster has been created and Kafka has been installed.
- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the [kafka_exporter](#) image to SoftWare Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Deploying Kafka Exporter

Step 1 Log in to the CCE console.

Step 2 Click the connected cluster. The cluster management page is displayed.

Step 3 Perform the following operations to deploy Exporter:

1. Deploy Kafka Exporter.

In the navigation pane, choose **Workloads**. In the upper right corner, click **Create Workload**. Then select the **Deployment** workload and select a desired namespace to deploy Kafka Exporter. YAML configuration example for deploying Exporter:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: kafka-exporter # Change the name based on service requirements. You are advised to add
the Kafka instance information, for example, ckafka-2vrgx9fd-kafka-exporter.
  name: kafka-exporter # Change the name based on service requirements. You are advised to add
the Kafka instance information, for example, ckafka-2vrgx9fd-kafka-exporter.
  namespace: default # Namespace of an existing cluster
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: kafka-exporter # Change the name based on service requirements. You are advised to
add the Kafka instance information, for example, ckafka-2vrgx9fd-kafka-exporter.
  template:
    metadata:
      labels:
        k8s-app: kafka-exporter # Change the name based on service requirements. You are advised to
add the Kafka instance information, for example, ckafka-2vrgx9fd-kafka-exporter.
    spec:
      containers:
        - args:
            - --kafka.server=120.46.215.4:30092 # Address of the Kafka instance
          image: swr.cn-north-4.myhuaweicloud.com/mall-swarm-demo/kafka-exporter:latest
          imagePullPolicy: IfNotPresent
          name: kafka-exporter
          ports:
            - containerPort: 9308
              name: metric-port # Required when you configure a collection task
          securityContext:
            privileged: false
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          dnsPolicy: ClusterFirst
          imagePullSecrets:
```

```
- name: default-secret
restartPolicy: Always
schedulerName: default-scheduler
securityContext: {}
terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  name: kafka-exporter
spec:
  type: NodePort
  selector:
    k8s-app: kafka-exporter
  ports:
    - protocol: TCP
      nodePort: 30091
      port: 9308
      targetPort: 9308
```

NOTE

For more details about Exporter parameters, see [kafka-exporter](#).

2. Check whether Kafka Exporter is successfully deployed.
 - a. On the **Deployments** tab page, click the Deployment created in [Step 3.1](#). In the pod list, choose **More > View Logs** in the **Operation** column. The Exporter is successfully started and its access address is exposed.
 - b. Perform verification using one of the following methods:
 - Log in to a cluster node and run either of the following commands:
`curl http://{Cluster IP address}:9308/metrics`
`curl http://{Private IP address of any node in the cluster}:30091/metrics`
 - In the instance list, choose **More > Remote Login** in the **Operation** column and run the following command:
`curl http://localhost:9308/metric`
 - Access `http://{Public IP address of any node in the cluster}:30091/metrics`.

Figure 4-5 Accessing a cluster node

```

30091/metrics
go_memstats_mcache_inuse_bytes 19200
# HELP go_memstats_mcache_sys_bytes Number of bytes used for mcache structures obtained from system.
# TYPE go_memstats_mcache_sys_bytes gauge
go_memstats_mcache_sys_bytes 25768
# HELP go_memstats_mspan_inuse_bytes Number of bytes in use by mspan structures.
# TYPE go_memstats_mspan_inuse_bytes gauge
go_memstats_mspan_inuse_bytes 46240
# HELP go_memstats_mspan_sys_bytes Number of bytes used for mspan structures obtained from system.
# TYPE go_memstats_mspan_sys_bytes gauge
go_memstats_mspan_sys_bytes 49152
# HELP go_memstats_next_gc_bytes Number of heap bytes when next garbage collection will take place.
# TYPE go_memstats_next_gc_bytes gauge
go_memstats_next_gc_bytes 4.473824e+06
# HELP go_memstats_other_sys_bytes Number of bytes used for other system allocations.
# TYPE go_memstats_other_sys_bytes gauge
go_memstats_other_sys_bytes 1.074585e+06
# HELP go_memstats_stack_inuse_bytes Number of bytes in use by the stack allocator.
# TYPE go_memstats_stack_inuse_bytes gauge
go_memstats_stack_inuse_bytes 524288
# HELP go_memstats_stack_sys_bytes Number of bytes obtained from system for stack allocator.
# TYPE go_memstats_stack_sys_bytes gauge
go_memstats_stack_sys_bytes 524288
# HELP go_memstats_sys_bytes Number of bytes obtained from system.
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 1.5156485e+07
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 6
# HELP kafka_brokers Number of Brokers in the Kafka Cluster.
# TYPE kafka_brokers gauge
kafka_brokers 1
# HELP kafka_exporter_build_info A metric with a constant '1' value labeled by version, revision, branch, and goversion from which kafka_exporter was built.
# TYPE kafka_exporter_build_info gauge
kafka_exporter_build_info {branch="HEAD", goversion="go1.17.3", revision="15e4ad6a9ea8203135d4b974e825f22e31c750e5", version="1.4.2"} 1
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.02
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 10
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.513472e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.70253782409e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 7.3420944e+08
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.8446744037709552e+19
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.

```

----End

Collecting Service Data of the CCE Cluster

Add PodMonitor to configure a collection rule for monitoring the service data of applications deployed in the CCE cluster.

NOTE

In the following example, metrics are collected every 30s. Therefore, you can check the reported metrics on the AOM page about 30s later.

Configuration information:

```

apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: kafka-exporter
  namespace: default
spec:
  namespaceSelector:
    matchNames:
      - default # Namespace where Exporter is located.
  podMetricsEndpoints:
    - interval: 30s
      path: /metrics
      port: metric-port
  selector:
    matchLabels:
      k8s-app: kafka-exporter

```

Verifying that Metrics Can Be Reported to AOM

Step 1 Log in to the AOM 2.0 console.

Step 2 In the navigation pane on the left, choose **Prometheus Monitoring > Instances**.

- Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
- Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
- Step 5** Select job `{namespace}/kafka-exporter` to query custom metrics starting with **kafka**.
- End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.4 Connecting Memcached Exporter

Application Scenario

When using Memcached, you need to monitor their running and locate their faults in a timely manner. The Prometheus monitoring function monitors Memcached running using Exporter in the CCE container scenario. This section describes how to monitor Memcached.

NOTE

You are advised to use CCE for unified Exporter management.

Prerequisites

- A CCE cluster has been created and Memcached has been installed.
- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the [memcached_exporter](#) image to Software Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Deploying Memcached Exporter

Step 1 Log in to the CCE console.

Step 2 Click the connected cluster. The cluster management page is displayed.

Step 3 Perform the following operations to deploy Exporter:

1. Configure a secret.

In the navigation pane, choose **ConfigMaps and Secrets**. Then click **Create from YAML** in the upper right corner of the page. YAML configuration example:

```
apiVersion: v1
kind: Secret
metadata:
  name: memcached-exporter-secret
  namespace: default
type: Opaque
stringData:
  memcachedURI: 120.46.215.4:11211 # Memcached address
```

NOTE

- Format of the Memcached connection string: **http://{ip}:{port}**.
- For details about how to configure a secret, see [Creating a Secret](#).

2. Deploy Memcached Exporter.

In the navigation pane, choose **Workloads**. On the **Deployments** tab page, click **Create from YAML** in the upper right corner and then configure a YAML file to deploy Exporter.

YAML configuration example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: memcached-exporter # Change the value based on service requirements.
  name: memcached-exporter # Change the value based on service requirements.
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: memcached-exporter # Change the value based on service requirements.
  template:
    metadata:
      labels:
        k8s-app: memcached-exporter # Change the value based on service requirements.
    spec:
      containers:
        - env:
            - name: Memcached_Url
              valueFrom:
                secretKeyRef:
                  name: memcached-exporter-secret # Secret name specified in the previous step.
                  key: memcachedURI # Secret key specified in the previous step.
            - name: Memcached_ALL
              value: "true"
          image: swr.cn-east-3.myhuaweicloud.com/aom-org/bitnami/memcached-exporter:0.13.0 #
          Image uploaded to SWR, as described in "Prerequisites".
          imagePullPolicy: IfNotPresent
          name: memcached-exporter
          ports:
            - containerPort: 9150
              name: metric-port
```

```

securityContext:
  privileged: false
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
dnsPolicy: ClusterFirst
imagePullSecrets:
- name: default-secret
restartPolicy: Always
schedulerName: default-scheduler
securityContext: {}
terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  name: memcached-exporter
spec:
  type: NodePort
  selector:
    k8s-app: memcached-exporter
  ports:
    - protocol: TCP
      nodePort: 30122
      port: 9150
      targetPort: 9150

```

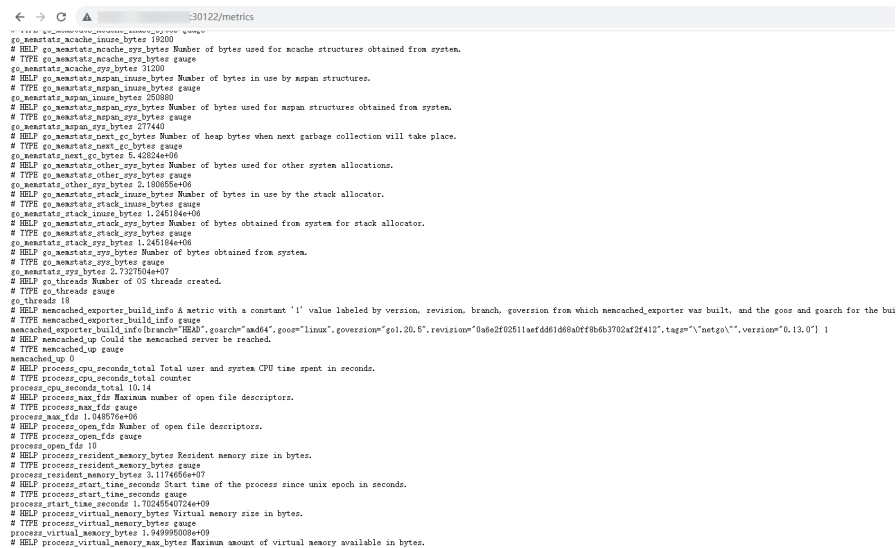
NOTE

For more details about Exporter parameters, see [memcached_exporter](#).

3. Check whether Memcached Exporter is successfully deployed.
 - a. On the **Deployments** tab page, click the Deployment created in **Step 3.2**. In the pod list, choose **More > View Logs** in the **Operation** column. The Exporter is successfully started and its access address is exposed.
 - b. Perform verification using one of the following methods:

- Log in to a cluster node and run either of the following commands:
curl http://{Cluster IP address}:9150/metrics
curl http://{Private IP address of any node in the cluster}:30122/metrics
- Access **http://{Public IP address of any node in the cluster}:30122/metrics**.

Figure 4-6 Accessing a cluster node



- In the instance list, choose **More > Remote Login** in the **Operation** column and run the following command:

```
curl http://localhost:9150/metric
```

Figure 4-7 Executing the command

```
user@ungnt6cs5eps2ff-machine:~$ curl http://localhost:9150/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 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.20.5"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 504008
# 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 504008
# 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 4545
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 0
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 6.74584e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 504008
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 1.753088e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
```

----End

Adding a Collection Task

Add PodMonitor to configure a collection rule for monitoring the service data of applications deployed in the CCE cluster.

NOTE

In the following example, metrics are collected every 30s. Therefore, you can check the reported metrics on the AOM page about 30s later.

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: memcached-exporter
  namespace: default
spec:
  namespaceSelector:
    matchNames:
      - default # Namespace where Exporter is located.
  podMetricsEndpoints:
    - interval: 30s
      path: /metrics
      port: metric-port
  selector:
    matchLabels:
      k8s-app: memcached-exporter
```

Verifying that Metrics Can Be Reported to AOM

- Step 1** Log in to the AOM 2.0 console.
 - Step 2** In the navigation pane on the left, choose **Prometheus Monitoring > Instances**.
 - Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
 - Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
 - Step 5** Select job `{namespace}/memcached-exporter` to query metrics starting with `go_memstats`.
- End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.5 Connecting MongoDB Exporter

Application Scenario

When using MongoDB, you need to monitor MongoDB running and locate their faults in a timely manner. The Prometheus monitoring function monitors MongoDB running using Exporter in the CCE container scenario. This section describes how to deploy MongoDB Exporter and implement alarm access.

NOTE

You are advised to use CCE for unified Exporter management.

Prerequisites

- A CCE cluster has been created and MongoDB has been installed.

- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the [mongodb_exporter](#) image to SoftWare Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Deploying MongoDB Exporter

Step 1 Log in to the CCE console.

Step 2 Click the connected cluster. The cluster management page is displayed.

Step 3 Perform the following operations to deploy Exporter:

1. Configure a secret.

In the navigation pane, choose **ConfigMaps and Secrets**. Then click **Create from YAML** in the upper right corner of the page. YAML configuration example:

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

NOTE

- The password has been encrypted based on Opaque requirements.
- For details about how to configure a secret, see [Creating a Secret](#).

2. Deploy MongoDB Exporter.

In the navigation pane, choose **Workloads**. In the upper right corner, click **Create Workload**. Then select the **Deployment** workload and select a desired namespace to deploy MongoDB Exporter. YAML configuration example for deploying Exporter:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: mongodb-exporter # Change the value based on service requirements. You are advised to
add the MongoDB instance information.
  name: mongodb-exporter # Change the value based on service requirements. You are advised to add
the MongoDB instance information.
  namespace: default #Must be the same as the namespace of MongoDB.
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mongodb-exporter # Change the value based on service requirements. You are advised
to add the MongoDB instance information.
  template:
    metadata:
      labels:
        k8s-app: mongodb-exporter # Change the value based on service requirements. You are advised
to add the MongoDB instance information.
    spec:
      containers:
        - args:
            - --collect.database # Enable collection of database metrics.
```

```
- --collect.collection # Enable collection of metric sets.
- --collect.topmetrics # Enable collection of database header metrics.
- --collect.indexusage # Enable collection of index usage statistics.
- --collect.connpoolstats # Enable collection of MongoDB connection pool statistics.
env:
- name: MONGODB_URI
  valueFrom:
    secretKeyRef:
      name: mongodb-secret-test
      key: datasource
image: swr.cn-north-4.myhuaweicloud.com/mall-swarm-demo/mongodb-exporter:0.10.0
imagePullPolicy: IfNotPresent
name: mongodb-exporter
ports:
- containerPort: 9216
  name: metric-port # Required when you configure a collection task.
securityContext:
  privileged: false
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
dnsPolicy: ClusterFirst
imagePullSecrets:
- name: default-secret
restartPolicy: Always
schedulerName: default-scheduler
securityContext: { }
terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  name: mongodb-exporter
spec:
  type: NodePort
  selector:
    k8s-app: mongodb-exporter
  ports:
    - protocol: TCP
      nodePort: 30003
      port: 9216
      targetPort: 9216
```

NOTE

For more details about Exporter parameters, see [mongodb_exporter](#).

3. Check whether MongoDB Exporter is successfully deployed.
 - a. On the **Deployments** tab page, click the Deployment created in [Step 3.2](#). In the pod list, choose **More > View Logs** in the **Operation** column. The Exporter is successfully started and its access address is exposed.
 - b. Perform verification using one of the following methods:
 - Log in to a cluster node and run either of the following commands:
`curl http://{Cluster IP address}:9216/metrics`
`curl http://{Private IP address of any node in the cluster}:30003/metrics`
 - Access `http://{Public IP address of any node in the cluster}:30003/metrics`.

Figure 4-8 Accessing a cluster node


```

# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# 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 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.11.13"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.81956e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.81956e+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 3124
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 3308
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 2.23436e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 1.81956e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 6.3234048e+07
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.31776e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 16998
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 0
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 6.6551808e+07
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 0
# HELP go_memstats_lookups_total Total number of pointer lookups.
# TYPE go_memstats_lookups_total counter
go_memstats_lookups_total 0

```

- In the instance list, choose **More > Remote Login** in the **Operation** column and run the following command:
curl http://localhost:9216/metric

----End

Collecting Service Data of the CCE Cluster

Add PodMonitor to configure a collection rule for monitoring the service data of applications deployed in the CCE cluster.

NOTE

In the following example, metrics are collected every 30s. Therefore, you can check the reported metrics on the AOM page about 30s later.

```

apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: mongodb-exporter
  namespace: default
spec:
  namespaceSelector:
    matchNames:
      - default # Namespace where Exporter is located.
  podMetricsEndpoints:
    - interval: 30s
      path: /metrics
      port: metric-port
      selector:
        matchLabels:
          k8s-app: mongodb-exporter

```


Verifying that Metrics Can Be Reported to AOM

- Step 1** Log in to the AOM 2.0 console.
 - Step 2** In the navigation pane on the left, choose **Prometheus Monitoring > Instances**.
 - Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
 - Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
 - Step 5** Select job *{namespace}/MongoDB-exporter* to query custom metrics starting with **mongodb**.
- End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.6 Connecting Elasticsearch Exporter

Application Scenario

When using Elasticsearch, you need to monitor Elasticsearch running, such as the cluster and index status. The Prometheus monitoring function monitors Elasticsearch running using Exporter in the CCE container scenario. This section describes how to deploy Elasticsearch Exporter and implement alarm access.

NOTE

You are advised to use CCE for unified Exporter management.

Prerequisites

- A CCE cluster has been created and Elasticsearch has been installed.

- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the [elasticsearch_exporter](#) image to SoftWare Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Deploying Elasticsearch Exporter

Step 1 Log in to the CCE console.

Step 2 Click the connected cluster. The cluster management page is displayed.

Step 3 Perform the following operations to deploy Exporter:

1. Configure a secret.

In the navigation pane, choose **ConfigMaps and Secrets**. Then click **Create from YAML** in the upper right corner of the page. The following shows a YAML configuration example:

```
apiVersion: v1
kind: Secret
metadata:
  name: es-secret-test
  namespace: default
type: Opaque
stringData:
  esURI: http://124.70.14.51:30920 # URI of Elasticsearch. Use the IP address of the cluster or any
  node in the cluster.
```

NOTE

- Format of the Elasticsearch connection string: `<proto>://<user>:<password>@<host>:<port>`, for example, **http://admin:pass@localhost:9200**. You can also leave the password blank, for example, **http://10.247.43.50:9200**.
 - The password has been encrypted based on Opaque requirements.
 - For details about how to configure a secret, see [Creating a Secret](#).
2. Deploy Elasticsearch Exporter.

In the navigation pane, choose **Workloads**. In the upper right corner, click **Create Workload**. Then select the **Deployment** workload and a desired namespace to deploy Elasticsearch Exporter. YAML configuration example for deploying Exporter:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: es-exporter # Change the value based on service requirements.
  name: es-exporter # Change the value based on service requirements.
  namespace: default #Select a proper namespace to deploy Exporter. If no namespace is available,
  create one.
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: es-exporter # Change the value based on service requirements.
  template:
    metadata:
      labels:
        k8s-app: es-exporter # Change the value based on service requirements.
    spec:
      containers:
```

```
- env:
  - name: ES_URI
    valueFrom:
      secretKeyRef:
        name: es-secret-test # Secret name specified in the previous step.
        key: esURI # Secret key specified in the previous step.
  - name: ES_ALL
    value: "true"
image: swr.cn-north-4.myhuaweicloud.com/mall-swarm-demo/es-exporter:1.1.0
imagePullPolicy: IfNotPresent
name: es-exporter
ports:
  - containerPort: 9114
    name: metric-port
securityContext:
  privileged: false
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
dnsPolicy: ClusterFirst
imagePullSecrets:
  - name: default-secret
restartPolicy: Always
schedulerName: default-scheduler
securityContext: {}
terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  name: es-exporter
  name-space: default # Must be the same as the namespace where Exporter is deployed.
spec:
  type: NodePort
  selector:
    k8s-app: es-exporter
  ports:
    - protocol: TCP
      nodePort: 30921
      port: 9114
      targetPort: 9114
```

NOTE

In the preceding example, **ES_ALL** is used to collect all Elasticsearch monitoring items. You can change parameters if needed. For more details about Exporter parameters, see [elasticsearch_exporter](#).

3. Check whether Elasticsearch Exporter is successfully deployed.
 - a. On the **Deployments** tab page, click the Deployment created in [Step 3.2](#). In the pod list, choose **More > View Logs** in the **Operation** column. The Exporter is successfully started and its access address is exposed.
 - b. Perform verification using one of the following methods:
 - Log in to a cluster node and run either of the following commands:
`curl http://{Cluster IP address}:9114/metrics`
`curl http://{Private IP address of any node in the cluster}:30921/metrics`
 - Access `http://{Public IP address of any node in the cluster}:30921/metrics`.

- Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
- Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
- Step 5** Select job `{namespace}/elasticsearch-exporter` to query custom metrics starting with `elasticsearch`.
- End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.7 Connecting Redis Exporter

Application Scenario

When using Redis, you need to monitor Redis running and locate their faults in a timely manner. The Prometheus monitoring function monitors Redis running using Exporter in the CCE container scenario. This section describes how to monitor Redis.

NOTE

You are advised to use CCE for unified Exporter management.

Prerequisites

- A CCE cluster has been created and Redis has been installed.
- Your service has been connected for Prometheus monitoring and a CCE cluster has also been connected. For details, see [Prometheus Instance for CCE](#).
- You have uploaded the `redis_exporter` image to Software Repository for Container (SWR). For details, see [Uploading an Image Through a Container Engine Client](#).

Deploying Redis Exporter

Step 1 Log in to the CCE console.

Step 2 Click the connected cluster. The cluster management page is displayed.

Step 3 Perform the following operations to deploy Exporter:

1. In the navigation pane, choose **ConfigMaps and Secrets**. Switch to the **Secrets** tab. Then click **Create from YAML** in the upper right corner of the page. The following shows a YAML configuration example:

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-secret-test
  namespace: default # Must be the same as the namespace where Exporter is deployed.
type: Opaque
stringData:
  password: redis123 # Redis password.
```

NOTE

- The password has been encrypted based on Opaque requirements.
 - For details about how to configure a secret, see [Creating a Secret](#).
2. Deploy Redis Exporter.

In the navigation pane, choose **Workloads**. On the displayed page, click the **Deployments** tab, click **Create from YAML** in the upper right corner, and select a namespace. You can deploy Exporter through the console or using a YAML file. The following shows a YAML configuration example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: redis-exporter # Change the value based on service requirements. You are advised to add
the Redis instance information, for example, crs-66e112fp-redis-exporter.
  name: redis-exporter # Change the value based on service requirements. You are advised to add the
Redis instance information, for example, crs-66e112fp-redis-exporter.
  namespace: default #Select a proper namespace to deploy Exporter. If no namespace is available,
create one.
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: redis-exporter # Change the name based on service requirements. You are advised to
add the Redis instance information, for example, crs-66e112fp-redis-exporter.
  template:
    metadata:
      labels:
        k8s-app: redis-exporter # Change the name based on service requirements. You are advised to
add the Redis instance information, for example, crs-66e112fp-redis-exporter.
    spec:
      containers:
        - env:
            - name: REDIS_ADDR
              value: 120.46.215.4:30379 # IP address:port number of Redis
            - name: REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: redis-secret-test # Secret name specified in the previous step.
                  key: password # Secret key specified in the previous step.
          image: swr.cn-north-4.myhuaweicloud.com/mall-swarm-demo/redis-exporter:v1.32.0 # Replace
the value with the address of the image you uploaded to SWR.
          imagePullPolicy: IfNotPresent
          name: redis-exporter
        ports:
```

```
- containerPort: 9121
  name: metric-port # Required when you configure a collection task.
  securityContext:
    privileged: false
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-secret
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  name: redis-exporter
  namespace: default # Must be the same as the namespace where Exporter is deployed.
spec:
  type: NodePort
  selector:
    k8s-app: redis-exporter
  ports:
    - protocol: TCP
      nodePort: 30378
      port: 9121
      targetPort: 9121
```

NOTE

For more details about Exporter parameters, see [redis_exporter](#).

3. Check whether Redis Exporter is successfully deployed.
 - a. On the **Deployments** tab page, click the Deployment created in [Step 3.2](#). In the pod list, choose **More** > **View Logs** in the **Operation** column. The Exporter is successfully started and its access address is exposed.
 - b. Perform verification using one of the following methods:
 - Log in to a cluster node and run either of the following commands:
`curl http://{Cluster IP address}:9121/metrics`
`curl http://{Private IP address of any node in the cluster}:30378/metrics`
 - Access `http://{Public IP address of any node in the cluster}:30378/metrics`.
If no data is obtained, check whether the values of "REDIS_ADDR" and "REDIS_PASSWORD" in the YAML file set during [Redis Exporter deployment](#) are correct. The following shows an example:

Figure 4-10 Accessing a cluster node

```

← → C ▲ 30378/metrics

# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 962888
# 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 962888
# 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 4236
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 178
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 4.067e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 962888
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 1.769472e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 1.998848e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 4037
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 1.769472e+06

```

- In the instance list, choose **More > Remote Login** in the **Operation** column and run the following command:
`curl http://localhost:9121/metrics`

Figure 4-11 Executing the command

```

redis-exporter      nodeport      10.241.222.95      <none>      9121:30378/ILP      35
user@anisfyg9ulitku8-machine:~$ curl http://localhost:30378/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 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.029288e+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 2.029288e+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 4236
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 384
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 0
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 4.09784e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge

```

----End

Adding a Collection Task

Add PodMonitor to configure a collection rule for monitoring the service data of applications deployed in the CCE cluster.

NOTE

In the following example, metrics are collected every 30s. Therefore, you can check the reported metrics on the AOM page about 30s later.


```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: redis-exporter
  namespace: default
spec:
  namespaceSelector: # Select the namespace where the target Exporter pod is located.
  matchNames:
    - default # Namespace where Exporter is located.
  podMetricsEndpoints:
    - interval: 30s # Set the metric collection period.
      path: /metrics # Enter the path corresponding to Prometheus Exporter. Default: /metrics.
      port: metric-port# Enter the name of "ports" in the YAML file corresponding to Prometheus Exporter.
      selector: # Enter the label of the target Exporter pod.
      matchLabels:
        k8s-app: redis-exporter
```

Verifying that Metrics Can Be Reported to AOM

- Step 1** Log in to the AOM 2.0 console.
- Step 2** In the navigation pane on the left, choose **Prometheus Monitoring > Instances**.
- Step 3** Click the Prometheus instance connected to the CCE cluster. The instance details page is displayed.
- Step 4** On the **Metrics** tab page of the **Metric Management** page, select your target cluster.
- Step 5** Enter **redis** in the search box to search. If metrics starting with **redis** are displayed, the metrics are successfully connected to AOM.

----End

Setting a Dashboard and Alarm Rule on AOM

By setting a dashboard, you can monitor CCE cluster data on the same screen. By setting an alarm rule, you can detect cluster faults and implement warning in a timely manner.

- Setting a dashboard
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Dashboard**. On the displayed page, click **Add Dashboard** to add a dashboard. For details, see [Creating a Dashboard](#).
 - c. On the **Dashboard** page, select a Prometheus instance for CCE and click **Add Graph**. For details, see [Adding a Graph to a Dashboard](#).
- Setting an alarm rule
 - a. Log in to the AOM 2.0 console.
 - b. In the navigation pane, choose **Alarm Management > Alarm Rules**.
 - c. On the **Metric/Event Alarm Rules** tab page, click **Create** to create an alarm rule. For details, see [Creating a Metric Alarm Rule](#).

4.8 Connecting Other Exporters

Application Scenario

Guidance has been provided for connecting common Exporters. AOM is compatible with the native Prometheus, so you can also connect other Exporters in the community.

Methods

Customize dashboards or use either of the following methods to integrate basic components for monitoring:

1. [Integrating Exporters in the open-source community](#)
2. Instructions in [connecting common self-built middleware in the container scenario](#)