



OpenShift Container Platform 4.7

Metering

在 OpenShift Container Platform 中配置和使用 Metering

OpenShift Container Platform 4.7 Metering

在 OpenShift Container Platform 中配置和使用 Metering

法律通告

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本文档提供在 OpenShift Container Platform 中配置和使用 Metering 的说明。

目录

第 1 章 关于 METERING	3
1.1. METERING 概述	3
第 2 章 安装 METERING	4
2.1. 先决条件	4
2.2. 安装 METERING OPERATOR	4
2.3. 安装 METERING 堆栈	6
2.4. 先决条件	7
2.5. 验证 METERING 安装	7
2.6. 其他资源	9
第 3 章 升级 METERING	10
3.1. 先决条件	10
第 4 章 配置 METERING	13
4.1. 关于配置 METERING	13
4.2. 通用配置选项	13
4.3. 配置持久性存储	16
4.4. 配置 HIVE METASTORE	24
4.5. 配置 REPORTING OPERATOR	26
4.6. 配置 AWS 账单相关性	31
第 5 章 REPORTS	34
5.1. 关于报告	34
5.2. 存储位置	41
第 6 章 使用 METERING	43
6.1. 先决条件	43
6.2. 编写报告	43
6.3. 查看报告结果	44
第 7 章 METERING 使用示例	47
7.1. 先决条件	47
7.2. 每小时和每日测量集群容量	47
7.3. 通过一次性报告来衡量集群使用量	48
7.4. 使用 CRON 表达式来测量集群利用率	48
第 8 章 METERING 故障排除与调试	49
8.1. METERING 故障排除	49
8.2. METERING 调试	52
第 9 章 卸载 METERING	57
9.1. 从集群中移除 METERING OPERATOR	57
9.2. 卸载 METERING 命名空间	57
9.3. 卸载 METERING 自定义资源定义	57

第 1 章 关于 METERING

1.1. METERING 概述

Metering 是一个通用数据分析工具，您可使用该工具编写报告，以处理来自不同数据源的数据。作为集群管理员，您可使用 Metering 来分析集群中的情况。您可以自行编写报告，也可以使用预定义的 SQL 查询来定义如何处理来自现有不同数据源的数据。

Metering 侧重于处理集群内的指标数据，使用 Prometheus 作为默认数据源，支持 Metering 用户针对 Pod、命名空间和 Kubernetes 的其他大部分资源进行报告。

您可以在 OpenShift Container Platform 4.x 集群及更高版本上安装 metering。

1.1.1. Metering 资源

Metering 具有很多资源，可用于管理 Metering 的部署与安装以及 Metering 提供的报告功能。

Metering 使用以下自定义资源定义 (CRD) 来管理：

MeteringConfig	为部署配置 metering 堆栈。包含用于控制 metering 堆栈各个组件的自定义和配置选项。
报告	控制要使用的查询、查询运行时间、运行频率以及查询结果的存储位置。
ReportQuery	包含用于对 ReportDataSource 资源中包含的数据进行分析的 SQL 查询。
ReportDataSource	控制 ReportQuery 和 Report 资源可用的数据。支持配置 metering 中使用的不同数据库的访问权限。

第 2 章 安装 METERING

在将 metering 安装到集群中之前，请查看以下部分。

开始安装 Metering 前，请先通过 OperatorHub 安装 Metering Operator。然后再通过创建一个 **MeteringConfig** 自定义资源（CR）来配置 Metering 实例。安装 Metering Operator 会创建一个默认 **MeteringConfig** 资源，您可使用文档中的示例对其进行修改。创建 **MeteringConfig** 资源后，安装 metering 堆栈。最后，验证您的安装。

2.1. 先决条件

Metering 需包含以下组件：

- 用于动态卷置备的 **StorageClass** 资源。Metering 支持多种不同存储解决方案。
- 4GB 内存和 4 个 CPU 内核可用集群容量，至少要有一个具有 2 个 CPU 内核和 2GB 内存容量的节点。
- 由 metering 安装的单个最大 pod 至少需要 2GB 内存和 2 个 CPU 内核的资源。
 - 内存和 CPU 消耗通常较低，但在运行报告或为较大集群收集数据时会激增。

2.2. 安装 METERING OPERATOR

您可以通过部署 Metering Operator 来安装 metering。Metering Operator 会创建和管理 metering 堆栈的组件。



注意

您不能通过 web 控制台或在 CLI 中使用 **oc new-project** 命令创建名称以 **openshift-** 开始的项目。



注意

如果 Metering Operator 使用 **openshift-metering** 以外的命名空间安装，则 Metering 报告只能使用 CLI 查看。在整个安装步骤中强烈建议您使用 **openshift-metering** 命名空间。

2.2.1. 使用 web 控制台安装 metering

使用 OpenShift Container Platform web 控制台安装 Metering Operator。

流程

1. 使用 **oc create -f <file-name>.yaml** 命令为 Metering Operator 创建一个命名空间对象 YAML 文件。您必须使用 CLI 创建命名空间。例如，**metering-namespace.yaml**：

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-metering 1
  annotations:
```



```
openshift.io/node-selector: "" 2
labels:
  openshift.io/cluster-monitoring: "true"
```

- 1 强烈建议在 **openshift-metering** 命名空间中部署 metering。
 - 2 在为操作数 pod 配置特定的节点选择器前包括此注解。
2. 在 OpenShift Container Platform Web 控制台中，点击 **Operators** → **OperatorHub**。使用 **metering** 过滤以查找 Metering Operator。
 3. 点 **Metering** 卡，查看软件包描述，然后点 **Install**。
 4. 选择一个 **Update Channel**、**Installation Mode** 和 **Approval Strategy**。
 5. 点击 **Install**。
 6. 通过切换到 **Operators** → **Installed Operators** 页来验证 Metering Operator 已被安装。安装完成后，Metering Operator 的状态为 **Succeeded**。



注意

这可能需要几分钟时间才会显示 Metering Operator。

7. 点 **Installed Operators** 页中的 **Metering** 来查看 Operator 的详细信息。在 **Operator Details** 页中，还可创建其他与 metering 相关的资源。

要完成 metering 的安装，创建一个 **MeteringConfig** 资源来配置 metering 并安装 metering 堆栈的组件。

2.2.2. 使用 CLI 安装 metering

您可以使用 OpenShift Container Platform CLI 安装 Metering Operator。

流程

1. 为 Metering Operator 创建 **Namespace** 对象 YAML 文件。您必须使用 CLI 创建命名空间。例如，**metering-namespace.yaml**：

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-metering 1
  annotations:
    openshift.io/node-selector: "" 2
  labels:
    openshift.io/cluster-monitoring: "true"
```

- 1 强烈建议在 **openshift-metering** 命名空间中部署 metering。
 - 2 在为操作数 pod 配置特定的节点选择器前包括此注解。
2. 创建 **Namespace** 对象：

```
$ oc create -f <file-name>.yaml
```

例如：

```
$ oc create -f openshift-metering.yaml
```

3. 创建 **OperatorGroup** 对象 YAML 文件。例如, **metering-og**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-metering ①
  namespace: openshift-metering ②
spec:
  targetNamespaces:
  - openshift-metering
```

- ① 名称是任意名称。
- ② 指定 **openshift-metering** 命名空间。

4. 创建一个 **Subscription** 对象 YAML 文件，以便为 Metering Operator 订阅一个命名空间。此对象以 **redhat-operators** 目录源中最新发布的目标版本为目标。例如, **metering-sub.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metering-ocp ①
  namespace: openshift-metering ②
spec:
  channel: "4.7" ③
  source: "redhat-operators" ④
  sourceNamespace: "openshift-marketplace"
  name: "metering-ocp"
  installPlanApproval: "Automatic" ⑤
```

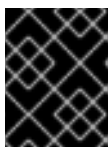
- ① 名称是任意名称。
- ② 您必须指定 **openshift-metering** 命名空间。
- ③ 指定 4.7 作为频道。
- ④ 指定包含 **metering-ocp** 软件包清单的 **redhat-operators** 目录源。如果 OpenShift Container Platform 集群安装在受限网络中（也称为断开连接的集群），请指定配置 Operator LifeCycle Manager (OLM) 时创建的 **CatalogSource** 对象的名称。
- ⑤ 指定 "Automatic" 安装计划批准。

2.3. 安装 METERING 堆栈

将 Metering Operator 添加至集群后，您可通过安装 metering 堆栈来安装 metering 组件。

2.4. 先决条件

- 查看 [配置选项](#)
- 创建 **MeteringConfig** 资源。您可通过以下流程生成默认 **MeteringConfig** 资源，然后根据您安装的具体情况，使用文档中的示例来修改默认文件。查看以下主题以创建 **MeteringConfig** 资源：
 - 有关配置选项，请查看[关于配置 metering](#)。
 - 您至少需要[配置持久性存储](#)和[配置 Hive metastore](#)。



重要

openshift-metering 命名空间中只能有一个 **MeteringConfig** 资源。不支持任何其它配置。

流程

1. 在 Web 控制台中，确保进入 **openshift-metering** 项目中的 Metering Operator 的 **Operator Details** 页面。您可通过点 **Operators** → **Installed Operators**，然后选择 Metering Operator 进入该页面。
2. 在 **Provided APIs** 项下，点 Metering Configuration 卡上的 **Create Instance**。这会打开 YAML 编辑器，其中有一个默认 **MeteringConfig** 资源文件，您可通过该文件定义您的配置。



注意

如需示例配置文件和所有支持的配置选项,请参阅 [配置 metering 文档](#)。

3. 在 YAML 编辑器中输入 **MeteringConfig** 资源并点 **Create**。

MeteringConfig 资源开始为您的 metering 堆栈创建所需资源。您现在可继续验证您的安装。

2.5. 验证 METERING 安装

您可以通过执行以下任一方式来验证 metering 的安装：

- 检查 metering 版本中的 Metering Operator **ClusterServiceVersion** (CSV) 资源。这可以通过 web 控制台或 CLI 完成。

流程 (UI)

1. 进入 **openshift-metering** 命名空间中的 **Operators** → **Installed Operators**。
2. 点 **Metering Operator**
3. 点 **Subscription** 查看 **Subscription Details**。
4. 检查 **Installed Version**。

流程 (CLI)

- 检查 **openshift-metering** 命名空间中的 Metering Operator CSV:

-

```
$ oc --namespace openshift-metering get csv
```

输出示例

NAME	DISPLAY	VERSION	REPLACES
elasticsearch-operator.4.7.0-202006231303.p0		OpenShift Elasticsearch Operator	
4.7.0-202006231303.p0	Succeeded		
metering-operator.v4.7.0	Metering	4.7.0	
Succeeded			

- 检查是否在 **openshift-metering** 命名空间中创建了所有必需的 pod:这可以通过 web 控制台或 CLI 完成。



注意

很多 pod 在就绪前需要依靠其他组件才能发挥作用。如果其他 pod 需要很长时间才能启动，则一些 Pod 可能会重启。这预期会在 Metering Operator 安装过程中发生。

流程 (UI)

- 在 metering 命名空间中导航至 **Workloads → Pods**，验证是否已创建 pod。安装 metering 堆栈后可能需要几分钟时间。

流程 (CLI)

- 检查是否在 **openshift-metering** 命名空间中创建了所有必需的 pod :

```
$ oc -n openshift-metering get pods
```

输出示例

NAME	READY	STATUS	RESTARTS	AGE
hive-metastore-0	2/2	Running	0	3m28s
hive-server-0	3/3	Running	0	3m28s
metering-operator-68dd64cfb6-2k7d9	2/2	Running	0	5m17s
presto-coordinator-0	2/2	Running	0	3m9s
reporting-operator-5588964bf8-x2tkn	2/2	Running	0	2m40s

- 验证 **ReportDataSource** 资源是否开始导入数据，这可以通过 **EARLIEST METRIC** 栏中的有效时间戳指示。这可能需要几分钟。过滤掉未导入数据的带有“-raw”的 **ReportDataSource** 资源 :

```
$ oc get reportdatasources -n openshift-metering | grep -v raw
```

输出示例

NAME	EARLIEST METRIC	NEWEST METRIC	IMPORT
START	IMPORT END	LAST IMPORT TIME	AGE
node-allocatable-cpu-cores	2019-08-05T16:52:00Z	2019-08-05T18:52:00Z	
2019-08-05T16:52:00Z	2019-08-05T18:52:00Z	2019-08-05T18:54:45Z	9m50s
node-allocatable-memory-bytes	2019-08-05T16:51:00Z	2019-08-05T18:51:00Z	

```

2019-08-05T16:51:00Z 2019-08-05T18:51:00Z 2019-08-05T18:54:45Z 9m50s
node-capacity-cpu-cores          2019-08-05T16:51:00Z 2019-08-05T18:29:00Z
2019-08-05T16:51:00Z 2019-08-05T18:29:00Z 2019-08-05T18:54:39Z 9m50s
node-capacity-memory-bytes      2019-08-05T16:52:00Z 2019-08-05T18:41:00Z
2019-08-05T16:52:00Z 2019-08-05T18:41:00Z 2019-08-05T18:54:44Z 9m50s
persistentvolumeclaim-capacity-bytes 2019-08-05T16:51:00Z 2019-08-05T18:29:00Z
2019-08-05T16:51:00Z 2019-08-05T18:29:00Z 2019-08-05T18:54:43Z 9m50s
persistentvolumeclaim-phase     2019-08-05T16:51:00Z 2019-08-05T18:29:00Z
2019-08-05T16:51:00Z 2019-08-05T18:29:00Z 2019-08-05T18:54:28Z 9m50s
persistentvolumeclaim-request-bytes 2019-08-05T16:52:00Z 2019-08-05T18:30:00Z
2019-08-05T16:52:00Z 2019-08-05T18:30:00Z 2019-08-05T18:54:34Z 9m50s
persistentvolumeclaim-usage-bytes 2019-08-05T16:52:00Z 2019-08-05T18:30:00Z
2019-08-05T16:52:00Z 2019-08-05T18:30:00Z 2019-08-05T18:54:36Z 9m49s
pod-limit-cpu-cores             2019-08-05T16:52:00Z 2019-08-05T18:30:00Z 2019-
08-05T16:52:00Z 2019-08-05T18:30:00Z 2019-08-05T18:54:26Z 9m49s
pod-limit-memory-bytes          2019-08-05T16:51:00Z 2019-08-05T18:40:00Z 2019-
08-05T16:51:00Z 2019-08-05T18:40:00Z 2019-08-05T18:54:30Z 9m49s
pod-persistentvolumeclaim-request-info 2019-08-05T16:51:00Z 2019-08-05T18:40:00Z
2019-08-05T16:51:00Z 2019-08-05T18:40:00Z 2019-08-05T18:54:37Z 9m49s
pod-request-cpu-cores           2019-08-05T16:51:00Z 2019-08-05T18:18:00Z 2019-
08-05T16:51:00Z 2019-08-05T18:18:00Z 2019-08-05T18:54:24Z 9m49s
pod-request-memory-bytes        2019-08-05T16:52:00Z 2019-08-05T18:08:00Z
2019-08-05T16:52:00Z 2019-08-05T18:08:00Z 2019-08-05T18:54:32Z 9m49s
pod-usage-cpu-cores             2019-08-05T16:52:00Z 2019-08-05T17:57:00Z 2019-
08-05T16:52:00Z 2019-08-05T17:57:00Z 2019-08-05T18:54:10Z 9m49s
pod-usage-memory-bytes          2019-08-05T16:52:00Z 2019-08-05T18:08:00Z
2019-08-05T16:52:00Z 2019-08-05T18:08:00Z 2019-08-05T18:54:20Z 9m49s

```

当所有 pod 就绪，且验证了数据已可以被导入后，您就可以开始使用 metering 来收集数据并对集群进行报告。

2.6. 其他资源

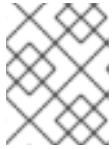
- 有关配置步骤和可用存储平台的更多信息，请参阅 [配置持久性存储](#)。
- 有关配置 Hive 的步骤，请参阅 [配置 Hive metastore](#)。

第 3 章 升级 METERING

您可以通过更新 Metering Operator 订阅将 metering 升级到 4.7。

3.1. 先决条件

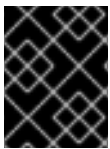
- 集群更新至 4.7。
- [Metering Operator](#) 从 OperatorHub 安装。



注意

您必须手动将 Metering Operator 升级到 4.7。如果您在以前的安装中选择了 "Automatic" **批准策略**，则 metering 不会自动升级。

- 配置了 [MeteringConfig](#) 自定义资源。
- 已安装 [metering](#) 堆栈。
- 通过检查所有 pod 就绪，确保 metering 处于健康状态。



重要

如果在安装或升级 metering 后修改 metering 存储配置，可能会发生潜在的数据丢失的问题。

流程

1. 在 web 控制台中，点 **Operators → Installed Operators**。
2. 选择 **openshift-metering** 项目：
3. 点 **Metering Operator**
4. 点 **Subscription → Channel**。
5. 在 **Change Subscription Update Channel** 窗口，选择 **4.7** 并点 **Save**。



注意

在继续执行下一步前，等待几秒钟以允许更新订阅。

6. 点 **Operators → Installed Operators**。
Metering Operator 显示为 4.7。例如：

```
Metering
4.7.0-202007012112.p0 provided by Red Hat, Inc
```

验证

您可以通过执行以下任一方式来验证 metering 的升级：

- 检查新 metering 版本中的 Metering Operator 集群服务版本 (CSV)。这可以通过 web 控制台或 CLI 完成。

流程 (UI)

1. 进入 metering 命名空间中的 **Operators** → **Installed Operators**。
2. 点 **Metering Operator**
3. 点 **Subscription** 查看 **Subscription Details**。
4. 检查 **Installed Version** 是否是已升级的 metering 版本。**Starting Version** 显示升级前的版本。

流程 (CLI)

- 检查 Metering Operator CSV:

```
$ oc get csv | grep metering
```

metering 从 4.6 升级到 4.7 的输出示例

NAME	DISPLAY	VERSION	REPLACES
metering-operator.4.7.0-202007012112.p0	Metering		4.7.0-202007012112.p0
metering-operator.4.6.0-202007012112.p0	Succeeded		

- 检查是否在 **openshift-metering** 命名空间中创建了所有必需的 pod:这可以通过 web 控制台或 CLI 完成。



注意

很多 pod 在就绪前需要依靠其他组件才能发挥作用。如果其他 pod 需要很长时间才能启动，则一些 Pod 可能会重启。这个情况可能会在 Metering Operator 升级过程中发生。

流程 (UI)

- 在 metering 命名空间中导航至 **Workloads** → **Pods**，验证是否已创建 pod。升级 metering 堆栈后可能需要几分钟时间。

流程 (CLI)

- 检查是否在 **openshift-metering** 命名空间中创建了所有必需的 pod :

```
$ oc -n openshift-metering get pods
```

输出示例

```
NAME
```

NAME	READY	STATUS	RESTARTS	AGE
hive-metastore-0	2/2	Running	0	3m28s
hive-server-0	3/3	Running	0	3m28s
metering-operator-68dd64cfb6-2k7d9	2/2	Running	0	5m17s
presto-coordinator-0	2/2	Running	0	3m9s
reporting-operator-5588964bf8-x2tkn	2/2	Running	0	2m40s

- 验证 **ReportDataSource** 资源是否导入新数据，这由 **NEWEST METRIC** 栏中的有效时间戳指示。这可能需要几分钟。过滤掉未导入数据的带有“-raw”的 **ReportDataSource** 资源：

```
$ oc get reportdatasources -n openshift-metering | grep -v raw
```

NEWEST METRIC 栏中的时间戳代表 **ReportDataSource** 资源开始导入新数据。

输出示例

NAME	EARLIEST METRIC	NEWEST METRIC	IMPORT
START	IMPORT END	LAST IMPORT TIME	AGE
node-allocatable-cpu-cores	2020-05-18T21:10:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:56:44Z	23h
node-allocatable-memory-bytes	2020-05-18T21:10:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:52:07Z	23h
node-capacity-cpu-cores	2020-05-18T21:10:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:56:52Z	23h
node-capacity-memory-bytes	2020-05-18T21:10:00Z	2020-05-19T19:57:00Z	
2020-05-18T19:10:00Z	2020-05-19T19:57:00Z	2020-05-19T19:57:03Z	23h
persistentvolumeclaim-capacity-bytes	2020-05-18T21:09:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:56:46Z	23h
persistentvolumeclaim-phase	2020-05-18T21:10:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:52:36Z	23h
persistentvolumeclaim-request-bytes	2020-05-18T21:10:00Z	2020-05-19T19:57:00Z	
2020-05-18T19:10:00Z	2020-05-19T19:57:00Z	2020-05-19T19:57:03Z	23h
persistentvolumeclaim-usage-bytes	2020-05-18T21:09:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:52:02Z	23h
pod-limit-cpu-cores	2020-05-18T21:10:00Z	2020-05-19T19:57:00Z	2020-
05-18T19:10:00Z	2020-05-19T19:57:00Z	2020-05-19T19:57:02Z	23h
pod-limit-memory-bytes	2020-05-18T21:10:00Z	2020-05-19T19:58:00Z	2020-
05-18T19:11:00Z	2020-05-19T19:58:00Z	2020-05-19T19:59:06Z	23h
pod-persistentvolumeclaim-request-info	2020-05-18T21:10:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:52:07Z	23h
pod-request-cpu-cores	2020-05-18T21:10:00Z	2020-05-19T19:58:00Z	2020-
05-18T19:11:00Z	2020-05-19T19:58:00Z	2020-05-19T19:58:57Z	23h
pod-request-memory-bytes	2020-05-18T21:10:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:55:32Z	23h
pod-usage-cpu-cores	2020-05-18T21:09:00Z	2020-05-19T19:52:00Z	2020-
05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:54:55Z	23h
pod-usage-memory-bytes	2020-05-18T21:08:00Z	2020-05-19T19:52:00Z	
2020-05-18T19:11:00Z	2020-05-19T19:52:00Z	2020-05-19T19:55:00Z	23h
report-ns-pvc-usage			5h36m
report-ns-pvc-usage-hourly			

当所有 Pod 就绪，且验证了数据已可以被导入后，就可以开始使用 metering 来收集数据并对集群进行报告。查看之前[调度的报告](#)或创建[运行一次的 metering 报告](#)来确认 metering 的升级。

第 4 章 配置 METERING

4.1. 关于配置 METERING

MeteringConfig 自定义资源指定 metering 安装的所有配置详情。首次安装 metering 堆栈时，会生成默认 **MeteringConfig** 自定义资源。使用文档中的示例来修订此默认文件。请记住以下关键点：

- 您至少需要配置持久性存储和配置 Hive metastore。
- 大部分默认配置设置都可以正常工作，但对于大型部署或高度自定义的部署来说还应仔细检查所有配置选项。
- 部分配置选项在安装后将无法修改。

对于安装后可修改的配置选项，在 **MeteringConfig** 自定义资源中进行更改并重新应用该文件。

4.2. 通用配置选项

4.2.1. 资源请求和限值

您可针对 Pod 和卷调整 CPU、内存或存储资源请求和/或限值。以下 **default-resource-limits.yaml** 提供了一个针对各个组件设置资源请求和限值的示例。

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  reporting-operator:
    spec:
      resources:
        limits:
          cpu: 1
          memory: 500Mi
        requests:
          cpu: 500m
          memory: 100Mi
  presto:
    spec:
      coordinator:
        resources:
          limits:
            cpu: 4
            memory: 4Gi
          requests:
            cpu: 2
            memory: 2Gi
      worker:
        replicas: 0
        resources:
          limits:
            cpu: 8
            memory: 8Gi
```

```

    requests:
      cpu: 4
      memory: 2Gi
hive:
  spec:
    metastore:
      resources:
        limits:
          cpu: 4
          memory: 2Gi
        requests:
          cpu: 500m
          memory: 650Mi
    storage:
      class: null
      create: true
      size: 5Gi
  server:
    resources:
      limits:
        cpu: 1
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi

```

4.2.2. 节点选择器

您可以在特定节点集合上运行 metering 组件。在 metering 组件上设置 **nodeSelector** 来控制组件的调度位置。以下 **node-selectors.yaml** 文件提供了一个针对各个组件设置节点选择器的示例。



注意

为 operand Pod 配置特定节点选择器前，将 **openshift.io/node-selector: ""** 命名空间注解添加到 metering 命名空间 YAML 文件。将 "" 指定为注解值。

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  reporting-operator:
    spec:
      nodeSelector:
        "node-role.kubernetes.io/infra": "" ❶

  presto:
    spec:
      coordinator:
        nodeSelector:
          "node-role.kubernetes.io/infra": "" ❷

  worker:
    nodeSelector:

```

```

    "node-role.kubernetes.io/infra": "" 3
hive:
spec:
  metastore:
    nodeSelector:
      "node-role.kubernetes.io/infra": "" 4
  server:
    nodeSelector:
      "node-role.kubernetes.io/infra": "" 5

```

- 1 2 3 4 5 添加 **nodeSelector** 参数，并设为适用于您想要移动的组件的值。您可以根据为节点指定的值，按所示格式使用 **nodeSelector** 或使用键-值对。



注意

为 operand Pod 配置特定节点选择器前，将 **openshift.io/node-selector: ""** 命名空间注解添加到 **metering** 命名空间 YAML 文件。在项目上设置 **openshift.io/node-selector** 注解时，该值优先于集群范围的 **Scheduler** 对象中的 **spec.defaultNodeSelector** 字段的值。

验证

您可以通过执行以下任一检查来验证 **metering** 节点选择器：

- 验证 **MeteringConfig** 自定义资源中配置的节点的 IP 地址是否正确调度了 **metering** 的所有 pod：
 - 检查 **openshift-metering** 命名空间中的所有 pod:

```
$ oc --namespace openshift-metering get pods -o wide
```

输出显示了在 **openshift-metering** 命名空间中运行的每个 Pod 的 **NODE** 和对应 **IP**。

输出示例

```

NAME                                READY STATUS RESTARTS AGE IP          NODE
NOMINATED NODE READINESS GATES
hive-metastore-0                    1/2  Running  0       4m33s 10.129.2.26 ip-10-0-210-167.us-east-2.compute.internal <none> <none>
hive-server-0                      2/3  Running  0       4m21s 10.128.2.26 ip-10-0-150-175.us-east-2.compute.internal <none> <none>
metering-operator-964b4fb55-4p699  2/2  Running  0       7h30m 10.131.0.33 ip-10-0-189-6.us-east-2.compute.internal <none> <none>
nfs-server                         1/1  Running  0       7h30m 10.129.2.24 ip-10-0-210-167.us-east-2.compute.internal <none> <none>
presto-coordinator-0              2/2  Running  0       4m8s  10.131.0.35 ip-10-0-189-6.us-east-2.compute.internal <none> <none>
reporting-operator-869b854c78-8g2x5 1/2  Running  0       7h27m 10.128.2.25 ip-10-0-150-175.us-east-2.compute.internal <none> <none>

```

- 将 **openshift-metering** 命名空间中的节点与集群中的每个节点 **NAME** 进行比较：

```
$ oc get nodes
```

输出示例

```

NAME                                STATUS ROLES  AGE  VERSION
ip-10-0-147-106.us-east-2.compute.internal Ready  master  14h  v1.20.0+6025c28
ip-10-0-150-175.us-east-2.compute.internal Ready  worker  14h  v1.20.0+6025c28
ip-10-0-175-23.us-east-2.compute.internal Ready  master  14h  v1.20.0+6025c28
ip-10-0-189-6.us-east-2.compute.internal Ready  worker  14h  v1.20.0+6025c28
ip-10-0-205-158.us-east-2.compute.internal Ready  master  14h  v1.20.0+6025c28
ip-10-0-210-167.us-east-2.compute.internal Ready  worker  14h  v1.20.0+6025c28

```

- 验证 **MeteringConfig** 自定义资源中节点选择器配置不会影响集群范围节点选择器配置，因此没有调度 metering operand Pod。
 - 检查集群范围的 **Scheduler** 对象 **spec.defaultNodeSelector** 字段，这显示调度 pod 的默认位置：

```
$ oc get schedulers.config.openshift.io cluster -o yaml
```

4.3. 配置持久性存储

Metering 需要持久性存储来保留 Metering Operator 收集的数据并存储报告结果。它支持多种存储系统和存储格式。选择您的存储系统并修改示例配置文件，以便为您的 metering 安装配置持久性存储。

4.3.1. 将数据存储至 Amazon S3 中

Metering 可以使用现有的 Amazon S3 存储桶，或为存储创建存储桶。



注意

Metering 不会管理或删除任何 S3 存储桶数据。您必须手动清理用于存储 metering 数据的 S3 存储桶。

流程

1. 编辑 **s3-storage.yaml** 文件中的 **spec.storage** 部分：

s3-storage.yaml 文件示例

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "s3"
      s3:
        bucket: "bucketname/path/" ①
        region: "us-west-1" ②
        secretName: "my-aws-secret" ③

```

```
# Set to false if you want to provide an existing bucket, instead of
# having metering create the bucket on your behalf.
```

```
createBucket: true 4
```

- 1** 指定要存储数据的存储桶的名称。可选：指定存储桶中的路径。
- 2** 指定存储桶的区域。
- 3** metering 命名空间中的一个 secret 的名称。它包含了 AWS 凭证信息（**data.aws-access-key-id** 和 **data.aws-secret-access-key**）。更多详情请查看以下示例 **Secret** 对象。
- 4** 如果要提供现有 S3 存储桶，或者如果不想提供具有 **CreateBucket** 权限的 IAM 凭证，则把该字段设置为 **false**。

2. 使用以下 **Secret** 对象作为模板：

AWS Secret 对象示例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-aws-secret
data:
  aws-access-key-id: "dGVzdAo="
  aws-secret-access-key: "c2VjcmV0Cg=="
```

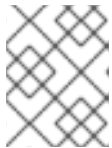


注意

aws-access-key-id 值和 **aws-secret-access-key** 值必须采用 base64 编码。

3. 创建 secret：

```
$ oc create secret -n openshift-metering generic my-aws-secret \
  --from-literal=aws-access-key-id=my-access-key \
  --from-literal=aws-secret-access-key=my-secret-key
```



注意

该命令会对您的 **aws-access-key-id** 值和 **aws-secret-access-key** 值自动进行 base64 编码。

aws-access-key-id 和 **aws-secret-access-key** 凭证必须具有存储桶的读取和写入权限。以下 **aws/read-write.json** 文件显示授予所需权限的 IAM 策略：

aws/read-write.json 文件示例

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
```

```

    "Action": [
      "s3:AbortMultipartUpload",
      "s3:DeleteObject",
      "s3:GetObject",
      "s3:HeadBucket",
      "s3:ListBucket",
      "s3:ListMultipartUploadParts",
      "s3:PutObject"
    ],
    "Resource": [
      "arn:aws:s3:::operator-metering-data/*",
      "arn:aws:s3:::operator-metering-data"
    ]
  }
]
}

```

如果 `s3-storage.yaml` 文件中的 `spec.storage.hive.s3.createBucket` 被设置为 `true` 或取消设置，那么您应该使用包含创建和删除存储桶权限的 `aws/read-write-create.json` 文件：

aws/read-write-create.json 文件示例

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:HeadBucket",
        "s3:ListBucket",
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:ListMultipartUploadParts",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::operator-metering-data/*",
        "arn:aws:s3:::operator-metering-data"
      ]
    }
  ]
}

```

4.3.2. 将数据存储至 S3 兼容存储中

您可以使用 S3 兼容存储，如 Noobaa。

流程

1. 编辑 `s3-compatible-storage.yaml` 文件中的 `spec.storage` 部分：

s3-compatible-storage.yaml 文件示例

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "s3Compatible"
      s3Compatible:
        bucket: "bucketname" ❶
        endpoint: "http://example:port-number" ❷
        secretName: "my-aws-secret" ❸

```

❶ 指定 S3 兼容存储桶的名称。

❷ 指定存储端点。

❸ metering 命名空间中的一个 secret 的名称。它包含了 AWS 凭证信息 (**data.aws-access-key-id** 和 **data.aws-secret-access-key**)。更多详情请查看以下示例 **Secret** 对象。

2. 使用以下 **Secret** 对象作为模板：

S3 兼容 Secret 对象示例

```

apiVersion: v1
kind: Secret
metadata:
  name: my-aws-secret
data:
  aws-access-key-id: "dGVzdAo="
  aws-secret-access-key: "c2VjcmV0Cg=="

```

4.3.3. 将数据存储至 Microsoft Azure 中

要将数据存储至 Azure blob 存储中，必须使用已有容器。

流程

1. 编辑 **azure-blob-storage.yaml** 文件中的 **spec.storage** 部分：

azure-blob-storage.yaml 文件示例

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:

```

```

type: "azure"
azure:
  container: "bucket1" ❶
  secretName: "my-azure-secret" ❷
  rootDirectory: "/testDir" ❸

```

- ❶ 指定容器名称。
- ❷ 指定 metering 命名空间中的 secret。更多详情请查看以下示例 **Secret** 对象。
- ❸ 可选：指定要存储数据的目录。

2. 使用以下 **Secret** 对象作为模板：

Azure Secret 对象示例

```

apiVersion: v1
kind: Secret
metadata:
  name: my-azure-secret
data:
  azure-storage-account-name: "dGVzdAo="
  azure-secret-access-key: "c2VjcmV0Cg=="

```

3. 创建 secret：

```

$ oc create secret -n openshift-metering generic my-azure-secret \
  --from-literal=azure-storage-account-name=my-storage-account-name \
  --from-literal=azure-secret-access-key=my-secret-key

```

4.3.4. 将数据存储至 Google Cloud Storage 中

要将数据存储至 Google Cloud Storage 中，您必须使用现有的存储桶。

流程

1. 编辑 **gcs-storage.yaml** 文件中的 **spec.storage** 部分：

gcs-storage.yaml 文件示例

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "gcs"
      gcs:
        bucket: "metering-gcs/test1" ❶
        secretName: "my-gcs-secret" ❷

```


- 1 指定存储桶的名称。您可选择在存储桶中指定要存储数据的目录。
- 2 指定 metering 命名空间中的 secret。更多详情请查看以下示例 **Secret** 对象。

2. 使用以下 **Secret** 对象作为模板：

Google Cloud Storage Secret 对象示例

```
apiVersion: v1
kind: Secret
metadata:
  name: my-gcs-secret
data:
  gcs-service-account.json: "c2VjcmV0Cg=="
```

3. 创建 secret：

```
$ oc create secret -n openshift-metering generic my-gcs-secret \
--from-file gcs-service-account.json=/path/to/my/service-account-key.json
```

4.3.5. 将数据存储至共享卷中

Metering 默认不配置存储。但是，您可以使用任何 ReadWriteMany 持久性卷（PV）或置备 ReadWriteMany PV 的存储类作为 Metering 存储。



注意

不建议在生产环境中使用 NFS。将 RHEL 上的 NFS 服务器用作存储后端可能无法满足 metering 要求，并可能无法提供 Metering Operator 正常工作所需的性能。

市场中的其他 NFS 实现可能没有这些问题，如 Parallel Network File System（pNFS）。pNFS 是一个带有分布式和并行功能的 NFS 实现。如需了解更多与此问题相关的信息，请联络相关的 NFS 厂商。

流程

1. 修改 **shared-storage.yaml** 文件，以使用 ReadWriteMany 持久性卷进行存储：

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "sharedPVC"
      sharedPVC:
        claimName: "metering-nfs" 1
        # Uncomment the lines below to provision a new PVC using the specified storageClass.
```

2

```
# createPVC: true
# storageClass: "my-nfs-storage-class"
# size: 5Gi
```

从以下配置选项中任选一个：

- 1 将 **storage.hive.sharedPVC.claimName** 设置为一个现存 ReadWriteMany 持久性卷声明 (PVC) 的名称。如果您没有动态卷置备，或想要对持久性卷的创建方式拥有更多控制权，则需要此配置。
 - 2 将 **storage.hive.sharedPVC.createPVC** 设置为 **true**，并将 **storage.hive.sharedPVC.storageClass** 设置为具有 ReadWriteMany 访问模式的存储类的名称。此配置使用动态卷置备来自动创建卷。
2. 为 metering 创建部署 NFS 服务器所需的以下资源对象。使用 **oc create -f <file-name>\.yaml** 命令来创建对象 YAML 文件。

- a. 配置 **PersistentVolume** 资源对象：

nfs_persistentvolume.yaml 文件示例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
  labels:
    role: nfs-server
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteMany
  storageClassName: nfs-server 1
  nfs:
    path: "/"
    server: REPLACEME
  persistentVolumeReclaimPolicy: Delete
```

- 1 必须与 **[kind: StorageClass].metadata.name** 字段值完全匹配。

- b. 使用 **nfs-server** 角色配置 **Pod** 资源对象：

nfs_server.yaml 文件示例

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-server
  labels:
    role: nfs-server
spec:
  containers:
  - name: nfs-server
```

```

image: <image_name> ❶
imagePullPolicy: IfNotPresent
ports:
  - name: nfs
    containerPort: 2049
securityContext:
  privileged: true
volumeMounts:
  - mountPath: "/mnt/data"
    name: local
volumes:
  - name: local
    emptyDir: {}

```

❶ 安装您的 NFS 服务器镜像。

c. 使用 **nfs-server** 角色配置 **Service** 资源对象：

nfs_service.yaml 文件示例

```

apiVersion: v1
kind: Service
metadata:
  name: nfs-service
labels:
  role: nfs-server
spec:
  ports:
    - name: 2049-tcp
      port: 2049
      protocol: TCP
      targetPort: 2049
  selector:
    role: nfs-server
  sessionAffinity: None
  type: ClusterIP

```

d. 配置 **StorageClass** 资源对象：

nfs_storageclass.yaml 文件示例

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-server ❶
provisioner: example.com/nfs
parameters:
  archiveOnDelete: "false"
reclaimPolicy: Delete
volumeBindingMode: Immediate

```

❶ 必须与 **[kind: PersistentVolume].spec.storageClassName** 字段值完全匹配。



警告

NFS 存储以及所有相关资源对象的具体配置，会取决于您用于 Metering 存储的 NFS 服务器镜像。

4.4. 配置 HIVE METASTORE

Hive metastore 负责存储所有在 Presto 和 Hive 中创建的数据库表的元数据。metastore 默认会将这些信息存储在附加到 pod 的持久性卷中的本地嵌入式 Derby 数据库中。

通常，Hive metastore 的默认配置适用于小型集群，但用户可能希望通过使用专用 SQL 数据库存储 Hive metastore 数据来提高性能或从集群中移出存储要求。

4.4.1. 配置持久性卷

Hive 默认需要一个持久性卷才可运行。

Hive-metastore-db-data 是默认需要的主持久性卷声明 (PVC)。Hive metastore 使用该 PVC 存储与表相关的元数据，如表名称、列和位置。处理查询时，Presto 和 Hive 服务器可使用 Hive metastore 来查找表元数据。如果使用 MySQL 或 PostgreSQL 作为 Hive metastore 的数据库，则不需要这个要求。

为了进行安装，Hive metastore 要求在存储类中启用动态卷置备功能，一个有正确容量的持久性卷必须预先手动创建，或使用已存在的 MySQL 或 PostgreSQL 数据库。

4.4.1.1. 为 Hive metastore 配置存储类

要为 **hive-metastore-db-data** 持久性卷声明配置和指定存储类，请在 **MeteringConfig** 自定义资源中指定存储类。下面的 **metastore-storage.yaml** 文件中包含一个带有 **class** 字段的 **storage** 部分示例。

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  hive:
    spec:
      metastore:
        storage:
          # Default is null, which means using the default storage class if it exists.
          # If you wish to use a different storage class, specify it here
          # class: "null" 1
          size: "5Gi"

```

1 取消此行的注释，并将 **null** 替换为要使用的存储类的名称。使用 **null** 值会使 metering 使用集群的默认存储类。

4.4.1.2. 配置 Hive metastore 的卷大小

使用以下 **metastore-storage.yaml** 文件作为模板来配置 Hive metastore 的卷大小。

```

apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  hive:
    spec:
      metastore:
        storage:
          # Default is null, which means using the default storage class if it exists.
          # If you wish to use a different storage class, specify it here
          # class: "null"
          size: "5Gi" ❶

```

- ❶ 将 **size** 值替换为您所需容量。示例文件显示“5Gi”。

4.4.2. 为 Hive metastore 使用 MySQL 或 PostgreSQL

默认安装的 metering 会把 Hive 配置为使用名为 Derby 的嵌入式 Java 数据库。该配置不适用于较大环境，它可以被替换为使用 MySQL 或 PostgreSQL 数据库。如果您的部署需要 Hive 使用 MySQL 或 PostgreSQL 数据库，则请使用以下配置示例文件。

有三个配置选项可用来控制 Hive metastore 使用的数据库：**url**、**driver** 和 **secretName**。

使用用户名和密码创建 MySQL 或 Postgres 实例。然后使用 OpenShift CLI (**oc**) 或 YAML 文件创建 secret。您为此 secret 创建的 **secretName** 必须映射到 **MeteringConfig** 对象资源中的 **spec.hive.spec.config.db.secretName** 字段。

流程

1. 使用 OpenShift CLI (**oc**) 或使用 YAML 文件创建 secret:

- 使用以下命令创建 secret：

```

$ oc --namespace openshift-metering create secret generic <YOUR_SECRETNAME> --
from-literal=username=<YOUR_DATABASE_USERNAME> --from-literal=password=
<YOUR_DATABASE_PASSWORD>

```

- 使用 YAML 文件创建 secret。例如：

```

apiVersion: v1
kind: Secret
metadata:
  name: <YOUR_SECRETNAME> ❶
data:
  username: <BASE64_ENCODED_DATABASE_USERNAME> ❷
  password: <BASE64_ENCODED_DATABASE_PASSWORD> ❸

```

- ❶ secret 的名称。
- ❷ base64 编码的数据库用户名。
- ❸ base64 编码的数据库密码。

2. 创建配置文件以将 MySQL 或 PostgreSQL 数据库用于 Hive :

- 要将 MySQL 数据库用于 Hive, 请使用以下示例配置文件。Metering 支持将内部 Hive metastore 配置为使用 MySQL 服务器版本 5.6、5.7 和 8.0。

```
spec:
  hive:
    spec:
      metastore:
        storage:
          create: false
      config:
        db:
          url: "jdbc:mysql://mysql.example.com:3306/hive_metastore" ❶
          driver: "com.mysql.cj.jdbc.Driver"
          secretName: "REPLACEME" ❷
```

**注意**

当将 Metering 配置为使用旧的 MySQL 服务器版本时, 如 5.6 或 5.7 时, 您可能需要在配置内部 Hive metastore 时添加 `enabledTLSProtocols` [JDBC URL 参数](#)。

- ❶ 要使用 TLS v1.2 密码套件, 将 `url` 设置为 `jdbc:mysql://<hostname> → :<port> → /<schema> → ?enabledTLSProtocols=TLSv1.2`。
- ❷ 包含 base64 加密用户名和密码数据库凭证的 secret 名称。

您可使用 `spec.hive.config.url` 来传递其他 JDBC 参数。如需了解更多详细信息, 请参阅 [MySQL Connector/J 8.0 文档](#)。

- 要将 PostgreSQL 数据库用于 Hive, 请使用以下示例配置文件 :

```
spec:
  hive:
    spec:
      metastore:
        storage:
          create: false
      config:
        db:
          url: "jdbc:postgresql://postgresql.example.com:5432/hive_metastore"
          driver: "org.postgresql.Driver"
          username: "REPLACEME"
          password: "REPLACEME"
```

您可使用 `spec.hive.config.url` 来传递其他 JDBC 参数。如需了解更多详细信息, 请参阅 [PostgreSQL JDBC 驱动程序文档](#)。

4.5. 配置 REPORTING OPERATOR

Reporting Operator 负责从 Prometheus 中收集数据，存储指标数据至 Presto 中，对 Presto 运行报告查询，并通过 HTTP API 公开查询结果。配置 Reporting Operator 主要在您的 **MeteringConfig** 自定义资源中进行。

4.5.1. 保护 Prometheus 连接

在 OpenShift Container Platform 上安装 metering 时，使用 <https://prometheus-k8s.openshift-monitoring.svc:9091/> 访问 Prometheus。

为保护与 Prometheus 的连接，默认 metering 安装使用 OpenShift Container Platform 证书颁发机构 (CA)。如果您的 Prometheus 实例使用不同的 CA，您可以通过配置映射注入 CA。您还可以将 Reporting Operator 配置为使用指定的 bearer 令牌与 Prometheus 进行身份验证。

流程

- 通过配置映射注入 Prometheus 实例使用的 CA。例如：

```
spec:
  reporting-operator:
    spec:
      config:
        prometheus:
          certificateAuthority:
            useServiceAccountCA: false
          configMap:
            enabled: true
            create: true
            name: reporting-operator-certificate-authority-config
            filename: "internal-ca.crt"
            value: |
              -----BEGIN CERTIFICATE-----
              (snip)
              -----END CERTIFICATE-----
```

另外，要使系统的 CA 可以支持公共的证书，请将 **ServiceAccountCA** 和 **configMap.enabled** 设置为 **false**。

- 指定 bearer 令牌以通过 Prometheus 进行身份验证。例如：

```
spec:
  reporting-operator:
    spec:
      config:
        prometheus:
          metricsImporter:
            auth:
              useServiceAccountToken: false
              tokenSecret:
                enabled: true
                create: true
                value: "abc-123"
```

4.5.2. 公开 reporting API

在 OpenShift Container Platform 中，默认 metering 安装会自动公开路由，以提供报告 API。它提供以下功能：

- 自动 DNS
- 基于集群 CA 的自动 TLS

此外，默认安装还支持利用 OpenShift 服务来提供证书，以通过 TLS 保护报告 API。OpenShift OAuth 代理被部署为 Reporting Operator 的 sidecar 容器，通过身份验证来保护报告 API。

4.5.2.1. 使用 OpenShift 身份验证

报告 API 默认通过 TLS 和身份验证进行保护。可通过配置 Reporting Operator 来部署包含 Reporting Operator 容器和运行 OpenShift auth-proxy 的 sidecar 容器的 pod 来实现这一目的。

要访问报告 API，Metering Operator 会公开一个路由。路由安装好后，即可运行以下命令获取该路由的主机名。

```
$ METERING_ROUTE_HOSTNAME=$(oc -n openshift-metering get routes metering -o json | jq -r '.status.ingress[].host')
```

下一步，设置身份验证，可使用服务帐户令牌，也可通过用户名和密码进行基础身份验证。

4.5.2.1.1. 使用服务帐户令牌进行身份验证

要使用此法，您需要在报告 Operator 的服务帐户中使用令牌，并将 bearer 令牌传输至以下命令中的身份验证标头中：

```
$ TOKEN=$(oc -n openshift-metering serviceaccounts get-token reporting-operator)
curl -H "Authorization: Bearer $TOKEN" -k
"https://$METERING_ROUTE_HOSTNAME/api/v1/reports/get?name=[Report
Name]&namespace=openshift-metering&format=[Format]"
```

务必要替换上面 URL 中的 **name=[Report Name]** 和 **format=[Format]** 参数。**format** 参数可为 json、csv 或 tabular。

4.5.2.1.2. 使用用户名和密码进行身份验证

Metering 支持使用用户名和密码组合配置基本身份验证，该组合在 httpasswd 文件的内容中指定。默认情况下，会创建一个包含空 httpasswd 数据的 secret。您可通过配置 **reporting-operator.spec.authProxy.htpasswd.data** 和 **reporting-operator.spec.authProxy.htpasswd.createSecret** 键来使用此法。

在 **MeteringConfig** 资源中指定了以上内容后，即可运行以下命令：

```
$ curl -u testuser:password123 -k "https://$METERING_ROUTE_HOSTNAME/api/v1/reports/get?
name=[Report Name]&namespace=openshift-metering&format=[Format]"
```

务必将 **testuser:password123** 替换为有效的用户名和密码组合。

4.5.2.2. 手动配置身份验证

要手动配置 OAuth，或在 Reporting Operator 中禁用 OAuth，必须在 **MeteringConfig** 资源中设置 **spec.tls.enabled: false**。



警告

这也禁用 Reporting Operator、Presto 和 Hive 之间的所有 TLS 和身份验证。您需要自行手动配置这些资源。

身份验证可通过配置以下选项来启用。启用身份验证会将 Reporting Operator pod 配置为将 OpenShift auth-proxy 作为 pod 中的 sidecar 容器运行。这会调整端口，以便报告 API 不会被直接公开，而是通过 auth-proxy sidecar 容器进行代理。

- **reporting-operator.spec.authProxy.enabled**
- **reporting-operator.spec.authProxy.cookie.createSecret**
- **reporting-operator.spec.authProxy.cookie.seed**

您需要将 **reporting-operator.spec.authProxy.enabled** 和 **reporting-operator.spec.authProxy.cookie.createSecret** 设置为 **true**，将 **reporting-operator.spec.authProxy.cookie.seed** 设置为 32 个字符的随机字符串。

您可使用以下命令来生成 32 个字符的随机字符串。

```
$ openssl rand -base64 32 | head -c32; echo.
```

4.5.2.2.1. 令牌身份验证

当以下选项被设置为 **true** 时，将针对报告 REST API 启用使用 bearer 令牌的身份验证。bearer 令牌可由服务帐户或用户提供。

- **reporting-operator.spec.authProxy.subjectAccessReview.enabled**
- **reporting-operator.spec.authProxy.delegateURLs.enabled**

启用身份验证后，必须通过以下任一角色向用于查询该用户或服务帐户报告 API 的 bearer 令牌授予访问权限：

- report-exporter
- reporting-admin
- reporting-viewer
- metering-admin
- metering-viewer

Metering Operator 能够为您创建角色绑定，通过在 **spec.permissions** 部分中指定主体列表来授予这些权限。例如，请参阅以下 **advanced-auth.yaml** 示例配置。

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
```

```

name: "operator-metering"
spec:
permissions:
  # anyone in the "metering-admins" group can create, update, delete, etc any
  # metering.openshift.io resources in the namespace.
  # This also grants permissions to get query report results from the reporting REST API.
meteringAdmins:
- kind: Group
  name: metering-admins
  # Same as above except read only access and for the metering-viewers group.
meteringViewers:
- kind: Group
  name: metering-viewers
  # the default serviceaccount in the namespace "my-custom-ns" can:
  # create, update, delete, etc reports.
  # This also gives permissions query the results from the reporting REST API.
reportingAdmins:
- kind: ServiceAccount
  name: default
  namespace: my-custom-ns
  # anyone in the group reporting-readers can get, list, watch reports, and
  # query report results from the reporting REST API.
reportingViewers:
- kind: Group
  name: reporting-readers
  # anyone in the group cluster-admins can query report results
  # from the reporting REST API. So can the user bob-from-accounting.
reportExporters:
- kind: Group
  name: cluster-admins
- kind: User
  name: bob-from-accounting

reporting-operator:
spec:
  authProxy:
    # htpasswd.data can contain htpasswd file contents for allowing auth
    # using a static list of usernames and their password hashes.
    #
    # username is 'testuser' password is 'password123'
    # generated htpasswdData using: `htpasswd -nb -s testuser password123`
    # htpasswd:
    # data: |
    #   testuser:{SHA}y/2sYAj5yrQIN4TL0YdPdmGNKpc=
    #
    # change REPLACEME to the output of your htpasswd command
  htpasswd:
    data: |
      REPLACEME

```

另外，您还可使用任何具有授予 **reports/export get** 权限规则的角色。这代表，**get** 访问 Reporting Operator 命名空间中的 **Report** 资源的 **export** 子资源。例如：**admin** 和 **cluster-admin**。

默认情况下，Reporting Operator 和 Metering Operator 服务帐户均具有这些权限，其令牌可用于身份验证。

4.5.2.2.2. 使用用户名和密码进行基本身份验证

在进行基础身份验证时，您可以在 `reporting-operator.spec.authProxy.htpasswd.data` 字段中提供用户名和密码。用户名和密码的格式必须与 `htpasswd` 文件中相同。设置完成后，即可使用 HTTP 基础身份验证来提供您的用户名和密码，`htpasswdData` 内容中具有该用户名和密码的对应条目。

4.6. 配置 AWS 账单相关性

metering 可将集群使用量信息与 [AWS 详细账单信息](#) 相关联，并在资源使用量中附上相应金额。对于 EC2 中运行的集群，您可以通过修改以下示例 `aws-billing.yaml` 文件来启用这一功能。

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  openshift-reporting:
    spec:
      awsBillingReportDataSource:
        enabled: true
        # Replace these with where your AWS billing reports are
        # stored in S3.
        bucket: "<your-aws-cost-report-bucket>" ❶
        prefix: "<path/to/report>"
        region: "<your-buckets-region>"

  reporting-operator:
    spec:
      config:
        aws:
          secretName: "<your-aws-secret>" ❷

  presto:
    spec:
      config:
        aws:
          secretName: "<your-aws-secret>" ❸

  hive:
    spec:
      config:
        aws:
          secretName: "<your-aws-secret>" ❹
```

要启用 AWS 账单关联功能，先要确保启用了 AWS 成本和使用量报告。有关更多信息，请参阅 AWS 文档中的 [AWS 成本和使用量报告](#)。

❶ 在您的 AWS 详细账单报告对应位置更新存储桶、前缀和区域。

❷❸❹ 所有 `secretName` 字段均应设置为 metering 命名空间中的 secret 的名称，包含 `data.aws-access-key-id` 和 `data.aws-secret-access-key` 字段中的 AWS 凭证。更多详情请见以下示例 `secret` 文件。

```
apiVersion: v1
```

```

kind: Secret
metadata:
  name: <your-aws-secret>
data:
  aws-access-key-id: "dGVzdAo="
  aws-secret-access-key: "c2VjcmV0Cg=="

```

要将数据存储至 S3，**aws-access-key-id** 和 **aws-secret-access-key** 凭证必须具有存储桶的读取和写入权限。有关授予所需权限的 IAM 策略的示例，请参阅以下 **aws/read-write.json** 文件。

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:HeadBucket",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::operator-metering-data/*", 1
        "arn:aws:s3:::operator-metering-data" 2
      ]
    }
  ]
}
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:HeadBucket",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::operator-metering-data/*", 3
        "arn:aws:s3:::operator-metering-data" 4
      ]
    }
  ]
}

```

1 2 3 4 将 `operator-metering-data` 替换为您的存储桶的名称。

这一步可在安装前或安装后完成。安装后禁用可能会导致 Reporting Operator 中的错误。

第 5 章 REPORTS

5.1. 关于报告

Report 自定义资源提供了一种使用 SQL 查询来管理定期执行性能转换和 Load (ETL) 作业的方法。报告由其他 metering 资源组成，如提供要运行的实际 SQL 查询的 **ReportQuery** 资源，以及定义 **ReportQuery** 和 **Report** 资源可用数据的 **ReportDataSource** 资源。

很多用例均可通过与 metering 一同安装的预定义 **ReportQuery** 和 **ReportDataSource** 资源解决。因此，除非您有未被这些预定义资源覆盖的用例，否则不需要自行定义您的定义。

5.1.1. Reports

Report 自定义资源用于管理报告的执行和状态。metering 会生成通过使用量数据源导出的报告，用于进一步分析和过滤。一个 **Report** 资源代表一个任务，该任务管理数据库表并根据时间表使用新信息来更新报告。报告通过 Reporting Operator HTTP API 来公开表中数据。

带有 **spec.schedule** 字段集的报告会始终运行，并跟踪所收集数据的时间段。这样可确保如果 metering 关闭或不可用，报告将从停止处开始回填数据。如果未设置调度，则报告将在 **reportingStart** 和 **reportingEnd** 处指定的时间运行一次。报告默认会等待 **ReportDataSource** 资源完全导入报告周期内的所有数据。如果报告设置了调度，则会等到当前处理周期内的数据导入完成后才会开始运行。

5.1.1.1. 带有调度的报告示例

以下示例 **Report** 对象包含每个 pod 的 CPU 请求信息，每小时运行一次，每次运行时均会添加最后几小时的数据。

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  reportingStart: "2019-07-01T00:00:00Z"
  schedule:
    period: "hourly"
    hourly:
      minute: 0
      second: 0
```

5.1.1.2. 没有调度的报告示例（运行一次）

以下 **Report** 对象示例包含 7 月份所有 Pod 的 CPU 请求信息。完成后，便不会再运行。

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  reportingStart: "2019-07-01T00:00:00Z"
  reportingEnd: "2019-07-31T00:00:00Z"
```

5.1.1.3. 查询

query 字段命名用于生成报告的 **ReportQuery** 资源。报告查询会控制报告的架构以及结果的处理方式。

query 为**必填**字段。

使用以下命令列出可用的 **ReportQuery** 资源：

```
$ oc -n openshift-metering get reportqueries
```

输出示例

NAME	AGE
cluster-cpu-capacity	23m
cluster-cpu-capacity-raw	23m
cluster-cpu-usage	23m
cluster-cpu-usage-raw	23m
cluster-cpu-utilization	23m
cluster-memory-capacity	23m
cluster-memory-capacity-raw	23m
cluster-memory-usage	23m
cluster-memory-usage-raw	23m
cluster-memory-utilization	23m
cluster-persistentvolumeclaim-request	23m
namespace-cpu-request	23m
namespace-cpu-usage	23m
namespace-cpu-utilization	23m
namespace-memory-request	23m
namespace-memory-usage	23m
namespace-memory-utilization	23m
namespace-persistentvolumeclaim-request	23m
namespace-persistentvolumeclaim-usage	23m
node-cpu-allocatable	23m
node-cpu-allocatable-raw	23m
node-cpu-capacity	23m
node-cpu-capacity-raw	23m
node-cpu-utilization	23m
node-memory-allocatable	23m
node-memory-allocatable-raw	23m
node-memory-capacity	23m
node-memory-capacity-raw	23m
node-memory-utilization	23m
persistentvolumeclaim-capacity	23m
persistentvolumeclaim-capacity-raw	23m
persistentvolumeclaim-phase-raw	23m
persistentvolumeclaim-request	23m
persistentvolumeclaim-request-raw	23m
persistentvolumeclaim-usage	23m
persistentvolumeclaim-usage-raw	23m
persistentvolumeclaim-usage-with-phase-raw	23m
pod-cpu-request	23m
pod-cpu-request-raw	23m
pod-cpu-usage	23m
pod-cpu-usage-raw	23m
pod-memory-request	23m

pod-memory-request-raw	23m
pod-memory-usage	23m
pod-memory-usage-raw	23m

带有 **-raw** 后缀的报告查询会被其他 **ReportQueries** 用于构建更为复杂的查询，而不该直接用于报告。

带有 **namespace-** 前缀的查询会按命名空间聚合 pod CPU 和内存请求，根据资源请求提供命名空间及其总体使用量列表。

带有 **pod-** 前缀的查询与带有 **namespace-** 前缀的查询类似，但会根据 pod 而不是命名空间来聚合信息。这些查询包含 pod 的命名空间和节点。

带有 **node-** 前缀的查询会返回有关各个节点总可用资源的信息。

带有 **aws-** 前缀的查询为 AWS 特定查询。带有 **-aws** 后缀的查询返回的数据与无此后缀的同名查询返回的数据相同，并将使用量与 EC2 计费数据相关联。

aws-ec2-billing-data 报告可供其他查询使用，但不应用作独立报告。**aws-ec2-cluster-cost** 报告根据集群中所含节点提供总成本，以及所报告时间段的成本总额。

使用以下命令将 **ReportQuery** 资源作为 YAML 获取，并检查 **spec.columns** 字段。例如，运行：

```
$ oc -n openshift-metering get reportqueries namespace-memory-request -o yaml
```

输出示例

```
apiVersion: metering.openshift.io/v1
kind: ReportQuery
metadata:
  name: namespace-memory-request
  labels:
    operator-metering: "true"
spec:
  columns:
    - name: period_start
      type: timestamp
      unit: date
    - name: period_end
      type: timestamp
      unit: date
    - name: namespace
      type: varchar
      unit: kubernetes_namespace
    - name: pod_request_memory_byte_seconds
      type: double
      unit: byte_seconds
```

5.1.1.4. 时间表

spec.schedule 配置块用于定义报告的运行时间。**schedule** 部分的主要字段为 **period**。根据 **period** 的值，还可通过 **hourly**、**daily**、**weekly** 和 **monthly** 字段来微调报告运行时间。

例如：如果 **period** 设置为 **weekly**，您则可将 **weekly** 字段添加至 **spec.schedule** 块中。以下示例中报告将于每周三下午 1 点 (13:00) 运行一次。


```
...
schedule:
  period: "weekly"
  weekly:
    dayOfWeek: "wednesday"
    hour: 13
...
```

5.1.1.4.1. 周期

下面列出一些 **schedule.period** 有效值，同时还列出了给定周期内的可设置选项。

- **hourly**
 - **minute**
 - **second**
- **daily**
 - **hour**
 - **minute**
 - **second**
- **weekly**
 - **dayOfWeek**
 - **hour**
 - **minute**
 - **second**
- **monthly**
 - **dayOfMonth**
 - **hour**
 - **minute**
 - **second**
- **cron**
 - **expression**

一般来说，**hour**、**minute**、**second** 字段控制报告在一天中的哪个时间运行，如果为按周或按月运行，则可使用 **dayOfWeek/dayOfMonth** 来控制报告在一周或一个月中的哪一天运行。

以上各个字段均设有有效值区间：

- **hour** 为整数值，介于 0-23 之间。
- **minute** 为整数值，介于 0-59 之间。

- **second** 为整数值，介于 0-59 之间。
- **Dayofweek** 为字符串值，应为一周中的某一天（需要拼写）。
- **dayOfMonth** 为整数值，介于 1-31 之间。

对于 cron 周期，只要为正常 cron 表达式即有效：

- 表达式：`"*/5 * * * *"`

5.1.1.5. reportingStart

要根据现有数据运行报告，可将 **spec.reportingStart** 字段设置为 [RFC3339 时间戳](#)，以告知报告根据其 **schedule** 从 **reportingStart** 而非当前时间开始运行。



注意

将 **spec.reportingStart** 字段设置为指定时间会导致报告 Operator 在 **reportingStart** 时间和当前时间之间的调度中为每个间隔运行多个查询。如果周期短于每日，且 **reportingStart** 在数月之前，则可能会进行数千次查询。如果未设置 **reportingStart**，则报告将在报告创建后的下一个完整 **reportingPeriod** 内运行。

例如，如果您已收集 2019 年 1 月 1 日的数据，且希望在 **Report** 对象中包含这些数据，则可使用以下值创建报告：

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  schedule:
    period: "hourly"
  reportingStart: "2019-01-01T00:00:00Z"
```

5.1.1.6. reportingEnd

要将报告配置为仅运行至指定时间，您可将 **spec.reportingEnd** 字段设置为 [RFC3339 时间戳](#)。此字段值将导致报告生成从开始时间至 **reportingEnd** 周期的数据报告后随即按时间表停止运行。

因为调度很可能与 **reportingEnd** 不一致，所以调度中的最后周期将被缩短至指定的 **reportingEnd** 时间。如果未设置，则报告将会一直运行，或直到报告设置了 **reportingEnd**。

例如，为 7 月创建每周运行一次的报告，使用以下命令：

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  schedule:
```

```

period: "weekly"
reportingStart: "2019-07-01T00:00:00Z"
reportingEnd: "2019-07-31T00:00:00Z"

```

5.1.1.7. 过期

添加 **expiration** 字段，以在调度的 metering 报告上设置保留周期。您可以通过设置 **expiration** 持续时间值来避免手动删除报告。报告保留周期等于 **Report** 对象 **creationDate** 再加上 **expiration** 持续时间。如果没有其他报告或报告查询取决于报告过期，则报告会在保留周期结束时从集群中删除。从集群中删除报告可能需要几分钟。



注意

对于汇总报告，不建议设置 **expiration** 字段。如果报告由其他报告或报告查询依赖，则不会在保留周期结束时删除报告。您可以在 debug 级别查看 **report-operator** 日志，以了解与报告保留决定相关的时间输出。

例如，以下调度的报告会在报告的 **creationDate** 设置时间的 30 分钟后删除：

```

apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  schedule:
    period: "weekly"
    reportingStart: "2020-09-01T00:00:00Z"
    expiration: "30m" 1

```

1 **expiration** 可用的时间单位是 **ns**、**us**（或 **μs**）、**ms**、**s**、**m** 和 **h**。



注意

Report 对象的 **expiration** 保留周期不完全精确，它是按几分钟而不是几纳秒的来工作的。

5.1.1.8. runImmediately

当 **runImmediately** 设置为 **true** 时，报告将立即运行。这个行为可确保立即处理报告并将报告放入队列，而无需额外的调度参数。



注意

当将 **runImmediately** 设为 **true** 时，您必须设置 **reportingEnd** 和 **reportingStart** 值。

5.1.1.9. 输入

Report 对象的 **spec.inputs** 字段可用于覆盖或设置 **ReportQuery** 资源的 **spec.inputs** 字段中定义的值。

spec.inputs 是一个名称值对列表：

```
spec:
  inputs:
  - name: "NamespaceCPUUsageReportName" ❶
    value: "namespace-cpu-usage-hourly" ❷
```

- ❶ **name** 输入值必须存在于 ReportQuery 的 **inputs** 列表中。
- ❷ **Value** 输入值必须为正确的输入 **type**。

5.1.1.10. 卷起 (roll-up) 报告

报告数据存储在数据库中，与指标数据非常相似，因此可用于聚合或汇总报告。卷起报告的一个简单用例是将生成报告所需时间分散到更长时间内。这与通过每月的报告来查询和添加整个月的所有数据。例如，任务可以被分成每日报告，每个报告大约有 1/30 的数据。

自定义汇总报告需要自定义报告查询。**ReportQuery** 资源模板处理器提供了一个 **reportTableName** 功能，它可从 **Report** 对象的 **metadata.name** 中获取必要表名称。

下面是内置查询中的一个片段：

pod-cpu.yaml

```
spec:
...
  inputs:
  - name: ReportingStart
    type: time
  - name: ReportingEnd
    type: time
  - name: NamespaceCPUUsageReportName
    type: Report
  - name: PodCpuUsageRawDataSourceName
    type: ReportDataSource
    default: pod-cpu-usage-raw
...

  query: |
...
  {{- if .Report.Inputs.NamespaceCPUUsageReportName }}
    namespace,
    sum(pod_usage_cpu_core_seconds) as pod_usage_cpu_core_seconds
  FROM {{ .Report.Inputs.NamespaceCPUUsageReportName | reportTableName }}
...

```

aggregated-report.yaml roll-up 报告示例

```
spec:
  query: "namespace-cpu-usage"
  inputs:
  - name: "NamespaceCPUUsageReportName"
    value: "namespace-cpu-usage-hourly"
```

5.1.1.10.1. 报告状态

已调度报告的执行可通过其状态字段进行跟踪。报告准备过程中出现的任何错误均会记录在此处。

Report 对象的 **status** 字段目前包含两个字段：

- **conditions**：是一个状况列表，每个状况均包含 **type**、**status**、**reason** 和 **message** 字段。状况中 **type** 字段的可能值包括 **Running** 和 **Failure**，表明已调度报告的当前状态。**reason** 字段揭示其 **condition** 处于当前状态的原因，**status** 值可为 **true**、**false** 或 **unknown**。**message** 字段提供一条人类可读信息，揭示该状况处于当前状态的原因。有关 **reason** 值的详细信息，请参阅 pkg/apis/metering/v1/util/report_util.go。
- **lastReportTime**：指定 Metering 最后一次收集数据的时间。

5.2. 存储位置

StorageLocation 自定义资源配置 Reporting Operator 存储数据的位置。其中包括从 Prometheus 收集的数据，以及通过生成 **Report** 自定义资源所产生的结果。

如果您要在多个位置（如多个 S3 存储桶或 S3 和 HDFS）存储数据，或者需要访问并非由 metering 在 Hive 和 Presto 中创建的数据库，您只需配置 **StorageLocation** 自定义资源。对于大多数用户来说，这不是必须的。有关配置 **metering** 的文档足以配置所有必要存储组件。

5.2.1. 存储位置示例

以下示例显示了内置的本地存储选项，并配置为使用 Hive。默认情况下，数据存储存储在 Hive 配置为使用存储的位置，如 HDFS、S3 或 **ReadWriteMany** 持久性卷声明（PVC）。

本地存储示例

```
apiVersion: metering.openshift.io/v1
kind: StorageLocation
metadata:
  name: hive
  labels:
    operator-metering: "true"
spec:
  hive: ❶
    databaseName: metering ❷
    unmanagedDatabase: false ❸
```

- ❶ 如果存在 **hive** 部分，则 **StorageLocation** 资源将配置为通过使用 Hive 服务器创建表来存储数据至 Presto 中。只有 **databaseName** 和 **unmanagedDatabase** 是必填字段。
- ❷ Hive 中数据库的名称。
- ❸ 如果为 **true**，则 **StorageLocation** 资源不会被主动管理，Hive 中预期已存在 **databaseName**。如果为 **false**，则 Reporting Operator 将在 Hive 中创建数据库。

以下示例将 AWS S3 存储桶用于存储。在构造要使用的路径时，前缀会附加至存储桶名称中。

远程存储示例

```
apiVersion: metering.openshift.io/v1
kind: StorageLocation
metadata:
```

```

name: example-s3-storage
labels:
  operator-metering: "true"
spec:
  hive:
    databaseName: example_s3_storage
    unmanagedDatabase: false
    location: "s3a://bucket-name/path/within/bucket" 1

```

1 可选：用于数据库的 Presto 和 Hive 的文件系统 URL。可为 **hdfs://** 或 **s3a://** 文件系统 URL。

hive 部分还可指定其他一些可选字段：

- **defaultTableProperties**：包含使用 Hive 创建表的配置选项。
- **FileFormat**：存储在文件系统中的文件格式。如需选项列表和更多详情，请参阅[文件存储格式 Hive 文档](#)。
- **rowFormat**：控制 [Hive 行格式](#)。该字段控制行的序列化和反序列化方式。更多详情请参阅[行格式和 SerDe Hive 文档](#)。

5.2.2. 默认存储位置

如果存在 **storagelocation.metering.openshift.io/is-default** 注解，且该注解在 **StorageLocation** 资源上被设置为 **true**，则该资源将成为默认存储资源。任何组件在存储配置选项中没有指定存储位置时，都将使用默认存储资源。只能有一个默认存储资源。如果多个资源都存在注解，则会记录一个错误，因为 Reporting Operator 无法确定默认值。

默认存储示例

```

apiVersion: metering.openshift.io/v1
kind: StorageLocation
metadata:
  name: example-s3-storage
  labels:
    operator-metering: "true"
  annotations:
    storagelocation.metering.openshift.io/is-default: "true"
spec:
  hive:
    databaseName: example_s3_storage
    unmanagedDatabase: false
    location: "s3a://bucket-name/path/within/bucket"

```

第 6 章 使用 METERING

6.1. 先决条件

- [安装 Metering](#)
- 查看可为 [报告](#) 配置的可用选项及其功能的详细信息。

6.2. 编写报告

编写报告是一种通过 `metering` 来处理和分析数据的方法。

要编写报告，必须在 YAML 文件中定义一个 **Report** 资源，指定所需参数，并在 `openshift-metering` 命名空间中创建报告。

先决条件

- 安装 Metering。

流程

1. 进入 `openshift-metering` 项目：

```
$ oc project openshift-metering
```

2. 创建一个 **Report** 资源作为一个 YAML 文件：

- a. 使用以下内容创建 YAML 文件：

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: namespace-cpu-request-2019 1
  namespace: openshift-metering
spec:
  reportingStart: '2019-01-01T00:00:00Z'
  reportingEnd: '2019-12-30T23:59:59Z'
  query: namespace-cpu-request 2
  runImmediately: true 3
```

- 2** **query** 指定用于生成报告的 **ReportQuery** 资源。您可根据要报告的内容修改此值。如需选项列表，请运行 `oc get reportqueries | grep -v raw`。
- 1** 使用描述性名称来说明报告对 `metadata.name` 的作用。一个好的名称描述了查询，以及所用时间表或周期。
- 3** 如果无论有任何可用数据都运行它时，将 `runImmediately` 设置为 `true`；如果要等到 `reportingEnd` 才运行，则设置为 `false`。

- b. 运行以下命令来创建 **Report** 资源：

```
$ oc create -f <file-name>.yaml
```

输出示例

```
report.metering.openshift.io/namespace-cpu-request-2019 created
```

3. 您可以使用以下命令列出报告及其运行状态：

```
$ oc get reports
```

输出示例

NAME	QUERY	SCHEDULE	RUNNING	FAILED	LAST
REPORT TIME	AGE				
namespace-cpu-request-2019	namespace-cpu-request		Finished		2019-12-30T23:59:59Z 26s

6.3. 查看报告结果

查看报告结果需要查询报告 API 路由，并使用您的 OpenShift Container Platform 凭证对 API 进行身份验证。报告可以以 **JSON**、**CSV** 或 **Tabular** 的格式获得。

先决条件

- 安装 Metering。
- 要访问报告结果，您需为集群管理员，或需要在 **openshift-metering** 命名空间中被授予 **report-exporter** 角色的访问权限。

流程

1. 进入 **openshift-metering** 项目：

```
$ oc project openshift-metering
```

2. 查询报告 API，获取结果：

- a. 为 metering **reporting-api** 路由创建变量，然后获取路由：

```
$ meteringRoute="$(oc get routes metering -o jsonpath='{.spec.host}')
```

```
$ echo "$meteringRoute"
```

- b. 获取请求中会用到的当前用户的令牌：

```
$ token="$(oc whoami -t)"
```

- c. 将 **reportName** 设置为您所创建报告的名称。

```
$ reportName=namespace-cpu-request-2019
```

- d. 将 **reportFormat** 设置为 **csv**、**json** 或 **tabular**，以指定 API 响应的输出格式：


```
$ reportFormat=csv
```

- e. 要获取结果，请使用 **curl** 向报告 API 发出请求，请求获取您的报告：

```
$ curl --insecure -H "Authorization: Bearer ${token}"
"https://${meteringRoute}/api/v1/reports/get?
name=${reportName}&namespace=openshift-metering&format=$reportFormat"
```

带有 **reportName=namespace-cpu-request-2019** 和 **reportFormat=csv** 的输出示例

```
period_start,period_end,namespace,pod_request_cpu_core_seconds
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
apiserver,11745.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-apiserver-
operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
authentication,522.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
authentication-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-cloud-
credential-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-cluster-
machine-approver,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-cluster-
node-tuning-operator,3385.800000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-cluster-
samples-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-cluster-
version,522.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
console,522.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-console-
operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-controller-
manager,7830.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-controller-
manager-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
dns,34372.800000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-dns-
operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
etcd,23490.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-image-
registry,5993.400000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-
ingress,5220.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-ingress-
operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-
apiserver,12528.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-
apiserver-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-
controller-manager,8613.000000
```

2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-controller-manager-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-machine-api,1305.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-machine-config-operator,9637.800000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-metering,19575.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-monitoring,6256.800000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-network-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-sdn,94503.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-service-ca,783.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-service-ca-operator,261.000000

第 7 章 METERING 使用示例

使用以下示例报告开始衡量集群的容量、使用量和利用率。这些示例提供了 metering 提供的各种报告类型，以及预定义查询选择。

7.1. 先决条件

- [安装 metering](#)
- [查看有关报告编写与查看的详细信息。](#)

7.2. 每小时和每日测量集群容量

以下报告演示了如何每小时和每日测量集群容量。每日报告通过汇总每小时报告的结果来实现。

以下报告会每小时测量集群的 CPU 容量。

集群的每小时 CPU 容量示例

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: cluster-cpu-capacity-hourly
spec:
  query: "cluster-cpu-capacity"
  schedule:
    period: "hourly" ①
```

- ① 您可将周期设置为**每日**，获取每日报告，但对于较大数据集，建议使用每小时报告，然后通过汇总每小时报告来获取每日报告。

以下报告通过汇总每小时数据来获取每日报告。

集群的每日 CPU 容量示例

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: cluster-cpu-capacity-daily ①
spec:
  query: "cluster-cpu-capacity" ②
  inputs: ③
  - name: ClusterCpuCapacityReportName
    value: cluster-cpu-capacity-hourly
  schedule:
    period: "daily"
```

- ① 为保持组织有序，如果变更了任何其他值，请务必更改报告的 **name**。
- ② 此外，您还可测量 **cluster-memory-capacity**。记住还要更新相关每小时报告中的查询。
- ③

在 **inputs** 部分将本报告配置为汇总每小时报告。具体来说，**value: cluster-cpu-capacity-hourly** 代表要汇总的每小时报告的名称。

7.3. 通过一次性报告来衡量集群使用量

以下报告从特定的开始日期开始测量集群使用量。这个报告在您保存并应用后仅会运行一次。

集群的 CPU 使用量示例

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: cluster-cpu-usage-2019 1
spec:
  reportingStart: '2019-01-01T00:00:00Z' 2
  reportingEnd: '2019-12-30T23:59:59Z'
  query: cluster-cpu-usage 3
  runImmediately: true 4
```

- 1** 为保持组织有序，如果变更了任何其他值，请务必更改报告的 **name**。
- 2** 对报告进行配置，以使用自 **reportingStart** 时间至 **reportingEnd** 时间之间的数据。
- 3** 通过此处调整查询。您还可使用 **cluster-memory-usage** 查询来测量集群使用量。
- 4** 将报告配置为在保存并应用报告后立即运行。

7.4. 使用 CRON 表达式来测量集群利用率

在配置报告周期时还可使用 cron 表达式。以下报告通过查看每个工作日 9am -5pm 的 CPU 利用率来测量集群使用量。

集群的工作日 CPU 利用率示例

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: cluster-cpu-utilization-weekdays 1
spec:
  query: "cluster-cpu-utilization" 2
  schedule:
    period: "cron"
    expression: 0 0 * * 1-5 3
```

- 1** 为保持组织有序，如果变更了任何其他值，请务必更改报告 **nam**。
- 2** 通过此处调整查询。您还可通过查询 **cluster-memory-utilization** 来测量集群利用率。
- 3** 对于 cron 周期，只要为正常 cron 表达式即有效：

第 8 章 METERING 故障排除与调试

参考以下部分来协助排除和调试与 metering 相关的问题。

除本部分所述信息外，还需查看以下主题：

- [安装 metering 的先决条件。](#)
- [关于配置 metering](#)

8.1. METERING 故障排除

metering 常会遇到 pod 无法启动的问题。Pod 可能会因为缺少资源或其所依赖的资源（如 **StorageClass** 或 **Secret** 资源）不存在而无法启动。

8.1.1. 计算资源不足

安装或运行 metering 时常会遇到计算资源不足的问题。随着集群增长并创建更多报告，Reporting Operator pod 需要更多内存。如果内存用量达到 pod 限制，集群会认为 pod 内存不足（OOM），并以 **OOMKilled** 状态终止它。确保向 metering 分配的资源满足安装先决条件中描述的最低资源要求。



注意

Metering Operator 不根据集群中的负载自动扩展 Reporting Operator。因此，Reporting Operator pod 的 CPU 使用量不会随着集群增长而增加。

要确定资源或调度方面是否存在问题，请按照 Kubernetes 文档 [Managing Compute Resources for Containers](#) 中的故障排除说明操作。

要排除由于缺少计算资源而出现的问题，请在 **openshift-metering** 命名空间中检查以下内容。

先决条件

- 目前处于 **openshift-metering** 命名空间中。运行以下命令来进入 **openshift-metering** 命名空间：

```
$ oc project openshift-metering
```

流程

1. 检查未完成的、显示 **ReportingPeriodUnmetDependencies** 状态的 metering **Report** 资源：

```
$ oc get reports
```

输出示例

```
NAME                                QUERY                                SCHEDULE  RUNNING
FAILED  LAST REPORT TIME  AGE
namespace-cpu-utilization-adhoc-10  namespace-cpu-utilization           Finished
2020-10-31T00:00:00Z  2m38s
namespace-cpu-utilization-adhoc-11  namespace-cpu-utilization
ReportingPeriodUnmetDependencies    2m23s
namespace-memory-utilization-202010 namespace-memory-utilization
```

```
ReportingPeriodUnmetDependencies          26s
namespace-memory-utilization-202011 namespace-memory-utilization
ReportingPeriodUnmetDependencies          14s
```

2. 检查 **NEWEST METRIC** 小于报告结束日期的 **ReportDataSource** 资源：

```
$ oc get reportdatasource
```

输出示例

```
NAME                               EARLIEST METRIC   NEWEST METRIC     IMPORT
START      IMPORT END      LAST IMPORT TIME  AGE
...
node-allocatable-cpu-cores         2020-04-23T09:14:00Z 2020-08-31T10:07:00Z
2020-04-23T09:14:00Z 2020-10-15T17:13:00Z 2020-12-09T12:45:10Z 230d
node-allocatable-memory-bytes      2020-04-23T09:14:00Z 2020-08-30T05:19:00Z
2020-04-23T09:14:00Z 2020-10-14T08:01:00Z 2020-12-09T12:45:12Z 230d
...
pod-usage-memory-bytes             2020-04-23T09:14:00Z 2020-08-24T20:25:00Z
2020-04-23T09:14:00Z 2020-10-09T23:31:00Z 2020-12-09T12:45:12Z 230d
```

3. 检查 **reporting-operator Pod** 资源的健康状态，以检查大量 pod 重启：

```
$ oc get pods -l app=reporting-operator
```

输出示例

```
NAME                               READY STATUS RESTARTS AGE
reporting-operator-84f7c9b7b6-fr697 2/2 Running 542 8d 1
```

- 1** Reporting Operator pod 以更高的率重启。

4. 检查 **reporting-operator Pod** 资源是否有 **OOMKilled** 终止：

```
$ oc describe pod/reporting-operator-84f7c9b7b6-fr697
```

输出示例

```
Name:      reporting-operator-84f7c9b7b6-fr697
Namespace: openshift-metering
Priority:   0
Node:      ip-10-xx-xx-xx.ap-southeast-1.compute.internal/10.xx.xx.xx
...
Ports:     8080/TCP, 6060/TCP, 8082/TCP
Host Ports: 0/TCP, 0/TCP, 0/TCP
State:     Running
  Started:  Thu, 03 Dec 2020 20:59:45 +1000
Last State: Terminated
  Reason:   OOMKilled 1
  Exit Code: 137
  Started:  Thu, 03 Dec 2020 20:38:05 +1000
  Finished: Thu, 03 Dec 2020 20:59:43 +1000
```

- 1 Reporting Operator pod 因 OOM 终止而终止。

增加 reporting-operator pod 内存限制

如果您增加 pod 重启和 OOM 终止事件，您可以检查 Reporting Operator pod 的当前内存限值。增加内存限值可让 Reporting Operator Pod 更新报告数据源。如有必要，将 **MeteringConfig** 资源中的内存限值增加到 25% - 50%。

流程

1. 检查 **reporting-operator Pod** 资源的当前内存限值：

```
$ oc describe pod reporting-operator-67d6f57c56-79mrt
```

输出示例

```
Name:      reporting-operator-67d6f57c56-79mrt
Namespace: openshift-metering
Priority:   0
...
Ports:     8080/TCP, 6060/TCP, 8082/TCP
Host Ports: 0/TCP, 0/TCP, 0/TCP
State:     Running
  Started:  Tue, 08 Dec 2020 14:26:21 +1000
Ready:     True
Restart Count: 0
Limits:
  cpu:      1
  memory:   500Mi 1
Requests:
  cpu:      500m
  memory:   250Mi
Environment:
...
```

- 1 Reporting Operator pod 的当前内存限值。

2. 编辑 **MeteringConfig** 资源以更新内存限制：

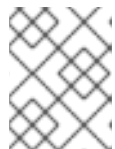
```
$ oc edit meteringconfig/operator-metering
```

MeteringConfig 资源示例

```
kind: MeteringConfig
metadata:
name: operator-metering
namespace: openshift-metering
spec:
reporting-operator:
spec:
resources: 1
  limits:
    cpu: 1
```

```
memory: 750Mi
requests:
cpu: 500m
memory: 500Mi
...
```

1 在 **MeteringConfig** 资源的 **resources** 字段中添加或增加内存限值。



注意

如果在增加内存限制后仍然有大量 OOM 终止事件，这可能代表报告处于待处理状态。

8.1.2. 未配置 StorageClass 资源

Metering 要求为动态置备配置默认 **StorageClass** 资源。

有关如何检查是否为集群配置了 **StorageClass** 资源、如何设置默认值以及如何配置 metering 以使用默认存储类的信息，请参阅有关配置 metering 的文档。

8.1.3. 未正确配置 secret

metering 常会遇到在配置持久性存储时所提供的 secret 不正确的问题。请务必查看示例配置文件并根据您的存储提供程序指南创建 secret。

8.2. METERING 调试

当直接与各种组件交互时，可以更容易地调试 metering。以下部分详细介绍如何连接和查询 Presto 和 Hive 以及如何查看 HDFS 组件的仪表盘。



注意

本部分所有命令均假设您已通过 OperatorHub 在 **openshift-metering** 命名空间中安装了 metering。

8.2.1. 获取报告 Operator 日志

使用以下命令跟踪 **reporting-operator** 日志：

```
$ oc -n openshift-metering logs -f "$(oc -n openshift-metering get pods -l app=reporting-operator -o name | cut -c 5-)" -c reporting-operator
```

8.2.2. 使用 presto-cli 查询 Presto

以下命令将打开交互式 presto-cli 会话，您可以通过该会话查询 Presto。该会话与 Presto 在相同容器中运行，并会启动一个额外 Java 实例，可为 pod 创建内存限值。如果出现这种情况，您需要提高 Presto pod 的内存请求和限值。

Presto 默认配置为使用 TLS 进行通信。您必须运行以下命令来运行 Presto 查询：

```
$ oc -n openshift-metering exec -it "$(oc -n openshift-metering get pods -l
```



```
app=presto,presto=coordinator -o name | cut -d/ -f2)" \
-- /usr/local/bin/presto-cli --server https://presto:8080 --catalog hive --schema default --user root --
keystore-path /opt/presto/tls/keystore.pem
```

运行完该命令后，系统会提示您运行查询。使用 **show tables from metering;** 查询来查看表列表：

```
$ presto:default> show tables from metering;
```

输出示例

```
Table

datasource_your_namespace_cluster_cpu_capacity_raw
datasource_your_namespace_cluster_cpu_usage_raw
datasource_your_namespace_cluster_memory_capacity_raw
datasource_your_namespace_cluster_memory_usage_raw
datasource_your_namespace_node_allocatable_cpu_cores
datasource_your_namespace_node_allocatable_memory_bytes
datasource_your_namespace_node_capacity_cpu_cores
datasource_your_namespace_node_capacity_memory_bytes
datasource_your_namespace_node_cpu_allocatable_raw
datasource_your_namespace_node_cpu_capacity_raw
datasource_your_namespace_node_memory_allocatable_raw
datasource_your_namespace_node_memory_capacity_raw
datasource_your_namespace_persistentvolumeclaim_capacity_bytes
datasource_your_namespace_persistentvolumeclaim_capacity_raw
datasource_your_namespace_persistentvolumeclaim_phase
datasource_your_namespace_persistentvolumeclaim_phase_raw
datasource_your_namespace_persistentvolumeclaim_request_bytes
datasource_your_namespace_persistentvolumeclaim_request_raw
datasource_your_namespace_persistentvolumeclaim_usage_bytes
datasource_your_namespace_persistentvolumeclaim_usage_raw
datasource_your_namespace_persistentvolumeclaim_usage_with_phase_raw
datasource_your_namespace_pod_cpu_request_raw
datasource_your_namespace_pod_cpu_usage_raw
datasource_your_namespace_pod_limit_cpu_cores
datasource_your_namespace_pod_limit_memory_bytes
datasource_your_namespace_pod_memory_request_raw
datasource_your_namespace_pod_memory_usage_raw
datasource_your_namespace_pod_persistentvolumeclaim_request_info
datasource_your_namespace_pod_request_cpu_cores
datasource_your_namespace_pod_request_memory_bytes
datasource_your_namespace_pod_usage_cpu_cores
datasource_your_namespace_pod_usage_memory_bytes
(32 rows)

Query 20190503_175727_00107_3venm, FINISHED, 1 node
Splits: 19 total, 19 done (100.00%)
0:02 [32 rows, 2.23KB] [19 rows/s, 1.37KB/s]

presto:default>
```

8.2.3. 使用 beeling 来查询 Hive

以下命令将打开交互式 beeline 会话，您可以通过该会话查询 Hive。该会话与 Hive 在相同容器中运行，并会启动一个额外 Java 实例，可为 pod 创建内存限值。如果出现这种情况，您需要提高 Hive pod 的内存请求和限值。

```
$ oc -n openshift-metering exec -it $(oc -n openshift-metering get pods -l app=hive,hive=server -o name | cut -d/ -f2) \
-c hiveserver2 -- beeline -u 'jdbc:hive2://127.0.0.1:10000/default;auth=noSasl'
```

运行完该命令后，系统会提示您运行查询。使用 **show tables** 查询来查看表列表：

```
$ 0: jdbc:hive2://127.0.0.1:10000/default> show tables from metering;
```

输出示例

```
+-----+
|          tab_name          |
+-----+
| datasource_your_namespace_cluster_cpu_capacity_raw |
| datasource_your_namespace_cluster_cpu_usage_raw |
| datasource_your_namespace_cluster_memory_capacity_raw |
| datasource_your_namespace_cluster_memory_usage_raw |
| datasource_your_namespace_node_allocatable_cpu_cores |
| datasource_your_namespace_node_allocatable_memory_bytes |
| datasource_your_namespace_node_capacity_cpu_cores |
| datasource_your_namespace_node_capacity_memory_bytes |
| datasource_your_namespace_node_cpu_allocatable_raw |
| datasource_your_namespace_node_cpu_capacity_raw |
| datasource_your_namespace_node_memory_allocatable_raw |
| datasource_your_namespace_node_memory_capacity_raw |
| datasource_your_namespace_persistentvolumeclaim_capacity_bytes |
| datasource_your_namespace_persistentvolumeclaim_capacity_raw |
| datasource_your_namespace_persistentvolumeclaim_phase |
| datasource_your_namespace_persistentvolumeclaim_phase_raw |
| datasource_your_namespace_persistentvolumeclaim_request_bytes |
| datasource_your_namespace_persistentvolumeclaim_request_raw |
| datasource_your_namespace_persistentvolumeclaim_usage_bytes |
| datasource_your_namespace_persistentvolumeclaim_usage_raw |
| datasource_your_namespace_persistentvolumeclaim_usage_with_phase_raw |
| datasource_your_namespace_pod_cpu_request_raw |
| datasource_your_namespace_pod_cpu_usage_raw |
| datasource_your_namespace_pod_limit_cpu_cores |
| datasource_your_namespace_pod_limit_memory_bytes |
| datasource_your_namespace_pod_memory_request_raw |
| datasource_your_namespace_pod_memory_usage_raw |
| datasource_your_namespace_pod_persistentvolumeclaim_request_info |
| datasource_your_namespace_pod_request_cpu_cores |
| datasource_your_namespace_pod_request_memory_bytes |
| datasource_your_namespace_pod_usage_cpu_cores |
| datasource_your_namespace_pod_usage_memory_bytes |
+-----+
32 rows selected (13.101 seconds)
0: jdbc:hive2://127.0.0.1:10000/default>
```

8.2.4. 将端口转发到 Hive Web UI

运行以下命令来将端口转发到 Hive Web UI:

```
$ oc -n openshift-metering port-forward hive-server-0 10002
```

您现在可从浏览器窗口中打开 <http://127.0.0.1:10002> 以进入 Hive Web 界面。

8.2.5. 端口转发至 HDFS

运行以下命令来将 HDFS 名称节点转发到端口：

```
$ oc -n openshift-metering port-forward hdfs-namenode-0 9870
```

您现在可从浏览器窗口中打开 <http://127.0.0.1:9870> 以进入 HDFS Web 界面。

运行以下命令将端口转发到第一个 HDFS 数据节点：

```
$ oc -n openshift-metering port-forward hdfs-datanode-0 9864 1
```

1 要检查其他数据节点，将 **hdfs-datanode-0** 替换为您要查看信息的 pod。

8.2.6. Metering Ansible Operator

metering 会使用 Ansible Operator 来监控和协调集群环境中的资源。调试失败的 metering 安装时，查看 Ansible 日志或 **MeteringConfig** 自定义资源的状态会很有帮助。

8.2.6.1. 访问 Ansible 日志

在默认安装中，metering Operator 会被部署为 pod。这种情况下，可进入 pod 检查 Ansible 容器日志：

```
$ oc -n openshift-metering logs $(oc -n openshift-metering get pods -l app=metering-operator -o name | cut -d/ -f2) -c ansible
```

或者，还可查看 Operator 容器的日志（将 **-c ansible** 替换为 **-c operator**）以获取压缩输出。

8.2.6.2. 检查 MeteringConfig 状态

查看 **MeteringConfig** 自定义资源的 **.status** 字段有助于调试任何最新故障。以下命令显示类型为 **Invalid** 的状态消息：

```
$ oc -n openshift-metering get meteringconfig operator-metering -o=jsonpath='{.status.conditions[?(@.type=="Invalid")].message}'
```

8.2.6.3. 检查 MeteringConfig 事件

检查 Metering Operator 正在生成的事件。这对安装或升级调试任何资源失败非常有用。按最后一次时间戳对事件排序：

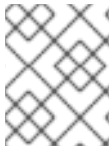
```
$ oc -n openshift-metering get events --field-selector involvedObject.kind=MeteringConfig --sort-by='.lastTimestamp'
```

输出显示 **MeteringConfig** 资源的最新更改：

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
4m40s	Normal	Validating	meteringconfig/operator-metering	Validating the user-provided configuration
4m30s	Normal	Started	meteringconfig/operator-metering	Configuring storage for the metering-ansible-operator
4m26s	Normal	Started	meteringconfig/operator-metering	Configuring TLS for the metering-ansible-operator
3m58s	Normal	Started	meteringconfig/operator-metering	Configuring reporting for the metering-ansible-operator
3m53s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling metering resources
3m47s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling monitoring resources
3m41s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling HDFS resources
3m23s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling Hive resources
2m59s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling Presto resources
2m35s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling reporting-operator resources
2m14s	Normal	Reconciling	meteringconfig/operator-metering	Reconciling reporting resources

第 9 章 卸载 METERING

您可以从 OpenShift Container Platform 集群中删除 metering。



注意

Metering 不会管理或删除 Amazon S3 存储桶数据。卸载 metering 后，必须手动清理用于存储 metering 数据的 S3 存储桶。

9.1. 从集群中移除 METERING OPERATOR

按照从集群中删除 Operator 的文档[从集群中删除 Metering Operator](#)。



注意

从集群中移除 Metering Operator 不删除其自定义资源定义或受管资源。如需了解删除所有剩余的 metering 组件的步骤,请参阅以下部分中有关[卸载 metering 命名空间](#)和[卸载 metering 自定义资源定义](#)的内容。

9.2. 卸载 METERING 命名空间

通过删除 **MeteringConfig** 资源并删除 **openshift-metering** 命名空间来卸载 metering 命名空间（如 **openshift-metering** 命名空间）。

先决条件

- Metering Operator 已从集群中移除。

流程

1. 删除 Metering Operator 创建的所有资源：

```
$ oc --namespace openshift-metering delete meteringconfig --all
```

2. 完成上一步后，验证 **openshift-metering** 命名空间中的所有 pod 是否已删除，或处于终止状态：

```
$ oc --namespace openshift-metering get pods
```

3. 删除 **openshift-metering** 命名空间：

```
$ oc delete namespace openshift-metering
```

9.3. 卸载 METERING 自定义资源定义

在卸载 Metering Operator 并删除 **openshift-metering** 命名空间后，metering CustomResourceDefinitions (CRD) 会保留在集群中。



重要

删除 metering CRD 会破坏集群中其他命名空间中已安装的任何其他 metering。在继续操作前，确保没有其他 metering。

先决条件

- **openshift-metering** 命名空间中的 **MeteringConfig** 自定义资源已被删除。
- **openshift-metering** 命名空间已被删除。

流程

- 删除剩余的 metering CRD:

```
$ oc get crd -o name | grep "metering.openshift.io" | xargs oc delete
```