OpenShift Container Platform 4.5

Metering

Configuring and using Metering in OpenShift Container Platform

Last Updated: 2021-07-26

OpenShift Container Platform 4.5 Metering

Configuring and using Metering in OpenShift Container Platform

Legal Notice

Abstract

This document provides instructions for configuring and using metering in OpenShift Container Platform.

Table of Contents

CHAPTER 1. ABOUT METERING **4**
1.1. METERING OVERVIEW 4
1.1.1. Metering resources 4

CHAPTER 2. INSTALLING METERING **5**
2.1. PREREQUISITES 5
2.2. INSTALLING THE METERING OPERATOR 5
2.2.1. Installing metering using the web console 5
2.2.2. Installing metering using the CLI 6
2.3. INSTALLING THE METERING STACK 8
2.4. PREREQUISITES 8
2.5. VERIFYING THE METERING INSTALLATION 8
2.6. ADDITIONAL RESOURCES 10

CHAPTER 3. UPGRADING METERING **11**
3.1. PREREQUISITES 11

CHAPTER 4. CONFIGURING METERING **14**
4.1. ABOUT CONFIGURING METERING 14
4.2. COMMON CONFIGURATION OPTIONS 14
4.2.1. Resource requests and limits 14
4.2.2. Node selectors 15
4.3. CONFIGURING PERSISTENT STORAGE 17
4.3.1. Storing data in Amazon S3 17
4.3.2. Storing data in S3-compatible storage 20
4.3.3. Storing data in Microsoft Azure 21
4.3.4. Storing data in Google Cloud Storage 21
4.3.5. Storing data in shared volumes 22
4.4. CONFIGURING THE HIVE METASTORE 25
4.4.1. Configuring persistent volumes 25
4.4.1.1. Configuring the storage class for the Hive metastore 26
4.4.1.2. Configuring the volume size for the Hive metastore 26
4.4.2. Use MySQL or PostgreSQL for the Hive metastore 27
4.5. CONFIGURING THE REPORTING OPERATOR 27
4.5.1. Securing a Prometheus connection 27
4.5.2. Exposing the reporting API 28
4.5.2.1. Using OpenShift Authentication 29
4.5.2.1.1. Authenticate using a service account token 29
4.5.2.1.2. Authenticate using a username and password 29
4.5.2.2. Manually Configuring Authentication 29
4.5.2.2.1. Token authentication 30
4.5.2.2.2. Basic authentication with a username and password 32
4.6. CONFIGURE AWS BILLING CORRELATION 32

CHAPTER 5. REPORTS **35**
5.1. ABOUT REPORTS 35
5.1.1. Reports 35
5.1.1.1. Example report with a schedule 35
5.1.1.2. Example report without a schedule (run-once) 35
5.1.1.3. query 36
5.1.1.4. schedule 38
5.1.1.4.1. period 38

5.1.1.5. reportingStart 39
5.1.1.6. reportingEnd 40
5.1.1.7. runImmediately 40
5.1.1.8. inputs 40
5.1.1.9. Roll-up reports 40
5.1.1.9.1. Report status 41
5.2. STORAGE LOCATIONS 42
5.2.1. Storage location examples 42
5.2.2. Default storage location 43

CHAPTER 6. USING METERING 44
6.1. PREREQUISITES 44
6.2. WRITING REPORTS 44
6.3. VIEWING REPORT RESULTS 45

CHAPTER 7. EXAMPLES OF USING METERING 48
7.1. PREREQUISITES 48
7.2. MEASURE CLUSTER CAPACITY HOURLY AND DAILY 48
7.3. MEASURE CLUSTER USAGE WITH A ONE-TIME REPORT 49
7.4. MEASURE CLUSTER UTILIZATION USING CRON EXPRESSIONS 49

CHAPTER 8. TROUBLESHOOTING AND DEBUGGING METERING 51
8.1. TROUBLESHOOTING METERING 51
8.1.1. Not enough compute resources 51
Increasing the reporting-operator pod memory limit 53
8.1.2. StorageClass resource not configured 54
8.1.3. Secret not configured correctly 54
8.2. DEBUGGING METERING 54
8.2.1. Get reporting operator logs 54
8.2.2. Query Presto using presto-cli 55
8.2.3. Query Hive using beeline 56
8.2.4. Port-forward to the Hive web UI 57
8.2.5. Port-forward to HDFS 57
8.2.6. Metering Ansible Operator 57
8.2.6.1. Accessing Ansible logs 57
8.2.6.2. Checking the MeteringConfig Status 58
8.2.6.3. Checking MeteringConfig Events 58

CHAPTER 9. UNINSTALLING METERING 59
9.1. REMOVING THE METERING OPERATOR FROM YOUR CLUSTER 59
9.2. UNINSTALLING A METERING NAMESPACE 59
9.3. UNINSTALLING METERING CUSTOM RESOURCE DEFINITIONS 59

CHAPTER 1. ABOUT METERING

1.1. METERING OVERVIEW

Metering is a general purpose data analysis tool that enables you to write reports to process data from different data sources. As a cluster administrator, you can use metering to analyze what is happening in your cluster. You can either write your own, or use predefined SQL queries to define how you want to process data from the different data sources you have available.

Metering focuses primarily on in-cluster metric data using Prometheus as a default data source, enabling users of metering to do reporting on pods, namespaces, and most other Kubernetes resources.

You can install metering on OpenShift Container Platform 4.x clusters and above.

1.1.1. Metering resources

Metering has many resources which can be used to manage the deployment and installation of metering, as well as the reporting functionality metering provides.

Metering is managed using the following custom resource definitions (CRDs):

MeteringConfig	Configures the metering stack for deployment. Contains customizations and configuration options to control each component that makes up the metering stack.
Report	Controls what query to use, when, and how often the query should be run, and where to store the results.
ReportQuery	Contains the SQL queries used to perform analysis on the data contained within **ReportDataSource** resources.
ReportDataSource	Controls the data available to **ReportQuery** and **Report** resources. Allows configuring access to different databases for use within metering.

CHAPTER 2. INSTALLING METERING

Review the following sections before installing metering into your cluster.

To get started installing metering, first install the Metering Operator from OperatorHub. Next, configure your instance of metering by creating a **MeteringConfig** custom resource (CR). Installing the Metering Operator creates a default **MeteringConfig** resource that you can modify using the examples in the documentation. After creating your **MeteringConfig** resource, install the metering stack. Last, verify your installation.

2.1. PREREQUISITES

Metering requires the following components:

- A **StorageClass** resource for dynamic volume provisioning. Metering supports a number of different storage solutions.
- 4GB memory and 4 CPU cores available cluster capacity and at least one node with 2 CPU cores and 2GB memory capacity available.
- The minimum resources needed for the largest single pod installed by metering are 2GB of memory and 2 CPU cores.
  - Memory and CPU consumption may often be lower, but will spike when running reports, or collecting data for larger clusters.

2.2. INSTALLING THE METERING OPERATOR

You can install metering by deploying the Metering Operator. The Metering Operator creates and manages the components of the metering stack.

NOTE

You cannot create a project starting with **openshift-** using the web console or by using the **oc new-project** command in the CLI.

NOTE

If the Metering Operator is installed using a namespace other than **openshift-metering**, the metering reports are only viewable using the CLI. It is strongly suggested throughout the installation steps to use the **openshift-metering** namespace.

2.2.1. Installing metering using the web console

You can use the OpenShift Container Platform web console to install the Metering Operator.

Procedure

1. Create a namespace object YAML file for the Metering Operator with the **oc create -f <file-name>.yaml** command. You must use the CLI to create the namespace. For example, **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) It is strongly recommended to deploy metering in the **openshift-metering** namespace.

(2) Include this annotation before configuring specific node selectors for the operand pods.

2. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**. Filter for **metering** to find the Metering Operator.

3. Click the **Metering** card, review the package description, and then click **Install**.

4. Select an **Update Channel**, **Installation Mode**, and **Approval Strategy**.

5. Click **Install**.

6. Verify that the Metering Operator is installed by switching to the **Operators** → **Installed Operators** page. The Metering Operator has a **Status** of **Succeeded** when the installation is complete.

> **NOTE**
>
> It might take several minutes for the Metering Operator to appear.

7. Click **Metering** on the **Installed Operators** page for Operator **Details**. From the **Details** page you can create different resources related to metering.

To complete the metering installation, create a **MeteringConfig** resource to configure metering and install the components of the metering stack.

2.2.2. Installing metering using the CLI

You can use the OpenShift Container Platform CLI to install the Metering Operator.

Procedure

1. Create a **Namespace** object YAML file for the Metering Operator. You must use the CLI to create the namespace. For example, **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) It is strongly recommended to deploy metering in the **openshift-metering** namespace.

2 Include this annotation before configuring specific node selectors for the operand pods.

2. Create the **Namespace** object:

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

For example:

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

3. Create the **OperatorGroup** object YAML file. For example, **metering-og**:

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

1 The name is arbitrary.

2 Specify the **openshift-metering** namespace.

4. Create a **Subscription** object YAML file to subscribe a namespace to the Metering Operator. This object targets the most recently released version in the **redhat-operators** catalog source. For example, **metering-sub.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metering-ocp 1
  namespace: openshift-metering 2
spec:
  channel: "4.5" 3
  source: "redhat-operators" 4
  sourceNamespace: "openshift-marketplace"
  name: "metering-ocp"
  installPlanApproval: "Automatic" 5
```

1 The name is arbitrary.

2 You must specify the **openshift-metering** namespace.

3 Specify 4.5 as the channel.

4 Specify the **redhat-operators** catalog source, which contains the **metering-ocp** package manifests. If your OpenShift Container Platform is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object you created when you configured the Operator LifeCycle Manager (OLM).

5 Specify "Automatic" install plan approval.

2.3. INSTALLING THE METERING STACK

After adding the Metering Operator to your cluster you can install the components of metering by installing the metering stack.

2.4. PREREQUISITES

- Review the configuration options
- Create a **MeteringConfig** resource. You can begin the following process to generate a default **MeteringConfig** resource, then use the examples in the documentation to modify this default file for your specific installation. Review the following topics to create your **MeteringConfig** resource:
  - For configuration options, review About configuring metering.
  - At a minimum, you need to configure persistent storage and configure the Hive metastore.

> **IMPORTANT**
>
> There can only be one **MeteringConfig** resource in the **openshift-metering** namespace. Any other configuration is not supported.

Procedure

1. From the web console, ensure you are on the **Operator Details** page for the Metering Operator in the **openshift-metering** project. You can navigate to this page by clicking **Operators → Installed Operators**, then selecting the Metering Operator.
2. Under **Provided APIs**, click **Create Instance** on the Metering Configuration card. This opens a YAML editor with the default **MeteringConfig** resource file where you can define your configuration.

   > **NOTE**
   >
   > For example configuration files and all supported configuration options, review the configuring metering documentation.

3. Enter your **MeteringConfig** resource into the YAML editor and click **Create**.

The **MeteringConfig** resource begins to create the necessary resources for your metering stack. You can now move on to verifying your installation.

2.5. VERIFYING THE METERING INSTALLATION

You can verify the metering installation by performing any of the following checks:

- Check the Metering Operator **ClusterServiceVersion** (CSV) resource for the metering version. This can be done through either the web console or CLI.

Procedure (UI)

1. Navigate to **Operators → Installed Operators** in the **openshift-metering** namespace.
2. Click **Metering Operator**.
3. Click **Subscription** for **Subscription Details**.
4. Check the **Installed Version**.

Procedure (CLI)

- Check the Metering Operator CSV in the **openshift-metering** namespace:

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

Example output

```
NAME                                          DISPLAY                  VERSION                  REPLACES
PHASE
elasticsearch-operator.4.5.0-202006231303.p0   Elasticsearch Operator   4.5.0-
202006231303.p0              Succeeded
metering-operator.v4.5.0                       Metering                 4.5.0
Succeeded
```

- Check that all required pods in the **openshift-metering** namespace are created. This can be done through either the web console or CLI.

NOTE

Many pods rely on other components to function before they themselves can be considered ready. Some pods may restart if other pods take too long to start. This is to be expected during the Metering Operator installation.

Procedure (UI)

- Navigate to **Workloads → Pods** in the metering namespace and verify that pods are being created. This can take several minutes after installing the metering stack.

Procedure (CLI)

- Check that all required pods in the **openshift-metering** namespace are created:

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

Example output

```
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
```

- Verify that the **ReportDataSource** resources are beginning to import data, indicated by a valid timestamp in the **EARLIEST METRIC** column. This might take several minutes. Filter out the "-raw" **ReportDataSource** resources, which do not import data:

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

Example output

```
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
```

After all pods are ready and you have verified that data is being imported, you can begin using metering to collect data and report on your cluster.

2.6. ADDITIONAL RESOURCES

- For more information on configuration steps and available storage platforms, see Configuring persistent storage.
- For the steps to configure Hive, see Configuring the Hive metastore.

CHAPTER 3. UPGRADING METERING

You can upgrade metering to 4.5 by updating the Metering Operator subscription.

3.1. PREREQUISITES

- The cluster is updated to 4.5.
- The Metering Operator is installed from OperatorHub.

> **NOTE**
>
> You must upgrade the Metering Operator to 4.5 manually. Metering does not upgrade automatically if you selected the "Automatic" **Approval Strategy** in a previous installation.

- The MeteringConfig custom resource is configured.
- The metering stack is installed.
- Ensure that metering status is healthy by checking that all pods are ready.

> **IMPORTANT**
>
> Potential data loss can occur if you modify your metering storage configuration after installing or upgrading metering.

Procedure

1. Click **Operators → Installed Operators** from the web console.
2. Select the **openshift-metering** project.
3. Click **Metering Operator**.
4. Click **Subscription → Channel**.
5. In the **Change Subscription Update Channel** window, select **4.5** and click **Save**.

   > **NOTE**
   >
   > Wait several seconds to allow the subscription to update before proceeding to the next step.

6. Click **Operators → Installed Operators**.
   The Metering Operator is shown as 4.5. For example:

   ```
   Metering
   4.5.0-202007012112.p0 provided by Red Hat, Inc
   ```

Verification

You can verify the metering upgrade by performing any of the following checks:

- Check the Metering Operator cluster service version (CSV) for the new metering version. This can be done through either the web console or CLI.

  Procedure (UI)

  1. Navigate to **Operators → Installed Operators** in the metering namespace.
  2. Click **Metering Operator**.
  3. Click **Subscription** for **Subscription Details**.
  4. Check the **Installed Version** for the upgraded metering version. The **Starting Version** shows the metering version prior to upgrading.

  Procedure (CLI)

  - Check the Metering Operator CSV:

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

    Example output for metering upgrade from 4.4 to 4.5

    ```
    NAME                                  DISPLAY           VERSION               REPLACES
    PHASE
    metering-operator.4.5.0-202007012112.p0     Metering              4.5.0-202007012112.p0
    metering-operator.4.4.0-202005252114   Succeeded
    ```

- Check that all required pods in the **openshift-metering** namespace are created. This can be done through either the web console or CLI.

  > **NOTE**
  >
  > Many pods rely on other components to function before they themselves can be considered ready. Some pods may restart if other pods take too long to start. This is to be expected during the Metering Operator upgrade.

  Procedure (UI)

  - Navigate to **Workloads → Pods** in the metering namespace and verify that pods are being created. This can take several minutes after upgrading the metering stack.

  Procedure (CLI)

  - Check that all required pods in the **openshift-metering** namespace are created:

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

    Example output

    ```
    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
```

- Verify that the **ReportDataSource** resources are importing new data, indicated by a valid timestamp in the **NEWEST METRIC** column. This might take several minutes. Filter out the "-raw" **ReportDataSource** resources, which do not import data:

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

Timestamps in the **NEWEST METRIC** column indicate that **ReportDataSource** resources are beginning to import new data.

Example output

```
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
```

After all pods are ready and you have verified that new data is being imported, metering continues to collect data and report on your cluster. Review a previously scheduled report or create a run-once metering report to confirm the metering upgrade.

CHAPTER 4. CONFIGURING METERING

4.1. ABOUT CONFIGURING METERING

The **MeteringConfig** custom resource specifies all the configuration details for your metering installation. When you first install the metering stack, a default **MeteringConfig** custom resource is generated. Use the examples in the documentation to modify this default file. Keep in mind the following key points:

- At a minimum, you need to configure persistent storage and configure the Hive metastore.
- Most default configuration settings work, but larger deployments or highly customized deployments should review all configuration options carefully.
- Some configuration options can not be modified after installation.

For configuration options that can be modified after installation, make the changes in your **MeteringConfig** custom resource and reapply the file.

4.2. COMMON CONFIGURATION OPTIONS

4.2.1. Resource requests and limits

You can adjust the CPU, memory, or storage resource requests and/or limits for pods and volumes. The **default-resource-limits.yaml** below provides an example of setting resource request and limits for each component.

```
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. Node selectors

You can run the metering components on specific sets of nodes. Set the **nodeSelector** on a metering component to control where the component is scheduled. The **node-selectors.yaml** file below provides an example of setting node selectors for each component.

> **NOTE**
>
> Add the **openshift.io/node-selector: ""** namespace annotation to the metering namespace YAML file before configuring specific node selectors for the operand pods. Specify **""** as the annotation value.

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  reporting-operator:
    spec:
      nodeSelector:
        "node-role.kubernetes.io/infra": "" (1)

  presto:
```

```
    spec:
      coordinator:
        nodeSelector:
          "node-role.kubernetes.io/infra": "" (2)
      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 Add a **nodeSelector** parameter with the appropriate value to the component you want to move. You can use a **nodeSelector** in the format shown or use key-value pairs, based on the value specified for the node.

> **NOTE**
>
> Add the **openshift.io/node-selector: ""** namespace annotation to the metering namespace YAML file before configuring specific node selectors for the operand pods. When the **openshift.io/node-selector** annotation is set on the project, the value is used in preference to the value of the **spec.defaultNodeSelector** field in the cluster-wide **Scheduler** object.

Verification

You can verify the metering node selectors by performing any of the following checks:

- Verify that all pods for metering are correctly scheduled on the IP of the node that is configured in the **MeteringConfig** custom resource:
  1. Check all pods in the **openshift-metering** namespace:

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

     The output shows the **NODE** and corresponding **IP** for each pod running in the **openshift-metering** namespace.

     #### Example output

     ```
     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>
```

2. Compare the nodes in the **openshift-metering** namespace to each node **NAME** in your cluster:

```
$ oc get nodes
```

Example output

```
NAME                                         STATUS   ROLES    AGE   VERSION
ip-10-0-147-106.us-east-2.compute.internal   Ready    master   14h   v1.18.3+6025c28
ip-10-0-150-175.us-east-2.compute.internal   Ready    worker   14h   v1.18.3+6025c28
ip-10-0-175-23.us-east-2.compute.internal    Ready    master   14h   v1.18.3+6025c28
ip-10-0-189-6.us-east-2.compute.internal     Ready    worker   14h   v1.18.3+6025c28
ip-10-0-205-158.us-east-2.compute.internal   Ready    master   14h   v1.18.3+6025c28
ip-10-0-210-167.us-east-2.compute.internal   Ready    worker   14h   v1.18.3+6025c28
```

- Verify that the node selector configuration in the **MeteringConfig** custom resource does not interfere with the cluster-wide node selector configuration such that no metering operand pods are scheduled.
  - Check the cluster-wide **Scheduler** object for the **spec.defaultNodeSelector** field, which shows where pods are scheduled by default:

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

4.3. CONFIGURING PERSISTENT STORAGE

Metering requires persistent storage to persist data collected by the Metering Operator and to store the results of reports. A number of different storage providers and storage formats are supported. Select your storage provider and modify the example configuration files to configure persistent storage for your metering installation.

4.3.1. Storing data in Amazon S3

Metering can use an existing Amazon S3 bucket or create a bucket for storage.

> **NOTE**
>
> Metering does not manage or delete any S3 bucket data. You must manually clean up S3 buckets that are used to store metering data.

Procedure

1. Edit the **spec.storage** section in the **s3-storage.yaml** file:

Example s3-storage.yaml file

```
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. Specify the name of the bucket where you would like to store your data. Optional: Specify the path within the bucket.

2. Specify the region of your bucket.

3. The name of a secret in the metering namespace containing the AWS credentials in the **data.aws-access-key-id** and **data.aws-secret-access-key** fields. See the example **Secret** object below for more details.

4. Set this field to **false** if you want to provide an existing S3 bucket, or if you do not want to provide IAM credentials that have **CreateBucket** permissions.

2. Use the following **Secret** object as a template:

Example AWS Secret object

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

NOTE

The values of the **aws-access-key-id** and **aws-secret-access-key** must be base64 encoded.

3. Create the 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
```

> **NOTE**
>
> This command automatically base64 encodes your **aws-access-key-id** and **aws-secret-access-key** values.

The **aws-access-key-id** and **aws-secret-access-key** credentials must have read and write access to the bucket. The following **aws/read-write.json** file shows an IAM policy that grants the required permissions:

Example aws/read-write.json file

```
{
    "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"
            ]
        }
    ]
}
```

If **spec.storage.hive.s3.createBucket** is set to **true** or unset in your **s3-storage.yaml** file, then you should use the **aws/read-write-create.json** file that contains permissions for creating and deleting buckets:

Example aws/read-write-create.json file

```
{
    "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. Storing data in S3-compatible storage

You can use S3-compatible storage such as Noobaa.

Procedure

1. Edit the **spec.storage** section in the **s3-compatible-storage.yaml** file:

   Example s3-compatible-storage.yaml file


```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "s3Compatible"
      s3Compatible:
        bucket: "bucketname" (1)
        endpoint: "http://example:port-number" (2)
        secretName: "my-aws-secret" (3)
```


   1. Specify the name of your S3-compatible bucket.
   2. Specify the endpoint for your storage.
   3. The name of a secret in the metering namespace containing the AWS credentials in the **data.aws-access-key-id** and **data.aws-secret-access-key** fields. See the example **Secret** object below for more details.

2. Use the following **Secret** object as a template:

   Example S3-compatible Secret object


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

4.3.3. Storing data in Microsoft Azure

To store data in Azure blob storage, you must use an existing container.

Procedure

1. Edit the **spec.storage** section in the **azure-blob-storage.yaml** file:

   Example azure-blob-storage.yaml file

   ```
   apiVersion: metering.openshift.io/v1
   kind: MeteringConfig
   metadata:
     name: "operator-metering"
   spec:
     storage:
       type: "hive"
       hive:
         type: "azure"
         azure:
           container: "bucket1" (1)
           secretName: "my-azure-secret" (2)
           rootDirectory: "/testDir" (3)
   ```

   (1) Specify the container name.

   (2) Specify a secret in the metering namespace. See the example **Secret** object below for more details.

   (3) Optional: Specify the directory where you would like to store your data.

2. Use the following **Secret** object as a template:

   Example Azure Secret object

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

3. Create the 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. Storing data in Google Cloud Storage

To store your data in Google Cloud Storage, you must use an existing bucket.

Procedure

1. Edit the **spec.storage** section in the **gcs-storage.yaml** file:

   Example gcs-storage.yaml file

   ```
   apiVersion: metering.openshift.io/v1
   kind: MeteringConfig
   metadata:
     name: "operator-metering"
   spec:
     storage:
       type: "hive"
       hive:
         type: "gcs"
         gcs:
           bucket: "metering-gcs/test1" 1
           secretName: "my-gcs-secret" 2
   ```

   1 Specify the name of the bucket. You can optionally specify the directory within the bucket where you would like to store your data.

   2 Specify a secret in the metering namespace. See the example **Secret** object below for more details.

2. Use the following **Secret** object as a template:

   Example Google Cloud Storage Secret object

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

3. Create the 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. Storing data in shared volumes

Metering does not configure storage by default. However, you can use any ReadWriteMany persistent volume (PV) or any storage class that provisions a ReadWriteMany PV for metering storage.

NOTE

NFS is not recommended to use in production. Using an NFS server on RHEL as a storage back end can fail to meet metering requirements and to provide the performance that is needed for the Metering Operator to work appropriately.

Other NFS implementations on the marketplace might not have these issues, such as a Parallel Network File System (pNFS). pNFS is an NFS implementation with distributed and parallel capability. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against OpenShift Container Platform core components.

Procedure

1. Modify the **shared-storage.yaml** file to use a ReadWriteMany persistent volume for storage:

```
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
```

Select one of the configuration options below:

1 Set **storage.hive.sharedPVC.claimName** to the name of an existing ReadWriteMany persistent volume claim (PVC). This configuration is necessary if you do not have dynamic volume provisioning or want to have more control over how the persistent volume is created.

2 Set **storage.hive.sharedPVC.createPVC** to **true** and set the **storage.hive.sharedPVC.storageClass** to the name of a storage class with ReadWriteMany access mode. This configuration uses dynamic volume provisioning to create a volume automatically.

2. Create the following resource objects that are required to deploy an NFS server for metering. Use the **oc create -f <file-name>.yaml** command to create the object YAML files.

   a. Configure a **PersistentVolume** resource object:

   Example nfs_persistentvolume.yaml file

```
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. Must exactly match the **[kind: StorageClass].metadata.name** field value.

b. Configure a **Pod** resource object with the **nfs-server** role:

Example nfs_server.yaml file

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-server
  labels:
    role: nfs-server
spec:
  containers:
    - name: nfs-server
      image: <image_name> (1)
      imagePullPolicy: IfNotPresent
      ports:
        - name: nfs
          containerPort: 2049
      securityContext:
        privileged: true
      volumeMounts:
      - mountPath: "/mnt/data"
        name: local
  volumes:
    - name: local
      emptyDir: {}
```

1. Install your NFS server image.

c. Configure a **Service** resource object with the **nfs-server** role:

Example nfs_service.yaml file

```
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. Configure a **StorageClass** resource object:

Example nfs_storageclass.yaml file

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-server (1)
provisioner: example.com/nfs
parameters:
  archiveOnDelete: "false"
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

(1) Must exactly match the **[kind: PersistentVolume].spec.storageClassName** field value.

> **WARNING**
>
> Configuration of your NFS storage, and any relevant resource objects, will vary depending on the NFS server image that you use for metering storage.

4.4. CONFIGURING THE HIVE METASTORE

Hive metastore is responsible for storing all the metadata about the database tables created in Presto and Hive. By default, the metastore stores this information in a local embedded Derby database in a persistent volume attached to the pod.

Generally, the default configuration of the Hive metastore works for small clusters, but users may wish to improve performance or move storage requirements out of cluster by using a dedicated SQL database for storing the Hive metastore data.

4.4.1. Configuring persistent volumes

By default, Hive requires one persistent volume to operate.

hive-metastore-db-data is the main persistent volume claim (PVC) required by default. This PVC is used by the Hive metastore to store metadata about tables, such as table name, columns, and location. Hive metastore is used by Presto and the Hive server to look up table metadata when processing queries. You remove this requirement by using MySQL or PostgreSQL for the Hive metastore database.

To install, Hive metastore requires that dynamic volume provisioning is enabled in a storage class, a persistent volume of the correct size must be manually pre-created, or you use a pre-existing MySQL or PostgreSQL database.

4.4.1.1. Configuring the storage class for the Hive metastore

To configure and specify a storage class for the **hive-metastore-db-data** persistent volume claim, specify the storage class in your **MeteringConfig** custom resource. An example **storage** section with the **class** field is included in the **metastore-storage.yaml** file below.

```
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) Uncomment this line and replace **null** with the name of the storage class to use. Leaving the value **null** will cause metering to use the default storage class for the cluster.

4.4.1.2. Configuring the volume size for the Hive metastore

Use the **metastore-storage.yaml** file below as a template to configure the volume size for the Hive metastore.

```
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" (1)
```

(1) Replace the value for **size** with your desired capacity. The example file shows "5Gi".

4.4.2. Use MySQL or PostgreSQL for the Hive metastore

The default installation of metering configures Hive to use an embedded Java database called Derby. This is unsuited for larger environments and can be replaced with either a MySQL or PostgreSQL database. Use the following example configuration files if your deployment requires a MySQL or PostgreSQL database for Hive.

There are 4 configuration options you can use to control the database used by Hive metastore: url, driver, username, and password.

Use the example configuration file below to use a MySQL database for Hive:

```
spec:
  hive:
    spec:
      metastore:
        storage:
          create: false
      config:
        db:
          url: "jdbc:mysql://mysql.example.com:3306/hive_metastore"
          driver: "com.mysql.jdbc.Driver"
          username: "REPLACEME"
          password: "REPLACEME"
```

You can pass additional JDBC parameters using the **spec.hive.config.url**. For more details see the MySQL Connector/J documentation.

Use the example configuration file below to use a PostgreSQL database for Hive:

```
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"
```

You can pass additional JDBC parameters using the URL. For more details see the PostgreSQL JDBC driver documentation.

4.5. CONFIGURING THE REPORTING OPERATOR

The Reporting Operator is responsible for collecting data from Prometheus, storing the metrics in Presto, running report queries against Presto, and exposing their results via an HTTP API. Configuring the Reporting Operator is primarily done in your **MeteringConfig** custom resource.

4.5.1. Securing a Prometheus connection

When you install metering on OpenShift Container Platform, Prometheus is available at https://prometheus-k8s.openshift-monitoring.svc:9091/.

To secure the connection to Prometheus, the default metering installation uses the OpenShift Container Platform certificate authority (CA). If your Prometheus instance uses a different CA, you can inject the CA through a config map. You can also configure the Reporting Operator to use a specified bearer token to authenticate with Prometheus.

Procedure

- Inject the CA that your Prometheus instance uses through a config map. For example:

```
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-----
```

  Alternatively, to use the system certificate authorities for publicly valid certificates, set both **useServiceAccountCA** and **configMap.enabled** to **false**.

- Specify a bearer token to authenticate with Prometheus. For example:

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

4.5.2. Exposing the reporting API

On OpenShift Container Platform the default metering installation automatically exposes a route, making the reporting API available. This provides the following features:

- Automatic DNS
- Automatic TLS based on the cluster CA

Also, the default installation makes it possible to use the OpenShift service for serving certificates to protect the reporting API with TLS. The OpenShift OAuth proxy is deployed as a sidecar container for the Reporting Operator, which protects the reporting API with authentication.

4.5.2.1. Using OpenShift Authentication

By default, the reporting API is secured with TLS and authentication. This is done by configuring the Reporting Operator to deploy a pod containing both the Reporting Operator's container, and a sidecar container running OpenShift auth-proxy.

To access the reporting API, the Metering Operator exposes a route. Once that route has been installed, you can run the following command to get the route's hostname.

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

Next, set up authentication using either a service account token or basic authentication with a username and password.

4.5.2.1.1. Authenticate using a service account token

With this method, you use the token in the Reporting Operator's service account, and pass that bearer token to the Authorization header in the following command:

```
$ 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]"
```

Be sure to replace the **name=[Report Name]** and **format=[Format]** parameters in the URL above. The **format** parameter can be json, csv, or tabular.

4.5.2.1.2. Authenticate using a username and password

Metering supports configuring basic authentication using a username and password combination, which is specified in the contents of an htpasswd file. By default, a secret containing empty htpasswd data is created. You can, however, configure the **reporting-operator.spec.authProxy.htpasswd.data** and **reporting-operator.spec.authProxy.htpasswd.createSecret** keys to use this method.

Once you have specified the above in your **MeteringConfig** resource, you can run the following command:

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

Be sure to replace **testuser:password123** with a valid username and password combination.

4.5.2.2. Manually Configuring Authentication

To manually configure, or disable OAuth in the Reporting Operator, you must set **spec.tls.enabled: false** in your **MeteringConfig** resource.

> **WARNING**
>
> This also disables all TLS and authentication between the Reporting Operator, Presto, and Hive. You would need to manually configure these resources yourself.

Authentication can be enabled by configuring the following options. Enabling authentication configures the Reporting Operator pod to run the OpenShift auth-proxy as a sidecar container in the pod. This adjusts the ports so that the reporting API isn't exposed directly, but instead is proxied to via the auth-proxy sidecar container.

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

You need to set **reporting-operator.spec.authProxy.enabled** and **reporting-operator.spec.authProxy.cookie.createSecret** to **true** and **reporting-operator.spec.authProxy.cookie.seed** to a 32-character random string.

You can generate a 32-character random string using the following command.

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

4.5.2.2.1. Token authentication

When the following options are set to **true**, authentication using a bearer token is enabled for the reporting REST API. Bearer tokens can come from service accounts or users.

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

When authentication is enabled, the Bearer token used to query the reporting API of the user or service account must be granted access using one of the following roles:

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

The Metering Operator is capable of creating role bindings for you, granting these permissions by specifying a list of subjects in the **spec.permissions** section. For an example, see the following **advanced-auth.yaml** example configuration.

```
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
```

Alternatively, you can use any role which has rules granting **get** permissions to **reports/export**. This means **get** access to the **export** sub-resource of the **Report** resources in the namespace of the Reporting Operator. For example: **admin** and **cluster-admin**.

By default, the Reporting Operator and Metering Operator service accounts both have these permissions, and their tokens can be used for authentication.

4.5.2.2.2. Basic authentication with a username and password

For basic authentication you can supply a username and password in the **reporting-operator.spec.authProxy.htpasswd.data** field. The username and password must be the same format as those found in an htpasswd file. When set, you can use HTTP basic authentication to provide your username and password that has a corresponding entry in the **htpasswdData** contents.

4.6. CONFIGURE AWS BILLING CORRELATION

Metering can correlate cluster usage information with AWS detailed billing information , attaching a dollar amount to resource usage. For clusters running in EC2, you can enable this by modifying the example **aws-billing.yaml** file below.

```
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>" 1
        prefix: "<path/to/report>"
        region: "<your-buckets-region>"

  reporting-operator:
    spec:
      config:
        aws:
          secretName: "<your-aws-secret>" 2

  presto:
    spec:
      config:
        aws:
          secretName: "<your-aws-secret>" 3

  hive:
    spec:
      config:
        aws:
          secretName: "<your-aws-secret>" 4
```

To enable AWS billing correlation, first ensure the AWS Cost and Usage Reports are enabled. For more information, see Turning on the AWS Cost and Usage Report in the AWS documentation.

1. Update the bucket, prefix, and region to the location of your AWS Detailed billing report.

2 3 4 All **secretName** fields should be set to the name of a secret in the metering namespace containing AWS credentials in the **data.aws-access-key-id** and **data.aws-secret-access-key**

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

To store data in S3, the **aws-access-key-id** and **aws-secret-access-key** credentials must have read and write access to the bucket. For an example of an IAM policy granting the required permissions, see the **aws/read-write.json** file below.

```
{
    "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 Replace **operator-metering-data** with the name of your bucket.

This can be done either pre-installation or post-installation. Disabling it post-installation can cause errors in the Reporting Operator.

CHAPTER 5. REPORTS

5.1. ABOUT REPORTS

A **Report** custom resource provides a method to manage periodic Extract Transform and Load (ETL) jobs using SQL queries. Reports are composed from other metering resources, such as **ReportQuery** resources that provide the actual SQL query to run, and **ReportDataSource** resources that define the data available to the **ReportQuery** and **Report** resources.

Many use cases are addressed by the predefined **ReportQuery** and **ReportDataSource** resources that come installed with metering. Therefore, you do not need to define your own unless you have a use case that is not covered by these predefined resources.

5.1.1. Reports

The **Report** custom resource is used to manage the execution and status of reports. Metering produces reports derived from usage data sources, which can be used in further analysis and filtering. A single **Report** resource represents a job that manages a database table and updates it with new information according to a schedule. The report exposes the data in that table via the Reporting Operator HTTP API.

Reports with a **spec.schedule** field set are always running and track what time periods it has collected data for. This ensures that if metering is shutdown or unavailable for an extended period of time, it backfills the data starting where it left off. If the schedule is unset, then the report runs once for the time specified by the **reportingStart** and **reportingEnd**. By default, reports wait for **ReportDataSource** resources to have fully imported any data covered in the reporting period. If the report has a schedule, it waits to run until the data in the period currently being processed has finished importing.

5.1.1.1. Example report with a schedule

The following example **Report** object contains information on every pod's CPU requests, and runs every hour, adding the last hours worth of data each time it runs.

```
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. Example report without a schedule (run-once)

The following example **Report** object contains information on every pod's CPU requests for all of July. After completion, it does not run again.

```
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. query

The **query** field names the **ReportQuery** resource used to generate the report. The report query controls the schema of the report as well as how the results are processed.

query is a required field.

Use the following command to list available **ReportQuery** resources:

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

Example output

```
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
```

Report queries with the **-raw** suffix are used by other **ReportQuery** resources to build more complex queries, and should not be used directly for reports.

namespace- prefixed queries aggregate pod CPU and memory requests by namespace, providing a list of namespaces and their overall usage based on resource requests.

pod- prefixed queries are similar to **namespace-** prefixed queries but aggregate information by pod rather than namespace. These queries include the pod's namespace and node.

node- prefixed queries return information about each node's total available resources.

aws- prefixed queries are specific to AWS. Queries suffixed with **-aws** return the same data as queries of the same name without the suffix, and correlate usage with the EC2 billing data.

The **aws-ec2-billing-data** report is used by other queries, and should not be used as a standalone report. The **aws-ec2-cluster-cost** report provides a total cost based on the nodes included in the cluster, and the sum of their costs for the time period being reported on.

Use the following command to get the **ReportQuery** resource as YAML, and check the **spec.columns** field. For example, run:

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

Example output

```
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. schedule

The **spec.schedule** configuration block defines when the report runs. The main fields in the **schedule** section are **period**, and then depending on the value of **period**, the fields **hourly**, **daily**, **weekly**, and **monthly** allow you to fine-tune when the report runs.

For example, if **period** is set to **weekly**, you can add a **weekly** field to the **spec.schedule** block. The following example will run once a week on Wednesday, at 1 PM (hour 13 in the day).

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

5.1.1.4.1. period

Valid values of **schedule.period** are listed below, and the options available to set for a given period are also listed.

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

  - **minute**
  - **second**
- **cron**
  - **expression**

Generally, the **hour**, **minute**, **second** fields control when in the day the report runs, and **dayOfWeek**/**dayOfMonth** control what day of the week, or day of month the report runs on, if it is a weekly or monthly report period.

For each of these fields, there is a range of valid values:

- **hour** is an integer value between 0-23.
- **minute** is an integer value between 0-59.
- **second** is an integer value between 0-59.
- **dayOfWeek** is a string value that expects the day of the week (spelled out).
- **dayOfMonth** is an integer value between 1-31.

For cron periods, normal cron expressions are valid:

- **expression: "*/5 * * * *"**

5.1.1.5. reportingStart

To support running a report against existing data, you can set the **spec.reportingStart** field to a RFC3339 timestamp to tell the report to run according to its **schedule** starting from **reportingStart** rather than the current time.

> **NOTE**
>
> Setting the **spec.reportingStart** field to a specific time will result in the Reporting Operator running many queries in succession for each interval in the schedule that is between the **reportingStart** time and the current time. This could be thousands of queries if the period is less than daily and the **reportingStart** is more than a few months back. If **reportingStart** is left unset, the report will run at the next full **reportingPeriod** after the time the report is created.

As an example of how to use this field, if you had data already collected dating back to January 1st, 2019 that you want to include in your **Report** object, you can create a report with the following values:

```
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

To configure a report to only run until a specified time, you can set the **spec.reportingEnd** field to an RFC3339 timestamp. The value of this field will cause the report to stop running on its schedule after it has finished generating reporting data for the period covered from its start time until **reportingEnd**.

Because a schedule will most likely not align with the **reportingEnd**, the last period in the schedule will be shortened to end at the specified **reportingEnd** time. If left unset, then the report will run forever, or until a **reportingEnd** is set on the report.

For example, if you want to create a report that runs once a week for the month of July:

```
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. runImmediately

When **runImmediately** is set to **true**, the report runs immediately. This behavior ensures that the report is immediately processed and queued without requiring additional scheduling parameters.

> **NOTE**
>
> When **runImmediately** is set to **true**, you must set a **reportingEnd** and **reportingStart** value.

5.1.1.8. inputs

The **spec.inputs** field of a **Report** object can be used to override or set values defined in a **ReportQuery** resource's **spec.inputs** field.

spec.inputs is a list of name-value pairs:

```
spec:
  inputs:
  - name: "NamespaceCPUUsageReportName" (1)
    value: "namespace-cpu-usage-hourly" (2)
```

1. The **name** of an input must exist in the ReportQuery's **inputs** list.
2. The **value** of the input must be the correct type for the input's **type**.

5.1.1.9. Roll-up reports

Report data is stored in the database much like metrics themselves, and therefore, can be used in aggregated or roll-up reports. A simple use case for a roll-up report is to spread the time required to produce a report over a longer period of time. This is instead of requiring a monthly report to query and add all data over an entire month. For example, the task can be split into daily reports that each run over 1/30 of the data.

A custom roll-up report requires a custom report query. The **ReportQuery** resource template processor provides a **reportTableName** function that can get the necessary table name from a **Report** object's **metadata.name**.

Below is a snippet taken from a built-in query:

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 |}
...
```

Example aggregated-report.yaml roll-up report

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

5.1.1.9.1. Report status

The execution of a scheduled report can be tracked using its status field. Any errors occurring during the preparation of a report will be recorded here.

The **status** field of a **Report** object currently has two fields:

- **conditions**: Conditions is a list of conditions, each of which have a **type**, **status**, **reason**, and **message** field. Possible values of a condition's **type** field are **Running** and **Failure**, indicating the current state of the scheduled report. The **reason** indicates why its **condition** is in its current state with the **status** being either **true**, **false** or, **unknown**. The **message** provides a human readable indicating why the condition is in the current state. For detailed information on the **reason** values, see **pkg/apis/metering/v1/util/report_util.go**.

- **lastReportTime**: Indicates the time metering has collected data up to.

5.2. STORAGE LOCATIONS

A **StorageLocation** custom resource configures where data will be stored by the Reporting Operator. This includes the data collected from Prometheus, and the results produced by generating a **Report** custom resource.

You only need to configure a **StorageLocation** custom resource if you want to store data in multiple locations, like multiple S3 buckets or both S3 and HDFS, or if you wish to access a database in Hive and Presto that was not created by metering. For most users this is not a requirement, and the documentation on configuring metering is sufficient to configure all necessary storage components.

5.2.1. Storage location examples

The following example shows the built-in local storage option, and is configured to use Hive. By default, data is stored wherever Hive is configured to use storage, such as HDFS, S3, or a **ReadWriteMany** persistent volume claim (PVC).

Local storage example

```
apiVersion: metering.openshift.io/v1
kind: StorageLocation
metadata:
  name: hive
  labels:
    operator-metering: "true"
spec:
  hive: (1)
    databaseName: metering (2)
    unmanagedDatabase: false (3)
```

1. If the **hive** section is present, then the **StorageLocation** resource will be configured to store data in Presto by creating the table using the Hive server. Only **databaseName** and **unmanagedDatabase** are required fields.
2. The name of the database within hive.
3. If **true**, the **StorageLocation** resource will not be actively managed, and the **databaseName** is expected to already exist in Hive. If **false**, the Reporting Operator will create the database in Hive.

The following example uses an AWS S3 bucket for storage. The prefix is appended to the bucket name when constructing the path to use.

Remote storage example

```
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. Optional: The filesystem URL for Presto and Hive to use for the database. This can be an **hdfs://** or **s3a://** filesystem URL.

There are additional optional fields that can be specified in the **hive** section:

- **defaultTableProperties**: Contains configuration options for creating tables using Hive.
- **fileFormat**: The file format used for storing files in the filesystem. See the Hive Documentation on File Storage Format for a list of options and more details.
- **rowFormat**: Controls the Hive row format. This controls how Hive serializes and deserializes rows. See the Hive Documentation on Row Formats and SerDe for more details.

5.2.2. Default storage location

If an annotation **storagelocation.metering.openshift.io/is-default** exists and is set to **true** on a **StorageLocation** resource, then that resource becomes the default storage resource. Any components with a storage configuration option where the storage location is not specified will use the default storage resource. There can be only one default storage resource. If more than one resource with the annotation exists, an error is logged because the Reporting Operator cannot determine the default.

Default storage example

```
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"
```

CHAPTER 6. USING METERING

6.1. PREREQUISITES

- Install Metering
- Review the details about the available options that can be configured for a report and how they function.

6.2. WRITING REPORTS

Writing a report is the way to process and analyze data using metering.

To write a report, you must define a **Report** resource in a YAML file, specify the required parameters, and create it in the **openshift-metering** namespace.

Prerequisites

- Metering is installed.

Procedure

1. Change to the **openshift-metering** project:

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

2. Create a **Report** resource as a YAML file:

   a. Create a YAML file with the following content:

```
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) The **query** specifies the **ReportQuery** resources used to generate the report. Change this based on what you want to report on. For a list of options, run **oc get reportqueries | grep -v raw**.

(1) Use a descriptive name about what the report does for **metadata.name**. A good name describes the query, and the schedule or period you used.

(3) Set **runImmediately** to **true** for it to run with whatever data is available, or set it to **false** if you want it to wait for **reportingEnd** to pass.

   b. Run the following command to create the **Report** resource:

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

Example output

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

3. You can list reports and their **Running** status with the following command:

```
$ oc get reports
```

Example output

```
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. VIEWING REPORT RESULTS

Viewing a report's results involves querying the reporting API route and authenticating to the API using your OpenShift Container Platform credentials. Reports can be retrieved as **JSON**, **CSV**, or **Tabular** formats.

Prerequisites

- Metering is installed.
- To access report results, you must either be a cluster administrator, or you need to be granted access using the **report-exporter** role in the **openshift-metering** namespace.

Procedure

1. Change to the **openshift-metering** project:

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

2. Query the reporting API for results:

a. Create a variable for the metering **reporting-api** route then get the route:

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

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

b. Get the token of your current user to be used in the request:

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

c. Set **reportName** to the name of the report you created:

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

d. Set **reportFormat** to one of **csv**, **json**, or **tabular** to specify the output format of the API response:

```
$ reportFormat=csv
```

e. To get the results, use **curl** to make a request to the reporting API for your report:

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

Example output with reportName=namespace-cpu-request-2019 and reportFormat=csv

```
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
```

CHAPTER 7. EXAMPLES OF USING METERING

Use the following example reports to get started measuring capacity, usage, and utilization in your cluster. These examples showcase the various types of reports metering offers, along with a selection of the predefined queries.

7.1. PREREQUISITES

- Install metering
- Review the details about writing and viewing reports.

7.2. MEASURE CLUSTER CAPACITY HOURLY AND DAILY

The following report demonstrates how to measure cluster capacity both hourly and daily. The daily report works by aggregating the hourly report's results.

The following report measures cluster CPU capacity every hour.

Hourly CPU capacity by cluster example

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

(1) You could change this period to **daily** to get a daily report, but with larger data sets it is more efficient to use an hourly report, then aggregate your hourly data into a daily report.

The following report aggregates the hourly data into a daily report.

Daily CPU capacity by cluster example

```
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) To stay organized, remember to change the **name** of your report if you change any of the other values.

2 You can also measure **cluster-memory-capacity**. Remember to update the query in the associated hourly report as well.

3 The **inputs** section configures this report to aggregate the hourly report. Specifically, **value: cluster-cpu-capacity-hourly** is the name of the hourly report that gets aggregated.

7.3. MEASURE CLUSTER USAGE WITH A ONE-TIME REPORT

The following report measures cluster usage from a specific starting date forward. The report only runs once, after you save it and apply it.

CPU usage by cluster example

```
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 To stay organized, remember to change the **name** of your report if you change any of the other values.

2 Configures the report to start using data from the **reportingStart** timestamp until the **reportingEnd** timestamp.

3 Adjust your query here. You can also measure cluster usage with the **cluster-memory-usage** query.

4 Configures the report to run immediately after saving it and applying it.

7.4. MEASURE CLUSTER UTILIZATION USING CRON EXPRESSIONS

You can also use cron expressions when configuring the period of your reports. The following report measures cluster utilization by looking at CPU utilization from 9am-5pm every weekday.

Weekday CPU utilization by cluster example

```
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. To say organized, remember to change the **name** of your report if you change any of the other values.
2. Adjust your query here. You can also measure cluster utilization with the **cluster-memory-utilization** query.
3. For cron periods, normal cron expressions are valid.

CHAPTER 8. TROUBLESHOOTING AND DEBUGGING METERING

Use the following sections to help troubleshoot and debug specific issues with metering.

In addition to the information in this section, be sure to review the following topics:

- Prerequisites for installing metering.
- About configuring metering

8.1. TROUBLESHOOTING METERING

A common issue with metering is pods failing to start. Pods might fail to start due to lack of resources or if they have a dependency on a resource that does not exist, such as a **StorageClass** or **Secret** resource.

8.1.1. Not enough compute resources

A common issue when installing or running metering is a lack of compute resources. As the cluster grows and more reports are created, the Reporting Operator pod requires more memory. If memory usage reaches the pod limit, the cluster considers the pod out of memory (OOM) and terminates it with an **OOMKilled** status. Ensure that metering is allocated the minimum resource requirements described in the installation prerequisites.

NOTE

The Metering Operator does not autoscale the Reporting Operator based on the load in the cluster. Therefore, CPU usage for the Reporting Operator pod does not increase as the cluster grows.

To determine if the issue is with resources or scheduling, follow the troubleshooting instructions included in the Kubernetes document Managing Compute Resources for Containers.

To troubleshoot issues due to a lack of compute resources, check the following within the **openshift-metering** namespace.

Prerequisites

- You are currently in the **openshift-metering** namespace. Change to the **openshift-metering** namespace by running:

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

Procedure

1. Check for metering **Report** resources that fail to complete and show the status of **ReportingPeriodUnmetDependencies**:

```
$ oc get reports
```

Example output

```
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. Check the **ReportDataSource** resources where the **NEWEST METRIC** is less than the report end date:

```
$ oc get reportdatasource
```

Example output

```
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. Check the health of the **reporting-operator Pod** resource for a high number of pod restarts:

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

Example output

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

(1) The Reporting Operator pod is restarting at a high rate.

4. Check the **reporting-operator Pod** resource for an **OOMKilled** termination:

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

Example output

```
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 The Reporting Operator pod was terminated due to OOM kill.

Increasing the reporting-operator pod memory limit

If you are experiencing an increase in pod restarts and OOM kill events, you can check the current memory limit set for the Reporting Operator pod. Increasing the memory limit allows the Reporting Operator pod to update the report data sources. If necessary, increase the memory limit in your **MeteringConfig** resource by 25% - 50%.

Procedure

1. Check the current memory limits of the **reporting-operator Pod** resource:

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

Example output

```
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 The current memory limit for the Reporting Operator pod.

2. Edit the **MeteringConfig** resource to update the memory limit:

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

Example MeteringConfig resource

```
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. Add or increase memory limits within the **resources** field of the **MeteringConfig** resource.

> NOTE
>
> If there continue to be numerous OOM killed events after memory limits are increased, this might indicate that a different issue is causing the reports to be in a pending state.

8.1.2. StorageClass resource not configured

Metering requires that a default **StorageClass** resource be configured for dynamic provisioning.

See the documentation on configuring metering for information on how to check if there are any **StorageClass** resources configured for the cluster, how to set the default, and how to configure metering to use a storage class other than the default.

8.1.3. Secret not configured correctly

A common issue with metering is providing the incorrect secret when configuring your persistent storage. Be sure to review the example configuration files and create you secret according to the guidelines for your storage provider.

8.2. DEBUGGING METERING

Debugging metering is much easier when you interact directly with the various components. The sections below detail how you can connect and query Presto and Hive as well as view the dashboards of the Presto and HDFS components.

> NOTE
>
> All of the commands in this section assume you have installed metering through OperatorHub in the **openshift-metering** namespace.

8.2.1. Get reporting operator logs

Use the command below to follow the logs of the **reporting-operator**:

```
$ 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. Query Presto using presto-cli

The following command opens an interactive presto-cli session where you can query Presto. This session runs in the same container as Presto and launches an additional Java instance, which can create memory limits for the pod. If this occurs, you should increase the memory request and limits of the Presto pod.

By default, Presto is configured to communicate using TLS. You must use the following command to run Presto queries:

```
$ 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
```

Once you run this command, a prompt appears where you can run queries. Use the **show tables from metering;** query to view the list of tables:

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

Example output

```
                              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. Query Hive using beeline

The following opens an interactive beeline session where you can query Hive. This session runs in the same container as Hive and launches an additional Java instance, which can create memory limits for the pod. If this occurs, you should increase the memory request and limits of the Hive pod.

```
$ 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'
```

Once you run this command, a prompt appears where you can run queries. Use the **show tables;** query to view the list of tables:

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

Example output

```
+----------------------------------------------------+
|                      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. Port-forward to the Hive web UI

Run the following command to port-forward to the Hive web UI:

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

You can now open http://127.0.0.1:10002 in your browser window to view the Hive web interface.

8.2.5. Port-forward to HDFS

Run the following command to port-forward to the HDFS namenode:

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

You can now open http://127.0.0.1:9870 in your browser window to view the HDFS web interface.

Run the following command to port-forward to the first HDFS datanode:

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

(1) To check other datanodes, replace **hdfs-datanode-0** with the pod you want to view information on.

8.2.6. Metering Ansible Operator

Metering uses the Ansible Operator to watch and reconcile resources in a cluster environment. When debugging a failed metering installation, it can be helpful to view the Ansible logs or status of your **MeteringConfig** custom resource.

8.2.6.1. Accessing Ansible logs

In the default installation, the Metering Operator is deployed as a pod. In this case, you can check the logs of the Ansible container within this pod:

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

Alternatively, you can view the logs of the Operator container (replace **-c ansible** with **-c operator**) for condensed output.

8.2.6.2. Checking the MeteringConfig Status

It can be helpful to view the **.status** field of your **MeteringConfig** custom resource to debug any recent failures. The following command shows status messages with type **Invalid**:

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

8.2.6.3. Checking MeteringConfig Events

Check events that the Metering Operator is generating. This can be helpful during installation or upgrade to debug any resource failures. Sort events by the last timestamp:

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

Example output with latest changes in the MeteringConfig resources

```
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
```

CHAPTER 9. UNINSTALLING METERING

You can remove metering from your OpenShift Container Platform cluster.

> **NOTE**
>
> Metering does not manage or delete Amazon S3 bucket data. After uninstalling metering, you must manually clean up S3 buckets that were used to store metering data.

9.1. REMOVING THE METERING OPERATOR FROM YOUR CLUSTER

Remove the Metering Operator from your cluster by following the documentation on deleting Operators from a cluster.

> **NOTE**
>
> Removing the Metering Operator from your cluster does not remove its custom resource definitions or managed resources. See the following sections on Uninstalling a metering namespace and Uninstalling metering custom resource definitions for steps to remove any remaining metering components.

9.2. UNINSTALLING A METERING NAMESPACE

Uninstall your metering namespace, for example the **openshift-metering** namespace, by removing the **MeteringConfig** resource and deleting the **openshift-metering** namespace.

Prerequisites

- The Metering Operator is removed from your cluster.

Procedure

1. Remove all resources created by the Metering Operator:

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

2. After the previous step is complete, verify that all pods in the **openshift-metering** namespace are deleted or are reporting a terminating state:

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

3. Delete the **openshift-metering** namespace:

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

9.3. UNINSTALLING METERING CUSTOM RESOURCE DEFINITIONS

The metering custom resource definitions (CRDs) remain in the cluster after the Metering Operator is uninstalled and the **openshift-metering** namespace is deleted.

IMPORTANT

Deleting the metering CRDs disrupts any additional metering installations in other namespaces in your cluster. Ensure that there are no other metering installations before proceeding.

Prerequisites

- The **MeteringConfig** custom resource in the **openshift-metering** namespace is deleted.
- The **openshift-metering** namespace is deleted.

Procedure

- Delete the remaining metering CRDs:

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