Metering

Configuring and using Metering in OpenShift Container Platform
OpenShift Container Platform 4.3 Metering

Configuring and using Metering in OpenShift Container Platform
Abstract

This document provides instructions for configuring and using metering in OpenShift Container Platform.
# Table of Contents

## CHAPTER 1. ABOUT METERING

1.1. METERING OVERVIEW

## CHAPTER 2. INSTALLING METERING

2.1. PREREQUISITES

2.2. INSTALLING THE METERING OPERATOR

2.3. INSTALLING THE METERING STACK

2.4. VERIFY INSTALLATION

2.5. ADDITIONAL RESOURCES

## CHAPTER 3. CONFIGURING METERING

3.1. ABOUT CONFIGURING METERING

3.2. COMMON CONFIGURATION OPTIONS

3.2.1. Resource requests and limits

3.2.2. Node selectors

3.3. CONFIGURING PERSISTENT STORAGE

3.3.1. Storing data in Amazon S3

3.3.2. Storing data in S3-compatible storage

3.3.3. Storing data in Microsoft Azure

3.3.4. Storing data in Google Cloud Storage

3.3.5. Storing data in shared volumes

3.4. CONFIGURING THE HIVE METASTORE

3.4.1. Configuring PersistentVolumes

3.4.1.1. Configuring the storage class for Hive metastore

3.4.1.2. Configuring the volume sizes for the Hive Metastore

3.4.2. Use MySQL or PostgreSQL for the Hive metastore

3.5. CONFIGURING THE REPORTING-OPERATOR

3.5.1. Prometheus connection

3.5.2. Exposing the reporting API

3.5.2.1. Using OpenShift Authentication

3.5.2.1.1. Authenticate using a service account token

3.5.2.1.2. Authenticate using a username and password

3.5.2.2. Manually Configuring Authentication

3.5.2.2.1. Token authentication

3.5.2.2.2. Basic authentication (username/password)

3.6. CONFIGURE AWS BILLING CORRELATION

## CHAPTER 4. REPORTS

4.1. ABOUT REPORTS

4.1.1. Reports

4.1.1.1. Example Report with a Schedule

4.1.1.2. Example Report without a Schedule (Run-Once)

4.1.1.3. query

4.1.1.4. schedule

4.1.1.4.1. period

4.1.1.5. reportingStart

4.1.1.6. reportingEnd

4.1.1.7. runImmediately

4.1.1.8. inputs

4.1.1.9. Roll-up Reports

4.1.1.9.1. Report Status
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 CustomResourceDefinitions (CRDs):

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeteringConfig</td>
<td>Configures the metering stack for deployment. Contains customizations and configuration options to control each component that makes up the metering stack.</td>
</tr>
<tr>
<td>Reports</td>
<td>Controls what query to use, when, and how often the query should be run, and where to store the results.</td>
</tr>
<tr>
<td>ReportQueries</td>
<td>Contains the SQL queries used to perform analysis on the data contained within ReportDataSources.</td>
</tr>
<tr>
<td>ReportDataSources</td>
<td>Controls the data available to ReportQueries and Reports. Allows configuring access to different databases for use within metering.</td>
</tr>
</tbody>
</table>
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 CustomResource, referred to here as your MeteringConfig. Installing the Metering Operator creates a default MeteringConfig that you can modify using the examples in the documentation. After creating your MeteringConfig, install the metering stack. Last, verify your installation.

2.1. PREREQUISITES

Metering requires the following components:

- A StorageClass 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

To install the Metering Operator first create an openshift-metering namespace, and then install the Metering Operator from OperatorHub.

Procedure

1. In the OpenShift Container Platform web console, click Administration → Namespaces → Create Namespace.
2. Set the name to openshift-metering. No other namespace is supported. Label the namespace with openshift.io/cluster-monitoring=true, and click Create.
3. Next, click Operators → OperatorHub, and filter for metering to find the Metering Operator.
4. Click the Metering card, review the package description, and then click Install.
5. On the Create Operator Subscription screen, select the openshift-metering namespace you created above. Specify your update channel and approval strategy, then click Subscribe to install metering.
6. On the Installed Operators screen the Status displays InstallSucceeded when metering has finished installing. Click the name of the operator in the first column to view the Operator Details page.

NOTE

It may take a few minutes for the metering operator to appear.
From the **Operator Details** page you can create different resources related to metering. To complete your installation create a MeteringConfig resource to configure metering and install the components of the metering stack.

### 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.

**Prerequisites**

- Review the **configuration options**
- Create a MeteringConfig resource. You can begin the following process to generate a default MeteringConfig, 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 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 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.

### 2.4. VERIFY INSTALLATION

To verify your metering installation you can check that all the required Pods have been created, and check that your report data sources are beginning to import data.

**Procedure**

1. 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.
You can run the same check using the `oc` CLI:

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hive-metastore-0</td>
<td>1/2</td>
<td>Running</td>
<td>0</td>
<td>52s</td>
</tr>
<tr>
<td>hive-server-0</td>
<td>2/3</td>
<td>Running</td>
<td>0</td>
<td>52s</td>
</tr>
<tr>
<td>metering-operator-68dd64cfb6-pxh8v</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>2m49s</td>
</tr>
<tr>
<td>presto-coordinator-0</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>31s</td>
</tr>
<tr>
<td>reporting-operator-56c6c878fb-2zbhp</td>
<td>0/2</td>
<td>ContainerCreating</td>
<td>0</td>
<td>4s</td>
</tr>
</tbody>
</table>

2. Continue to check your Pods until they show `Ready`. This can take several minutes. 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 okay and can be expected during installation.

Using the `oc` CLI, the same check shows output similar to the following:

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hive-metastore-0</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>3m28s</td>
</tr>
<tr>
<td>hive-server-0</td>
<td>3/3</td>
<td>Running</td>
<td>0</td>
<td>3m28s</td>
</tr>
<tr>
<td>metering-operator-68dd64cfb6-2k7d9</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>5m17s</td>
</tr>
<tr>
<td>presto-coordinator-0</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>3m9s</td>
</tr>
<tr>
<td>reporting-operator-5588964bf8-x2tkn</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>2m40s</td>
</tr>
</tbody>
</table>

3. Next, use the `oc` CLI to verify that the ReportDataSources are beginning to import data, indicated by a valid timestamp in the `EARLIEST METRIC` column (this may take a few minutes).

We filter out the "-raw" ReportDataSources which do not import data:

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

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>EARLIEST METRIC</th>
<th>NEWEST METRIC</th>
<th>IMPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-allocatable-cpu-cores</td>
<td>2019-08-05T16:52:00Z</td>
<td>2019-08-05T18:52:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>node-allocatable-memory-bytes</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>node-capacity-cpu-cores</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>node-capacity-memory-bytes</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>persistentvolumeclaim-capacity-bytes</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>persistentvolumeclaim-phase</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>persistentvolumeclaim-request-bytes</td>
<td>2019-08-05T16:52:00Z</td>
<td>2019-08-05T18:52:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>persistentvolumeclaim-usage-bytes</td>
<td>2019-08-05T16:52:00Z</td>
<td>2019-08-05T18:52:00Z</td>
<td>9m50s</td>
</tr>
<tr>
<td>pod-limit-cpu-cores</td>
<td>2019-08-05T16:52:00Z</td>
<td>2019-08-05T18:52:00Z</td>
<td>9m49s</td>
</tr>
<tr>
<td>pod-limit-memory-bytes</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m49s</td>
</tr>
<tr>
<td>pod-persistentvolumeclaim-request-info</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m49s</td>
</tr>
<tr>
<td>pod-request-cpu-cores</td>
<td>2019-08-05T16:51:00Z</td>
<td>2019-08-05T18:51:00Z</td>
<td>9m49s</td>
</tr>
</tbody>
</table>
```
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.

2.5. 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. CONFIGURING METERING

3.1. ABOUT CONFIGURING METERING

A CustomResource called your MeteringConfig specifies all the configuration details for your metering installation. When you first install the metering stack, a default MeteringConfig 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 and reapply the file.

3.2. COMMON CONFIGURATION OPTIONS

3.2.1. 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.

```yaml
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:
```
3.2.2. Node selectors

If you want to run the metering components on specific sets of nodes, then you can set **nodeSelectors** on each component to control where each component of metering is scheduled. The **node-selectors.yaml** file below provides an example of setting node selectors for each component.

```
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
```

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

presto:
spec:
  coordinator:
    nodeSelector:
      "node-role.kubernetes.io/infra": "true"
  worker:
    nodeSelector:
      "node-role.kubernetes.io/infra": "true"
```

hive:
3.3. 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.

3.3.1. 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. When uninstalling metering, any S3 buckets used to store metering data must be manually cleaned up.

To use Amazon S3 for storage, edit the `spec.storage` section in the example `s3-storage.yaml` file below.

```yaml
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "s3"
      s3:
        bucket: "bucketname/path/"  
        region: "us-west-1"
        secretName: "my-aws-secret"
        # Set to false if you want to provide an existing bucket, instead of having metering create the bucket on your behalf.
        createBucket: true
```

1. Specify the name of the bucket where you would like to store your data. You may optionally 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 examples that follow for more details.
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.

Use the example secret below as a template.

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

You can use the following command to create the secret.

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

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/*",
        "arn:aws:s3:::operator-metering-data"
      ]
    }
  ]
}
If you left `spec.storage.hive.s3.createBucket` set to `true`, or unset, then you should use the `aws/read-write-create.json` file below, which contains permissions for creating and deleting buckets.

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

### 3.3.2. Storing data in S3-compatible storage

To use S3-compatible storage such as Noobaa, edit the `spec.storage` section in the example `s3-compatible-storage.yaml` file below.

```yaml
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.
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 that follows for more details.

Use the example secret below as a template.

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

### 3.3.3. Storing data in Microsoft Azure

To store data in Azure blob storage you must use an existing container. Edit the `spec.storage` section in the example `azure-blob-storage.yaml` file below.

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

1. Specify the container name.
2. Specify a secret in the metering namespace. See the examples that follow for more details.
3. You can optionally specify the directory where you would like to store your data.

Use the example secret below as a template.

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

You can use the following command to create the secret.

```
oc create secret -n openshift-metering generic your-azure-secret --from-literal=azure-storage-account-name=your-storage-account-name --from-literal=azure-secret-access-key=your-secret-key
```
3.3.4. Storing data in Google Cloud Storage

To store your data in Google Cloud Storage you must use an existing bucket. Edit the `spec.storage` section in the example `gcs-storage.yaml` file below.

```
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. Use the example that follows for more details.

Use the example secret below as a template:

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

You can use the following command to create the secret.

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

3.3.5. Storing data in shared volumes

**NOTE**

NFS is not recommended to use with metering.

Metering has no storage by default, but it can use any ReadWriteMany PersistentVolume or any StorageClass that provisions a ReadWriteMany PersistentVolume.

**Procedure**

- To use a ReadWriteMany PersistentVolume for storage, modify the `shared-storage.yaml` file below.

```
apiversion: metering.openshift.io/v1
```
Select one of the configuration options below:

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

2. Set `storage.hive.sharedPVC.createPVC` to `true` and set the `storage.hive.sharedPVC.storageClass` to the name of a StorageClass with ReadWriteMany access mode. This will use dynamic volume provisioning to have a volume created automatically.

3.4. 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 PersistentVolume 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.

3.4.1. Configuring PersistentVolumes

By default, Hive requires one PersistentVolume to operate.

`hive-metastore-db-data` is the main PersistentVolumeClaim (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 be enabled via a StorageClass, a persistent volume of the correct size must be manually pre-created, or that you use a pre-existing MySQL or PostgreSQL database.

3.4.1.1. Configuring the storage class for Hive metastore

To configure and specify a StorageClass for the `hive-metastore-db-data` PVC, specify the StorageClass in your MeteringConfig. An example StorageClass section is included in `metastore-storage.yaml` file below.
Uncomment this line and replace `null` with the name of the StorageClass to use. Leaving the value `null` will cause metering to use the default StorageClass for the cluster.

### 3.4.1.2. Configuring the volume sizes for the Hive Metastore

Use the `metastore-storage.yaml` file below as a template.

```yaml
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"
```

Replace the value for `size` with your desired capacity. The example file shows "5Gi".

### 3.4.2. 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:
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:

```yaml
spec:
  hive:
    config:
      url: "jdbc:postgresql://postgresql.example.com:5432/hive_metastore"
      driver: "org.postgresql.Driver"
      username: "REPLACE ME"
      password: "REPLACE ME"
```

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

### 3.5. 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 Operator is primarily done through your `MeteringConfig` file.

#### 3.5.1. 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. If your Prometheus instance uses a different CA, the CA can be injected through a ConfigMap. See the following example.

```yaml
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`.

The `reporting-operator` can also be configured to use a specified bearer token to auth with Prometheus. See the following example.

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

### 3.5.2. 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 side-car container for `reporting-operator`, which protects the reporting API with authentication.

#### 3.5.2.1. 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.

In order 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.

```bash
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/password.

#### 3.5.2.1.1. 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:

```bash
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.

3.5.2.1.2. Authenticate using a username and password

We are able to do basic authentication using a username and password combination, which is specified in the contents of a htpasswd file. By default, we create a secret containing an empty htpasswd data. 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, 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.

3.5.2.2. Manually Configuring Authentication

In order to manually configure, or disable OAuth in the reporting-operator, you must set spec.tls.enabled: false in your MeteringConfig.

WARNING

This also disables all TLS/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-operator 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.

d $ openssl rand -base64 32 | head -c32; echo.
3.5.2.2.1. 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 serviceAccounts 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 serviceAccount 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 RoleBindings 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.

```yaml
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:
```
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` `serviceAccounts` both have these permissions, and their tokens can be used for authentication.

### 3.5.2.2.2. Basic authentication (username/password)

For basic authentication you can supply a username and password in `reporting-operator.spec.authProxy.htpasswd.data`. 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.

#### 3.6. 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.

```yaml
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>"  
        prefix: "<path/to/report>"
```

OpenShift Container Platform 4.3 Metering
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. 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` fields. See the example secret file below for more details.

```yaml
apiVersion: v1
kind: Secret
metadata:  # Example secret: this is the name of the secret in your metering namespace
    name: <your-aws-secret>
data:
    aws-access-key-id: "dGVzdAo="
    aws-secret-access-key: "c2VjcmV0Cg=="
```

3. 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:DeleteObject",
                "s3:GetObject",
                "s3:HeadBucket",
                "s3:ListBucket",
                "s3:ListMultipartUploadParts",
```
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`. 

```json
{
   "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"
         ]
      }
   ]
}
```
4.1. ABOUT REPORTS

A Report is an API object that provides a method to manage periodic ETL (Extract Transform and Load) jobs using SQL queries. They are composed using other Metering resources such as ReportQueries, which provide the actual SQL query to run, and ReportDataSources, which are what define the data available to the ReportQueries and Reports.

Many use cases are addressed out-of-the-box with the predefined ReportQueries and ReportDataSources that come installed with metering, so you do not need to define your own unless you have a use-case not covered by what is predefined.

4.1. 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 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 will backfill the data starting where it left off. If the schedule is unset, then the Report will run once for the time specified by the reportingStart and reportingEnd. By default, reports wait for ReportDataSources to have fully imported any data covered in the reporting period. If the Report has a schedule, it will wait to run until the data in the period currently being processed has finished importing.

4.1.1. Example Report with a Schedule

The following example Report will contain information on every Pod’s CPU requests, and will run every hour, adding the last hours worth of data each time it runs.

```yaml
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
```

4.1.1.2. Example Report without a Schedule (Run-Once)

The following example Report will contain information on every Pod’s CPU requests for all of July. After completion, it does not run again.

```yaml
apiVersion: metering.openshift.io/v1
kind: Report
```
4.1.1.3. query

Names the ReportQuery 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 `oc` CLI to obtain a list of available ReportQuery objects:

```sh
$ oc -n openshift-metering get reportqueries
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
```
ReportQueries with the -raw suffix are used by other ReportQueries to build more complex queries, and should not be used directly for reports.

**namespace-** prefixed queries aggregate Pod CPU/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.

For a complete list of fields, use the **oc** CLI to get the ReportQuery as YAML, and check the **spec.columns** field:

For example, run:

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

You should see output like:

```
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
```

4.1.1.4. 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).

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

4.1.1.4.1. 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**: 

4.1.1.5. **reportingStart**

To support running a Report against existing data, you can set the `spec.reportingStart` field to a [RFC3339 timestamp](https://tools.ietf.org/html/rfc3339) to tell the Report to run according to its `schedule` starting from `reportingStart` rather than the current time. One important thing to understand is that this 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, which you wanted to be included in your Report, you could create a report with the following values:

```yaml
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"
```

4.1.1.6. **reportingEnd**

To configure a Report to only run until a specified time, you can set the `spec.reportingEnd` field to an [RFC3339 timestamp](https://tools.ietf.org/html/rfc3339). 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 `reportingEnd`, the last period in the schedule will be shortened to end at the specified `reportingEnd` time. If `reportingEnd` is left unset, then the Report will run forever, or until a `reportingEnd` is set on the Report.

For example, if you wanted to create a report that runs once a week for the month of July:
4.1.1.7. runImmediately

When `runImmediately` is set to `true`, the report will be run 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.

4.1.1.8. inputs

The `spec.inputs` field of a Report can be used to override or set values defined in a ReportQuery’s `spec.inputs` field.

It is a list of name-value pairs:

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

The `name` of an input must exist in the ReportQuery’s `inputs` list. The `value` of the input must be the correct type for the input’s `type`.

4.1.1.9. 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; instead of requiring a monthly report to query and add all data over an entire month, the task can be split into daily reports that each run over a thirtieth of the data.

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

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

```yaml
# Taken from 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: |
...
namespace,
  sum(pod_usage_cpu_core_seconds) as pod_usage_cpu_core_seconds
FROM |.Report.Inputs.NamespaceCPUUsageReportName | reportTableName |
...

# aggregated-report.yaml
spec:
  query: "namespace-cpu-usage"
  inputs:
    - name: "NamespaceCPUUsageReportName"
      value: "namespace-cpu-usage-hourly"

4.1.1.9.1. 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 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.

4.2. STORAGE LOCATIONS

A StorageLocation is a custom resource that 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 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/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.

4.2.1. StorageLocation examples
This first example is what the built-in local storage option looks like. It is configured to use Hive, and by default data is stored wherever Hive is configured to use storage (HDFS, S3, or a ReadWriteMany PVC).

**Local storage example**

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

1. If the `hive` section is present, then the StorageLocation will be configured to store data in Presto by creating the table using Hive server. Only `databaseName` and `unmanagedDatabase` are required fields.

2. The name of the database within hive.

3. If `true`, then this StorageLocation will not be actively managed, and the `databaseName` is expected to already exist in Hive. If `false`, this will cause the reporting-operator to create the database in Hive.

The next 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**

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

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 some additional optional fields that can be specified in the `hive` section:

- (optional) `defaultTableProperties`: Contains configuration options for creating tables using Hive.
- (optional) `fileFormat`: The file format used for storing files in the filesystem. See the [Hive Documentation on File Storage Format](https://prestodb.io/docs/current/hive/storage-file-format.html) for a list of options and more details.
• (optional) 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.

4.2.2. Default StorageLocation

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 StorageLocation is not specified will use the default storage resource. There can only be one default storage resource. If more than one resource with the annotation exists, an error will be logged and the operator will consider there to be no default.

```yaml
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 5. USING METERING

Prerequisites

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

5.1. 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 by using oc.

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:

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

   - The query specifies ReportQuery 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`.
   - Use a descriptive name about what the Report does for metadata.name. A good name is the query, and the schedule or period you used.
   - 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:
$ oc create -f <file-name>.yaml

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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>QUERY</th>
<th>SCHEDULE</th>
<th>RUNNING</th>
<th>FAILED</th>
<th>LAST</th>
</tr>
</thead>
</table>

### 5.2. 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. Get the route to the **reporting-api**:

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

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

   **Set reportName to the name of the Report you created.**
Set `reportFormat` to one of `csv`, `json`, or `tabular` to specify the output format of the API response.

The response should look similar to the following (example output is with `reportName=namespace-cpu-request-2019` and `reportFormat=csv`):

<table>
<thead>
<tr>
<th>period_start</th>
<th>period_end</th>
<th>namespace</th>
<th>pod_request_cpu_core_seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-apiserver</td>
<td>11745.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-apiserver-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-authentication</td>
<td>522.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-authentication-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-cloud-credential-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-cluster-machine-approver</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-cluster-node-tuning-operator</td>
<td>3385.800000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-cluster-samples-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-cluster-version</td>
<td>522.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-console</td>
<td>522.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-console-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-controller-manager</td>
<td>7830.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-dns</td>
<td>34372.800000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-dns-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-etc</td>
<td>23490.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-image-registry</td>
<td>5993.400000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-ingress</td>
<td>5220.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-ingress-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-kube-apiserver</td>
<td>12528.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-kube-apiserver-operator</td>
<td>261.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-kube-controller-manager</td>
<td>8613.000000</td>
</tr>
<tr>
<td>2019-01-01</td>
<td>2019-12-30</td>
<td>openshift-machine-api</td>
<td>1305.000000</td>
</tr>
</tbody>
</table>
CHAPTER 5. USING METERING

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 6. 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.

Prerequisites

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

6.1. 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

```yaml
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

```yaml
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.

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.

### 6.2. MEASURE CLUSTER USAGE WITH A ONE-TIME REPORT

The following Reports to measure 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**

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

1. To stay organized, remember to change the name of your Report if you change any of the other values.
2. Configures the Reports 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. This tells the Report to run immediately after saving it and applying it.

### 6.3. 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**

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

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 7. 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

7.1. 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.

7.1.1. Not enough compute resources

A common issue when installing or running metering is lack of compute resources. Ensure that metering is allocated the minimum resource requirements described in the installation prerequisites.

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

7.1.2. StorageClass not configured

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

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

7.1.3. 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.

7.2. 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.

7.2.1. Get reporting operator logs

The command below will follow the logs of the reporting-operator.

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

7.2.2. 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 run 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;

<table>
<thead>
<tr>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasource_your_namespace_cluster_cpu_capacity_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_cluster_cpu_usage_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_cluster_memory_capacity_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_cluster_memory_usage_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_node_allocatable_cpu_cores</td>
</tr>
<tr>
<td>datasource_your_namespace_node_allocatable_memory_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_node_capacity_cpu_cores</td>
</tr>
<tr>
<td>datasource_your_namespace_node_capacity_memory_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_node_cpu_allocatable_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_node_cpu_capacity_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_node_memory_allocatable_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_node_memory_capacity_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_capacity_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_capacity_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_phase</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_phase_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_request_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_request_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_usage_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_usage_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_persistentvolumeclaim_usage_with_phase_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_cpu_request_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_cpu_usage_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_limit_cpu_cores</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_limit_memory_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_memory_request_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_memory_usage_raw</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_persistentvolumeclaim_request_info</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_request_cpu_cores</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_request_memory_bytes</td>
</tr>
<tr>
<td>datasource_your_namespace_pod_usage_cpu_cores</td>
</tr>
</tbody>
</table>
7.2.3. 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.

```bash
$ 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:

```bash
$ 0: jdbc:hive2://127.0.0.1:10000/default> show tables from metering;
+---------------------------------+
| tab_name                        |
+---------------------------------+
| datasource_your_namespace_cluster_cpu_capacity_raw | 32 rows
| 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_cpu_request_info     |
| datasource_your_namespace_pod_cpu_usage_info       |
```

CHAPTER 7. TROUBLESHOOTING AND DEBUGGING METERING

43
7.2.4. Port-forward to the Hive web UI

Run the following command:

```bash
$ 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.

7.2.5. Port-forward to hdfs

To the namenode:

```bash
$ 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.

To the first datanode:

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

To check other datanodes, run the above command, replacing `hdfs-datanode-0` with the Pod you want to view information on.

7.2.6. 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.

7.2.6.1. Accessing ansible logs

In the default installation, the metering Operator is deployed as a Pod. In this case, we can check the logs of the ansible container within this Pod:

```bash
$ 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.

7.2.6.2. Checking the MeteringConfig Status

It can be helpful to view the `.status` field of your MeteringConfig custom resource to debug any recent failures. You can do this with the following command:

```bash
$ oc -n openshift-metering get meteringconfig operator-metering -o json | jq '.status'
```
8.1. UNINSTALLING METERING FROM OPENSHIFT CONTAINER PLATFORM

You can remove metering from your cluster.

Prerequisites
- Metering must be installed.

Procedure
To remove metering:

1. In the OpenShift Container Platform web console, click Operators → Installed Operators.
2. Find the Metering Operator and click the menu. Select Uninstall Operator.
3. In the dialog box, click Remove to uninstall metering.

NOTE
Metering does not manage or delete any S3 bucket data. If you used Amazon S3 for storage, any S3 buckets used to store metering data must be manually cleaned up.