OpenShift Container Platform 4.6

メータリング

OpenShift Container Platform でのメータリングの設定および使用

Last Updated: 2023-02-23

OpenShift Container Platform 4.6 メータリング

OpenShift Container Platform でのメータリングの設定および使用

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本書では、OpenShift Container Platform でメータリングを設定および使用する方法を説明します。

目次

第1章 メータリング

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

1.1. メータリングの概要

メータリングは、異なるデータソースからデータを処理するためのレポートを作成できる汎用のデータ分析ツールです。クラスター管理者として、メータリングを使用してクラスターの内容を分析できます。独自のクエリーを作成するか、または事前定義 SQL クエリーを使用して、利用可能な異なるデータソースからデータを処理する方法を定義できます。

メータリングは主にデフォルトデータとして Prometheus を使用するクラスター内のメトリクスデータにフォーカスを置き、メータリングのユーザーが Pod、namespace、および他のほとんどの Kubernetes リソースについてのレポートを行えるようにします。

メータリングは OpenShift Container Platform 4.x クラスター以降にインストールできます。

1.1.1. メータリングのインストール

メータリングは、CLI および Web コンソールを使用して OpenShift Container Platform 4.x 以降にインストールできます。詳細は、メータリングのインストール について参照してください。

1.1.2. メータリングのアップグレード

メータリングは、メータリング Operator サブスクリプションを更新してアップグレードできます。以下のタスクを確認します。

- **MeteringConfig** カスタムリソースはメータリングのインストールについてのすべての設定の詳細を指定します。メータリングスタックを最初にインストールすると、デフォルトの **MeteringConfig** カスタムリソースが生成されます。このデフォルトファイルを変更するには、ドキュメントのサンプルを使用します。
- Report カスタムリソースは、SQL クエリーを使用して定期的な ETL (Extract Transform および Load) ジョブを管理する方法を提供します。レポートは、実行する実際の SQL クエリーを提供する **ReportQuery** リソースや、**ReportQuery** および **Report** リソースで利用できるデータを定義する **ReportDataSource** リソースなどの他のメータリングリソースで設定されます。

1.1.3. メータリングの使用

メータリングを使用してレポートを作成し、レポート結果を表示できます。詳細は、メータリングの使用例 を参照してください。

1.1.4. メータリングのトラブルシューティング

以下のセクションを使用して、メータリングに関する特定の問題のトラブルシューティングを行うことができます。

- 十分なコンピュートリソースがない
- **StorageClass** リソースが設定されていない
- シークレットが正しく設定されていない

1.1.5. メータリングのデバッグ

以下のセクションを使用して、メータリングの特定の問題をデバッグ できます。

- レポート Operator ログの取得
- presto-cli を使用した Presto のクエリー
- beeline を使用した Hive のクエリー
- Hive Web UI へのポート転送
- HDFS へのポート転送
- メータリング Ansible Operator

1.1.6. メータリングのアンインストール

メータリングリソースを OpenShift Container Platform クラスターから削除し、これをクリーンアップすることができます。詳細は、メータリングのアンインストール について参照してください。

1.1.7. メータリングリソース

メータリングには、メータリングのデプロイメントやインストール、およびメータリングが提供するレポート機能を管理するために使用できるリソースが多数含まれています。

メータリングは以下のカスタムリソース定義 (CRD) を使用して管理されます。

Metering Config	デプロイメントのメータリングスタックを設定します。メータリングスタックを設定する各コンポーネントを制御するカスタマイズおよび設定オプションが含まれます。
レポート	使用するクエリー、クエリーを実行するタイミングおよび頻度、および結果を保存する場所を制御します。
レポートクエリー	**ReportDataSource** 内に含まれるデータに対して分析を実行するために使用される SQL クエリーが含まれます。
ReportDataSource	**ReportQuery** および **Report** で利用可能なデータを制御します。メータリング内で使用できるように複数の異なるデータベースへのアクセスの設定を可能にします。

第2章 メータリングのインストール

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

メータリングをクラスターにインストールする前に、以下のセクションを確認します。

メータリングのインストールを開始するには、まず OperatorHub からメータリング Operator をインストールします。次に、**MeteringConfig** カスタムリソース (CR) を作成してメータリングのインスタンスを設定します。メータリング Operator をインストールすると、ドキュメントのサンプルを使用して変更できるデフォルトの **MeteringConfig** リソースが作成されます。**MeteringConfig** リソースを作成したら、メータリングスタックをインストールします。最後に、インストールを検証します。

2.1. 前提条件

メータリングには、以下のコンポーネントが必要です。

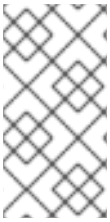
- ボリュームの動的プロビジョニング用の **StorageClass**。メータリングは、数多くの異なるストレージソリューションをサポートします。
- 4GB メモリー、4 CPU コアが利用できるクラスター容量と、2 CPU コアと 2GB メモリーの容量を持つ 1 つ以上のノード。
- メータリングによってインストールされている最大規模の単一 Pod に必要な最小リソースは 2GB のメモリーと 2 CPU コアです。
    - メモリーおよび CPU の消費量はこれより低くなることがありますが、レポートの実行時や大規模なクラスターのデータの収集時には、消費量は急上昇します。

2.2. メータリング OPERATOR のインストール

メータリングは、メータリング Operator をデプロイしてインストールできます。メータリング Operator はメータリングスタックのコンポーネントを作成し、管理します。

注記

Web コンソールまたは CLI の **oc new-project** コマンドを使用して、**openshift-** で始まる Project を作成することはできません。



注記

メータリング Operator が **openshift-metering** 以外の namespace を使用してインストールされている場合、メータリングレポートは CLI の使用によってのみ表示できます。この実行は、**openshift-metering** namespace を使用するために、インストール手順全体で強く推奨されます。

2.2.1. Web コンソールでのメータリングのインストール

OpenShift Container Platform Web コンソールを使ってメータリング Operator をインストールすることができます。

手順

1. **oc create -f <file-name>.yaml** コマンドで、メータリング Operator の namespace オブジェクト YAML ファイルを作成します。CLI を使用して namespace を作成する必要があります。たとえば、**metering-namespace.yaml** のようになります。

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

(1) メータリングを **openshift-metering** namespace にデプロイすることを強く推奨します。

(2) オペランド Pod の特定のノードセレクターを設定する前に、このアノテーションを追加します。

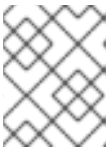
2. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。**metering** のフィルターで、メータリング Operator を検索します。

3. **Metering** カードをクリックして、パッケージの説明を確認してから **Install** をクリックします。

4. **Update Channel**、**Installation Mode**、および **Approval Strategy** を選択します。

5. **Install** をクリックします。

6. **Operators → Installed Operators** ページに切り替えて、メータリング Operator がインストールされていることを確認します。メータリング Operator では、インストールの完了時に **Status** が **Succeeded** になります。



注記

メータリング Operator が表示されるまでに数分の時間がかかる場合があります。

7. **Installed Operators** ページで **Metering** をクリックし、Operator **Details** を確認します。**Details** ページから、メータリングに関連する異なるリソースを作成できます。

メータリングのインストールを完了するには、メータリングを設定し、メータリングスタックのコンポーネントをインストールできるように **MeteringConfig** リソースを作成します。

2.2.2. CLI を使用したメータリングのインストール

OpenShift Container Platform CLI を使用して、メータリング Operator をインストールできます。

手順

1. メータリング Operator の **Namespace** オブジェクト YAML ファイルを作成します。CLI を使用して namespace を作成する必要があります。たとえば、**metering-namespace.yaml** のようになります。

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

(1) メータリングを **openshift-metering** namespace にデプロイすることを強く推奨します。

(2) オペランド Pod の特定のノードセレクターを設定する前に、このアノテーションを追加します。

2. **Namespace** オブジェクトを作成します。

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

以下は例になります。

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

3. **OperatorGroup** オブジェクト YAML ファイルを作成します。たとえば、**metering-og** のようになります。

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

(1) 名前は任意です。

(2) **openshift-metering** namespace を指定します。

4. **Subscription** オブジェクトの YAML ファイルを作成し、namespace をメータリング Operator にサブスクライブします。このオブジェクトは、**redhat-operators** カタログソースの最近リリースされたバージョンをターゲットにします。たとえば、**metering-sub.yaml** のようになります。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metering-ocp (1)
```

```
  namespace: openshift-metering (2)
spec:
  channel: "4.6" (3)
  source: "redhat-operators" (4)
  sourceNamespace: "openshift-marketplace"
  name: "metering-ocp"
  installPlanApproval: "Automatic" (5)
```

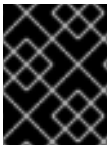
1. 名前は任意です。
2. **openshift-logging** namespace を指定する必要があります。
3. 4.6 をチャネルとして指定します。
4. **metering-ocp** パッケージマニフェストが含まれる、**redhat-operators** カタログソースを指定します。OpenShift Container Platform が、非接続クラスターとも呼ばれる制限されたネットワークにインストールされている場合、Operator LifeCycle Manager (OLM) の設定時に作成した **CatalogSource** オブジェクトの名前を指定します。
5. 自動インストール計画の承認を指定します。

2.3. メータリングスタックのインストール

メータリング Operator をクラスターに追加した後に、メータリングスタックをインストールしてメータリングのコンポーネントをインストールできます。

2.4. 前提条件

- 設定オプション を確認します。
- **MeteringConfig** リソースを作成します。以下のプロセスを開始し、デフォルトの **MeteringConfig** リソースを生成し、ドキュメントのサンプルを使用して特定のインストール用にこのデフォルトファイルを変更します。以下のトピックを参照して、**MeteringConfig** リソースを作成します。
  - 設定オプションについては、メータリングの設定について を参照してください。
  - 少なくとも、永続ストレージを設定 し、Hive メタストアを設定 する必要があります。



重要

openshift-metering namespace には、1 つの **MeteringConfig** リソースのみを配置できます。その他の設定はサポートされません。

手順

1. Web コンソールから、**openshift-metering** プロジェクトのメータリング Operator についての **Operator Details** ページにいることを確認します。**Operators → Installed Operators** をクリックしてこのページに移動してから、メータリング Operator を選択します。
2. **Provided APIs** の下で、メータリング設定カードの **Create Instance** をクリックします。これにより、YAML エディターがデフォルトの **MeteringConfig** リソースファイルと共に開き、ここで設定を定義できます。

注記

設定ファイルやサポートされるすべての設定オプションの例については、メータリングの設定についてのドキュメント を参照してください。

3. **MeteringConfig** リソースを YAML エディターに入力し、**Create** をクリックします。

MeteringConfig リソースは、メータリングスタックに必要なリソースの作成を開始します。これで、インストールを検証できるようになります。

2.5. メータリングインストールの確認

以下のチェックのいずれかを実行してメータリングのインストールを確認することができます。

- メータリングのバージョンについて、メータリング Operator の **ClusterServiceVersion** (CSV) リソースを確認します。これは、Web コンソールまたは CLI のいずれかで実行できます。

  手順 (UI)

  1. **openshift-metering** namespace の **Operators → Installed Operators** に移動します。
  2. **Metering Operator** をクリックします。
  3. **Subscription Details** の **Subscription** をクリックします。
  4. **Installed Version** を確認します。

  手順 (CLI)

  - **openshift-metering** namespace でメータリング Operator CSV を確認します。

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

    出力例

    ```
    NAME                         DISPLAY     VERSION   REPLACES
    PHASE
    metering-operator.v4.6.0     Metering    4.6.0
    Succeeded
    ```

- **openshift-metering** namespace のすべての必要な Pod が作成されていることを確認します。これは、Web コンソールまたは CLI のいずれかで実行できます。

  注記

  多くの Pod は、それらが準備状態にあると見なされる前に機能するために他のコンポーネントに依存する必要があります。他の Pod の起動に時間がかかりすぎる場合、一部の Pod は再起動する可能性があります。これはメータリング Operator のインストール時に予想されます。

  手順 (UI)

- メータリング namespace で **Workloads → Pods** に移動し、Pod が作成されていることを確認します。これには、メータリングスタックをインストールしてから数分の時間がかかることがあります。

手順 (CLI)

- **openshift-metering** namespace のすべての必要な Pod が作成されていることを確認します。

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

出力例

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

- **ReportDataSource** リソースが新規データをインポートし、**EARLIEST METRIC** 列の有効なタイムスタンプによって示唆されていることを確認します。これは数分の時間がかかる可能性があります。データをインポートしない-raw**ReportDataSource** リソースを除外します。

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

出力例

```
NAME                                  EARLIEST METRIC        NEWEST METRIC          IMPORT
START          IMPORT END            LAST IMPORT TIME       AGE
node-allocatable-cpu-cores            2019-08-05T16:52:00Z   2019-08-05T18:52:00Z
2019-08-05T16:52:00Z   2019-08-05T18:52:00Z   2019-08-05T18:54:45Z   9m50s
node-allocatable-memory-bytes         2019-08-05T16:51:00Z   2019-08-05T18:51:00Z
2019-08-05T16:51:00Z   2019-08-05T18:51:00Z   2019-08-05T18:54:45Z   9m50s
node-capacity-cpu-cores               2019-08-05T16:51:00Z   2019-08-05T18:29:00Z
2019-08-05T16:51:00Z   2019-08-05T18:29:00Z   2019-08-05T18:54:39Z   9m50s
node-capacity-memory-bytes            2019-08-05T16:52:00Z   2019-08-05T18:41:00Z
2019-08-05T16:52:00Z   2019-08-05T18:41:00Z   2019-08-05T18:54:44Z   9m50s
persistentvolumeclaim-capacity-bytes  2019-08-05T16:51:00Z   2019-08-05T18:29:00Z
2019-08-05T16:51:00Z   2019-08-05T18:29:00Z   2019-08-05T18:54:43Z   9m50s
persistentvolumeclaim-phase           2019-08-05T16:51:00Z   2019-08-05T18:29:00Z
2019-08-05T16:51:00Z   2019-08-05T18:29:00Z   2019-08-05T18:54:28Z   9m50s
persistentvolumeclaim-request-bytes   2019-08-05T16:52:00Z   2019-08-05T18:30:00Z
2019-08-05T16:52:00Z   2019-08-05T18:30:00Z   2019-08-05T18:54:34Z   9m50s
persistentvolumeclaim-usage-bytes     2019-08-05T16:52:00Z   2019-08-05T18:30:00Z
2019-08-05T16:52:00Z   2019-08-05T18:30:00Z   2019-08-05T18:54:36Z   9m49s
pod-limit-cpu-cores                   2019-08-05T16:52:00Z   2019-08-05T18:30:00Z   2019-
08-05T16:52:00Z   2019-08-05T18:30:00Z   2019-08-05T18:54:26Z   9m49s
pod-limit-memory-bytes                2019-08-05T16:51:00Z   2019-08-05T18:40:00Z   2019-
08-05T16:51:00Z   2019-08-05T18:40:00Z   2019-08-05T18:54:30Z   9m49s
pod-persistentvolumeclaim-request-info  2019-08-05T16:51:00Z   2019-08-05T18:40:00Z
2019-08-05T16:51:00Z   2019-08-05T18:40:00Z   2019-08-05T18:54:37Z   9m49s
pod-request-cpu-cores                 2019-08-05T16:51:00Z   2019-08-05T18:18:00Z   2019-
08-05T16:51:00Z   2019-08-05T18:18:00Z   2019-08-05T18:54:24Z   9m49s
```

```
pod-request-memory-bytes                2019-08-05T16:52:00Z   2019-08-05T18:08:00Z
2019-08-05T16:52:00Z   2019-08-05T18:08:00Z   2019-08-05T18:54:32Z   9m49s
pod-usage-cpu-cores                     2019-08-05T16:52:00Z   2019-08-05T17:57:00Z   2019-
08-05T16:52:00Z   2019-08-05T17:57:00Z   2019-08-05T18:54:10Z   9m49s
pod-usage-memory-bytes                  2019-08-05T16:52:00Z   2019-08-05T18:08:00Z
2019-08-05T16:52:00Z   2019-08-05T18:08:00Z   2019-08-05T18:54:20Z   9m49s
```

すべての Pod が準備状態にあり、データがインポートされていることを確認したら、メータリングを使用してクラスターについてのデータを収集し、報告することができます。

2.6. 追加リソース

- 設定手順および利用可能なストレージプラットフォームについての詳細は、永続ストレージの設定 を参照してください。
- Hive を設定する手順については、Hive メタストアの設定 を参照してください。

第3章 メータリングのアップグレード

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

メータリングを 4.6 にアップグレードするには、メータリング Operator サブスクリプションを更新します。

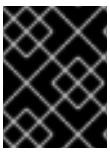
3.1. 前提条件

- クラスターは 4.6 に更新されます。
- メータリング Operator は OperatorHub からインストールされます。

注記

メータリング Operator を 4.6 に手動でアップグレードする必要があります。以前のインストールで Automatic **承認ストラテジー** を選択した場合は、メータリングは自動的にアップグレードされません。

- MeteringConfig カスタムリソース が設定されている。
- メータリングスタック がインストール済みです。
- すべての Pod が準備状態にあることを確認して、メータリングのステータスが正常であることを確認する。



重要

メータリングのインストールまたはアップグレード後にメータリングストレージ設定を変更すると、データ損失が発生する可能性があります。

手順

1. Web コンソールで **Operators → Installed Operators** をクリックします。
2. **openshift-metering** プロジェクトを選択します。
3. **Metering Operator** をクリックします。
4. **Subscription → Channel** をクリックします。
5. **Change Subscription Update Channel** ウィンドウで **4.6** を選択し、**Save** をクリックします。

注記

次のステップに進む前に、サブスクリプションの更新が許可されるまで数秒待機します。

6. **Operators → Installed Operators** をクリックします。
メータリング Operator は 4.6 と表示されます。以下に例を示します。

```
Metering
4.6.0-202007012112.p0 provided by Red Hat, Inc
```

検証

以下のチェックのいずれかを実行してメータリングのアップグレードを確認することができます。

- 新規メータリングバージョンについて、メータリング Operator のクラスターサービスバージョン (CSV) を確認します。これは、Web コンソールまたは CLI のいずれかで実行できます。

手順 (UI)

1. メータリング namespace の **Operators → Installed Operators** に移動します。
2. **Metering Operator** をクリックします。
3. **Subscription Details** の **Subscription** をクリックします。
4. アップグレードしたメータリングバージョンの **Installed Version** を確認します。**Starting Version** には、アップグレード前のメータリングバージョンが表示されます。

手順 (CLI)

- メータリング Operator CSV を確認します。

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

4.5 から 4.6 へのメータリングアップグレードの出力例

```
NAME                                 DISPLAY     VERSION              REPLACES
PHASE
metering-operator.4.6.0-202007012112.p0   Metering    4.6.0-202007012112.p0
metering-operator.4.5.0-202005252114   Succeeded
```

- **openshift-metering** namespace のすべての必要な Pod が作成されていることを確認します。これは、Web コンソールまたは CLI のいずれかで実行できます。

注記

多くの Pod は、それらが準備状態にあると見なされる前に機能するために他のコンポーネントに依存する必要があります。他の Pod の起動に時間がかかりすぎる場合、一部の Pod は再起動する可能性があります。これはメータリング Operator のアップグレード時に予想されます。

手順 (UI)

- メータリング namespace で **Workloads → Pods** に移動し、Pod が作成されていることを確認します。これには、メータリングスタックをアップグレードしてから数分の時間がかかることがあります。

手順 (CLI)

- **openshift-metering** namespace のすべての必要な Pod が作成されていることを確認します。

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

出力例

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

- **ReportDataSource** リソースが新規データをインポートし、**NEWEST METRIC** 列の有効なタイムスタンプによって示唆されていることを確認します。これは数分の時間がかかる可能性があります。データをインポートしない-raw**ReportDataSource** リソースを除外します。

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

NEWEST METRIC 列のタイムスタンプは、**ReportDataSource** が新規データのインポートを開始していることを示します。

出力例

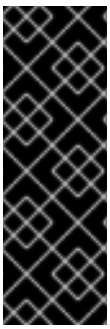
```
NAME                                 EARLIEST METRIC       NEWEST METRIC         IMPORT
START          IMPORT END            LAST IMPORT TIME      AGE
node-allocatable-cpu-cores           2020-05-18T21:10:00Z  2020-05-19T19:52:00Z
2020-05-18T19:11:00Z  2020-05-19T19:52:00Z  2020-05-19T19:56:44Z 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

```
2020-05-18T19:11:00Z   2020-05-19T19:52:00Z   2020-05-19T19:52:07Z   23h
pod-request-cpu-cores                  2020-05-18T21:10:00Z   2020-05-19T19:58:00Z   2020-
05-18T19:11:00Z   2020-05-19T19:58:00Z   2020-05-19T19:58:57Z   23h
pod-request-memory-bytes               2020-05-18T21:10:00Z   2020-05-19T19:52:00Z
2020-05-18T19:11:00Z   2020-05-19T19:52:00Z   2020-05-19T19:55:32Z   23h
pod-usage-cpu-cores                    2020-05-18T21:09:00Z   2020-05-19T19:52:00Z   2020-
05-18T19:11:00Z   2020-05-19T19:52:00Z   2020-05-19T19:54:55Z   23h
pod-usage-memory-bytes                 2020-05-18T21:08:00Z   2020-05-19T19:52:00Z
2020-05-18T19:11:00Z   2020-05-19T19:52:00Z   2020-05-19T19:55:00Z   23h
report-ns-pvc-usage
5h36m
report-ns-pvc-usage-hourly
```

すべての Pod が準備状態にあり、データがインポートされていることを確認したら、メータリングは継続してデータを収集し、クラスターについて報告します。以前に スケジュールされたレポート を確認するか、または 1 回実行されるメータリングレポート を作成してメータリングのアップグレードを確認します。

第4章 メータリングの設定

4.1. メータリングの設定について



重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

MeteringConfig カスタムリソースはメータリングのインストールについてのすべての設定の詳細を指定します。メータリングスタックを最初にインストールすると、デフォルトの **MeteringConfig** カスタムリソースが生成されます。このデフォルトファイルを変更するには、ドキュメントのサンプルを使用します。以下の重要な点に留意してください。

- 少なくとも、永続ストレージを設定 し、Hive メタストアを設定 する必要があります。
- デフォルト設定のほとんどは機能しますが、大規模なデプロイメントまたは高度にカスタマイズされたデプロイメントの場合は、すべての設定オプションを注意して確認する必要があります。
- いくつかの設定オプションは、インストール後に変更することができません。

インストール後に変更可能な設定オプションについては、**MeteringConfig** カスタムリソースで変更し、ファイルを再度適用します。

4.2. 一般的な設定オプション

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

4.2.1. リソース要求および制限

Pod およびボリュームの CPU、メモリー、またはストレージリソースの要求および/または制限を調整できます。以下の **default-resource-limits.yaml** は、各コンポーネントのリソース要求および制限を設定する例を示しています。

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

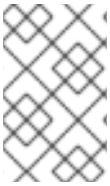
```
reporting-operator:
  spec:
    resources:
      limits:
        cpu: 1
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
presto:
  spec:
    coordinator:
      resources:
        limits:
          cpu: 4
          memory: 4Gi
        requests:
          cpu: 2
          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. ノードセレクター

特定のノードセットでメータリングコンポーネントを実行できます。メータリングコンポーネントに **nodeSelector** を設定し、コンポーネントがスケジュールされる場所を制御します。以下の **node-selectors.yaml** ファイルは、各コンポーネントのノードセレクターを設定する例を示しています。



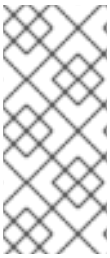
注記

オペランド Pod の特定のノードセレクターを設定する前に、**openshift.io/node-selector: ""** namespace アノテーションをメータリング namespace YAML ファイルに追加します。**""** をアノテーションの値として指定します。

```
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 適切な値が設定された **nodeSelector** パラメーターを、移動する必要のあるコンポーネントに追加します。表示されている形式の **nodeSelector** を使用することも、ノードに指定された値に基づいてキーと値のペアを使用することもできます。



注記

オペランド Pod の特定のノードセレクターを設定する前に、**openshift.io/node-selector: ""** namespace アノテーションをメータリング namespace YAML ファイルに追加します。**openshift.io/node-selector** アノテーションがプロジェクトに設定されている場合、その値はクラスター全体の **Scheduler** オブジェクトの **spec.defaultNodeSelector** フィールドの値に優先して使用されます。

検証

以下のチェックのいずれかを実行してメータリングノードセレクターを検証できます。

- メータリングのすべての Pod が **MeteringConfig** カスタムリソースで設定されるノードの IP に適切にスケジュールされていることを確認します。

  1. **openshift-metering** namespace のすべての Pod を確認します。

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

     出力には、**openshift-metering** namespace で実行される各 Pod の **NODE** および対応する **IP** が表示されます。

     出力例

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

  2. **openshift-metering** namespace のノードを、クラスター内の各ノードの **NAME** と比較します。

     ```
     $ oc get nodes
     ```

     出力例

     ```
     NAME                                         STATUS   ROLES    AGE   VERSION
     ip-10-0-147-106.us-east-2.compute.internal   Ready    master   14h   v1.19.0+6025c28
     ip-10-0-150-175.us-east-2.compute.internal   Ready    worker   14h   v1.19.0+6025c28
     ip-10-0-175-23.us-east-2.compute.internal    Ready    master   14h   v1.19.0+6025c28
     ip-10-0-189-6.us-east-2.compute.internal     Ready    worker   14h   v1.19.0+6025c28
     ip-10-0-205-158.us-east-2.compute.internal   Ready    master   14h   v1.19.0+6025c28
     ip-10-0-210-167.us-east-2.compute.internal   Ready    worker   14h   v1.19.0+6025c28
     ```

- **MeteringConfig** カスタムリソースのノードセレクターの設定が、メータリングオペランド Pod がスケジュールされないようにクラスター全体のノードセレクター設定に干渉しないことを確認します。

  - クラスター全体の **Scheduler** オブジェクトで **spec.defaultNodeSelector** フィールドを確認します。ここには、デフォルトで Pod がスケジュールされている場所が示されます。

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

4.3. 永続ストレージの設定

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

メータリングでは、メータリング Operator によって収集されるデータを永続化し、レポートの結果を保存するための永続ストレージが必要です。数多くの異なるストレージプロバイダーおよびストレージ形式がサポートされています。ストレージプロバイダーを選択し、設定ファイルのサンプルを変更して、メータリングのインストール用に永続ストレージを設定します。

4.3.1. Amazon S3 でのデータの保存

メータリングは既存の Amazon S3 バケットを使用するか、またはストレージのバケットを作成できます。

注記

メータリングは S3 バケットデータを管理したり、削除したりしません。メータリングデータを保存するために使用される S3 バケットを手動でクリーンアップする必要があります。

手順

1. **s3-storage.yaml** ファイルの **spec.storage** セクションを編集します。

   例: s3-storage.yaml ファイル

```
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. データを格納するバケットの名前を指定します。オプション: バケット内でパスを指定します。
   2. バケットのリージョンを指定します。

3 **data.aws-access-key-id** および **data.aws-secret-access-key** フィールドに AWS 認証情報を含むメータリング namespace のシークレットの名前。詳細は、以下の **Secret** オブ

4 既存の S3 バケットを指定する必要がある場合や、**CreateBucket** パーミッションを持つ IAM 認証情報を指定する必要がない場合は、このフィールドを **false** に設定します。

2. 以下の **Secret** オブジェクトをテンプレートとして使用します。

AWS Secret オブジェクトの例

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

注記

aws-access-key-id および **aws-secret-access-key** の値は base64 でエンコードされる必要があります。

3. シークレットを作成します。

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

注記

このコマンドは、**aws-access-key-id** と **aws-secret-access-key** の値を自動的に base64 でエンコードします。

aws-access-key-id および **aws-secret-access-key** 認証情報には、バケットへの読み取りおよび書き込みアクセスがなければなりません。以下の **aws/read-write.json** ファイルは、必要なパーミッションを付与する IAM ポリシーを示しています。

aws/read-write.json ファイルの例

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

```
            "s3:ListMultipartUploadParts",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::operator-metering-data/*",
            "arn:aws:s3:::operator-metering-data"
        ]
    }
  ]
}
```

spec.storage.hive.s3.createBucket を **true** に設定しているか、または **s3-storage.yaml** ファイルで未設定にされている場合、バケットの作成および削除のためのパーミッションが含まれる **aws/read-write-create.json** ファイルを使用する必要があります。

aws/read-write-create.json ファイルの例

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

4.3.2. S3 互換ストレージへのデータの保存

Noobaa などの S3 互換ストレージを使用できます。

手順

1. **s3-compatible-storage.yaml** ファイルで **spec.storage** セクションを編集します。

   例: s3-compatible-storage.yaml ファイル

   ```
   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. S3 互換バケットの名前を指定します。
2. ストレージのエンドポイントを指定します。
3. **data.aws-access-key-id** および **data.aws-secret-access-key** フィールドに AWS 認証情報を含むメータリング namespace のシークレットの名前。詳細は、以下の **Secret** オブジェクトのサンプルを参照してください。

2. 以下の **Secret** オブジェクトをテンプレートとして使用します。

S3 と互換性のある Secret オブジェクトの例

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

4.3.3. Microsoft Azure へのデータの保存

Azure Blob ストレージにデータを保存するには、既存のコンテナーを使用する必要があります。

手順

1. **azure-blob-storage.yaml** ファイルで **spec.storage** セクションを編集します。

azure-blob-storage.yaml ファイルの例

```
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. コンテナー名を指定します。
2. シークレットをメータリング namespace に指定します。詳細は、以下の **Secret** オブジェクトのサンプルを参照してください。
3. オプション: データを格納するディレクトリーを指定します。

2. 以下の **Secret** オブジェクトをテンプレートとして使用します。

Azure Secret オブジェクトの例

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

3. シークレットを作成します。

```
$ 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. Google Cloud Storage へのデータの保存

Google Cloud Storage にデータを保存するには、既存のバケットを使用する必要があります。

手順

1. **gcs-storage.yaml** ファイルで **spec.storage** セクションを編集します。

gcs-storage.yaml ファイルの例

```
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. バケットの名前を指定します。オプションで、データを保存するバケット内でディレクトリーを指定することができます。
2. シークレットをメータリング namespace に指定します。詳細は、以下の **Secret** オブジェクトのサンプルを参照してください。

2. 以下の **Secret** オブジェクトをテンプレートとして使用します。

Google Cloud Storage Secret オブジェクトの例

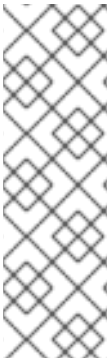
```
apiVersion: v1
kind: Secret
metadata:
  name: my-gcs-secret
data:
  gcs-service-account.json: "c2VjcmV0Cg=="
```

3. シークレットを作成します。

```
$ 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. 共有ボリュームへのデータの保存

メータリングはデフォルトでストレージを設定しません。ただし、メータリングストレージ用に ReadWriteMany 永続ボリューム (PV) または ReadWriteMany PV をプロビジョニングするすべてのストレージクラスを使用できます。



注記

NFS を実稼働環境で使用することは推奨されません。RHEL の NFS サーバーをストレージバックエンドとして使用すると、メータリングの要件を満たせず、メータリング Operator が適切に機能するために必要なパフォーマンスを出せない可能性があります。

marketplace の他の NFS 実装にはこれらの問題が検出されない可能性があります (Parallel Network File System (pNFS) など)。pNFS は分散および並列機能を持つ NFS 実装です。OpenShift Container Platform コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

手順

1. ストレージに ReadWriteMany 永続ボリュームを使用するには、**shared-storage.yaml** ファイルを変更します。

shared-storage.yaml ファイルの例

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

```
  hive:
    type: "sharedPVC"
    sharedPVC:
      claimName: "metering-nfs" 1
      # Uncommment the lines below to provision a new PVC using the specified storageClass.
2
      # createPVC: true
      # storageClass: "my-nfs-storage-class"
      # size: 5Gi
```

以下のいずれかの設定オプションを選択します。

1 **storage.hive.sharedPVC.claimName** を既存の ReadWriteMany 永続ボリューム要求 (PVC) の名前に設定します。この設定は、動的ボリュームプロビジョニングがない場合や、永続ボリュームの作成方法をより詳細に制御する必要がある場合に必要です。

2 **storage.hive.sharedPVC.createPVC** を **true** に設定し、**storage.hive.sharedPVC.storageClass** を ReadWriteMany アクセスモードのストレージクラスの名前に設定します。この設定は、動的ボリュームプロビジョニングを使用して、ボリュームを自動的に作成します。

2. メータリング用に NFS サーバーをデプロイするために必要な以下のリソースオブジェクトを作成します。**oc create -f <file-name>.yaml** コマンドを使用してオブジェクト YAML ファイルを作成します。

   a. **PersistentVolume** リソースオブジェクトを設定します。

   nfs_persistentvolume.yaml ファイルの例

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

1 **[kind: StorageClass].metadata.name** フィールドの値に一致する必要があります。

   b. **Pod** リソースオブジェクトを **nfs-server** ロールで設定します。

   nfs_server.yaml ファイルの例

```
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 NFS サーバーイメージをインストールします。

c. **Service** リソースオブジェクトを **nfs-server** ロールで設定します。

nfs_service.yaml ファイルの例:

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

d. **StorageClass** リソースオブジェクトを設定します。

nfs_storageclass.yaml ファイルの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-server 1
provisioner: example.com/nfs
```

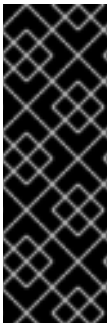
```
parameters:
  archiveOnDelete: "false"
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 **[kind: PersistentVolume].spec.storageClassName** フィールドの値に一致する必要があります。

警告

NFS ストレージおよび関連するリソースオブジェクトの設定は、メータリングストレージに使用する NFS サーバーイメージによって異なります。

4.4. HIVE メタストアの設定



重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

Hive メタストアは、Presto および Hive で作成されるデータベーステーブルに関するすべてのメタデータを保管します。デフォルトで、メタストアはこの情報を、Pod に割り当てられる永続ボリュームのローカルの組み込み Derby データベースに保管します。

通常、Hive メタストアのデフォルト設定は小規模なクラスターで機能しますが、ユーザーは Hive メタストアデータを格納するための専用の SQL データベースを使用することで、クラスターのパフォーマンスを改善したり、ストレージ要件の一部をクラスターから外したりできます。

4.4.1. 永続ボリュームの設定

デフォルトで、Hive が動作するために 1 つの永続ボリュームが必要になります。

hive-metastore-db-data は、デフォルトで必要となる主な永続ボリューム要求 (PVC) です。この PVC は Hive メタストアによって、テーブル名、列、場所などのテーブルに関するメタデータを保存するために使用されます。Hive メタストアは、Presto および Hive サーバーによって、クエリーの処理時にテーブルメタデータを検索するために使用されます。この要件は、Hive メタストアデータベースに MySQL または PostgreSQL を使用することで削除できます。

インストールするには、Hive メタストアでストレージクラスを使用して動的ボリュームプロビジョニングを有効にし、適切なサイズの永続ボリュームを手動で事前に作成するか、または既存の MySQL または PostgreSQL データベースを使用する必要があります。

4.4.1.1. Hive メタストア用のストレージクラスの設定

hive-metastore-db-data 永続ボリューム要求にストレージクラスを設定し、指定するには、ストレージクラスを **MeteringConfig** カスタムリソースに指定します。以下は、**class** フィールドが **metastore-storage.yaml** ファイルに含まれた **storage** セクションのサンプルになります。

```
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. この行のコメントを解除し、**null** を使用するストレージクラスの名前に置き換えます。値 **null** をそのままにすると、メータリングはクラスターのデフォルトのストレージクラスを使用します。

4.4.1.2. Hive メタストアのボリュームサイズの設定

以下の **metastore-storage.yaml** ファイルをテンプレートとして使用し、Hive メタストアのボリュームサイズを設定します。

```
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. **size** の値を必要な容量に置き換えます。このサンプルファイルは "5Gi" を示しています。

4.4.2. Hive メタストアに MySQL または PostgreSQL を使用する

メータリングのデフォルトインストールは、Hive を Derby という組み込み Java データベースを使用するすように設定します。これは大規模な環境には適していませんが、MySQL または PostgreSQL データベースのいずれかに置き換えることができます。デプロイメントで Hive に MySQL または PostgreSQL データベースが必要な場合は、以下の設定ファイルのサンプルを使用します。

3 つの設定オプションを使用して、Hive メタストアで使用されるデータベースを制御できます (**url**、**driver**、および **secretName**)。

ユーザー名とパスワードで MySQL または Postgres インスタンスを作成します。次に、OpenShift CLI

または YAML ファイルを使用してシークレットを作成します。このシークレット用に作成する secretName は、MeteringConfig リソースの spec.hive.spec.config.db.secretName フィールドにマップする必要があります。

OpenShift CLI でシークレットを作成するには、以下のコマンドを使用できます。

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

YAML ファイルを使用してシークレットを作成するには、以下のサンプルファイルを使用します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <YOUR_SECRETNAME> 1
data:
  username: <BASE64_ENCODED_DATABASE_USERNAME> 2
  password: <BASE64_ENCODED_DATABASE_PASSWORD> 3
```

1. シークレットの名前。
2. base64 でエンコードされたデータベースのユーザー名。
3. base64 でエンコードされたデータベースのパスワード。

以下の設定ファイルのサンプルを使用して、Hive に MySQL データベースを使用します。

```
spec:
  hive:
    spec:
      metastore:
        storage:
          create: false
      config:
        db:
          url: "jdbc:mysql://mysql.example.com:3306/hive_metastore"
          driver: "com.mysql.jdbc.Driver"
          secretName: "REPLACEME" 1
```

1. base64 で暗号化されたユーザー名およびパスワードのデータベース認証情報が含まれるシークレットの名前。

spec.hive.config.url を使用して追加の JDBC パラメーターを渡すことができます。詳細は MySQL Connector/J のドキュメント を参照してください。

以下の設定ファイルのサンプルを使用して、Hive に PostgreSQL データベースを使用します。

```
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"
```

URL を使用して追加の JDBC パラメーターを渡すことができます。詳細は、PostgreSQL JDBC ドライバーのドキュメント を参照してください。

4.5. レポート OPERATOR の設定

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

レポート Operator は、Prometheus からデータを収集し、メトリクスを Presto に保存して、Presto に対してレポートクエリーを実行し、それらの結果を HTTP API 経由で公開します。レポート Operator の設定は主に **MeteringConfig** カスタムリソースで実行されます。

4.5.1. Prometheus 接続のセキュリティー保護

メータリングを OpenShift Container Platform にインストールする場合、Prometheus は https://prometheus-k8s.openshift-monitoring.svc:9091/ で利用できます。

Prometheus への接続のセキュリティーを保護するために、デフォルトのメータリングのインストールでは OpenShift Container Platform の認証局 (CA) を使用します。Prometheus インスタンスが別の CA を使用する場合、CA は設定マップを使用して挿入できます。レポート Operator は、指定されたベアラートークンを使用して Prometheus で認証するように設定することもできます。

手順

- 設定マップを使用して Prometheus インスタンスが使用する CA を挿入します。以下は例になります。

```
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-----
```

または、一般に有効な証明書のシステム認証局を使用するには、 **useServiceAccountCA** および **configMap.enabled** の両方を **false** に設定します。

- Prometheus で認証するベアラートークンを指定します。以下は例になります。

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

4.5.2. レポート API の公開

OpenShift Container Platform では、デフォルトのメータリングインストールはルートを自動的に公開し、レポート API を利用可能にします。これにより、以下の機能が提供されます。

- 自動 DNS
- クラスター CA に基づく自動 TLS

また、デフォルトのインストールでは、OpenShift サービスを使用して証明書を提供し、レポート API を TLS で保護することができます。OpenShift OAuth プロキシーはレポート Operator のサイドカーコンテナーとしてデプロイされ、レポート API を認証で保護します。

4.5.2.1. OpenShift 認証の使用

デフォルトで、レポート API のセキュリティーは TLS および認証で保護されます。これは、レポート Operator をレポート Operator のコンテナーおよび OpenShift 認証プロキシーを実行するサイドカーコンテナーの両方を含む Pod をデプロイするように設定して実行されます。

レポート API にアクセスするために、メータリング Operator はルートを公開します。ルートがインストールされたら、以下のコマンドを実行してルートのホスト名を取得できます。

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

次に、サービスアカウントトークンまたはユーザー名およびパスワードによる基本認証のいずれかを使用して認証を設定します。

4.5.2.1.1. サービスアカウントトークンを使用した認証

この方法では、以下のコマンドを使用してトークンをレポート Operator のサービスアカウントで使用し、そのベアラートークンを Authorization ヘッダーに渡します。

```
$ 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]"
```

上記の URL の **name=[Report Name]** および **format=[Format]** パラメーターを置き換えます。**format** パラメーターは、json、csv、または tabular にすることができます。

4.5.2.1.2. ユーザー名とパスワードを使用した認証

メータリングは、htpasswd ファイルの内容に指定されるユーザー名とパスワードの組み合わせを使用する基本認証の設定をサポートします。デフォルトで、空の htpasswd データを含むシークレットが作成されます。ただし、**reporting-operator.spec.authProxy.htpasswd.data** および **reporting-operator.spec.authProxy.htpasswd.createSecret** キーを、この方法を使用するように設定できます。

MeteringConfig で上記を指定した後に、以下のコマンドを実行できます。

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

testuser:password123 を有効なユーザー名とパスワードの組み合わせに置き換えます。

4.5.2.2. 認証の手動設定

レポート Operator で OAuth を手動で設定するか、または無効にするには、**MeteringConfig** リソースで **spec.tls.enabled: false** を設定する必要があります。

警告

これは、レポート Operator、Presto、および Hive 間のすべての TLS および認証も無効にします。これらのリソースは手動で設定する必要があります。

認証を有効にするには、以下のオプションを設定します。認証を有効にすると、レポート Operator Pod が OpenShift 認証プロキシーを Pod のサイドカーコンテナーとして実行するように設定されます。これによりポートが調整され、レポート API が直接公開されず、代わりに認証プロキシーサイドカーコンテナーでプロキシーされます。

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

reporting-operator.spec.authProxy.enabled および **reporting-operator.spec.authProxy.cookie.createSecret** を **true** に設定し、**reporting-operator.spec.authProxy.cookie.seed** を 32 文字のランダムな文字列に設定する必要があります。

以下のコマンドを使用して、32 文字のランダムな文字列を生成できます。

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

4.5.2.2.1. トークン認証

以下のオプションが **true** に設定されている場合、ベアラートークンを使用する認証がレポート REST API に対して有効になります。ベアラートークンはサービスアカウントまたはユーザーから送られる場合があります。

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

認証が有効にされると、ユーザーまたはサービスアカウントのレポート API をクエリーするために使用されるベアラートークンに、以下のロールのいずれかを使用するアクセスが付与される必要があります。

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

メータリング Operator は、**spec.permissions** セクションにサブジェクトの一覧を指定して、ロールバインディングを作成し、これらのパーミッションを付与できます。たとえば、以下の **advanced-auth.yaml** の設定例を参照してください。

```
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
```

または、**get** パーミッションを **reports/export** に付与するルールを持つすべてのロールを使用できます。これは、レポート Operator の namespace の **Report** リソースの **export** サブリソースに対する **get** アクセスです。例: **admin** および **cluster-admin**

デフォルトで、レポート Operator およびメータリング Operator サービスアカウントにはどちらにもこれらのパーミッションがあり、それらのトークンを認証に使用することができます。

4.5.2.2.2. ユーザー名とパスワードを使用した基本認証

基本認証では、**reporting-operator.spec.authproxy.htpasswd.data** フィールドにユーザー名とパスワードを指定することができます。ユーザー名とパスワードは htpasswd ファイルにあるものと同じ形式である必要があります。設定されている場合、**htpasswdData** のコンテンツに対応するエントリーのあるユーザー名とパスワードを指定するために HTTP 基本認証を使用できます。

4.6. AWS 請求情報の関連付けの設定

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

メータリングは、クラスターの使用状況に関する情報を、 AWS の詳細の請求情報 に関連付け、金額 (ドル) をリソースの使用量に割り当てます。EC2 で実行しているクラスターの場合、以下の **aws-billing.yaml** ファイルのサンプルを変更してこれを有効にできます。

```
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
```

AWS 請求情報の関連付けを有効にするには、まず AWS コストと使用状況のレポートを有効にします。詳細は、AWS ドキュメントの Creating Cost and Usage Reports を参照してください。

1 バケット、接頭辞、およびリージョンを AWS の詳細請求レポートの場所で更新します。

2 3 4 すべての **secretName** フィールドは、**data.aws-access-key-id** および **data.aws-secret-access-key** フィールドの AWS 認証情報が含まれるメータリング namespace のシークレットの名前に設定される必要があります。詳細は、以下のシークレットファイルのサンプルを参照してください。

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

S3 にデータを保存するには、**aws-access-key-id** および **aws-secret-access-key** の認証情報にバケットへの読み書きアクセスが必要になります。IAM ポリシーが必要なパーミッションを付与する例については、以下の **aws/read-write.json** ファイルを参照してください。

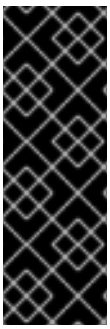
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "1",
         "Effect": "Allow",
         "Action": [
            "s3:AbortMultipartUpload",
            "s3:DeleteObject",
            "s3:GetObject",
            "s3:HeadBucket",
            "s3:ListBucket",
            "s3:ListMultipartUploadParts",
            "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 **operator-metering-data** をバケットの名前に置き換えます。

これは、インストール前またはインストール後のいずれかに実行できます。インストール後にこれを無効にすると、レポート Operator でエラーが発生する場合があります。

第5章 REPORT

5.1. REPORT について



重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

Report カスタムリソースは、SQL クエリーを使用して定期的な ETL (Extract Transform および Load) ジョブを管理する方法を提供します。レポートは、実行する実際の SQL クエリーを提供する **ReportQuery** リソースや、**ReportQuery** および **Report** リソースで利用できるデータを定義する **ReportDataSource** リソースなどの他のメータリングリソースで設定されます。

多くのユースケースは、メータリングと共にインストールされる事前に定義された **ReportQuery** および **ReportDataSource** リソースで対応されます。したがって、これらの事前定義済みのリソースで対応されないユースケースがない場合、独自の定義は必要ありません。

5.1.1. Report

Report カスタムリソースは、レポートの実行およびステータスを管理するために使用されます。メータリングは、使用状況のデータソースから派生するレポートを生成します。これは、詳細な分析およびフィルターで使用できます。単一の **Report** リソースは、データベーステーブルを管理するジョブを示し、これをスケジュールに応じて新しい情報で更新します。レポートは、テーブルのデータをレポート Operator HTTP API 経由で公開します。

spec.schedule フィールドが設定された Report は常に実行された状態となり、データの収集期間を追跡します。メータリングが長期間シャットダウンするか、または使用できない状態になる場合、データの停止時点からデータをバックフィルします。スケジュールが設定されていない場合、レポートは **reportingStart** および **reportingEnd** で指定された期間に 1 回実行されます。デフォルトで、レポートは **ReportDataSource** リソースがレポート期間内のデータを完全にインポートするのを待機します。レポートにスケジュールがある場合、現在処理されている期間内のデータのインポートがすべて完了するまで待機します。

5.1.1.1. スケジュールが設定されたレポートの例

以下のサンプル **Report** にはすべての Pod の CPU 要求についての情報が含まれ、1 時間に 1 回実行され、レポートが実行されるごとにその 1 時間前からの関連データが追加されます。

```
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. スケジュールなしのサンプルレポート (1回のみ実行)

以下のサンプル **Report** オブジェクトには、7 月中のすべての Pod の CPU 要求についての情報が含まれます。完了後に再度実行されることはありません。

```
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

query フィールドは、レポートを生成するために使用される **ReportQuery** リソースに名前を指定します。レポートクエリーは、結果の処理方法と共にレポートのスキーマを制御します。

query は必須フィールドです。

利用可能な **ReportQuery** リソースを一覧表示するには、以下のコマンドを使用します。

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

出力例

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

```
node-cpu-capacity-raw                       23m
node-cpu-utilization                        23m
node-memory-allocatable                     23m
node-memory-allocatable-raw                 23m
node-memory-capacity                        23m
node-memory-capacity-raw                    23m
node-memory-utilization                     23m
persistentvolumeclaim-capacity              23m
persistentvolumeclaim-capacity-raw          23m
persistentvolumeclaim-phase-raw             23m
persistentvolumeclaim-request               23m
persistentvolumeclaim-request-raw           23m
persistentvolumeclaim-usage                 23m
persistentvolumeclaim-usage-raw             23m
persistentvolumeclaim-usage-with-phase-raw  23m
pod-cpu-request                             23m
pod-cpu-request-raw                         23m
pod-cpu-usage                               23m
pod-cpu-usage-raw                           23m
pod-memory-request                          23m
pod-memory-request-raw                      23m
pod-memory-usage                            23m
pod-memory-usage-raw                        23m
```

-raw 接尾辞のあるレポートクエリーは、より複雑なクエリーを作成するために他の **ReportQuery** によって使用されます。これらはレポートに直接使用できません。

namespace- の接頭辞が付けられたクエリーは namespace 別に Pod CPU およびメモリー要求を集計し、リソース要求に基づいて namespace およびそれらの全体の使用状況の一覧を提供します。

pod- の接頭辞が付けられたクエリーは **namespace-** の接頭辞が付けられたクエリーと同様ですが、情報を namespace 別ではなく Pod 別に集計します。これらのクエリーには、Pod の namespace およびノードが含まれます。

node- の接頭辞が付けられたクエリーは各ノードの利用可能な合計リソースについての情報を返します。

aws- の接頭辞が付けられたクエリーは AWS に固有のものです。**aws** の接尾辞が付けられたクエリーは、接尾辞なしの同じ名前のクエリーと同じデータを返し、使用状況を EC2 請求データに関連付けます。

aws-ec2-billing-data レポートは他のクエリーによって使用され、スタンドアロンのレポートとしては使用できません。**aws-ec2-cluster-cost** レポートは、クラスターに含まれるノードに基づく総コストと、レポート期間のコストの合計を提供します。

以下のコマンドを使用して **ReportQuery** リソースを YAML として取得し、**spec.columns** フィールドを確認します。たとえば、以下を実行します。

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

出力例

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

```
  name: namespace-memory-request
  labels:
    operator-metering: "true"
spec:
  columns:
  - name: period_start
    type: timestamp
    unit: date
  - name: period_end
    type: timestamp
    unit: date
  - name: namespace
    type: varchar
    unit: kubernetes_namespace
  - name: pod_request_memory_byte_seconds
    type: double
    unit: byte_seconds
```

5.1.1.4. schedule

spec.schedule 設定ブロックは、レポートが実行される時を定義します。**schedule** セクションの主なフィールドは **period** であり、**period** の値によって、**hourly**、**daily**、**weekly**、および **monthly** フィールドでレポートが実行されるタイミングをさらに調整できます。

たとえば、**period** が **weekly** に設定されている場合、**weekly** フィールドを **spec.schedule** ブロックに追加できます。以下の例は、週ごとに毎週水曜日の 1 pm (hour 13) に実行されます。

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

5.1.1.4.1. period

schedule.period の有効な値が以下に一覧表示されており、特定の期間に設定できる選択可能なオプションも一覧表示されています。

- **hourly**
  - **minute**
  - **second**
- **daily**
  - **hour**
  - **minute**
  - **second**
- **weekly**

    - **dayOfWeek**
    - **hour**
    - **minute**
    - **second**
- **monthly**
    - **dayOfMonth**
    - **hour**
    - **minute**
    - **second**
- **cron**
    - **expression**

通常、**hour**、**minute**、**second** フィールドは 1 日のどの時間にレポートが実行されるかを制御し、**dayOfWeek**/**dayOfMonth** は、レポートの期間が週または月ごとに区切られている場合にレポートが実行される曜日または日を制御します。

上記の各フィールドには、有効な値の範囲があります。

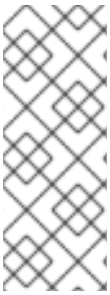
- **hour** は 0-23 の整数値です。
- **minute** は 0-59 の整数値です。
- **second** は 0-59 の整数値です。
- **dayOfWeek** は曜日が入ることが予想される文字列の値です (略さずに入力します)。
- **dayOfMonth** は 1-31 の整数値です。

cron 期間については、通常の cron 式は有効です。

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

5.1.1.5. reportingStart

既存データに対するレポートの実行をサポートするには、**spec.reportingStart** フィールドを RFC3339 タイムスタンプ に設定し、レポートが現在の時間ではなく、**reportingStart** から始まる **schedule** に基づいて実行するように指示します。



注記

spec.reportingStart フィールドを特定の時間に設定すると、レポート Operator が **reportingStart** の時間から現在の時間までの間のスケジュール期間に連続して多数のクエリーを実行する可能性があります。レポート期間が日次よりも短く区切られ、**reportingStart** が数ヶ月前に遡る場合、クエリーの数は数千に上る可能性があります。**reportingStart** が未設定のままの場合、レポートはレポート作成後の次の **reportingPeriod** 全体で実行されます。

このフィールドの使い方を示す一例として、**Report** オブジェクトに組み込む必要のある 2019 年月 1 日まで遡ったデータをすでに収集している場合、以下の値を使用してレポートを作成できます。

```
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

指定された時点までのみ実行されるようにレポートを設定するには、**spec.reportingEnd** フィールドを RFC3339 タイムスタンプ に設定できます。このフィールドの値により、レポートは開始時点から **reportingEnd** までの期間のレポートデータの生成の終了後にスケジュールに基づいて実行を停止します。

スケジュールと **reportingEnd** は連動しない場合が多いため、スケジュールの最終期間は指定の **reportingEnd** 時間に終了するように短縮されます。これが未設定のままの場合、レポートは永久に実行されるか、または **reportingEnd** がレポートに設定されるまで実行されます。

たとえば、7 月に週 1 回実行されるレポートを作成する場合は、以下のようになります。

```
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. expiration

expiration フィールドを追加して、スケジュールされるメータリングレポートに保持期間を設定します。**expiration** 期間の値を設定すると、レポートを手動で削除することを避けることができます。保持期間はレポートの **Report** オブジェクトの **creationDate** に **expiration** を加えた期間と等しくなります。レポートまたはレポートクエリーが期限切れのレポートに依存しない場合、レポートが保持期間の終了時にクラスターから削除されます。レポートをクラスターから削除するには数分の時間がかかる場合があります。

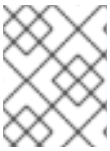
注記

ロールアップまたは集計レポートに **expiration** フィールドを設定することは推奨されません。他のレポートまたはレポートクエリーがレポートに依存する場合、そのレポートは保持期間の終了時に削除されません。レポート保持の決定に関連したタイミングの出力についてデバッグレベルで **report-operator** ログを表示できます。

たとえば、以下のスケジュールされたレポートは、レポートの **creationDate** の 30 秒後に削除されます。

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

1 **expiration** 期間の有効な時間単位は、**ns**、**us** (または **µs**)、**ms**、**s**、**m**、および **h** です。



注記

Report オブジェクトの **expiration** 保持期間は厳密ではなく、(ナノ秒ではなく) 数分間隔の順序で機能します。

5.1.1.8. runImmediately

runImmediately を **true** に設定すると、レポートは即座に実行されます。この動作により、追加のスケジューリングパラメーターなしにレポートが即座に処理され、キューに入れられます。

注記

runImmediately が **true** に設定されている場合、**reportingEnd** および **reportingStart** の値を設定する必要があります。

5.1.1.9. inputs

Report オブジェクトの **spec.inputs** フィールドは、 **ReportQuery** リソースの **spec.inputs** フィールドで定義された値を上書きまたは設定するために使用できます。

spec.inputs は名前と値のペアの一覧です。

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

1 inputs の **name** は ReportQuery の **inputs** 一覧に存在している必要があります。

2 inputs の **value** は inputs の **type** に適切なタイプである必要があります。

5.1.1.10. ロールアップレポート

レポートデータはメトリクス自体と同様にデータベースに保存されるため、集計またはロールアップレポートで使用できます。ロールアップレポートの単純なユースケースとして、レポートの作成に必要な時間をより長い期間にわたって分散します。これにより、クエリーし、1 カ月全体でのすべてのデータ

を追加する月次レポートは不要になります。たとえば、タスクは、それぞれがデータの 1/30 に対して実行される日次レポートに分割できます。

カスタムのロールアップレポートには、カスタムレポートクエリーが必要です。**ReportQuery** リソーステンプレートプロセッサーは、**Report** オブジェクトの **metadata.name** から必要なテーブル名を取得できる **reportTableName** 機能を提供します。

以下は、組み込みクエリーのスニペットです。

pod-cpu.yaml

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

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

aggregated-report.yaml ロールアップレポートの例

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

5.1.1.10.1. レポートのステータス

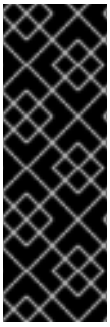
スケジュールされたレポートの実行は、status フィールドを使用して追跡できます。レポートの作成中に発生したエラーはここに記録されます。

現時点で **Report** オブジェクトの **status** フィールドには 2 つのフィールドがあります。

- **conditions**: これは、それぞれに **type**、**status**、**reason**、および **message** フィールドのある状態についての一覧です。状態の **type** フィールドに使用できる値は **Running** および **Failure** であり、スケジュールされたレポートの現在の状態を示します。**reason** は、**condition** が **true**、**false** または、**unknown** のいずれかの **status** で示される現在の状態にある理由を示します。**message** は、condition が現在の状態にある理由についての人が判別できる情報を提供します。**reason** の値の詳細情報については、**pkg/apis/metering/v1/util/report_util.go** を参照してください。

- **lastReportTime**: メータリングが最後にデータを収集した時を示します。

5.2. ストレージの場所



重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

StorageLocation カスタムリソースは、データがレポート Operator によって保存される場所を設定します。これには、Prometheus から収集されるデータと **Report** カスタムリソースを生成して生成される結果が含まれます。

複数の S3 バケットや S3 と HDFS の両方などの複数の場所にデータを保存する必要がある場合や、メータリングによって作成されていない Hive/Presto のデータベースにアクセスする必要がある場合は、**StorageLocation** カスタムリソースのみを設定する必要があります。ほとんどのユーザーの場合、この設定は不要であり、必要なすべてのストレージコンポーネントを設定するには、メータリングの設定についてのドキュメント を参照するだけで十分です。

5.2.1. ストレージの場所の例

以下の例は、ビルトインローカルストレージオプションを示しています。これは、Hive を使用するように設定されています。デフォルトで、データは Hive がストレージ (HDFS、S3、または **ReadWriteMany** 永続ボリューム要求 (PVC)) を使用するように設定される場合には常に保存されます。

ローカルストレージの例

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

(1) **hive** セクションが存在する場合、Hive サーバーを使用してテーブルを作成し、**StorageLocation** をデータを Presto に保管するように設定します。**databaseName** および **unmanagedDatabase** のみが必須フィールドです。

(2) Hive 内のデータベースの名前。

(3) **true** の場合、**StorageLocation** リソースは能動的に管理されず、**databaseName** が Hive に常に存在することが予想されます。**false** の場合、レポート Operator はデータベースを Hive に作成します。

以下の例では、ストレージに AWS S3 バケットを使用します。使用するパスを作成する際に、接頭辞がバケット名に追加されます。

リモートストレージの例

```
apiVersion: metering.openshift.io/v1
kind: StorageLocation
metadata:
  name: example-s3-storage
  labels:
    operator-metering: "true"
spec:
  hive:
    databaseName: example_s3_storage
    unmanagedDatabase: false
    location: "s3a://bucket-name/path/within/bucket" 1
```

1 オプション: データベースに使用する Presto および Hive のファイルシステムの URL。これには、**hdfs://** または **s3a://** ファイルシステム URL を使用できます。

hive セクションに指定できる追加のオプションフィールドがあります。

- **defaultTableProperties**: Hive を使用してテーブルを作成する設定オプションが含まれます。
- **fileFormat**: ファイルシステムにファイルを保存するために使用するファイル形式です。オプションの一覧や詳細については、File Storage Format の Hive ドキュメント を参照してください。
- **rowFormat**: Hive row フォーマット を制御します。これは、Hive が行をシリアライズ/デシリアライズする方法を制御します。詳細は、Hive Documentation on Row Formats and SerDe を参照してください。

5.2.2. デフォルトのストレージの場所

アノテーションの **storagelocation.metering.openshift.io/is-default** が存在し、**StorageLocation** リソースで **true** に設定されている場合、そのリソースはデフォルトのストレージリソースになります。ストレージの場所が指定されていないストレージ設定オプションを持つすべてのコンポーネントはデフォルトのストレージリソースを使用します。デフォルトのストレージリソースは 1 つのみです。アノテーションを持つ複数のリソースが存在する場合、レポート Operator がデフォルトを判別できないためエラーがログに記録されます。

デフォルトのストレージの例

```
apiVersion: metering.openshift.io/v1
kind: StorageLocation
metadata:
  name: example-s3-storage
  labels:
    operator-metering: "true"
  annotations:
    storagelocation.metering.openshift.io/is-default: "true"
spec:
  hive:
```

```
databaseName: example_s3_storage
unmanagedDatabase: false
location: "s3a://bucket-name/path/within/bucket"
```

第6章 メータリングの使用

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

6.1. 前提条件

- メータリングをインストールします。
- レポート に設定できる利用可能なオプションと、その機能の詳細を確認してください。

6.2. レポートの作成

レポートの作成は、メータリングを使用してデータを処理し、分析する手段です。

レポートを作成するには、YAML ファイルで **Report** リソースを定義し、必要なパラメーターを指定し、これを **openshift-metering** namespace に作成する必要があります。

前提条件

- メータリングがインストール済みです。

手順

1. **openshift-metering** プロジェクトに切り替えます。

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

2. **Report** リソースを YAML ファイルとして作成します。
   a. 以下の内容を含む YAML ファイルを作成します。

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

(2) **query** は、レポートの生成に使用する **ReportQuery** を指定します。レポートする内容に応じて、この値を変更します。オプションの一覧については、**oc get reportqueries | grep -v raw** を実行します。

1 レポートが **metadata.name** について実行する内容を説明する名前を使用します。使用したクエリー、スケジュールまたは期間を説明する適切な名前を使用してくださ

3 利用可能なデータを使用して実行できるようにするには、**runImmediately** を **true** に設定するか、または **reportingEnd** が経過するのを待機するようにするには **false** に設定します。

b. 以下のコマンドを実行して **Report** リソースを作成します。

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

出力例

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

3. 以下のコマンドで、レポートおよびそれらの **Running** ステータスを一覧表示できます。

```
$ oc get reports
```

出力例

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

6.3. レポート結果の表示

レポートの結果を表示するには、レポート API ルートを使用し、OpenShift Container Platform 認証情報を使用して API に対して認証する必要があります。レポートは、**JSON**、**CSV**、または **Tabular** 形式で取得できます。

前提条件

- メータリングがインストールされている。
- レポートの結果にアクセスするには、クラスター管理者であるか、または **openshift-metering** namespace で **report-exporter** ロールを使用するアクセスが付与される必要があります。

手順

1. **openshift-metering** プロジェクトに切り替えます。

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

2. レポート API で結果についてクエリーします。

a. メータリング **reporting-api** ルートの変数を作成し、ルートを取得します。

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

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

b. 要求で使用する現行ユーザーのトークンを取得します。

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

c. **reportName** を作成したレポートの名前に設定します。

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

d. **reportFormat** を **csv**、**json**、または **tabular** のいずれかに設定し、API 応答の出力形式を指定します。

```
$ reportFormat=csv
```

e. 結果を取得するには、**curl** を使用してレポートについてのレポート API に対する要求を実行します。

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

reportName=namespace-cpu-request-2019 および reportFormat=csv の出力例

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

```
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-dns-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-etcd,23490.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-image-registry,5993.400000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-ingress,5220.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-ingress-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-apiserver,12528.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-apiserver-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-controller-manager,8613.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-kube-controller-manager-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-machine-api,1305.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-machine-config-operator,9637.800000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-metering,19575.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-monitoring,6256.800000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-network-operator,261.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-sdn,94503.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-service-ca,783.000000
2019-01-01 00:00:00 +0000 UTC,2019-12-30 23:59:59 +0000 UTC,openshift-service-ca-operator,261.000000
```

第7章 メータリングの使用例

重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

以下のサンプルレポートを使用して、クラスター内の容量、使用および使用状況の測定を開始します。これらのサンプルでは、メータリングが提供するさまざまなタイプのレポートが事前に定義されたクエリーの選択と共に表示されるケースを示しています。

7.1. 前提条件

- メータリングをインストールします。
- レポートの作成および表示 についての詳細を確認します。

7.2. クラスター容量の毎時および日次の測定

以下のレポートは、クラスター容量を毎時および日次に測定する方法を示しています。日次レポートは毎時レポートの結果を集計して生成されます。

以下は、クラスターの CPU 容量を毎時に測定するレポートです。

クラスターの毎時の CPU 容量の例

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

(1) この期間は **daily** に変更して日次レポートを取得することができますが、大規模なデータセットの場合、毎時レポートを使用してから、毎時データを日次レポートに集計する方がはるかに効率的です。

以下のレポートは、毎時データを日次レポートに集計します。

クラスターの日次の CPU 容量の例

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: cluster-cpu-capacity-daily (1)
```

```
spec:
  query: "cluster-cpu-capacity" 2
  inputs: 3
  - name: ClusterCpuCapacityReportName
    value: cluster-cpu-capacity-hourly
  schedule:
    period: "daily"
```

1 レポートの編成を維持するには、他の値のいずれかを変更する場合にレポートの **name** を変更するようにしてください。

2 **cluster-memory-capacity** を測定することもできます。関連付けられた毎時レポートでクエリーも更新するようにしてください。

3 **inputs** セクションでは、このレポートを毎次レポートを集計するように設定します。具体的には、**value: cluster-cpu-capacity-hourly** は集計される毎時レポートの名前になります。

7.3.1回のみ実行されるレポートを使用したクラスター使用状況の測定

以下のレポートは、クラスターの使用状況を特定の開始日以降から測定します。レポートは一度だけ実行され、その後は保存して適用します。

クラスターの CPU 使用状況の例

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

1 レポートの編成を維持するには、他の値のいずれかを変更する場合にレポートの **name** を変更するようにしてください。

2 レポートを、**reportingStart** タイムスタンプから **reportingEnd** タイムスタンプまでのデータの使用を開始するように設定します。

3 ここでクエリーを調整します。**cluster-memory-usage** クエリーでクラスターの使用状況を測定することもできます。

4 レポートを、保存および適用後すぐに実行するように設定します。

7.4. CRON 式を使用したクラスター使用状況の測定

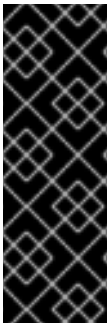
レポートの期間を設定する際に cron 式を使用することもできます。以下のレポートは、平日の 9am-5pm の間にクラスターの使用状況を観察して CPU の使用状況を測定します。

クラスターの平日の CPU 使用状況の例

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

1. レポートの編成を維持するには、他の値のいずれかを変更する場合にレポートの **name** を変更するようにしてください。
2. ここでクエリーを調整します。**cluster-memory-utilization** クエリーでクラスターの使用状況を測定することもできます。
3. cron の期間については、通常の cron 式が有効です。

第8章 メータリングのトラブルシューティングおよびデバッグ



重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

以下のセクションを参照して、メータリングに関連する特定の問題のトラブルシューティングとデバッグを行ってください。

このセクションの情報に加えて、次のトピックを確認してください。

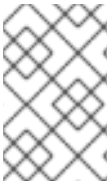
- メータリングのインストールの前提条件
- メータリングの設定について

8.1. メータリングのトラブルシューティング

メータリングに関連する一般的な問題として、Pod が起動に失敗する問題があります。Pod はリソースがないか、または **StorageClass** または **Secret** リソースなど、存在しないリソースへの依存関係がある場合に起動に失敗する可能性があります。

8.1.1. 十分なコンピュートリソースがない

メータリングのインストールまたは実行時に、コンピュートリソースがないという問題がよく生じます。クラスターが拡大し、より多くのレポートが作成されると、レポート Operator Pod にはより多くのメモリーが必要になります。メモリー使用量が Pod の制限に達すると、クラスターは Pod のメモリー不足 (OOM) を考慮し、これを **OOMKilled** ステータスで終了します。メータリングにインストールの前提条件で説明されている最小限のリソース要件が適用されていることを確認します。



注記

メータリング Operator は、クラスターの負荷に基づいてレポート Operator を自動スケーリングしません。そのため、クラスターが大きくなると、レポート Operator Pod の CPU 使用率は増加しません。

問題がリソースまたはスケジュールに関連するかどうかを判別するには、Kubernetes ドキュメントの Managing Resources for Containers にあるトラブルシューティングの指示に従ってください。

コンピュートリソースがないために問題のトラブルシューティングを行うには、**openshift-metering** namespace 内で以下を確認します。

前提条件

- 現在の位置は **openshift-metering** namespace である。以下を実行して **openshift-metering** namespace に切り替えます。

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

手順

1. 完了しなかったメータリング **Report** リソースの有無を確認し、**ReportingPeriodUnmetDependencies** のステータスを表示します。

```
$ oc get reports
```

出力例

```
NAME                                QUERY                       SCHEDULE    RUNNING
FAILED   LAST REPORT TIME       AGE
namespace-cpu-utilization-adhoc-10    namespace-cpu-utilization                Finished
2020-10-31T00:00:00Z   2m38s
namespace-cpu-utilization-adhoc-11    namespace-cpu-utilization
ReportingPeriodUnmetDependencies                                 2m23s
namespace-memory-utilization-202010   namespace-memory-utilization
ReportingPeriodUnmetDependencies                                 26s
namespace-memory-utilization-202011   namespace-memory-utilization
ReportingPeriodUnmetDependencies                                 14s
```

2. **NEWEST METRIC** の値がレポートの終了日よりも小さい **ReportDataSource** リソースを確認します。

```
$ oc get reportdatasource
```

出力例

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

3. 多数の Pod の再起動のについて、**reporting-operator Pod** リソースの正常性を確認します。

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

出力例

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

(1) レポート Operator Pod は高い率で再起動します。

4. **reporting-operator Pod** リソースで **OOMKilled** の終了について確認します。

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

出力例

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

1. レポート Operator Pod は OOM による強制終了により終了しています。

reporting-operator Pod メモリー制限の引き上げ

Pod の再起動や OOM による強制終了イベントが増加している場合、レポート Operator Pod に設定された現在のメモリー制限を確認できます。メモリー制限を増やすと、レポート Operator Pod はレポートデータソースを更新できます。必要な場合は、**MeteringConfig** リソースのメモリー制限を 25% - 50% 引き上げます。

手順

1. **reporting-operator Pod** リソースの現在のメモリー制限を確認します。

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

出力例

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

1. レポート Operator Pod の現在のメモリー制限。

2. **MeteringConfig** リソースを編集してメモリー制限を更新します。

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

MeteringConfig リソース例

```
kind: MeteringConfig
metadata:
  name: operator-metering
  namespace: openshift-metering
spec:
  reporting-operator:
  spec:
    resources: 1
      limits:
        cpu: 1
        memory: 750Mi
      requests:
        cpu: 500m
        memory: 500Mi
...
```

1 **MeteringConfig** リソースの **resources** フィールド内でメモリー制限を追加または引き上げます。

注記

メモリー制限が引き上げられた後でも引き続き OOM で強制終了された多数のイベントがある場合は、別の問題がレポートを保留状態にしていることを示唆している可能性があります。

8.1.2. StorageClass リソースが設定されない

メータリングでは、デフォルトの **StorageClass** リソースが動的プロビジョニングに設定されている必要があります。

クラスターに設定された **StorageClass** があるかどうかをチェックする方法、デフォルトの設定方法、およびメータリングをデフォルト以外のストレージクラス を使用するように設定する方法についての詳細は、メータリングの設定方法についてのドキュメントを参照してください。

8.1.3. シークレットが正しく設定されていない

メータリングに関連する一般的な問題として、永続ストレージの設定時に誤ったシークレットが指定されることがあります。設定ファイルのサンプルを確認し、ストレージプロバイダーのガイドラインに従ってシークレットを作成することを確認してください。

8.2. メータリングのデバッグ

メータリングのデバッグは、各種のコンポーネントと直接対話する場合に大幅に容易になります。以下のセクションでは、Presto および Hive への接続とクエリー方法、および Presto および HDFS コンポーネントのダッシュボードの表示方法について詳しく説明します。

注記

このセクションのコマンドではすべて、メータリングを **openshift-metering** namespace の OperatorHub 経由でインストールしていることを前提とします。

8.2.1. レポート Operator ログの取得

以下のコマンドを使用して **reporting-operator** のログに従います。

```
$ 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. presto-cli を使用した Presto のクエリー

以下のコマンドは、Presto をクエリーできる対話型の presto-cli セッションを開きます。このセッションは Presto と同じコンテナー内で実行され、Pod のメモリー制限を作成できる追加の Java インスタンスを起動します。これが実行される場合は、Presto Pod のメモリー要求および制限を引き上げる必要があります。

デフォルトでは、Presto は TLS を使用して通信するように設定されます。Presto クエリーを実行するには、以下のコマンドを使用する必要があります。

```
$ 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
```

このコマンドを実行すると、クエリーを実行できるようにプロンプトが表示されます。**show tables from metering;** クエリーを使用してテーブルの一覧を表示します。

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

出力例

```
                        Table

datasource_your_namespace_cluster_cpu_capacity_raw
datasource_your_namespace_cluster_cpu_usage_raw
datasource_your_namespace_cluster_memory_capacity_raw
datasource_your_namespace_cluster_memory_usage_raw
datasource_your_namespace_node_allocatable_cpu_cores
datasource_your_namespace_node_allocatable_memory_bytes
datasource_your_namespace_node_capacity_cpu_cores
datasource_your_namespace_node_capacity_memory_bytes
datasource_your_namespace_node_cpu_allocatable_raw
datasource_your_namespace_node_cpu_capacity_raw
datasource_your_namespace_node_memory_allocatable_raw
datasource_your_namespace_node_memory_capacity_raw
datasource_your_namespace_persistentvolumeclaim_capacity_bytes
datasource_your_namespace_persistentvolumeclaim_capacity_raw
datasource_your_namespace_persistentvolumeclaim_phase
datasource_your_namespace_persistentvolumeclaim_phase_raw
datasource_your_namespace_persistentvolumeclaim_request_bytes
datasource_your_namespace_persistentvolumeclaim_request_raw
```

```
 datasource_your_namespace_persistentvolumeclaim_usage_bytes
 datasource_your_namespace_persistentvolumeclaim_usage_raw
 datasource_your_namespace_persistentvolumeclaim_usage_with_phase_raw
 datasource_your_namespace_pod_cpu_request_raw
 datasource_your_namespace_pod_cpu_usage_raw
 datasource_your_namespace_pod_limit_cpu_cores
 datasource_your_namespace_pod_limit_memory_bytes
 datasource_your_namespace_pod_memory_request_raw
 datasource_your_namespace_pod_memory_usage_raw
 datasource_your_namespace_pod_persistentvolumeclaim_request_info
 datasource_your_namespace_pod_request_cpu_cores
 datasource_your_namespace_pod_request_memory_bytes
 datasource_your_namespace_pod_usage_cpu_cores
 datasource_your_namespace_pod_usage_memory_bytes
(32 rows)

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

presto:default>
```

8.2.3. beeline を使用した Hive のクエリー

以下のコマンドでは、Hive をクエリーできる対話型の beeline セッションを開きます。このセッションは Hive と同じコンテナー内で実行され、Pod のメモリー制限を作成できる追加の Java インスタンスを起動します。これが実行される場合は、Hive Pod のメモリー要求および制限を引き上げる必要があります。

```
$ 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'
```

このコマンドを実行すると、クエリーを実行できるようにプロンプトが表示されます。**show tables;** クエリーを使用してテーブルの一覧を表示します。

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

出力例

```
+----------------------------------------------------+
|                      tab_name                      |
+----------------------------------------------------+
| datasource_your_namespace_cluster_cpu_capacity_raw |
| datasource_your_namespace_cluster_cpu_usage_raw  |
| datasource_your_namespace_cluster_memory_capacity_raw |
| datasource_your_namespace_cluster_memory_usage_raw |
| datasource_your_namespace_node_allocatable_cpu_cores |
| datasource_your_namespace_node_allocatable_memory_bytes |
| datasource_your_namespace_node_capacity_cpu_cores |
| datasource_your_namespace_node_capacity_memory_bytes |
| datasource_your_namespace_node_cpu_allocatable_raw |
| datasource_your_namespace_node_cpu_capacity_raw  |
| datasource_your_namespace_node_memory_allocatable_raw |
```

```
| datasource_your_namespace_node_memory_capacity_raw |
| datasource_your_namespace_persistentvolumeclaim_capacity_bytes |
| datasource_your_namespace_persistentvolumeclaim_capacity_raw |
| datasource_your_namespace_persistentvolumeclaim_phase |
| datasource_your_namespace_persistentvolumeclaim_phase_raw |
| datasource_your_namespace_persistentvolumeclaim_request_bytes |
| datasource_your_namespace_persistentvolumeclaim_request_raw |
| datasource_your_namespace_persistentvolumeclaim_usage_bytes |
| datasource_your_namespace_persistentvolumeclaim_usage_raw |
| datasource_your_namespace_persistentvolumeclaim_usage_with_phase_raw |
| datasource_your_namespace_pod_cpu_request_raw    |
| datasource_your_namespace_pod_cpu_usage_raw      |
| datasource_your_namespace_pod_limit_cpu_cores    |
| datasource_your_namespace_pod_limit_memory_bytes |
| datasource_your_namespace_pod_memory_request_raw |
| datasource_your_namespace_pod_memory_usage_raw   |
| datasource_your_namespace_pod_persistentvolumeclaim_request_info |
| datasource_your_namespace_pod_request_cpu_cores  |
| datasource_your_namespace_pod_request_memory_bytes |
| datasource_your_namespace_pod_usage_cpu_cores    |
| datasource_your_namespace_pod_usage_memory_bytes |
+--------------------------------------------------+
32 rows selected (13.101 seconds)
0: jdbc:hive2://127.0.0.1:10000/default>
```

8.2.4. Hive Web UI へのポート転送

以下のコマンドを実行して、Hive Web UI へのポート転送を実行します。

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

ブラウザーウィンドウで http://127.0.0.1:10002 を開き、Hive Web インターフェイスを表示します。

8.2.5. HDFS へのポート転送

以下のコマンドを実行して、HDFS namenode へのポート転送を実行します。

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

ブラウザーウィンドウで http://127.0.0.1:9870 を開き、HDFS Web インターフェイスを表示します。

以下のコマンドを実行して、最初の HDFS データノードへのポート転送を実行します。

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

1 他のデータノードをチェックするには、**hdfs-datanode-0** を情報を表示する Pod に置き換えます。

8.2.6. メータリング Ansible Operator

メータリングは Ansible Operator を使用してクラスター環境のリソースを監視し、調整します。メータリングのインストールの失敗をデバッグする場合、Ansible ログや、**MeteringConfig** カスタムリソースのステータスを確認することが役立ちます。

8.2.6.1. Ansible ログへのアクセス

デフォルトのインストールでは、メータリング Operator は Pod としてデプロイされます。この場合、Ansible コンテナーのログを Pod 内で確認できます。

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

または、Operator コンテナーのログで出力の要約を確認できます (**-c ansible** を **-c operator** に置き換えます)。

8.2.6.2. MeteringConfig ステータスの確認

最近の障害についてデバッグするには、**MeteringConfig** カスタムリソースの **.status** フィールドを確認することが役立ちます。以下のコマンドは、**Invalid** タイプのステータスメッセージを表示します。

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

8.2.6.3. MeteringConfig イベントの確認

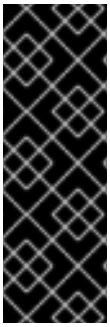
メータリング Operator が生成するイベントを確認します。これは、インストール時またはアップグレード時のリソース障害のデバッグに役立ちます。イベントを最後のタイムスタンプで並べ替えます。

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

出力例には、MeteringConfig リソースの最新の変更が表示されます。

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

第9章 メータリングのアンインストール

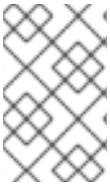


重要

メータリングは非推奨の機能です。非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

OpenShift Container Platform で非推奨となったか、または削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの **非推奨および削除された機能**セクションを参照してください。

メータリングをお使いの OpenShift Container Platform クラスターから削除することができます。

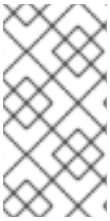


注記

メータリングは Amazon S3 バケットデータを管理したり、削除したりしません。メータリングのアンインストール後に、メータリングデータを保存するために使用される S3 バケットを手動でクリーンアップする必要があります。

9.1. クラスターからのメータリング OPERATOR の削除

Operator のクラスターからの削除 についてのドキュメントを参照して、メータリング Operator を削除します。



注記

クラスターからメータリング Operator を削除しても、そのカスタムリソース定義や管理されるリソースは削除されません。残りのメータリングコンポーネントを削除する方法については、メータリング namespace のアンインストール および メータリングカスタムリソース定義のアンインストール について参照してください。

9.2. メータリング NAMESPACE のアンインストール

MeteringConfig リソースを取り除き、 **openshift-metering** namespace を削除して、メータリング namespace (例: **openshift-metering** namespace) をアンインストールします。

前提条件

- メータリング Operator がクラスターから削除されます。

手順

1. メータリング Operator によって作成されるすべてのリソースを削除します。

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

2. 直前の手順が完了したら、**openshift-metering** namespace のすべての Pod が削除されるか、または終了状態を報告していることを確認します。

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

3. **openshift-metering** namespace を削除します。

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

9.3. メータリングカスタムリソース定義のアンインストール

メータリングのカスタムリソース定義 (CRD) はメータリング Operator のアンインストールおよび **openshift-metering** namespace の削除後もクラスターに残ります。

重要

メータリング CRD を削除すると、クラスターの他の namespace での追加のメータリングインストールが中断されます。次に進む前に、他のメータリングのインストールがないことを確認します。

前提条件

- **openshift-metering** namespace の **MeteringConfig** カスタムリソースが削除されている。
- **openshift-metering** namespace が削除されている。

手順

- 残りのメータリング CRD を削除します。

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