OpenShift Container Platform 4.2

メータリング

OpenShift Container Platform 4.2 でのメータリングの設定および使用

Last Updated: 2020-06-18

OpenShift Container Platform 4.2 メータリング

OpenShift Container Platform 4.2 でのメータリングの設定および使用

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

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 CustomResourceDefinition (CRD) を使用して管理されます。

MeteringConfig	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 スタックを構成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。
Report	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。
ReportQuery	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

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、ここで MeteringConfig という **CustomResource** 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 が作成されます。MeteringConfig 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。

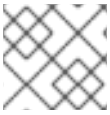
- 永続ボリュームプロビジョニング用の StorageClass。メータリングは、数多くのストレージソリューションをサポートします。
- 4GB メモリー、4 CPU コアが利用できるクラスター容量と、2 CPU コアと 2GB メモリーの容量を持つ 1 つ以上のノード。
- メータリングによってインストールされている最大規模の単一 Pod に必要な最小リソースは 2GB のメモリーと 2 CPU コアです。
    - メモリーおよび CPU の消費量はこれより低くなることがありますが、レポートの実行時や大規模なクラスターのデータの収集時には、消費量は急上昇します。

2.2. メータリング OPERATOR のインストール

メータリング Operator をインストールするには、まず**openshift-metering** namespace を作成し、OperatorHub からメータリング Operator をインストールします。

手順

1. OpenShift Container Platform Web コンソールで **Administration → Namespaces → Create Namespace** をクリックします。
2. 名前を **openshift-metering** に設定します。他の namespace はサポートされません。namespace に **openshift.io/cluster-monitoring=true**のラベルを付け、**Create** をクリックします。
3. 次に、**Operators → OperatorHub** をクリックし、 **metering** についてフィルターし、メータリング Operator を検索します。
4. メータリング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。
5. **Create Operator Subscription** 画面で、上記で作成した **openshift-metering** namespace を選択します。更新チャネルおよび承認ストラテジーを指定してから、 **Subscribe** をクリックしてメータリングをインストールします。
6. **Installed Operators** 画面の **Status** には、メータリングのインストールの終了時に **InstallSucceeded** が表示されます。最初の列の Operator の名前をクリックし、**Operator Details** ページを表示します。



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。

Operator 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。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。

2.3. 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。

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 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 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 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。

2.4. インストールの検証

メータリングのインストールを確認するには、必要な Pod すべてが作成されていること、またレポートデータソースがデータのインポートを開始していることを確認できます。

手順

1. 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。
   oc CLI を使用して同じチェックを実行できます。

```
$ oc -n openshift-metering get pods
NAME                                  READY   STATUS              RESTARTS   AGE
hive-metastore-0                      1/2     Running             0          52s
hive-server-0                         2/3     Running             0          52s
metering-operator-68dd64cfb6-pxh8v    2/2     Running             0          2m49s
presto-coordinator-0                  2/2     Running             0          31s
reporting-operator-56c6c878fb-2zbhp   0/2     ContainerCreating   0          4s
```

2. **Ready**が表示されるまで Pod を継続的にチェックします。これには数分の時間がかかる場合があります。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。一部の Pod には、他の Pod の起動に時間がかかり過ぎる場合に再起動するものがありますが、これが問題になることはなく、インストール時の動作として予想されます。
   oc CLI を使用すると、同じチェックにより以下の様な出力が表示されます。

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

3. 次に、**oc** CLI を使用して ReportDataSource がデータのインポートを開始していることを確認します。これは、 **EARLIEST METRIC** 列の有効なタイムスタンプによって示唆されます (これには数分の時間がかかる場合があります)。データをインポートしない「-raw」ReportDataSource を除外します。

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

2.5. 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 メータリングの設定

3.1. メータリングの設定について

MeteringConfig という **CustomResource** 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。最初にメータリングスタックをインストールすると、デフォルトの **MeteringConfig** 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** で変更し、ファイルを再度適用します。

3.2. 一般的な設定オプション

3.2.1. 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
```

3.2.2. 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する必要がある場合、それぞれのコンポーネントに **nodeSelectors** を設定し、メータリングの各コンポーネントがスケジュールされる場所を制御できます。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。

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

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

```
      nodeSelector:
        "node-role.kubernetes.io/infra": "true"
  hive:
    spec:
      metastore:
        nodeSelector:
          "node-role.kubernetes.io/infra": "true"
      server:
        nodeSelector:
          "node-role.kubernetes.io/infra": "true"
```

3.3. 永続ストレージの設定

メータリングでは、**metering-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。

3.3.1. 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または削除しません。メータリングをアンインストールする際に、メータリングデータを保存するために使用される S3 バケットは手動でクリーンアップする必要があります。

ストレージに Amazon S3 を使用するには、以下のサンプル **s3-storage.yaml** ファイルの **spec.storage** セクションを編集します。

```
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。オプションで、バケット内のパスを指定できます。
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、以下のサンプルを参照してください。

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。

テンプレートとして以下のシークレットサンプルを使用します。

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。

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

以下のコマンドを使用してシークレットを作成できます。

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。

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

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。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/*",
                "arn:aws:s3:::operator-metering-data"
```

```
        ]
      }
    ]
}
```

spec.storage.hive.s3.createBucket を **true** に設定しているか、または未設定にしている場合、以下の **aws/read-write-create.json** ファイルを使用する必要があります。このファイルには、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ます。

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

3.3.2. 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するには、以下のサンプルの **s3-compatible-storage.yaml** ファイルの **spec.storage** セクションを編集します。

```
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、以下のサンプルを参照してください。

テンプレートとして以下のシークレットサンプルを使用します。

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

3.3.3. 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。以下のサンプルの **azure-blob-storage.yaml** ファイルで **spec.storage** セクションを編集します。

```
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、以下のサンプルを参照してください。

3 オプションで、データを格納するディレクトリーを指定できます。

テンプレートとして以下のシークレットサンプルを使用します。

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

以下のコマンドを使用してシークレットを作成できます。

-

```
oc create secret -n openshift-metering generic your-azure-secret --from-literal=azure-storage-account-name=your-storage-account-name --from-literal=azure-secret-access-key-your-secret-key
```

3.3.4. 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。以下のサンプルの **gcs-storage.yaml** ファイルで **spec.storage** セクションを編集します。

```
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、以下の例を参照してください。

以下のサンプルのシークレットをテンプレートとして使用します。

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

以下のコマンドを使用してシークレットを作成できます。

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

3.3.5. 共有ボリュームへのデータの保存

注記

NFS をメータリングと併用することは推奨されません。

メータリングにはデフォルトでストレージがありませんが、ReadWriteMany PersistentVolume または ReadWriteMany PersistentVolume をプロビジョニングする StorageClass を使用できます。

手順

- ストレージに ReadWriteMany PersistentVolume を使用するには、以下の **shared-storage.yaml** ファイルを変更します。

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "sharedPVC"
      sharedPVC:
        claimName: "metering-nfs" 1
        # uncomment the lines below to provision a new PVC using the specified 2
        # storageClass.
        # 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 PersistentVolumeClaim (PVC) の名前に設定します。これは、動的ボリュームプロビジョニングがない場合や、PersistentVolume 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 StorageClass に設定します。これにより、動的ボリュームの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できます。

3.4. HIVE メタストアの設定

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、メタデータはこの情報を、Pod に割り当てられる **PersistentVolume** 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。

3.4.1. PersistentVolume の設定

デフォルトで、Hive が動作するために 1 つの PersistentVolume 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 PersistentVolumeClaim (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 StorageClass 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。

3.4.1.1. Hive メタストア用のストレージクラスの設定

hive-metastore-db-data PVC に StorageClass を設定し、指定するには、StorageClass を MeteringConfig に指定します。StorageClass セクションのサンプルは以下の **metastore-storage.yaml** ファイルに含まれます。

```
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 StorageClass の名前に置き換えます。値を **null** 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 StorageClass を使用します。

3.4.1.2. 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。

3.4.2. 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。

4 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、ドライバー、ユーザー名、およびパスワード)。

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"
          username: "REPLACEME"
          password: "REPLACEME"
```

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。

3.5. REPORTING-OPERATOR の設定

reporting-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。Operator の設定は主に **MeteringConfig** ファイルを使用して行われます。

3.5.1. Prometheus 接続

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 ConfigMap を使用して挿入できます。以下の例を参照してください。

```
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。

reporting-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。以下の例を参照してください。

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

3.5.2. 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 **reporting-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。

3.5.2.1. 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、**reporting-operator** を、 **reporting-operator** のコンテナーおよび OpenShift 認証プロキシー (auth-proxy) 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/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。

3.5.2.1.1. 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。

3.5.2.1.2. ユーザー名とパスワードを使用した認証

htpasswd ファイル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して基本認証を実行できます。デフォルトで、空の htpasswd データを含むシークレットを作成します。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。

上記の設定を MeteringConfig に指定した後は、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。

3.5.2.2. 認証の手動設定

reporting-operatorで OAuth を手動で設定するか、または無効にするには、MeteringConfig で **spec.tls.enabled: false** を設定する必要があります。

警告

これは、**reporting-operator**、presto、および hive 間のすべての TLS/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、**reporting-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、**reporting-operator** 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に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.
```

3.5.2.2.1. 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 serviceAccount 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 **serviceAccount** 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

metering-operator は、**spec.permissions** セクションにサブジェクトの一覧を指定して、RoleBindings 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、**reporting-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、**reporting-operator** および **metering-operator serviceAccounts** 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。

3.5.2.2.2. 基本認証 (ユーザー名/パスワード)

基本認証では、**reporting-operator.spec.authproxy.htpasswd.data**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。

3.6. AWS 請求情報の関連付けの設定

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「Turning on the AWS Cost and Usage Report 」を参照してください。

1 バケット、プレフィックス、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、**reporting-operator**でエラーが発生する場合があります。

第4章 REPORT

4.1. REPORT について

Report 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する API オブジェクトです。これらは、実行する実際の SQL クエリーを提供する **ReportQueries** や、ReportQuery および Report で利用できるデータを定義する **ReportDataSources** などの他のメータリングリソースを使用して構成されます。

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定義の **ReportQueries** および **ReportDataSources** によって追加設定なしで対応されるため、事前定義されている内容で対応できないユースケースでない限り、独自の定義は不要になります。

4.1.1. 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 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。Report は、テーブルのデータを reporting-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、Report 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、Report は ReportDataSource 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。Report 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。

4.1.1.1. スケジュールが設定された Report 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、Report 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
```

4.1.1.2. スケジュールなしのサンプル Report (1 回のみ実行)

以下のサンプル Report 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"
```

4.1.1.3. クエリー

レポートの生成に使用される ReportQuery 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。

oc CLI を使用して、利用可能な ReportQuery オブジェクトの一覧を取得します。

```
$ oc -n openshift-metering get reportqueries
NAME                                   AGE
cluster-cpu-capacity                   23m
cluster-cpu-capacity-raw               23m
cluster-cpu-usage                      23m
cluster-cpu-usage-raw                  23m
cluster-cpu-utilization                23m
cluster-memory-capacity                23m
cluster-memory-capacity-raw            23m
cluster-memory-usage                   23m
cluster-memory-usage-raw               23m
cluster-memory-utilization             23m
cluster-persistentvolumeclaim-request  23m
namespace-cpu-request                  23m
namespace-cpu-usage                    23m
namespace-cpu-utilization              23m
namespace-memory-request               23m
namespace-memory-usage                 23m
namespace-memory-utilization           23m
namespace-persistentvolumeclaim-request  23m
namespace-persistentvolumeclaim-usage  23m
node-cpu-allocatable                   23m
node-cpu-allocatable-raw               23m
node-cpu-capacity                      23m
node-cpu-capacity-raw                  23m
node-cpu-utilization                   23m
node-memory-allocatable                23m
node-memory-allocatable-raw            23m
node-memory-capacity                   23m
node-memory-capacity-raw               23m
node-memory-utilization                23m
persistentvolumeclaim-capacity         23m
persistentvolumeclaim-capacity-raw     23m
persistentvolumeclaim-phase-raw        23m
persistentvolumeclaim-request          23m
persistentvolumeclaim-request-raw      23m
persistentvolumeclaim-usage            23m
persistentvolumeclaim-usage-raw        23m
```

```
persistentvolumeclaim-usage-with-phase-raw   23m
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 サフィックスのある ReportQuery 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- のプレフィックスが付けられたクエリーは namespace 別に Pod CPU/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- のプレフィックスが付けられたクエリーは **namespace-** のプレフィックス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- のプレフィックス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- のプレフィックスが付けられたクエリーは AWS に固有のものです。**aws** のサフィックスが付けられたクエリーは、サフィックス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。

フィールドの詳細の一覧については、**oc** CLI を使用して ReportQuery 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
```

以下のような出力が表示されるはずです。

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

```
    unit: kubernetes_namespace
  - name: pod_request_memory_byte_seconds
    type: double
    unit: byte_seconds
```

4.1.1.4. スケジュール

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
...
```

4.1.1.4.1. 期間

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 * * * *"**

4.1.1.5. reportingStart

既存データに対する Report の実行をサポートするには、 **spec.reportingStart** フィールドを RFC3339 timestamp に設定し、Report 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。1 つの重要な点として、これにより、reporting-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点に留意してください。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、Report 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 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"
```

4.1.1.6. reportingEnd

指定された時点までのみ実行されるように Report 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、Report 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、Report は永久に実行されるか、または **reportingEnd** が Report 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"
```

4.1.1.7. 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。

4.1.1.8. inputs

Report の **spec.inputs** フィールドは、ReportQuery 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。

名前と値のペアの一覧です。

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

inputs の **name** は ReportQuery の **inputs** 一覧に存在している必要があります。inputs の **value** は inputs の **type** に適切なタイプである必要があります。

4.1.1.9. ロールアップレポート

Report 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、レポートの生成のタイミングを長期間にまたがって分散し、月次レポートで月全体のすべてのデータをクエリーし、追加するのではなく、タスクを日次レポートに分割し、それぞれのレポートが 30 番目のデータに対して実行されるようにします。

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 テンプレートプロセッサーは、Report 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。

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

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

4.1.1.9.1. 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 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。

4.2. ストレージの場所

StorageLocation は、データが reporting-operator によって保存される場所を設定するカスタムリソースです。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 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。

4.2.1. StorageLocation の例

この最初の例は、組み込み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 を使用するよう設定されており、デフォルトで、データは Hive がストレージ (HDFS、S3、または ReadWriteMany 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 はデータを Presto に保管するように設定されます。DatabaseName および unmanagedDatabase のみは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、この StorageLocation は能動的に管理されず、databaseName がすでに Hive に存在することが予想されます。**false**の場合、reporting-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、プレフィックス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いくつかの追加のオプションフィールドがあります。

- (オプション) 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。
- (オプション) 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。
- (オプション) 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。

4.2.2. デフォルトの StorageLocation

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。StorageLocation 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、エラーがログに記録され、Operator はデフォルトがないと見なします。

```
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5章 メータリングの使用

前提条件

- メータリングのインストール
- Report 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。

5.1. REPORT の作成

Report 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。

Report を作成するには、**oc**を使用して、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 に作成します。

前提条件

- メータリングがインストールされている。

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、Report 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) Report 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などが適切な名前に含まれます。

(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 を作成します。

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

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

3. 以下のコマンドで、Report 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

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

5.2. REPORT 結果の表示

Report の結果を表示するには、reporting-api **Route** をクエリーし、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。Report 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。
- Report 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. **reporting-api** への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. 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。

```
$ reportName=namespace-cpu-request-2019 1
$ reportFormat=csv 2
$ curl --insecure -H "Authorization: Bearer ${token}"
"https://${meteringRoute}/api/v1/reports/get?
name=${reportName}&namespace=openshift-metering&format=$reportFormat"
```

1. **reportName** を作成した **Report** の名前に設定します。
2. **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。

応答は以下のようになります (出力例は **reportName=namespace-cpu-request-2019** および **reportFormat=csv** に関連します)。

```
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6章 メータリングの使用例

以下のサンプル Report 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。

前提条件

- メータリングのインストール
- レポートの作成および表示の詳細を確認します。

6.1. クラスター容量の毎時および日次の測定

以下の Report 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 Report は毎時 Report の結果を集計することで機能します。

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名前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す。

6.2.1 回のみ実行される REPORT を使用したクラスター使用状況の測定

以下の Report 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。Report 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名前を変更するようにしてください。

2 Report 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を使用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 これは Report に対し、保存および適用後すぐに実行するように指示します。

6.3. 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. Report 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 Report の名前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7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

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

7.1. 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 またはシークレットなど、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。

7.1.1. 十分なコンピュートリソースはない

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場合に共通に生じる問題。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。

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 Compute 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。

7.1.2. StorageClass が設定されていない

メータリングでは、デフォルトの StorageClass 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。

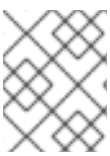
クラスターに設定された StorageClass があるかどうかをチェックする方法、デフォルトの設定方法、およびメータリングをデフォルト以外の StorageClass を使用するように設定する方法についての詳細は、メータリングの設定方法についてのドキュメントを参照してください。

7.1.3. 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。

7.2. 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。

7.2.1. レポート Operator ログの取得

以下のコマンドは、**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
```

7.2.2. 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、以下のコマンドを実行します。

```
$ 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;

                        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>
```

7.2.3. 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;
+----------------------------------------------------+
|                      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>
```

7.2.4. Hive Web UI へのポート転送

次のコマンドを実行します。

```
$ 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 インターフェースを表示します。

7.2.5. hdfs へのポート転送

namenode に対して以下を実行します。

```
$ 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 インターフェースを表示します。

最初のデータノードに対して以下を実行します。

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

他のデータノードをチェックするには、上記のコマンドを実行し、**hdfs-datanode-0** を情報を表示する Pod に置き換えます。

7.2.6. 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。

7.2.6.1. 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)。

7.2.6.2. 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。これは以下のコマンドで確認できます。

```
$ oc -n openshift-metering get meteringconfig operator-metering -o json | jq '.status'
```

第8章 メータリングのアンインストール

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。

8.1. OPENSHIFT CONTAINER PLATFORM からのメータリングのアンインストール

メータリングをクラスターから削除できます。

前提条件

- メータリングがインストールされていること

手順

メータリングを削除するには、以下を実行します。

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。
2. メータリング Operator を検索し、 メニュー**Uninstall Operator** を選択します。
3. ダイアログボックスで、**Remove** をクリックしてメータリングをアンインストールします。

注記

メータリングは S3 バケットデータを管理または削除しません。ストレージに Amazon S3 を使用している場合、メータリングデータを保存するために使用される S3 バケットは手動でクリーンアップする必要があります。