OpenShift Container Platform 4.8

미터링

OpenShift Container Platform에서 미터링 구성 및 사용

Last Updated: 2023-04-07

OpenShift Container Platform 4.8 미터링

OpenShift Container Platform에서 미터링 구성 및 사용

법적 공지

초록

이 문서에서는 OpenShift Container Platform에서 미터링을 구성 및 사용하는 방법에 대한 지침을 설명합니다.

차례

1장. 미터링 정보

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

1.1. 미터링 개요

미터링은 다양한 데이터 소스의 데이터를 처리하기 위해 보고서를 작성할 수 있는 일반적인 목적 데이터 분석 도구입니다. 클러스터 관리자는 미터링을 사용하여 클러스터에서 발생하는 상황을 분석할 수 있습니다. 자체적으로 작성하거나 사전 정의된 SQL 쿼리를 사용하여 사용 가능한 다양한 데이터 소스에서 데이터를 처리하는 방법을 정의할 수 있습니다.

미터링은 기본 데이터 소스로 Prometheus를 사용하는 클러스터 내부 메트릭 데이터에 주로 중점을 두고 미터링 사용자는 Pod, 네임스페이스 및 대부분의 기타 Kubernetes 리소스에서 보고할 수 있습니다.

OpenShift Container Platform 4.x 클러스터 이상에서 미터링을 설치할 수 있습니다.

1.1.1. 미터링 설치

OpenShift Container Platform 4.x 이상에서 CLI 및 웹 콘솔을 사용하여 미터링을 설치할 수 있습니다. 자세한 내용은 미터링 설치를 참조하십시오.

1.1.2. 미터링 업그레이드

Metering Operator 서브스크립션을 업데이트하여 미터링을 업그레이드할 수 있습니다. 다음 작업을 검토합니다.

- **MeteringConfig** 사용자 정의 리소스는 미터링 설치에 대한 모든 설정 세부 정보를 지정합니다. 미터링 스택을 처음 설치하면 기본 **MeteringConfig** 사용자 정의 리소스가 생성됩니다. 문서의 예제를 사용하여 이 기본 파일을 수정합니다.
- 보고서 사용자 지정 리소스 는 SQL 쿼리를 사용하여 주기적인 ETL(Extract Transform and Load) 작업을 관리하는 방법을 제공합니다. Report는 실행할 실제 SQL 쿼리를 제공하는 **ReportQuery** 리소스와 **ReportQuery** 및 **Report** 리소스에 사용할 수 있는 데이터를 정의하는 **ReportDataSource** 리소스 등의 기타 미터링 리소스로 구성됩니다.

1.1.3. 미터링 사용

미터링을 사용하여 보고서를 작성하고 보고서 결과를 볼 수 있습니다. 자세한 내용은 미터링 사용 예를 참조하십시오.

1.1.4. 미터링 문제 해결

다음 섹션을 사용하여 미터링과 관련된 특정 문제를 해결할 수 있습니다.

- 컴퓨팅 리소스가 충분하지 않음

- **StorageClass** 리소스가 구성되지 않음
- secret이 올바르게 구성되지 않은 경우

1.1.5. 미터링 디버깅

다음 섹션을 사용하여 미터링과 관련된 특정 문제를 디버깅 할 수 있습니다.

- Operator 로그 보고
- presto-cli를 사용하여 Presto 쿼리
- Beeline을 사용하여 Hive 쿼리
- Hive 웹 UI로 포트-전달
- HDFS로의 포트-전달
- 미터링 Ansible Operator

1.1.6. 미터링 설치 제거

OpenShift Container Platform 클러스터에서 미터링 리소스를 제거하고 정리할 수 있습니다. 자세한 내용은 미터링 설치 제거를 참조하십시오.

1.1.7. 미터링 리소스

미터링에는 많은 리소스가 있으며, 기능 미터링을 보고하는 기능과 함께 미터링 배포 및 설치를 관리하는 데 사용할 수 있습니다.

미터링은 다음 CRD(Custom Resource Definitions)를 사용하여 관리합니다.

MeteringConfig	배포에 대해 미터링 스택을 구성합니다. 미터링 스택을 구성하는 각 구성 요소를 제어하기 위한 사용자 정의 및 구성 옵션이 포함되어 있습니다.
보고서	사용할 쿼리, 쿼리를 실행할 빈도 및 결과를 저장할 위치를 제어합니다.
ReportQuery	**ReportDataSource**리소스에 포함된 데이터에서 분석을 수행하는 데 사용되는 SQL 쿼리를 포함합니다.
ReportDataSource	**ReportQuery** 및 **Report** 리소스에서 사용할 수 있는 데이터를 제어합니다. 미터링 내에서 사용할 수 있도록 다양한 데이터베이스에 대한 액세스를 구성할 수 있습니다.

2장. 미터링 설치

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

미터링을 클러스터에 설치하기 전에 다음 섹션을 검토합니다.

미터링 설치를 시작하려면 먼저 OperatorHub에서 Metering Operator를 설치합니다. 다음으로 **MeteringConfig** 사용자 정의 리소스(CR)를 생성하여 미터링 인스턴스를 구성합니다. Metering Operator를 설치하면 문서의 예제를 사용하여 수정할 수 있는 기본 **MeteringConfig** 리소스가 생성됩니다. **MeteringConfig** 리소스를 생성한 후 미터링 스택을 설치합니다. 마지막으로 설치를 확인합니다.

2.1. 사전 요구 사항

미터링에는 다음 구성 요소가 필요합니다.

- 동적 볼륨 프로비저닝을 위한 **StorageClass**리소스입니다. 미터링은 다양한 스토리지 솔루션을 지원합니다.
- 4GB 메모리 및 CPU 코어 4개를 사용할 수 있는 클러스터 용량과 2개의 CPU 코어 및 2GB 메모리 용량을 사용할 수 있는 노드가 한 개 이상 있습니다.
- 미터링에 의해 설치된 가장 큰 단일 Pod에 필요한 최소 리소스는 2GB의 메모리와 CPU 코어 2개입니다.
    - 메모리와 CPU 소비는 더 낮을 수 있지만 보고서를 실행하거나 대규모 클러스터의 데이터를 수집할 때 급증할 수 있습니다.

2.2. METERING OPERATOR 설치

Metering Operator를 배포하여 미터링을 설치할 수 있습니다. Metering Operator는 미터링 스택의 구성 요소를 생성하고 관리합니다.

참고

웹 콘솔을 사용하거나 CLI에서 **oc new-project** 명령을 사용하여 **openshift-**로 시작하는 프로젝트를 생성할 수 없습니다.

참고

Metering Operator가 **openshift-metering** 이외의 네임스페이스를 사용하여 설치하는 경우 CLI를 사용하여 미터링 보고서만 볼 수 있습니다. 설치 단계를 통해 **openshift-metering** 네임스페이스를 사용하는 것이 좋습니다.

2.2.1. 웹 콘솔을 사용하여 미터링 설치

OpenShift Container Platform 웹 콘솔을 사용하여 Metering Operator를 설치할 수 있습니다.

절차

1. **oc create -f <file-name>.yaml** 명령으로 Metering Operator에 대한 네임스페이스 오브젝트 YAML 파일을 생성합니다. CLI를 사용하여 네임스페이스를 생성해야 합니다. 예를 들어 **metering-namespace.yaml**은 다음과 같습니다.

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

(1) **openshift-metering** 네임스페이스에 미터링을 배포하는 것이 좋습니다.

(2) 피연산자 Pod에 대한 특정 노드 선택기를 구성하기 전에 이 주석을 포함합니다.

2. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub**를 클릭합니다. Metering Operator를 찾으려면 **metering**에 대해 필터링합니다.

3. **미터링** 카드를 클릭하고 패키지 설명을 검토한 다음 **설치**를 클릭합니다.

4. **업데이트 채널**, **설치 모드** 및 **승인 전략**을 선택합니다.

5. **설치**를 클릭합니다.

6. **Operators → 설치된 Operators** 페이지로 전환하여 Metering Operator가 설치되었는지 확인합니다. 설치가 완료되면 Metering Operator에 **성공**이라는 **상태**가 있습니다.

참고

Metering Operator가 표시되는 데 몇 분이 걸릴 수 있습니다.

7. Operator **세부 정보**는 **설치된 Operator** 페이지에서 **미터링**을 클릭합니다. **세부 정보** 페이지에서 미터링과 관련된 다른 리소스를 생성할 수 있습니다.

미터링 설치를 완료하려면 **MeteringConfig** 리소스를 생성하여 미터링을 구성하고 미터링 스택의 구성 요소를 설치합니다.

2.2.2. CLI를 사용하여 미터링 설치

OpenShift Container Platform CLI를 사용하여 Metering Operator를 설치할 수 있습니다.

프로세스

1. Metering Operator의 **Namespace** 오브젝트 YAML 파일을 생성합니다. CLI를 사용하여 네임스페이스를 생성해야 합니다. 예를 들어 **metering-namespace.yaml**은 다음과 같습니다.

```
apiVersion: v1
```

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

(1) **openshift-metering** 네임스페이스에 미터링을 배포하는 것이 좋습니다.

(2) 피연산자 Pod에 대한 특정 노드 선택기를 구성하기 전에 이 주석을 포함합니다.

2. **Namespace** 오브젝트를 생성합니다.

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

예를 들면 다음과 같습니다.

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

3. **OperatorGroup** 오브젝트 YAML 파일을 생성합니다. 예를 들어 **Metering-og**는 다음과 같습니다.

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

(1) 이름은 임의의 이름입니다.

(2) **openshift-metering** 네임스페이스를 지정합니다.

4. **Subscription** 오브젝트 YAML 파일을 생성하여 Metering Operator에 네임스페이스를 등록합니다. 이 오브젝트는 **redhat-operators** 카탈로그 리소스에서 가장 최근 릴리스된 버전을 대상으로 합니다. 예를 들어 **metering-sub.yaml**은 다음과 같습니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metering-ocp (1)
  namespace: openshift-metering (2)
spec:
  channel: "4.8" (3)
  source: "redhat-operators" (4)
  sourceNamespace: "openshift-marketplace"
  name: "metering-ocp"
  installPlanApproval: "Automatic" (5)
```

(1) 이름은 임의의 이름입니다.

2 **openshift-metering** 네임스페이스를 지정해야 합니다.

3 4.8을 채널로 지정합니다.

4 **metering-ocp** 패키지 매니페스트를 포함하는 **redhat-operators** 카탈로그 리소스를 지정합니다. OpenShift Container Platform이 제한된 네트워크(연결이 끊긴 클러스터)에 설치된 경우 OLM(Operator Lifecycle Manager)을 구성할 때 생성된 **CatalogSource** 오브젝트의 이름을 지정합니다.

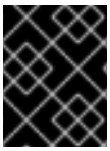
5 "자동" 설치 계획 승인을 지정합니다.

2.3. 미터링 스택 설치

Metering Operator를 클러스터에 추가한 후 미터링 스택을 설치하여 미터링 구성 요소를 설치할 수 있습니다.

2.4. 사전 요구 사항

- 설정 옵션 검토
- **MeteringConfig** 리소스를 생성합니다. 다음 프로세스를 시작하여 기본 **MeteringConfig**를 생성한 다음 문서의 예제를 사용하여 특정 설치에 대한 기본 파일을 수정할 수 있습니다. 다음 주제를 검토하여 **MeteringConfig** 리소스를 생성합니다.
  - 구성 옵션의 경우 미터링 구성 정보를 검토합니다.
  - 최소한으로 영구 스토리지를 구성하고 Hive 메타 저장소를 구성해야 합니다.



중요

openshift-metering 네임스페이스에는 하나의 **MeteringConfig** 리소스만 있을 수 있습니다. 다른 구성은 지원되지 않습니다.

프로세스

1. 웹 콘솔에서 **openshift-metering** 프로젝트의 Metering Operator에 대한 **Operator 상세 정보** 페이지에 있는지 확인합니다. **Operators → 설치된 Operator**를 클릭한 다음 Metering Operator를 선택하여 이 페이지로 이동할 수 있습니다.
2. **제공된 API** 아래의 미터링 설정 카드에서 **인스턴스 생성**을 클릭합니다. 이렇게 하면 설정을 정의할 수 있는 기본 **MeteringConfig** 리소스 파일을 사용하여 YAML 편집기가 열립니다.

참고

예를 들어 설정 파일 및 지원되는 모든 설정 옵션의 경우 미터링 문서 구성을 검토하십시오.

3. **MeteringConfig** 리소스를 YAML 편집기에 입력하고 **만들기**를 클릭합니다.

MeteringConfig 리소스는 미터링 스택에 필요한 리소스를 생성하기 시작합니다. 이제 설치를 검증하도록 이동할 수 있습니다.

2.5. 미터링 설치 확인

다음 점검 중 하나를 수행하여 미터링 설치를 확인할 수 있습니다.

- 미터링 버전에 대해 Metering Operator **ClusterServiceVersion** (CSV)를 확인합니다. 웹 콘솔 또는 CLI를 통해 이 작업을 수행할 수 있습니다.

  프로세스(UI)

  1. **openshift-metering** 네임스페이스에서 **Operators → 설치된 Operators**로 이동합니다.
  2. **Metering Operator**를 클릭합니다.
  3. **서브스크립션 세부 정보**에 대해 **서브스크립션**을 클릭합니다.
  4. **설치된 버전**을 확인합니다.

  프로세스(CLI)

  - **openshift-metering** 네임스페이스에서 Metering Operator CSV를 확인합니다.

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

    출력 예

    ```
    NAME                                    DISPLAY                   VERSION                 REPLACES
    PHASE
    elasticsearch-operator.4.8.0-202006231303.p0   OpenShift Elasticsearch Operator
    4.8.0-202006231303.p0              Succeeded
    metering-operator.v4.8.0                Metering                  4.8.0
    Succeeded
    ```

- **openshift-metering** 네임스페이스의 필요한 모든 Pod가 생성되었는지 확인합니다. 웹 콘솔 또는 CLI를 통해 이 작업을 수행할 수 있습니다.

  참고

  많은 Pod는 자체적으로 준비가 되었다고 간주되기 전에 다른 구성 요소를 사용합니다. 다른 Pod가 시작하는 데 너무 오래 걸리는 경우 일부 Pod가 다시 시작할 수 있습니다. 이는 Metering Operator 설치 중에 예상됩니다.

  프로세스(UI)

  - 미터링 네임스페이스에서 **워크로드 → Pod**로 이동하여 Pod가 생성되고 있는지 확인합니다. 미터링 스택을 설치한 후 몇 분이 걸릴 수 있습니다.

  프로세스(CLI)

  - **openshift-metering** 네임스페이스의 필요한 모든 Pod가 생성되었는지 확인합니다.

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

    출력 예

    ```
    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
```

- **EARLIEST METRIC** 열의 유효한 타임 스탬프에 표시된 **ReportDataSources** 리소스가 데이터를 가져오기 시작하는지 확인합니다. 이 작업은 몇 분 정도 걸릴 수 있습니다. 데이터를 가져오지 않는 "-raw" **ReportDataSource** 리소스를 필터링합니다.

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

출력 예

```
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
```

모든 Pod가 준비되어 있고 데이터가 가져오는 것을 확인한 후 미터링을 사용하여 데이터를 수집하고 클러스터를 보고할 수 있습니다.

2.6. 추가 리소스

- 설정 단계 및 사용 가능한 스토리지 플랫폼에 대한 자세한 내용은 영구 스토리지 구성을 참조하십시오.

- Hive를 구성하는 단계는 Hive 메타 저장소 구성 을 참조하십시오.

3장. 미터링 업그레이드

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

Metering Operator 서브스크립션을 업데이트하여 미터링을 4.8로 업그레이드할 수 있습니다.

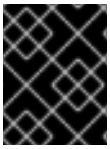
3.1. 사전 요구 사항

- 클러스터가 4.8로 업데이트되었습니다.
- OperatorHub에서 Metering Operator가 설치됩니다.

참고

Metering Operator를 4.8로 수동 업그레이드해야 합니다. 미터링은 이전 설치에서 "자동" **승인 전략**을 선택한 경우 자동으로 업그레이드되지 않습니다.

- MeteringConfig 사용자 정의 리소스가 구성되어 있습니다.
- 미터링 스택이 설치되어 있습니다.
- 모든 Pod가 준비되었는지 확인하여 미터링 상태가 정상인지 확인합니다.



중요

미터링을 설치하거나 업그레이드한 후 미터링 스토리지 구성을 수정하면 잠재적인 데이터 손실이 발생할 수 있습니다.

프로세스

1. 웹 콘솔에서 **Operators → 설치된 Operators**를 클릭합니다.
2. **openshift-metering** 프로젝트를 선택합니다.
3. **Metering Operator**를 클릭합니다.
4. **서브스크립션 → 채널**을 클릭합니다.
5. **서브스크립션 업데이트 채널 변경** 창에서 **4.8**을 선택하고 **저장**을 클릭합니다.

참고

다음 단계로 진행하기 전에 서브스크립션을 업데이트할 수 있도록 몇 초 정도 기다립니다.

6. **Operators → 설치된 Operators**를 클릭합니다.
Metering Operator가 4.8로 표시됩니다. 예를 들면 다음과 같습니다.

```
Metering
4.8.0-202107012112.p0 provided by Red Hat, Inc
```

검증

다음 점검 중 하나를 수행하여 미터링 업그레이드를 확인할 수 있습니다.

- 새 미터링 버전의 Metering Operator CSV(Cluster Service Version)를 확인합니다. 웹 콘솔 또는 CLI를 통해 이 작업을 수행할 수 있습니다.

프로세스(UI)

1. 미터링 네임스페이스에서 **Operators → 설치된 Operators**로 이동합니다.
2. **Metering Operator**를 클릭합니다.
3. **서브스크립션 세부 정보**에 대해 **서브스크립션**을 클릭합니다.
4. 업그레이드된 미터링 버전에 대해 **설치된 버전**을 확인합니다. **시작 버전**에서는 업그레이드하기 전에 미터링 버전을 보여줍니다.

프로세스(CLI)

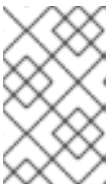
- Metering Operator CSV를 확인합니다.

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

4.7에서 4.8로 미터링 업그레이드의 출력 예

```
NAME                                 DISPLAY     VERSION              REPLACES
PHASE
metering-operator.4.8.0-202107012112.p0   Metering    4.8.0-202107012112.p0
metering-operator.4.7.0-202007012112.p0   Succeeded
```

- **openshift-metering** 네임스페이스의 필요한 모든 Pod가 생성되었는지 확인합니다. 웹 콘솔 또는 CLI를 통해 이 작업을 수행할 수 있습니다.



참고

많은 Pod는 자체적으로 준비가 되었다고 간주되기 전에 다른 구성 요소를 사용합니다. 다른 Pod가 시작하는 데 너무 오래 걸리는 경우 일부 Pod가 다시 시작할 수 있습니다. 이는 Metering Operator 업그레이드 중에 예상됩니다.

프로세스(UI)

- 미터링 네임스페이스에서 **워크로드 → Pod**로 이동하여 Pod가 생성되고 있는지 확인합니다. 미터링 스택을 업그레이드한 후 몇 분이 걸릴 수 있습니다.

프로세스(CLI)

- **openshift-metering** 네임스페이스의 필요한 모든 Pod가 생성되었는지 확인합니다.

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

출력 예

```
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
```

- **NEWEST METRIC** 열의 유효한 타임 스탬프에 표시된 **ReportDataSources** 리소스가 새 데이터를 가져오고 있는지 확인합니다. 이 작업은 몇 분 정도 걸릴 수 있습니다. 데이터를 가져오지 않는 "-raw" **ReportDataSource** 리소스를 필터링합니다.

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

NEWEST METRIC 열의 타임 스탬프는 **ReportDataSources** 리소스가 새 데이터를 가져오기 시작하는지를 나타냅니다.

출력 예

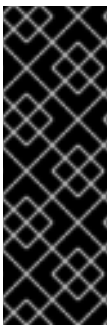
```
NAME                                EARLIEST METRIC        NEWEST METRIC          IMPORT START          IMPORT END            LAST IMPORT TIME       AGE
node-allocatable-cpu-cores          2021-07-01T21:10:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:56:44Z   23h
node-allocatable-memory-bytes       2021-07-01T21:10:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:52:07Z   23h
node-capacity-cpu-cores             2021-07-01T21:10:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:56:52Z   23h
node-capacity-memory-bytes          2021-07-01T21:10:00Z   2021-07-02T19:57:00Z   2021-07-01T19:10:00Z   2021-07-02T19:57:00Z   2021-07-02T19:57:03Z   23h
persistentvolumeclaim-capacity-bytes   2021-07-01T21:09:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:56:46Z   23h
persistentvolumeclaim-phase         2021-07-01T21:10:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:52:36Z   23h
persistentvolumeclaim-request-bytes   2021-07-01T21:10:00Z   2021-07-02T19:57:00Z   2021-07-01T19:10:00Z   2021-07-02T19:57:00Z   2021-07-02T19:57:03Z   23h
persistentvolumeclaim-usage-bytes   2021-07-01T21:09:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:52:02Z   23h
pod-limit-cpu-cores                 2021-07-01T21:10:00Z   2021-07-02T19:57:00Z   2021-07-01T19:10:00Z   2021-07-02T19:57:00Z   2021-07-02T19:57:02Z   23h
pod-limit-memory-bytes              2021-07-01T21:10:00Z   2021-07-02T19:58:00Z   2021-07-01T19:11:00Z   2021-07-02T19:58:00Z   2021-07-02T19:59:06Z   23h
pod-persistentvolumeclaim-request-info   2021-07-01T21:10:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:52:07Z   23h
pod-request-cpu-cores               2021-07-01T21:10:00Z   2021-07-02T19:58:00Z   2021-07-01T19:11:00Z   2021-07-02T19:58:00Z   2021-07-02T19:58:57Z   23h
pod-request-memory-bytes            2021-07-01T21:10:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:55:32Z   23h
pod-usage-cpu-cores                 2021-07-01T21:09:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:54:55Z   23h
pod-usage-memory-bytes              2021-07-01T21:08:00Z   2021-07-02T19:52:00Z   2021-07-01T19:11:00Z   2021-07-02T19:52:00Z   2021-07-02T19:55:00Z   23h
```

```
report-ns-pvc-usage
5h36m
report-ns-pvc-usage-hourly
```

모든 Pod가 준비되어 있고 새 데이터가 가져오는 것을 확인한 후 미터링이 계속 데이터를 수집하여 클러스터에 보고합니다. 이전에 예약된 보고서를 검토하거나 일회성 미터링 보고서를 작성하여 미터링 업그레이드를 확인합니다.

4장. 미터링 구성

4.1. 미터링 구성 정보



중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

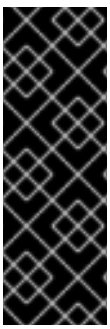
OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

MeteringConfig 사용자 정의 리소스는 미터링 설치에 대한 모든 설정 세부 정보를 지정합니다. 미터링 스택을 처음 설치하면 기본 **MeteringConfig** 사용자 정의 리소스가 생성됩니다. 문서의 예제를 사용하여 이 기본 파일을 수정합니다. 다음 주요 포인트를 유의하십시오.

- 최소한으로 영구 스토리지를 구성 하고 Hive 메타 저장소를 구성 해야 합니다.
- 대부분의 기본 구성 설정이 적용되지만 더 큰 배포 또는 고도의 사용자 지정 배포는 모든 설정 옵션을 주의 깊게 검토해야 합니다.
- 일부 설정 옵션은 설치 후 수정할 수 없습니다.

설치 후 수정할 수 있는 설정 옵션의 경우 **MeteringConfig** 사용자 정의 리소스에서 변경하고 파일을 다시 적용합니다.

4.2. 공통 설정 옵션



중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

4.2.1. 리소스 요청 및 제한

Pod 및 볼륨의 CPU, 메모리 또는 스토리지 리소스 요청 및/또는 제한을 조정할 수 있습니다. 아래 **default-resource-limits.yaml**은 각 구성 요소에 대한 리소스 요청 및 제한 설정의 예를 제공합니다.

```
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. 노드 선택기

특정 노드 집합에서 미터링 구성 요소를 실행할 수 있습니다. 미터링 구성 요소에서 **nodeSelector**를 설정하여 구성 요소가 예약된 위치를 제어합니다. 아래 **node-selectors.yaml** 파일은 각 구성 요소에 대한 노드 선택기를 설정하는 예를 제공합니다.

참고

피연산자 Pod에 대한 특정 노드 선택기를 구성하기 전에 **openshift.io/node-selector: ""** 네임스페이스 주석을 미터링 네임스페이스 YAML 파일에 추가합니다. 주석 값으로 **""**를 지정합니다.

```
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 적절한 값이 설정된 **nodeSelector** 매개변수를 이동하려는 구성 요소에 추가합니다. 표시된 형식으로 **nodeSelector**를 사용하거나 노드에 지정된 값에 따라 키-값 쌍을 사용할 수 있습니다.

참고

피연산자 Pod에 대한 특정 노드 선택기를 구성하기 전에 **openshift.io/node-selector: ""** 네임스페이스 주석을 미터링 네임스페이스 YAML 파일에 추가합니다. 프로젝트에 **openshift.io/node-selector** 주석이 설정되어 있으면 해당 값은 클러스터 전체 **Scheduler** 오브젝트에서 **spec.defaultNodeSelector** 필드의 값보다 우선하여 사용됩니다.

검증

다음 점검 중 하나를 수행하여 미터링 노드 선택기를 확인할 수 있습니다.

- **MeteringConfig** 사용자 정의 리소스에 구성된 노드의 IP에 미터링의 모든 Pod가 올바르게 예약되어 있는지 확인합니다.

  1. **openshift-metering** 네임스페이스의 모든 Pod를 확인합니다.

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

출력에는 **openshift-metering** 네임스페이스에서 실행되는 각 Pod의 **NODE** 및 해당 **IP**가 표시됩니다.

출력 예

```
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** 네임스페이스의 노드를 클러스터의 각 노드 **NAME**과 비교합니다.

```
$ oc get nodes
```

출력 예

```
NAME                                         STATUS   ROLES    AGE   VERSION
ip-10-0-147-106.us-east-2.compute.internal   Ready    master   14h   v1.21.0+6025c28
ip-10-0-150-175.us-east-2.compute.internal   Ready    worker   14h   v1.21.0+6025c28
ip-10-0-175-23.us-east-2.compute.internal    Ready    master   14h   v1.21.0+6025c28
ip-10-0-189-6.us-east-2.compute.internal     Ready    worker   14h   v1.21.0+6025c28
ip-10-0-205-158.us-east-2.compute.internal   Ready    master   14h   v1.21.0+6025c28
ip-10-0-210-167.us-east-2.compute.internal   Ready    worker   14h   v1.21.0+6025c28
```

- **MeteringConfig** 사용자 정의 리소스의 노드 선택기 구성이 미터링 피연산자 Pod가 예약되지 않도록 클러스터 전체 노드 선택기 설정을 적용하지 않는지 확인합니다.
  - 기본적으로 Pod 예약 위치를 표시하는 **spec.defaultNodeSelector** 필드에 대해 클러스터 전체 **Scheduler** 오브젝트를 확인합니다.

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

4.3. 영구 스토리지 구성

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

미터링에는 Metering Operator에서 수집한 데이터를 유지하고 보고서 결과를 저장하기 위해 영구 스토리지가 필요합니다. 다양한 스토리지 공급자 및 스토리지 형식이 지원됩니다. 스토리지 공급자를 선택하고 예제 구성 파일을 수정하여 미터링 설치에 대한 영구 스토리지를 구성합니다.

4.3.1. Amazon S3에 데이터 저장

미터링은 기존 Amazon S3 버킷을 사용하거나 스토리지에 버킷을 생성할 수 있습니다.

참고

Metering 작업은 S3 버킷 데이터를 관리하거나 삭제하지 않습니다. 미터링 데이터를 저장하는 데 사용되는 S3 버킷은 수동으로 정리해야 합니다.

절차

1. 아래 **s3-storage.yaml** 파일에서 **spec.storage** 섹션을 편집합니다.

s3-storage.yaml 파일의 예

```
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. 데이터를 저장하려는 버킷의 이름을 지정합니다. 선택 사항: 버킷 내 경로를 지정합니다.
2. 버킷의 리전을 지정합니다.
3. **data.aws-access-key-id** 및 **data.aws-secret-access-key** 필드에 AWS 인증 정보를 포함하는 미터링 네임스페이스의 시크릿 이름입니다. 자세한 내용은 아래 **Secret** 오브젝트 예제를 참조하십시오.

4 기존 S3 버킷을 제공하거나 **CreateBucket** 권한이 있는 IAM 인증 정보를 제공하지 않으려면 이 필드를 **false**로 설정합니다.

2. 다음 **Secret** 오브젝트를 템플릿으로 사용합니다.

AWS Secret 오브젝트의 예

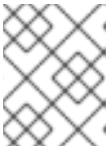
```
apiVersion: v1
kind: Secret
metadata:
  name: my-aws-secret
data:
  aws-access-key-id: "dGVzdAo="
  aws-secret-access-key: "c2VjcmV0Cg=="
```

참고

aws-access-key-id 및 **aws-secret-access-key**의 값은 base64로 인코딩되어야 합니다.

3. 시크릿을 생성합니다.

```
$ 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
```



참고

이 명령은 **aws-access-key-id** 및 **aws-secret-access-key** 값을 자동으로 base64로 인코딩합니다.

aws-access-key-id 및 **aws-secret-access-key** 인증 정보에 버킷에 대한 읽기 및 쓰기 권한이 있어야 합니다. 다음 **aws/read-write.json** 파일은 필요한 권한을 부여하는 IAM 정책을 보여줍니다.

aws/read-write.json 파일의 예

```
{
   "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.createBucket을 **true**로 설정하거나 **s3-storage.yaml**에 설정하지 않은 경우 버킷 생성 및 삭제 권한이 포함된 **aws/read-write-create.json** 파일을 사용해야 합니다.

aws/read-write-create.json 파일의 예

```
{
  "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 호환 스토리지에 데이터 저장

Noobaa와 같은 S3 호환 스토리지를 사용할 수 있습니다.

절차

1. 아래 **s3-compatible-storage.yaml** 파일 예에서 **spec.storage** 섹션을 편집합니다.

   s3-compatible-storage.yaml 파일의 예

   ```
   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 호환 버킷의 이름을 지정합니다.
2. 스토리지의 끝점을 지정합니다.
3. **data.aws-access-key-id** 및 **data.aws-secret-access-key** 필드에 AWS 인증 정보를 포함하는 미터링 네임스페이스의 시크릿 이름입니다. 자세한 내용은 아래 **Secret** 오브젝트 예제를 참조하십시오.

2. 다음 **Secret** 오브젝트를 템플릿으로 사용합니다.

S3 호환 Secret 오브젝트의 예

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

4.3.3. Microsoft Azure에 데이터 저장

데이터를 Azure Blob 스토리지에 저장하려면 기존 컨테이너를 사용해야 합니다.

프로세스

1. **azure-blob-storage.yaml** 파일에서 **spec.storage** 섹션을 편집합니다.

azure-blob-storage.yaml 파일 예

```
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. 컨테이너 이름을 지정합니다.

2 미터링 네임스페이스에 시크릿을 지정합니다. 자세한 내용은 아래 **Secret** 오브젝트 예제를 참조하십시오.

3 선택 사항: 데이터를 저장하려는 디렉터리를 지정합니다.

2. 다음 **Secret** 오브젝트를 템플릿으로 사용합니다.

Azure Secret 오브젝트의 예

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

3. 시크릿을 생성합니다.

```
$ 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 Storage에 데이터 저장

데이터를 Google Cloud Storage에 저장하려면 기존 버킷을 사용해야 합니다.

프로세스

1. 아래 **gcs-storage.yaml** 파일 예에서 **spec.storage** 섹션을 편집합니다.

gcs-storage.yaml 파일 예

```
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 버킷 이름을 지정합니다. 선택적으로 데이터를 저장하려는 버킷 내 디렉터리를 지정할 수 있습니다.

2 미터링 네임스페이스에 시크릿을 지정합니다. 자세한 내용은 아래 **Secret** 오브젝트 예제를 참조하십시오.

2. 다음 **Secret** 오브젝트를 템플릿으로 사용합니다.

Google Cloud Storage Secret 오브젝트의 예

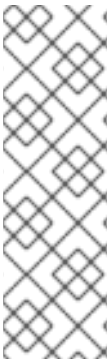
```
apiVersion: v1
kind: Secret
metadata:
  name: my-gcs-secret
data:
  gcs-service-account.json: "c2VjcmV0Cg=="
```

3. 시크릿을 생성합니다.

```
$ 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. 공유 볼륨에 데이터 저장

미터링은 기본적으로 스토리지를 구성하지 않습니다. 그러나 ReadWriteMany PV(영구 볼륨)를 사용하거나 미터링 스토리지에 대한 ReadWriteMany PV를 프로비저닝하는 모든 스토리지 클래스를 사용할 수 있습니다.



참고

NFS를 프로덕션 환경에서 사용하지 않는 것이 좋습니다. RHEL의 NFS 서버를 스토리지 백엔드로 사용하면 미터링 요구 사항을 충족하지 못하고 Metering Operator가 제대로 작동하는 데 필요한 성능을 제공하지 못할 수 있습니다.

마켓플레이스의 다른 NFS 구현에는 PNFS(Parallel Network File System)와 같은 이러한 문제가 발생하지 않을 수 있습니다. pNFS는 분산 및 병렬 기능을 갖춘 NFS 구현입니다. OpenShift Container Platform 핵심 구성 요소에 대해 완료된 테스트에 대한 자세한 내용은 개별 NFS 구현 벤더에 문의하십시오.

프로세스

1. 스토리지에 대한 ReadWriteMany 영구 볼륨을 사용하도록 **shared-storage.yaml** 파일을 수정합니다.

```
apiVersion: metering.openshift.io/v1
kind: MeteringConfig
metadata:
  name: "operator-metering"
spec:
  storage:
    type: "hive"
    hive:
      type: "sharedPVC"
      sharedPVC:
        claimName: "metering-nfs" 1
        # Uncomment the lines below to provision a new PVC using the specified storageClass.
2
        # createPVC: true
        # storageClass: "my-nfs-storage-class"
        # size: 5Gi
```

-

아래 설정 옵션 중 하나를 선택합니다.

1. **storage.hive.sharedPVC.claimName**을 기존 ReadWriteMany 영구 볼륨 클레임(PVC) 이름으로 설정합니다. 동적 볼륨 프로비저닝이 없거나 영구 볼륨 생성 방법에 대한 더 많은 제어가 필요한 경우 이 구성이 필요합니다.

2. **storage.hive.sharedPVC.createPVC**를 **true**로 설정하고 **storage.hive.sharedPVC.storageClass**를 ReadWriteMany 액세스 모드가 있는 스토리지 클래스 이름으로 설정합니다. 이 설정은 동적 볼륨 프로비저닝을 사용하여 자동으로 볼륨을 생성합니다.

2. 미터링을 위해 NFS 서버를 배포하는 데 필요한 다음 리소스 오브젝트를 생성합니다. **oc create -f <file-name>.yaml** 명령을 사용하여 오브젝트 YAML 파일을 생성합니다.

   a. **PersistentVolume** 리소스 오브젝트를 구성합니다.

   nfs_persistentvolume.yaml 예제 파일

```
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** 필드 값.

   b. **nfs-server** 역할을 사용하여 **Pod** 리소스 오브젝트를 구성합니다.

   nfs_server.yaml 예제 파일

```
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 서버 이미지를 설치합니다.

c. **nfs-server** 역할을 사용하여 **Service** 리소스 오브젝트를 구성합니다.

nfs_service.yaml 예제 파일

```
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** 리소스 오브젝트를 구성합니다.

nfs_storageclass.yaml 예제 파일

```
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** 필드 값입니다.

주의

NFS 스토리지 및 관련 리소스 오브젝트의 구성은 미터링 스토리지에 사용하는 NFS 서버 이미지에 따라 다릅니다.

4.4. HIVE 메타 저장소 구성

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

Hive 메타 저장소는 Presto 및 Hive에서 생성된 데이터베이스 테이블에 대한 모든 메타데이터 저장을 담당합니다. 기본적으로 메타 저장소는 Pod에 연결된 영구 볼륨에 포함된 로컬 Derby 데이터베이스에 이 정보를 저장합니다.

일반적으로 Hive 메타 저장소의 기본 설정은 소규모 클러스터에서 작동하지만 사용자는 Hive 메타 저장소 데이터를 저장하기 위해 전용 SQL 데이터베이스를 사용하여 성능을 개선하거나 스토리지 요구 사항을 클러스터에서 이동하고자 할 수 있습니다.

4.4.1. 영구 볼륨 구성

기본적으로 Hive는 작동을 위해 하나의 영구 볼륨이 필요합니다.

hive-metastore-db-data는 기본적으로 필요한 주요 PVC(Persistent Volume Claim)입니다. 이 PVC는 Hive 메타 저장소에서 테이블 이름, 열 및 위치와 같은 테이블에 대한 메타데이터를 저장하는 데 사용됩니다. Presto 및 Hive 서버는 쿼리 처리 시 테이블 메타데이터를 조회하는 데 Presto 및 Hive 서버에서 사용합니다. Hive 메타 저장소 데이터베이스에 MySQL 또는 PostgreSQL을 사용하여 이러한 요구 사항을 제거합니다.

설치하려면 Hive 메타 저장소에서 스토리지 클래스에 있는 동적 볼륨 프로비저닝을 활성화해야 하며 올바른 크기의 영구 볼륨을 수동으로 미리 생성하거나 이미 있는 MySQL 또는 PostgreSQL 데이터베이스를 사용해야 합니다.

4.4.1.1. Hive 메타 저장소의 스토리지 클래스 구성

hive-metastore-db-data 영구 볼륨 클레임에 대해 스토리지 클래스를 구성하고 지정하려면 **MeteringConfig** 사용자 지정 리소스에서 스토리지 클래스를 지정합니다. **class** 필드가 있는 **storage** 섹션의 예는 아래 **metastore-storage.yaml** 파일에 포함되어 있습니다.

```
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 이 라인의 주석을 제거하고 사용할 스토리지 클래스의 이름으로 **null**을 바꿉니다. **null** 값을 남겨 두면 미터링이 클러스터에 기본 스토리지 클래스를 사용합니다.

4.4.1.2. Hive 메타 저장소의 볼륨 크기 구성

아래 **metastore-storage.yaml** 파일을 템플릿으로 사용하여 Hive 메타 저장소의 볼륨 크기를 구성합니다.

```
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** 값을 원하는 용량으로 바꿉니다. 예제 파일은 "5Gi"를 표시합니다.

4.4.2. Hive 메타 저장소에 MySQL 또는 PostgreSQL 사용

미터링의 기본 설치는 Derby라는 포함된 Java 데이터베이스를 사용하도록 Hive를 구성합니다. 이는 더 큰 환경에 적합하지 않으며 MySQL 또는 PostgreSQL 데이터베이스로 바꿀 수 있습니다. 배포에 Hive의 MySQL 또는 PostgreSQL 데이터베이스가 필요한 경우 다음 예제 설정 파일을 사용합니다.

데이터베이스를 제어하는 데 사용할 수 있는 세 가지 설정 옵션은 **url**, **driver** 및 **secretName**입니다.

사용자 이름과 암호로 MySQL 또는 Postgres 인스턴스를 생성합니다. 그런 다음 OpenShift CLI(**oc**) 또는 YAML 파일을 사용하여 시크릿을 생성합니다. 이 시크릿에 대해 생성하는 **secretName**은 **MeteringConfig** 오브젝트 리소스의 **spec.hive.spec.db.secretName** 필드에 매핑해야 합니다.

절차

1. OpenShift CLI(**oc**)를 사용하거나 YAML 파일을 사용하여 시크릿을 생성합니다.
   - 다음 명령을 사용하여 시크릿을 생성합니다.

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

- YAML 파일을 사용하여 시크릿을 생성합니다. 예를 들면 다음과 같습니다.

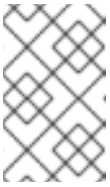
```
apiVersion: v1
kind: Secret
metadata:
  name: <YOUR_SECRETNAME> 1
data:
  username: <BASE64_ENCODED_DATABASE_USERNAME> 2
  password: <BASE64_ENCODED_DATABASE_PASSWORD> 3
```

1. 시크릿의 이름입니다.
2. base64로 인코딩된 데이터베이스 사용자 이름입니다.
3. base64로 인코딩된 데이터베이스 암호입니다.

2. Hive에 MySQL 또는 PostgreSQL 데이터베이스를 사용하려면 설정 파일을 생성합니다.

- Hive에 MySQL 데이터베이스를 사용하려면 아래 설정 파일 예제를 사용합니다. 미터링은 MySQL 서버 버전 5.6, 5.7 및 8.0을 사용하도록 내부 Hive 메타 저장소 구성을 지원합니다.

```
spec:
  hive:
    spec:
      metastore:
        storage:
          create: false
      config:
        db:
          url: "jdbc:mysql://mysql.example.com:3306/hive_metastore" 1
          driver: "com.mysql.cj.jdbc.Driver"
          secretName: "REPLACEME" 2
```



참고

5.6 또는 5.7과 같은 이전 MySQL 서버 버전과 작동하도록 미터링을 구성하는 경우 내부 Hive 메타 저장소를 구성할 때 **enabledTLSProtocols** JDBC URL 매개변수를 추가해야 할 수 있습니다.

1. TLS v1.2 암호화 제품군을 사용하려면 **url을 "jdbc:mysql://<hostname>:<port>/<schema>?enabledTLSProtocols=TLSv1.2로 설정합니다.**
2. base64로 암호화된 사용자 이름과 암호 데이터베이스 인증 정보가 포함된 시크릿의 이름입니다.

spec.hive.config.url을 사용하여 추가 JDBC 매개변수를 전달할 수 있습니다. 자세한 내용은 MySQL Connector/J 8.0 문서를 참조하십시오.

- Hive에 PostgreSQL 데이터베이스를 사용하려면 아래 설정 파일 예제를 사용합니다.

```
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"
```

spec.hive.config.url을 사용하여 추가 JDBC 매개변수를 전달할 수 있습니다. 자세한 내용은 PostgreSQL JDBC 드라이버 문서를 참조하십시오.

4.5. REPORTING OPERATOR 구성

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

Reporting Operator는 Prometheus에서 데이터를 수집하고, Presto에 메트릭을 저장하고, Presto에 대한 보고서 쿼리를 실행하며 HTTP API를 통해 결과를 노출합니다. Reporting Operator 구성은 주로 **MeteringConfig** 사용자 정의 리소스에서 수행됩니다.

4.5.1. Prometheus 연결 보안

OpenShift Container Platform에 미터링을 설치하는 경우 Prometheus는 https://prometheus-k8s.openshift-monitoring.svc:9091/에서 사용할 수 있습니다.

Prometheus에 대한 연결을 보호하려면 기본 미터링 설치에서 OpenShift Container Platform 인증 기관(CA)을 사용합니다. Prometheus 인스턴스에서 다른 CA를 사용하는 경우 구성 맵을 통해 CA를 삽입할 수 있습니다. Prometheus를 사용하여 지정된 전달자 토큰을 사용하도록 Reporting Operator를 구성할 수도 있습니다.

절차

- 구성 맵을 통해 Prometheus 인스턴스에서 사용하는 CA를 삽입합니다. 예를 들면 다음과 같습니다.

```
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-----
```

또는 공개적으로 유효한 인증서에 시스템 인증 기관을 사용하려면 **useServiceAccountCA** 및 **configMap.enabled**를 모두 **false**로 설정합니다.

- Prometheus로 인증할 전달자 토큰을 지정합니다. 예를 들면 다음과 같습니다.

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

4.5.2. 보고 API 노출

OpenShift Container Platform에서 기본 미터링 설치는 경로를 자동으로 노출하여 보고 API를 사용할 수 있습니다. 이는 다음 기능을 제공합니다.

- 자동 DNS
- 클러스터 CA 기반 자동 TLS

또한 기본 설치를 통해 인증서를 제공하는 데 OpenShift Container Platform 서비스를 사용하여 TLS로 보고 API를 보호할 수 있습니다. OpenShift Container Platform OAuth 프록시는 보고 API를 인증으로 보호하는 Reporting Operator의 사이드카 컨테이너로 배포됩니다.

4.5.2.1. OpenShift Container Platform 인증 사용

기본적으로 보고 API는 TLS 및 인증으로 보호됩니다. 이는 Reporting Operator의 컨테이너와 OpenShift Container Platform auth-proxy를 실행하는 사이드카 컨테이너를 모두 포함하는 Pod를 배포하도록 Reporting Operator를 구성하여 수행됩니다.

보고 API에 액세스하기 위해 Metering Operator가 경로를 노출합니다. 해당 경로가 설치되면 다음 명령을 실행하여 경로의 호스트 이름을 가져올 수 있습니다.

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

다음으로 사용자 이름 및 암호와 함께 서비스 계정 토큰 또는 기본 인증을 사용하여 인증을 설정합니다.

4.5.2.1.1. 서비스 계정 토큰을 사용하여 인증

이 방법을 사용하여 Reporting Operator의 서비스 계정에서 토큰을 사용하고 다음 명령에서 해당 전달자 토큰을 인증 헤더에 전달합니다.

```
$ 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]"
```

위 URL에서 **name=[Report Name]** 및 **format=[Format]** 매개변수를 바꿔야 합니다. **format** 매개변수는 json, csv 또는 tabular일 수 있습니다.

4.5.2.1.2. 사용자 이름과 암호를 사용하여 인증

미터링은 htpasswd 파일의 콘텐츠에 지정된 사용자 이름 및 암호 조합을 사용하여 기본 인증 구성을 지원합니다. 기본적으로 빈 htpasswd 데이터를 포함하는 시크릿을 생성합니다. 그러나 이 방법을 사용하기 위해 **reporting-operator.spec.authProxy.htpasswd.data** 및 **reporting-operator.spec.authProxy.htpasswd.createSecret** 키를 구성할 수 있습니다.

MeteringConfig 리소스에서 위를 지정하면 다음 명령을 실행할 수 있습니다.

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

testuser:password123을 유효한 사용자 이름 및 암호 조합으로 바꿉니다.

4.5.2.2. 수동으로 인증 구성

Reporting Operator에서 OAuth를 수동으로 구성하거나 비활성화하려면 **MeteringConfig** 리소스에서 **spec.tls.enabled: false**를 설정해야 합니다.

주의

이는 또한 Reporting Operator, Presto, Hive간의 모든 TLS 및 인증을 비활성화합니다. 이러한 리소스를 수동으로 설정해야 합니다.

다음 옵션을 구성하여 인증을 활성화할 수 있습니다. 인증을 활성화하면 Reporting Operator Pod가 Pod에서 사이드카 컨테이너로 OpenShift Container Platform auth-proxy를 실행하도록 구성됩니다. 이렇게 하면 보고 API가 직접 노출되지 않도록 포트가 조정되지만 대신 자동 프록시 사이드카 컨테이너를 통해 프록시됩니다.

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

reporting-operator.spec.authProxy.enabled 및 **reporting-operator.spec.authProxy.cookie.createSecret**을 **true**로 설정하고 **reporting-operator.spec.authProxy.cookie.seed**를 32자 임의 문자열로 설정해야 합니다.

다음 명령을 사용하여 32자 임의 문자열을 생성할 수 있습니다.

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

4.5.2.2.1. 토큰 인증

다음 옵션을 **true**로 설정하면 REST API 보고에 대해 전달자 토큰을 사용한 인증이 활성화됩니다. 전달자 토큰은 서비스 계정 또는 사용자에서 가져올 수 있습니다.

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

인증이 활성화되면 다음 역할 중 하나를 사용하여 사용자 또는 서비스 계정의 보고 API를 쿼리하는 데 사용되는 전달자 토큰에 대한 액세스 권한이 있어야 합니다.

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

Metering Operator는 역할 바인딩을 생성할 수 있으며 **spec.permissions**섹션에 주체 목록을 지정하여 이러한 권한을 부여할 수 있습니다. 예를 들어 다음 **advanced-auth.yaml** 예제 구성을 참조하십시오.

```
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
```

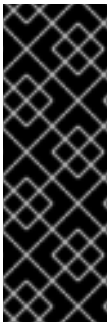
또는 **get** 권한을 **reports/export**에 부여하는 규칙이 있는 모든 역할을 사용할 수 있습니다. 즉, Reporting Operator의 네임스페이스에서 **Report** 리소스의 **export** 하위 리소스에 대한 **get** 액세스 권한을 의미합니다. 예: **admin** 및 **cluster-admin**.

기본적으로 Reporting Operator 및 Metering Operator 서비스 계정에는 모두 이러한 권한이 있으며 해당 토큰은 인증에 사용할 수 있습니다.

4.5.2.2.2. 사용자 이름과 암호를 사용하는 기본 인증

기본 인증의 경우 **reporting-operator.spec.authProxy.htpasswd.data** 필드에 사용자 이름 및 암호를 제공할 수 있습니다. 사용자 이름과 암호는 htpasswd 파일에서 찾은 항목과 동일해야 합니다. 설정된 경우 HTTP 기본 인증을 사용하여 **htpasswdData** 콘텐츠에 해당 항목이 있는 사용자 이름 및 암호를 제공할 수 있습니다.

4.6. AWS 청구 상관관계 구성



중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

미터링은 클러스터 사용 정보를 AWS 세부 청구 정보와 연결할 수 있으며 리소스 사용량에 달러 액수를 연결합니다. EC2에서 실행되는 클러스터의 경우 아래 **aws-billing.yaml** 파일 예제를 수정하여 이를 활성화할 수 있습니다.

```
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 청구 상관관계를 활성화하려면 먼저 AWS Cost 및 Usage Reports가 활성화되어 있는지 확인하십시오. 자세한 내용은 AWS 문서의 AWS Cost 및 Usage Report 를 참조하십시오.

1 버킷, 접두사 및 리전을 AWS 상세 청구 보고서의 위치로 업데이트합니다.

2 3 4 모든 **secretName** 필드는 **data.aws-access-key-id** 및 **data.aws-secret-access-key** 필드에 AWS 인증 정보를 포함하는 미터링 네임스페이스의 시크릿 이름으로 설정해야 합니다. 자세한 내용은 아래 시크릿 파일 예제를 참조하십시오.

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

S3에 데이터를 저장하려면 **aws-access-key-id** 및 **aws-secret-access-key** 인증 정보에 버킷에 대한 읽기 및 쓰기 권한이 있어야 합니다. 필요한 권한을 부여하는 IAM 정책의 예는 아래 **aws/read-write.json** 파일을 참조하십시오.

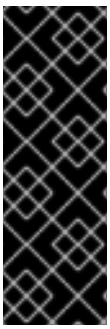
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:DeleteObject",
                "s3:GetObject",
                "s3:HeadBucket",
                "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**를 버킷 이름으로 바꿉니다.

이 작업은 사전 설치 또는 설치 후 수행될 수 있습니다. 설치 후 비활성화하면 Reporting Operator에 오류가 발생할 수 있습니다.

5장. REPORTS

5.1. 보고서 정보



중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

Report 사용자 정의 리소스는 SQL 쿼리를 사용하여 주기적인 ETL(Extract Transform and Load) 작업을 관리하는 방법을 제공합니다. Report는 실행할 실제 SQL 쿼리를 제공하는 **ReportQuery** 리소스와 **ReportQuery** 및 **Report** 리소스에 사용할 수 있는 데이터를 정의하는 **ReportDataSource** 리소스 등의 기타 미터링 리소스로 구성됩니다.

많은 사용 사례는 미터링과 함께 설치된 사전 정의된 **ReportQuery** 및 **ReportDataSource** 리소스에서 처리합니다. 따라서 이러한 사전 정의된 리소스에서 다루지 않는 사용 사례가 없는 한 자체를 정의할 필요가 없습니다.

5.1.1. Reports

Report 사용자 정의 리소스는 보고서 실행 및 상태를 관리하는 데 사용됩니다. 미터링은 사용 데이터 소스에서 파생되며 추가 분석 및 필터링에 사용될 수 있습니다. 단일 **Report** 리소스는 데이터베이스 테이블을 관리하고 스케줄에 따라 새 정보로 업데이트하는 작업을 나타냅니다. Report는 Reporting Operator HTTP API를 통해 해당 테이블에 데이터를 노출합니다.

spec.schedule 필드 집합을 사용하는 Report는 항상 실행 중이고 데이터 수집 기간을 추적합니다. 이를 통해 미터링이 종료되거나 연장된 기간에 사용할 수 없는 경우 데이터가 해제된 위치에서 백필링됩니다. 일정이 설정되지 않으면 **reportingStart** 및 **reportingEnd**에서 지정한 시간 동안 보고서가 한 번 실행됩니다. 기본적으로 보고서는 **ReportDataSource** 리소스가 보고 기간에 적용되는 모든 데이터를 완전히 가져올 때까지 기다립니다. 보고서에 일정이 있는 경우 현재 처리 중인 기간에 데이터의 가져오기가 완료될 때까지 실행을 기다립니다.

5.1.1.1. 일정이 있는 보고서의 예

다음 예제 **Report** 오브젝트에는 모든 Pod의 CPU 요청에 대한 정보가 포함되어 있으며 매시간 실행되고 데이터 가치가 있는 마지막 시간을 추가합니다.

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  reportingStart: "2021-07-01T00:00:00Z"
  schedule:
    period: "hourly"
```

```
    hourly:
      minute: 0
      second: 0
```

5.1.1.2. 일정이 없는 보고서의 예(한 번 실행)

다음 예제 **Report** 오브젝트에서는 7월 전체에 대한 모든 Pod의 CPU 요청에 대한 정보가 포함되어 있습니다. 완료되면 다시 실행되지 않습니다.

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  reportingStart: "2021-07-01T00:00:00Z"
  reportingEnd: "2021-07-31T00:00:00Z"
```

5.1.1.3. query

query 필드에는 보고서를 생성하는 데 사용되는 **ReportQuery** 리소스의 이름을 지정합니다. 보고서 쿼리는 보고서의 스키마와 결과 처리 방법을 제어합니다.

query는 필수 필드입니다.

다음 명령을 사용하여 사용 가능한 **ReportQuery** 리소스를 나열합니다.

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

출력 예

```
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 접미사가 있는 Report 쿼리는 다른 **ReportQuery** 리소스에서 더 복잡한 쿼리를 빌드하기 위해 사용되며 보고서에는 직접 사용해서는 안 됩니다.

namespace- 접두사가 지정된 쿼리에서는 네임스페이스별 Pod CPU 및 메모리 요청을 집계하여 네임스페이스 목록과 리소스 요청에 따른 전체 사용량을 제공합니다.

pod- 접두사가 지정된 쿼리는 **namespace-** 접두사가 지정된 쿼리와 유사하지만 네임스페이스가 아닌 Pod별 정보를 집계합니다. 이러한 쿼리에는 Pod의 네임스페이스와 노드가 포함됩니다.

node- 접두사가 지정된 쿼리는 각 노드의 사용 가능한 전체 리소스에 대한 정보를 반환합니다.

aws- 접두사가 지정된 쿼리는 AWS에 고유합니다. **-aws** 접미사가 지정된 쿼리는 접미사가 없는 동일한 이름의 쿼리와 동일한 데이터를 반환하며 EC2 청구 데이터 사용과 상관관계가 있습니다.

aws-ec2-billing-data 보고서는 다른 쿼리에서 사용하며 독립형 보고서로 사용해서는 안 됩니다. **aws-ec2-cluster-cost** 보고서는 클러스터에 포함된 노드를 기반으로 한 총비용과 보고되는 기간에 비용의 총합을 제공합니다.

다음 명령을 사용하여 **ReportQuery** 리소스를 YAML로 가져오고 **spec.columns** 필드를 확인합니다. 예를 들어 다음을 실행합니다.

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

출력 예

```
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. 일정

spec.schedule 설정 블록은 보고서 실행 시기를 정의합니다. **schedule** 섹션의 주요 필드는 **period**이며 **period**의 값에 따라 **hourly**, **daily**, **weekly**, **monthly** 필드를 통해 보고서 실행 시기를 미세 조정할 수 있습니다.

예를 들어 **period**이 **weekly**으로 설정되어 있으면 **spec.schedule** 블록에 **weekly** 필드를 추가할 수 있습니다. 다음 예는 수요일 오후 1시(하루 중 13시)에 일주일에 한 번 실행됩니다.

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

5.1.1.4.1. 기간

schedule.period의 유효한 값은 아래에 나열되며, 지정된 기간에 설정할 수 있는 옵션도 나열됩니다.

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

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

일반적으로 **hour**, **minute**, **second** 필드는 보고서를 실행한 날을 제어하며 주간 또는 월간 보고 기간인 경우 **dayOfWeek**/**dayOfMonth**는 주중 요일 또는 보고서가 실행되는 월의 일을 제어합니다.

이러한 필드마다 유효한 값의 범위가 있습니다.

- **hour**은 0에서 23 사이의 정수 값입니다.
- **minute**은 0에서 59 사이의 정수 값입니다.
- **second**는 0에서 59 사이의 정수 값입니다.
- **dayOfWeek**는 주중 요일(상세 설명)을 예상하는 문자열 값입니다.
- **dayOfMonth**는 1에서 31 사이의 정수 값입니다.

cron 기간의 경우 정상적인 cron 표현식은 다음과 같습니다.

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

5.1.1.5. reportingStart

기존 데이터에 대한 보고서 실행을 지원하기 위해 **spec.reportingstartt** 필드를 RFC3339 타임 스탬프로 설정하여 현재 시간 대신 **reportingStart**에서 시작되는 **schedule**에 따라 Report가 실행되도록 지시할 수 있습니다.

참고

spec.reportingStart 필드를 특정 시간으로 설정하면 Reporting Operator가 **reportingStart** 시간과 현재 시간 사이에 있는 일정의 각 간격에 대해 많은 쿼리가 연속으로 실행됩니다. 이는 기간이 일별보다 적고 보고 **reportingStart**가 몇 개월 전보다 많은 경우 수천 개의 쿼리가 될 수 있습니다. **reportingStart**가 설정되지 않은 경우 보고서가 생성된 후 Report가 다음 전체 **reportingPeriod**에서 실행됩니다.

이 필드를 사용하는 방법의 예로서, **Report** 오브젝트에 포함하려는 2019년 1월 1일로 돌아가 이미 수집한 데이터가 있는 경우 다음 값을 사용하여 보고서를 생성할 수 있습니다.

```
apiVersion: metering.openshift.io/v1
kind: Report
```

```
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  schedule:
    period: "hourly"
  reportingStart: "2021-01-01T00:00:00Z"
```

5.1.1.6. reportingEnd

지정된 시간까지만 실행되도록 보고서를 구성하려면 **spec.reportingEnd** 필드를 RFC3339 타임 스탬프로 설정할 수 있습니다. 이 필드의 값으로 인해 **reportingEnd**까지 시작 시간부터 적용된 기간에 보고 데이터 생성을 완료한 후 해당 일정에서 보고서가 실행을 중지합니다.

일정이 **reportingEnd**와 정렬되지 않을 수 있으므로 일정의 마지막 기간은 지정된 **reportingEnd** 시간에 종료되도록 단축됩니다. 설정되어 있지 않은 경우 보고서는 영구적으로 실행되거나 **reportingEnd**가 보고서에 설정될 때까지 계속 실행됩니다.

예를 들어 7월 한 달 동안 1주일에 한 번 실행하는 보고서를 생성하려는 경우입니다.

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: pod-cpu-request-hourly
spec:
  query: "pod-cpu-request"
  schedule:
    period: "weekly"
  reportingStart: "2021-07-01T00:00:00Z"
  reportingEnd: "2021-07-31T00:00:00Z"
```

5.1.1.7. 만료

예약된 미터링 보고서에 보관 기간을 설정하려면 **expiration** 필드를 추가합니다. **expiration** 기간 값을 설정하여 보고서를 수동으로 제거하는 것을 방지할 수 있습니다. 보존 기간은 **Report** 오브젝트 **creationDate** 및 **expiration** 기간과 동일합니다. 다른 보고서 또는 보고서 쿼리가 만료 보고서에 의존하지 않는 경우 보존 기간의 마지막에 클러스터에서 보고서가 제거됩니다. 클러스터에서 보고서를 삭제하는 데 몇 분이 걸릴 수 있습니다.

참고

롤업 또는 집계 보고서에는 **expiration** 필드를 설정하는 것이 권장되지 않습니다. 다른 보고서 또는 보고서 쿼리에 따라 보고서가 사용되는 경우 보존 기간이 끝나면 해당 보고서가 제거되지 않습니다. 보고서 보존 결정에 대한 타이밍 출력의 디버그 수준에서 **report-operator** 로그를 볼 수 있습니다.

예를 들어 다음 예약된 보고서는 보고서의 **creationDate** 30분 후 삭제됩니다.

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

```
schedule:
  period: "weekly"
reportingStart: "2021-07-01T00:00:00Z"
expiration: "30m" 1
```

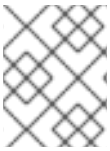
1 **expiration** 기간에 유효한 시간 단위는 **ns**, **us**(또는 **µs**), **ms**, **s**, **m** 및 **h**입니다.

참고

Report 오브젝트의 **expiration** 보존 기간은 정확하지 않으며 나노초가 아닌 수 분 단위로 작동합니다.

5.1.1.8. runImmediately

runImmediately가 **true**로 설정되면 보고서가 즉시 실행됩니다. 이 동작은 추가 일정 매개변수를 사용할 필요 없이 보고서가 즉시 처리되어 큐에 저장되도록 합니다.



참고

runImmediately가 **true**로 설정되면 **reportingEnd** 및 **reportingStart** 값을 설정해야 합니다.

5.1.1.9. 입력

Report 오브젝트의 **spec.inputs** 필드를 사용하여 **ReportQuery** 리소스의 **spec.inputs** 필드에 정의된 값을 덮어쓰거나 설정할 수 있습니다.

spec.inputs는 이름-값 쌍의 목록입니다.

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

1 입력 **name**은 ReportQuery의 **inputs** 목록에 있어야 합니다.

2 입력 **value**는 입력 **type**에 대한 올바른 유형이어야 합니다.

5.1.1.10. 롤업 보고서

보고서 데이터는 메트릭 자체와 마찬가지로 데이터베이스에 저장되므로 집계 또는 롤업 보고서에 사용할 수 있습니다. 롤업 보고서의 간단한 사용 사례는 장기간에 걸쳐 보고서를 생성하는 데 필요한 시간을 분산하는 것입니다. 이는 월별 보고서를 사용하여 한 달에 걸쳐 모든 데이터를 쿼리하고 추가하는 대신 사용됩니다. 예를 들어, 작업은 각각 데이터의 1/30에 걸쳐 실행되는 일일 보고서로 분할될 수 있습니다.

사용자 정의 롤업 보고서에는 사용자 정의 보고서 쿼리가 필요합니다. **ReportQuery** 리소스 템플릿 프로세서는 **Report** 오브젝트의 **metadata.name**에서 필요한 테이블 이름을 가져올 수 있는 **reportTableName** 함수를 제공합니다.

다음은 내장된 쿼리에서 가져온 스니펫입니다.

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 롤업 보고서 예

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

5.1.1.10.1. 보고서 상태

예약된 보고서의 실행은 상태 필드를 사용하여 추적할 수 있습니다. 보고서를 준비하는 동안 발생한 오류는 여기에 기록됩니다.

현재 **Report** 오브젝트의 **status** 필드에는 두 개의 필드가 있습니다.

- **조건**: conditions는 조건 목록으로, 각각 **type**,**status**,**reason** 및 **message** 필드가 있습니다. 조건 **type** 필드의 가능한 값은 **Running** 및 **Failure**이며 예약된 보고서의 현재 상태를 나타냅니다. **reason**은 해당 **condition**이 **true**, **false** 또는 **unknown**인 상태와 함께 현재 **status**에 있는 이유를 나타냅니다. **message**는 해당 조건이 현재 상태에 있는 이유를 나타내는, 사람이 읽을 수 있는 정보를 제공합니다. **reason** 값에 대한 자세한 정보는 **pkg/apis/metering/v1/util/report_util.go**를 참조하십시오.

- **lastReportTime**: 미터링이 최대 데이터를 수집한 시간을 나타냅니다.

5.2. 스토리지 위치

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

StorageLocation 사용자 지정 리소스는 Reporting Operator에서 데이터를 저장할 위치를 구성하는 사용자 정의 리소스입니다. 여기에는 Prometheus에서 수집한 데이터와 **Report** 사용자 정의 리소스를 생성하여 생성된 결과가 포함됩니다.

여러 S3 버킷 또는 S3 및 HDFS 둘 다와 같은 여러 위치에 데이터를 저장하거나 미터링할 때 생성되지 않은 Hive 및 Presto의 데이터베이스에 액세스하려는 경우에만 **StorageLocation** 사용자 지정 리소스를 구성해야 합니다. 대부분의 사용자에게는 필수 항목이 아니며 미터링 설정에 대한 문서 는 필요한 모든 스토리지 구성 요소를 구성하는 것입니다.

5.2.1. 스토리지 위치 예제

다음 예제에서는 기본 제공 로컬 스토리지 옵션을 보여주며 Hive를 사용하도록 구성되어 있습니다. 기본적으로 데이터는 HDFS, S3 또는 **ReadWriteMany** PVC(Persistent Volume Claim)와 같이 스토리지를 사용하도록 Hive가 구성된 모든 위치에 저장됩니다.

로컬 스토리지 예제

```
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** 섹션이 있는 경우 Hive 서버를 사용하는 테이블을 생성하여 Presto에 데이터를 저장하도록 **StorageLocation** 리소스가 구성됩니다. **databaseName** 및 **unmanagedDatabase**만 필수 필드입니다.
2. hive 내 데이터베이스의 이름입니다.
3. **true**인 경우 **StorageLocation** 리소스는 적극적으로 관리되지 않으며 **databaseName**은 이미 Hive에 있을 것으로 예상됩니다. **false**인 경우, Reporting Operator를 통해 Hive에서 데이터베이스가 생성됩니다.

다음 예에서는 스토리지에 AWS S3 버킷을 사용합니다. 사용할 경로를 구성할 때 접두사가 버킷 이름에 추가됩니다.

원격 스토리지 예

```
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) 선택 사항: 데이터베이스에 사용할 Presto 및 Hive의 파일 시스템 URL입니다. 이는 **hdfs://** 또는 **s3a://** 파일 시스템 URL이 될 수 있습니다.

hive 섹션에 지정할 수 있는 몇 가지 추가 선택적 필드가 있습니다.

- **defaultTableProperties**: Hive를 사용하여 테이블을 생성하기 위한 구성 옵션이 포함되어 있습니다.
- **fileFormat**: 파일 시스템에 파일을 저장하는 데 사용되는 파일 형식입니다. 옵션 목록 및 자세한 내용은 파일 스토리지 형식에 대한 Hive 문서 를 참조하십시오.
- **rowFormat**: Hive 행 형식을 제어합니다. 이는 Hive가 행을 직렬화 및 역직렬화하는 방법을 제어합니다. 자세한 내용은 행 형식 및 SerDe에 대한 Hive 문서 를 참조하십시오.

5.2.2. 기본 스토리지 위치

주석 **storagelocation.metering.openshift.io/is-default**가 있으며 **StorageLocation** 리소스에서 **true**로 설정된 경우 해당 리소스는 기본 스토리지 리소스가 됩니다. 스토리지 위치가 지정되지 않은 스토리지 설정 옵션이 있는 모든 구성 요소에서는 기본 스토리지 리소스를 사용합니다. 하나의 기본 스토리지 리소스만 있을 수 있습니다. 주석이 있는 리소스가 여러 개 있으면 Reporting Operato가 기본값을 확인할 수 없기 때문에 오류가 기록됩니다.

기본 스토리지 예

```
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"
```

6장. 미터링 사용

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

6.1. 사전 요구 사항

- 미터링 설치
- 보고서 및 기능에 대해 구성할 수 있는 사용 가능한 옵션에 대한 세부 사항을 검토합니다.

6.2. 보고서 작성

보고서 작성은 미터링을 사용하여 데이터를 처리하고 분석하는 방법입니다.

보고서를 작성하려면 YAML 파일에 **Report** 리소스를 정의하고 필요한 매개변수를 지정하고 **openshift-metering** 네임스페이스에 생성해야 합니다.

사전 요구 사항

- 미터링이 설치되어 있어야 합니다.

절차

1. **openshift-metering** 프로젝트로 변경합니다.

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

2. YAML 파일로 **Report** 리소스를 생성합니다.
   a. 다음 콘텐츠를 사용하여 YAML 파일을 생성합니다.

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: namespace-cpu-request-2020 (1)
  namespace: openshift-metering
spec:
  reportingStart: '2020-01-01T00:00:00Z'
  reportingEnd: '2020-12-30T23:59:59Z'
  query: namespace-cpu-request (2)
  runImmediately: true (3)
```

(2) **query**는 보고서를 생성하는 데 사용되는 **ReportQuery** 리소스를 지정합니다. 보고하려는 내용에 따라 변경합니다. 옵션 목록은 **oc get reportqueries | grep -v raw**를 실행합니다.

1 보고서가 **metadata.name**에 수행하는 사항에 대해 설명적 이름을 사용합니다. 좋은 이름은 쿼리와 사용한 일정 또는 기간을 나타냅니다.

3 사용 가능한 데이터로 실행하도록 **runImmediately**를 **true**로 설정하거나 **reportingEnd**가 전달될 때까지 기다리려는 경우 **false**로 설정합니다.

b. 다음 명령을 실행하여 **Report** 리소스를 생성합니다.

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

출력 예

```
report.metering.openshift.io/namespace-cpu-request-2020 created
```

3. 다음 명령을 사용하여 보고서 및 **Running** 상태를 나열할 수 있습니다.

```
$ oc get reports
```

출력 예

```
NAME                         QUERY                SCHEDULE   RUNNING    FAILED   LAST
REPORT TIME       AGE
namespace-cpu-request-2020   namespace-cpu-request            Finished            2020-12-
30T23:59:59Z   26s
```

6.3. 보고서 결과 보기

보고서의 결과 보기에는 보고 API 경로를 쿼리하고 OpenShift Container Platform 인증 정보를 사용하여 API에 대한 인증이 포함됩니다. Report는 **JSON**, **CSV** 또는 **Tabular** 형식으로 검색할 수 있습니다.

사전 요구 사항

- 미터링이 설치되어 있어야 합니다.
- 보고 결과에 액세스하려면 클러스터 관리자여야 하거나 **openshift-metering** 네임스페이스에서 **report-exporter** 역할을 사용하여 액세스 권한이 부여되어야 합니다.

절차

1. **openshift-metering** 프로젝트로 변경합니다.

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

2. 결과에 대해 보고 API를 쿼리합니다.

a. 미터링 **reporting-api** 경로에 대한 변수를 생성한 후 경로를 가져옵니다.

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

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

b. 요청에 사용할 현재 사용자의 토큰을 가져옵니다.

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

c. 생성한 보고서의 이름으로 **reportName**을 설정합니다.

```
$ reportName=namespace-cpu-request-2020
```

d. API 응답이 출력 형식을 지정하려면 **reportFormat**을 **csv**, **json** 또는 **tabular** 중 하나로 설정합니다.

```
$ reportFormat=csv
```

e. 결과를 얻으려면 **curl**을 사용하여 보고서에 대한 보고 API에 요청합니다.

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

reportName=namespace-cpu-request-2020 및 reportFormat=csv의 출력 예

```
period_start,period_end,namespace,pod_request_cpu_core_seconds
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-
apiserver,11745.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-apiserver-
operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-
authentication,522.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-
authentication-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-cloud-
credential-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-cluster-
machine-approver,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-cluster-
node-tuning-operator,3385.800000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-cluster-
samples-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-cluster-
version,522.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-
console,522.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-console-
operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-controller-
manager,7830.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-controller-
manager-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-
dns,34372.800000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-dns-
operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-
etcd,23490.000000
```

```
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-image-registry,5993.400000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-ingress,5220.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-ingress-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-kube-apiserver,12528.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-kube-apiserver-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-kube-controller-manager,8613.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-kube-controller-manager-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-machine-api,1305.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-machine-config-operator,9637.800000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-metering,19575.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-monitoring,6256.800000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-network-operator,261.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-sdn,94503.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-service-ca,783.000000
2020-01-01 00:00:00 +0000 UTC,2020-12-30 23:59:59 +0000 UTC,openshift-service-ca-operator,261.000000
```

7장. 미터링 사용 예

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

다음 예제 보고서를 사용하여 클러스터에서 용량, 사용량 및 사용률 측정을 시작합니다. 이 예에서는 사전 정의된 쿼리 선택과 함께 다양한 유형의 보고서 미터링이 제공하는 방법을 보여줍니다.

7.1. 사전 요구 사항

- 미터링 설치
- 보고서 작성 및 보기 에 대한 세부 사항을 검토하십시오.

7.2. 시간별 및 일별 클러스터 용량 측정

다음 보고서는 시간별 및 일별 클러스터 용량을 모두 측정하는 방법을 보여줍니다. 일별 보고서는 시간별 보고의 결과를 집계하여 작동합니다.

다음 보고서에서는 매시간 클러스터 CPU 용량을 측정합니다.

클러스터당 시간별 CPU 용량 예

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

1 이 기간을 **daily**로 변경하여 일별 보고서를 받을 수 있지만 더 큰 데이터 집합을 사용하면 시간별 보고서를 사용한 다음 시간별 데이터를 일별 보고서로 집계하는 것이 더 효율적입니다.

다음 보고서는 시간별 데이터를 일별 보고서로 집계합니다.

클러스터당 일별 CPU 용량 예

```
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 구성을 유지하려면 다른 값을 변경할 때 보고서의 **name**을 변경해야 합니다.

2 **cluster-memory-capacity**도 측정할 수 있습니다. 연결된 시간별 보고서에서도 쿼리를 업데이트해야 합니다.

3 **inputs** 섹션에서는 이 보고서가 시간별로 집계하도록 구성합니다. 특히 **값: cluster-cpu-capacity-hour**은 집계되는 시간별 보고서의 이름입니다.

7.3. 일회성 보고서로 클러스터 사용량 측정

다음 보고서는 클러스터 사용량을 특정 시작 날짜로부터 측정합니다. 보고서는 저장하고 적용한 후에 한 번만 실행됩니다.

클러스터당 CPU 사용량 예

```
apiVersion: metering.openshift.io/v1
kind: Report
metadata:
  name: cluster-cpu-usage-2020 1
spec:
  reportingStart: '2020-01-01T00:00:00Z' 2
  reportingEnd: '2020-12-30T23:59:59Z'
  query: cluster-cpu-usage 3
  runImmediately: true 4
```

1 구성을 유지하려면 다른 값을 변경할 때 보고서의 **name**을 변경해야 합니다.

2 **reportingStart** 타임 스탬프부터 **reportingEnd** 타임 스탬프까지 데이터 사용을 시작하도록 보고서를 구성합니다.

3 여기에서 쿼리를 조정합니다. **cluster-memory-usage** 쿼리를 사용하여 클러스터 사용량을 측정할 수도 있습니다.

4 보고서를 저장하고 적용한 후 즉시 실행되도록 구성합니다.

7.4. CRON 표현식을 사용하여 클러스터 사용률 측정

보고서의 기간을 구성할 때 cron 표현식을 사용할 수도 있습니다. 다음 보고서에서는 평일 9am-5pm의 CPU 사용률을 확인하여 클러스터 사용률을 측정합니다.

클러스터당 평일 CPU 사용률 예

```
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 구성을 유지하려면 다른 값을 변경할 때 보고서의 **name**을 변경해야 합니다.

2 여기에서 쿼리를 조정합니다. **cluster-memory-utilization** 쿼리를 사용하여 클러스터 사용률을 측정할 수도 있습니다.

3 ㅠcron 기간에 정상적인 cron 표현식은 유효합니다.

8장. 문제 해결 및 디버깅 미터링

중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

다음 섹션을 사용하면 미터링과 관련된 특정 문제를 해결하고 디버그하는 데 도움이 됩니다.

이 섹션의 정보 외에도 다음 주제를 검토하십시오.

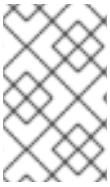
- 미터링을 설치하기 위한 사전 요구 사항 입니다.
- 미터링 구성 정보

8.1. 미터링 문제 해결

미터링과 관련된 일반적인 문제는 Pod가 시작되지 않는 것입니다. 리소스가 부족하거나 **StorageClass** 또는 **Secret** 리소스와 같이 존재하지 않는 리소스에 대한 종속성이 있는 경우 Pod를 시작하지 못할 수 있습니다.

8.1.1. 컴퓨팅 리소스가 충분하지 않음

미터링을 설치하거나 실행하는 경우 일반적인 문제는 컴퓨팅 리소스의 부족입니다. 클러스터가 증가하고 더 많은 보고서가 생성되면 Reporting Operator Pod에 더 많은 메모리가 필요합니다. 메모리 사용량이 Pod 제한에 도달하면 클러스터에서 OOM(메모리 부족)을 고려하고 **OOMKilled** 상태로 종료합니다. 미터링에 설치 사전 요구 사항에 설명된 최소 리소스 요구 사항이 할당되어야 합니다.



참고

Metering Operator는 클러스터의 부하를 기반으로 Reporting Operator를 자동 확장하지 않습니다. 따라서 클러스터가 확장되면 Reporting Operator Pod의 CPU 사용량이 늘어나지 않습니다.

리소스 또는 일정 관련 문제가 있는지 확인하려면 Kubernetes 문서 Containers용 컴퓨팅 리소스 관리 에 포함된 문제 해결 지침을 따르십시오.

컴퓨팅 리소스가 부족하여 문제를 해결하려면 **openshift-metering** 네임스페이스에서 다음을 확인합니다.

사전 요구 사항

- **openshift-metering** 네임스페이스에 있어야 합니다. 다음을 실행하여 **openshift-metering** 네임스페이스로 변경합니다.

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

절차

1. 미터링 **Report** 리소스가 완료되지 않고 **ReportingPeriodUnmetDependencies**의 상태를 표시하는지 확인합니다.

```
$ oc get reports
```

출력 예

```
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**이 보고서 종료일보다 작은 **ReportDataSource** 리소스를 확인합니다.

```
$ oc get reportdatasource
```

출력 예

```
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. 많은 Pod 재시작이 필요한 경우 **reporting-operator Pod** 리소스의 상태를 확인합니다.

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

출력 예

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

(1) Reporting Operator Pod가 높은 속도로 다시 시작됩니다.

4. **OOMKilled** 종료의 **reporting-operator Pod** 리소스를 확인합니다.

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

출력 예

```
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) OOM 종료로 인해 Reporting Operator Pod가 종료되었습니다.

reporting-operator Pod 메모리 제한 증가

Pod 재시작이 증가하고 OOM 종료 이벤트가 발생하면 Reporting Operator Pod에 설정된 현재 메모리 제한을 확인할 수 있습니다. 메모리 제한을 늘리면 Reporting Operator Pod에서 보고서 데이터 소스를 업데이트할 수 있습니다. 필요한 경우 **MeteringConfig** 리소스의 메모리 제한을 25% - 50%까지 늘립니다.

절차

1. **reporting-operator Pod** 리소스의 현재 메모리 제한을 확인합니다.

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

출력 예

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

(1) Reporting Operator Pod의 현재 메모리 제한입니다.

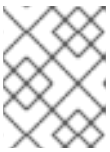
2. **MeteringConfig** 리소스를 편집하여 메모리 제한을 업데이트합니다.

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

MeteringConfig 리소스의 예입니다.

```
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** 리소스의 **resources** 필드 내에 메모리 제한을 추가하거나 늘립니다.



참고

메모리 제한이 증가한 후에도 수많은 OOM 종료 이벤트가 계속 있는 경우 다른 문제로 인해 보고서가 보류 중임을 나타낼 수 있습니다.

8.1.2. StorageClass 리소스가 구성되지 않음

미터링에는 동적 프로비저닝을 위해 기본 **StorageClass** 리소스를 구성해야 합니다.

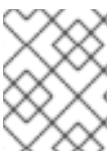
클러스터에 대한 **StorageClass** 리소스가 구성되어 있는지 확인하는 방법, 기본값을 설정하는 방법, 기본값 이외의 스토리지 클래스를 사용하도록 미터링을 구성하는 방법에 대한 자세한 내용은 미터링 설정 방법에 대한 문서를 참조하십시오.

8.1.3. secret이 올바르게 구성되지 않은 경우

미터링과 관련된 일반적인 문제는 영구 스토리지를 구성할 때 잘못된 시크릿을 제공하는 것입니다. 구성 파일 예제를 검토하고 스토리지 공급자의 지침에 따라 시크릿을 생성하십시오.

8.2. 미터링 디버깅

다양한 구성 요소와 직접 상호 작용할 때 디버깅 미터링은 훨씬 쉬워집니다. 아래 섹션에서는 Presto 및 Hive뿐만 아니라 Presto 및 HDFS 구성 요소의 대시보드를 확인할 수 있는 방법에 대해 자세히 설명합니다.



참고

이 섹션의 모든 명령은 **openshift-metering** 네임스페이스의 OperatorHub를 통해 미터링을 설치했다고 가정합니다.

8.2.1. Reporting Operator 로그 가져오기

아래 명령을 사용하여 **reporting-operator**의 로그를 따릅니다.

```
$ 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-cli를 사용하여 Presto 쿼리

다음 명령은 Presto를 쿼리할 수 있는 대화식 presto-cli 세션을 엽니다. 이 세션은 Presto와 동일한 컨테이너에서 실행되며 추가 Java 인스턴스를 시작하므로 Pod에 대한 메모리 제한을 생성할 수 있습니다. 이 경우 Presto Pod의 메모리 요청 및 제한을 늘려야 합니다.

기본적으로 Presto는 TLS를 사용하여 전달하도록 구성됩니다. 다음 명령을 사용하여 Presto 쿼리를 실행해야 합니다.

```
$ 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
```

이 명령을 실행하면 쿼리를 실행할 수 있는 프롬프트가 표시됩니다. **show tables from metering;**을 사용; 표 목록을 보려면 쿼리합니다.

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

출력 예

```
                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 20210503_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. Beeline을 사용하여 Hive 쿼리

다음에서는 Hive를 쿼리할 수 있는 대화식 Beeline 세션을 엽니다. 이 세션은 Hive와 동일한 컨테이너에서 실행되며 추가 Java 인스턴스를 시작하므로 Pod에 대한 메모리 제한을 생성할 수 있습니다. 이 경우 Hive Pod의 메모리 요청 및 제한을 늘려야 합니다.

```
$ 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'
```

이 명령을 실행하면 쿼리를 실행할 수 있는 프롬프트가 표시됩니다. **show tables;** 사용. 표 목록을 보려면 쿼리합니다.

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

출력 예

```
+----------------------------------------------------+
|                      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 웹 UI로 포트-전달

다음 명령을 실행하여 Hive 웹 UI로 포트-전달을 수행합니다.

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

이제 브라우저 창에서 http://127.0.0.1:10002를 열어 Hive 웹 인터페이스를 볼 수 있습니다.

8.2.5. HDFS로의 포트-전달

다음 명령을 실행하여 HDFS 이름 노드에 포트-전달을 수행합니다.

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

이제 브라우저 창에서 http://127.0.0.1:9870을 열어 HDFS 웹 인터페이스를 볼 수 있습니다.

다음 명령을 실행하여 첫 번째 HDFS 데이터 노드에 포트-전달을 수행합니다.

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

(1) 다른 데이터 노드를 확인하려면 정보를 확인할 Pod로 **hdfs-datanode-0**을 바꿉니다.

8.2.6. 미터링 Ansible Operator

미터링은 Ansible Operator를 사용하여 클러스터 환경에서 리소스를 감시하고 조정합니다. 실패한 미터링 설치를 디버깅하는 경우 **MeteringConfig** 사용자 정의 리소스의 Ansible 로그 또는 상태를 확인하는 데 도움이 될 수 있습니다.

8.2.6.1. Ansible 로그 액세스

기본 설치에서 Metering Operator는 Pod로 배포됩니다. 이 경우 이 Pod 내에서 Ansible 컨테이너의 로그를 확인할 수 있습니다.

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

또는 요약된 출력에 대해 Operator 컨테이너(**-c ansible**을 **-c operator**로 교체)의 로그를 볼 수 있습니다.

8.2.6.2. MeteringConfig 상태 확인

최근 오류를 디버깅하도록 **MeteringConfig** 사용자 정의 리소스의 **.status** 필드를 보는 데 유용할 수 있습니다. 다음 명령은 **Invalid** 유형이 있는 상태 메시지를 표시합니다.

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

8.2.6.3. MeteringConfig Events 확인

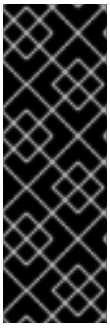
Metering Operator가 생성 중인 이벤트를 확인합니다. 이는 리소스 오류를 디버깅하기 위해 설치 중 또는 업그레이드 중에 유용할 수 있습니다. 마지막 타임 스탬프로 이벤트를 정렬합니다.

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

MeteringConfig 리소스의 최신 변경이 있는 출력 예

```
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
```

9장. 미터링 설치 제거

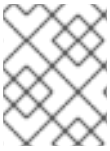


중요

미터링은 더 이상 사용되지 않는 기능입니다. 더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

OpenShift Container Platform에서 더 이상 사용되지 않거나 삭제된 주요 기능의 최신 목록은 OpenShift Container Platform 릴리스 노트에서 *더 이상 사용되지 않고 삭제된 기능* 섹션을 참조하십시오.

OpenShift Container Platform 클러스터에서 미터링을 제거할 수 있습니다.



참고

미터링 작업은 S3 버킷 데이터를 관리하거나 삭제하지 않습니다. Metering을 설치 제거한 후 미터링 데이터를 저장하는 데 사용된 S3 버킷을 수동으로 정리해야 합니다.

9.1. 클러스터에서 METERING OPERATOR 제거

클러스터에서 Operator 제거에 대한 문서에 따라 클러스터에서 Metering Operator를 제거합니다.

참고

클러스터에서 Metering Operator를 제거해도 사용자 지정 리소스 정의 또는 관리 리소스는 제거되지 않습니다. 나머지 미터링 구성 요소를 제거하는 단계는 미터링 네임스페이스 설치 제거 및 미터링 사용자 지정 리소스 정의 설치 제거 의 다음 섹션을 참조하십시오.

9.2. 미터링 네임스페이스 설치 제거

MeteringConfig 리소스를 제거하고 **openshift-metering** 네임스페이스를 삭제하여 미터링 네임스페이스(예: **openshift-metering**)를 설치 제거합니다.

사전 요구 사항

- Metering Operator가 클러스터에서 제거되었습니다.

프로세스

1. Metering Operator가 생성한 모든 리소스를 제거합니다.

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

2. 이전 단계가 완료되면 **openshift-metering** 네임스페이스의 모든 Pod가 삭제되거나 종료 상태를 보고하는지 확인합니다.

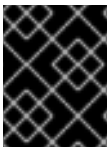
```
$ oc --namespace openshift-metering get pods
```

3. **openshift-metering** 네임스페이스를 삭제합니다.

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

9.3. 미터링 사용자 지정 리소스 정의 설치 제거

미터링 CRD(Custom Resource Definitions)는 Metering Operator가 제거되고 **openshift-metering** 네임스페이스가 삭제된 후에도 클러스터에 남아 있습니다.



중요

미터링 CRD를 삭제하면 클러스터의 다른 네임스페이스에 있는 추가 미터링 설치가 중단됩니다. 진행하기 전에 다른 미터링 설치가 없는지 확인합니다.

사전 요구 사항

- **openshift-metering** 네임스페이스에서 **MeteringConfig** 사용자 정의 리소스가 삭제됩니다.
- **openshift-metering** 네임스페이스가 삭제됩니다.

프로세스

- 나머지 미터링 CRD를 삭제합니다.

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