Red Hat build of MicroShift 4.15

ストレージ

クラスターストレージの設定と管理

Last Updated: 2024-11-02

Red Hat build of MicroShift 4.15 ストレージ

クラスターストレージの設定と管理

法律上の通知

概要

このドキュメントでは、MicroShift のストレージの使用に関する情報を提供します。

目次

第1章 ストレージの概要

MicroShift は、オンプレミスプロバイダーとクラウドプロバイダーの両方で、複数のタイプのストレージをサポートしています。MicroShift クラスター内の永続データと非永続データのコンテナーストレージを管理できます。

1.1. ストレージタイプ

MicroShift ストレージは、一時ストレージと永続ストレージという 2 つのカテゴリーに大きく分類されます。

1.1.1. 一時ストレージ

Pod およびコンテナーは性質上、一時的または遷移的であり、ステートレスアプリケーション用に設計されています。一時ストレージを使用すると、管理者および開発者は一部の操作についてローカルストレージをより適切に管理できるようになります。一時ストレージの詳細は、一時ストレージについて を参照してください。

1.1.2. 永続ストレージ

コンテナーにデプロイされるステートフルアプリケーションには永続ストレージが必要です。MicroShift は、永続ボリューム (PV) と呼ばれる事前にプロビジョニングされたストレージフレームワークを使用して、クラスター管理者が永続ストレージをプロビジョニングできるようにします。これらのボリューム内のデータは、個々の Pod のライフサイクルを超えて存在することができます。開発者は、永続ボリューム要求を使用してストレージ要件を要求できます。永続ストレージの詳細は、永続ストレージについて を参照してください。

1.1.3. 動的ストレージプロビジョニング

動的プロビジョニングを使用すると、オンデマンドでストレージボリュームを作成できるため、事前にプロビジョニングされたストレージが不要になります。MicroShift で動的プロビジョニングがどのように機能するかの詳細は、動的プロビジョニング を参照してください。

第2章 一時ストレージについて

一時ストレージは構造化されておらず、一時的なものです。不変アプリケーションでよく使用されます。このガイドでは、一時ストレージが MicroShift でどのように機能するかを説明します。

2.1. 概要

永続ストレージに加え、Pod とコンテナーは、操作に一時または短期的なローカルストレージを必要とする場合があります。この一時ストレージは、個別の Pod の寿命より長くなることはなく、一時ストレージは Pod 間で共有することはできません。

Pod は、スクラッチ領域、キャッシュ、ログに一時ローカルストレージを使用します。ローカルストレージのアカウントや分離がないことに関連する問題には、以下が含まれます。

- Pod が利用可能なローカルストレージの量を検出できない。
- Pod がローカルストレージを要求しても確実に割り当てられない可能性がある。
- ローカルストレージがベストエフォートのリソースである。
- Pod は他の Pod でローカルストレージが一杯になるとエビクトされる可能性があり、十分なストレージが回収されるまで新しい Pod は許可されない。

永続ボリュームとは異なり、一時ストレージは構造化されておらず、領域はノードで実行されているすべての Pod、システムによる他の使用、および MicroShift の間で共有されます。一時ストレージフレームワークにより、Pod は短期的なローカルストレージのニーズを指定できます。また、MicroShift がローカルストレージの過剰な使用からノードを保護できるようにします。

一時ストレージフレームワークを使用すると、管理者および開発者はローカルストレージをより適切に管理できますが、I/O スループットやレイテンシーに直接影響はありません。

2.2. 一時ストレージのタイプ

一時ローカルストレージは常に、プライマリーパーティションで利用できるようになっています。プライマリーパーティションを作成する基本的な方法には、root、ランタイムの 2 つがあります。

Root
このパーティションでは、kubelet の root ディレクトリー **/var/lib/kubelet/** (デフォルト) と **/var/log/** ディレクトリーを保持します。このパーティションは、ユーザーの Pod、OS、Kubernetes システムのデーモン間で共有できます。Pod は、**EmptyDir** ボリューム、コンテナーログ、イメージ階層、コンテナーの書き込み可能な階層を使用して、このパーティションを使用できます。Kubelet はこのパーティションの共有アクセスおよび分離を管理します。このパーティションは一時的なもので、アプリケーションは、このパーティションからディスク IOPS などのパフォーマンス SLA は期待できません。

ランタイム
これは、ランタイムがオーバーレイファイルシステムに使用可能なオプションのパーティションです。MicroShift は、このパーティションの分離および共有アクセスを特定して提供するように試行します。コンテナーイメージ階層と書き込み可能な階層は、ここに保存されます。ランタイムパーティションが存在すると、**root** パーティションにはイメージ階層もその他の書き込み可能階層も含まれません。

2.3. 一時ストレージ管理

クラスター管理者は、非終了状態のすべての Pod の一時ストレージに対して制限範囲や一時ストレージの要求数を定義するクォータを設定することで、プロジェクト内で一時ストレージを管理できます。開発者は Pod およびコンテナーのレベルで、このコンピュートリソースの要求および制限を設定する

こともできます。

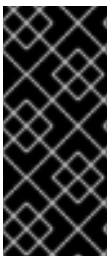
要求と制限を指定することで、ローカルの一時ストレージを管理できます。Pod 内の各コンテナーは、以下を指定できます。

- **spec.containers[].resources.limits.ephemeral-storage**
- **spec.containers[].resources.requests.ephemeral-storage**

2.3.1. 一時ストレージの制限と要求の単位

一時ストレージの制限と要求は、バイト単位で測定されます。ストレージは、E、P、T、G、M、k のいずれかの接尾辞を使用して、単純な整数または固定小数点数として表すことができます。Ei、Pi、Ti、Gi、Mi、Ki の 2 のべき乗も使用できます。

たとえば、128974848、129e6、129M、および 123Mi はすべてほぼ同じ値を表します。



重要

各バイト量の接尾辞では大文字と小文字が区別されます。必ず大文字と小文字を正しく使い分けてください。要求を 400 メガバイトに設定する場合は、"400M" のように、大文字の "M" を使用します。400 メビバイトを要求するには、大文字の "400Mi" を使用します。"400m" の一時ストレージを指定すると、ストレージが 0.4 バイトしか要求されません。

2.3.2. 一時ストレージの要求と制限の例

次のサンプル設定ファイルは、2 つのコンテナーを持つ Pod を示しています。

- 各コンテナーは、2GiB のローカル一時ストレージを要求します。
- 各コンテナーには、4GiB のローカル一時ストレージの制限があります。
- Pod レベルでは、kubelet は、その Pod 内のすべてのコンテナーの制限を合計することで、Pod 全体のストレージ制限を計算します。
  - この場合、Pod レベルでの合計ストレージ使用量は、すべてのコンテナーからのディスク使用量と Pod の **emptyDir** ボリュームの合計になります。
  - したがって、Pod には 4GiB のローカル一時ストレージの要求と、8GiB のローカル一時ストレージの制限があります。

クォータと制限を含む一時ストレージ設定の例

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        ephemeral-storage: "2Gi" (1)
```

```
      limits:
        ephemeral-storage: "4Gi" (2)
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
      emptyDir: {}
```

(1) ローカル一時ストレージに対するコンテナーの要求。

(2) ローカル一時ストレージに対するコンテナーの制限。

2.3.3. 一時ストレージ設定が Pod の削除に影響する

Pod 仕様の設定は、kubelet が Pod を削除するタイミングに影響します。最初のコンテナーによってリソース制限が設定されるため、kubelet エビクションマネージャーがこのコンテナーのディスク使用量をコンテナーレベルで測定し、コンテナーのストレージ使用量がその制限 (4 GiB) を超えた場合に Pod をエビクトします。kubelet エビクションマネージャーは、合計使用量が全体の Pod ストレージ制限 (8 GiB) を超えた場合にも、Pod にエビクションのマークを付けます。

注記

このポリシーは **emptyDir** ボリュームのみに適用され、永続ストレージには適用されません。Pod の **priorityClass** を指定して、Pod をエビクションから除外することができます。

2.4. 一時ストレージのモニタリング

/bin/df をツールとして使用し、一時コンテナーデータが置かれているボリューム (**/var/lib/kubelet** および **/var/lib/containers**) の一時ストレージの使用を監視できます。**/var/lib/kubelet** のみが使用できる領域は、クラスター管理者によって **/var/lib/containers** が別のディスクに置かれる場合に **df** コマンドを使用すると表示されます。

/var/lib での使用済みおよび利用可能な領域の人間が判読できる値を表示するには、以下のコマンドを実行します。

```
$ df -h /var/lib
```

この出力には、**/var/lib** での一時ストレージの使用状況が表示されます。

出力例

```
Filesystem  Size  Used Avail Use% Mounted on
/dev/disk/by-partuuid/4cd1448a-01    69G   32G   34G  49% /
```

第3章 汎用的な一時ボリューム

MicroShift の一時ボリュームのライフサイクル、セキュリティー、命名などを説明します。

3.1. 概要

汎用一時ボリュームは、永続ボリュームおよび動的プロビジョニングをサポートするすべてのストレージドライバーが提供できる一時ボリュームの一種です。汎用の一時ボリュームは、スクラッチデータ用に Pod ごとのディレクトリー (通常、プロビジョニング後は空) を提供する点で **emptyDir** ボリュームと類似しています。

汎用の一時ボリュームは Pod 仕様に準拠して指定され、Pod のライフサイクルに従います。これらは Pod と共に作成され、削除されます。

汎用の一時ボリュームには、以下の特徴があります。

- ストレージは、ローカルまたはネットワーク接続タイプとすることができます。
- ボリュームには、Pod が超過できない固定サイズを指定できます。
- ドライバーおよびパラメーターによっては、ボリュームに特定の初期データが含まれる場合があります。
- ドライバーがサポートしていれば、スナップショットの作成、クローンの作成、サイズ変更、ストレージ容量の追跡など、ボリュームに対する一般的な操作がサポートされます。

注記

汎用の一時ボリュームは、オフラインのスナップショットやサイズ変更をサポートしません。

3.2. ライフサイクルおよび永続ボリューム要求

ボリューム要求のパラメーターは Pod のボリュームソース内で許可されます。ラベル、アノテーション、および永続ボリューム要求 (PVC) のフィールドの全セットがサポートされます。このような Pod が作成されると、一時ボリュームコントローラーは (**汎用一時ボリュームの作成** の手順に示すテンプレートから) Pod と同じ namespace に実際の PVC オブジェクトを作成し、Pod が削除されると PVC が削除されるようにします。これがトリガーとなり、以下の 2 つの方法のいずれかでボリュームのバインディングおよびプロビジョニングが行われます。

- 直ちに (ストレージクラスが即時ボリュームバインディングを使用する場合は)
  即時バインディングの場合、スケジューラーはボリュームが利用可能になった後にボリュームにアクセスできるノードを強制的に選択させられます。
- Pod が一時的にノードにスケジュールされる場合 (**WaitForFirstConsumervolume** バインディングモード)
  スケジューラーは Pod に適したノードを選択できるため、このボリュームバインディングオプションは、汎用一時ボリュームに推奨されます。

リソースの所有権に関しては、汎用一時ストレージを持つ Pod は、その一時ストレージを提供する PVC の所有者となります。Pod が削除されると、Kubernetes ガベージコレクターによって PVC が削除され、ストレージクラスのデフォルトの再利用ポリシーがボリュームを削除することになっているため、通常はボリュームの削除がトリガーされます。回収ポリシーが保持のストレージクラスを使用して、準一時ローカルストレージを作成できます。Pod が削除されてもストレージは存続するため、この場合は別途ボリュームのクリーンアップが行われるようにする必要があります。これらの PVC は存在

しますが、それらは他の PVC と同様に使用できます。特に、それらはボリュームのクローン作成またはスナップショット作成時にデータソースとして参照できます。PVC オブジェクトは、ボリュームの現在のステータスも保持します。

3.3. セキュリティー

汎用一時ボリューム機能を有効にすると、Pod を作成できるユーザーが永続ボリューム要求 (PVC) も間接的に作成できるようになります。この機能は、これらのユーザーが PVC を直接作成するパーミッションを持たない場合でも機能します。クラスター管理者はこれを認識している必要があります。これがセキュリティーモデルに適さない場合は、汎用的な一時ボリュームを持つ Pod などのオブジェクトを拒否する容認 Webhook を使用します。

PVC に対する通常の namespace クォータがそのまま適用されるため、この新しいメカニズムを使用できる場合でも新しいメカニズムを使用して他のポリシーを回避できません。

3.4. 永続ボリューム要求の命名

自動的に作成される永続ボリューム要求 (PVC) には、Pod 名とボリューム名を組み合わせ、間にハイフン (-) を挿入した名前が付けられます。この命名規則では、異なる Pod 間および Pod と手動で作成された PVC 間で競合が生じる可能性があります。

たとえば、**pod-a** とボリューム **scratch** の組み合わせと、**pod** とボリューム **a-scratch** の組み合わせは、どちらも同じ PVC 名 **pod-a-scratch** になります。

このような競合は検出され、Pod 用に作成された場合にのみ、PVC は一時ボリュームに使用されます。このチェックは所有者の関係に基づいています。既存の PVC は上書きまたは変更されませんが、競合は解決されません。適切な PVC がないと、Pod は起動できません。

重要

同じ namespace 内で Pod とボリュームに名前を付ける際には、命名の競合が発生しないように注意してください。

3.5. 汎用一時ボリュームの作成

手順

1. **pod** オブジェクト定義を作成し、これをファイルに保存します。
2. ファイルに汎用一時ボリュームの情報を追加します。

my-example-pod-with-generic-vols.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: my-app
spec:
  containers:
    - name: my-frontend
      image: busybox:1.28
      volumeMounts:
      - mountPath: "/mnt/storage"
        name: data
```

```
    command: [ "sleep", "1000000" ]
  volumes:
    - name: data (1)
      ephemeral:
        volumeClaimTemplate:
          metadata:
            labels:
              type: my-app-ephvol
          spec:
            accessModes: [ "ReadWriteOnce" ]
            storageClassName: "topolvm-provisioner"
            resources:
              requests:
                storage: 1Gi
```

(1) 汎用一時ボリューム要求

第4章 永続ストレージについて

ストレージの管理は、コンピュートリソースの管理とは異なります。MicroShift は Kubernetes 永続ボリューム (PV) フレームワークを使用してクラスター管理者がクラスターの永続ストレージのプロビジョニングを実行できるようにします。開発者は、永続ボリューム要求 (PVC) を使用すると、基礎となるストレージインフラストラクチャーに関する特定の知識がなくても PV リソースを要求することができます。

4.1. SCC (SECURITY CONTEXT CONSTRAINTS)によるパーミッションの制御

ROSA では、Security Context Constraints (SCC) を使用して、クラスター内の Pod のアクセス許可を制御できます。これらのアクセス許可によって、Pod が実行できるアクションとアクセスできるリソースが決まります。SCC を使用して、Pod がシステムに受け入れられるために必要な Pod の実行に関する条件の一覧を定義できます。

詳細は、Security Context Constraints の管理 を参照してください。

重要

RWO ボリュームマウントのみがサポートされます。Pod が SCC コンテキストで動作しない場合、SCC はブロックされる可能性があります。

4.2. 永続ストレージの概要

PVC は namespace に固有であり、PV を使用する手段として開発者によって作成および使用されます。PV リソース自体は、単一の namespace に限定されません。これらは MicroShift クラスター全体で共有でき、任意の namespace から要求できます。PV が PVC にバインドされた後は、その PV を追加の PVC にバインドできません。これにより、バインドされた PV が単一の namespace にスコープ設定されます。

PV は、クラスター管理者によって静的にプロビジョニングされているか、**StorageClass** オブジェクトを使用して動的にプロビジョニングされているクラスター内の既存ストレージの一部を表す、**PersistentVolume** API オブジェクトで定義されます。これは、ノードがクラスターリソースであるのと同様にクラスター内のリソースです。

PV は **Volumes** などのボリュームプラグインですが、PV を使用する個々の Pod から独立したライフサイクルを持ちます。PV オブジェクトは、そのLVM、ホストパスなどのホストファイルシステム、または raw ブロックデバイスなど、ストレージの実装の詳細をキャプチャーします。

重要

インフラストラクチャーにおけるストレージの高可用性は、基礎となるストレージのプロバイダーに委ねられています。

PersistentVolumes と同様に、**PersistentVolumeClaims** (PVC) は API オブジェクトであり、開発者によるストレージの要求を表します。これは Pod がノードリソースを消費する点で Pod に似ており、PVC は PV リソースを消費します。たとえば、Pod は特定のレベルのリソース (CPU およびメモリーなど) を要求し、PVC は特定のストレージ容量およびアクセスモードを要求できます。OpenShift Container Platform でサポートされるアクセスモードは、MicroShift でも定義できます。ただし、MicroShift はマルチノードデプロイメントをサポートしていないため、ReadWriteOnce (RWO) のみが適切です。

4.3. 関連情報

- 永続ストレージのアクセスモード

4.4. ボリュームおよび要求のライフサイクル

PV はクラスターのリソースです。PVC はそれらのリソースの要求であり、リソースに対する要求チェックとして機能します。PV と PVC 間の相互作用には以下のライフサイクルが設定されます。

4.4.1. ストレージのプロビジョニング

PVC で定義される開発者からの要求に対応し、クラスター管理者はストレージおよび一致する PV をプロビジョニングする 1 つ以上の動的プロビジョナーを設定します。

4.4.2. 要求のバインド

PVC の作成時に、ストレージの特定容量の要求、必要なアクセスモードの指定のほか、ストレージクラスを作成してストレージの記述や分類を行います。マスターのコントロールループは新規 PVC の有無を監視し、新規 PVC を適切な PV にバインドします。適切な PV がない場合は、ストレージクラスのプロビジョナーが PV を作成します。

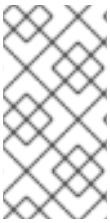
すべての PV のサイズが PVC サイズを超える可能性があります。これは、特に、手動でプロビジョニングされる PV の場合に当てはまります。超過を最小限に抑えるために、MicroShift は他のすべての基準に一致する最小の PV にバインドします。

要求は、一致するボリュームが存在しないか、ストレージクラスを提供するいずれの利用可能なプロビジョナーで作成されない場合は、無期限でバインドされないままになります。要求は、一致するボリュームが利用可能になるとバインドされます。たとえば、多数の手動でプロビジョニングされた 50Gi ボリュームを持つクラスターは 100Gi を要求する PVC に一致しません。PVC は 100Gi PV がクラスターに追加されるとバインドされます。

4.4.3. Pod および要求した PV の使用

Pod は要求をボリュームとして使用します。クラスターは要求を検査して、バインドされたボリュームを検索し、Pod にそのボリュームをマウントします。複数のアクセスモードをサポートするボリュームの場合は、要求を Pod のボリュームとして使用する際に適用するモードを指定する必要があります。

要求が存在し、その要求がバインドされている場合は、バインドされた PV を必要な期間保持できます。Pod のスケジュールおよび要求された PV のアクセスは、**persistentVolumeClaim** を Pod のボリュームブロックに組み込んで実行できます。



注記

ファイル数が多い永続ボリュームを Pod に割り当てる場合、それらの Pod は失敗するか、起動に時間がかかる場合があります。詳細は、When using Persistent Volumes with high file counts in OpenShift, why do pods fail to start or take an excessive amount of time to achieve "Ready" state? を参照してください。

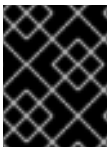
4.4.4. 永続ボリュームの解放

ボリュームの処理が終了したら、API から PVC オブジェクトを削除できます。これにより、リソースを回収できるようになります。ボリュームは要求の削除時に解放 (リリース) されたものとみなされますが、別の要求で利用できる状態にはなりません。以前の要求側に関連するデータはボリューム上に残るため、ポリシーに基づいて処理される必要があります。

4.4.5. 永続ボリュームの回収ポリシー

永続ボリュームの回収ポリシーは、クラスターに対してリリース後のボリュームの処理方法を指示します。ボリュームの回収ポリシーは、**Retain**、**Recycle** または **Delete** のいずれかにすることができます。

- **Retain** 回収ポリシーは、サポートするボリュームプラグインのリソースの手動による回収を許可します。
- **Recycle** 回収ポリシーは、ボリュームがその要求からリリースされると、バインドされていない永続ボリュームのプールにボリュームをリサイクルします。



重要

Recycle 再利用ポリシーは、MicroShift 4 で非推奨になりました。動的プロビジョニングは、同等またはそれ以上の機能で推奨されます。

- **Delete** 回収ポリシーは、Red Hat build of MicroShift の **PersistentVolume** オブジェクトと、Amazon Elastic Block Store (Amazon EBS) または VMware vSphere などの外部インフラストラクチャーの関連するストレージアセットの両方を削除します。

注記

動的にプロビジョニングされたボリュームは常に削除されます。

4.4.6. 永続ボリュームの手動回収

永続ボリューム要求 (PVC) が削除されると、基礎となる論理ボリュームは **reclaimPolicy** に従って処理されます。

手順

クラスター管理者として PV を手動で回収するには、以下を実行します。

1. PV を削除します。

```
$ oc delete pv <pv-name>
```

   AWS EBS、GCE PD、Azure Disk、Cinder ボリュームなどの外部インフラストラクチャーの関連するストレージアセットは、PV の削除後も引き続き存在します。

2. 関連するストレージアセットのデータをクリーンアップします。
3. 関連するストレージアセットを削除します。または、同じストレージアセットを再利用するには、ストレージアセットの定義で新規 PV を作成します。

回収される PV が別の PVC で使用できるようになります。

4.4.7. 永続ボリュームの回収ポリシーの変更

永続ボリュームの回収ポリシーを変更するには、以下を実行します。

1. クラスターの永続ボリュームをリスト表示します。

```
$ oc get pv
```

出力例

```
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM            STORAGECLASS     REASON    AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim1    manual                    10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim2    manual                    6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim3    manual                    3s
```

2. 永続ボリュームの 1 つを選択し、その回収ポリシーを変更します。

```
$ oc patch pv <your-pv-name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

3. 選択した永続ボリュームに正しいポリシーがあることを確認します。

```
$ oc get pv
```

出力例

```
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM            STORAGECLASS     REASON    AGE
 pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim1    manual                    10s
 pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Delete          Bound
default/claim2    manual                    6s
 pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94   4Gi        RWO           Retain          Bound
default/claim3    manual                    3s
```

上記の出力では、要求 **default/claim3** にバインドされたボリュームに **Retain** 回収ポリシーが含まれるようになりました。ユーザーが要求 **default/claim3** を削除しても、ボリュームは自動的に削除されません。

4.5. 永続ボリューム

各 PV には、以下の例のように、ボリュームの仕様およびステータスである **spec** および **status** が含まれます。

PersistentVolume オブジェクト定義の例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
    - ReadWriteOnce 3
  persistentVolumeReclaimPolicy: Retain 4
```

```
  ...
status:
  ...
```

1. 永続ボリュームの名前。
2. ボリュームに利用できるストレージの量。
3. 読み取り書き込みおよびマウントパーミッションを定義するアクセスモード。
4. リソースのリリース後にそれらのリソースがどのように処理されるかを示す回収ポリシー。

4.5.1. Capacity

通常、永続ボリューム (PV) には特定のストレージ容量があります。これは PV の **capacity** 属性を使用して設定されます。

現時点で、ストレージ容量は設定または要求できる唯一のリソースです。今後は属性として IOPS、スループットなどが含まれる可能性があります。

4.5.2. サポートされているアクセスモード

LVMS は、MicroShift がサポートする唯一の CSI プラグインです。OpenShift Container Platform に組み込まれている hostPath および LV も RWO をサポートします。

4.5.3. フェーズ

ボリュームは以下のフェーズのいずれかにあります。

表4.1 ボリュームのフェーズ

フェーズ	説明
Available	まだ要求にバインドされていない空きリソースです。
Bound	ボリュームが要求にバインドされています。
Released	要求が削除されていますが、リソースがまだクラスターにより回収されていません。
Failed	ボリュームが自動回収に失敗しています。

以下のコマンドを実行して、PV にバインドされている PVC の名前を表示できます。

```
$ oc get pv <pv-claim>
```

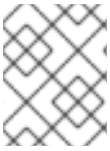
4.5.3.1. マウントオプション

属性 **mountOptions** を使用して PV のマウント中にマウントオプションを指定できます。

以下に例を示します。

マウントオプションの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: topolvm-provisioner
mountOptions:
  - uid=1500
  - gid=1500
parameters:
  csi.storage.k8s.io/fstype: xfs
provisioner: topolvm.io
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```



注記

mountOptions は検証されません。値が間違っていると、マウントが失敗し、イベントが PVC に記録されます。

4.6. RWO アクセスモードのパーミッションを持つ永続ボリューム

永続ボリューム要求(PVC)は、さまざまなアクセスモードで作成できます。**ReadWriteOnce** (RWO)アクセスモードセットを持つ PVC により、同じノード上の複数の Pod が同じ PV に一度に読み取りまたは書き込みできるようになります。

同じノードの Pod が同じ PV に読み取りまたは書き込みできない場合があります。これは、ノードの Pod に同じ SELinux コンテキストがない場合に発生します。永続ボリュームは、RWO アクセスモードで PVC によって後で要求できます。

4.7. POD の不一致を確認する

次の手順を使用して、Pod が一致しているかどうかを確認します。

重要

- &lt **;pod_name_a&gt**; を以下の手順の最初の Pod の名前に置き換えます。
- &lt **;pod_name_b>** を以下の手順の 2 つ目の Pod の名前に置き換えます。
- &lt **;pvc_mountpoint&gt**; を Pod 内のマウントポイントに置き換えます。

手順

1. 次のコマンドを実行して、最初の Pod 内のマウントポイントを一覧表示します。

```
$ oc get pods -n <pod_name_a> -
ojsonpath='{.spec.containers[].volumeMounts[].mountPath}' 1
```

1 &lt **;pod_name_a&gt**; は最初の Pod の名前に置き換えます。

出力例

```
/files /var/run/secrets/kubernetes.io/serviceaccount
```

2. 次のコマンドを実行して、2 つ目の Pod 内のマウントポイントを一覧表示します。

```
$ oc get pods -n <pod_name_b> -
ojsonpath='{.spec.containers[].volumeMounts[].mountPath}' 1
```

1 &lt ;pod_name_b> を 2 つ目の Pod の名前に置き換えます。

出力例

```
/files /var/run/secrets/kubernetes.io/serviceaccount
```

3. 次のコマンドを実行して、最初の Pod 内のコンテキストとパーミッションを確認します。

```
$ oc rsh <pod_name_a> ls -lZah <pvc_mountpoint> 1
```

1 &lt ;pod_name_a > を最初の Pod の名前に置き換え、< pvc_mountpoint > を最初の Pod 内のマウントポイントに置き換えます。

出力例

```
total 12K
dr-xr-xr-x.  1 root root system_u:object_r:container_file_t:s0:c398,c806   40 Feb 17 13:36 .
dr-xr-xr-x.  1 root root system_u:object_r:container_file_t:s0:c398,c806   40 Feb 17 13:36 ..
[...]
```

4. 次のコマンドを実行して、2 つ目の Pod 内のコンテキストとパーミッションを確認します。

```
$ oc rsh <pod_name_b> ls -lZah <pvc_mountpoint> 1
```

1 &lt ;pod_name_b > を 2 つ目の Pod の名前に置き換え、< pvc_mountpoint > を 2 つ目の Pod 内のマウントポイントに置き換えます。

出力例

```
total 12K
dr-xr-xr-x.  1 root root system_u:object_r:container_file_t:s0:c15,c25   40 Feb 17 13:34 .
dr-xr-xr-x.  1 root root system_u:object_r:container_file_t:s0:c15,c25   40 Feb 17 13:34 ..
[...]
```

5. 両方の出力を比較して、SELinux コンテキストの不一致があるかどうかを確認します。

4.8. 不一致のある POD の更新

次の手順を使用して、不一致が見つかった場合は Pod の SELinux コンテキストを更新します。

手順

1. SELinux コンテンツが一致しない場合は、新しい SCC (Security Context Constraints)を作成し、これを両方の Pod に割り当てます。SCC を作成するには、SCC (Security Context Constraints)の作成 を参照してください。

2. 以下の例のように SELinux コンテキストを更新します。

出力例

```
[...]
securityContext:privileged
    seLinuxOptions:MustRunAs
      level: "s0:cXX,cYY"
 [...]
```

4.9. 不一致の解決後の POD の検証

以下の検証手順を使用して、SCC (Security Context Constraints)と両方の SELinux ラベルを確認します。

検証

1. 次のコマンドを実行して、同じ SCC が最初の Pod に割り当てられていることを確認します。

```
$ oc describe pod <pod_name_a> |grep -i scc 1
```

1 &lt ;pod_name_a> は最初の Pod の名前に置き換えます。

出力例

```
openshift.io/scc: restricted
```

2. 次のコマンドを実行して、同じ SCC が最初の Pod に割り当てられていることを確認します。

```
$ oc describe pod <pod_name_b> |grep -i scc 1
```

1 &lt ;pod_name_b> を 2 つ目の Pod の名前に置き換えます。

出力例

```
openshift.io/scc: restricted
```

3. 次のコマンドを実行して、同じ SELinux ラベルが最初の Pod に適用されていることを確認します。

```
$ oc exec <pod_name_a> -- ls -laZ <pvc_mountpoint> 1
```

1 &lt ;pod_name_a > を最初の Pod の名前に置き換え、< pvc_mountpoint > を最初の Pod 内のマウントポイントに置き換えます。

出力例

```
total 4
drwxrwsrwx. 2 root       1000670000 system_u:object_r:container_file_t:s0:c10,c26 19 Aug 29 18:17 .
dr-xr-xr-x. 1 root       root       system_u:object_r:container_file_t:s0:c10,c26 61 Aug 29 18:16 ..
-rw-rw-rw-. 1 1000670000 1000670000 system_u:object_r:container_file_t:s0:c10,c26 29 Aug 29 18:17 test1
[...]
```

4. 次のコマンドを実行して、同じ SELinux ラベルが 2 つ目の Pod に適用されていることを確認します。

```
$ oc exec <pod_name_b> -- ls -laZ <pvc_mountpoint> (1)
```

(1) &lt ;**pod_name_b** > を 2 つ目の Pod の名前に置き換え、< **pvc_mountpoint** > を 2 つ目の Pod 内のマウントポイントに置き換えます。

出力例

```
total 4
drwxrwsrwx. 2 root       1000670000 system_u:object_r:container_file_t:s0:c10,c26 19 Aug 29 18:17 .
dr-xr-xr-x. 1 root       root       system_u:object_r:container_file_t:s0:c10,c26 61 Aug 29 18:16 ..
-rw-rw-rw-. 1 1000670000 1000670000 system_u:object_r:container_file_t:s0:c10,c26 29 Aug 29 18:17 test1
[...]
```

関連情報

- 一般的なマウントオプション

4.10. 永続ボリューム要求

各 **PersistentVolumeClaim** オブジェクトには、永続ボリューム要求 (PVC) の仕様およびステータスである **spec** および **status** が含まれます。以下が例になります。

PersistentVolumeClaim オブジェクト定義の例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim (1)
spec:
  accessModes:
    - ReadWriteOnce (2)
  resources:
    requests:
      storage: 8Gi (3)
```

```
  storageClassName: gold 4
status:
  ...
```

1 PVC の名前。

2 読み取り書き込みおよびマウントパーミッションを定義するアクセスモード。

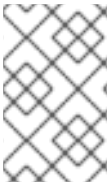
3 PVC に利用できるストレージの量。

4 要求で必要になる **StorageClass** の名前。

4.10.1. ストレージクラス

要求は、ストレージクラスの名前を **storageClassName** 属性に指定して特定のストレージクラスをオプションでリクエストできます。リクエストされたクラスの PV、つまり PVC と同じ **storageClassName** を持つ PV のみが PVC にバインドされます。クラスター管理者は 1 つ以上のストレージクラスを提供するように動的プロビジョナーを設定できます。クラスター管理者は、PVC の仕様に一致する PV をオンデマンドで作成できます。

クラスター管理者は、すべての PVC にデフォルトストレージクラスを設定することもできます。デフォルトのストレージクラスが設定されると、PVC は **""** に設定された **StorageClass** または **storageClassName** アノテーションがストレージクラスなしの PV にバインドされるように明示的に要求する必要があります。



注記

複数のストレージクラスがデフォルトとしてマークされている場合、PVC は **storageClassName** が明示的に指定されている場合にのみ作成できます。そのため、1 つのストレージクラスのみをデフォルトとして設定する必要があります。

4.10.2. アクセスモード

要求は、特定のアクセスモードのストレージを要求する際にボリュームと同じ規則を使用します。

4.10.3. リソース

要求は、Pod の場合のようにリソースの特定の数量を要求できます。今回の例では、ストレージに対する要求です。同じリソースモデルがボリュームと要求の両方に適用されます。

4.10.4. ボリュームとしての要求

Pod は要求をボリュームとして使用することでストレージにアクセスします。この要求を使用して、Pod と同じ namespace 内に要求を共存させる必要があります。クラスターは Pod の namespace で要求を見つけ、これを使用して要求をサポートする **PersistentVolume** を取得します。以下のように、ボリュームはホストにマウントされ、Pod に組み込まれます。

ホストおよび Pod のサンプルへのボリュームのマウント

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
```

```
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html" 1
        name: mypd 2
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim 3
```

1 Pod 内にボリュームをマウントするためのパス

2 マウントするボリュームの名前。コンテナーのルート (/) や、ホストとコンテナーで同じパスにはマウントしないでください。これは、コンテナーに十分な特権が付与されている場合に、ホストシステムを破壊する可能性があります (例: ホストの **/dev/pts** ファイル)。ホストをマウントするには、**/host** を使用するのが安全です。

3 使用する同じ namespace にある PVC の名前

4.11. FSGROUP を使用した POD タイムアウトの削減

ストレージボリュームに多数のファイル (~1,000,000 以上) が含まれる場合は、Pod のタイムアウトが生じる可能性があります。

デフォルトでは、MicroShift は、ボリュームがマウントされる際に Pod の **securityContext** で指定される **fsGroup** に一致するよう、各ボリュームのコンテンツの所有者とパーミッションを再帰的に変更するため、これが発生する可能性があります。大規模なボリュームでは、所有者とパーミッションの確認と変更には時間がかかり、Pod の起動が遅くなる場合があります。**securityContext** 内の **fsGroupChangePolicy** フィールドを使用して、MicroShift がボリュームの所有権と権限をチェックおよび管理する方法を制御できます。

fsGroupChangePolicy は、Pod 内で公開される前にボリュームの所有者およびパーミッションを変更する動作を定義します。このフィールドは、**fsGroup**で制御される所有者およびパーミッションをサポートするボリュームタイプにのみ適用されます。このフィールドには、以下の 2 つの値を指定できます。

- **OnRootMismatch**: ルートディレクトリーのパーミッションと所有者が、ボリュームの予想されるパーミッションと一致しない場合にのみ、パーミッションと所有者を変更します。これにより、ボリュームの所有者とパーミッションを変更するのに必要な時間を短縮でき、Pod のタイムアウトを減らすことができます。

- **Always**: ボリュームのマウント時に、常にボリュームのパーミッションと所有者を変更します。

fsGroupChangePolicy の例

```
securityContext:
  runAsUser: 1000
  runAsGroup: 3000
  fsGroup: 2000
  fsGroupChangePolicy: "OnRootMismatch" 1
  ...
```

1 **OnRootMismatch** は、再帰的なパーミッション変更をスキップさせるため、Pod のタイムアウトの問題を回避するのに役立ちます。

注記

fsGroupChangePolicyfield は、secret、configMap、emptydir などの一時ボリュームタイプには影響を及ぼしません。

第5章 永続ボリュームの拡張

MicroShift で永続ボリュームを拡張する方法を学びます。

5.1. CSI ボリュームの拡張

Container Storage Interface (CSI) を使用して、作成後にストレージボリュームを拡張することができます。

CSI ボリューム拡張は、以下をサポートしません。

- ボリューム拡張時の障害からの復旧
- 縮小

前提条件

- 基礎となる CSI ドライバーがサイズ変更をサポートする。
- 動的プロビジョニングが使用される。
- 制御する側の **StorageClass** オブジェクトには **allowVolumeExpansion** が **true** に設定されている。詳細は、「ボリューム拡張サポートの有効化」を参照してください。

手順

1. 永続ボリューム要求 (PVC) の場合は、**.spec.resources.requests.storage** を必要な新しいサイズに設定します。
2. PVC の **status.conditions** フィールドを監視し、サイズ変更が完了したかどうかを確認します。MicroShift は、拡張中に **Resizing** の条件を PVC に追加します。これは、拡張の完了後に削除されます。

5.2. ローカルボリュームの拡張

ローカルストレージ Operator (LSO) を使用して作成された永続ボリューム (PV) および永続ボリューム要求 (PVC) を手動で拡張できます。

手順

1. 基礎となるデバイスを拡張します。これらのデバイスで適切な容量が利用できるようにします。
2. PV の **.spec.capacity** フィールドを編集して、新しいデバイスサイズに一致するように対応する PV オブジェクトを更新します。
3. PVC を PVet にバインドするためのストレージクラスに **allowVolumeExpansion:true** を設定します。
4. PVC に新しいサイズに一致するように **.spec.resources.requests.storage** を設定します。

Kubelet は、ボリューム上の基礎となるファイルシステムを自動的に拡張するはずです。必要に応じて、新しいサイズを反映するように PVC の status フィールドを更新します。

5.3. ファイルシステムを使用した永続ボリューム要求 (PVC) の拡張

GCE Persistent Disk ボリューム (gcePD)、AWS Elastic Block Store EBS (EBS)、Cinder など、ファイルシステムのサイズ変更が必要なボリュームタイプに基づいて PVC を拡張するには、2 段階のプロセスがあります。まず、クラウドプロバイダーのボリュームオブジェクトを拡張します。次に、ノードのファイルシステムを拡張します。

ノードでのファイルシステムの拡張は、新規 Pod がボリュームと共に起動する場合にのみ実行されます。

前提条件

- 制御する側の **StorageClass** オブジェクトでは、**allowVolumeExpansion** が **true** に設定されている必要がある。

手順

1. **spec.resources.requests** を編集して PVC を編集し、新規サイズを要求します。たとえば、以下では **ebs** PVC を 8 Gi に拡張します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ebs
spec:
  storageClass: "storageClassWithFlagSet"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi (1)
```

(1) **spec.resources.requests** をさらに大きな量を表す値に更新すると、PVC が拡張されます。

2. クラウドプロバイダーオブジェクトのサイズ変更が終了すると、PVC は **FileSystemResizePending** に設定されます。以下のコマンドを入力して状態を確認します。

```
$ oc describe pvc <pvc_name>
```

3. クラウドプロバイダーオブジェクトのサイズ変更が終了すると、**PersistentVolume** オブジェクトは **PersistentVolume.Spec.Capacity** に新規に要求されたサイズを反映します。この時点で、PVC から新規 Pod を作成または再作成してファイルシステムのサイズ変更を終了できます。Pod が実行している場合は、新たに要求されたサイズが利用可能になり、**FileSystemResizePending** 状態が PVC から削除されます。

5.4. ボリューム拡張時の障害からの復旧

基礎となるストレージの拡張に失敗した場合に MicroShift の管理者は Persistent Volume Claim (永続ボリューム要求、PVC) の状態を手動で復旧し、サイズ変更要求を取り消します。そうでない場合には、サイズ変更要求がコントローラーによって継続的に再試行されます。

手順

1. **Retain** 回収ポリシーで要求 (PVC) にバインドされている永続ボリューム (PV) にマークを付けます。これは、PV を編集し、**persistentVolumeReclaimPolicy** を **Retain** に変更して実行できます。

2. PVC を削除します。

3. PV を手動で編集し、PV 仕様から **claimRef** エントリーを削除して、新しく作成された PVC を **Retain** とマークされた PV にバインドできるようにします。これで、PV には **Available** というマークが付けられます。

4. より小さいサイズ、または基礎となるストレージプロバイダーによって割り当て可能なサイズで PVC を再作成します。

5. PVC の **volumeName** フィールドを PV の名前に設定します。これにより、PVC がプロビジョニングされた PV にのみバインドされます。

6. PV で回収ポリシーを復元します。

第6章 LVMS プラグインを使用した動的ストレージ

MicroShift は動的ストレージプロビジョニングに対応しています。動的ストレージプロビジョニングは、論理ボリュームマネージャーストレージ (LVMS) Container Storage Interface (CSI) プロバイダーですぐに使用できます。LVMS プラグインは、Kubernetes の論理ボリューム管理 (LVM) 論理ボリューム (LV) を管理するための CSI プラグインである TopoLVM の Red Hat ダウンストリームバージョンです。

LVMS は、適切に設定された永続ボリューム要求 (PVC) を使用して、コンテナーワークロード用の新しい LVM 論理ボリュームをプロビジョニングします。各 PVC は、ホストノード上の LVM ボリュームグループ (VG) を表すストレージクラスを参照します。LV は、スケジュールされた Pod に対してのみプロビジョニングされます。

6.1. LVMS システム要件

MicroShift で LVMS を使用するには、次のシステム仕様が必要です。

6.1.1. ボリュームグループ名

/etc/microshift/ ディレクトリーにある **lvmd.yaml** ファイルで LVMS を設定しなかった場合、MicroShift は **vgs** コマンドを実行してデフォルトのボリュームグループ (VG) を動的に割り当てます。

- VG が 1 つだけ検出された場合、MicroShift はデフォルトの VG を割り当てます。
- 複数の VG が存在する場合、**microshift** という名前の VG がデフォルトとして割り当てられます。
- **microshift** という名前の VG が存在しない場合、LVMS はデプロイされません。

MicroShift ホストにボリュームグループがない場合、LVMS は無効になります。

特定の VG を使用する場合は、その VG を選択するように LVMS を設定する必要があります。設定ファイルで VG のデフォルト名を変更できます。詳細は、このドキュメントの「LVMS の設定」セクションを参照してください。

設定ファイルで VG のデフォルト名を変更できます。詳細は、このドキュメントの「LVMS の設定」セクションを参照してください。

MicroShift が起動したら、**lvmd.yaml** を更新して VG を追加または削除できます。変更を実装するには、MicroShift を再起動する必要があります。**lvmd.yaml** が削除された場合、MicroShift はデフォルトの VG を再度検索します。

6.1.2. ボリュームサイズの増分

LVMS は、ストレージを 1 ギガバイト (GB) 単位でプロビジョニングします。ストレージ要求は、最も近い GB に切り上げられます。ボリュームグループの容量が 1 GB 未満の場合、**PersistentVolumeClaim** は **ProvisioningFailed** イベントを登録します。次に例を示します。

出力例

```
Warning  ProvisioningFailed    3s (x2 over 5s)  topolvm.cybozu.com_topolvm-controller-858c78d96c-xttzp_0fa83aef-2070-4ae2-bcb9-163f818dcd9f failed to provision volume with StorageClass "topolvm-provisioner": rpc error: code = ResourceExhausted desc = no enough space left on VG: free=(BYTES_INT), requested=(BYTES_INT)
```

6.2. LVMS のデプロイメント

MicroShift の起動後、LVMS は **openshift-storage** namespace のクラスターに自動的にデプロイされます。

LVMS は **StorageCapacity** 追跡を使用して、要求されたストレージがボリュームグループの空きストレージよりも大きい場合は、LVMS PVC を持つ Pod がスケジュールされないようにします。**StorageCapacity** 追跡の詳細は、Storage Capacity を参照してください。

6.3. LVM STORAGE で使用するデバイスのサイズを設定する際の制限事項

LVM Storage を使用したストレージのプロビジョニングで使用できるデバイスのサイズを設定する際の制限は、次のとおりです。

- プロビジョニングできる合計ストレージサイズは、基礎となる論理ボリュームマネージャー (LVM) シンプールのサイズとオーバープロビジョニング係数によって制限されます。
- 論理ボリュームのサイズは、物理エクステント (PE) のサイズと論理エクステント (LE) のサイズによって異なります。
  - PE および LE のサイズは、物理デバイスおよび論理デバイスの作成時に定義できます。
  - デフォルトの PE および LE サイズは 4 MB です。
  - PE のサイズを大きくした場合、LVM の最大サイズは、カーネルの制限とディスク領域によって決定されます。
  - デフォルトの PE および LE サイズを使用する Red Hat Enterprise Linux (RHEL) 9 のサイズ制限は 8 EB です。
  - 以下は、各ファイルシステムタイプに対して要求できる最小ストレージサイズです。
    - **block**: 8 MiB
    - **xfs**: 300 MiB
    - **ext4**: 32 MiB

6.4. LVMS 設定ファイルの作成

MicroShift が実行するとき、指定されている場合は **/etc/microshift/lvmd.yaml** からの LVMS 設定が使用されます。作成するすべての設定ファイルを **/etc/microshift/** ディレクトリーに配置する必要があります。

手順

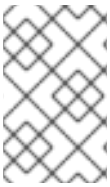
- **lvmd.yaml** 設定ファイルを作成するには、次のコマンドを実行します。

```
$ sudo cp /etc/microshift/lvmd.yaml.default /etc/microshift/lvmd.yaml
```

6.5. 基本的な LVMS 設定例

MicroShift は、LVM 設定のパススルーをサポートしており、カスタムボリュームグループ、シンボリュームプロビジョニングパラメーター、予約済みの未割り当てボリュームグループ領域を指定できます。作成した LVMS 設定ファイルはいつでも編集できます。ファイルの編集後に設定の変更をデプロイ

するには、MicroShift を再起動する必要があります。



注記

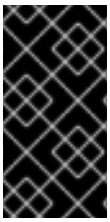
ボリュームのスナップショットを作成する必要がある場合は、**lvmd.conf** ファイルでシンプロビジョニングを使用する必要があります。ボリュームスナップショットを作成する必要がない場合は、シックボリュームを使用できます。

次の **lvmd.yaml** サンプルファイルは、基本的な LVMS 設定を示しています。

LVMS 設定例

```
socket-name: 1
device-classes: 2
  - name: "default" 3
    volume-group: "VGNAMEHERE" 4
    spare-gb: 0 5
    default: 6
```

1. 文字列。gRPC の UNIX ドメインソケットエンドポイント。デフォルトは '/run/lvmd/lvmd.socket' です。
2. 各 **device-class** の設定のマップのリスト。
3. 文字列。**device-class** の名前。
4. 文字列。**device-class** が論理ボリュームを作成するグループ。
5. 未署名の 64 ビット整数ボリュームグループに未割り当てのままにする GB 単位のストレージ容量。デフォルトは **0** です。
6. ブール値。**device-class** がデフォルトで使用されることを示します。デフォルトは **false** です。これが **true** に設定されている場合は、YAML ファイルの値に少なくとも 1 つの値を入力する必要があります。



重要

複数の PVC が同時に作成されると、競合状態が原因で、LVMS は割り当てられた領域を正確に追跡してデバイスクラスの **spare-gb** を保持できなくなります。個別のボリュームグループとデバイスクラスを使用して、非常に動的なワークロードのストレージを相互に保護します。

6.6. LVMS の使用

LVMS **StorageClass** は、デフォルトの **StorageClass** でデプロイされます。**.spec.storageClassName** が定義されていない **PersistentVolumeClaim** オブジェクトには、デフォルトの **StorageClass** からプロビジョニングされた **PersistentVolume** が自動的に含まれます。次の手順を使用して、論理ボリュームをプロビジョニングし、Pod にマウントします。

手順

- 論理ボリュームを Pod にプロビジョニングしてマウントするには、次のコマンドを実行します。

```
$ cat <<EOF | oc apply -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-lv-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1G
---
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx
    command: ["/usr/bin/sh", "-c"]
    args: ["sleep", "1h"]
    volumeMounts:
    - mountPath: /mnt
      name: my-volume
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
  volumes:
    - name: my-volume
      persistentVolumeClaim:
        claimName: my-lv-pvc
EOF
```

6.6.1. デバイスクラス

device-classes 配列を論理ボリュームマネージャーストレージ (LVMS) 設定に追加することで、カスタムデバイスクラスを作成できます。配列を **/etc/microshift/lvmd.yaml** 設定ファイルに追加します。1 つのデバイスクラスをデフォルトとして設定する必要があります。設定の変更を有効にするには、MicroShift を再起動する必要があります。

警告

デバイスクラスに接続されている永続ボリュームまたは **VolumeSnapshotContent** オブジェクトがまだあるにもかかわらず、デバイスクラスを削除すると、シックプロビジョニングとシンプロビジョニングの両方が中断されます。

device-classes 配列で複数のデバイスクラスを定義できます。これらのクラスは、シックボリューム設定とシンボリューム設定を組み合わせて使用できます。

混合 デバイスクラス 配列の例

```
socket-name: /run/topolvm/lvmd.sock
device-classes:
  - name: ssd
    volume-group: ssd-vg
    spare-gb: 0 (1)
    default: true
  - name: hdd
    volume-group: hdd-vg
    spare-gb: 0
  - name: thin
    spare-gb: 0
    thin-pool:
      name: thin
      overprovision-ratio: 10
    type: thin
    volume-group: ssd
  - name: striped
    volume-group: multi-pv-vg
    spare-gb: 0
    stripe: 2
    stripe-size: "64"
    lvcreate-options: (2)
```

(1) 予備容量を **0** 以外に設定すると、予想よりも多くのスペースが割り当てられる可能性があります。

(2) **lvcreate** コマンドに渡す追加の引数 (**--type=<type>** など)。**lvcreate-options** の値は、MicroShift や LVMS によって検証されません。これらのオプションの値は、そのまま **lvcreate** コマンドに渡されます。ここで指定したオプションが正しいことを確認してください。

第7章 ボリュームスナップショットの使用

クラスター管理者は、ボリュームスナップショットを使用すると、サポート対象の MicroShift 論理ボリュームマネージャーストレージ (LVMS) Container Storage Interface (CSI) プロバイダーを使用してデータ損失を防ぐことができます。永続ボリューム に関する知識が必要です。

スナップショットは、特定の時点でのクラスター内のストレージボリュームの状態を表します。ボリュームスナップショットを使用して、新しいボリュームをプロビジョニングすることもできます。スナップショットは、元のデータと同じデバイス上にある読み取り専用の論理ボリューム (LV) として作成されます。

クラスター管理者は、CSI ボリュームスナップショットを使用して以下のタスクを実行できます。

- 既存の永続ボリューム要求 (PVC) のスナップショットを作成します。
- ボリュームスナップショットを安全な場所にバックアップします。
- ボリュームスナップショットを別の PVC として復元します。
- 既存のボリュームスナップショットを削除します。

重要

MicroShift では、Logical Volume Manager Storage (LVMS) プラグイン CSI ドライバーのみがサポートされています。

関連情報

- 永続ボリュームについて
- 論理ボリュームの設定および管理
- CSI スナップショット: **VolumeSnapshot** API
- VolumeSnapshot API 仕様

7.1. LVM シンボリュームについて

ボリュームスナップショットの作成やボリュームクローン作成などの高度なストレージ機能を使用するには、以下のアクションを実行する必要があります。

- 論理ボリュームマネージャーストレージ (LVMS) プロバイダーとクラスターの両方を設定します。
- RHEL for Edge 上に論理ボリュームマネージャー (LVM) シンプールをプロビジョニングします。
- LVM シンプールをボリュームグループに接続します。

重要

Container Storage Interface (CSI) スナップショットを作成するには、RHEL for Edge ホスト上でシンボリュームを設定する必要があります。CSI はボリュームの縮小をサポートしていません。

LVMS がシン論理ボリューム (LV) を管理するには、**etc/lvmd.yaml** 設定ファイルでシンプールの **デバイスクラス** アレイを指定する必要があります。複数のシンプールデバイスクラスが許可されます。

追加のストレージプールがデバイスクラスで設定されている場合、ストレージプールをユーザーおよびワークロードに公開するには、追加のストレージクラスも存在する必要があります。シンプールで動的プロビジョニングを有効にするには、**StorageClass** リソースがクラスター上に存在する必要があります。**StorageClass** リソースは、**topolvm.io/device-class** パラメーターでソース **device-class** クラス配列を指定します。

シンプール用の単一のデバイスクラスを指定する lvmd.yaml ファイルの例

```
socket-name: 1
device-classes: 2
  - name: thin 3
    default: true
    spare-gb: 0 4
    thin-pool:
      name: thin
      overprovision-ratio: 10 5
    type: thin 6
    volume-group: ssd 7
```

1. 文字列。gRPC の UNIX ドメインソケットエンドポイント。デフォルトは **/run/lvmd/lvmd.socket** です。
2. 各 **device-class** の設定のマップのリスト。
3. 文字列。**device-class** の一意の名前。
4. 未署名の 64 ビット整数ボリュームグループに未割り当てのままにする GB 単位のストレージ容量。デフォルトは **0** です。
5. 同じプールを共有する複数のシンプロビジョニングされたデバイスがある場合に、これらのデバイスは過剰にプロビジョニングされる可能性があります。オーバープロビジョニングには、1 以上の浮動小数点値が必要です。
6. ボリュームのスナップショットの作成には、シンプロビジョニングが必要です。
7. 文字列。**device-class** が論理ボリュームを作成するグループ。

重要

複数の PVC が同時に作成されると、競合状態が原因で、LVMS は割り当てられた領域を正確に追跡してデバイスクラスのストレージ容量を保持できなくなります。個別のボリュームグループとデバイスクラスを使用して、非常に動的なワークロードのストレージを相互に保護します。

関連情報

- ホストにシンプールを作成するには、シンプロビジョニングされたボリュームの作成および管理を参照してください。
- シンプロビジョニングされた論理ボリュームの作成

- シンプロビジョニングされたボリュームの設定および管理
- ストレージクラス
- ストレージデバイスクラス

7.1.1. ストレージクラス

ストレージクラスは、デバイスクラスを選択するためのワークロード層インターフェイスを提供します。MicroShift では、次のストレージクラスパラメーターがサポートされています。

- **csi.storage.k8s.io/fstype** パラメーターは、ファイルシステムタイプを選択します。**xfs** および **ext4** の両方のファイルシステムタイプがサポートされています。
- **topolvm.io/device-class** パラメーターは、デバイスクラスの名前です。デバイスクラスが指定されていない場合は、デフォルトのデバイスクラスが使用されます。

複数のストレージクラスが同じデバイスクラスを参照できます。**xfs** や **ext4** バリアントなど、同じバッキングデバイスクラスに、さまざまなパラメーターセットを指定できます。

MicroShift のデフォルトのストレージクラスリソースの例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" (1)
  name: topolvm-provisioner
parameters:
  "csi.storage.k8s.io/fstype": "xfs" (2)
provisioner: topolvm.io (3)
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer (4)
allowVolumeExpansion: (5)
```

(1) デフォルトのストレージクラスの例。PVC がストレージクラスを指定しない場合、このクラスが使用されます。クラスター内に存在できるデフォルトのストレージクラスは 1 つだけです。このアノテーションに値を割り当てないというオプションもサポートされています。

(2) ボリューム上にプロビジョニングするファイルシステムを指定します。オプションは "xfs" および "ext4" です。

(3) このクラスを管理するプロビジョナーを識別します。

(4) クライアント Pod が存在するか、または即時にボリュームをプロビジョニングするかどうかを指定します。オプションは **WaitForFirstConsumer** および **Immediate** です。スケジュール可能な Pod に対してのみストレージがプロビジョニングされるようにするには、**WaitForFirstConsumer** を使用することを推奨します。

(5) **StorageClass** からプロビジョニングされる PVC で拡張を許可するかどうかを指定します。MicroShift の LVMS CSI プラグインはボリューム拡張をサポートしますが、この値が **false** に設定されている場合、拡張はブロックされます。

関連情報

- ストレージクラスの定義
- ストレージデバイスクラス

7.2. ボリュームスナップショットクラス

スナップショットは、LVMS でサポートされる CSI ストレージ機能です。動的スナップショットを有効にするには、少なくとも 1 つの **VolumeSnapshotClass** 設定ファイルがクラスターに存在する必要があります。

重要

論理ボリュームのスナップショットを作成するには、シン論理ボリュームを有効にする必要があります。

VolumeSnapshotClass 設定ファイルの例

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: topolvm-snapclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true" (1)
driver: topolvm.io (2)
deletionPolicy: Delete (3)
```

(1) ユーザーによるボリュームのスナップショットの要求 **VolumeSnapshot** で何も指定されていない場合に、どの **VolumeSnapshotClass** 設定ファイルを使用するかを決定します。

(2) このクラスのユーザーによるボリュームのスナップショット要求をどのスナップショットプロビジョナーが管理するかを指定します。

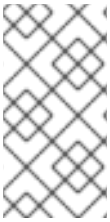
(3) バインドされた **VolumeSnapshot** が削除されたときに、**VolumeSnapshotContent** オブジェクトとバッキングスナップショットを保持するか削除するかを決定します。有効な値は **Retain** または **Delete** です。

関連情報

- OpenShift CSI ボリュームスナップショット

7.3. ボリュームスナップショットについて

論理ボリュームマネージャー (LVM) シンボリュームでボリュームスナップショットを使用すると、MicroShift クラスターで実行されているアプリケーションからのデータ損失を防ぐことができます。MicroShift は、論理ボリュームマネージャーストレージ (LVMS) Container Storage Interface (CSI) プロバイダーのみをサポートします。



注記

LVMS は、**WaitForFirstConsumer** に設定されているストレージクラスの **volumeBindingMode** のみをサポートします。この設定では、Pod がストレージボリュームをマウントする準備ができるまで、ストレージボリュームがプロビジョニングされません。

単一 Pod および PVC をデプロイするワークロードの例

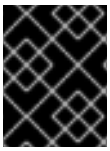
```
$ oc apply -f - <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-claim-thin
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: topolvm-provisioner-thin
---
apiVersion: v1
kind: Pod
metadata:
  name: base
spec:
  containers:
  - command:
     - nginx
     - -g
     - 'daemon off;'
    image: registry.redhat.io/rhel8/nginx-
122@sha256:908ebb0dec0d669caaf4145a8a21e04fdf9ebffbba5fd4562ce5ab388bf41ab2
    name: test-container
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
    volumeMounts:
    - mountPath: /vol
      name: test-vol
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  volumes:
  - name: test-vol
    persistentVolumeClaim:
      claimName: test-claim-thin
EOF
```

7.3.1. ボリュームスナップショットの作成

MicroShift ストレージボリュームのスナップショットを作成するには、まず RHEL for Edge とクラスターを設定する必要があります。以下の手順例では、ソースボリュームがマウントされている Pod が削除されます。Pod を削除すると、スナップショットの作成時にデータが Pod に書き込まれないようにします。実行可能なスナップショットを作成するには、スナップショット中にデータが書き込まれていない点を確認することが重要です。

前提条件

- MicroShift クラスターへの root アクセス権がある。
- MicroShift クラスターが実行されている。
- デバイスクラスが LVM シンプールを定義している。
- **volumeSnapshotClass** が **driver: topolvm.io** を指定している。
- ソース PVC に割り当てられたワークロードは一時停止または削除されている。これにより、データの破損を回避できます。



重要

スナップショットの作成時には、ボリュームへの書き込みをすべて停止する必要があります。書き込みを停止しないと、データが破損する可能性があります。

手順

1. 次の 2 つの手順のいずれかを使用して、スナップショットの作成中にデータがボリュームに書き込まれないようにします。
   a. 次のコマンドを実行して、Pod を削除し、スナップショット中にデータがボリュームに書き込まれないようにします。

      ```
      $ oc delete my-pod
      ```

   b. レプリケーションコントローラーで管理されている Pod 上のレプリカ数をゼロにスケールします。カウントをゼロに設定すると、新しい Pod が削除されたときにすぐに新しい Pod が作成されなくなります。

2. ボリュームへの書き込みがすべて停止したら、以下のようなコマンドを実行します。独自の設定情報を挿入します。

   スナップショット設定の例

   ```
   # oc apply -f <<EOF
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot (1)
   metadata:
     name: <snapshot_name> (2)
   spec:
     volumeSnapshotClassName: topolvm-snapclass (3)
     source:
       persistentVolumeClaimName: test-claim-thin (4)
   EOF
   ```

1. **VolumeSnapshot** オブジェクトを作成します。
2. スナップショットに指定する名前。
3. 必要な **VolumeSnapshotClass** オブジェクトの名前を指定します。
4. **persistentVolumeClaimName** または **volumeSnapshotContentName** のいずれかを指定します。この例では、**test-claim-thin** という名前の PVC からスナップショットが作成されます。

3. 次のコマンドを実行して、ストレージドライバーがスナップショットの作成を完了するのを待ちます。

```
$ oc wait volumesnapshot/<snapshot_name> --for=jsonpath\='{.status.readyToUse}=true'
```

次のステップ

1. **volumeSnapshot** オブジェクトが **ReadyToUse** 状態にある場合は、今後の PVC 用のボリュームとして復元できます。Pod を再起動するか、レプリカ数を必要な数に戻します。
2. ボリュームスナップショットの作成後に、ソース PVC を新規 Pod に再マウントできます。

重要

ボリュームスナップショットは元のデータと同じデバイスにあります。ボリュームスナップショットをバックアップとして使用するには、スナップショットを安全な場所に移動します。

7.3.2. ボリュームスナップショットのバックアップ

MicroShift クラスター上で実行されているアプリケーションからのデータのスナップショットは、元のデータと同じデバイス上にある読み取り専用の論理ボリューム (LV) として作成されます。ローカルボリュームを永続ボリューム (PV) としてコピーし、バックアップコピーとして使用する前に、ローカルボリュームを手動でマウントする必要があります。MicroShift ストレージボリュームのスナップショットをバックアップとして使用するには、ローカルホスト上でスナップショットを見つけて、セキュアな場所に移動します。

特定のスナップショットを検索してコピーするには、以下の手順を行います。

前提条件

- ホストマシンへの root アクセス権限がある。
- 既存のボリュームスナップショットがある。

手順

1. 次のコマンドを実行して、ボリュームスナップショットの名前を取得します。

```
$ oc get volumesnapshot -n <namespace> <snapshot_name> -o 'jsonpath=
{.status.volumeSnapshotContentName}'
```

2. 以下のコマンドを使用して、前のステップで取得した名前を挿入して、ストレージバックエンドで作成したボリュームの一意の ID を取得します。

```
$ oc get volumesnapshotcontent snapcontent-<retrieved_volume_identity> -o 'jsonpath=
{.status.snapshotHandle}'
```

3. 次のコマンドを実行して、前の手順で取得したボリュームの一意 ID を使用してスナップショットを表示し、どのボリュームをバックアップするかを決定します。

```
$ sudo lvdisplay <retrieved_snapshot_handle>
```

出力例

```
--- Logical volume ---
LV Path                /dev/rhel/732e45ff-f220-49ce-859e-87ccca26b14c
LV Name                732e45ff-f220-49ce-859e-87ccca26b14c
VG Name                rhel
LV UUID                6Ojwc0-YTfp-nKJ3-F9FO-PvMR-Ic7b-LzNGSx
LV Write Access        read only
LV Creation host, time rhel-92.lab.local, 2023-08-07 14:45:26 -0500
LV Pool name           thinpool
LV Thin origin name    a2d2dcdc-747e-4572-8c83-56cd873d3b07
LV Status              available
# open                 0
LV Size                1.00 GiB
Mapped size            1.04%
Current LE             256
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:11
```

4. 以下のコマンドを実行して、LV のマウントに使用するディレクトリーを作成します。

```
$ sudo mkdir /mnt/snapshot
```

5. 次のコマンドを実行して、取得したスナップショットハンドルのデバイス名を使用して LV をマウントします。

```
$ sudo mount /dev/<retrieved_snapshot_handle> /mnt/snapshot
```

6. 次のコマンドを実行して、マウントされた場所からファイルをコピーし、安全な場所に保存します。

```
$ sudo cp -r /mnt/snapshot <destination>
```

7.3.3. ボリュームスナップショットの復元

以下のワークフローは、スナップショットの復元を示しています。この例では、ソースの永続ボリューム要求 (PVC) に書き込まれたデータが保存され、新しい PVC に復元されることを確認するための検証手順も示されています。

重要

スナップショットは、スナップショットのソースボリュームと全く同じサイズの PVC に復元される必要があります。より大きな PVC が必要な場合は、スナップショットが正常に復元された後に PVC のサイズを変更できます。

手順

1. 次のコマンドを入力して、**VolumeSnapshot** オブジェクトを永続ボリューム要求のデータソースとして指定して、スナップショットを復元します。

```
$ oc apply -f <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-restore
spec:
  accessModes:
  - ReadWriteOnce
  dataSource:
    apiGroup: snapshot.storage.k8s.io
    kind: VolumeSnapshot
    name: my-snap
  resources:
    requests:
      storage: 1Gi
  storageClassName: topolvm-provisioner-thin
---
apiVersion: v1
kind: Pod
metadata:
  name: base
spec:
  containers:
  - command:
      - nginx
     - -g
     - 'daemon off;'
    image: registry.redhat.io/rhel8/nginx-
122@sha256:908ebb0dec0d669caaf4145a8a21e04fdf9ebffbba5fd4562ce5ab388bf41ab2
    name: test-container
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
    volumeMounts:
    - mountPath: /vol
      name: test-vol
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  volumes:
  - name: test-vol
```

```
    persistentVolumeClaim:
      claimName: snapshot-restore
EOF
```

検証

1. Pod が **Ready** 状態になるまで待機します。

```
$ oc wait --for=condition=Ready pod/base
```

2. 新規 Pod の準備ができたら、スナップショット内でアプリケーションのデータが正しいことを確認します。

関連情報

- ボリュームスナップショットの復元

7.3.4. ボリュームスナップショットの削除

MicroShift によるボリュームスナップショットの削除方法を設定できます。

手順

1. 以下の例のように、**VolumeSnapshotClass** オブジェクトで必要な削除ポリシーを指定します。

volumesnapshotclass.yaml

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snap
driver: hostpath.csi.k8s.io
deletionPolicy: Delete (1)
```

(1) ボリュームスナップショットの削除時に **Delete** 値を設定すると、**VolumeSnapshotContent** オブジェクトと共に基礎となるスナップショットが削除されます。**Retain** 値を設定すると、基礎となるスナップショットと **VolumeSnapshotContent** オブジェクトの両方が残ります。
Retain 値を設定し、対応する **VolumeSnapshotContent** オブジェクトを削除せずに **VolumeSnapshot** オブジェクトを削除すると、コンテンツは残ります。スナップショット自体はストレージバックエンドにも保持されます。

2. 以下のコマンドを入力してボリュームスナップショットを削除します。

```
$ oc delete volumesnapshot <volumesnapshot_name>
```

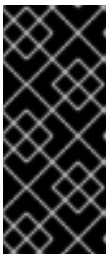
出力例

```
volumesnapshot.snapshot.storage.k8s.io "mysnapshot" deleted
```

3. 削除ポリシーが **Retain** に設定されている場合は、以下のコマンドを入力してボリュームスナップショットのコンテンツを削除します。

```
$ oc delete volumesnapshotcontent <volumesnapshotcontent_name>
```

4. オプション: **VolumeSnapshot** オブジェクトが正常に削除されていない場合は、以下のコマンドを実行して残されているリソースのファイナライザーを削除し、削除操作を続行できるようにします。



重要

永続ボリューム要求またはボリュームスナップショットのコンテンツのいずれかから **VolumeSnapshot** オブジェクトへの既存の参照がない場合にのみファイナライザーを削除します。**--force** オプションを使用する場合でも、すべてのファイナライザーが削除されるまで削除操作でスナップショットオブジェクトは削除されません。

```
$ oc patch -n $PROJECT volumesnapshot/$NAME --type=merge -p '{"metadata":
{"finalizers":null}}'
```

出力例

```
volumesnapshotclass.snapshot.storage.k8s.io "csi-ocs-rbd-snapclass" deleted
```

ファイナライザーが削除され、ボリュームスナップショットが削除されます。

7.4. LVM ボリュームのクローン作成について

論理ボリュームマネージャーストレージ (LVMS) は、論理ボリュームマネージャー (LVM) シンボリュームの永続ボリューム要求 (PVC) のクローン作成をサポートします。クローンは、既存のボリュームの複製で、他のボリュームと同様に使用できます。プロビジョニング時に、データソースが同じ namespace 内のソース PVC を参照すると、元のボリュームと全く同じ複製が作成されます。クローン作成された PVC の作成後に、これは新規オブジェクトと見なされ、ソース PVC と完全に分離されます。クローンは、作成された時点のソースからのデータを表します。

注記

ソースおよび宛先 PVC が同じ namespace にある場合のみクローン作成が可能です。PVC クローンを作成するには、RHEL for Edge ホスト上にシンボリュームを設定する必要があります。

関連情報

- CSI ボリュームのクローン作成
- シングルノード OpenShift の LVMS ボリュームのクローン作成
- クローンを有効にするようにホストを設定するには、LVM シンボリュームについてを参照してください。

第8章 KUBE STORAGE VERSION MIGRATOR を使用したストレージの移行

ストレージバージョンの移行は、クラスター内の既存のオブジェクトを現在のバージョンから最新バージョンに更新するために使用されます。Kube Storage Version Migrator は、組み込みのコントローラーであり、リソースを再作成することなくリソースを移行するために MicroShift で使用されます。ユーザーまたはコントローラーは、Migrator コントローラーを介して移行を要求する **StorageVersionMigration** カスタムリソース (CR) を作成できます。

8.1. ストレージ移行要求の作成

ストレージの移行は、保存されたデータを最新のストレージバージョンに更新するプロセスです (例: **v1beta1** から **v1beta2)**。ストレージのバージョンを更新するには、次の手順を実行します。

手順

- ユーザーまたは **StorageVersionMigration** API をサポートするコントローラーが移行リクエストをトリガーする必要があります。参考として次のリクエストの例を使用してください。

  要求の例

```
apiVersion: migration.k8s.io/v1alpha1
kind: StorageVersionMigration
metadata:
  name: snapshot-v1
spec:
  resource:
    group: snapshot.storage.k8s.io
    resource: volumesnapshotclasses (1)
    version: v1 (2)
```

  (1) リソースの複数名を使用する必要があります。

  (2) 更新先のバージョン。

- 移行の進行状況は、**StorageVersionMigration** ステータスに通知されます。

注記

- グループまたはリソースの名前が間違っているために障害が発生する可能性があります。
- 移行の失敗は、以前のバージョンと最新のバージョンの間に互換性がない場合にも発生する可能性があります。