Red Hat build of MicroShift 4.12

ストレージ

クラスターストレージの設定と管理

Last Updated: 2023-10-20

Red Hat build of MicroShift 4.12 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 RED HAT BUILD OF MICROSHIFT ストレージの概要 3
1.1. ストレージタイプ 3

第2章 RED HAT BUILD OF MICROSHIFT の一時ストレージについて 4
2.1. 概要 4
2.2. 一時ストレージのタイプ 4
2.3. 一時ストレージ管理 5
2.4. 一時ストレージのモニタリング 6

第3章 RED HAT BUILD OF MICROSHIFT の汎用エフェメラルボリューム 7
3.1. 概要 7
3.2. ライフサイクルおよび永続ボリューム要求 7
3.3. セキュリティー 8
3.4. 永続ボリューム要求の命名 8
3.5. 汎用一時ボリュームの作成 8

第4章 RED HAT BUILD OF MICROSHIFT の永続ストレージについて 10
4.1. 永続ストレージの概要 10
4.2. ボリュームおよび要求のライフサイクル 10
4.3. 永続ボリューム 13
4.4. 永続ボリューム要求 (PVC) 15
4.5. FSGROUP を使用した POD タイムアウトの削減 17

第5章 RED HAT BUILD OF MICROSHIFT の永続ボリュームの拡張 19
5.1. CSI ボリュームの拡張 19
5.2. ローカルボリュームの拡張 19
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 20
5.4. ボリューム拡張時の障害からの復旧 20

第6章 LVMS プラグインを使用した動的ストレージ 22
6.1. LVMS システム要件 22
6.2. LVMS のデプロイメント 22
6.3. LVMS 設定ファイルの作成 23
6.4. LVMS の設定 23
6.5. LVMS の使用 24

第1章 RED HAT BUILD OF MICROSHIFT ストレージの概要

Red Hat build of 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。Red Hat build of 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

Red Hat build of 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。Red Hat build of 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 Persistent Volume Claim (永続ボリューム要求、PVC) 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。Red Hat build of 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 RED HAT BUILD OF MICROSHIFT の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 Red Hat build of 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しているすべての Pod、システムによる他の使用、および Red Hat build of 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、Red Hat build of 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。Red Hat build of 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**

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。接尾辞の大文字と小文字は区別する必要があります。400m の一時ストレージを指定すると、おそらく意図されたものであろう 400 メビバイト (400Mi) または 400 メガバイト (400M) ではなく、0.4 バイトが要求されます。

次の例は、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。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。

```
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" (3)
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
      emptyDir: {}
```

(1)(3) ローカル一時ストレージの要求。

2 ローカル一時ストレージの制限。

Pod 仕様のこの設定は、スケジューラーが Pod のスケジューリングを決定する方法と、kubelet が Pod を削除する方法に影響します。まず、スケジューラーは、スケジュールされたコンテナーのリソース要求の合計がノードの容量よりも少ないことを確認します。この場合は、使用可能な一時ストレージ (割り当て可能なリソース) が 4GiB を超える場合にのみ、Pod をノードに割り当てることができます。

次に、コンテナーレベルでは、最初のコンテナーがリソース制限を設定するため、kubelet エビクションマネージャーはこのコンテナーのディスク使用量を測定し、このコンテナーのストレージ使用量がその制限 (4GiB) を超える場合は、Pod をエビクトします。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。この合計使用量が全体的な Pod ストレージの制限 (4GiB) を超えた場合、kubelet は Pod にエビクションのマークも付けます。

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/sda1    69G   32G   34G  49% /
```

第3章 RED HAT BUILD OF MICROSHIFT の汎用エフェメラルボリューム

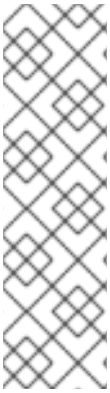
3.1. 概要

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。

この制約により、以下の Container Storage Interface (CSI) ドライバーは、以下の汎用一時ボリューム機能をサポートしません。

- Azure Disk CSI ドライバーは、サイズ変更をサポートしません。
- Cinder CSI ドライバーは、スナップショット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。

- 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直ちに
  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選択するように強制されます。
- 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。

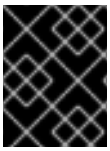
PVC に対する通常の namespace クォータは依然として適用されるため、ユーザーがこの新しいメカニズムを使用できる場合でも、これを使用して他のポリシーを回避できません。

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: "gp2-csi"
          resources:
            requests:
              storage: 1Gi
```

(1) 汎用一時ボリューム要求

第4章 RED HAT BUILD OF MICROSHIFT の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。Red Hat build of 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ついての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. 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 Red Hat build of 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 にバインドすることはできません。これには、バインドされた PV を単一の namespace にスコープする効果があります。

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、Red Hat build of MicroShift でも定義できます。ただし、Red Hat build of 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。

関連情報

- 永続ストレージのアクセスモード

4.2. 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。

4.2.1. 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。

4.2.2. 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、手動でプロビジョニングされる PV に特に当てはまります。超過を最小限に抑えるために、Red Hat build of 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。

4.2.3. 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 を必要な期間保持することができます。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。

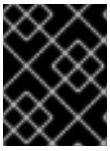
4.2.4. 永続ボリュームの解放

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。

4.2.5. 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について指示します。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、Red Hat build of 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、AWS 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。

4.2.6. 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。

4.2.7. 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一覧表示します。

```
$ 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。

4.3. 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。

4.3.1. 容量

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。

4.3.2. サポートされているアクセスモード

LVMS は、Red Hat build of 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。

4.3.3. 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>
```

4.3.3.1. 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: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  mountOptions: 1
```

```
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

1 指定のマウントオプションは、PV がディスクにマウントされている時に使用されます。

以下の PV タイプがマウントオプションをサポートします。

- AWS Elastic Block Store (EBS)
- Azure Disk
- Azure File
- Cinder
- GCE Persistent Disk
- iSCSI
- ローカルボリューム
- NFS
- Red Hat OpenShift Data Foundation (Ceph RBD のみ)
- VMware vSphere

注記

ファイバーチャネルおよび HostPath PV はマウントオプションをサポートしません。

4.4. 永続ボリューム要求 (PVC)

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

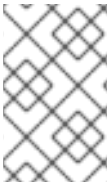
3 PVC に利用できるストレージの量

4 要求で必要になる **StorageClass** の名前

4.4.1. 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。

4.4.2. 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。

4.4.3. 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。

4.4.4. 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、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

4.5. 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、Red Hat build of 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、Red Hat build of MicroShift がボリュームの所有権とパーミッション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 RED HAT BUILD OF MICROSHIFT の永続ボリュームの拡張

Red Hat build of MicroShift で永続ボリュームを拡張する方法を学習します。

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。Red Hat build of 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 Red Hat build of MicroShift の管理者は Persistent Volume Claim (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

Red Hat build of MicroShift は、論理ボリュームマネージャーストレージ (LVMS) コンテナーストレージインターフェイス (CSI) プロバイダーですぐに使用できる動的ストレージプロビジョニングを可能にします。LVMS プラグインは、Kubernetes の LVM ボリュームを管理するための 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論理ボリューム管理 (LVM) 論理ボリューム (LV) 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

Red Hat build of 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

LVMS のデフォルトの統合では、デフォルトのボリュームグループ (VG) が動的に選択されます。Red Hat build of 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。

Red Hat build of MicroShift ホストの Red Hat ビルドに VG が 1 つだけある場合は、その VG が使用されます。複数のボリュームグループがある場合は、グループ **microshift** が使用されます。**microshift** グループが見つからない場合、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。

起動する前に、**lvmd.yaml** 設定ファイルで、ワークロードストレージに十分な容量を持つノード上の既存のボリュームグループを指定する必要があります。ボリュームグループが存在しないと、ノードコントローラーは起動に失敗し、**CrashLoopBackoff** 状態に入ります。

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

Red Hat build of 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. LVMS 設定ファイルの作成

Red Hat build of 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
```

6.4. LVMS の設定

Red Hat build of 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、Red Hat build of MicroShift を再起動する必要があります。

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: 3
    volume-group: 4
    spare-gb: 5
    default: 6
  - name: hdd
    volume-group: hdd-vg
    spare-gb: 10
  - name: striped
    volume-group: multi-pv-vg
    spare-gb: 10
    stripe: 7
    stripe-size: 8
  - name: raid
    volume-group: raid-vg
    lvcreate-options: 9
      - --type=raid1
```

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/topolvm/lvmd.sock** です。
2. **map[string]DeviceClass.device-class** 設定。
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 Unit64。ボリュームグループに未割り当てのままにする GiB 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。

7 Unit。論理ボリューム内のストライプ数。

8 文字列。次のデバイスに移動する前にデバイスに書き込まれるデータの量です。

9 文字列。**[--type=raid1"**]. など、pas **lvcreate** への追加の引数。

警告

複数の PVC が同時に作成されると、競合状態により、LVMS は割り当てられた領域を正確に追跡し、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。

ストライピングは、専用オプション (**stripe** および **stripe-size**) および **lvcreate-options** を使用して設定できます。どちらのオプションも使用できますが、同時に使用することはできません。**lvcreate-options** で **stripe** と **stripe-size** を使用すると、**lvcreate** への引数が重複します。**lvcreate-options: ["--stripes=n"]** と **stripe: n** を同時に設定しないでください。ただし、ストライピングに **lvcreate-options** を使用しない場合は、両方を使用できます。以下に例を示します。

```
stripe: 2
lvcreate-options: ["--mirrors=1"]
```

6.5. 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
```