



OpenShift Container Platform 4.3

ストレージ

OpenShift Container Platform でのストレージの設定および管理

OpenShift Container Platform 4.3 ストレージ

OpenShift Container Platform でのストレージの設定および管理

法律上の通知

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、各種のストレージのバックエンドから永続ボリュームを設定し、Pod からの動的な割り当てを管理する方法について説明します。

目次

第1章 一時ストレージについて	3
1.1. 概要	3
1.2. 一時ストレージのタイプ	3
1.3. 一時ストレージ管理	3
1.4. 一時ストレージのモニタリング	4
第2章 永続ストレージについて	5
2.1. 永続ストレージの概要	5
2.2. ボリュームおよび要求のライフサイクル	5
2.3. 永続ボリューム	8
2.4. PERSISTENT VOLUME CLAIM (永続ボリューム要求、PVC)	13
2.5. ブロックボリュームのサポート	15
第3章 永続ストレージの設定	19
3.1. AWS ELASTIC FILE SYSTEM を使用した永続ストレージ	19
3.2. AWS ELASTIC BLOCK STORE を使用した永続ストレージ	25
3.3. AZURE を使用した永続ストレージ	27
3.4. AZURE FILE を使用した永続ストレージ	28
3.5. CINDER を使用した永続ストレージ	31
3.6. CONTAINER STORAGE INTERFACE (CSI) を使用した永続ストレージ	34
3.7. ファイバーチャネルを使用した永続ストレージ	37
3.8. FLEXVOLUME を使用した永続ストレージ	38
3.9. GCE PERSISTENT DISK を使用した永続ストレージ	42
3.10. HOSTPATH を使用した永続ストレージ	44
3.11. ISCSI を使用した永続ストレージ	46
3.12. ローカルボリュームを使用した永続ストレージ	49
3.13. NFS を使用した永続ストレージ	59
3.14. RED HAT OPENSIFT CONTAINER STORAGE	66
3.15. VMWARE VSPHERE ボリュームを使用した永続ストレージ	67
第4章 永続ボリュームの拡張	72
4.1. ボリューム拡張サポートの有効化	72
4.2. CSI ボリュームの拡張	72
4.3. サポートされているドライバーでの FLEXVOLUME の拡張	72
4.4. ファイルシステムを使用した PERSISTENT VOLUME CLAIM (永続ボリューム要求、PVC) の拡張	73
4.5. ボリューム拡張時の障害からの復旧	74
第5章 動的プロビジョニング	76
5.1. 動的プロビジョニングについて	76
5.2. 利用可能な動的プロビジョニングプラグイン	76
5.3. STORAGECLASS の定義	77
5.4. デフォルト STORAGECLASS の変更	83

第1章 一時ストレージについて

1.1. 概要

永続ストレージに加え、Pod とコンテナは、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、スクラッチスペース、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 は利用可能なローカルストレージのサイズを認識しない。
- 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は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いっぱい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 は入れない。

一時ストレージは、永続ボリュームとは異なり、体系化されておらず、システム、コンテナランタイム、OpenShift Container Platform での他の用途に加え、ノードで実行中のすべての Pod 間で領域を共有します。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。またこれにより、OpenShift Container Platform は該当する場合に Pod をスケジュールし、ローカルストレージの過剰な使用に対してノードを保護することができます。

一時ストレージフレームワークでは、管理者および開発者がこのローカルストレージの管理を改善できますが、I/O スループットやレイテンシーに関する確約はありません。

1.2. 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、コンテナログ、イメージ階層、コンテナ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。OpenShift Container Platform 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します。このパーティションには、イメージ階層と書き込み可能な階層が含まれます。ランタイムパーティションが存在する場合は、**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な階層も含まれません。

1.3. 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。

1.4. 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モニターすることができます。`/var/lib/kubelet` のみを使用できる領域は、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2章 永続ストレージについて

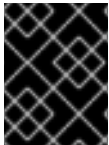
2.1. 永続ストレージの概要

ストレージの管理は、コンピュータリソースの管理とは異なります。OpenShift Container Platform 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、Persistent Volume Claim (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。

PVC はプロジェクトに固有のもので、開発者が PV を使用する手段として作成し、使用します。PV リソース自体のスコープはいずれの単一プロジェクトにも設定されず、それらは OpenShift Container Platform クラスター全体で共有でき、すべてのプロジェクトから要求できます。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 (バインディングプロジェクトの namespace) にスコープ設定する作用があります。

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、または 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、NFS、iSCSI、またはクラウドプロバイダー固有のストレージシステムのいずれの場合でも、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。

PVC は、開発者によるストレージの要求を表す **PersistentVolumeClaim** API オブジェクトによって定義されます。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。たとえば、それらは読み取り/書き込みで 1 回、読み取り専用で複数回マウントできます。

2.2. 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。

2.2.1. 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。

または、クラスター管理者は、使用可能な実際のストレージの詳細を保持する多数の PV を前もって作成できます。PV は API に存在し、利用可能な状態になります。

2.2.2. 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 がない場合には、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 にとくに当てはまります。超過を最小限にするために、OpenShift Container Platform は他のすべての条件に一致する最小の 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に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。

2.2.3. 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、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、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 のスケジューリングおよび要求された 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。

2.2.4. 使用中のストレージオブジェクトの保護

使用中のストレージオブジェクトの保護機能を使用すると、Pod または PVC にバインドされる PV によってアクティブに使用されている PVC がシステムから削除されないようにすることができます。これらが削除されると、データが失われる可能性があります。

使用中のストレージオブジェクトの保護はデフォルトで有効にされています。



注記

PVC は、PVC を使用する Pod オブジェクトが存在する場合に Pod によってアクティブに使用されます。

ユーザーが Pod によってアクティブに使用されている PVC を削除する場合でも、PVC はすぐに削除されません。PVC の削除は、PVC が Pod によってアクティブに使用されなくなるまで延期されます。また、クラスター管理者が PVC にバインドされる PV を削除しても、PV はすぐに削除されません。PV の削除は、PV が PVC にバインドされなくなるまで延期されます。

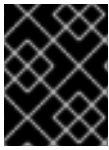
2.2.5. PersistentVolume のリリース

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ので、ポリシーに基づいて処理される必要があります。

2.2.6. PersistentVolume の回収ポリシー

PersistentVolume 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 回収ポリシーは OpenShift Container Platform 4 では非推奨となっています。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、OpenShift Container Platform 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。

2.2.7. PersistentVolume の手動回収

PersistentVolumeClaim (PVC) が削除されても、PersistentVolume (PV) は依然として存在し、「released」とみなされます。ただし、PV は、以前の要求側のデータがボリューム上に残るため、別の要求には利用できません。

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-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セットを削除します。または、同じストレージセットを再利用するには、ストレージ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。

2.2.8. PersistentVolume の回収ポリシーの変更

PersistentVolume 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 PersistentVolume 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				

- PersistentVolume 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"}}'
```

- 選択した PersistentVolume 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** を削除した場合、ボリュームは自動的に削除されません。

2.3. 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。

PV オブジェクト定義例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 5Gi ❷
  accessModes:
    - ReadWriteOnce ❸
  persistentVolumeReclaimPolicy: Retain ❹
  ...
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。

2.3.1. PV の種類

OpenShift Container Platform は以下の **PersistentVolume** プラグインをサポートします。

- AWS Elastic Block Store (EBS)
- Azure Disk
- Azure File
- Cinder
- ファイバーチャネル
- GCE Persistent Disk
- HostPath
- iSCSI
- ローカルボリューム
- NFS
- OpenStack Manila
- Red Hat OpenShift Container Storage
- VMware vSphere

2.3.2. 容量

通常、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。

2.3.3. アクセスモード

PersistentVolume はリソースプロバイダーでサポートされるすべての方法でホストにマウントできます。プロバイダーには各種の機能があり、それぞれの PV のアクセスモードは特定のボリュームでサポートされる特定のモードに設定されます。たとえば、NFS は複数の読み取り/書き込みクライアントをサポートしますが、特定の NFS PV は読み取り専用としてサーバー上でエクスポートされる可能性があります。それぞれの PV は、その特定の PV の機能について記述するアクセスモードの独自のセットを取得します。

要求は、同様のアクセスモードのボリュームに一致します。一致する条件はアクセスモードとサイズの2つの条件のみです。要求のアクセスモードは要求 (request) を表します。そのため、より多くのアクセスを付与することはできますが、アクセスを少なくすることはできません。たとえば、要求により RWO が要求されるものの、利用できる唯一のボリュームが NFS PV (RWO+ROX+RWX) の場合に、要求は RWO をサポートする NFS に一致します。

直接的なマッチングが常に最初に試行されます。ボリュームのモードは、要求モードと一致するか、要求した内容以上のものを含む必要があります。サイズは予想されるものより多いか、またはこれと同等である必要があります。2つのタイプのボリューム (NFS および iSCSI など) のどちらにも同じセットのアクセスモードがある場合、それらのいずれかがそれらのモードを持つ要求に一致する可能性があります。ボリュームのタイプ間で順序付けすることはできず、タイプを選択することはできません。

同じモードのボリュームはすべて分類され、サイズ別 (一番小さいものから一番大きいもの順) に分類されます。バインダーは一致するモードのグループを取得し、1つのサイズが一致するまでそれぞれを (サイズの順序で) 繰り返し処理します。

以下の表では、アクセスモードをまとめています。

表2.1 アクセスモード

アクセスモード	CLI の省略形	説明
ReadWriteOnce	RWO	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。
ReadOnlyMany	ROX	ボリュームは数多くのノードで読み取り専用としてマウントできます。
ReadWriteMany	RWX	ボリュームは数多くのノードで読み取り/書き込みとしてマウントできます。

重要

ボリュームの **AccessModes** は、ボリュームの機能の記述子です。それらは施行されている制約ではありません。ストレージプロバイダーはリソースの無効な使用から生じるランタイムエラーに対応します。

たとえば、NFS は **ReadWriteOnce** アクセスモードを提供します。ボリュームの ROX 機能を使用する必要がある場合は、要求に **read-only** のマークを付ける必要があります。プロバイダーのエラーは、マウントエラーとしてランタイム時に表示されます。

iSCSI およびファイバーチャネルボリュームには現在、フェンシングメカニズムがありません。ボリュームが一度に1つのノードでのみ使用されるようにする必要があります。ノードのドレイン (解放) などの特定の状況では、ボリュームは2つのノードで同時に使用できます。ノードをドレイン (解放) する前に、まずこれらのボリュームを使用する Pod が削除されていることを確認してください。

表2.2 サポート対象の PV 向けアクセスモード

ボリュームプラグイン	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
AWS EBS [2]	■	-	-

ボリュームプラグイン	ReadWriteOnce [1]	ReadOnlyMany	ReadWriteMany
Azure File	■	■	■
Azure Disk	■	-	-
Cinder	■	-	-
ファイバーチャネル	■	■	-
GCE Persistent Disk	■	-	-
HostPath	■	-	-
iSCSI	■	■	-
ローカルボリューム	■	-	-
NFS	■	■	■
OpenStack Manila	-	-	■
Red Hat OpenShift Container Storage	■	-	■
VMware vSphere	■	-	-

1. ReadWriteOnce (RWO) ボリュームは複数のノードにマウントできません。ノードに障害が発生すると、システムは、すでに障害が発生しているノードに割り当てられているため、割り当てられた RWO ボリュームを新規ノードにマウントすることはできません。その結果、複数接続のエラーメッセージが表示された場合には、障害が発生したノードを回復するか、または削除して、ボリュームを他のノードで利用できるようにすることができます。
2. AWS EBS に依存する Pod の再作成デプロイメントストラテジーを使用します。

2.3.4. 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2.3 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	要求が検出されていますが、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>
```

2.3.4.1. マウントオプション

アノテーション **volume.beta.kubernetes.io/mount-options** 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。

以下は例になります。

マウントオプションの例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
  annotations:
    volume.beta.kubernetes.io/mount-options: rw,nfsvers=4,noexec ❶
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

❶ 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 Container Storage (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。

2.4. PERSISTENT VOLUME CLAIM (永続ボリューム要求、PVC)

各 Persistent Volume Claim (永続ボリューム要求、PVC) には、要求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。以下に例を示します。

PVC オブジェクト定義例

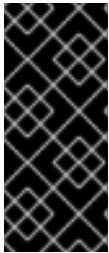
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim ❶
spec:
  accessModes:
    - ReadWriteOnce ❷
  resources:
    requests:
      storage: 8Gi ❸
  storageClassName: gold ❹
status:
  ...
```

- ❶ PVC の名前
- ❷ 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
- ❸ PVC に利用できるストレージの量
- ❹ 要求で必要になる **StorageClass** の名前

2.4.1. 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。



重要

ClusterStorageOperator は、使用されるプラットフォームによってデフォルトの StorageClass をインストールする可能性があります。この StorageClass は Operator によって所有され、制御されます。アノテーションとラベルを定義するほかは、これを削除したり、変更したりすることはできません。異なる動作が必要な場合は、カスタム StorageClass を定義する必要があります。

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 StorageClass 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 StorageClass のみをデフォルトとして設定する必要があります。

2.4.2. 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。

2.4.3. 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、これ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。

2.4.4. 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" ❶
          name: mypd ❷
  volumes:
```

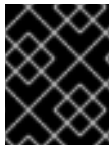
```
- name: mypd
  persistentVolumeClaim:
    claimName: myclaim ❸
```

- ❶ Pod 内にボリュームをマウントするためのパス
- ❷ マウントするボリュームの名前
- ❸ 使用する同じ namespace にある PVC の名前

2.5. ブロックボリュームのサポート

OpenShift Container Platform は、raw ブロックボリュームを静的にプロビジョニングできます。これらのボリュームにはファイルシステムがなく、ディスクに直接書き込むアプリケーションや、独自のストレージサービスを実装するアプリケーションにはパフォーマンス上の利点があります。

raw ブロックボリュームは、PV および PVC 仕様で **volumeMode: Block** を指定してプロビジョニングされます。



重要

raw ブロックボリュームを使用する Pod は、特権付きコンテナを許可するように設定する必要があります。

以下の表は、ブロックボリュームをサポートするボリュームプラグインを表示しています。

表2.4 ブロックボリュームのサポート

ボリュームプラグイン	手動のプロビジョニング	動的なプロビジョニング	完全対応
AWS EBS	■	■	■
Azure Disk	■	■	■
Azure File			
Cinder	■	■	
ファイバーチャネル	■		
GCP	■	■	■
HostPath			
iSCSI	■		■
ローカルボリューム	■		■
NFS			

ボリュームプラグイン	手動のプロビジョニング	動的なプロビジョニング	完全対応
Red Hat OpenShift Container Storage	■	■	■
VMware vSphere	■	■	■



注記

手動でプロビジョニングできるものの、完全にサポートされていないブロックボリュームはいずれも、テクノロジープレビューとしてのみ提供されます。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

2.5.1. ブロックボリュームの例

PV の例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block 1
  persistentVolumeReclaimPolicy: Retain
  fc:
    targetWWNs: ["50060e801049cfd1"]
    lun: 0
    readOnly: false
```

1 **volumeMode** を **Block** に設定して、この PV が raw ブロックボリュームであることを示します。

PVC の例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
```

```

volumeMode: Block ❶
resources:
  requests:
    storage: 10Gi

```

- ❶ **volumeMode** を **Block** に設定して、raw ブロック PVC が要求されていることを示します。

Pod 仕様の例

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-with-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: [ "tail -f /dev/null" ]
      volumeDevices: ❶
        - name: data
          devicePath: /dev/xvda ❷
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc ❸

```

- ❶ **volumeMounts** ではなく **volumeDevices** がブロックデバイスに使用されます。**PersistentVolumeClaim** ソースのみを raw ブロックボリュームと共に使用できます。
- ❷ **mountPath** ではなく **devicePath** が raw ブロックがシステムにマップされる物理デバイスへのパスを表します。
- ❸ ボリュームソースのタイプは **persistentVolumeClaim** であり、予想通りに PVC の名前に一致する必要があります。

表2.5 VolumeMode の許容値

値	デフォルト
Filesystem	Yes
Block	No

表2.6 ブロックボリュームのバインディングシナリオ

PV VolumeMode	PVC VolumeMode	バインディングの結果
Filesystem	Filesystem	バインド

PV VolumeMode	PVC VolumeMode	バインディングの結果
Unspecified	Unspecified	バインド
Filesystem	Unspecified	バインド
Unspecified	Filesystem	バインド
Block	Block	バインド
Unspecified	Block	バインドなし
Block	Unspecified	バインドなし
Filesystem	Block	バインドなし
Block	Filesystem	バインドなし



重要

値を指定しないと、**Filesystem** のデフォルト値が指定されます。

第3章 永続ストレージの設定

3.1. AWS ELASTIC FILE SYSTEM を使用した永続ストレージ

OpenShift Container Platform では、Amazon Web Services (AWS) Elastic File System ボリューム (EFS) を使用できます。AWS EC2 を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes および AWS についてのある程度の理解があることが前提となります。

重要

Elastic File System はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。AWS Elastic Block Store ボリュームは動的にプロビジョニングできます。永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。PersistentVolumeClaim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。

3.1.1. 前提条件

- EFS ボリュームのセキュリティーグループからのインバウンド NFS トラフィックを許可するように AWS セキュリティーグループを設定します。
- AWS EFS ボリュームを、任意のホストからの着信 SSH トラフィックを許可するように設定します。

追加の参考資料

- [Amazon EFS](#)
- [EFS の Amazon セキュリティーグループ](#)

3.1.2. EFS 変数を ConfigMap に保存します。

ConfigMap を使用して、EFS プロビジョナーに必要なすべての環境変数を含めることが推奨されます。

手順

1. 以下の内容が含まれる **configmap.yaml** ファイルを作成して、環境変数を含む OpenShift Container Platform ConfigMap を定義します。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: efs-provisioner
data:
  file.system.id: <file-system-id> ❶
  aws.region: <aws-region> ❷
  provisioner.name: openshift.org/aws-efs ❸
  dns.name: "" ❹

```

- ❶ Amazon Web Services (AWS) EFS ファイルシステム ID を定義します。
- ❷ EFS ファイルシステムの AWS リージョン (例: **us-east-1**)。
- ❸ 関連付けられた StorageClass のプロビジョナーの名前。
- ❹ EFS ボリュームが置かれる新規 DNS 名を指定するオプションの引数。DNS 名が指定されていない場合、プロビジョナーは **<file-system-id>.efs.<aws-region>.amazonaws.com** で EFS ボリュームを検索します。

2. ファイルを設定したら、以下のコマンドを実行してクラスター内にこのファイルを作成します。

```
$ oc create -f configmap.yaml -n <namespace>
```

3.1.3. EFS ボリュームの認可の設定

EFS プロビジョナーは、OpenShift Container Platform ストレージリソースを確認し、更新することに加えて、AWS エンドポイントと通信することができる必要があります。以下の手順では、EFS プロビジョナーに必要なパーミッションを作成します。

手順

1. **efs-provisioner** サービスアカウントを作成します。

```
$ oc create serviceaccount efs-provisioner
```

2. 必要なパーミッションを定義する **clusterrole.yaml** ファイルを作成します。

```

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: efs-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]

```



```
- apiGroups: [""]
  resources: ["events"]
  verbs: ["create", "update", "patch"]
- apiGroups: ["security.openshift.io"]
  resources: ["securitycontextconstraints"]
  verbs: ["use"]
  resourceNames: ["hostmount-anyuid"]
```

3. 定義されたロールをサービスアカウントに割り当てるクラスターのロールバインディングを定義する **clusterrolebinding.yaml** ファイルを作成します。

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-efs-provisioner
subjects:
- kind: ServiceAccount
  name: efs-provisioner
  namespace: default ❶
roleRef:
  kind: ClusterRole
  name: efs-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
```

- ❶ EFS プロビジョナー Pod が実行される namespace。EFS プロビジョナーが **default**以外の namespace で実行されている場合、この値は更新する必要があります。

4. 必要なパーミッションを持つロールを定義する **role.yaml** ファイルを作成します。

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-efs-provisioner
rules:
- apiGroups: [""]
  resources: ["endpoints"]
  verbs: ["get", "list", "watch", "create", "update", "patch"]
```

5. このロールをサービスアカウントに割り当てるロールバインディングを定義する **rolebinding.yaml** ファイルを作成します。

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-efs-provisioner
subjects:
- kind: ServiceAccount
  name: efs-provisioner
  namespace: default ❶
roleRef:
  kind: Role
  name: leader-locking-efs-provisioner
  apiGroup: rbac.authorization.k8s.io
```

- 1 EFS プロビジョナー Pod が実行される namespace。EFS プロビジョナーが **default**以外の namespace で実行されている場合、この値は更新する必要があります。

6. OpenShift Container Platform クラスター内にリソースを作成します。

```
$ oc create -f clusterrole.yaml,clusterrolebinding.yaml,role.yaml,rolebinding.yaml
```

3.1.4. EFS StorageClass の作成

PersistentVolumeClaim を作成する前に、StorageClass が OpenShift Container Platform クラスターに存在する必要があります。以下の手順では、EFS プロビジョナーの StorageClass を作成します。

手順

1. 以下の内容を含む **storageclass.yaml** を作成して、環境変数を含む OpenShift Container Platform ConfigMap を定義します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aws-efs
provisioner: openshift.org/aws-efs
parameters:
  gidMin: "2048" ①
  gidMax: "2147483647" ②
  gidAllocate: "true" ③
```

- ① ボリュームの割り当てに使用する最小グループ ID (GID) を定義するオプションの引数。デフォルト値は **2048** です。
- ② ボリューム割り当てに使用する最大の GID を定義するオプションの引数。デフォルト値は **2147483647** です。
- ③ GID がボリュームに割り当てられているかどうかを判別するオプションの引数。**false** の場合、動的にプロビジョニングされるボリュームには GID が割り当てられません。これにより、すべてのユーザーによる作成されたボリュームの読み取りおよび書き込みが可能になります。デフォルト値は **true** です。

2. ファイルを設定したら、以下のコマンドを実行してクラスター内にこのファイルを作成します。

```
$ oc create -f storageclass.yaml
```

3.1.5. EFS プロビジョナーの作成

EFS プロビジョナーは、EFS ボリュームを NFS 共有としてマウントする OpenShift Container Platform Pod です。

前提条件

- EFS 環境変数を定義する ConfigMap を作成します。

- 必要なクラスターおよびロールパーミッションを含むサービスアカウントを作成します。
- ボリュームをプロビジョニングするための StorageClass を作成します。
- Amazon Web Services (AWS) セキュリティーグループを、すべての OpenShift Container Platform ノード上での着信 NFS トラフィックを許可するように設定します。
- AWS EFS ボリュームセキュリティグループを、すべてのソースからの着信 SSH トラフィックを許可するように設定します。

手順

1. 以下の内容を含む **provisioner.yaml** を作成し、EFS プロビジョナーを定義します。

```
kind: Pod
apiVersion: v1
metadata:
  name: efs-provisioner
spec:
  serviceAccount: efs-provisioner
  containers:
    - name: efs-provisioner
      image: quay.io/external_storage/efs-provisioner:latest
      env:
        - name: PROVISIONER_NAME
          valueFrom:
            configMapKeyRef:
              name: efs-provisioner
              key: provisioner.name
        - name: FILE_SYSTEM_ID
          valueFrom:
            configMapKeyRef:
              name: efs-provisioner
              key: file.system.id
        - name: AWS_REGION
          valueFrom:
            configMapKeyRef:
              name: efs-provisioner
              key: aws.region
        - name: DNS_NAME
          valueFrom:
            configMapKeyRef:
              name: efs-provisioner
              key: dns.name
          optional: true
      volumeMounts:
        - name: pv-volume
          mountPath: /persistentvolumes
  volumes:
    - name: pv-volume
      nfs:
        server: <file-system-id>.efs.<region>.amazonaws.com ❶
        path: / ❷
```

- ❶ EFS ボリュームの DNS 名が含まれます。このフィールドは、Pod が EFS ボリュームを検出できるように更新される必要があります。

- 2 EFS ボリュームのマウントパス。それぞれの永続ボリュームは EFS ボリューム上に別々のサブディレクトリーとして作成されます。この EFS ボリュームが OpenShift Container

2. ファイルを設定したら、以下のコマンドを実行してクラスター内にこのファイルを作成します。

```
$ oc create -f provisioner.yaml
```

3.1.6. EFS PersistentVolumeClaim の作成

EFS PersistentVolumeClaim は、Pod が基礎となる EFS ストレージをマウントできるように作成されます。

前提条件

- EFS プロビジョナー Pod を作成します。

手順 (UI)

1. OpenShift Container Platform コンソールで、**Storage → Persistent Volume Claims** をクリックします。
2. Persistent Volume Claim (永続ボリューム要求、PVC) の概要で、**Create Persistent Volume Claim** をクリックします。
3. 結果のページで必要なオプションを定義します。
 - a. 一覧から作成したストレージクラスを選択します。
 - b. ストレージ要求の一意の名前を入力します。
 - c. アクセスモードを選択し、作成されるストレージ要求の読み取り/書き込みアクセスを決定します。
 - d. ストレージ要求のサイズを定義します。



注記

サイズを入力する必要がありますが、EFS ボリュームにアクセスするすべての Pod には無制限のストレージがあります。**1Mi** などの値を定義します。これにより、ストレージサイズは無制限になります。

4. **Create** をクリックして Persistent Volume Claim (永続ボリューム要求、PVC) を作成し、永続ボリュームを生成します。

手順 (CLI)

1. または、以下の内容でファイル **pvc.yaml** を作成して EFS の PersistentVolumeClaim を定義することができます。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
```

```

name: efs-claim ❶
namespace: test-efs
annotations:
  volume.beta.kubernetes.io/storage-provisioner: openshift.org/aws-efs
finalizers:
  - kubernetes.io/pvc-protection
spec:
  accessModes:
    - ReadWriteOnce ❷
  resources:
    requests:
      storage: 5Gi ❸
  storageClassName: aws-efs ❹
  volumeMode: Filesystem

```

- ❶ PVC の一意の名前。
- ❷ 作成された PVC の読み取りおよび書き込みアクセスを判別するためのアクセスモード。
- ❸ PVC のサイズを定義します。
- ❹ EFS プロビジョナーの StorageClass の名前。

2. ファイルを設定したら、以下のコマンドを実行してクラスター内にこのファイルを作成します。

```
$ oc create -f pvc.yaml
```

3.2. AWS ELASTIC BLOCK STORE を使用した永続ストレージ

OpenShift Container Platform は AWS Elastic Block Store volumes (EBS) をサポートします。AWS EC2 を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes および AWS についてのある程度の理解があることが前提となります。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。AWS Elastic Block Store ボリュームは動的にプロビジョニングできます。永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。Persistent Volume Claim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。



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。

追加の参考資料

- [Amazon EC2](#)

3.2.1. EBS ストレージクラスの作成

ストレージクラスを使用すると、ストレージのレベルや使用状況を区別し、記述することができます。ストレージクラスを定義することにより、ユーザーは動的にプロビジョニングされた永続ボリュームを取得できます。

手順

1. OpenShift Container Platform コンソールで、**Storage → Storage Classes** をクリックします。
2. ストレージクラスの概要では、**Create Storage Class** をクリックします。
3. 表示されるページで必要なオプションを定義します。
 - a. ストレージクラスを参照するための名前を入力します。
 - b. オプションの説明を入力します。
 - c. 回収ポリシーを選択します。
 - d. ドロップダウンリストから **kubernetes.io/aws-ebs** を選択します。
 - e. 必要に応じてストレージクラスの追加パラメーターを入力します。
4. **Create** をクリックしてストレージクラスを作成します。

3.2.2. Persistent Volume Claim (永続ボリューム要求、PVC) の作成

前提条件

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

手順

1. OpenShift Container Platform コンソールで、**Storage → Persistent Volume Claims** をクリックします。
2. Persistent Volume Claim (永続ボリューム要求、PVC) の概要で、**Create Persistent Volume Claim** をクリックします。
3. 表示されるページで必要なオプションを定義します。
 - a. ドロップダウンメニューから以前に作成されたストレージクラスを選択します。
 - b. ストレージ要求の一意の名前を入力します。
 - c. アクセスモードを選択します。これにより、作成されたストレージ要求の読み取り/書き込みアクセスが決定されます。
 - d. ストレージ要求のサイズを定義します。
4. **Create** をクリックして Persistent Volume Claim (永続ボリューム要求、PVC) を作成し、永続ボリュームを生成します。

3.2.3. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。

デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない AWS ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

3.2.4. ノード上の EBS ボリュームの最大数

OpenShift Container Platform では、デフォルトで1つのノードに最大 39 の EBS ボリュームを割り当てることができます。この制限は、[AWS ボリュームの制限](#) に合致します。ボリュームの制限は、インスタンスのタイプによって異なります。

3.3. AZURE を使用した永続ストレージ

OpenShift Container Platform では、Microsoft Azure Disk ボリュームがサポートされます。Azure を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と Azure についてのある程度の理解があることが前提となります。Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。Azure Disk ボリュームは動的にプロビジョニングできます。永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。Persistent Volume Claim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。



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。

追加の参考資料

- [Microsoft Azure Disk](#)

3.3.1. Azure ストレージクラスの作成

ストレージクラスを使用すると、ストレージのレベルや使用状況を区別し、記述することができます。ストレージクラスを定義することにより、ユーザーは動的にプロビジョニングされた永続ボリュームを取得できます。

追加の参考資料

- [Azure Disk Storage Class](#)

手順

1. OpenShift Container Platform コンソールで、**Storage → Storage Classes** をクリックします。
2. ストレージクラスの概要では、**Create Storage Class** をクリックします。
3. 表示されるページで必要なオプションを定義します。
 - a. ストレージクラスを参照するための名前を入力します。
 - b. オプションの説明を入力します。

- c. 回収ポリシーを選択します。
 - d. ドロップダウンリストから **kubernetes.io/azure-disk** を選択します。
 - i. ストレージアカウントのタイプを入力します。これは、Azure ストレージアカウントの SKU の層に対応します。有効なオプションは、**Premium_LRS**、**Standard_LRS**、**StandardSSD_LRS**、および **UltraSSD_LRS** です。
 - ii. アカウントの種類を入力します。有効なオプションは **shared**、**dedicated** および **managed** です。
 - e. 必要に応じてストレージクラスの追加パラメーターを入力します。
4. **Create** をクリックしてストレージクラスを作成します。

3.3.2. Persistent Volume Claim (永続ボリューム要求、PVC) の作成

前提条件

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

手順

1. OpenShift Container Platform コンソールで、**Storage → Persistent Volume Claims** をクリックします。
2. Persistent Volume Claim (永続ボリューム要求、PVC) の概要で、**Create Persistent Volume Claim** をクリックします。
3. 表示されるページで必要なオプションを定義します。
 - a. ドロップダウンメニューから以前に作成されたストレージクラスを選択します。
 - b. ストレージ要求の一意の名前を入力します。
 - c. アクセスモードを選択します。これにより、作成されたストレージ要求の読み取り/書き込みアクセスが決定されます。
 - d. ストレージ要求のサイズを定義します。
4. **Create** をクリックして Persistent Volume Claim (永続ボリューム要求、PVC) を作成し、永続ボリュームを生成します。

3.3.3. ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

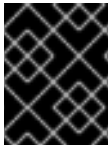
これにより、OpenShift Container Platform がフォーマットされていない Azure ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

3.4. AZURE FILE を使用した永続ストレージ

OpenShift Container Platform では、Microsoft Azure File ボリュームがサポートされます。Azure を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と Azure についてのある程度の理解があることが前提となります。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。Azure File ボリュームは動的にプロビジョニングできます。

永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。PersistentVolumeClaim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、アプリケーションで使用できるようにユーザーによって要求されます。



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。

追加の参考資料

- [Azure Files](#)

3.4.1. Azure File 共有 PersistentVolumeClaim の作成

PersistentVolumeClaim を作成するには、最初に Azure アカウントおよびキーを含むシークレットを定義する必要があります。このシークレットは PersistentVolume 定義に使用され、アプリケーションでできるように PersistentVolumeClaim によって参照されます。

前提条件

- Azure File 共有があること。
- この共有にアクセスするための認証情報 (とくにストレージアカウントおよびキー) が利用可能であること。

手順

1. Azure File の認証情報が含まれるシークレットを作成します。

```
$ oc create secret generic <secret-name> --from-literal=azurestorageaccountname=
<storage-account> \ ❶
--from-literal=azurestorageaccountkey=<storage-account-key> ❷
```

- ❶ Azure File ストレージアカウントの名前。
- ❷ Azure File ストレージアカウントキー。

2. 作成したシークレットを参照する PersistentVolume を作成します。

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
```

```

name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  storageClassName: azure-file-sc
  azureFile:
    secretName: <secret-name> ❸
    shareName: share-1 ❹
    readOnly: false

```

- ❶ PersistentVolume の名前。
- ❷ この PersistentVolume のサイズ。
- ❸ Azure File 共有の認証情報を含むシークレットの名前。
- ❹ Azure File 共有の名前。

3. 作成した PersistentVolume にマップする PersistentVolumeClaim を作成します。

```

apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "claim1" ❶
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "5Gi" ❷
  storageClassName: azure-file-sc ❸
  volumeName: "pv0001" ❹

```

- ❶ PersistentVolumeClaim の名前。
- ❷ この PersistentVolumeClaim のサイズ。
- ❸ PersistentVolume のプロビジョニングに使用される StorageClass の名前。
PersistentVolume 定義で使用する StorageClass を指定します。
- ❹ Azure File 共有を参照する既存の PersistentVolume の名前。

3.4.2. Azure File 共有の Pod へのマウント

PersistentVolumeClaim の作成後に、これをアプリケーション内で使用できます。以下の例は、この共有を Pod 内にマウントする方法を示しています。

前提条件

- 基礎となる Azure File 共有にマップされる PersistentVolumeClaim があること。

手順

- 既存の PersistentVolumeClaim をマウントする Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name ❶
spec:
  containers:
    ...
    volumeMounts:
      - mountPath: "/data" ❷
        name: azure-file-share
  volumes:
    - name: azure-file-share
      persistentVolumeClaim:
        claimName: claim1 ❸
```

- ❶ Pod の名前。
- ❷ Pod 内に Azure File 共有をマウントするパス。
- ❸ 以前に作成された PersistentVolumeClaim の名前。

3.5. CINDER を使用した永続ストレージ

OpenShift Container Platform は OpenStack Cinder をサポートします。これには、Kubernetes と OpenStack についてある程度の理解があることが前提となります。

Cinder ボリュームは動的にプロビジョニングできます。永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。Persistent Volume Claim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。

追加リソース

- OpenStack Block Storage が仮想ハードドライブの永続ブロックストレージ管理を提供する方法についての詳細は、「[OpenStack Block Storage \(cinder\)](#)」を参照してください。

3.5.1. Cinder を使用した手動プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

前提条件

- Red Hat OpenStack Platform (RHOSP) 用に設定された OpenShift Container Platform
- Cinder ボリューム ID

3.5.1.1. 永続ボリュームの作成

OpenShift Container Platform に永続ボリューム (PV) を作成する前に、オブジェクト定義でこれを定義する必要があります。

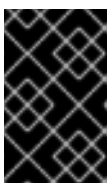
手順

1. オブジェクト定義をファイルに保存します。

cinder-persistentvolume.yaml

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "pv0001" ❶
spec:
  capacity:
    storage: "5Gi" ❷
  accessModes:
    - "ReadWriteOnce"
  cinder: ❸
    fsType: "ext3" ❹
    volumeID: "f37a03aa-6212-4c62-a805-9ce139fab180" ❺
```

- ❶ Persistent Volume Claim (PVC、永続ボリューム要求) または Pod によって使用されるボリュームの名前。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ Red Hat OpenStack Platform (RHOSP) Cinder ボリュームの **cinder** を示します。
- ❹ ボリュームの初回マウント時に作成されるファイルシステム。
- ❺ 使用する Cinder ボリューム



重要

ボリュームをフォーマットしてプロビジョニングした後は、**fstype** パラメーターの値は変更しないでください。この値を変更すると、データの損失や、Pod の障害につながる可能性があります。

2. 前のステップで保存したオブジェクト定義ファイルを作成します。

```
$ oc create -f cinder-persistentvolume.yaml
```

3.5.1.2. 永続ボリュームのフォーマット

OpenShift Container Platform は初回の使用前にフォーマットするため、フォーマットされていない Cinder ボリュームを PV として使用できます。

OpenShift Container Platform がボリュームをマウントし、これをコンテナに渡す前に、システムは PV 定義の **fsType** パラメーターで指定されたファイルシステムがボリュームに含まれるかどうかをチェックします。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

3.5.1.3. Cinder ボリュームのセキュリティー

お使いのアプリケーションで Cinder PV を使用する場合に、そのデプロイメント設定にセキュリティーを追加します。

前提条件

- 適切な **fsGroup** ストラテジーを使用する SCC が作成される必要があります。

手順

1. サービスアカウントを作成して、そのアカウントを SCC に追加します。

```
$ oc create serviceaccount <service_account>
$ oc adm policy add-scc-to-user <new_scc> -z <service_account> -n <project>
```

2. アプリケーションのデプロイ設定で、サービスアカウント名と **securityContext** を指定します。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend-1
spec:
  replicas: 1 ❶
  selector: ❷
    name: frontend
  template: ❸
    metadata:
      labels: ❹
        name: frontend ❺
    spec:
      containers:
        - image: openshift/hello-openshift
          name: helloworld
          ports:
            - containerPort: 8080
              protocol: TCP
          restartPolicy: Always
          serviceAccountName: <service_account> ❻
          securityContext:
            fsGroup: 7777 ❼
```

- ❶ 実行する Pod のコピー数。
- ❷ 実行する Pod のラベルセレクター。
- ❸ コントローラーが作成する Pod のテンプレート。
- ❹ Pod のラベル。ラベルセレクターからのラベルを組み込む必要があります。
- ❺ パラメーター拡張後の名前の最大長さは 63 文字です。
- ❻ 作成したサービスアカウントを指定します。

7 Pod の **fsGroup** を指定します。

3.6. CONTAINER STORAGE INTERFACE (CSI) を使用した永続ストレージ

Container Storage Interface (CSI) により、OpenShift Container Platform は [CSI インターフェース](#) を永続ストレージとして実装するストレージバックエンドからストレージを使用できます。

重要

OpenShift Container Platform には CSI ドライバーが含まれていません。 [コミュニティ](#) または [ストレージベンダー](#) が提供する CSI ドライバーを使用することが推奨されます。

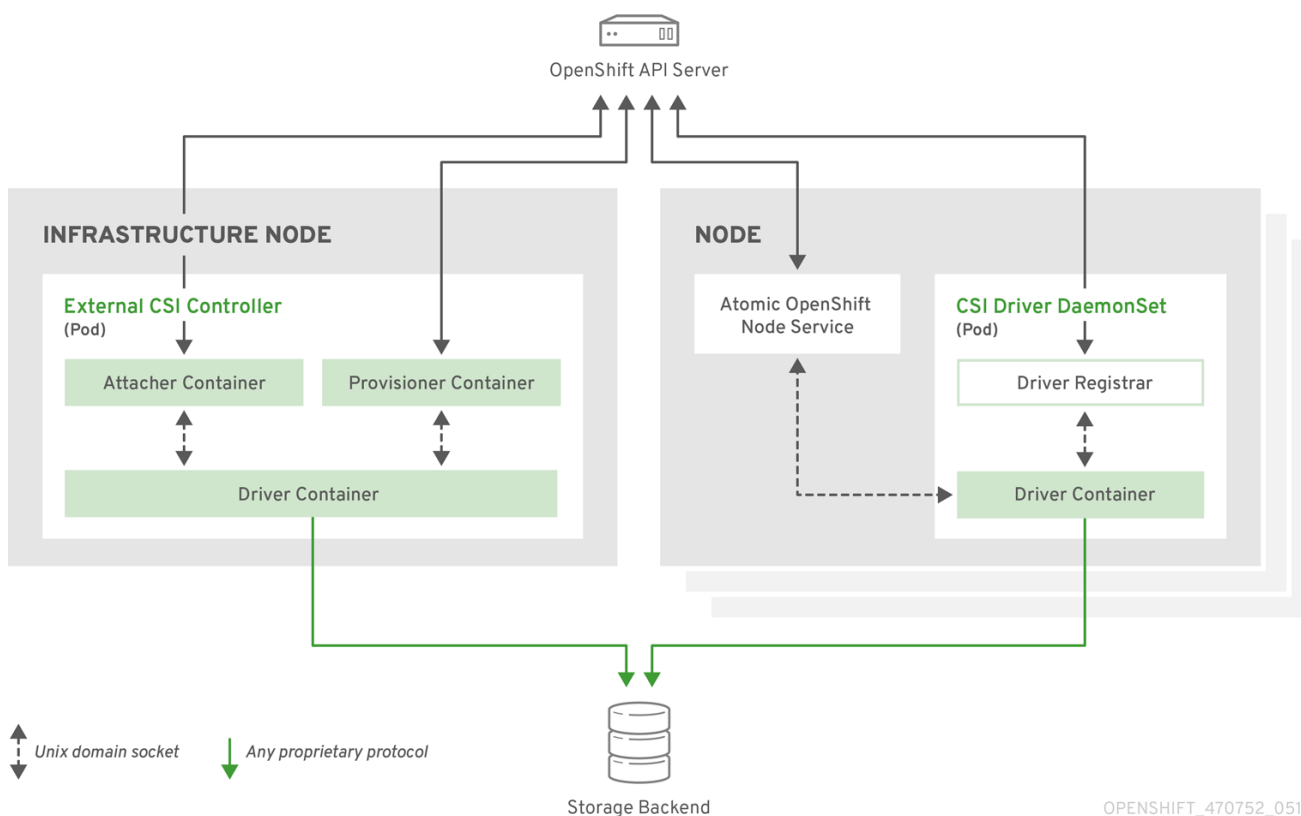
インストールの手順はドライバーによって異なりますが、各ドライバーのドキュメントに記載されています。CSI ドライバーの提供される手順に従います。

OpenShift Container Platform 4.3 は、[CSI 仕様](#) のバージョン 1.1.0 をサポートします。

3.6.1. CSI アーキテクチャー

CSI ドライバーは通常、コンテナイメージとして提供されます。これらのコンテナは、実行先の OpenShift Container Platform を認識しません。OpenShift Container Platform でサポートされる CSI 互換のストレージバックエンドを使用するには、クラスター管理者は、OpenShift Container Platform とストレージドライバーの橋渡しとして機能するコンポーネントを複数デプロイする必要があります。

以下の図では、OpenShift Container Platform クラスターの Pod で実行されるコンポーネントの俯瞰図を示しています。



OPENSIFT_470752_0518

異なるストレージバックエンドに対して複数の CSI ドライバーを実行できます。各ドライバーには、独自の外部コントローラーのデプロイメントおよびドライバーと CSI レジストラーを含む DaemonSet が必要です。

3.6.1.1. 外部の CSI コントローラー

外部の CSI コントローラーは、3 つのコンテナを含む 1 つまたは複数の Pod を配置するデプロイメントです。

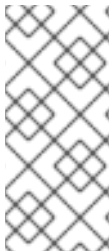
- OpenShift Container Platform からの **attach** および **detach** の呼び出しを適切な CSI ドライバーへの **ControllerPublish** および **ControllerUnpublish** 呼び出しに変換する外部の CSI アタッチャーコンテナ。
- OpenShift Container Platform からの **provision** および **delete** の呼び出しを適切な CSI ドライバーへの **CreateVolume** および **DeleteVolume** 呼び出しに変換する外部の CSI プロビジョナーコンテナ。
- CSI ドライバーコンテナ

CSI アタッチャーおよび CSI プロビジョナーコンテナは、Unix Domain Socket を使用して、CSI ドライバーコンテナと通信し、CSI の通信が Pod 外に出ないようにします。CSI ドライバーは Pod 外からはアクセスできません。



注記

通常、**attach**、**detach**、**provision** および **delete** 操作では、CSI ドライバーがストレージバックエンドに対する認証情報を使用する必要があります。CSI コントローラー Pod をインフラストラクチャーノードで実行し、コンピュータノードで致命的なセキュリティ違反が発生した場合でも認証情報がユーザープロセスに漏洩されないようにします。



注記

外部のアタッチャーは、サードパーティーの **attach** または **detach** 操作をサポートしない CSI ドライバーに対しても実行する必要があります。外部のアタッチャーは、CSI ドライバーに対して **ControllerPublish** または **ControllerUnpublish** 操作を実行します。ただし、必要な OpenShift Container Platform 割り当て API を実装できるように依然として実行する必要があります。

3.6.1.2. CSI ドライバーの DaemonSet

CSI ドライバーの DaemonSet は、OpenShift Container Platform が CSI ドライバーによって提供されるストレージをノードにマウントして、永続ボリューム (PV) としてユーザーワークロード (Pod) で使用できるように、全ノードで Pod を実行します。CSI ドライバーがインストールされた Pod には、以下のコンテナが含まれます。

- ノード上で実行中の **openshift-node** サービスに CSI ドライバーを登録する CSI ドライバーレジストラー。このノードで実行中の **openshift-node** プロセスは、ノードで利用可能な Unix Domain Socket を使用して CSI ドライバーに直接接続します。
- CSI ドライバー

ノードにデプロイされた CSI ドライバーには、ストレージバックエンドへの認証情報をできる限り少なく指定する必要があります。OpenShift Container Platform は、**NodePublish/NodeUnpublish** および **NodeStage/NodeUnstage** (実装されている場合) などの CSI 呼び出しのノードプラグインセットのみを使用します。

3.6.2. 動的プロビジョニング

永続ストレージの動的プロビジョニングは、CSI ドライバーおよび基礎となるストレージバックエンドの機能により異なります。CSI ドライバーのプロバイダーは、OpenShift Container Platform での StorageClass の作成方法および設定に利用でるパラメーターについての文書を作成する必要があります。

作成された StorageClass は、動的プロビジョニングを有効にするために設定できます。

手順

- デフォルトのストレージクラスを作成します。これにより、特殊なストレージクラスを必要としないすべての PVC がインストールされた CSI ドライバーでプロビジョニングされます。

```
# oc create -f - << EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class> ❶
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: <provisioner-name> ❷
parameters:
EOF
```

- ❶ 作成される StorageClass の名前。
- ❷ インストールされている CSI ドライバーの名前。

3.6.3. CSI ドライバーの使用例

以下の例では、テンプレートを変更せずにデフォルトの MySQL テンプレートをインストールします。

前提条件

- CSI ドライバーがデプロイされている。
- 動的プロビジョニング用に StorageClass が作成されている。

手順

- MySQL テンプレートを作成します。

```
# oc new-app mysql-persistent
--> Deploying template "openshift/mysql-persistent" to project default
...

# oc get pvc
NAME          STATUS  VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS  AGE
mysql         Bound       kubernetes-dynamic-pv-3271ffcb4e1811e8  1Gi
RWO           cinder       3s
```


3.7. ファイバーチャネルを使用した永続ストレージ

OpenShift Container Platform ではファイバーチャネルがサポートされており、ファイバーチャネルボリュームを使用して OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と Fibre Channel についてある程度の理解があることが前提となります。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。PersistentVolumeClaim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。



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。

追加の参考資料

- [ファイバーチャネル](#)

3.7.1. プロビジョニング

PersistentVolume API を使用してファイバーチャネルボリュームをプロビジョニングするには、以下が利用可能でなければなりません。

- **targetWWN** (ファイバーチャネルターゲットのワールドワイド名の配列)。
- 有効な LUN 番号。
- ファイルシステムの種類。

PersistentVolume と LUN には1対1のマッピングがあります。

前提条件

- ファイバーチャネル LUN は基礎となるインフラストラクチャーに存在している必要があります。

PersistentVolume オブジェクト定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  fc:
```

```
targetWWNs: ['500a0981891b8dc5', '500a0981991b8dc5'] ❶
lun: 2
fsType: ext4
```

- ❶ ファイバーチャネル WWN は、`/dev/disk/by-path/pci-<IDENTIFIER>-fc-0x<WWN>-lun-<LUN#>` として識別されます。ただし、**WWN** までのパス (0x を含む) と WWN の後の文字 (- (ハイフン) を含む) を入力する必要はありません。



重要

ボリュームをフォーマットしてプロビジョニングした後に **fstype** パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

3.7.1.1. ディスククォータの実施

LUN パーティションを使用してディスククォータとサイズ制限を実施します。各 LUN は単一の PersistentVolume にマップされ、固有の名前は PersistentVolume に使用する必要があります。

この方法でクォータを実施すると、エンドユーザーは永続ストレージを具体的な量 (10Gi など) で要求することができ、これを同等またはそれ以上の容量の対応するボリュームに一致させることができます。

3.7.1.2. ファイバーチャネルボリュームのセキュリティ

ユーザーは PersistentVolumeClaim でストレージを要求します。この要求はユーザーの namespace にのみ存在し、同じ namespace 内の Pod からのみ参照できます。namespace をまたいで PersistentVolume にアクセスしようとすると、Pod にエラーが発生します。

それぞれのファイバーチャネル LUN は、クラスター内のすべてのノードからアクセスできる必要があります。

3.8. FLEXVOLUME を使用した永続ストレージ

OpenShift Container Platform は、ドライバーとのインターフェースに実行可能なモデルを使用する out-of-tree 形式のプラグイン、FlexVolume をサポートします。

組み込みプラグインがないバックエンドのストレージを使用する場合は、FlexVolume ドライバーを使用して OpenShift Container Platform を拡張し、アプリケーションに永続ストレージを提供できます。

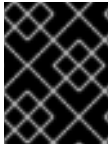
Pod は、**flexvolume** の in-tree 形式のプラグインを使用して FlexVolume ドライバーと対話します。

追加の参考資料

- [永続ボリュームの拡張](#)

3.8.1. FlexVolume ドライバーについて

FlexVolume ドライバーは、クラスター内のすべてのノードの明確に定義されたディレクトリーに格納されている実行可能ファイルです。OpenShift Container Platform は、**flexVolume** をソースとする **PersistentVolume** によって表されるボリュームのマウントまたはアンマウントが必要になるたびに、FlexVolume ドライバーを呼び出します。

**重要**

OpenShift Container Platform では、FlexVolume について割り当ておよび割り当て解除の操作はサポートされません。

3.8.2. FlexVolume ドライバーの例

FlexVolume ドライバーの最初のコマンドライン引数は常に操作名です。その他のパラメーターは操作ごとに異なります。ほとんどの操作は、JSON (JavaScript Object Notation) 文字列をパラメーターとして取ります。このパラメーターは完全な JSON 文字列であり、JSON データを含むファイルの名前ではありません。

FlexVolume ドライバーには以下が含まれます。

- すべての **flexVolume.options**。
- **kubernetes.io/** というプレフィックスが付いた **flexVolume** のいくつかのオプション。たとえば、**fsType** や **readwrite** などです。
- **kubernetes.io/secret/** というプレフィックスが付いた参照先シークレット (指定されている場合) の内容。

FlexVolume ドライバーの JSON 入力例

```
{
  "fooServer": "192.168.0.1:1234", ①
  "fooVolumeName": "bar",
  "kubernetes.io/fsType": "ext4", ②
  "kubernetes.io/readwrite": "ro", ③
  "kubernetes.io/secret/<key name>": "<key value>", ④
  "kubernetes.io/secret/<another key name>": "<another key value>",
}
```

- ① **flexVolume.options** のすべてのオプション。
- ② **flexVolume.fsType** の値。
- ③ **flexVolume.readOnly** に基づく **ro/rw**。
- ④ **flexVolume.secretRef** によって参照されるシークレットのすべてのキーと値。

OpenShift Container Platform は、ドライバーの標準出力に JSON データが含まれていると想定します。指定されていない場合、出力には操作の結果が示されます。

FlexVolume ドライバーのデフォルトの出力例

```
{
  "status": "<Success/Failure/Not supported>",
  "message": "<Reason for success/failure>"
}
```

ドライバーの終了コードは、成功の場合は **0**、エラーの場合は **1** です。

操作はべき等です。すでに割り当てられているボリュームのマウント操作は成功します。

3.8.3. FlexVolume ドライバーのインストール

OpenShift Container Platform を拡張するために使用される FlexVolume ドライバーはノードでのみ実行されます。FlexVolume を実装するには、呼び出す操作の一覧とインストールパスのみが必要になります。

前提条件

- FlexVolume ドライバーは、以下の操作を実装する必要があります。

init

ドライバーを初期化します。すべてのノードの初期化中に呼び出されます。

- 引数: なし
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

mount

ボリュームをディレクトリーにマウントします。これには、デバイスの検出、その後のデバイスのマウントを含む、ボリュームのマウントに必要なあらゆる操作が含まれます。

- 引数: **<mount-dir> <json>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

unmount

ボリュームをディレクトリーからアンマウントします。これには、アンマウント後にボリュームをクリーンアップするために必要なあらゆる操作が含まれます。

- 引数: **<mount-dir>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

mountdevice

ボリュームのデバイスを、個々の Pod がマウントをバインドするディレクトリーにマウントします。

この呼び出しでは FlexVolume 仕様に指定される「シークレット」を渡しません。ドライバーでシークレットが必要な場合には、この呼び出しを実装しないでください。

- 引数: **<mount-dir> <json>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON

unmountdevice

ボリュームのデバイスをディレクトリーからアンマウントします。

- 引数: **<mount-dir>**
- 実行場所: ノード
- 予期される出力: デフォルトの JSON
 - その他のすべての操作は、**{"status": "Not supported"}** と終了コード **1** を出して JSON を返します。

手順

FlexVolume ドライバーをインストールします。

1. この実行可能ファイルがクラスター内のすべてのノードに存在することを確認します。
2. この実行可能ファイルをボリュームプラグインのパス (**/etc/kubernetes/kubelet-plugins/volume/exec/<vendor>~<driver>/<driver>**) に配置します。

たとえば、ストレージ **foo** の FlexVolume ドライバーをインストールするには、実行可能ファイルを **/etc/kubernetes/kubelet-plugins/volume/exec/openshift.com~foo/foo** に配置します。

3.8.4. FlexVolume ドライバーを使用したストレージの使用

OpenShift Container Platform の各 **PersistentVolume** オブジェクトは、ストレージバックエンドの 1 つのストレージアセット (ボリュームなど) を表します。

手順

- インストールされているストレージを参照するには、**PersistentVolume** オブジェクトを使用します。

FlexVolume ドライバーを使用した永続ボリュームのオブジェクト定義例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 1Gi ❷
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: openshift.com/foo ❸
    fsType: "ext4" ❹
    secretRef: foo-secret ❺
    readOnly: true ❻
    options: ❼
      fooServer: 192.168.0.1:1234
      fooVolumeName: bar
```

- ❶ ボリュームの名前。これは Persistent Volume Claim (永続ボリューム要求) を使用するか、または Pod からボリュームを識別するために使用されます。この名前は、バックエンドストレージのボリューム名とは異なるものにすることができます。

- 2 このボリュームに割り当てられるストレージの量。
- 3 ドライバーの名前。このフィールドは必須です。
- 4 ボリュームに存在するオプションのファイルシステム。このフィールドはオプションです。
- 5 シークレットへの参照。このシークレットのキーと値は、起動時に FlexVolume ドライバーに渡されます。このフィールドはオプションです。
- 6 読み取り専用のフラグ。このフィールドはオプションです。
- 7 FlexVolume ドライバーの追加オプション。**options** フィールドでユーザーが指定するフラグに加え、以下のフラグも実行可能ファイルに渡されます。

```
"fsType": "<FS type>",
"readwrite": "<rw>",
"secret/key1": "<secret1>"
...
"secret/keyN": "<secretN>"
```



注記

シークレットは、呼び出しのマウント/マウント解除を目的とする場合にのみ渡されます。

3.9. GCE PERSISTENT DISK を使用した永続ストレージ

OpenShift Container Platform では、GCE Persistent Disk ボリューム (gcePD) がサポートされます。GCE を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と GCE についてある程度の理解があることが前提となります。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。

GCE Persistent Disk ボリュームは動的にプロビジョニングできます。

永続ボリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。Persistent Volume Claim (永続ボリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。



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。

追加の参考資料

- [GCE Persistent Disk](#)

3.9.1. GCE ストレージクラスの実装

ストレージクラスを使用すると、ストレージのレベルや使用状況を区別し、記述することができます。ストレージクラスを定義することにより、ユーザーは動的にプロビジョニングされた永続ボリュームを取得できます。

手順

1. OpenShift Container Platform コンソールで、**Storage → Storage Classes** をクリックします。
2. ストレージクラスの概要では、**Create Storage Class** をクリックします。
3. 表示されるページで必要なオプションを定義します。
 - a. ストレージクラスを参照するための名前を入力します。
 - b. オプションの説明を入力します。
 - c. 回収ポリシーを選択します。
 - d. ドロップダウンリストから **kubernetes.io/gce-pd** を選択します。
 - e. 必要に応じてストレージクラスの追加パラメーターを入力します。
4. **Create** をクリックしてストレージクラスを作成します。

3.9.2. Persistent Volume Claim (永続ボリューム要求、PVC) の作成

前提条件

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

手順

1. OpenShift Container Platform コンソールで、**Storage → Persistent Volume Claims** をクリックします。
2. Persistent Volume Claim (永続ボリューム要求、PVC) の概要で、**Create Persistent Volume Claim** をクリックします。
3. 表示されるページで必要なオプションを定義します。
 - a. ドロップダウンメニューから以前に作成されたストレージクラスを選択します。
 - b. ストレージ要求の一意の名前を入力します。
 - c. アクセスモードを選択します。これにより、作成されたストレージ要求の読み取り/書き込みアクセスが決定されます。
 - d. ストレージ要求のサイズを定義します。
4. **Create** をクリックして Persistent Volume Claim (永続ボリューム要求、PVC) を作成し、永続ボリュームを生成します。

3.9.3. ボリュームのフォーマット

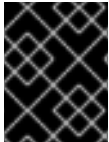
OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、永続ボリューム定義の **fsType** パラメーターで指定されたファイルシステムがボリュームにあるかどうか確認します。

デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

これにより、OpenShift Container Platform がフォーマットされていない GCE ボリュームを初回の使用前にフォーマットするため、それらを永続ボリュームとして使用することが可能になります。

3.10. HOSTPATH を使用した永続ストレージ

OpenShift Container Platform クラスター内の hostPath ボリュームは、ファイルまたはディレクトリーをホストノードのファイルシステムから Pod にマウントします。ほとんどの Pod には hostPath ボリュームは必要ありませんが、アプリケーションが必要とする場合は、テスト用のクイックオプションが提供されます。



重要

クラスター管理者は、特権付き Pod として実行するように Pod を設定する必要があります。これにより、同じノードの Pod へのアクセスが付与されます。

3.10.1. 概要

OpenShift Container Platform は単一ノードクラスターでの開発およびテスト用の hostPath マウントをサポートします。

実稼働クラスターでは、hostPath を使用しません。代わりにクラスター管理者は、GCE Persistent Disk ボリューム、NFS 共有、Amazon EBS ボリュームなどのネットワークリソースをプロビジョニングします。ネットワークリソースは、StorageClass を使用した動的プロビジョニングの設定をサポートします。

hostPath ボリュームは静的にプロビジョニングする必要があります。

3.10.2. hostPath ボリュームの静的プロビジョニング

hostPath ボリュームを使用する Pod は、手動の (静的) プロビジョニングで参照される必要があります。

手順

1. 永続ボリューム (PV) を定義します。PersistentVolume オブジェクト定義を使用して **pv.yaml** ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume 1
  labels:
    type: local
spec:
  storageClassName: manual 2
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce 3
```



```

persistentVolumeReclaimPolicy: Retain
hostPath:
  path: "/mnt/data" ④

```

- ① ボリュームの名前。この名前は PersistentVolumeClaim または Pod で識別されるものです。
- ② PersistentVolumeClaim 要求をこの PersistentVolume にバインドするために使用されます。
- ③ ボリュームは単一ノードで **read-write** としてマウントできます。
- ④ 設定ファイルでは、ボリュームがクラスターのノードの **/mnt/data** にあるように指定します。

2. ファイルから PV を作成します。

```
$ oc create -f pv.yaml
```

3. Persistent Volume Claim (永続ボリューム要求、PVC) を定義します。PersistentVolumeClaim オブジェクト定義を使用して、ファイル **pvc.yaml** を作成します。

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pvc-volume
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual

```

4. ファイルから PVC を作成します。

```
$ oc create -f pvc.yaml
```

3.10.3. 特権付き Pod での hostPath 共有のマウント

PersistentVolumeClaim の作成後に、これをアプリケーション内で使用できます。以下の例は、この共有を Pod 内にマウントする方法を示しています。

前提条件

- 基礎となる hostPath 共有にマップされる PersistentVolumeClaim があること。

手順

- 既存の PersistentVolumeClaim をマウントする特権付き Pod を作成します。

```

apiVersion: v1
kind: Pod

```

```

metadata:
  name: pod-name ❶
spec:
  containers:
    ...
  securityContext:
    privileged: true ❷
  volumeMounts:
    - mountPath: /data ❸
      name: hostpath-privileged
    ...
  securityContext: {}
  volumes:
    - name: hostpath-privileged
      persistentVolumeClaim:
        claimName: task-pvc-volume ❹

```

- ❶ Pod の名前。
- ❷ Pod はノードのストレージにアクセスするために特権付きとして実行される必要があります。
- ❸ 特権付き Pod 内に hostPath 共有をマウントするパス。
- ❹ 以前に作成された PersistentVolumeClaim の名前。

3.11. iSCSI を使用した永続ストレージ

iSCSI を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と iSCSI についてある程度の理解があることが前提となります。

Kubernetes 永続ボリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。



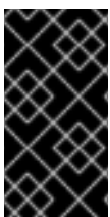
重要

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。



重要

Amazon Web Services で iSCSI を使用する場合、iSCSI ポートのノード間の TCP トラフィックを組み込むようにデフォルトのセキュリティーポリシーを更新する必要があります。デフォルトで、それらのポートは **860** および **3260** です。



重要

OpenShift では、クラスターのすべてのノードが iSCSI イニシエーターをすでに設定している、つまり、**iscsi-initiator-utils** パッケージをインストールし、それらのイニシエーターの名前を **/etc/iscsi/initiatorname.iscsi** に設定していることを前提とします。上記にリンクした『ストレージ管理ガイド』を参照してください。

3.11.1. プロビジョニング

OpenShift Container Platform でストレージをボリュームとしてマウントする前に、基礎となるインフラストラクチャーにストレージが存在することを確認します。iSCSI に必要なのは、iSCSI ターゲットポータル、有効な iSCSI 修飾名 (IQN)、有効な LUN 番号、ファイルシステムタイプ、および **PersistentVolume** API のみです。

例3.1 永続ボリュームオブジェクトの定義

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.16.154.81:3260
    iqn: iqn.2014-12.example.server:storage.target00
    lun: 0
    fsType: 'ext4'
```

3.11.2. ディスククォータの実施

LUN パーティションを使用してディスククォータとサイズ制限を実施します。それぞれの LUN には 1 つの永続ボリュームです。Kubernetes では、永続ボリュームに一意の名前を使用する必要があります。

上記の方法でクォータを実施すると、エンドユーザーは永続ストレージを具体的な量 (10Gi など) で要求することができ、これを同等またはそれ以上の容量の対応するボリュームに一致させることができます。

3.11.3. iSCSI ボリュームのセキュリティー

ユーザーは **PersistentVolumeClaim** でストレージを要求します。この要求はユーザーの namespace にのみ存在し、同じ namespace 内の Pod からのみ参照できます。namespace をまたいで Persistent Volume Claim (永続ボリューム要求) にアクセスしようとすると、Pod にエラーが発生します。

それぞれの iSCSI LUN は、クラスター内のすべてのノードからアクセスできる必要があります。

3.11.3.1. チャレンジハンドシェイク認証プロトコル (CHAP) 設定

オプションで、OpenShift は CHAP を使用して iSCSI ターゲットに対して自己認証を実行できます。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
```

```

accessModes:
  - ReadWriteOnce
iscsi:
  targetPortal: 10.0.0.1:3260
  iqn: iqn.2016-04.test.com:storage.target00
  lun: 0
  fsType: ext4
  chapAuthDiscovery: true ❶
  chapAuthSession: true ❷
  secretRef:
    name: chap-secret ❸

```

- ❶ iSCSI 検出の CHAP 認証を有効にします。
- ❷ iSCSI セッションの CHAP 認証を有効にします。
- ❸ ユーザー名 + パスワードを使用してシークレットオブジェクトの名前を指定します。このシークレットオブジェクトは、参照されるボリュームを使用できるすべての namespace で利用可能でなければなりません。

3.11.4. iSCSI のマルチパス化

iSCSI ベースのストレージの場合は、複数のターゲットポータルの IP アドレスに同じ IQN を使用することでマルチパスを設定できます。マルチパス化により、パス内の 1 つ以上のコンポーネントで障害が発生した場合でも、永続ボリュームにアクセスすることができます。

Pod 仕様でマルチパスを指定するには、**portals** フィールドを使用します。以下は例になります。

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260'] ❶
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false

```

- ❶ **portals** フィールドを使用してターゲットポータルを追加します。

3.11.5. iSCSI のカスタムイニシエーター IQN

iSCSI ターゲットが特定に IQN に制限されている場合に、カスタムイニシエーターの iSCSI Qualified Name (IQN) を設定します。ただし、iSCSI PV が割り当てられているノードが必ずこれらの IQN を使用する保証はありません。

カスタムのイニシエーター IQN を指定するには、**initiatorName** フィールドを使用します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 10.0.0.1:3260
    portals: ['10.0.2.16:3260', '10.0.2.17:3260', '10.0.2.18:3260']
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    initiatorName: iqn.2016-04.test.com:custom.iqn ❶
    fsType: ext4
    readOnly: false
```

❶ イニシエーターの名前を指定します。

3.12. ローカルボリュームを使用した永続ストレージ

OpenShift Container Platform は、ローカルボリュームを使用する永続ストレージでプロビジョニングすることが可能です。ローカルの永続ボリュームを使用すると、標準の PVC インターフェースを使用して、ディスクやパーティションなどのローカルのストレージデバイスにアクセスできます。

ローカルボリュームは、Pod をノードに手動でスケジュールせずに使用できます。ボリュームのノード制約がシステムによって認識されるためです。ただし、ローカルボリュームは、依然として基礎となるノードの可用性に依存しており、すべてのアプリケーションに適している訳ではありません。



注記

ローカルボリュームは、静的に作成された永続ボリュームとしてのみ使用できます。

3.12.1. ローカルストレージ Operator のインストール

ローカルストレージ Operator はデフォルトで OpenShift Container Platform にインストールされません。以下の手順を使用してこの Operator をインストールし、クラスター内でローカルボリュームを有効にできるように設定します。

前提条件

- OpenShift Container Platform Web コンソールまたはコマンドラインインターフェース (CLI) へのアクセス。

手順

1. **local-storage** プロジェクトを作成します。

```
$ oc new-project local-storage
```

- オプション: マスターおよびインフラストラクチャーノードでのローカルストレージの作成を許可します。
ロギングやモニタリングなどのコンポーネントに対応するために、ワーカーノードだけではなく、ローカルストレージ Operator を使用してマスターおよびインフラストラクチャーノードでボリュームを作成する必要がある場合があります。

マスターおよびインフラストラクチャーノードでローカルストレージを作成できるようにするには、以下のコマンドを入力して容認を DaemonSet に追加します。

```
$ oc patch ds local-storage-local-diskmaker -n local-storage -p '{"spec": {"template": {"spec": {"tolerations":[{"operator": "Exists"}]}}}}'
```

```
$ oc patch ds local-storage-local-provisioner -n local-storage -p '{"spec": {"template": {"spec": {"tolerations":[{"operator": "Exists"}]}}}}'
```

UI での操作

Web コンソールからローカルストレージ Operator をインストールするには、以下の手順を実行します。

- OpenShift Container Platform Web コンソールにログインします。
- Operators** → **OperatorHub** に移動します。
- Local Storage** をフィルターボックスに入力して、ローカルストレージ Operator を見つけます。
- Install** をクリックします。
- Create Operator Subscription** ページで、**A specific namespace on the cluster** を選択します。ドロップメニューから **local-storage** を選択します。
- Update Channel** および **Approval Strategy** の値を必要な値に調整します。
- Subscribe** をクリックします。

これが完了すると、ローカルストレージ Operator は Web コンソールの **Installed Operators** セクションに一覧表示されます。

CLI からの操作

- CLI からローカルストレージ Operator をインストールします。
 - ローカルストレージ Operator の namespace、OperatorGroup、およびサブスクリプションを定義するために、オブジェクト YAML ファイルを作成します (例: **local-storage.yaml**)。

Local-storage の例

```
apiVersion: v1
kind: Namespace
metadata:
  name: local-storage
---
apiVersion: operators.coreos.com/v1alpha2
kind: OperatorGroup
```

```

metadata:
  name: local-operator-group
  namespace: local-storage
spec:
  targetNamespaces:
    - local-storage
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: local-storage
spec:
  channel: "{product-version}" ❶
  installPlanApproval: Automatic
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace

```

- ❶ このフィールドは、OpenShift Container Platform のリリースの選択に一致するように編集できます。

2. 以下のコマンドを実行して、ローカルストレージ Operator オブジェクトを作成します。

```
$ oc apply -f local-storage.yaml
```

この時点で、Operator Lifecycle Manager (OLM) はローカルストレージ Operator を認識できるようになります。Operator の ClusterServiceVersion (CSV) はターゲット namespace に表示され、Operator で指定される API は作成用に利用可能になります。

3. すべての Pod およびローカルストレージ Operator が作成されていることを確認して、ローカルストレージのインストールを検証します。
- a. 必要な Pod すべてが作成されていることを確認します。

```

$ oc -n local-storage get pods
NAME                                READY STATUS RESTARTS AGE
local-storage-operator-746bf599c9-vlt5t 1/1   Running 0      19m

```

- b. ClusterServiceVersion (CSV) YAML マニフェストをチェックして、ローカルストレージ Operator が **local-storage** プロジェクトで利用できることを確認します。

```

$ oc get csvs -n local-storage
NAME                                DISPLAY          VERSION          REPLACES          PHASE
local-storage-operator.4.2.26-202003230335 Local Storage    4.2.26-202003230335
Succeeded

```

すべてのチェックが渡されると、ローカルストレージ Operator が正常にインストールされます。

3.12.2. ローカルボリュームのプロビジョニング

ローカルボリュームは動的プロビジョニングで作成できません。代わりに、PersistentVolume がローカルストレージ Operator によって作成される必要があります。このプロビジョナーは、定義されたりソースで指定されているパスでデバイス (ファイルシステムおよびブロックボリュームの両方) を検索し

ます。

前提条件

- ローカルストレージ Operator がインストールされていること。
- ローカルディスクが OpenShift Container Platform ノードに割り当てられていること。

手順

- ローカルボリュームリソースを作成します。これは、ノードおよびローカルボリュームへのパスを定義する必要があります。



注記

同じデバイスに別の StorageClass 名を使用しないでください。これを行うと、複数の永続ボリューム (PV) が作成されます。

例: ファイルシステム

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "local-storage" ❶
spec:
  nodeSelector: ❷
  nodeSelectorTerms:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
          - ip-10-0-140-183
          - ip-10-0-158-139
          - ip-10-0-164-33
  storageClassDevices:
    - storageClassName: "local-sc"
      volumeMode: Filesystem ❸
      fsType: xfs ❹
      devicePaths: ❺
        - /path/to/device ❻
```

- ❶ ローカルストレージ Operator がインストールされている namespace。
- ❷ オプション: ローカルストレージボリュームが割り当てられているノードの一覧が含まれるノードセレクター。以下の例では、**oc get node** から取得したノードホスト名を使用します。値が定義されない場合、ローカルストレージ Operator は利用可能なすべてのノードで一致するディスクの検索を試行します。
- ❸ ボリュームモード (**Filesystem** または **Block**) で、ローカルボリュームのタイプを定義します。
- ❹ ローカルボリュームの初回マウント時に作成されるファイルシステム。

- 5 選択するローカルストレージデバイスの一覧を含むパスです。
- 6 この値を、`/dev/xvddg` などの LocalVolume リソースへの実際のローカルディスクのファイルパスに置き換えます。プロビジョナーが正常にデプロイされると、これらのローカルディスク用に PV が作成されます。

例: ブロック

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "local-storage" 1
spec:
  nodeSelector: 2
  nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - ip-10-0-136-143
            - ip-10-0-140-255
            - ip-10-0-144-180
  storageClassDevices:
    - storageClassName: "localblock-sc"
      volumeMode: Block 3
      devicePaths: 4
        - /path/to/device 5
```

- 1 ローカルストレージ Operator がインストールされている namespace。
- 2 オプション: ローカルストレージボリュームが割り当てられているノードの一覧が含まれるノードセレクター。以下の例では、**oc get node** から取得したノードホスト名を使用します。値が定義されない場合、ローカルストレージ Operator は利用可能なすべてのノードで一致するディスクの検索を試行します。
- 3 ボリュームモード (**Filesystem** または **Block**) で、ローカルボリュームのタイプを定義します。
- 4 選択するローカルストレージデバイスの一覧を含むパスです。
- 5 この値を、`/dev/xvddg` などの LocalVolume リソースへの実際のローカルディスクのファイルパスに置き換えます。プロビジョナーが正常にデプロイされると、これらのローカルディスク用に PV が作成されます。

2. 先に作成したファイルを指定して、OpenShift Container Platform クラスターにローカルボリュームリソースを作成します。

```
$ oc create -f <local-volume>.yaml
```

3. プロビジョナーが作成され、対応する DaemonSet が作成されていることを確認します。

```
$ oc get all -n local-storage
```

NAME	READY	STATUS	RESTARTS	AGE
pod/local-disks-local-provisioner-h97hj	1/1	Running	0	46m
pod/local-disks-local-provisioner-j4mnn	1/1	Running	0	46m
pod/local-disks-local-provisioner-kbdnx	1/1	Running	0	46m
pod/local-disks-local-diskmaker-ldldw	1/1	Running	0	46m
pod/local-disks-local-diskmaker-lrvv4	1/1	Running	0	46m
pod/local-disks-local-diskmaker-phxdq	1/1	Running	0	46m
pod/local-storage-operator-54564d9988-vxvhx	1/1	Running	0	47m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/local-storage-operator	ClusterIP	172.30.49.90	<none>	60000/TCP	47m

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
daemonset.apps/local-disks-local-provisioner	3	3	3	3	3	<none>	46m
daemonset.apps/local-disks-local-diskmaker	3	3	3	3	3	<none>	46m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/local-storage-operator	1/1	1	1	47m

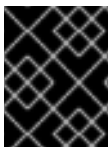
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/local-storage-operator-54564d9988	1	1	1	47m

DaemonSet プロセスの必要な数と現在の数に注意してください。必要な数が **0** の場合、これはラベルセクターが無効であることを示します。

- PersistentVolume が作成されていることを確認します。

```
$ oc get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE			
local-pv-1cec77cf	100Gi	RWO	Delete	Available	local-sc 88m
local-pv-2ef7cd2a	100Gi	RWO	Delete	Available	local-sc 82m
local-pv-3fa1c73	100Gi	RWO	Delete	Available	local-sc 48m



重要

LocalVolume オブジェクトを編集しても、既存の PersistentVolume の **fsType** または **volumeMode** は変更されません。これは破壊的な操作になる可能性があります。

3.12.3. ローカルボリューム PersistentVolumeClaim の作成

ローカルボリュームは、Pod でアクセスされる PersistentVolumeClaim (PVC) として静的に作成される必要があります。

前提条件

- Persistentvolume がローカルボリュームプロビジョナーを使用して作成されていること。

手順

1. 対応する StorageClass を使用して PVC を作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: local-pvc-name ❶
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem ❷
  resources:
    requests:
      storage: 100Gi ❸
  storageClassName: local-sc ❹
```

- ❶ PVC の名前。
- ❷ PVC のタイプ。デフォルトは **Filesystem** です。
- ❸ PVC に利用できるストレージの量。
- ❹ 要求で必要になる StorageClass の名前。

2. 作成したファイルを指定して、PVC を OpenShift Container Platform クラスターに作成します。

```
$ oc create -f <local-pvc>.yaml
```

3.12.4. ローカル要求を割り当てます。

ローカルボリュームが PersistentVolumeClaim (PVC) にマップされた後に、これをリソース内に指定できます。

前提条件

- PVC が同じ namespace に存在する。

手順

1. 定義された要求をリソースの仕様に追加します。以下の例では、Pod 内で PVC を宣言します。

```
apiVersion: v1
kind: Pod
spec:
  ...
  containers:
    volumeMounts:
      - name: localpvc ❶
        mountPath: "/data" ❷
  volumes:
```

```
- name: localpvc
  persistentVolumeClaim:
    claimName: localpvc ❸
```

- ❶ マウントするボリュームの名前。
- ❷ ボリュームがマウントされる Pod 内のパス。
- ❸ 使用する既存 PVC の名前。

2. 作成したファイルを指定して、OpenShift Container Platform クラスターにリソースを作成します。

```
$ oc create -f <local-pod>.yaml
```

3.12.5. ローカルストレージ Operator Pod での容認の使用

ティントはノードに適用し、それらが一般的なワークロードを実行しないようにすることができます。ローカルストレージ Operator がティントのマークが付けられたノードを使用できるようにするには、容認を Pod または DaemonSet 定義に追加する必要があります。これにより、作成されたリソースをこれらのティントのマークが付けられたノードで実行できるようになります。

容認を LocalVolume リソースでローカルストレージ Operator Pod に適用し、ティントをノード仕様にノードに適用します。ノードのティントはノードに対し、ティントを容認しないすべての Pod を拒否するよう指示します。他の Pod にはない特定のティントを使用することで、ローカルストレージ Operator Pod がそのノードでも実行されるようになります。



重要

ティントおよび容認は、key、value、および effect で構成されています。引数として、これは **key=value:effect** として表現されます。演算子により、これらの3つのパラメーターのいずれかを空のままにすることができます。

前提条件

- ローカルストレージ Operator がインストールされていること。
- ローカルディスクがティントを持つ OpenShift Container Platform ノードに割り当てられている。
- ティントのマークが付けられたノードがローカルストレージのプロビジョニングを行うことが想定されます。

手順

ティントのマークが付けられたノードでスケジュールするようにローカルボリュームを設定するには、以下を実行します。

1. 以下の例に示されるように、Pod を定義する YAML ファイルを変更し、**LocalVolume** 仕様を追加します。

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
```

```

name: "local-disks"
namespace: "local-storage"
spec:
  tolerations:
    - key: localstorage ❶
      operator: Equal ❷
      value: "localstorage" ❸
  storageClassDevices:
    - storageClassName: "localblock-sc"
      volumeMode: Block ❹
      devicePaths: ❺
        - /dev/xvdg

```

- ❶ ノードに追加したキーを指定します。
- ❷ **Equal** Operator を指定して、**key/value** パラメーターが一致するようにします。Operator が「Exists」の場合、システムはキーが存在することを確認し、値を無視します。Operator が **Equal** の場合、キーと値が一致する必要があります。
- ❸ テイントのマークが付けられたノードの値 **local** を指定します。
- ❹ ボリュームモード (**Filesystem** または **Block**) で、ローカルボリュームのタイプを定義します。
- ❺ 選択するローカルストレージデバイスの一覧を含むパスです。

定義された容認は結果として作成される DaemonSet に渡されます。これにより、diskmaker およびプロビジョナー Pod を指定されたテイントが含まれるノード用に作成できます。

3.12.6. ローカルストレージ Operator のリソースの削除

3.12.6.1. ローカルボリュームの削除

ローカルボリュームを削除する必要がある場合があります。LocalVolume リソースのエントリを削除し、PersistentVolume を削除することで通常は十分ですが、同じデバイスパスを再使用する場合や別の StorageClass でこれを管理する必要がある場合には、追加の手順が必要になります。



警告

以下の手順では、root ユーザーとしてノードにアクセスします。この手順のステップ以外にノードの状態を変更すると、クラスターが不安定になる可能性があります。

前提条件

- PersistentVolume の状態は **Released** または **Available** である必要があります。



警告

使用中の PersistentVolume を削除すると、データの損失や破損につながる可能性があります。

手順

1. 以前に作成した LocalVolume を編集して、不要なディスクを削除します。

- a. クラスタリソースを編集します。

```
$ oc edit localvolume <name> -n local-storage
```

- b. **devicePaths** の下の行に移動し、不要なディスクを表すものを削除します。

2. 作成した PersistentVolume を削除します。

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

3. ノードのシンボリックリンクを削除します。

- a. ノードにデバッグ Pod を作成します。

```
$ oc debug node/<node-name>
```

- b. ルートディレクトリーをホストに切り替えます。

```
$ chroot /host
```

- c. ローカルボリュームのシンボリックリンクを含むディレクトリーに移動します。

```
$ cd /mnt/local-storage/<sc-name> 1
```

- 1** ローカルボリュームの作成に使用される StorageClass の名前。

- d. 削除したデバイスに属するシンボリックリンクを削除します。

```
$ rm <symlink>
```

3.12.6.2. ローカルストレージ Operator のアンインストール

ローカルストレージ Operator をアンインストールするには、Operator および **local-storage** プロジェクトの作成されたすべてのリソースを削除する必要があります。



警告

ローカルストレージ PV がまだ使用中の状態でもローカルストレージ Operator をアンインストールすることは推奨されません。PV は Operator の削除後も残りますが、PV およびローカルストレージリソースを削除せずに Operator がアンインストールされ、再インストールされる場合に予測できない動作が生じる可能性があります。

前提条件

- OpenShift Container Platform Web コンソールへのアクセスが可能です。

手順

1. プロジェクトのローカルボリュームリソースを削除します。

```
$ oc delete localvolume --all --all-namespaces
```

2. Web コンソールからローカルストレージ Operator をアンインストールします。

- a. OpenShift Container Platform Web コンソールにログインします。
- b. **Operators** → **Installed Operators** に移動します。
- c. **Local Storage** をフィルターボックスに入力して、ローカルストレージ Operator を見つけます。
- d. Local Storage Operator の末尾にある Options メニュー  をクリックします。
- e. **Uninstall Operator** をクリックします。
- f. 表示されるウィンドウで **Remove** をクリックします。

3. ローカルストレージ Operator で作成された PV は削除されるまでクラスターに残ります。これらのボリュームが使用されなくなったら、以下のコマンドを実行してこれらのボリュームを削除します。

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

4. **local-storage** プロジェクトを削除します。

```
$ oc delete project local-storage
```

3.13. NFS を使用した永続ストレージ

OpenShift Container Platform クラスターは、NFS を使用する永続ストレージでプロビジョニングすることが可能です。永続ボリューム (PV) および Persistent Volume Claim (永続ボリューム要求、PVC) は、プロジェクト全体でボリュームを共有するための便利な方法を提供します。PV 定義に含まれる

NFS に固有の情報は、Pod 定義で直接定義することも可能ですが、この方法の場合にはボリュームが一意のクラスターリソースとして作成されされないため、ボリュームが競合の影響を受けやすくなります。

追加リソース

- 「[ネットワークファイルシステム \(NFS\)](#)」

3.13.1. プロビジョニング

ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。NFS ボリュームをプロビジョニングするには、NFS サーバーの一覧とエクスポートパスのみが必要です。

手順

1. PV のオブジェクト定義を作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001 ❶
spec:
  capacity:
    storage: 5Gi ❷
  accessModes:
    - ReadWriteOnce ❸
  nfs: ❹
    path: /tmp ❺
    server: 172.17.0.2 ❻
  persistentVolumeReclaimPolicy: Retain ❼
```

- ❶ ボリュームの名前。これは、各種の **oc <command> pod** コマンドの PV アイデンティティです。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ これはボリュームへのアクセスの制御に関連するよう見えますが、実際はラベルの場合と同様に、PVC を PV に一致させるために使用されます。現時点では、**accessModes** に基づくアクセスルールは適用されていません。
- ❹ 使用されているボリュームタイプ。この場合は **nfs** プラグインです。
- ❺ NFS サーバーがエクスポートしているパス。
- ❻ NFS サーバーのホスト名または IP アドレス
- ❼ PV の回収ポリシー。これはボリュームのリリース時に生じることを定義します。



注記

各 NFS ボリュームは、クラスター内のスケジュール可能なすべてのノードによってマウント可能でなければなりません。

2. PV が作成されたことを確認します。

```
$ oc get pv
NAME      LABELS    CAPACITY  ACCESSMODES  STATUS   CLAIM  REASON  AGE
pv0001    <none>    5Gi       RWO           Available             31s
```

3. 新規 PV にバインドされる Persistent Volume Claim (永続ボリューム要求、PVC) を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim1
spec:
  accessModes:
    - ReadWriteOnce ❶
  resources:
    requests:
      storage: 5Gi ❷
```

- ❶ PV について前述されているように、**accessModes** はセキュリティーを実施するのではなく、PV を PVC と一致させるラベルとして機能します。
- ❷ この要求は **5Gi** 以上の容量を提供する PV を検索します。

4. Persistent Volume Claim (永続ボリューム要求、PVC) が作成されたことを確認します。

```
$ oc get pvc
NAME          STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
nfs-claim1    Bound   pv0001  5Gi       RWO           gp2            2m
```

3.13.2. ディスククォータの実施

ディスクパーティションを使用して、ディスククォータとサイズ制限を実施することができます。それぞれのパーティションを独自のエクスポートとすることができ、それぞれのエクスポートは1つの PV になります。それぞれのエクスポートは1つの PV になります。OpenShift Container Platform は PV に固有の名前を適用しますが、NFS ボリュームのサーバーとパスの一意性については管理者に委ねられています。

この方法でクォータを実施すると、開発者は永続ストレージを具体的な量 (10Gi など) で要求することができ、同等かそれ以上の容量の対応するボリュームに一致させることができます。

3.13.3. NFS ボリュームのセキュリティー

このセクションでは、一致するパーミッションや SELinux の考慮点を含む、NFS ボリュームのセキュリティーについて説明します。ユーザーは、POSIX パーミッションやプロセス UID、補助グループおよび SELinux の基礎的な点を理解している必要があります。

開発者は、Pod 定義の **volumes** セクションで、PVC を名前で参照するか、または NFS ボリュームのプラグインを直接参照して NFS ストレージを要求します。

NFS サーバーの **/etc/exports** ファイルにはアクセス可能な NFS ディレクトリーが含まれています。ターゲットの NFS ディレクトリーには、POSIX の所有者とグループ ID があります。OpenShift

Container Platform NFS プラグインは、同じ POSIX の所有者とエクスポートされる NFS ディレクトリーにあるパーミッションを使って、コンテナの NFS ディレクトリーをマウントします。ただし、コンテナは NFS マウントの所有者と同等の有効な UID では実行されません。これは期待される動作です。

ターゲットの NFS ディレクトリーが NFS サーバーに表示される場合を例に取って見てみましょう。

```
$ ls -lZ /opt/nfs -d
drwxrws---. nfsnobody 5555 unconfined_u:object_r:usr_t:s0 /opt/nfs

$ id nfsnobody
uid=65534(nfsnobody) gid=65534(nfsnobody) groups=65534(nfsnobody)
```

次に、コンテナは SELinux ラベルに一致し、ディレクトリーにアクセスするために UID の **65534**、**nfsnobody** 所有者、または補助グループの **5555** のいずれかで実行される必要があります。



注記

所有者 ID **65534** は一例として使用されています。NFS の **root_squash** が **root**、uid **0** を **nfsnobody**、uid **65534** にマップしても、NFS エクスポートは任意の所有者 ID を持つことができます。所有者 **65534** は NFS エクスポートには必要ありません。

3.13.3.1. グループ ID

NFS アクセスに対応する際の推奨される方法として、補助グループを使用することができます (NFS エクスポートのパーミッションを変更するオプションがないことを前提としています)。OpenShift Container Platform の補助グループは共有ストレージに使用されます (例: NFS)。これとは対照的に、iSCSI などのブロックストレージは、Pod の **securityContext** で **fsGroup** SCC ストラテジーと **fsGroup** の値を使用します。



注記

永続ストレージへのアクセスを取得するには、通常はユーザー ID ではなく、補助グループ ID を使用することが推奨されます。

ターゲット NFS ディレクトリーの例で使用したグループ ID は **5555** なので、Pod は、**supplementalGroups** を使用してグループ ID を Pod の **securityContext** 定義の下で定義することができます。以下は例になります。

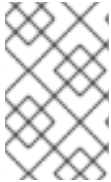
```
spec:
  containers:
    - name:
      ...
  securityContext: ❶
    supplementalGroups: [5555] ❷
```

❶ **securityContext** は特定のコンテナの下ではなく、この Pod レベルで定義される必要があります。

❷ Pod 向けに定義される GID の配列。この場合、配列には1つの要素があります。追加の GID はカンマで区切られます。

Pod の要件を満たすカスタム SCC が存在しない場合、Pod は **restricted** SCC に一致する可能性があります。この SCC では、**supplementalGroups** ストラテジーが **RunAsAny** に設定されています。これは、指定されるグループ ID は範囲のチェックなしに受け入れられることを意味します。

その結果、上記の Pod は受付をパスして起動します。しかし、グループ ID の範囲をチェックすることが望ましい場合は、カスタム SCC の使用が推奨されます。カスタム SCC は、最小および最大のグループ ID が定義され、グループ ID の範囲チェックが実施され、グループ ID の **5555** が許可されるように作成できます。



注記

カスタム SCC を使用するには、まずこれを適切なサービスアカウントに追加する必要があります。たとえば、Pod 仕様に指定がない場合には、指定されたプロジェクトで **default** サービスアカウントを使用します。

3.13.3.2. ユーザー ID

ユーザー ID は、コンテナイメージまたは Pod 定義で定義することができます。



注記

永続ストレージへのアクセスを取得する場合、通常はユーザー ID ではなく、補助グループ ID を使用することが推奨されます。

上記のターゲット NFS ディレクトリーの例では、コンテナは UID を **65534** (ここではグループ ID を省略します) に設定する必要があります。したがって以下を Pod 定義に追加することができます。

```
spec:
  containers: ❶
  - name:
    ...
    securityContext:
      runAsUser: 65534 ❷
```

❶ Pod には、各コンテナに固有の **securityContext** と、その Pod で定義されたすべてのコンテナに適用される Pod の **securityContext** が含まれます。

❷ **65534** は **nfsnobody** ユーザーです。

default プロジェクトと **restricted** SCC を前提とする場合は、Pod が要求するユーザー ID **65534** は許可されず、Pod は失敗します。Pod が失敗する理由は以下の通りです。

- **65534** をそのユーザー ID として要求する。
- ユーザー ID **65534** を許可する SCC を確認するために Pod で利用できるすべての SCC が検査される。SCC のすべてのポリシーがチェックされますが、ここでのフォーカスはユーザー ID になります。
- 使用可能なすべての SCC が独自の **runAsUser** ストラテジーとして **MustRunAsRange** を使用しているため、UID の範囲チェックが要求される。
- **65534** は SCC またはプロジェクトのユーザー ID 範囲に含まれていない。

一般に、事前定義された SCC は変更しないことが勧められています。ただし、この状況を改善するに

は、カスタム SCC を作成することが推奨されます。カスタム SCC は、最小および最大のユーザー ID が定義され、UID 範囲のチェックの実施が設定されており、UID **65534** が許可されるように作成できます。



注記

カスタム SCC を使用するには、まずこれを適切なサービスアカウントに追加する必要があります。たとえば、Pod 仕様に指定がない場合には、指定されたプロジェクトで **default** サービスアカウントを使用します。

3.13.3.3. SELinux

デフォルトでは、SELinux は Pod からリモートの NFS サーバーへの書き込みを許可していません。NFS ボリュームは正常にマウントされますが、読み取り専用です。

リモート NFS サーバーへの書き込みを有効にするには、以下の手順に従ってください。

前提条件

- **container-selinux** パッケージがインストールされている必要があります。このパッケージは **virt_use_nfs** SELinux ブール値を提供します。

手順

- 以下のコマンドを使用して **virt_use_nfs** ブール値を有効にします。-P オプションを使用すると、再起動後もこのブール値を永続化できます。

```
# setsebool -P virt_use_nfs 1
```

3.13.3.4. エクスポート設定

任意のコンテナユーザーにボリュームの読み取りと書き出しを許可するには、NFS サーバーにエクスポートされる各ボリュームは以下の条件を満たしている必要があります。

- すべてのエクスポートは、次の形式を使用してエクスポートする必要があります。

```
/<example_fs> *(rw,root_squash)
```

- ファイアウォールは、マウントポイントへのトラフィックを許可するように設定する必要があります。
 - NFSv4 の場合、デフォルトのポート **2049** (nfs) を設定します。

NFSv4

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

- NFSv3 の場合、以下の 3 つのポートを設定します。 **2049** (nfs)、 **20048** (mountd)、 **111** (portmapper)。

NFSv3

```
# iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 20048 -j ACCEPT
# iptables -I INPUT 1 -p tcp --dport 111 -j ACCEPT
```

- NFS エクスポートとディレクトリーは、ターゲット Pod からアクセスできるようにセットアップされる必要があります。この場合、エクスポートをコンテナのプライマリー UID で所有されるように設定するか、または上記のグループ ID に示されるように **supplementalGroups** を使用して Pod にグループアクセスを付与します。

3.13.4. リソースの回収

NFS は OpenShift Container Platform の **Recyclable** プラグインインターフェースを実装します。回収タスクは、それぞれの永続ボリュームに設定されるポリシーに基づいて自動プロセスによって処理されます。

デフォルトで、PV は **Retain** に設定されます。

PV への要求が削除され、PV がリリースされると、PV オブジェクトを再利用できません。代わりに、新規の PV が元のボリュームと同じ基本ボリュームの情報を使って作成されます。

たとえば、管理者は **nfs1** という名前の PV を作成するとします。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs1
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

ユーザーは、**nfs1** にバインドされる **PVC1** を作成します。次にユーザーは **PVC1** を削除し、**nfs1** への要求を解除します。これにより、**nfs1** は **Released** になります。管理者が同じ NFS 共有を利用可能にする必要がある場合には、同じ NFS サーバー情報を使って新規 PV を作成する必要があります。この場合、PV の名前は元の名前とは異なる名前にします。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs2
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.1.1
    path: "/"
```

元の PV を削除して、PV を同じ名前で再作成することは推奨されません。PV のステータスを **Released** から **Available** に手動で変更しようとする、エラーが発生し、データが失われる可能性があります。

3.13.5. その他の設定とトラブルシューティング

適切なエクスポートとセキュリティーマッピングを行うため、使用している NFS のバージョンおよびその設定方法に応じて追加の設定が必要になることがあります。以下は例になります。

NFSv4 のマウントにすべてのファイルの所有者が nobody:nobody と誤って表示される。	<ul style="list-style-type: none"> NFS の ID マッピング設定 (<code>/etc/idmapd.conf</code>) に原因がある可能性が高い。 「NFSv4 mount incorrectly shows all files with ownership as nobody:nobody」を参照してください。
NFSv4 の ID マッピングが無効になっている	<ul style="list-style-type: none"> NFS クライアントとサーバーの両方で以下を実行してください。 <pre># echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping</pre>

3.14. RED HAT OPENSIFT CONTAINER STORAGE

Red Hat OpenShift Container Storage は、インハウスまたはハイブリッドクラウドのいずれの場合でもファイル、ブロックおよびオブジェクトストレージをサポートし、OpenShift Container Platform のすべてに対応する永続ストレージのプロバイダーです。Red Hat のストレージソリューションとして、Red Hat OpenShift Container Storage は、デプロイメント、管理およびモニタリングを行うために OpenShift Container Platform に完全に統合されています。

Red Hat OpenShift Container Storage は、独自のドキュメントライブラリーを提供します。以下の Red Hat OpenShift Container Storage ドキュメントすべては https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/4.3/ から入手できます。

Red Hat OpenShift Container Storage についてのトピック	Red Hat OpenShift Container Storage ドキュメントの参照先
新機能、既知の問題、主なバグ修正およびテクノロジーレビュー	Red Hat OpenShift Container Storage 4.3 Release Notes
サポートされるワークロード、レイアウト、ハードウェアおよびソフトウェア要件、サイジング、スケールリングに関する推奨事項	Planning your Red Hat OpenShift Container Storage 4.3 deployment
Red Hat OpenShift Container Storage 4.3 の既存の OpenShift Container Platform クラスターへのデプロイ	Deploying Red Hat OpenShift Container Storage 4.3

Red Hat OpenShift Container Storage についてのトピック	Red Hat OpenShift Container Storage ドキュメントの参照先
Red Hat OpenShift Container Storage 4.3 クラスターの管理	Managing Red Hat OpenShift Container Storage 4.3
Red Hat OpenShift Container Storage 4.3 クラスターのモニタリング	Monitoring Red Hat OpenShift Container Storage 4.3
OpenShift Container Platform クラスターのバージョン 3 からバージョン 4 への移行	『Red Hat OpenShift Container Platform 4.5 移行』

3.15. VMWARE VSPHERE ボリリュームを使用した永続ストレージ

OpenShift Container Platform では、VMWare vSphere の仮想マシンディスク (VMDK: Virtual Machine Disk) ボリリュームの使用が可能となります。VMWare vSphere を使用して、OpenShift Container Platform クラスターに永続ストレージをプロビジョニングできます。これには、Kubernetes と VMWare vSphere についてのある程度の理解があることが前提となります。

VMware vSphere ボリリュームは動的にプロビジョニングできます。OpenShift Container Platform は vSphere にディスクを作成し、このディスクを正しいイメージに割り当てます。

Kubernetes 永続ボリリュームフレームワークは、管理者がクラスターのプロビジョニングを永続ストレージを使用して実行できるようにし、ユーザーが基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようにします。

永続ボリリュームは単一のプロジェクトまたは namespace にバインドされず、それらは OpenShift Container Platform クラスター間で共有できます。PersistentVolumeClaim (永続ボリリューム要求、PVC) はプロジェクトまたは namespace に固有のもので、ユーザーによって要求されます。

追加の参考資料

- [VMware vSphere](#)

3.15.1. VMware vSphere ボリリュームの動的プロビジョニング

VMware vSphere ボリリュームの動的プロビジョニングは推奨される方法です。

3.15.2. 前提条件

- 使用するコンポーネントの要件を満たす VMware vSphere バージョンにインストールされている OpenShift Container Platform クラスター。vSphere バージョンのサポートに関する詳細は、[vSphere へのクラスターのインストール](#)について参照してください。

以下のいずれかの手順を使用し、デフォルトの StorageClass を使用してそれらのボリリュームを動的にプロビジョニングできます。

3.15.2.1. UI を使用した VMware vSphere ボリリュームの動的プロビジョニング

OpenShift Container Platform は、ボリュームをプロビジョニングするために **thin** ディスク形式を使用する **thin** という名前のデフォルトの StorageClass をインストールします。

前提条件

- ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

手順

1. OpenShift Container Platform コンソールで、**Storage → Persistent Volume Claims** をクリックします。
2. Persistent Volume Claim (永続ボリューム要求、PVC) の概要で、**Create Persistent Volume Claim** をクリックします。
3. 結果のページで必要なオプションを定義します。
 - a. **thin** StorageClass を選択します。
 - b. ストレージ要求の一意の名前を入力します。
 - c. アクセスモードを選択し、作成されるストレージ要求の読み取り/書き込みアクセスを決定します。
 - d. ストレージ要求のサイズを定義します。
4. **Create** をクリックし、PersistentVolumeClaim を作成し、PersistentVolume を生成します。

3.15.2.2. CLI を使用した VMware vSphere ボリュームの動的プロビジョニング

OpenShift Container Platform は、ボリュームをプロビジョニングするために **thin** ディスク形式を使用する **thin** という名前のデフォルトの StorageClass をインストールします。

前提条件

- ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

手順 (CLI)

1. 以下の内容でファイル **pvc.yaml** を作成して VMware vSphere PersistentVolumeClaim を定義できます。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc ❶
spec:
  accessModes:
    - ReadWriteOnce ❷
  resources:
    requests:
      storage: 1Gi ❸
```


- ❶ PersistentVolumeClaim を表す一意の名前。
- ❷ PersistentVolumeClaim のアクセスモード。**ReadWriteOnce** では、ボリュームは単一ノードによって読み取り/書き込みパーミッションでマウントできます。
- ❸ PersistentVolumeClaim のサイズ。

2. ファイルから PersistentVolumeClaim を作成します。

```
$ oc create -f pvc.yaml
```

3.15.3. VMware vSphere ボリュームの静的プロビジョニング

VMware vSphere ボリュームを静的にプロビジョニングするには、永続ボリュームフレームワークが参照する仮想マシンディスクを作成する必要があります。

前提条件

- ストレージは、ボリュームとして OpenShift Container Platform にマウントされる前に基礎となるインフラストラクチャーになければなりません。

手順

- 仮想マシンディスクを作成します。VMware vSphere ボリュームを静的にプロビジョニングする前に、仮想マシンディスク (VMDK) を手動で作成する必要があります。以下の方法のいずれかを使用します。

- vmkfstools** を使用して作成します。セキュアシェル (SSH) を使用して ESX にアクセスし、以下のコマンドを使用して vmdk ボリュームを作成します。

```
$ vmkfstools -c <size> /vmfs/volumes/<datastore-name>/volumes/<disk-name>.vmdk
```

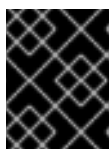
- vmware-diskmanager** を使用して作成します。

```
$ shell vmware-vdiskmanager -c -t 0 -s <size> -a lsilogic <disk-name>.vmdk
```

- VMDK を参照する PersistentVolume を作成します。PersistentVolume オブジェクト定義を使用して **pv1.yaml** ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1 ❶
spec:
  capacity:
    storage: 1Gi ❷
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: ❸
    volumePath: "[datastore1] volumes/myDisk" ❹
    fsType: ext4 ❺
```

- ❶ ボリュームの名前。この名前は PersistentVolumeClaim または Pod で識別されるものです。
- ❷ このボリュームに割り当てられるストレージの量。
- ❸ vSphere ボリュームの **vsphereVolume** で使用されるボリュームタイプ。ラベルは vSphere VMDK ボリュームを Pod にマウントするために使用されます。ボリュームの内容はアンマウントされても保持されます。このボリュームタイプは、VMFS データストアと VSAN データストアの両方がサポートされます。
- ❹ 使用する既存の VMDK ボリューム。 **vmkfstools** を使用した場合、前述のようにボリューム定義で、データストア名を角かっこ `[]` で囲む必要があります。
- ❺ マウントするファイルシステムタイプです。ext4、xfs、または他のファイルシステムなどが例になります。



重要

ボリュームをフォーマットしてプロビジョニングした後に fsType パラメーターの値を変更すると、データ損失や Pod にエラーが発生する可能性があります。

3. ファイルから PersistentVolume を作成します。

```
$ oc create -f pv1.yaml
```

4. 直前の手順で作成した PersistentVolume にマップする PersistentVolumeClaim を作成します。 PersistentVolumeClaim オブジェクト定義を使用して、ファイル **pvc1.yaml** を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1 ❶
spec:
  accessModes:
    - ReadWriteOnce ❷
  resources:
    requests:
      storage: "1Gi" ❸
  volumeName: pv1 ❹
```

- ❶ PersistentVolumeClaim を表す一意の名前。
- ❷ PersistentVolumeClaim のアクセスモード。ReadWriteOnce では、ボリュームは単一ノードによって読み取り/書き込みパーミッションでマウントできます。
- ❸ PersistentVolumeClaim のサイズ。
- ❹ 既存の PersistentVolume の名前。

5. ファイルから PersistentVolumeClaim を作成します。

```
$ oc create -f pvc1.yaml
```

3.15.3.1. VMware vSphere ボリュームのフォーマット

OpenShift Container Platform は、ボリュームをマウントしてコンテナに渡す前に、PersistentVolume (PV) 定義の **fsType** パラメーター値で指定されたファイルシステムがボリュームに含まれることを確認します。デバイスが指定されたファイルシステムでフォーマットされていない場合、デバイスのデータはすべて消去され、デバイスはそのファイルシステムで自動的にフォーマットされます。

OpenShift Container Platform は初回の使用前にフォーマットするため、フォーマットされていない vSphere ボリュームを PV として使用できます。

第4章 永続ボリュームの拡張

4.1. ボリューム拡張サポートの有効化

永続ボリュームを拡張する前に、StorageClass では **allowVolumeExpansion** フィールドを **true** に設定している必要があります。

手順

- StorageClass を編集し、**allowVolumeExpansion** 属性を追加します。以下の例では、StorageClass の設定の下部にこの行を追加する方法を示しています。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
...
parameters:
  type: gp2
reclaimPolicy: Delete
allowVolumeExpansion: true ❶
```

- ❶ この属性を **true** に設定すると、PVC を作成後に拡張することができます。

4.2. 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。

OpenShift Container Platform はデフォルトで CSI ボリューム拡張をサポートします。ただし、特定の CSI ドライバーが必要です。

OpenShift Container Platform には CSI ドライバーが含まれていません。[コミュニティまたはストレージベンダー](#) が提供する CSI ドライバーを使用することが推奨されます。CSI ドライバーの提供される手順に従います。

OpenShift Container Platform 4.3 は、[CSI 仕様](#) のバージョン 1.1.0 をサポートします。

重要

CSI ボリュームの拡張は、テクノロジープレビュー機能としてのみ利用できます。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

4.3. サポートされているドライバーでの FLEXVOLUME の拡張

FlexVolume を使用してバックエンドストレージシステムに接続する場合は、永続ストレージボリュームを作成後に拡張することができます。これは、OpenShift Container Platform で Persistent Volume Claim (永続ボリューム要求、PVC) を手動で更新して実行できます。

FlexVolume は、ドライバーが **RequiresFSResize** が **true** の状態で設定されている場合に拡張を許可します。FlexVolume は、Pod の再起動時に拡張できます。

他のボリュームタイプと同様に、FlexVolume ボリュームは Pod によって使用される場合にも拡張できます。

前提条件

- 基礎となるボリュームドライバーがサイズ変更をサポートする。
- ドライバーは **RequiresFSResize** 機能が **true** の状態で設定されている。
- 動的プロビジョニングが使用される。
- 制御する側の StorageClass には **allowVolumeExpansion** が **true** に設定されている。

手順

- FlexVolume プラグインのサイズ変更を使用するには、以下の方法で **ExpandableVolumePlugin** インターフェースを実装する必要があります。

RequiresFSResize

true の場合、容量を直接更新します。**false** の場合、**ExpandFS** メソッドを呼び出し、ファイルシステムのサイズ変更を終了します。

ExpandFS

true の場合、**ExpandFS** を呼び出し、物理ボリュームの拡張の実行後にファイルシステムのサイズを変更します。ボリュームドライバーは、ファイルシステムのサイズ変更と共に物理ボリュームのサイズ変更も実行できます。



重要

OpenShift Container Platform はマスターノードへの FlexVolume プラグインのインストールをサポートしないため、FlexVolume のコントロールプレーンの拡張をサポートしません。

4.4. ファイルシステムを使用した PERSISTENT VOLUME CLAIM (永続ボリューム要求、PVC) の拡張

ファイルサイズのサイズ変更を必要とするボリュームタイプ(GCE PD、EBS、および Cinder など)に基づいて PVC を拡張するには2つの手順からなるプロセスが必要です。このプロセスでは、クラウドプロバイダーでボリュームオブジェクトを拡張してから実際のノードで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 では、**allowVolumeExpansion** が **true** に設定されている必要があります。

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。以下のコマンドは、状態を確認するために使用されます。

```
$ 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、永続ボリュームオブジェクトは **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することができます。Pod が実行されている場合、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。

4.5. 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、OpenShift Container Platform 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、サイズ変更要求が管理者の介入なしに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。これは後ほど再作成されます。
3. 新規に作成された PVC が **Retain** というマークが付けられた PV にバインドされるには、PV を手動で編集し、PV 仕様から **claimRef** エントリーを削除します。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5章 動的プロビジョニング

5.1. 動的プロビジョニングについて

StorageClass リソースオブジェクトは、要求可能なストレージを記述し、分類するほか、動的にプロビジョニングされるストレージのパラメーターを要求に応じて渡すための手段を提供します。

StorageClass オブジェクトは、さまざまなレベルのストレージとストレージへのアクセスを制御するための管理メカニズムとしても機能します。クラスター管理者 (**cluster-admin**) またはストレージ管理者 (**storage-admin**) は、ユーザーが基礎となるストレージボリュームソースに関する詳しい知識がなくても要求できる StorageClass オブジェクトを定義し、作成します。

OpenShift Container Platform の永続ボリュームフレームワークはこの機能を有効にし、管理者がクラスターに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。このフレームワークにより、ユーザーは基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。

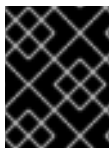
OpenShift Container Platform では、数多くのストレージタイプを永続ボリュームとして使用できます。これらはすべて管理者によって静的にプロビジョニングされますが、一部のストレージタイプは組み込みプロバイダーとプラグイン API を使用して動的に作成できます。

5.2. 利用可能な動的プロビジョニングプラグイン

OpenShift Container Platform は、以下のプロビジョナープラグインを提供します。これらには、クラスターの設定済みプロバイダーの API を使用して新規ストレージリソースを作成する動的プロビジョニング用の一般的な実装が含まれます。

ストレージタイプ	プロビジョナープラグインの名前	注記
Red Hat OpenStack Platform (RHOSP) Cinder	kubernetes.io/cinder	
AWS Elastic Block Store (EBS)	kubernetes.io/aws-ebs	複数クラスターを複数の異なるゾーンで使用する際の動的プロビジョニングの場合、各ノードに Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id> のタグを付けます。ここで、 <cluster_name> および <cluster_id> はクラスターごとに固有の値になります。
AWS Elastic File System (EFS)		動的プロビジョニングは、EFS プロビジョナー Pod を介して行われ、プロビジョナープラグインでは実行されません。
Azure Disk	kubernetes.io/azure-disk	

ストレージタイプ	プロビジョナープラグインの名前	注記
Azure File	kubernetes.io/azure-file	persistent-volume-binder ServiceAccount では、Azure ストレージアカウントおよびキーを保存するためにシークレットを作成し、取得するためのパーミッションが必要です。
GCE Persistent Disk (gcePD)	kubernetes.io/gce-pd	マルチゾーン設定では、GCE プロジェクトごとに OpenShift Container Platform クラスターを実行し、現行クラスターのノードが存在しないゾーンで PV が作成されないようにすることが推奨されます。
VMware vSphere	kubernetes.io/vsphere-volume	



重要

選択したプロビジョナープラグインでは、関連するクラウド、ホスト、またはサードパーティープロバイダーを、関連するドキュメントに従って設定する必要があります。

5.3. STORAGECLASS の定義

現時点で、StorageClass オブジェクトはグローバルスコープオブジェクトであり、**cluster-admin** または **storage-admin** ユーザーによって作成される必要があります。



重要

ClusterStorageOperator は、使用されるプラットフォームによってデフォルトの StorageClass をインストールする可能性があります。この StorageClass は Operator によって所有され、制御されます。アノテーションとラベルを定義するほかは、これを削除したり、変更したりすることはできません。異なる動作が必要な場合は、カスタム StorageClass を定義する必要があります。

以下のセクションでは、StorageClass の基本オブジェクトの定義とサポートされている各プラグインタイプの具体的な例について説明します。

5.3.1. 基本 StorageClass オブジェクト定義

以下のリソースは、StorageClass を設定するために使用するパラメーターおよびデフォルト値を示しています。この例では、AWS ElasticBlockStore (EBS) オブジェクト定義を使用します。

StorageClass 定義例

```
kind: StorageClass ❶
```

```

apiVersion: storage.k8s.io/v1 ❷
metadata:
  name: gp2 ❸
  annotations: ❹
    storageclass.kubernetes.io/is-default-class: 'true'
  ...
provisioner: kubernetes.io/aws-ebs ❺
parameters: ❻
  type: gp2
  ...

```

- ❶ (必須) API オブジェクトタイプ。
- ❷ (必須) 現在の apiVersion。
- ❸ (必須) StorageClass の名前。
- ❹ (オプション) StorageClass のアノテーション
- ❺ (必須) このストレージクラスに関連付けられているプロビジョナーのタイプ。
- ❻ (オプション) 特定のプロビジョナーに必要なパラメーター。これはプラグインによって異なります。

5.3.2. StorageClass のアノテーション

StorageClass をクラスター全体のデフォルトとして設定するには、以下のアノテーションを StorageClass のメタデータに追加します。

```
storageclass.kubernetes.io/is-default-class: "true"
```

以下は例になります。

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  ...

```

これにより、特定のボリュームを指定しない Persistent Volume Claim (永続ボリューム要求、PVC) がデフォルト StorageClass によって自動的にプロビジョニングされるようになります。



注記

ベータアノテーションの **storageclass.beta.kubernetes.io/is-default-class** は依然として使用可能ですが、今後のリリースで削除される予定です。

StorageClass の記述を設定するには、以下のアノテーションを StorageClass のメタデータに追加します。

```
kubernetes.io/description: My StorageClass Description
```

以下は例になります。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My StorageClass Description
...
```

5.3.3. RHOSP Cinder オブジェクトの定義

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast ❶
  availability: nova ❷
  fsType: ext4 ❸
```

- ❶ Cinder で作成されるボリュームタイプ。デフォルトは空です。
- ❷ アベイラビリティゾーン。指定しない場合、ボリュームは通常 OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。
- ❸ 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

5.3.4. AWS Elastic Block Store (EBS) オブジェクト定義

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 ❶
  iopsPerGB: "10" ❷
  encrypted: "true" ❸
  kmsKeyId: keyvalue ❹
  fsType: ext4 ❺
```

- ❶ (必須) **io1**、**gp2**、**sc1**、**st1** から選択します。デフォルトは **gp2** です。有効な Amazon Resource Name (ARN) 値については、[AWS のドキュメント](#) を参照してください。

- 2 (オプション) **io1** ボリュームのみ。1 GiB あたり 1 秒あたりの I/O 処理数。AWS ボリュームプラグインは、この値と要求されたボリュームのサイズを乗算してボリュームの IOPS を算出します。値
- 3 (オプション) EBS ボリュームを暗号化するかどうかを示します。有効な値は **true** または **false** です。
- 4 (オプション) ボリュームを暗号化する際に使用するキーの完全な ARN。値を指定しない場合でも **encrypted** が **true** に設定されている場合は、AWS によってキーが生成されます。有効な ARN 値については、[AWS のドキュメント](#) を参照してください。
- 5 (オプション) 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

5.3.5. Azure Disk オブジェクト定義

azure-advanced-disk-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: azure_storage_account_name 1
  storageaccounttype: Standard_LRS 2
  kind: Dedicated 3
```

- 1 Azure ストレージアカウントの名前。これはクラスターと同じリソースグループに存在している必要があります。ストレージアカウントを指定した場合、**location** は無視されます。ストレージアカウントを指定しない場合、新しいストレージアカウントがクラスターと同じリソースグループに作成されます。**storageAccount** を指定する場合は、**kind** の値は **Dedicated** でなければなりません。
- 2 Azure ストレージアカウントの SKU の層。デフォルトは空です。プレミアム VM は **Standard_LRS** ディスクと **Premium_LRS** ディスクの両方を割り当て、標準 VM は **Standard_LRS** ディスクのみを、マネージド VM はマネージドディスクのみを、アンマネージド VM はアンマネージドディスクのみを割り当てることができます。
- 3 許容値は、**Shared** (デフォルト)、**Dedicated** および **Managed** です。
 - a. **kind** が **Shared** に設定されている場合は、Azure は、クラスターと同じリソースグループにあるいくつかの共有ストレージアカウントで、アンマネージドディスクをすべて作成します。
 - b. **kind** が **Managed** に設定されている場合は、Azure は新しいマネージドディスクを作成します。
 - c. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されている場合には、Azure は、クラスターと同じリソースグループ内にある新規のアンマネージドディスク用に、指定のストレージアカウントを使用します。これを機能させるには、以下が前提となります。
 - 指定のストレージアカウントが、同じリージョン内にあること。

- Azure Cloud Provider にストレージアカウントへの書き込み権限があること。
- d. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されていない場合には、Azure はクラスターと同じリソースグループ内の新規のアンマネージドディスク用に、新しい専用のストレージアカウントを作成します。

5.3.6. Azure File のオブジェクト定義

Azure File StorageClass はシークレットを使用して Azure ストレージアカウント名と Azure ファイル共有の作成に必要なストレージアカウントキーを保存します。これらのパーミッションは、以下の手順の一部として作成されます。

手順

1. シークレットの作成および表示を可能にする ClusterRole を定義します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> ❶
rules:
- apiGroups: [""]
  resources: ['secrets']
  verbs: ['get','create']
```

- ❶ シークレットを表示し、作成するための ClusterRole の名前。

2. ClusterRole を ServiceAccount に追加します。

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
system:serviceaccount:kube-system:persistent-volume-binder
```

3. Azure File StorageClass を作成します。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> ❶
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus ❷
  skuName: Standard_LRS ❸
  storageAccount: <storage-account> ❹
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

- ❶ StorageClass の名前。PersistentVolumeClaim は、関連する PersistentVolume をプロビジョニングするためにこの StorageClass を使用します。

- ❷

eastus などの Azure ストレージアカウントの場所。デフォルトは空であり、新規 Azure ストレージアカウントが OpenShift Container Platform クラスターの場所に作成されま

- ③ SKU は、**Standard_LRS** などの Azure ストレージアカウントの層になります。デフォルトは空です。つまり、新しい Azure ストレージアカウントは **Standard_LRS** SKU で作成されます。
- ④ Azure ストレージアカウントの名前。ストレージアカウントが提供されると、**skuName** および **location** は無視されます。ストレージアカウントを指定しない場合、StorageClass は、定義された **skuName** と **location** に一致するアカウントのリソースグループに関連付けられたストレージアカウントを検索します。

5.3.6.1. Azure File を使用する場合の考慮事項

以下のファイルシステム機能は、デフォルトの Azure File StorageClass ではサポートされません。

- シンボリックリンク
- ハードリンク
- 拡張属性
- スパースファイル
- 名前付きパイプ

また、Azure File がマウントされるディレクトリーの所有者 ID (UID) は、コンテナのプロセス UID とは異なります。**uid** マウントオプションは StorageClass に指定して、マウントされたディレクトリーに使用する特定のユーザー ID を定義できます。

以下の StorageClass は、マウントされたディレクトリーのシンボリックリンクを有効にした状態で、ユーザーおよびグループ ID を変更する方法を示しています。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
mountOptions:
  - uid=1500 ①
  - gid=1500 ②
  - mfsymlinks ③
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

- ① マウントされたディレクトリーに使用するユーザー ID を指定します。
- ② マウントされたディレクトリーに使用するグループ ID を指定します。
- ③ シンボリックリンクを有効にします。

5.3.7. GCE PersistentDisk (gcePD) オブジェクトの定義

gce-pd-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard ❶
  replication-type: none
```

- ❶ **pd-standard** または **pd-ssd** のいずれかを選択します。デフォルトは **pd-ssd** です。

5.3.8. VMWare vSphere オブジェクトの定義

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume ❶
parameters:
  diskformat: thin ❷
```

- ❶ OpenShift Container Platform で VMware vSphere を使用方法の詳細については、[VMware vSphere のドキュメント](#) を参照してください。
- ❷ **diskformat: thin**、**zeroedthick** および **eagerzeroedthick** はすべて有効なディスクフォーマットです。ディスクフォーマットの種類に関する詳細は、vSphere のドキュメントを参照してください。デフォルト値は **thin** です。

5.3.9. Red Hat OpenShift Container Storage オブジェクト定義

Red Hat OpenShift Container Storage を使用する場合、動的ボリュームプロビジョニングのストレージクラスは、[ストレージクラスの作成と一覧表示の確認](#) について説明されているように、Red Hat OpenShift Container Storage 4.3 が Operator Hub からデプロイされる際に作成されます。

5.4. デフォルト STORAGECLASS の変更

AWS を使用している場合は、以下のプロセスを使用してデフォルトの StorageClass を変更します。このプロセスでは、**gp2** と **standard** の 2 つの StorageClass が定義されており、デフォルトの StorageClass を **gp2** から **standard** に変更する必要がある場合を想定しています。

1. StorageClass の一覧を表示します。

```
$ oc get storageclass
```

NAME	TYPE
------	------

gp2 (default)	kubernetes.io/aws-ebs 1
standard	kubernetes.io/aws-ebs

1 **(default)** はデフォルトの StorageClass を示します。

- デフォルトの StorageClass のアノテーション **storageclass.kubernetes.io/is-default-class** の値を **false** に変更します。

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

- アノテーション **storageclass.kubernetes.io/is-default-class=true** を追加するか、このアノテーションを変更して別の StorageClass をデフォルトにします。

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

- 変更内容を確認します。

```
$ oc get storageclass
```

NAME	TYPE
gp2	kubernetes.io/aws-ebs
standard (default)	kubernetes.io/aws-ebs