OpenShift Container Platform 4.1

ストレージ

OpenShift Container Platform 4.1 でのストレージの設定および管理

Last Updated: 2020-05-15

OpenShift Container Platform 4.1 ストレージ

OpenShift Container Platform 4.1 でのストレージの設定および管理

法律上の通知

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 永続ストレージについて

1.1. 永続ストレージの概要

ストレージの管理は、コンピュート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。

1.2. 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。

1.2.1. 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。

1.2.2. 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。

PV ボリュームは、要求したボリュームを上回る可能性がありますが、これは、手動でプロビジョニングされた PV の場合にとくにそう言えます。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。

1.2.3. 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 のスケジュールおよび要求された PV のアクセスは、**persistentVolumeClaim** を Pod のボリュームブロックに組み込んで実行できます。

1.2.4. 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。

1.2.5. ボリュームの解放

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

1.2.6. ボリュームの回収

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アセットの両方を削除します。

注記

動的にプロビジョニングされたボリュームは常に削除されます。

1.3. 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 (1)
spec:
  capacity:
    storage: 5Gi (2)
  accessModes:
    - ReadWriteOnce (3)
  persistentVolumeReclaimPolicy: Retain (4)
  ...
status:
  ...
```

(1) 永続ボリュームの名前。

(2) ボリュームに利用できるストレージの量。

(3) 読み取り書き込みおよびマウントパーミッションを定義するアクセスモード。

(4) リソースのリリース後にそれらのリソースがどのように処理されるかを示す回収ポリシー。

1.3.1. 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)
- ファイバーチャネル
- HostPath
- iSCSI
- NFS
- VMware vSphere

1.3.2. 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。

1.3.3. 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。

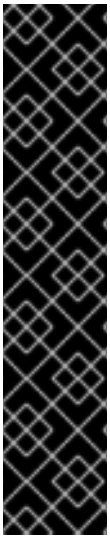
同じモードのボリュームはすべて分類され、サイズ別 (一番小さいものから一番大きいもの順) に分類されます。バインダーは一致するモードのグループを取得し、1 つのサイズが一致するまでそれぞれを (サイズの順序で) 繰り返し処理します。

以下の表では、アクセスモードをまとめています。

表1.1 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。

アクセスモード	CLI の省略形	説明
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

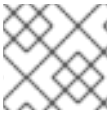
ボリュームの **AccessModes** は、ボリュームの機能の記述子です。それらは施行されている制約ではありません。ストレージプロバイダーはリソースの無効な使用から生じるランタイムエラーに対応します。

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1.2 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	ReadOnlyMany	ReadWriteMany
AWS EBS	▮	-	-
ファイバーチャネル	▮	▮	-
HostPath	▮	-	-
iSCSI	▮	▮	-
NFS	▮	▮	▮
VMware vSphere	▮	-	-



注記

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。

1.3.4. 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1.3 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>
```

1.3.4.1. 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 (1)
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

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

以下の PV タイプがマウントオプションをサポートします。

- AWS Elastic Block Store (EBS)
- iSCSI
- NFS
- 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。

1.4. 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 (1)
spec:
  accessModes:
    - ReadWriteOnce (2)
  resources:
    requests:
      storage: 8Gi (3)
  storageClassName: gold (4)
status:
  ...
```

1. PVC の名前
2. 読み取り書き込みおよびマウントパーミッションを定義するアクセスモード
3. PVC に利用できるストレージの量
4. 要求で必要になる **StorageClass** の名前

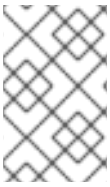
1.4.1. ストレージクラス

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。

1.4.2. 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。

1.4.3. 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。

1.4.4. ボリュームとしての要求

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

ホストおよび Pod のサンプルへのボリュームのマウント

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: dockerfile/nginx
      volumeMounts:
      - mountPath: "/var/www/html" 1
        name: mypd 2
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim 3
```

1. Pod 内にボリュームをマウントするためのパス
2. マウントするボリュームの名前
3. 使用する同じ namespace にある PVC の名前

1.5. ブロックボリュームのサポート

PV および PVC 仕様に　API フィールドを組み込み、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。

重要

raw ブロックボリュームを使用する Pod は、特権付きコンテナー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1.4 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
AWS EBS	▮	▮
ファイバーチャネル	▮	
HostPath		
iSCSI	▮	
NFS		
VMware vSphere	▮	▮

重要

ブロックボリュームのサポートはテクノロジープレビュー機能でのみ利用可能です。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、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

1.5.1. 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** フィールドは、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 (1)
  resources:
    requests:
      storage: 10Gi
```

(1) **volumeMode** フィールドは、raw ブロックの 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: (1)
        - name: data
          devicePath: /dev/xvda (2)
  volumes:
```

```
 - name: data
   persistentVolumeClaim:
     claimName: block-pvc 3
```

1. **volumeMounts** と同様に **volumeDevices** はブロックデバイスに使用され、**PersistentVolumeClaim** ソースでのみ使用できます。
2. **mountPath** と同様に **devicePath** は、物理デバイスへのパスを表します。
3. 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1.5 VolumeMode の許容値

値	デフォルト
Filesystem	Yes
Block	No

表1.6 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
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2章 永続ストレージの設定

2.1. 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

2.1.1. 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。

2.1.2. 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。

2.1.3. 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ーに渡す前に、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。

2.1.4. 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。

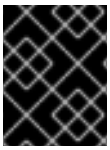
OpenShift Container Platform では、環境変数 **KUBE_MAX_PD_VOLS** を設定することで、より大きな制限値を設定できます。ただし、AWS では、割り当てられるデバイスに特定の命名スキーム (AWS デバイスの命名を使用する必要があります。この命名スキームでは、最大で 52 のボリュームしかサポートされません。これにより、OpenShift Container Platform 経由でノードに割り当てることができるボリュームの数が 52 に制限されます。

2.2. 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。

その他の参考資料

- ファイバーチャネル

2.2.1. 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'] 1
    lun: 2
    fsType: ext4
```

1 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。

2.2.1.1. 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。

2.2.1.2. ファイバーチャネル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 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。

2.3. 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)

2.3.1. 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 1
spec:
  capacity:
    storage: 5Gi 2
  accessModes:
```

```
  - ReadWriteOnce 3
  nfs: 4
    path: /tmp 5
    server: 172.17.0.2 6
  persistentVolumeReclaimPolicy: Retain 7
```

1. ボリュームの名前。これは、各種の **oc <command> pod** コマンドの PV アイデンティティーです。
2. このボリュームに割り当てられるストレージの量。
3. これはボリュームへのアクセスの制御に関連するように見えますが、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。
4. 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。
5. NFS サーバーがエクスポートしているパス。
6. NFS サーバーのホスト名または IP アドレス
7. 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 1
  resources:
    requests:
      storage: 5Gi 2
```

1. 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。
2. 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
```

2.3.2. 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。

2.3.3. 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。

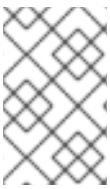
NFS サーバーの **/etc/exports** ファイルにはアクセス可能な NFS ディレクトリーが含まれています。ターゲットの NFS ディレクトリーには、POSIX の所有者とグループ ID があります。OpenShift Container Platform NFS プラグインは、同じ POSIX の所有者とエクスポートされる NFS ディレクトリーにあるパーミッションを使って、コンテナーの NFS ディレクトリーをマウントします。ただし、コンテナーは NFS マウントの所有者と同等の有効な UID では実行されません。 これは期待される動作です。

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、コンテナーは 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。



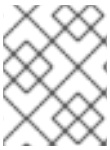
注記

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。

2.3.3.1. 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。

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: 1
    supplementalGroups: [5555] 2
```

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

2 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。

2.3.3.2. ユーザー ID

ユーザー ID は、コンテナーイメージまたは 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、コンテナーは 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: 1
  - name:
  ...
    securityContext:
      runAsUser: 65534 2
```

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

2 **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。

2.3.3.3. 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
```

2.3.3.4. エクスポート設定

任意のコンテナー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ーのプライマリー 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。

2.3.4. 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と、エラーが発生し、データが失われる可能性があります。

2.3.5. 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。	• NFS の ID マッピング設定 (**/etc/idmapd.conf**) に原因がある可能性が高い。 • この Red Hat ソリューション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	• 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。 `# echo 'Y' > /sys/module/nfsd/parameters/nfs4_disable_idmapping`

2.4. 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。

2.4.1. 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。

2.4.2. 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
```

2.4.3. 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 Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-name (1)
spec:
  containers:
    ...
    securityContext:
      privileged: true (2)
    volumeMounts:
    - mountPath: /data (3)
      name: hostpath-privileged
  ...
  securityContext: {}
```

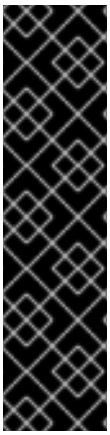
```
volumes:
  - name: hostpath-privileged
    persistentVolumeClaim:
      claimName: task-pvc-volume 4
```

1. Pod の名前。
2. 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。
3. 特権付き Pod 内に hostPath 共有をマウントするパス。
4. 以前に作成された PersistentVolumeClaim の名前。

2.5. 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。



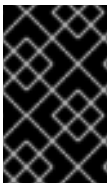
重要

iSCSI を使用した永続ストレージ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、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

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。

2.5.1. 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 に必要になるのは、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2.1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'
```

2.5.2. 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。

2.5.3. 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。

2.5.3.1. 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 (1)
    chapAuthSession: true (2)
    secretRef:
      name: chap-secret (3)
```

1. iSCSI 検出の CHAP 認証を有効にします。
2. iSCSI セッションの CHAP 認証を有効にします。
3. 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。

2.5.4. 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'] (1)
    iqn: iqn.2016-04.test.com:storage.target00
    lun: 0
    fsType: ext4
    readOnly: false
```

1. **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。

2.5.5. 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。

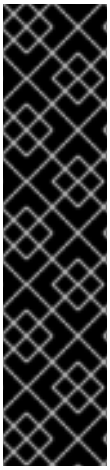
カスタムのイニシエーター IQN を指定するには、**initiatorName** フィールドを使用します。

```
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 (1)
    fsType: ext4
    readOnly: false
```

(1) イニシエーターの名前を指定します。

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

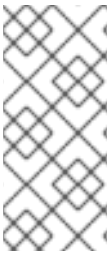
Container Storage Interface (CSI) により、OpenShift Container Platform は CSI インターフェース を永続ストレージとして実装するストレージバックエンドからストレージを使用できます。



重要

Container Storage Interface はテクノロジープレビュー機能でのみ利用可能です。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、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。



注記

OpenShift Container Platform には CSI ドライバーが含まれていません。コミュニティー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。

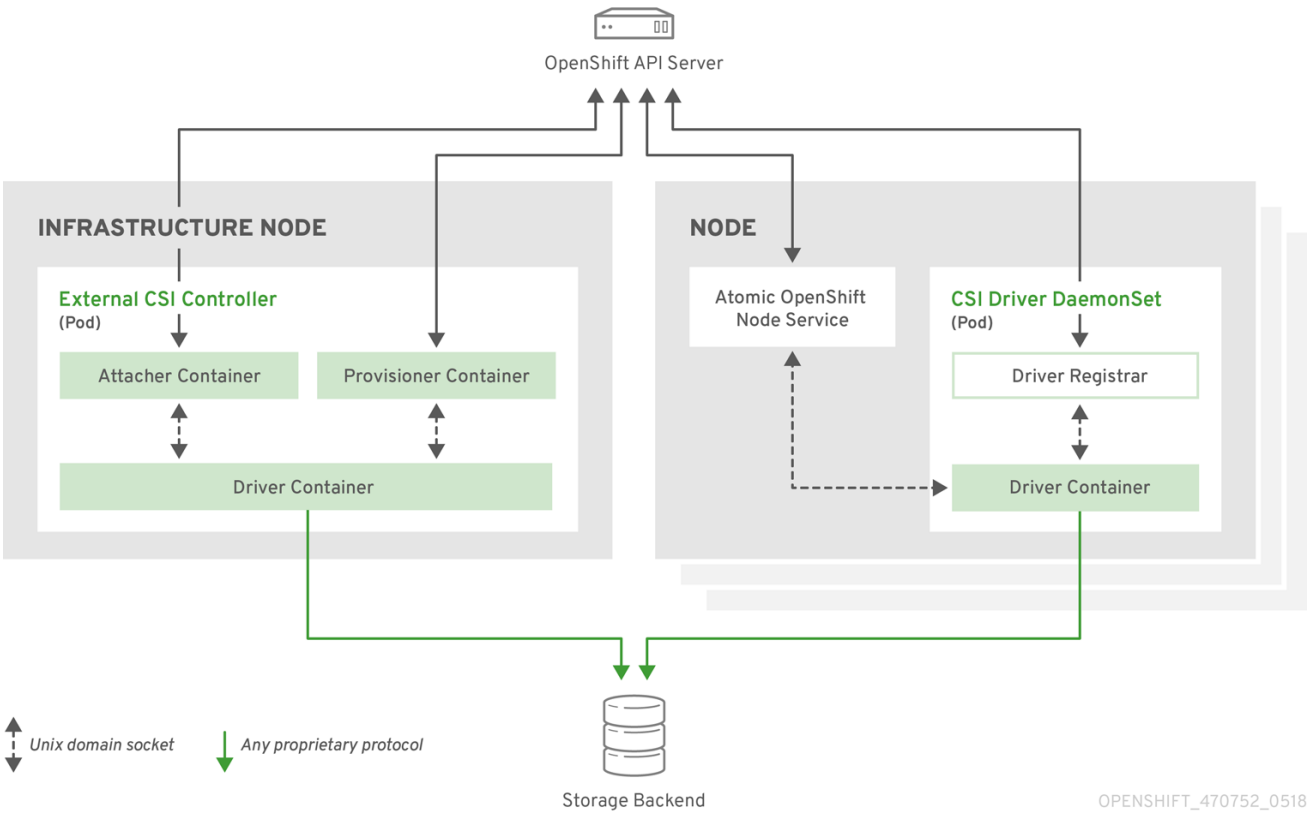
OpenShift Container Platform 4.1 は、CSI 仕様のバージョン 1.0.0 をサポートします。

2.6.1. CSI アーキテクチャー

CSI ドライバーは通常、コンテナーイメージとして提供されます。これらのコンテナーは、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。



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。

2.6.1.1. 外部の CSI コントローラー

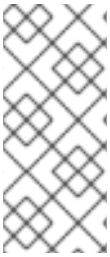
外部の CSI コントローラーは、3 つのコンテナーを含む 1 つまたは複数の Pod を配置するデプロイメントです。

- 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ー。
- OpenShift Container Platform からの **provision** および **delete** 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ー。
- CSI ドライバーコンテナー

CSI アタッチャーおよび CSI プロビジョナーコンテナーは、Unix Domain Socket を使用して、CSI ドライバーコンテナーと通信し、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、コンピュートノードで致命的なセキュリティー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** 操作を実行しません。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。

2.6.1.2. 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、以下のコンテナーが含まれます。

- 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。

2.6.2. 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じ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。

OpenStack Cinder の例に示されるように、動的プロビジョニングを有効にするためにこの StorageClass をデプロイできます。

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: cinder
  annotations:
```

```
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-cinderplugin
parameters:
EOF
```

2.6.3. 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
```

2.7. VMWARE VSPHERE ボ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。

その他の参考資料

- VMware vSphere

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

VMware vSphere ボ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。

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。『クラスターの vSphere へのインストール』を参照してください。

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ュームを動的にプロビジョニングできます。

2.7.1.1. UI 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

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。

2.7.1.2. 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 (1)
spec:
  accessModes:
  - ReadWriteOnce (2)
  resources:
    requests:
      storage: 1Gi (3)
```

(1) PersistentVolumeClaim を表す一意の名前。

(2) 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。

(3) PersistentVolumeClaim のサイズ。

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

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

2.7.2. 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

1. 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/DatastoreName/volumes/<disk-name>.vmdk
     ```

   - **vmware-diskmanager** を使用して作成します。

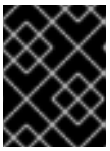
     ```
     $ shell vmware-vdiskmanager -c -t 0 -s <size> -a lsilogic <disk-name>.vmdk
     ```

2. 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 **pv.yaml** ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv (1)
```

```
spec:
  capacity:
    storage: 2Gi (2)
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  vsphereVolume: (3)
    volumePath: "[datastore1] volumes/myDisk" (4)
    fsType: ext4 (5)
```

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. このボリュームに割り当てられるストレージの量。
3. 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。
4. 使用する既存の VMDK ボリューム。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。
5. 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 pv.yaml
```

2.7.2.1. 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ーに渡す前に、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。

2.8. ボリュームスナップショットを使用した永続ストレージ

本書では、OpenShift Container Platform で VolumeSnapshot を使用してデータ損失から保護する方法を説明します。永続ボリュームについてある程度理解していることが推奨されます。

重要

ボリュームのスナップショット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、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

2.8.1. スナップショット

ボリュームスナップショットは、クラスター内のストレージボリュームから作成されたスナップショットです。外部のスナップショットコントローラーおよびプロビジョナーは、この機能を OpenShift Container Platform クラスターで使用して OpenShift Container Platform API を使用してボリュームスナップショットを処理します。

ボリュームのスナップショットを使用して、クラスター管理者は以下を行うことができます。

- PersistentVolumeClaim にバインドされる PersistentVolume のスナップショットを作成します。
- 既存の VolumeSnapshot を一覧表示します。
- 既存の VolumeSnapshot を削除します。
- 既存の VolumeSnapshot から PersistentVolume を新たに作成します。

サポートされている PersistentVolume タイプ:

- AWS Elastic Block Store (EBS)
- Google Compute Engine (GCE) Persistent Disk (PD)

2.8.2. 外部のコントローラーおよびプロビジョナー

コントローラーおよびプロビジョナーは、ボリュームのスナップショットを提供します。これらの外部コンポーネントはクラスターで実行されます。

ボリュームのスナップショットを提供する外部コンポーネントが 2 つあります。

外部コントローラー

ボリュームスナップショットのイベントを作成、削除、および報告します。

外部プロビジョナー

VolumeSnapshot から新規の PersistentVolume を作成します。

外部のコントローラーおよびプロビジョナーサービスはコンテナーイメージとして配布され、OpenShift Container Platform クラスターで通常どおり実行できます。

2.8.2.1. 外部のコントローラーおよびプロビジョナーの実行

クラスター管理者は、外部コントローラーおよびプロビジョナーを実行するようにアクセスを設定する必要があります。

手順

API オブジェクトを管理しているコンテナーを許可するには、以下の手順を実行します。

1. ServiceAccount と ClusterRole を作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: snapshot-controller-runner
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: snapshot-controller-role
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: ["list", "watch", "create", "update", "patch"]
  - apiGroups: ["apiextensions.k8s.io"]
    resources: ["customresourcedefinitions"]
    verbs: ["create", "list", "watch", "delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
  - apiGroups: ["volumesnapshot.external-storage.k8s.io"]
    resources: ["volumesnapshotdatas"]
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

2. クラスター管理者として、**hostNetwork** SCC (security context constraint) を提供します。

```
# oc adm policy add-scc-to-user hostnetwork -z snapshot-controller-runner
```

この SCC は、Pod が使用している **snapshot-controller-runner** サービスアカウントへのアクセスを制御します。

3. ClusterRoleBinding でルールをバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: snapshot-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

```
  name: snapshot-controller-role
subjects:
- kind: ServiceAccount
  name: snapshot-controller-runner
  namespace: default (1)
```

(1) snapshot-controller が置かれているプロジェクト名を指定します。

2.8.2.2. AWS および GCE 認証

外部のコントローラーおよびプロビジョナーを認証するには、クラウドプロバイダーの管理者がシークレットを提供する必要があります。

2.8.2.2.1. AWS 認証

外部のコントローラーおよびプロビジョナーを Amazon Web Services (AWS) にデプロイしている場合、AWS はアクセスキーを使用して認証できる必要があります。

認証情報を Pod に提供するために、クラスター管理者は以下のように新規のシークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: awskeys
type: Opaque
data:
  access-key-id: <base64 encoded AWS_ACCESS_KEY_ID>
  secret-access-key: <base64 encoded AWS_SECRET_ACCESS_KEY>
```

重要

awskeys シークレットに必要な base64 値を生成する際に、以下のように末尾の改行文字を削除します。

```
$ echo -n "<aws_access_key_id>" | base64
$ echo -n "<aws_secret_access_key>" | base64
```

以下の例は、外部コントローラーおよびプロビジョナーコンテナーの AWS デプロイメントを表示しています。これら両方の Pod コンテナーはシークレットを使用して AWS API にアクセスします。

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
```

```
      app: snapshot-controller
  spec:
    serviceAccountName: snapshot-controller-runner
    hostNetwork: true
    containers:
      - name: snapshot-controller
        image: "registry.redhat.io/openshift3/snapshot-controller:latest"
        imagePullPolicy: "IfNotPresent"
        args: ["-cloudprovider", "aws"]
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                name: awskeys
                key: access-key-id
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                name: awskeys
                key: secret-access-key
      - name: snapshot-provisioner
        image: "registry.redhat.io/openshift3/snapshot-provisioner:latest"
        imagePullPolicy: "IfNotPresent"
        args: ["-cloudprovider", "aws"]
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                name: awskeys
                key: access-key-id
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                name: awskeys
                key: secret-access-key
```

2.8.2.2.2. GCE 認証

Google Compute Engine (GCE) の場合、GCE API にアクセスするためにシークレットを使用する必要はありません。

管理者は、以下の例で示すようにデプロイメントに進むことができます。

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: snapshot-controller
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: snapshot-controller
```

```
spec:
  serviceAccountName: snapshot-controller-runner
  containers:
    - name: snapshot-controller
      image: "registry.redhat.io/openshift3/snapshot-controller:latest"
      imagePullPolicy: "IfNotPresent"
      args: ["-cloudprovider", "gce"]
    - name: snapshot-provisioner
      image: "registry.redhat.io/openshift3/snapshot-provisioner:latest"
      imagePullPolicy: "IfNotPresent"
      args: ["-cloudprovider", "gce"]
```

2.8.2.3. スナップショットユーザーの管理

クラスターの設定によっては、管理者以外のユーザーが API サーバーで VolumeSnapshot オブジェクトを操作できるようにする必要があります。これは、特定のユーザーまたはグループにバインドされる ClusterRole を作成して実行できます。

たとえば、ユーザー「alice」がクラスター内のスナップショットを操作する必要があるとします。クラスター管理者は以下の手順を実行します。

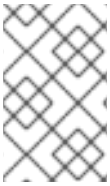
1. 新規の ClusterRole を定義します。

```
apiVersion: v1
kind: ClusterRole
metadata:
  name: volumesnapshot-admin
rules:
- apiGroups:
  - "volumesnapshot.external-storage.k8s.io"
  attributeRestrictions: null
  resources:
  - volumesnapshots
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
```

2. ClusterRole バインドオブジェクトを作成してクラスターロールをユーザー「alice」にバインドします。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: volumesnapshot-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: volumesnapshot-admin
```

```
subjects:
- kind: User
  name: alice
```



注記

これは API アクセス設定の一例にすぎません。VolumeSnapshot オブジェクトは他の OpenShift Container Platform API オブジェクトと同様に動作します。API RBAC の管理についての詳細は、API アクセス制御についてのドキュメントを参照してください。

2.8.3. スナップショットの作成および削除

Persistent Volume Claim (永続ボリューム要求、PVC) を永続ボリューム (PV) にバインドしてボリュームをプロビジョニングする方法と同様に、VolumeSnapshotData と VolumeSnapshot はボリュームスナップショットの作成に使用されます。

ボリュームスナップショットは、サポートされる PersistentVolume のタイプを使用する必要があります。

2.8.3.1. スナップショットの作成

PV のスナップショットを作成するには、以下の例のように VolumeSnapshot に基づいて VolumeSnapshotData オブジェクトを作成します。

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot (1)
metadata:
  name: snapshot-demo
spec:
  persistentVolumeClaimName: ebs-pvc (2)
```

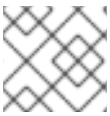
1. VolumeSnapshotData オブジェクトは VolumeSnapshot に基づいて自動的に作成されます。
2. **persistentVolumeClaimName** は、PersistentVolume にバインドされる PersistentVolumeClaim の名前です。この特定 PV のスナップショットが作成されます。

PV のタイプによっては、反映される VolumeSnapshot の状態に応じ、スナップショットの作成操作は複数の段階にわたる場合があります。

1. 新規 VolumeSnapshot オブジェクトを作成します。
2. コントローラーを起動します。スナップショット対象の PersistentVolume をフリーズし、アプリケーションを一時停止する必要が生じる場合があります。
3. スナップショットを作成します。スナップショット対象の PersistentVolume は通常の操作に戻りますが、スナップショット自体は準備状態ではありません(status=**True**、type=**Pending**)。
4. 実際のスナップショットを表す VolumeSnapshotData オブジェクトを作成します。
5. スナップショットが完了し、使用できる状態になります（status=**True**、type=**Ready**）。

重要

データの整合性はユーザーの責任で確保してください (Pod またはアプリケーションの停止、キャッシュのフラッシュ、ファイルシステムのフリーズなど)。



注記

エラーの場合は、VolumeSnapshot の状態にエラー状態が追加されます。

VolumeSnapshot の状態を表示するには、以下を実行します。

```
$ oc get volumesnapshot -o yaml
```

以下の例が示すように、ステータスが表示されます。

```
apiVersion: volumesnapshot.external-storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  clusterName: ""
  creationTimestamp: 2017-09-19T13:58:28Z
  generation: 0
  labels:
    Timestamp: "1505829508178510973"
  name: snapshot-demo
  namespace: default (1)
  resourceVersion: "780"
  selfLink: /apis/volumesnapshot.external-
storage.k8s.io/v1/namespaces/default/volumesnapshots/snapshot-demo
  uid: 9cc5da57-9d42-11e7-9b25-90b11c132b3f
spec:
  persistentVolumeClaimName: ebs-pvc
  snapshotDataName: k8s-volume-snapshot-9cc8813e-9d42-11e7-8bed-90b11c132b3f
status:
  conditions:
  - lastTransitionTime: null
    message: Snapshot created successfully
    reason: ""
    status: "True"
    type: Ready
  creationTimestamp: null
```

(1) snapshot-controller が置かれているプロジェクト名を指定します。

2.8.3.2. スナップショットの復元

PVC は、スナップショットの復元に使用されます。最初に、管理者は既存の VolumeSnapshot から PersistentVolume を復元するために StorageClass を作成する必要があります。

1. StorageClass を作成します。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
  name: snapshot-promoter
provisioner: volumesnapshot.external-storage.k8s.io/snapshot-promoter
parameters: (1)
  encrypted: "true"
  type: gp2
```

(1) **gp2 encryption** が設定された状態で AWS EBS ストレージを使用している場合、**encrypted** および **type** のパラメーターを設定する必要があります。

2. PVC を作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: snapshot-pv-provisioning-demo
  annotations:
    snapshot.alpha.kubernetes.io/snapshot: snapshot-demo (1)
spec:
  storageClassName: snapshot-promoter (2)
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi (3)
```

(1) 復元する VolumeSnapshot の名前。

(2) VolumeSnapshot を復元するために管理者によって作成されます。

(3) 復元されたスナップショットのストレージサイズは、元の PV サイズに対応するのに十分な大きさである必要があります。

新規の PersistentVolume が作成されて PersistentVolumeClaim にバインドされます。PV のタイプによっては処理に数分の時間がかかることがあります。

2.8.3.3. スナップショットの削除

VolumeSnapshot を削除するには、以下を実行します。

```
$ oc delete volumesnapshot/<snapshot-name>
```

VolumeSnapshot にバインドされている VolumeSnapshotData が自動的に削除されます。

第3章 永続ボリュームの拡張

3.1. 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 (1)
```

(1) 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。

3.2. 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。

3.3. 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4章 動的プロビジョニング

4.1. 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。

4.2. 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。

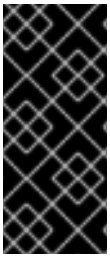
ストレージタイプ	プロビジョナープラグインの名前	備考
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	複数クラスターを複数の異なるゾーンで使用する際の動的プロビジョニングの場合、各ノードに **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>** のタグを付けます。 ここで、**<cluster_name>** および **<cluster_id>** はクラスターごとに固有の値になります。
VMware vSphere	**kubernetes.io/vsphere-volume**	

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もあります。

4.3. 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。

4.3.1. 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 (1)
apiVersion: storage.k8s.io/v1 (2)
metadata:
  name: gp2 (3)
  annotations: (4)
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs (5)
parameters: (6)
  type: gp2
...
```

(1) (必須) API オブジェクトタイプ。

(2) (必須) 現在の apiVersion。

(3) (必須) StorageClass の名前。

(4) (オプション) StorageClass のアノテーション

(5) (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。

(6) (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。

4.3.2. 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"
```

例:

```
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
```

例:

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

4.3.3. 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 (1)
  iopsPerGB: "10" (2)
  encrypted: "true" (3)
  kmsKeyId: keyvalue (4)
  fsType: ext4 (5)
```

1. (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。値の上限は、AWS でサポートされる最大値である 20,000 IOPS です。詳細については、AWS のドキュメントを参照してください。

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 **encypted** 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。

4.3.4. 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 1
parameters:
  diskformat: thin 2
```

1 OpenShift Container Platform で VMware vSphere を使用する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。

2 **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。

4.4. 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。

2. 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"}}}'
```

3. 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"}}}'
```

4. 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
```