OpenShift Container Platform 4.2

Container-native Virtualization

Container-native Virtualization のインストール、使用方法、およびリリースノート

Last Updated: 2020-06-18

OpenShift Container Platform 4.2 Container-native Virtualization

Container-native Virtualization のインストール、使用方法、およびリリースノート

法律上の通知

概要

本書では、OpenShift Container Platform 4.2 で Container-native Virtualization を使用する方法についての情報を提供します。

目次

第1章 CONTAINER-NATIVE VIRTUALIZATION インストール

1.1. CONTAINER-NATIVE VIRTUALIZATION について

Container-native Virtualization の機能およびサポート範囲について確認します。

1.1.1. Container-native Virtualization の機能

Container-native Virtualization は OpenShift Container Platform のアドオンであり、仮想マシンのワークロードを実行し、このワークロードをコンテナーのワークロードと共に管理することを可能にします。

Container-native Virtualization は、Kubernetes カスタムリソースを使って新規オブジェクトを OpenShift Container Platform クラスターに追加し、仮想化タスクを有効にします。これらのタスクには、以下が含まれます。

- Linux および Windows 仮想マシンの作成と管理
- 各種コンソールおよび CLI ツールの使用による仮想マシンへの接続
- 既存の仮想マシンのインポートおよびクローン作成
- ネットワークインターフェースコントローラーおよび仮想マシンに割り当てられたストレージディスクの管理
- 仮想マシンのノード間でのライブマイグレーション

機能強化された Web コンソールは、これらの仮想化されたリソースを OpenShift Container Platform クラスターコンテナーおよびインフラストラクチャーと共に管理するためのグラフィカルポータルを提供します。

1.1.2. Container-native Virtualization のサポート

重要

Container-native Virtualization 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。

1.2. CONTAINER-NATIVE VIRTUALIZATION のクラスターの設定

OpenShift Container Platform の評価版バージョンを取得するには、OpenShift Container Platform ホームページから評価版をダウンロードしてください。

Container-native Virtualization はデフォルトで OpenShift Container Platform と連携しますが、以下のインストール設定が推奨されます。

- OpenShift Container Platform クラスターを ベアメタルにインストールします。コンピュートノードを、クラスター内でホストする仮想マシンの数およびサイズに応じて管理します。
- モニタリング をクラスターに設定します。

1.3. CONTAINER-NATIVE VIRTUALIZATION のインストール

Container-native Virtualization をインストールし、仮想化機能を OpenShift Container Platform クラスターに追加します。

OpenShift Container Platform 4.2 Web コンソールを使用して、Container-native Virtualization Operator にサブスクライブし、これをデプロイすることができます。

前提条件

- OpenShift Container Platform 4.2

重要

Container-native Virtualization 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。

1.3.1. Container-native Virtualization のインストールの準備

Container-native Virtualization をインストールする前に、**openshift-cnv**という名前の namespace を作成します。

前提条件

- **cluster-admin** 権限を持つユーザー

手順

1. OpenShift Container Platform Web コンソールで、**Administration → Namespaces** ページに移動します。
2. **Create Namespace**をクリックします。
3. **Name** フィールドに、 **openshift-cnv** を入力します。
4. **Create** をクリックします。

1.3.1.1. KubeVirt HyperConverged Cluster Operator カタログへのサブスクライブ

Container-native Virtualization をインストールする前に、OpenShift Container Platform Web コンソールから **KubeVirt HyperConverged Cluster Operator** カタログにサブスクライブします。サブスクライ

ブにより、**openshift-cnv** namespace に Container-native Virtualization Operator へのアクセスが付与されます。

前提条件

- **openshift-cnv**という名前の namespace を作成します。

手順

1. ブラウザーウィンドウを開き、OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** ページに移動します。
3. **KubeVirt HyperConverged Cluster Operator** を見つけ、これを選択します。
4. Operator についての情報を確認してから、**Install** をクリックします。
5. **Create Operator Subscription** ページで以下を実行します。
   a. **Installation Mode** 一覧から **A specific namespace on the cluster**を選択し、**openshift-cnv** namespace を選択します。

警告

- **All namespaces on the cluster (default)**の場合、Operator をデフォルトの **openshift-operators** namespace にインストールし、クラスターのすべての namespace を監視し、この Operator をこれらの namespace に対して利用可能にします。このオプションは、Container-native Virtualization で使用する場合にはサポート**されません**。Operator は **openshift-cnv** namespace にのみインストールする必要があります。

   b. 選択可能な **Update Channel** オプションから **2.1** を選択します。
   c. **Approval Strategy**では、デフォルト値である **Automatic**が選択されていることを確認します。Container-native Virtualization は、z-stream の新規リリースが利用可能になると自動的に更新されます。
6. **Subscribe** をクリックし、Operator をこの OpenShift Container Platform クラスターの選択した namespace で利用可能にします。

1.3.2. Container-native Virtualization のデプロイ

KubeVirt HyperConverged Cluster Operator カタログにサブスクライブした後に、**KubeVirt HyperConverged Cluster Operator Deployment** カスタムリソースを作成し、Container-native Virtualization をデプロイします。

前提条件

- **openshift-cnv** namespace での **KubeVirt HyperConverged Cluster Operator** カタログへのアクティブなサブスクリプション

手順

1. **Operators → Installed Operators** ページに移動します。
2. **KubeVirt HyperConverged Cluster Operator** をクリックします。
3. **KubeVirt HyperConverged Cluster Operator Deployment** タブをクリックし、**Create HyperConverged** をクリックします。
   a. **Create HyperConverged** をクリックすると、YAML ファイルが表示されます。**'false'** という単語の下にある一重引用符を削除します。これは、BZ#1767167 で報告されている問題の回避策となります。
   YAML ファイルは、最初に表示される際に以下の例のようになります。

```
apiVersion: hco.kubevirt.io/v1alpha1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  BareMetalPlatform: 'false' 1
```

   1 次のステップに進む前に、この行が **BareMetalPlatform: false** を読み取ることを確認します。

4. **Create** をクリックして Container-native Virtualization を起動します。
5. **Workloads → Pods** ページに移動して、Container-native Virtualization Pod がすべて **Running** 状態になるまでこれらの Pod をモニターします。すべての Pod で **Running** 状態が表示された後に、Container-native Virtualization にアクセスできます。

1.4. VIRTCTL クライアントのインストール

virtctl クライアントは、Container-native Virtualization リソースを管理するためのコマンドラインユーティリティーです。

Container-native Virtualization リポジトリーを有効にし、**kubevirt-virtctl** パッケージをインストールしてクライアントをシステムにインストールします。

1.4.1. Container-native Virtualization リポジトリーの有効化

Red Hat は、Red Hat Enterprise Linux 8 および Red Hat Enterprise Linux 7 向けの Container-native Virtualization リポジトリーを提供します。

- Red Hat Enterprise Linux 8 リポジトリー: **cnv-2.1-for-rhel-8-x86_64-rpms**
- Red Hat Enterprise Linux 7 リポジトリー: **rhel-7-server-cnv-2.1-rpms**

subscription-manager でリポジトリーを有効にするプロセスはどちらのプラットフォームでも同様です。

手順

- **subscription manager** を使用して、お使いのシステムに適した Container-native Virtualization リポジトリーを有効にします。

```
# subscription-manager repos --enable <repository>
```

1.4.2. virtctl クライアントのインストール

kubevirt-virtctl パッケージから **virtctl** クライアントをインストールします。

手順

- **kubevirt-virtctl** パッケージをインストールします。

```
# yum install kubevirt-virtctl
```

Container-native virtualization の「CLI ツールの使用」も参照してください。

1.5. CONTAINER-NATIVE VIRTUALIZATION のアップグレード

Container-native virtualization の インストール時に自動更新を有効にします。ここで予想されること、および更新のステータスをチェックする方法を確認します。

重要

Container-native Virtualization 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。

1.5.1. Container-native Virtualization のアップグレードについて

Container-native Virtualization のインストール時に自動更新を有効にした場合、更新が利用可能になり次第、それらの更新を受信できます。

追加情報

- Container-native Virtualization バージョン 2.1 では、**z-stream** 更新のみが利用できます。たとえば、Container-native Virtualization 2.1.0 → Container-native virtualization 2.1.1 のアップグレードの例を見てみましょう。
- 更新は、OpenShift Container Platform のインストール時にデプロイされる **Marketplace Operator** 経由で送信されます。Marketplace Operator は外部 Operator をクラスターに対して利用可能にします。
- アップグレードを実行しても仮想マシンのワークロードは中断しません。

- 仮想マシン Pod は、アップグレード時に再起動したり、移行したりしません。**virt-launcher** Pod を更新する必要がある場合は、仮想マシンの再起動またはライブマイグレーションが必要になります。

注記

各仮想マシンには、仮想マシンインスタンスを実行する **virt-launcher** Pod があります。**virt-launcher** Pod は、仮想マシンのプロセスを管理するために使用される **libvirt** のインスタンスを実行します。

- アップグレードによってネットワーク接続が中断されることはありません。
- DataVolume およびその関連付けられた PersistentVolumeClaim はアップグレード時に保持されます。
- 更新の完了までにかかる時間は、ネットワーク接続によって異なります。ほとんどの自動更新は 15 分以内に完了します。

1.5.2. アップグレードステータスの監視

Container-native Virtualization アップグレードステータスをモニターする最適な方法として、ClusterServiceVersion (CSV) (CSV) **PHASE** を監視できます。Web コンソールを使用するか、ここに提供されているコマンドを実行して CSV の状態をモニターすることもできます。

注記

PHASE および状態の値は利用可能な情報に基づく近似値になります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。

手順

1. 次のコマンドを実行します。

```
$ oc get csv
```

2. 出力を確認し、**PHASE** フィールドをチェックします。以下は例になります。

```
VERSION  REPLACES                                   PHASE
2.1.1    kubevirt-hyperconverged-operator.v2.1.0    Installing
2.1.0                                               Replacing
```

3. オプション: 以下のコマンドを実行して、すべての Container-native Virtualization コンポーネントの状態の集約されたステータスをモニターします。

```
$ oc get hco -n openshift-cnv hyperconverged-cluster \
-o=jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.message}{"\n"}{end}'
```

アップグレードが成功すると、以下の出力が得られます。

```
ReconcileComplete  True  Reconcile completed successfully
Available          True  Reconcile completed successfully
Progressing        False Reconcile completed successfully
Degraded           False Reconcile completed successfully
Upgradeable        True  Reconcile completed successfully
```

追加情報

- ClusterServiceVersion (CSV)

1.6. CONTAINER-NATIVE VIRTUALIZATION のアンインストール

OpenShift Container Platform Web コンソールを使用して Container-native Virtualization をアンインストールできます。

前提条件

- Container-native Virtualization 2.1

1.6.1. KubeVirt HyperConverged カスタムリソースの削除

Container-native Virtualization をアンインストールするには、まず **KubeVirt HyperConverged Cluster Operator Deployment** カスタムリソースを削除する必要があります。

前提条件

- アクティブな **KubeVirt HyperConverged Cluster Operator Deployment** カスタムリソース

手順

1. OpenShift Container Platform Web コンソールから、**Projects** 一覧より **openshift-cnv** を選択します。
2. **Operators → Installed Operators** ページに移動します。
3. **KubeVirt HyperConverged Cluster Operator** をクリックします。
4. **KubeVirt HyperConverged Cluster Operator Deployment** タブをクリックします。
5. Options メニュー  を、**hyperconverged-cluster** カスタムリソースを含む行でクリックします。拡張されたメニューで、**Delete HyperConverged** をクリックします。
6. 確認ウィンドウで **Delete** をクリックします。
7. **Workloads → Pods** ページに移動し、Operator Pod のみが実行中であることを確認します。
8. ターミナルウィンドウを開き、以下のコマンドを実行して残りの KubeVirt リソースをクリーンアップします。

```
$ oc delete apiservices v1alpha3.subresources.kubevirt.io -n openshift-cnv
```

注記

現時点で、一部の KubeVirt リソースが不適切に保持される状態が確認されるため、それらを手動で削除する必要があります。これらのリソースは、(BZ1712429) の解決後に自動的に削除されます。

1.6.2. KubeVirt HyperConverged Cluster Operator カタログサブスクリプションの削除

Container-native Virtualization のアンインストールを完了するには、**KubeVirt HyperConverged Cluster Operator サブスクリプション**をアンインストールします。

前提条件

- アクティブな **KubeVirt HyperConverged Cluster Operator** カタログサブスクリプション

手順

1. **Catalog → OperatorHub** ページに移動します。
2. **KubeVirt HyperConverged Cluster Operator** を見つけ、これを選択します。
3. **Uninstall** をクリックします。

注記

openshift-cnv namespace を削除できるようになりました。

1.6.3. Web コンソールを使用した namespace の削除

OpenShift Container Platform Web コンソールを使用して namespace を削除できます。

注記

namespace を削除するパーミッションがない場合、**Delete Namespace** オプションは選択できなくなります。

手順

1. **Administration → Namespaces** に移動します。
2. namespace の一覧で削除する必要のある namespace を見つけます。
3. namespace の一覧の右端で、Options メニューから **Delete Namespace** を選択します。 をクリックします。
4. **Delete Namespace** ペインが表示されたら、フィールドから削除する namespace の名前を入力します。
5. **Delete** をクリックします。

第2章 CONTAINER-NATIVE VIRTUALIZATION ユーザーガイド

2.1. 仮想マシンの作成

以下のいずれかの手順を使用して、仮想マシンを作成します。

- 仮想マシンウィザードの実行
- 仮想マシンウィザードによる事前に設定された YAML ファイルの貼り付け
- CLI の使用
- 仮想マシンウィザードによる VMware 仮想マシンまたはテンプレートのインポート

2.1.1. 仮想マシンウィザードの実行による仮想マシンの作成

Web コンソールは、**Basic Settings**、**Networking**、および **Storage** 画面にナビゲートし、仮想マシンの作成プロセスを単純化するインタラクティブなウィザードを特長としています。すべての必須フィールドには * のマークが付けられます。ウィザードは必須フィールドの入力が完了するまで次の画面に移動することを防ぎます。

NIC およびストレージディスクを作成し、それらの作成後に仮想マシンに割り当てることができます。

ブート可能なディスク

Basic Settings 画面で **URL** または **Container** のいずれかが **Provision Source** として選択されている場合、 **rootdisk** ディスクが **Bootable Disk** として作成され、仮想マシンに割り当てられます。**rootdisk** を変更できますが、これを削除することはできません。

Bootable Disk は、仮想マシンにディスクが割り当てられていない場合、**PXE** ソースからプロビジョニングされる仮想マシンには不要です。1 つ以上のディスクが仮想マシンに割り当てられている場合、**Bootable Disk** を 1 つを選択する必要があります。

前提条件

- ウィザードを使用して仮想マシンを作成する場合、仮想マシンのストレージメディアは Read-Write-Many（RWM）PVC をサポートする必要があります。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. **Create Virtual Machine**をクリックし、**Create with Wizard**を選択します。
3. すべての必須の **Basic Settings** を入力します。**Template** を選択すると、これらのフィールドへの入力が自動的に行われます。
4. **Next** をクリックして **Networking** 画面に進みます。デフォルトで **nic0** NIC が割り当てられます。
    a. (オプション) **Create NIC** をクリックして追加の NIC を作成します。
    b. (オプション) ⋮ ボタンをクリックし、**Remove NIC** を選択して、NICS のいずれかまたはすべてを削除できます。仮想マシンの作成において、NIC が割り当てられている必要はありません。NIC は仮想マシンの作成後に作成することができます。

5. **Next** をクリックして **Storage** 画面に進みます。

    a. (オプション) **Create Disk** をクリックして追加のディスクを作成します。これらのディスクは、⋮ ボタンをクリックし、 **Remove Disk** を選択して削除できます。

    b. (オプション) ディスクをクリックして選択可能なフィールドを変更します。✓ ボタンをクリックして更新を保存します。

    c. (オプション) **Attach Disk** をクリックして、**Select Storage** ドロップダウンリストから利用可能なディスクを選択します。

6. **Create Virtual Machine** >をクリックします。**Results** 画面には、仮想マシンの JSON 設定ファイルが表示されます。

仮想マシンは **Workloads** → **Virtual Machines** に一覧表示されます。

Web コンソールウィザードを実行する際は、仮想マシンウィザードのフィールドを参照します。

2.1.1.1. 仮想マシンウィザードのフィールド

名前	パラメーター	説明
名前		この名前には、小文字 (**a-z**)、数字 (**0-9**)、およびハイフン (**-**) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (**.**)、または特殊文字を使用できません。
説明		オプションの説明フィールド。
Template		仮想マシンの作成に使用するテンプレート。テンプレートを選択すると、他のフィールドが自動的に入力されます。
Provision Source	PXE	PXE メニューから仮想マシンをプロビジョニングします。クラスターに PXE 対応の NIC が必要になります。
	URL	**HTTP** または **S3** エンドポイントで利用できるイメージから仮想マシンをプロビジョニングします。
	Container	クラスターからアクセスできるレジストリーの起動可能なオペレーティングシステムコンテナーから仮想マシンをプロビジョニングします。例: **kubevirt/cirros-registry-disk-demo**

名前	パラメーター	説明
	Cloned Disk	プロビジョニングソースはクローン作成されたディスクです。
	Import	サポートされているプロバイダーから仮想マシンをインポートします。
Operating System		クラスターで利用可能なオペレーティングシステムの一覧。これは、仮想マシンの主なオペレーティングシステムになります。**Import** を **Provider Source** として選択する場合、オペレーティングシステムはインポートされる VMware 仮想マシンのオペレーティングシステムに基づいて自動的に入力されます。
Flavor	small、medium、large、tiny、Custom	仮想マシンに割り当てられる CPU およびメモリーの量を決定するプリセット。
Workload Profile	desktop	デスクトップで使用するための仮想マシン設定。
	generic	各種のワークロードについてのパフォーマンスと互換性のバランスを取るための仮想マシンの設定。
	high performance	高パフォーマンスの負荷に対して最適化された仮想マシン設定。
Start virtual machine on creation		これを選択すると、作成時に仮想マシンが自動的に起動します。
Use cloud-init		これを選択し、cloud-init フィールドを有効にします。

2.1.1.2. Cloud-init フィールド

名前	説明
Hostname	仮想マシンの特定のホスト名を設定します。
Authenticated SSH Keys	仮想マシンの **~/.ssh/authorized_keys** にコピーされるユーザーの公開鍵。

名前	説明
Use custom script	他のオプションを、カスタム cloud-init スクリプトを貼り付けるフィールドに置き換えます。

2.1.1.3. ネットワークフィールド

名前	説明
Create NIC	仮想マシンの新規 NIC を作成します。
NIC NAME	NIC の名前。
MAC ADDRESS	ネットワークインターフェースの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。
NETWORK CONFIGURATION	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
BINDING METHOD	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、**masquerade** が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、**bridge** バインディングメソッドを使用します。**masquerade** メソッドは、デフォルト以外のネットワークではサポートされません。
PXE NIC	PXE 対応ネットワークの一覧。**PXE** が **Provision Source** として選択されている場合にのみ表示されます。

2.1.1.4. ストレージフィールド

名前	説明
Create Disk	仮想マシンの新規ディスクを作成します。
Attach Disk	利用可能な PVC の一覧から、仮想マシンに割り当てる既存のディスクを選択します。
DISK NAME	ディスクの名前。この名前には、小文字 (**a-z**)、数字 (**0-9**)、ハイフン (**-**) およびピリオド (**.**) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、または特殊文字を使用できません。

名前	説明
SIZE (GB)	ディスクのサイズ (GB)。
STORAGE CLASS	基礎となる **StorageClass** の名前。
Bootable Disk	仮想マシンの起動に利用できるディスクの一覧。仮想マシンの Provision Source が **URL** または **Container** の場合に **rootdisk** に固定されます。

2.1.2. 仮想マシンウィザードの作成用の事前設定 YAML ファイルの貼り付け

Web コンソールの **Workloads → Virtual Machines** 画面で YAML 設定ファイルを作成するか、またはこれを貼り付けて仮想マシンを作成します。YAML 編集画面を開くと、常に有効な **example** 仮想マシン設定がデフォルトで提供されます。

Create をクリックする際に YAML 設定が無効な場合、エラーメッセージでエラーが発生したパラメーターが示唆されます。エラーは一度に 1 つのみ表示されます。

注記

編集中に YAML 画面から離れると、設定に対して加えた変更が取り消されます。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. **Create Virtual Machine**をクリックし、**Create from YAML** を選択します。
3. 編集可能なウィンドウで仮想マシンの設定を作成するか、またはこれを貼り付けます。
    a. または、YAML 画面にデフォルトで提供される **example** 仮想マシンを使用します。
4. (オプション) **Download** をクリックして YAML 設定ファイルをその現在の状態でダウンロードします。
5. **Create** をクリックして仮想マシンを作成します。

仮想マシンは **Workloads → Virtual Machines** に一覧表示されます。

2.1.3. CLI の使用による仮想マシンの作成

手順

VirtualMachine 設定ファイルの **spec** オブジェクトは、コア数やメモリーの量、ディスクタイプおよび使用するボリュームなどの仮想マシン設定を参照します。

1. 関連する PVC **claimName** をボリュームとして参照し、仮想マシンディスクを仮想マシンに割り当てます。
2. OpenShift Container Platform クライアントで仮想マシンを作成するには、以下のコマンドを実行します。

```
$ oc create -f <vm.yaml>
```

3. 仮想マシンは **Stopped** 状態で作成されるため、これを起動して仮想マシンインスタンスを実行します。

注記

ReplicaSetは、一定数の同一 Pod の可用性を保証することを目的としています。現時点で、ReplicaSet は Container-native Virtualization でサポートされていません。

表2.1 ドメイン設定

設定	説明
Cores	仮想マシン内のコア数。1 以上の値である必要があります。
Memory	ノードによって仮想マシンに割り当てられる RAM の量。**M** (メガバイト) または **Gi** (ギガバイト) で値を指定します。
Disks: name	参照されるボリュームの名前。ボリュームの名前に一致する必要があります。

表2.2 ボリューム設定

設定	説明
名前	ボリュームの名前。 DNS ラベルであり、仮想マシン内で一意である必要があります。
PersistentVolumeClaim	仮想マシンに割り当てる PVC。PVC の **claimName** は仮想マシンと同じプロジェクトになければなりません。

2.1.4. 仮想マシンのストレージボリュームタイプ

仮想マシンのストレージボリュームタイプがドメインおよびボリューム設定と共に一覧表示されます。仮想マシン設定の具体的な一覧については、「 kubevirt API Reference 」を参照してください。

ephemeral	ネットワークボリュームを読み取り専用のバッキングストアとして使用するローカルの copy-on-write (COW) イメージ。バッキングボリュームは **PersistentVolumeClaim** である必要があります。一時イメージは仮想マシンの起動時に作成され、すべての書き込みをローカルに保存します。一時イメージは、仮想マシンの停止、再起動または削除時に破棄されます。バッキングボリューム (PVC) はいずれの方法でも変更されません。

persistentVolumeClaim	利用可能な PV を仮想マシンに割り当てます。PV の割り当てにより、仮想マシンデータのセッション間での永続化が可能になります。 CDI を使用して既存の仮想マシンディスクを PVC にインポートし、PVC を仮想マシンインスタンスに割り当てる方法は、既存の仮想マシンを OpenShift Container Platform にインポートするための推奨される方法です。ディスクを PVC 内で使用できるようにするためのいくつかの要件があります。
dataVolume	DataVolume は、インポート、クローンまたはアップロード操作で仮想マシンディスクの準備プロセスを管理することによって **persistentVolumeClaim** ディスクタイプにビルドされます。このボリュームタイプを使用する仮想マシンは、ボリュームが準備できるまで起動しないことが保証されます。
cloudInitNoCloud	参照される cloud-init NoCloud データソースが含まれるディスクを割り当て、ユーザーデータおよびメタデータを仮想マシンに提供します。cloud-init インストールは仮想マシンディスク内で必要になります。
containerDisk	コンテナーイメージレジストリーに保存される、仮想マシンディスクなどのイメージを参照します。イメージはレジストリーからプルされ、仮想マシンの作成時にボリュームに組み込まれます。**containerDisk** ボリュームは一時的なボリュームです。これは、仮想マシンが停止されるか、再起動するか、または削除される際に破棄されます。 コンテナーディスクは単一の仮想マシンに制限されず、永続ストレージを必要としない多数の仮想マシンのクローンを作成するのに役立ちます。 RAW および QCOW2 形式のみがコンテナーイメージレジストリーのサポートされるディスクタイプです。QCOW2 は、縮小されたイメージサイズの場合に推奨されます。
emptyDisk	仮想マシンインターフェースのライフサイクルに関連付けられるスパースの QCOW2 ディスクを追加で作成します。データは仮想マシンのゲストによって実行される再起動後も存続しますが、仮想マシンが Web コンソールから停止または再起動する場合には破棄されます。空のディスクは、アプリケーションの依存関係および一時ディスクの一時ファイルシステムの制限を上回るデータを保存するために使用されます。 ディスク **容量** サイズも指定する必要があります。

仮想マシン設定の具体的な一覧については、「 kubevirt API Reference 」を参照してください。

2.2. DATAVOLUME インポートの TLS 証明書

2.2.1. DataVolume インポートの認証に使用する TLS 証明書の追加

ソースからデータをインポートするには、レジストリーまたは HTTPS エンドポイントの TLS 証明書を ConfigMap に追加する必要があります。この ConfigMap は、宛先 DataVolume の namespace に存在する必要があります。

TLS 証明書の相対パスを参照して ConfigMap を作成します。

手順

1. 正しい namespace にあることを確認します。ConfigMap は、同じ namespace にある場合に DataVolume によってのみ参照されます。

```
$ oc get ns
```

2. ConfigMap を作成します。

```
$ oc create configmap <configmap-name> --from-file=</path/to/file/ca.pem>
```

2.2.2. 例: TLS 証明書から作成される ConfigMap

以下は、**ca.pem** TLS 証明書で作成される ConfigMap の例です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tls-certs
data:
  ca.pem: |
    -----BEGIN CERTIFICATE-----
    ... <base64 encoded cert> ...
    -----END CERTIFICATE-----
```

2.3. DATAVOLUME の使用による仮想マシンイメージのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスターにインポートできます。Container-native Virtualization は DataVolume を使用してデータのインポートおよび基礎となる PersistentVolumeClaim (PVC) の作成を自動化します。

注意

ディスクイメージを PVC にインポートする際に、ディスクイメージは PVC で要求されるストレージの全容量を使用するように拡張されます。この領域を使用するには、仮想マシンのディスクパーティションおよびファイルシステムの拡張が必要になる場合があります。

サイズ変更の手順は、仮想マシンにインストールされるオペレーティングシステムによって異なります。詳細は、該当するオペレーティングシステムのドキュメントを参照してください。

前提条件

- エンドポイントに TLS 証明書が必要な場合、証明書は DataVolume と同じ namespace の ConfigMap に組み込む必要があり、これは DataVolume 設定で参照されます。
- この操作を正常に実行するためには、StorageClass を定義するか、CDI のスクラッチ領域を用意する必要がある場合があります。

2.3.1. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.3.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.3.3. DataVolume のあるオブジェクトへの仮想マシンイメージのインポート

インポートされたイメージから仮想マシンを作成するには、仮想マシンを作成する前にイメージの場所を **VirtualMachine** 設定ファイルに指定します。

前提条件

- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) のインストール
- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- イメージがデータソースにアクセスするために必要な認証情報と共にホストされる **HTTP** エンドポイント
- 1 つ以上の利用可能な PersistentVolume

手順

1. インポートする必要のある仮想ディスクイメージをホストする **HTTP** ファイルサーバーを特定します。正しい形式での完全な URL が必要になります。

- **http://www.example.com/path/to/data**

2. データソースに認証情報が必要な場合、**endpoint-secret.yaml** ファイルを編集し、更新された設定をクラスターに適用します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <endpoint-secret>
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 1
  secretKey:   "" 2
```

1 オプション: キーまたはユーザー名 (base64 エンコード)

2 オプション: シークレットまたはパスワード、base64 エンコード

```
$ oc apply -f endpoint-secret.yaml
```

3. 仮想マシン設定ファイルを編集し、インポートする必要のあるイメージのデータソースを指定します。この例では、Fedora イメージがインポートされます。

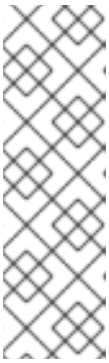
```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume
spec:
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: fedora-dv
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 2Gi
        storageClassName: local
      source:
        http:
          url:
https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora
-Cloud-Base-28-1.1.x86_64.qcow2 1
          secretRef: "" 2
          certConfigMap: "" 3
    status: {}
  running: false
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/vm: vm-fedora-datavolume
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: datavolumedisk1
        machine:
          type: ""
        resources:
          requests:
            memory: 64M
      terminationGracePeriodSeconds: 0
      volumes:
      - dataVolume:
          name: fedora-dv
        name: datavolumedisk1
status: {}
```

1. インポートする必要のあるイメージの **HTTP** ソース。
2. **secretRef** パラメーターはオプションです。
3. **certConfigMap** はエンドポイントが認証を必要とする場合のみ必要になります。参照される ConfigMap は DataVolume と同じ namespace にある必要があります。

4. 仮想マシンを作成します。

```
$ oc create -f vm-<name>-datavolume.yaml
```



注記

oc create コマンドは、DataVolume および仮想マシンを作成します。CDI コントローラーは適切なアノテーションを使って基礎となる PVC を作成し、インポートプロセスが開始されます。インポートが完了すると、DataVolume のステータスは **Succeeded** に変更され、仮想マシンの起動が可能になります。

DataVolume のプロビジョニングはバックグランドで実行されるため、これをモニターする必要はありません。仮想マシンは起動できますが、これはインポートが完了するまで実行されません。

オプションの検証手順

1. **oc get pods** を実行し、インポーター Pod を見つけます。この Pod は指定された URL からイメージをダウンロードし、これをプロビジョニングされた PV に保存します。
2. **Succeeded** が表示されるまで DataVolume のステータスをモニターします。

```
$ oc describe dv <data-label> 1
```

1 仮想マシン設定ファイルに指定された DataVolume のデータラベル。

3. プロビジョニングが完了し、VMI が起動したことを検証するには、そのシリアルコンソールへのアクセスを試行します。

```
$ virtctl console <vm-fedora-datavolume>
```

2.3.4. テンプレート: DataVolume 仮想マシン設定ファイル

example-dv-vm.yaml

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: example-vm
  name: example-vm
spec:
  dataVolumeTemplates:
  - metadata:
      name: example-dv
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 1G
      source:
          http:
            url: "" 1
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
        devices:
          disks:
          - disk:
              bus: virtio
            name: example-dv-disk
        machine:
          type: q35
        resources:
          requests:
            memory: 1G
      terminationGracePeriodSeconds: 0
      volumes:
```

```
    - dataVolume:
        name: example-dv
      name: example-dv-disk
```

1 インポートする必要のあるイメージの **HTTP** ソース (該当する場合)。

2.3.5. テンプレート: DataVolume インポート設定ファイル

example-import-dv.yaml

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: "example-import-dv"
spec:
  source:
      http:
        url: "" 1
        secretRef: "" 2
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: "1G"
```

1 インポートする必要のあるイメージの **HTTP** ソース。

2 **secretRef** パラメーターはオプションです。

2.4. 仮想マシンの編集

以下のタスクのいずれかを実行して仮想マシンを編集します。

- Web コンソールを使用した仮想マシンの YAML 設定の編集
- CLI を使用した仮想マシン YAML 設定の編集

2.4.1. Web コンソールを使用した仮想マシンの YAML 設定の編集

Web コンソールを使用して、仮想マシンの YAML 設定を編集します。

すべてのパラメーターを更新できる訳ではありません。変更不可の値を編集し、**Save** をクリックすると、更新できなかったパラメーターを示すエラーメッセージが出されます。

YAML 設定は、仮想マシンが **Running** の場合に編集できますが、変更は仮想マシンが停止され、再度起動された後にのみ有効になります。

注記

編集中に YAML 画面から離れると、設定に対して加えた変更が取り消されます。

手順

1. サイドメニューから **Workloads → Virtual Machine** をクリックします。
2. 仮想マシンを選択します。
3. **YAML** タブをクリックして編集可能な設定を表示します。
4. オプション: **Download** をクリックして YAML ファイルをその現在の状態でローカルにダウンロードできます。
5. ファイルを編集し、**Save** をクリックします。

オブジェクトの更新されたバージョン番号を含め、変更が正常に行われたことを示す確認メッセージが表示されます。

2.4.2. CLI を使用した仮想マシン YAML 設定の編集

前提条件

- YAML オブジェクト設定ファイルを使って仮想マシンを設定していること。
- **oc** CLI をインストールしていること。

手順

1. 以下のコマンドを実行して、仮想マシン設定を更新します。

```
$ oc edit <object_type> <object_ID>
```

2. オブジェクト設定を開きます。
3. YAML を編集します。
4. 実行中の仮想マシンを編集する場合は、以下のいずれかを実行する必要があります。
   - 仮想マシンの再起動
   - 新規の設定を有効にするために、以下のコマンドを実行します。

```
$ oc apply <object_type> <object_ID>
```

2.4.3. 仮想マシンへの仮想ディスクの追加

仮想マシンへのディスクの追加

手順

1. **Virtual Machines** タブで、仮想マシンを選択します。
2. **Disks** タブを選択します。
3. **Add Disks** をクリックして、 **Add Disk** ウィンドウを開きます。

4. **Add Disk** ウィンドウで、**Source**、**Name**、 **Size**、**Interface**、および **Storage Class** を指定します。
5. ドロップダウンリストおよびチェックボックスを使用して、ディスク設定を編集します。
6. **OK** をクリックします。

2.4.4. 仮想マシンへのネットワークインターフェースの追加

手順

1. **Virtual Machines** タブで、仮想マシンを選択します。
2. **Network Interfaces** タブを選択します。
3. **Create Network Interface** をクリックします。
4. **Create Network Interface** 一覧の行で、ネットワークインターフェースの **Name**、**Model**、**Network**、**Type**、および **MAC Address** を指定します。
5. 行の右にある緑色のチェックマークをクリックして、ネットワークインターフェースを追加します。
6. 仮想マシンを再起動して、アクセスを有効にします。
7. ドロップダウンリストとチェックボックスを編集して、ネットワークインターフェースを設定します。
8. **Save Changes** をクリックします。
9. **OK** をクリックします。

新規のネットワークインターフェースは、ユーザーが再起動するまで **Create Network Interface** 一覧の上部に表示されます。

新規のネットワークインターフェースには、仮想マシンを再起動するまで **Pending VM restart** のリンク状態が表示されます。Link State (リンク状態) にカーソルを合わせて詳細情報を表示します。

ネットワークインターフェースカードが仮想マシンで定義され、ネットワークに接続されている場合は、**Link State** はデフォルトで **Up** に設定されます。

2.5. 仮想マシンの削除

以下のいずれかの手順を使用して、仮想マシンを削除します。

- Web コンソールの使用
- CLI の使用

2.5.1. Web コンソールの使用による仮想マシンの削除

仮想マシンを削除すると、仮想マシンはクラスターから永続的に削除されます。

Workloads → Virtual Machines 一覧で仮想マシンの ⋮ ボタンを使用するか、または **Virtual Machine Details** 画面の **Actions** コントロールを使用して仮想マシンを削除します。

手順

1. Container-native Virtualization コンソールのサイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 削除する仮想マシンの ⋮ ボタンをクリックして **Delete Virtual Machine**を選択します。
   - または、仮想マシン名をクリックして **Virtual Machine Details**画面を開き、**Actions → Delete Virtual Machine**をクリックします。
3. 確認のポップアップウィンドウで、**Delete** をクリックし、仮想マシンを永続的に削除します。

2.5.2. CLI を使用した仮想マシンおよびその DataVolume の削除

仮想マシンを削除する際に、これが使用する DataVolume は自動的に削除されません。

クリーンな環境を維持し、混乱の可能性を避けるために、DataVolume を削除することを推奨します。

手順

これらのコマンドを実行して、仮想マシンと DataVolume を削除します。

注記

-n <project_name> オプションを指定しない場合、現在作業しているプロジェクトでのみオブジェクトを削除できます。

1. 以下のコマンドを実行し、仮想マシンを削除します。

```
$ oc delete vm <fedora-vm>
```

2. 以下のコマンドを実行し、DataVolume を削除します。

```
$ oc delete dv <datavolume-name>
```

2.6. 仮想マシンの状態の制御

Container-native Virtualization を使用すると、Web コンソールおよびコマンドラインインターフェース (CLI) から仮想マシンを停止し、起動し、再起動できます。

2.6.1. Web コンソールからの仮想マシンの制御

Web コンソールから仮想マシンを停止し、起動し、再起動することもできます。

2.6.1.1. 仮想マシンの起動

Web コンソールから仮想マシンを起動できます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。
2. この画面から仮想マシンを起動します。 これにより、1 つの画面で複数のマシンに対してアク

ションを実行することがより容易になります。 または、**Virtual Machine Details**画面から仮想マシンを起動することもできます。 この場合、選択された仮想マシンの総合的な詳細情報を確認できます。

- ディスクの Options メニュー  を仮想マシンの末尾でクリックし、**Start Virtual Machine** を選択します。
- 仮想マシン名をクリックして **Virtual Machine Details**画面を開き、**Actions** をクリックして **Start Virtual Machine**を選択します。

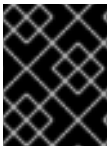
3. 確認ウィンドウで **Start** をクリックし、仮想マシンを起動します。

注記

URL ソースからプロビジョニングされる仮想マシンの初回起動時に、Container-native Virtualization が URL エンドポイントからコンテナーをインポートする間、仮想マシンの状態は **Importing** になります。このプロセスは、イメージのサイズによって数分の時間がかかる可能性があります。

2.6.1.2. 仮想マシンの再起動

Web コンソールから実行中の仮想マシンを再起動できます。



重要

ステータスが **Importing** の仮想マシンは再起動しないでください。この仮想マシンを再起動すると、エラーが発生します。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。
2. この画面から仮想マシンを再起動します。 これにより、1 つの画面で複数のマシンに対してアクションを実行することがより容易になります。 または、**Virtual Machine Details**画面から仮想マシンを起動することもできます。 この場合、選択された仮想マシンの総合的な詳細情報を確認できます。

- ディスクの Options メニュー  を仮想マシンの末尾でクリックし、**Restart Virtual Machine**を選択します。
- 仮想マシン名をクリックして **Virtual Machine Details**画面を開き、**Actions** クリックして **Restart Virtual Machine**を選択します。

3. 確認ウィンドウで **Restart** をクリックし、仮想マシンを再起動します。

2.6.1.3. 仮想マシンの停止

Web コンソールから仮想マシンを停止できます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。

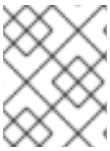
2. この画面から仮想マシンを停止します。 これにより、1 つの画面で複数のマシンに対してアクションを実行することがより容易になります。 または、**Virtual Machine Details**画面から仮想マシンを停止することもできます。 この場合、選択された仮想マシンの総合的な詳細情報を確認できます。

   - ディスクの Options メニュー  を仮想マシンの末尾でクリックし、**Stop Virtual Machine**を選択します。
   - 仮想マシン名をクリックして **Virtual Machine Details**画面を開き、**Actions** クリックして **Stop Virtual Machine** を選択します。

3. 確認ウィンドウで **Stop** をクリックし、仮想マシンを停止します。

2.6.2. 仮想マシンを制御するための CLI リファレンス

以下の **virtctl** クライアントユーティリティーおよび **oc** コマンドを使用して仮想マシンの状態を変更し、仮想マシンおよびそれらを表す仮想マシンインスタンスの一覧を表示します。



注記

virtctl コマンドを実行する場合、Web コンソールで仮想マシンを表す仮想マシンインスタンスではなく、仮想マシン自体を変更します。

2.6.2.1. start

仮想マシンを起動します。

例: 現行プロジェクトでの仮想マシンの起動

```
$ virtctl start <example-vm>
```

例: 特定プロジェクトでの仮想マシンの起動

```
$ virtctl start <example-vm> -n <project_name>
```

2.6.2.2. restart

実行中の仮想マシンを再起動します。

例: 現行プロジェクトでの仮想マシンの再起動

```
$ virtctl restart <example-vm>
```

例: 特定のプロジェクトでの仮想マシンの再起動

```
$ virtctl restart <example-vm> -n <project_name>
```

2.6.2.3. stop

実行中の仮想マシンを停止します。

例: 現行プロジェクトでの仮想マシンの停止

```
$ virtctl stop <example-vm>
```

例: 特定のプロジェクトでの仮想マシンの停止

```
$ virtctl stop <example-vm> -n <project_name>
```

2.6.2.4. list

プロジェクトに仮想マシンまたは仮想マシンインスタンスを一覧表示します。仮想マシンインスタンスは仮想マシン自体を抽象化したものです。

例: 現行プロジェクトでの仮想マシンの一覧表示

```
$ oc get vm
```

例: 特定プロジェクトでの仮想マシンの一覧表示

```
$ oc get vm -n <project_name>
```

例: 現行プロジェクトでの実行中の仮想マシンインスタンスの一覧表示

```
$ oc get vmi
```

例: 特定プロジェクトでの実行中の仮想マシンインスタンスの一覧表示

```
$ oc get vmi -n <project_name>
```

2.7. 仮想マシンコンソールへのアクセス

Container-native Virtualization は、異なる製品タスクを実現するために使用できる異なる仮想マシンコンソールを提供します。Web コンソールおよび CLI コマンドを使用してこれらのコンソールにアクセスできます。

2.7.1. 仮想マシンコンソールのセッション

Web コンソールの **Virtual Machine Details**画面の **Consoles** タブから、実行中の仮想マシンの VNC およびシリアルコンソールに接続することができます。

グラフィカルな **VNC コンソール** と **シリアルコンソール** の 2 つのコンソールを使用できます。**VNC コンソール** は、**Consoles** タブに移動する際には常にデフォルトで開きます。**VNC ConsoleSerial Console** リストを使用してコンソールを切り換えることができます。

コンソールのセッションは切断しない限り、バックグラウンドでアクティブな状態のままになります。**Disconnect before switching** チェックボックスがアクティブな場合にコンソールを切り替えると、現在のコンソールセッションは切断され、選択したコンソールの新規セッションが仮想マシンに接続されます。これにより、一度に 1 つのコンソールセッションのみが開かれます。

VNC コンソール のオプション

Send Key ボタンでは、仮想マシンに送信するキーの組み合わせを一覧表示します。

シリアルコンソール のオプション

Disconnect ボタンを使用して、仮想マシンから **Serial Console** セッションを手動で切断します。
Reconnect ボタンを使用して **Serial Console** セッションを仮想マシンに対して手動で開きます。

2.7.2. Web コンソールの使用による仮想マシンへの接続

2.7.2.1. ターミナルへの接続

Web コンソールを使用して仮想マシンに接続することができます。

手順

1. 正しいプロジェクトを指定していることを確認します。そうでない場合は、**Project** 一覧をクリックして適切なプロジェクトを選択します。
2. **Workloads → Virtual Machines** をクリックして、プロジェクトに仮想マシンを表示します。
3. 仮想マシンを選択します。
4. **Overview** タブで、**virt-launcher-<vm-name>** Pod をクリックします。
5. **Terminal** タブをクリックします。ターミナルが空白の場合、ターミナルをクリックし、任意のキーを押して接続を開始します。

2.7.2.2. シリアルコンソールへの接続

Web コンソールの **Virtual Machine Details**画面の **Consoles** タブから、実行中の仮想マシンの **Serial Console** に接続します。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。
2. 仮想マシンを選択します。
3. **Consoles** をクリックします。VNC コンソールがデフォルトで開きます。
4. **VNC Console** ドロップダウンリストをクリックし、**Serial Console** を選択します。

2.7.2.3. VNC コンソールへの接続

Web コンソールの **Virtual Machine Details**画面の **Consoles** タブから、実行中の仮想マシンの VNC コンソールに接続します。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。

2. 仮想マシンを選択します。

3. **Consoles** をクリックします。VNC コンソールがデフォルトで開きます。

2.7.2.4. RDP コンソールへの接続

Remote Desktop Protocol (RDP) を使用するデスクトップビューアーコンソールは、Windows 仮想マシンに接続するためのより使いやすいコンソールを提供します。

RDP を使用して Windows 仮想マシンに接続するには、Web コンソールの **Virtual Machine Details** 画面の **Consoles** タブから仮想マシンの **console.rdp** ファイルをダウンロードし、これを優先する RDP クライアントに指定します。

前提条件

- QEMU ゲストエージェントがインストールされた実行中の Windows 仮想マシン。**qemu-guest-agent** は VirtIO ドライバーに含まれています。
- 仮想マシンに接続されたレイヤー2 vNIC。
- Windows 仮想マシンと同じネットワーク上のマシンにインストールされた RDP クライアント。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。
2. Windows 仮想マシンを選択します。
3. **Consoles** タブをクリックします。
4. **Consoles** 一覧をクリックし、**Desktop Viewer** を選択します。
5. **Network Interface** 一覧で、レイヤー 2 vNIC を選択します。
6. **Launch Remote Desktop** をクリックし、 **console.rdp** ファイルをダウンロードします。
7. RDP クライアントを開き、**console.rdp** ファイルを参照します。たとえば、**remmina** を使用します。

```
$ remmina --connect /path/to/console.rdp
```

8. **Administrator** ユーザー名およびパスワードを入力して、Windows 仮想マシンに接続します。

2.7.3. CLI コマンドの使用による仮想マシンコンソールへのアクセス

2.7.3.1. SSH 経由での仮想マシンインスタンスへのアクセス

仮想マシンにポート 22 を公開した後に、SSH を使用して仮想マシンにアクセスできます。

virtctl expose コマンドは、仮想マシンインスタンスのポートをノードポートに転送し、有効にされたアクセスのサービスを作成します。以下の例では、**fedora-vm-ssh** サービスを作成します。 このサービスは、**<fedora-vm>** 仮想マシンのポート 22 をノード上のポートに転送します。

前提条件

- アクセスする仮想マシンインスタンスは、**masquerade** バインディングメソッド方法を使用してデフォルトの Pod ネットワークに接続されている。
- アクセスする仮想マシンインスタンスが実行中であること。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。

手順

1. 以下のコマンドを実行して **fedora-vm-ssh** サービスを作成します。

```
$ virtctl expose vm <fedora-vm> --port=20022 --target-port=22 --name=fedora-vm-ssh --
type=NodePort (1)
```

(1) **<fedora-vm>** は、**fedora-vm-ssh** サービスを実行する仮想マシンの名前です。

2. サービスをチェックし、サービスが取得したポートを見つけます。

```
$ oc get svc
NAME          TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
fedora-vm-ssh   NodePort   127.0.0.1      <none>        20022:32551/TCP   6s
```

この例では、サービスは **32551** ポートを取得しています。

3. SSH 経由で仮想マシンインスタンスにログインします。ノードの **ipAddress** および直前の手順で確認したポートを使用します。

```
$ ssh username@<node_IP_address> -p 32551
```

2.7.3.2. 仮想マシンインスタンスのシリアルコンソールへのアクセス

virtctl console コマンドは、指定された仮想マシンインスタンスへのシリアルコンソールを開きます。

前提条件

- **virt-viewer** パッケージがインストールされていること。
- アクセスする仮想マシンインスタンスが実行中であること。

手順

- **virtctl** でシリアルコンソールに接続します。

```
$ virtctl console <VMI>
```

2.7.3.3. VNC を使用した仮想マシンインスタンスのグラフィカルコンソールへのアクセス

virtctl クライアントユーティリティーは **remote-viewer** 機能を使用し、実行中の仮想マシンインスタンスに対してグラフィカルコンソールを開くことができます。この機能は **virt-viewer** パッケージに組み込まれています。

前提条件

- **virt-viewer** パッケージがインストールされていること。
- アクセスする仮想マシンインスタンスが実行中であること。

注記

リモートマシンで SSH 経由で **virtctl** を使用する場合、X セッションをマシンに転送する必要があります。

手順

1. **virtctl** ユーティリティーを使用してグラフィカルインターフェースに接続します。

```
$ virtctl vnc <VMI>
```

2. コマンドが失敗した場合には、トラブルシューティング情報を収集するために **-v** フラグの使用を試行します。

```
$ virtctl vnc <VMI> -v 4
```

2.7.3.4. RDP コンソールの使用による Windows 仮想マシンへの接続

Remote Desktop Protocol (RDP) は、Windows 仮想マシンに接続するためのより使いやすいコンソールを提供します。

RDP を使用して Windows 仮想マシンに接続するには、割り当てられた L2 vNIC の IP アドレスを RDP クライアントに対して指定します。

前提条件

- QEMU ゲストエージェントがインストールされた実行中の Windows 仮想マシン。**qemu-guest-agent** は VirtIO ドライバーに含まれています。
- 仮想マシンに接続されたレイヤー2 vNIC。
- Windows 仮想マシンと同じネットワーク上のマシンにインストールされた RDP クライアント。

手順

1. アクセストークンを持つユーザーとして、**oc** CLI ツールを使って Container-native Virtualization クラスターにログインします。

```
$ oc login -u <user> https://<cluster.example.com>:8443
```

2. **oc describe vmi** を使用して、実行中の Windows 仮想マシンの設定を表示します。

```
$ oc describe vmi <windows-vmi-name>
```

```
...
spec:
```

```
  networks:
  - name: default
    pod: {}
  - multus:
      networkName: cnv-bridge
    name: bridge-net
...
status:
  interfaces:
  - interfaceName: eth0
    ipAddress: 198.51.100.0/24
    ipAddresses:
      198.51.100.0/24
    mac: a0:36:9f:0f:b1:70
    name: default
  - interfaceName: eth1
    ipAddress: 192.0.2.0/24
    ipAddresses:
      192.0.2.0/24
      2001:db8::/32
    mac: 00:17:a4:77:77:25
    name: bridge-net
...
```

3. レイヤー 2 ネットワークインターフェースの IP アドレスを特定し、これをコピーします。これは直前の例では **192.0.2.0** であり、IPv6 を選択する場合は **2001:db8::** になります。

4. RDP クライアントを開き、接続用に直前の手順でコピーした IP アドレスを使用します。

5. **Administrator** ユーザー名およびパスワードを入力して、Windows 仮想マシンに接続します。

2.8. CLI ツールの使用

クラスターでリソースを管理するために使用される 2 つの主な CLI ツールは以下の通りです。

- Container-native Virtualization **virtctl** クライアント
- OpenShift Container Platform **oc** クライアント

前提条件

- **virtctl** クライアントをインストールする必要があります。

2.8.1. Virtctl クライアントコマンド

virtctl クライアントは、Container-native Virtualization リソースを管理するためのコマンドラインユーティリティーです。以下の表には、Container-native Virtualization のドキュメント全体で使用されている **virtctl** コマンドが記載されています。

表2.3 virtctl クライアントコマンド

コマンド	説明
virtctl start <vm>	仮想マシンを起動します。

コマンド	説明
virtctl stop <vm>	仮想マシンを停止します。
virtctl restart <vm>	仮想マシンを再起動します。
virtctl expose <vm>	仮想マシンまたは仮想マシンインスタンスの指定されたポートを転送するサービスを作成し、このサービスをノードの指定されたポートで公開します。
virtctl console <vmi>	仮想マシンインスタンスのシリアルコンソールに接続します。
virtctl vnc <vmi>	仮想マシンインスタンスへの VNC 接続を開きます。
virtctl image-upload <...>	仮想マシンイメージを PersistentVolumeClaim にアップロードします。

2.8.2. OpenShift Container Platform クライアントコマンド

OpenShift Container Platform **oc** クライアントは、OpenShift Container Platform リソースを管理するためのコマンドラインユーティリティーです。以下の表には、Container-native Virtualization のドキュメント全体で使用される **oc** コマンドが記載されています。

表2.4 oc コマンド

コマンド	説明
oc login -u <user_name>	OpenShift Container Platform クラスターに **<user_name>** としてログインします。
oc get <object_type>	プロジェクトの指定されたオブジェクトタイプのオブジェクトの一覧を表示します。
oc describe <object_type> <resource_name>	プロジェクトで特定のリソースの詳細を表示します。
oc create -f <object_config>	プロジェクトで、ファイル名または標準入力 (stdin) からリソースを作成します。
oc edit <object_type> <resource_name>	プロジェクトのリソースを編集します。
oc delete <object_type> <resource_name>	プロジェクトのリソースを削除します。

oc client コマンドについてのより総合的な情報については、OpenShift Container Platform CLI ツールのドキュメントを参照してください。

2.9. 管理タスクの自動化

Red Hat Ansible Automation Platform を使用すると、Container-native Virtualization 管理タスクを自動化できます。Ansible Playbook を使用して新規の仮想マシンを作成する際の基本事項を確認します。

2.9.1. Red Hat Ansible Automation について

Ansible は、システムの設定、ソフトウェアのデプロイ、およびローリングアップデートの実行に使用する自動化ツールです。Ansible には Container-native Virtualization のサポートが含まれ、Ansible モジュールを使用すると、テンプレート、Persistent Volume Claim (PVC、永続ボリューム要求) および仮想マシンの操作などのクラスター管理タスクを自動化できます。

Ansible は、**oc** CLI ツールや API を使用しても実行できるContainer-native Virtualization の管理を自動化する方法を提供します。Ansible は、KubeVirt モジュールを他の Ansible モジュールと統合できる点でユニークであると言えます。

2.9.2. 仮想マシン作成の自動化

kubevirt_vm Ansible Playbook を使用し、Red Hat Ansible Automation Platform を使用して OpenShift Container Platform クラスターに仮想マシンを作成できます。

前提条件

- Red Hat Ansible Engine バージョン 2.8 以降

手順

1. **kubevirt_vm** タスクを含むように Ansible Playbook YAML ファイルを編集します。

```
kubevirt_vm:
  namespace:
  name:
  cpu_cores:
  memory:
  disks:
    - name:
      volume:
        containerDisk:
          image:
      disk:
        bus:
```

注記

このスニペットには Playbook の **kubevirt_vm** 部分のみが含まれます。

2. **namespace**、 **cpu_cores** の数、**memory**、および **disks** を含む、作成する必要のある仮想マシンを反映させるように値を編集します。例:

```
kubevirt_vm:
  namespace: default
  name: vm1
  cpu_cores: 1
```

```
    memory: 64Mi
    disks:
      - name: containerdisk
        volume:
          containerDisk:
            image: kubevirt/cirros-container-disk-demo:latest
        disk:
          bus: virtio
```

3. 仮想マシンを作成後すぐに起動する必要がある場合には、**state: running** を YAML ファイルに追加します。例:

```
  kubevirt_vm:
    namespace: default
    name: vm1
    state: running 1
    cpu_cores: 1
```

1 この値を **state: absent** に変更すると、すでに存在する場合に仮想マシンは削除されます。

4. Playbook のファイル名を引数としてのみ使用して、 **ansible-playbook** コマンドを実行します。

```
$ ansible-playbook create-vm.yaml
```

5. 出力を確認し、プレイが正常に実行されたかどうかを確認します。

```
(...)
TASK [Create my first VM] ********************************************************************
changed: [localhost]

PLAY RECAP
*************************************************************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

6. Playbook ファイルに **state: running** を含めず、すぐに仮想マシンを起動する必要がある場合には、 **state: running** を含めるようにファイルを編集し、Playbook を再度実行します。

```
$ ansible-playbook create-vm.yaml
```

仮想マシンが作成されたことを確認するには、仮想マシンコンソールへのアクセスを試行します。

2.9.3. 例: 仮想マシンを作成するための Ansible Playbook

kubevirt_vm Ansible Playbook を使用して仮想マシン作成を自動化できます。

以下の YAML ファイルは **kubevirt_vm** Playbook の例です。これには、Playbook を実行する際に独自の情報を置き換える必要のあるサンプルの値が含まれます。

```
---
- name: Ansible Playbook 1
```

```
hosts: localhost
connection: local
tasks:
  - name: Create my first VM
    kubevirt_vm:
      namespace: default
      name: vm1
      cpu_cores: 1
      memory: 64Mi
      disks:
        - name: containerdisk
          volume:
            containerDisk:
              image: kubevirt/cirros-container-disk-demo:latest
          disk:
            bus: virtio
```

追加情報

- Playbook の概要
- Playbook の検証ツール

2.10. 仮想マシンのデフォルト POD ネットワークの使用

Container-native Virtualization でデフォルトの Pod ネットワークを使用できます。これを実行するには、**masquerade** バインディングメソッドを使用する必要があります。これは、デフォルトの Pod ネットワークを使用する場合にのみ推奨されるバインディングメソッドです。デフォルト以外のネットワークには、**masquerade** モードを使用しないでください。

注記

セカンダリーネットワークの場合は、**bridge** バインディングメソッドを使用します。

2.10.1. コマンドラインでのマスカレードモードの設定

マスカレードモードを使用し、仮想マシンの送信トラフィックを Pod IP アドレスの背後で非表示にすることができます。マスカレードモードは、ネットワークアドレス変換 (NAT) を使用して仮想マシンを Linux ブリッジ経由で Pod ネットワークバックエンドに接続します。

仮想マシンの設定ファイルを編集して、マスカレードモードを有効にし、トラフィックが仮想マシンに到達できるようにします。

前提条件

- 仮想マシンは、IPv4 アドレスを取得するために DHCP を使用できるように設定される必要があります。以下の例では、DHCP を使用するように設定されます。

手順

1. 仮想マシン設定ファイルの **interfaces** 仕様を編集します。

```
kind: VirtualMachine
spec:
```

```
domain:
  devices:
    interfaces:
      - name: red
        masquerade: {} 1
        ports:
          - port: 80 2
networks:
- name: red
  pod: {}
```

1 マスカレードモードを使用した接続

2 ポート 80 での受信トラフィックの許可

2. 仮想マシンを作成します。

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

2.10.2. バインディング方法の選択

Container-native virtualization Web コンソールウィザードから仮想マシンを作成する場合、**Networking** 画面で必要なバインディングメソッドを選択します。

2.10.2.1. ネットワークフィールド

名前	説明
Create NIC	仮想マシンの新規 NIC を作成します。
NIC NAME	NIC の名前。
MAC ADDRESS	ネットワークインターフェースの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。
NETWORK CONFIGURATION	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
BINDING METHOD	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、**masquerade** が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、**bridge** バインディングメソッドを使用します。**masquerade** メソッドは、デフォルト以外のネットワークではサポートされません。
PXE NIC	PXE 対応ネットワークの一覧。**PXE** が **Provision Source** として選択されている場合にのみ表示されます。

2.10.3. デフォルトネットワーク用の仮想マシン設定の例

2.10.3.1. テンプレート: 仮想マシンの設定ファイル

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: default
spec:
  running: false
  template:
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
          - masquerade: {}
            name: default
        resources:
          requests:
            memory: 1024M
      networks:
        - name: default
          pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: kubevirt/fedora-cloud-container-disk-demo
        - name: cloudinitdisk
          cloudInitNoCloud:
            userData: |
              #!/bin/bash
              echo "fedora" | passwd fedora --stdin
```

2.10.3.2. テンプレート: Windows 仮想マシンインスタンスの設定ファイル

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-windows
  name: vmi-windows
spec:
  domain:
    clock:
      timer:
        hpet:
```

```
        present: false
      hyperv: {}
      pit:
        tickPolicy: delay
      rtc:
        tickPolicy: catchup
    utc: {}
  cpu:
    cores: 2
  devices:
    disks:
    - disk:
        bus: sata
      name: pvcdisk
    interfaces:
    - masquerade: {}
      model: e1000
      name: default
  features:
    acpi: {}
    apic: {}
    hyperv:
      relaxed: {}
      spinlocks:
        spinlocks: 8191
      vapic: {}
  firmware:
    uuid: 5d307ca9-b3ef-428c-8861-06e72d69f223
  machine:
    type: q35
  resources:
    requests:
      memory: 2Gi
networks:
- name: default
  pod: {}
terminationGracePeriodSeconds: 0
volumes:
- name: pvcdisk
  persistentVolumeClaim:
    claimName: disk-windows
```

2.11. 仮想マシンの複数ネットワークへの割り当て

Container-native Virtualization は、仮想マシンの複数ネットワークへの接続を可能にするレイヤー 2 ネットワーク機能を提供します。複数インターフェースへのアクセスによって異なる既存のワークロードを持つ仮想マシンをインポートできます。また、仮想マシンをネットワーク経由で起動できるように PXE ネットワークを設定することもできます。

最初に、ネットワーク管理者はタイプ **cnv-bridge** の NetworkAttachmentDefinition を設定します。次に、ユーザーは Pod、仮想マシンインスタンス、および仮想マシンをブリッジネットワークに割り当てることができます。Container-native Virtualization Web コンソールから、ブリッジネットワークを参照する vNIC を作成できます。

2.11.1. Container-native Virtualization ネットワークの用語集

Container-native Virtualization は、カスタムリソースおよびプラグインを使用して高度なネットワーク機能を提供します。

以下の用語は、Container-native Virtualization ドキュメント全体で使用されています。

Container Network Interface (CNI)

コンテナーのネットワーク接続に重点を置く Cloud Native Computing Foundation プロジェクト。Container-native Virtualization は CNI プラグインを使用して基本的な Kubernetes ネットワーク機能を強化します。

Multus

複数の CNI の存在を可能にし、Pod または仮想マシンが必要なインターフェースを使用できるようにする「メタ」 CNI プラグイン。

カスタムリソース定義 (CRD、Customer Resource Definition)

カスタムリソースの定義を可能にする Kubernetes API リソース、または CRD API リソースを使用して定義されるオブジェクト。

NetworkAttachmentDefinition

Pod、仮想マシン、および仮想マシンインスタンスを 1 つ以上のネットワークに割り当てることを可能にする Multus プロジェクトによって導入される CRD。

PXE (Preboot eXecution Environment)

管理者がネットワーク経由でサーバーからクライアントマシンを起動できるようにするインターフェース。ネットワークのブートにより、オペレーティングシステムおよび他のソフトウェアをクライアントにリモートでロードできます。

2.11.2. リソースのブリッジベースのネットワークへの接続

ネットワーク管理者は、タイプ **cnv-bridge** の NetworkAttachmentDefinition をレイヤー 2 ネットワークを Pod および仮想マシンに提供するように設定できます。

前提条件

- Container-native Virtualization 2.0 以降
- Linux ブリッジはすべてのノードで設定され、適切な Network Interface Card に割り当てられる必要があります。
- VLAN を使用する場合、**vlan_filtering** はブリッジで有効にされる必要があります。
- NIC はすべての関連する VLAN に対してタグ付けされる必要があります。
  - 例: **bridge vlan add dev bond0 vid 1-4095 master**

手順

1. 任意のローカルディレクトリーで NetworkAttachmentDefinition の新規ファイルを作成します。このファイルには、お使いの設定に合わせて変更された以下の内容が含まれる必要があります。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
  annotations:
```

```
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br0 (1)
spec:
  config: '{
    "cniVersion": "0.3.1",
    "plugins": [
      {
        "type": "cnv-bridge", (2)
        "bridge": "br0", (3)
        "ipam": {}
      },
      {
        "type": "tuning" (4)
      }
    ]
  }'
```

(1) このアノテーションを NetworkAttachmentDefinition に追加する場合、仮想マシンインスタンスは **br0** ブリッジが接続されているノードでのみ実行されます。

(2) この NetworkAttachmentDefinition のネットワークを提供する Container Network Interface (CNI) プラグインの実際の名前。異なる CNI を使用するのでない限り、このフィールドは変更しないでください。

(3) ブリッジの名前が **br0** でない場合、ブリッジの実際の名前に置き換える必要があります。

(4) 必須。これにより、MAC プールマネージャーが接続に固有の MAC アドレスを割り当てます。

```
$ oc create -f <resource_spec.yaml>
```

2. ブリッジネットワークに接続する必要のある仮想マシンまたは仮想マシンインスタンスの設定を編集します。

```
apiVersion: v1
kind: VirtualMachine
metadata:
  name: example-vm
  annotations:
    k8s.v1.cni.cncf.io/networks: a-bridge-network (1)
spec:
...
```

(1) NetworkAttachmentDefinition の実際の **name** 値に置き換える必要があります。

この例では、NetworkAttachmentDefinition および Pod が同じ namespace に置かれています。

異なる namespace を指定するには、以下の構文を使用します。

```
...
  annotations:
    k8s.v1.cni.cncf.io/networks: <namespace>/a-bridge-network
...
```

3. 設定ファイルをリソースに適用します。

```
$ oc create -f <local/path/to/network-attachment-definition.yaml>
```

注記

次のセクションで vNIC を定義する際に、**NETWORK** の値が直前のセクションで作成した NetworkAttachmentDefinition のブリッジネットワーク名であることを確認します。

2.11.3. 仮想マシンの NIC の作成

Web コンソールから追加の NIC を作成し、これを仮想マシンに割り当てます。

手順

1. Container-native Virtualization コンソールの適切なプロジェクトで、**Workloads → Virtual Machines** をクリックします。
2. 仮想マシンテンプレートを選択します。
3. **Network Interfaces** をクリックし、仮想マシンにすでに割り当てられている NIC を表示します。
4. **Create NIC** をクリックして、新規スロットを一覧に作成します。
5. 新規 NIC の **NAME**、**NETWORK**、**MAC ADDRESS**、および **BINDING METHOD** に入力します。
6. ✓ ボタンをクリックして NIC を保存し、これを仮想マシンに割り当てます。

2.11.4. ネットワークフィールド

名前	説明
Create NIC	仮想マシンの新規 NIC を作成します。
NIC NAME	NIC の名前。
MAC ADDRESS	ネットワークインターフェースの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。
NETWORK CONFIGURATION	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
BINDING METHOD	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、**masquerade** が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、**bridge** バインディングメソッドを使用します。**masquerade** メソッドは、デフォルト以外のネットワークではサポートされません。

名前	説明
PXE NIC	PXE 対応ネットワークの一覧。**PXE** が **Provision Source** として選択されている場合にのみ表示されます。

仮想マシンにオプションの QEMU ゲストエージェントをインストールし、ホストが追加のネットワークについての関連情報を表示できるようにします。

2.12. QEMU ゲストエージェントの仮想マシンへのインストール

QEMU ゲストエージェントは仮想マシンで実行されるデーモンです。エージェントは仮想マシン上で、追加ネットワークの IP アドレスなどのネットワーク情報をホストに渡します。

前提条件

- 以下のコマンドを実行して、ゲストエージェントがインストールされており、実行中であることを確認します。

```
$ systemctl status qemu-guest-agent
```

2.12.1. QEMU ゲストエージェントの Linux 仮想マシンへのインストール

qemu-guest-agent は広く利用されており、Red Hat 仮想マシンでデフォルトで利用できます。このエージェントをインストールし、サービスを起動します。

手順

1. コンソールのいずれか、または SSH を使用して仮想マシンのコマンドラインにアクセスします。
2. QEMU ゲストエージェントを仮想マシンにインストールします。

```
$ yum install -y qemu-guest-agent
```

3. QEMU ゲストエージェントサービスを起動します。

```
$ systemctl start qemu-guest-agent
```

4. サービスに永続性があることを確認します。

```
$ systemctl enable qemu-guest-agent
```

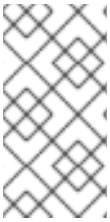
Web コンソールで仮想マシンまたは仮想マシンテンプレートのいずれかを作成する際に、ウィザードの **cloud-init** セクションの **custom script** フィールドを使用して QEMU ゲストエージェントをインストールし、起動することもできます。

2.12.2. QEMU ゲストエージェントの Windows 仮想マシンへのインストール

Windows 仮想マシンの場合、QEMU ゲストエージェントは、以下の手順のいずれかを使用してインストールできる VirtIO ドライバーに含まれています。

2.12.2.1. VirtIO ドライバーの既存 Windows 仮想マシンへのインストール

VirtIO ドライバーを、割り当てられた SATA CD ドライブから既存の Windows 仮想マシンにインストールします。



注記

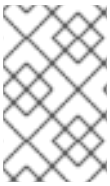
この手順では、ドライバーを Windows に追加するための汎用的なアプローチを使用しています。このプロセスは Windows のバージョンごとに若干異なる可能性があります。特定のインストール手順については、お使いの Windows バージョンについてのインストールドキュメントを参照してください。

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。
2. Windows ユーザーセッションにログインします。
3. **Device Manager** を開き、**Other devices** を拡張して、**Unknown device** を一覧表示します。
    a. **Device Properties** を開いて、不明なデバイスを特定します。デバイスを右クリックし、**Properties** を選択します。
    b. **Details** タブをクリックし、**Property** リストで **Hardware Ids** を選択します。
    c. **Hardware Ids** の **Value** をサポートされる VirtIO ドライバーと比較します。
4. デバイスを右クリックし、**Update Driver Software** を選択します。
5. **Browse my computer for driver software**をクリックし、VirtIO ドライバーが置かれている割り当て済みの SATA CD ドライブの場所に移動します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。
6. **Next** をクリックしてドライバーをインストールします。
7. 必要なすべての VirtIO ドライバーに対してこのプロセスを繰り返します。
8. ドライバーのインストール後に、**Close** をクリックしてウィンドウを閉じます。
9. 仮想マシンを再起動してドライバーのインストールを完了します。

2.12.2.2. Windows インストール時の VirtIO ドライバーのインストール

Windows のインストール時に割り当てられた SATA CD ドライバーから VirtIO ドライバーをインストールします。



注記

この手順では、Windows インストールの汎用的なアプローチを使用しますが、インストール方法は Windows のバージョンごとに異なる可能性があります。インストールする Windows のバージョンについてのドキュメントを参照してください。

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。

2. Windows インストールプロセスを開始します。
3. **Advanced** インストールを選択します。
4. ストレージの宛先は、ドライバーがロードされるまで認識されません。**Load driver** をクリックします。
5. ドライバーは SATA CD ドライブとして割り当てられます。**OK** をクリックし、CD ドライバーでロードするストレージドライバーを参照します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。
6. 必要なすべてのドライバーについて直前の 2 つの手順を繰り返します。
7. Windows インストールを完了します。

2.13. VNIC の IP アドレスの仮想マシンへの表示

QEMU ゲストエージェントは仮想マシンで実行され、割り当てられた vNIC の IP アドレスをホストに渡します。 これにより、Web コンソールおよび **oc** クライアントの両方から IP アドレスを表示できます。

前提条件

1. 以下のコマンドを実行して、ゲストエージェントがインストールされており、実行中であることを確認します。

```
$ systemctl status qemu-guest-agent
```

2. ゲストエージェントがインストールされておらず、実行されていない場合は、仮想マシン上でゲストエージェントをインストールし、実行します。

2.13.1. CLI での仮想マシンインターフェースの IP アドレスの表示

ネットワークインターフェース設定は **oc describe vmi <vmi_name>** コマンドに含まれます。

IP アドレス情報は、仮想マシン上で **ip addr** を実行するか、または **oc get vmi <vmi_name> -o yaml** を実行して表示することもできます。

手順

- **oc describe** コマンドを使用して、仮想マシンインターフェース設定を表示します。

```
$ oc describe vmi <vmi_name>

...
Interfaces:
   Interface Name:  eth0
   Ip Address:      10.244.0.37/24
   Ip Addresses:
     10.244.0.37/24
     fe80::858:aff:fef4:25/64
   Mac:             0a:58:0a:f4:00:25
   Name:            default
   Interface Name:  v2
   Ip Address:      1.1.1.7/24
```

```
Ip Addresses:
  1.1.1.7/24
  fe80::f4d9:70ff:fe13:9089/64
Mac:              f6:d9:70:13:90:89
Interface Name:  v1
Ip Address:      1.1.1.1/24
Ip Addresses:
  1.1.1.1/24
  1.1.1.2/24
  1.1.1.4/24
  2001:de7:0:f101::1/64
  2001:db8:0:f101::1/64
  fe80::1420:84ff:fe10:17aa/64
Mac:              16:20:84:10:17:aa
```

2.13.2. Web コンソールでの仮想マシンインターフェースの IP アドレスの表示

IP 情報は、仮想マシンの **Virtual Machine Overview** 画面に表示されます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines** をクリックします。
2. 仮想マシンの名前をクリックして、**Virtual Machine Overview** 画面を開きます。

それぞれの割り当てられた vNIC の情報は **IP ADDRESSES** の下に表示されます。

2.14. 仮想マシンの PXE ブートの設定

PXE ブートまたはネットワークブートは Container-native Virtualization で利用できます。ネットワークブートにより、ローカルに割り当てられたストレージデバイスなしにコンピューターを起動し、オペレーティングシステムまたは他のプログラムを起動し、ロードすることができます。たとえば、これにより、新規ホストのデプロイ時に PXE サーバーから必要な OS イメージを選択できます。

前提条件

- Linux ブリッジが接続されていること。
- PXE サーバーがブリッジとして同じ VLAN に接続されていること。

2.14.1. Container-native Virtualization ネットワークの用語集

Container-native Virtualization は、カスタムリソースおよびプラグインを使用して高度なネットワーク機能を提供します。

以下の用語は、Container-native Virtualization ドキュメント全体で使用されています。

Container Network Interface (CNI)

コンテナーのネットワーク接続に重点を置く Cloud Native Computing Foundation プロジェクト。Container-native Virtualization は CNI プラグインを使用して基本的な Kubernetes ネットワーク機能を強化します。

Multus

複数の CNI の存在を可能にし、Pod または仮想マシンが必要なインターフェースを使用できるようにする「メタ」 CNI プラグイン。

カスタムリソース定義 (CRD、Customer Resource Definition)

カスタムリソースの定義を可能にする Kubernetes API リソース、または CRD API リソースを使用して定義されるオブジェクト。

NetworkAttachmentDefinition

Pod、仮想マシン、および仮想マシンインスタンスを 1 つ以上のネットワークに割り当てることを可能にする Multus プロジェクトによって導入される CRD。

PXE (Preboot eXecution Environment)

管理者がネットワーク経由でサーバーからクライアントマシンを起動できるようにするインターフェース。ネットワークのブートにより、オペレーティングシステムおよび他のソフトウェアをクライアントにリモートでロードできます。

2.14.2. MAC アドレスを指定した PXE ブート

まず、管理者は PXE ネットワークの NetworkAttachmentDefinition オブジェクトを作成し、ネットワーク経由でクライアントを起動できます。次に、仮想マシンインスタンスの設定ファイルで NetworkAttachmentDefinition を参照して仮想マシンインスタンスを起動します。また PXE サーバーで必要な場合には、仮想マシンインスタンスの設定ファイルで MAC アドレスを指定することもできます。

前提条件

- Linux ブリッジが接続されていること。
- PXE サーバーがブリッジとして同じ VLAN に接続されていること。

手順

1. クラスターに PXE ネットワークを設定します。
   a. PXE ネットワーク **pxe-net-conf** の NetworkAttachmentDefinition ファイルを作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pxe-net-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "pxe-net-conf",
    "plugins": [
      {
        "type": "cnv-bridge",
        "bridge": "br1",
        "ipam": {}
      },
      {
        "type": "cnv-tuning" (1)
      }
    ]
  }'
```

1 **cnv-tuning** プラグインは、カスタム MAC アドレスのサポートを提供します。

注記

仮想マシンインスタンスは、必要な VLAN のアクセスポートでブリッジ **br1** に割り当てられます。

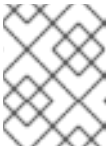
2. 直前の手順で作成したファイルを使用して NetworkAttachmentDefinition オブジェクトを作成します。

```
$ oc create -f pxe-net-conf.yaml
```

3. 仮想マシンインスタンス設定ファイルを、インターフェースおよびネットワークの詳細を含めるように編集します。

   a. PXE サーバーで必要な場合には、ネットワークおよび MAC アドレスを指定します。MAC アドレスが指定されていない場合、値は自動的に割り当てられます。ただし、この時点で自動的に割り当てられる MAC アドレスは永続しないことに注意してください。
   bootOrder が **1** に設定されており、インターフェースが最初に起動することを確認します。この例では、インターフェースは **<pxe-net>** というネットワークに接続されています。

```
interfaces:
- masquerade: {}
  name: default
- bridge: {}
  name: pxe-net
  macAddress: de:00:00:00:00:de
  bootOrder: 1
```



注記

複数のインターフェースおよびディスクのブートの順序はグローバル順序になります。

   b. オペレーティングシステムのプロビジョニング後に起動が適切に実行されるよう、ブートデバイス番号をディスクに割り当てます。
   ディスク **bootOrder** の値を **2** に設定します。

```
devices:
  disks:
  - disk:
      bus: virtio
    name: containerdisk
    bootOrder: 2
```

   c. 直前に作成された NetworkAttachmentDefinition に接続されるネットワークを指定します。このシナリオでは、**<pxe-net>** は **<pxe-net-conf>** という NetworkAttachmentDefinition に接続されます。

```
networks:
- name: default
```

```
    pod: {}
  - name: pxe-net
    multus:
      networkName: pxe-net-conf
```

4. 仮想マシンインスタンスを作成します。

```
$ oc create -f vmi-pxe-boot.yaml
  virtualmachineinstance.kubevirt.io "vmi-pxe-boot" created
```

5. 仮想マシンインスタンスの実行を待機します。

```
$ oc get vmi vmi-pxe-boot -o yaml | grep -i phase
  phase: Running
```

6. VNC を使用して仮想マシンインスタンスを表示します。

```
$ virtctl vnc vmi-pxe-boot
```

7. ブート画面で、PXE ブートが正常に実行されていることを確認します。

8. 仮想マシンインスタンスにログインします。

```
$ virtctl console vmi-pxe-boot
```

9. 仮想マシンのインターフェースおよび MAC アドレスを確認し、ブリッジに接続されたインターフェースに MAC アドレスが指定されていることを確認します。この場合、PXE ブートには IP アドレスなしに **eth1** を使用しています。他のインターフェース **eth0** は OpenShift Container Platform から IP アドレスを取得しています。

```
$ ip addr
...
3. eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen
1000
   link/ether de:00:00:00:00:de brd ff:ff:ff:ff:ff:ff
```

2.14.3. テンプレート: PXE ブートの仮想マシンインスタンス設定ファイル

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  creationTimestamp: null
  labels:
    special: vmi-pxe-boot
  name: vmi-pxe-boot
spec:
  domain:
    devices:
      disks:
      - disk:
          bus: virtio
        name: containerdisk
        bootOrder: 2
```

```
      - disk:
          bus: virtio
        name: cloudinitdisk
      interfaces:
      - masquerade: {}
        name: default
      - bridge: {}
        name: pxe-net
        macAddress: de:00:00:00:00:de
        bootOrder: 1
    machine:
      type: ""
    resources:
      requests:
        memory: 1024M
  networks:
  - name: default
    pod: {}
  - multus:
      networkName: pxe-net-conf
    name: pxe-net
  terminationGracePeriodSeconds: 0
  volumes:
  - name: containerdisk
    containerDisk:
      image: kubevirt/fedora-cloud-container-disk-demo
  - cloudInitNoCloud:
      userData: |
        #!/bin/bash
        echo "fedora" | passwd fedora --stdin
    name: cloudinitdisk
status: {}
```

2.15. ゲストメモリーの管理

ゲストメモリー設定を特定のユースケースに合わせて調整する必要がある場合、ゲストの YAML 設定ファイルを編集してこれを実行できます。Container-native Virtualization は、ゲストメモリーのオーバーコミットの設定と、ゲストメモリーのオーバーコミットアカウンティングの無効化を許可します。

この手順にはどちらの場合もリスクが伴います。そのため、経験のある管理者のみが対応するようにしてください。

2.15.1. ゲストメモリーのオーバーコミットの設定

仮想ワークロードに利用可能な量を上回るメモリーが必要な場合、メモリーのオーバーコミットを使用してホストのメモリーのすべてまたはそのほとんどを仮想マシンインスタンスに割り当てることができます。メモリーのオーバーコミットを有効にすることは、通常ホストに予約されるリソースを最大化できることを意味します。

たとえば、ホストに 32 GB RAM がある場合、メモリーのオーバーコミットを使用してそれぞれ 4 GB RAM を持つ 8 つの仮想マシンに対応できます。これは、仮想マシンがそれらのメモリーのすべてを同時に使用しないという前提で機能します。

手順

1. 仮想マシンインスタンスに対し、クラスターから要求された以上のメモリーが利用可能であることを明示的に示すために、仮想マシン設定ファイルを編集し、**spec.domain.memory.guest** を **spec.domain.resources.requests.memory** よりも高い値に設定します。このプロセスはメモリーのオーバーコミットと呼ばれています。
以下の例では、**1024M** がクラスターから要求されますが、仮想マシンインスタンスには **2048M** が利用可能であると通知されます。ノードに利用可能な空のメモリーが十分にある限り、仮想マシンインスタンスは最大 2048M を消費します。

```
kind: VirtualMachine
spec:
  template:
    domain:
    resources:
        requests:
          memory: 1024M
    memory:
        guest: 2048M
```

注記

ノードがメモリー不足の状態になると、Pod のエビクションルールと同じルールが仮想マシンインスタンスに適用されます。

2. 仮想マシンを作成します。

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

2.15.2. ゲストメモリーオーバーヘッドアカウンティングの無効化

警告

この手順は、特定のユースケースでのみ有効であり、上級ユーザーのみが試行するようにしてください。

要求する量に加えて、少量のメモリーが各仮想マシンインスタンスによって要求されます。追加のメモリーは、それぞれの **VirtualMachineInstance** プロセスをラップするインフラストラクチャーに使用されます。

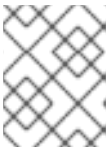
通常は推奨される方法ではありませんが、ゲストメモリーオーバーヘッドアカウンティングを無効にすることでノード上の仮想マシンインスタンスの密度を増やすことは可能です。

手順

1. ゲストメモリーオーバーヘッドアカウンティングを無効にするには、YAML 設定ファイルを編集し、**overcommitGuestOverhead** の値を **true** に設定します。このパラメーターはデフォルトで無効にされています。

```
kind: VirtualMachine
```

```
spec:
  template:
    domain:
    resources:
      overcommitGuestOverhead: true
      requests:
        memory: 1024M
```



注記

overcommitGuestOverhead が有効にされている場合、これはゲストのオーバーヘッドをメモリー制限 (ある場合) に追加します。

2. 仮想マシンを作成します。

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

2.16. 仮想マシンテンプレートの作成

仮想マシンテンプレートの使用は、同様の設定を持つ複数の仮想マシンを作成するための簡単な方法です。テンプレートの作成後、仮想マシンの作成時にテンプレートを参照できます。

2.16.1. Web コンソールでのインタラクティブなウィザードを使用した仮想マシンテンプレートの作成

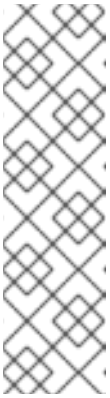
Web コンソールは、**Basic Settings**、**Networking**、および **Storage** 画面にナビゲートし、仮想マシンテンプレートの作成プロセスを単純化するインタラクティブなウィザードを特長としています。すべての必須フィールドには * のマークが付けられます。ウィザードは必須フィールドの入力が完了するまで次の画面に移動することを防ぎます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machine Templates** をクリックします。
2. **Create Template** をクリックし、**Create with Wizard** を選択します。
3. すべての必須の **Basic Settings** を入力します。
4. **Next** をクリックして **Networking** 画面に進みます。デフォルトで **nic0** という名前の NIC が割り当てられます。
   a. オプション: **Create NIC** をクリックして追加の NIC を作成します。
   b. オプション: Options メニュー  をクリックし、**Remove NIC** を選択していずれか、またはすべての NIC を削除できます。テンプレートから作成される仮想マシンには、割り当てられている NIC は不要です。NIC は仮想マシンの作成後に作成できます。
5. **Next** をクリックして **Storage** 画面に進みます。
   a. オプション: **Create Disk** をクリックして追加のディスクを作成します。

b. オプション: ディスクをクリックして選択可能なフィールドを変更します。✓ ボタンをクリックして変更を保存します。

c. オプション: **Attach Disk** をクリックして、**Select Storage** リストから利用可能なディスクを選択します。



注記

URL または **Container** のいずれかが **Basic Settings** 画面で **Provision Source** として選択されてる場合、**rootdisk** ディスクが作成され、**Bootable Disk** として仮想マシンに割り当てられます。**rootdisk** を変更できますが、これを削除することはできません。

Bootable Disk は、仮想マシンにディスクが割り当てられていない場合、**PXE** ソースからプロビジョニングされる仮想マシンには不要です。1 つ以上のディスクが仮想マシンに割り当てられている場合、**Bootable Disk** を 1 つを選択する必要があります。

6. **Create Virtual Machine Template** >をクリックします。**Results** 画面には、仮想マシンテンプレートの JSON 設定ファイルが表示されます。
テンプレートは **Workloads** → **Virtual Machine Templates** に一覧表示されます。

2.16.2. 仮想マシンテンプレートのインタラクティブなウィザードのフィールド

以下の表は、**Create Virtual Machine Template**のインタラクティブなウィザードの **Basic Settings**、**Networking**、および **Storage** ペインのフィールドを説明しています。

2.16.2.1. 仮想マシンテンプレートウィザードのフィールド

名前	パラメーター	説明
名前		この名前には、小文字 (**a-z**)、数字 (**0-9**)、およびハイフン (**-**) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (.)、または特殊文字を使用できません。
説明		オプションの説明フィールド。
Provision Source	PXE	PXE メニューから仮想マシンをプロビジョニングします。クラスターに PXE 対応の NIC が必要になります。
	URL	**HTTP** または **S3** エンドポイントで利用できるイメージから仮想マシンをプロビジョニングします。

名前	パラメーター	説明
	Container	クラスターからアクセスできるレジストリーの起動可能なオペレーティングシステムコンテナーから仮想マシンをプロビジョニングします。例: **kubevirt/cirros-registry-disk-demo**
	Cloned Disk	プロビジョニングソースはクローン作成されたディスクです。
	Import	サポートされているプロバイダーから仮想マシンをインポートします。
Operating System		クラスターで利用可能なオペレーティングシステムの一覧。これは、仮想マシンの主なオペレーティングシステムになります。**Import** を Provider Source として選択する場合、オペレーティングシステムはインポートされる VMware 仮想マシンのオペレーティングシステムに基づいて自動的に入力されます。
Flavor	small、medium、large、tiny、Custom	仮想マシンに割り当てられる CPU およびメモリーの量を決定するプリセット。
Workload Profile	desktop	デスクトップで使用するための仮想マシン設定。
	generic	各種のワークロードについてのパフォーマンスと互換性のバランスを取るための仮想マシンの設定。
	high performance	高パフォーマンスの負荷に対して最適化された仮想マシン設定。
Use cloud-init		これを選択し、cloud-init フィールドを有効にします。

2.16.2.2. Cloud-init フィールド

名前	説明
Hostname	仮想マシンの特定のホスト名を設定します。
Authenticated SSH Keys	仮想マシンの **~/.ssh/authorized_keys** にコピーされるユーザーの公開鍵。
Use custom script	他のオプションを、カスタム cloud-init スクリプトを貼り付けるフィールドに置き換えます。

2.16.2.3. ネットワークフィールド

名前	説明
Create NIC	仮想マシンの新規 NIC を作成します。
NIC NAME	NIC の名前。
MAC ADDRESS	ネットワークインターフェースの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。
NETWORK CONFIGURATION	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
BINDING METHOD	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、**masquerade** が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、**bridge** バインディングメソッドを使用します。**masquerade** メソッドは、デフォルト以外のネットワークではサポートされません。
PXE NIC	PXE 対応ネットワークの一覧。**PXE** が **Provision Source** として選択されている場合にのみ表示されます。

2.16.2.4. ストレージフィールド

名前	説明
Create Disk	仮想マシンの新規ディスクを作成します。
Attach Disk	利用可能な PVC の一覧から、仮想マシンに割り当てる既存のディスクを選択します。

名前	説明
DISK NAME	ディスクの名前。この名前には、小文字 (**a-z**)、数字 (**0-9**)、ハイフン (**-**) およびピリオド (**.**) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、または特殊文字を使用できません。
SIZE (GB)	ディスクのサイズ (GB)。
STORAGE CLASS	基礎となる **StorageClass** の名前。
Bootable Disk	仮想マシンの起動に利用できるディスクの一覧。仮想マシンの Provision Source が **URL** または **Container** の場合に **rootdisk** に固定されます。

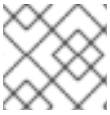
2.17. 仮想マシンテンプレートの編集

Web コンソールで仮想マシンテンプレートを削除できます。

2.17.1. Web コンソールでの仮想マシンテンプレートの編集

Web コンソールで仮想マシンテンプレートの YAML 設定を編集できます。

すべてのパラメーターが変更可能である訳ではありません。無効な設定で **Save** をクリックすると、エラーメッセージで変更できないパラメーターが示唆されます。



注記

編集中に YAML 画面から離れると、設定に対して加えた変更が取り消されます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machine Templates** をクリックします。
2. テンプレートを選択します。
3. **YAML** タブをクリックして編集可能な設定を表示します。
4. ファイルを編集し、**Save** をクリックします。

オブジェクトの更新されたバージョン番号を含め、変更が正常に行われたことを示す確認メッセージが表示されます。

2.17.2. 仮想マシンテンプレートへの仮想ディスクの追加

仮想マシンへのディスクの追加

手順

1. **Virtual Machine Templates** タブで、仮想マシンテンプレートを選択します。
2. **Disks** タブを選択します。
3. **Add Disks** をクリックして、 **Add Disk** ウィンドウを開きます。
4. **Add Disk** ウィンドウで、**Source**、**Name**、 **Size**、**Interface**、および **Storage Class** を指定します。
5. ドロップダウンリストおよびチェックボックスを使用して、ディスク設定を編集します。
6. **OK** をクリックします。

2.17.3. ネットワークインターフェースの仮想マシンテンプレートへの追加

手順

1. **Virtual Machine Templates** タブで、仮想マシンテンプレートを選択します。
2. **Network Interfaces** タブを選択します。
3. **Create Network Interface** をクリックします。
4. **Create Network Interface** 一覧の行で、ネットワークインターフェースの **Name**、**Model**、**Network**、**Type**、および **MAC Address** を指定します。
5. 行の右にある緑色のチェックマークをクリックして、ネットワークインターフェースを追加します。
6. 仮想マシンを再起動して、アクセスを有効にします。
7. ドロップダウンリストとチェックボックスを編集して、ネットワークインターフェースを設定します。
8. **Save Changes** をクリックします。
9. **OK** をクリックします。

新規のネットワークインターフェースは、ユーザーが再起動するまで **Create Network Interface** 一覧の上部に表示されます。

新規のネットワークインターフェースには、仮想マシンを再起動するまで **Pending VM restart** のリンク状態が表示されます。Link State (リンク状態) にカーソルを合わせて詳細情報を表示します。

ネットワークインターフェースカードが仮想マシンで定義され、ネットワークに接続されている場合は、**Link State** はデフォルトで **Up** に設定されます。

2.18. 仮想マシンテンプレートの削除

Web コンソールで仮想マシンテンプレートを削除できます。

2.18.1. Web コンソールでの仮想マシンテンプレートの削除

仮想マシンテンプレートを削除すると、仮想マシンはクラスターから永続的に削除されます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machine Templates** をクリックします。
2. このペインから仮想マシンテンプレートを削除できます。 これにより、1 つのペインで複数のマシンに対してアクションを実行することがより容易になります。 または、**Virtual Machine Template Details** ペインから仮想マシンテンプレートを削除することもできます。 この場合、選択されたテンプレートの総合的な詳細情報を確認できます。

   - Options メニュー  をクリックし、**Delete Template** を選択します。
   - テンプレート名をクリックして **Virtual Machine Template Details**ペインを開き、**Actions → Delete Template** をクリックします。

3. 確認のポップアップウィンドウで、**Delete** をクリックし、テンプレートを永続的に削除します。

2.19. 新規 DATAVOLUME への仮想マシンディスクのクローン作成

DataVolume 設定ファイルでソース PVC を参照し、新規 DataVolume に仮想マシンディスクの PersistentVolumeClaim (PVC) のクローンを作成できます。

前提条件

- この操作を正常に実行するためには、StorageClass を定義するか、CDI のスクラッチ領域を用意する必要がある場合があります。CDI がサポートする操作マトリックスは、スクラッチ領域を必要とする条件を示しています。

2.19.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.19.2. 新規 DataVolume への仮想マシンディスクの PersistentVolumeClaim のクローン作成

既存の仮想マシンディスクの PersistentVolumeClaim (PVC) のクローンを新規 DataVolume に作成できます。その後、新規 DataVolume は新規の仮想マシンに使用できます。

注記

Volume が仮想マシンとは別に作成される場合、DataVolume のライフサイクルは仮想マシンから切り離されます。仮想マシンが削除されても、DataVolume もその関連付けられた PVC も削除されません。

前提条件

- 使用する既存の仮想マシンディスクの PVC を判別します。クローン作成の前に、PVC に関連付けられた仮想マシンの電源を切る必要があります。

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。

手順

1. 関連付けられた PVC の名前および namespace を特定するために、クローン作成に必要な仮想マシンディスクを確認します。

2. 新規 DataVolume の名前、ソース PVC の名前および namespace、および新規 DataVolume のサイズを指定する DataVolume オブジェクトの YAML ファイルを作成します。
例:

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <cloner-datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source-namespace>" 2
      name: "<my-favorite-vm-disk>" 3
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 4
```

1. 新規 DataVolume の名前。
2. ソース PVC が存在する namespace。
3. ソース PVC の名前。
4. 新規 DataVolume のサイズ。十分な領域を割り当てる必要があります。そうでない場合には、クローン操作は失敗します。サイズはソース PVC と同じか、またはそれよりも大きくなければなりません。

3. DataVolume を作成して PVC のクローン作成を開始します。

```
$ oc create -f <cloner-datavolume>.yaml
```

注記

DataVolume は仮想マシンが PVC の作成前に起動することを防ぐため、PVC のクローン作成中に新規 DataVolume を参照する仮想マシンを作成できます。

2.19.3. テンプレート: DataVolume クローン設定ファイル

example-clone-dv.yaml

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
```

```
  name: "example-clone-dv"
spec:
  source:
    pvc:
      name: source-pvc
      namespace: example-ns
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: "1G"
```

2.19.4. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.20. DATAVOLUMETEMPLATE の使用による仮想マシンのクローン作成

既存の仮想マシンの PersistentVolumeClaim (PVC) のクローン作成により、新規の仮想マシンを作成できます。**dataVolumeTemplate** を仮想マシン設定ファイルに含めることにより、元の PVC から新規の DataVolume を作成します。

前提条件

- この操作を正常に実行するためには、StorageClass を定義するか、CDI のスクラッチ領域を用意する必要がある場合があります。CDI がサポートする操作マトリクスは、スクラッチ領域を必要とする条件を表示します。

2.20.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.20.2. DataVolumeTemplate の使用による、クローン作成された PersistentVolumeClaim からの仮想マシンの新規作成

既存の仮想マシンの PersistentVolumeClaim (PVC) のクローンを DataVolume に作成する仮想マシンを作成できます。仮想マシン **spec** の **dataVolumeTemplate** を参照することにより、**source** PVC のクローンが DataVolume に作成され、これは次に仮想マシンを作成するために自動的に使用されます。

注記

DataVolume が仮想マシンの DataVolumeTemplate の一部として作成されると、DataVolume のライフサイクルは仮想マシンに依存します。つまり、仮想マシンが削除されると、DataVolume および関連付けられた PVC も削除されます。

前提条件

- 使用する既存の仮想マシンディスクの PVC を判別します。クローン作成の前に、PVC に関連付けられた仮想マシンの電源を切る必要があります。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。

手順

1. 関連付けられた PVC の名前および namespace を特定するために、クローン作成に必要な仮想マシンを確認します。
2. **VirtualMachine** オブジェクトの YAML ファイルを作成します。以下の仮想マシンのサンプルでは、**source-namespace** namespace にある **my-favorite-vm-disk** のクローンを作成します。**favorite-clone** という **2Gi** DataVolume が **my-favorite-vm-disk** から作成されます。
例:

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-dv-clone
  name: vm-dv-clone 1
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-dv-clone
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
```

```
        name: root-disk
      resources:
        requests:
          memory: 64M
    volumes:
    - dataVolume:
        name: favorite-clone
      name: root-disk
  dataVolumeTemplates:
  - metadata:
      name: favorite-clone
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 2Gi
      source:
        pvc:
          namespace: "source-namespace"
          name: "my-favorite-vm-disk"
```

1 作成する仮想マシン。

3. PVC のクローンが作成された DataVolume で仮想マシンを作成します。

```
$ oc create -f <vm-clone-datavolumetemplate>.yaml
```

2.20.3. テンプレート: DataVolume 仮想マシン設定ファイル

example-dv-vm.yaml

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: example-vm
  name: example-vm
spec:
  dataVolumeTemplates:
  - metadata:
      name: example-dv
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 1G
      source:
        http:
          url: "" 1
  running: false
```

```
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
        devices:
          disks:
          - disk:
              bus: virtio
            name: example-dv-disk
        machine:
          type: q35
        resources:
          requests:
            memory: 1G
      terminationGracePeriodSeconds: 0
      volumes:
      - dataVolume:
          name: example-dv
        name: example-dv-disk
```

1 インポートする必要のあるイメージの **HTTP** ソース (該当する場合)。

2.20.4. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.21. VIRTCTL ツールの使用によるローカルディスクイメージのアップロード

virtctl コマンドラインユーティリティーを使用して、ローカルに保存されるディスクイメージをアップロードできます。

前提条件

- **kubevirt-virtctl** パッケージのインストール
- この操作を正常に実行するためには、StorageClass を定義するか、CDI のスクラッチ領域を用意する必要がある場合があります。

2.21.1. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.21.2. ローカルディスクイメージの新規 PersistentVolumeClaim へのアップロード

virtctl CLI ユーティリティーを使用し、仮想マシンディスクイメージをクライアントマシンからクラスターにアップロードできます。ディスクイメージをアップロードすることにより、仮想マシンに関連付けることのできる PersistentVolumeClaim (PVC) が作成されます。

前提条件

- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- **kubevirt-virtctl** パッケージがクライアントマシンにインストールされていること。
- クライアントマシンが OpenShift Container Platform ルーターの証明書を信頼するように設定されていること。

手順

1. 以下を特定します。
   - アップロードする VM ディスクイメージのファイルの場所
   - 結果として生成される PVC の名前およびこれに必要なサイズ
2. **virtctl image-upload** コマンドを実行して仮想マシンイメージをアップロードします。PVC 名、PVC サイズ、およびファイルの場所を指定する必要があります。以下は例になります。

```
$ virtctl image-upload --pvc-name=<upload-fedora-pvc> --pvc-size=<2Gi> --image-path=
</images/fedora.qcow2>
```

   注意

   HTTPS を使用したセキュアでないサーバー接続を許可するには、**--insecure** パラメーターを使用します。**--insecure** フラグを使用する際に、アップロードエンドポイントの信頼性は検証 **されない** 点に注意してください。

3. PVC が作成されていることを確認するには、すべての PVC オブジェクトを表示します。

```
$ oc get pvc
```

2.22. ブロックストレージ DATAVOLUME へのローカルディスクイメージのアップロード

virtctl コマンドラインユーティリティーを使用して、ローカルのディスクイメージをブロック DataVolume にアップロードできます。

このワークフローでは、ローカルブロックデバイスを使用して PersistentVolume を使用し、このブロックボリュームを **upload** DataVolume に関連付け、 **virtctl** を使用してローカルディスクイメージを DataVolume にアップロードできます。

前提条件

- CDI でサポートされる操作マトリックスに応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように StorageClass を定義するか、またはCDI スクラッチ領域を用意します。

2.22.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は

KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.22.2. ブロック PersistentVolume について

ブロック PersistentVolume (PV) は、raw ブロックデバイスによってサポートされる PV です。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仮想マシンや、独自のストレージサービスを実装する仮想マシ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 PersistentVolumeClaim (PVC) 仕様で **volumeMode: Block** を指定してプロビジョニングされます。

2.22.3. ローカルブロック PersistentVolume の作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック PersistentVolume (PV) を作成します。次に、このループデバイスを PV 設定で **Block** ボリュームとして参照し、これを仮想マシンイメージのブロックデバイスとして使用できます。

手順

1. ローカル PV を作成するノードに **root** としてログインします。この手順では、**node01** を例に使用します。
2. ファイルを作成して、これを null 文字で設定し、ブロックデバイスとして使用できるようにします。以下の例では、2Gb (20 100Mb ブロック) のサイズのファイル **loop10** を作成します。

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. **loop10** ファイルをループデバイスとしてマウントします。

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

1. ループデバイスがマウントされているファイルパスです。
2. 前の手順で作成したファイルはループデバイスとしてマウントされます。

4. マウントされたループデバイスを参照する **PersistentVolume** 設定を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> 1
  capacity:
    storage: <2Gi>
  volumeMode: Block 2
  storageClassName: local 3
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
```

```
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - <node01> 4
```

1. ノード上のループデバイスのパス。
2. ブロック PV であることを指定します。
3. オプション: PV に StorageClass を設定します。これを省略する場合、クラスターのデフォルトが使用されます。
4. ブロックデバイスがマウントされたノード。

5. ブロック PV を作成します。

```
# oc create -f <local-block-pv10.yaml> 1
```

1. 直前の手順で作成された PersistentVolume のファイル名。

2.22.4. アップロード DataVolume の作成

ローカルディスクイメージのアップロードに使用する **upload** データソースで DataVolume を作成します。

手順

1. **spec: source: upload{}** を指定する DataVolume 設定を作成します。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1
spec:
  source:
      upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 2
```

1. DataVolume の名前
2. DataVolume のサイズ

2. DataVolume を作成します。

```
$ oc create -f <upload-datavolume>.yaml
```

2.22.5. ローカルディスクイメージの新規 DataVolume へのアップロード

virtctl CLI ユーティリティーを使用し、仮想マシンディスクイメージをクライアントマシンからクラスター内の DataVolume (DV) にアップロードします。イメージのアップロード後に、仮想マシンに追加できます。

前提条件

- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- **kubevirt-virtctl** パッケージがクライアントマシンにインストールされていること。
- クライアントマシンが OpenShift Container Platform ルーターの証明書を信頼するように設定されていること。
- アップロードするディスクと同じか、これより大きいスペア DataVolume。

手順

1. 以下を特定します。
   - アップロードする仮想マシンディスクイメージのファイルの場所
   - DataVolume の名前
2. **virtctl image-upload** コマンドを実行してディスクイメージをアップロードします。DV 名とファイルの場所を指定する必要があります。以下は例になります。

```
$ virtctl image-upload --dv-name=<upload-datavolume> --image-path=
</images/fedora.qcow2> (1) (2)
```

(1) 作成する DataVolume の名前。

(2) アップロードする仮想マシンディスクイメージのファイルパス。

注意

HTTPS を使用したセキュアでないサーバー接続を許可するには、**--insecure** パラメーターを使用します。**--insecure** フラグを使用する際に、アップロードエンドポイントの信頼性は検証 **されない** 点に注意してください。

3. DV が作成されていることを確認するには、すべての DV オブジェクトを表示します。

```
$ oc get dvs
```

2.22.6. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.23. 空のディスクイメージを追加して仮想ストレージを拡張する

空のディスクイメージを Container-native Virtualization に追加することによって、ストレージ容量を拡張したり、新規のデータパーティションを作成したりできます。

2.23.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.23.2. DataVolume を使用した空のディスクイメージの作成

DataVolume 設定ファイルをカスタマイズし、デプロイすることにより、新規の空のディスクイメージを PersistentVolumeClaim に作成することができます。

前提条件

- 1 つ以上の利用可能な PersistentVolume
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) をダウンロードします。

手順

1. DataVolume 設定ファイルを編集します。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
```

```
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi
```

2. 以下のコマンドを実行して、空のディスクイメージを作成します。

```
$ oc create -f <blank-image-datavolume>.yaml
```

2.23.3. テンプレート: 空のディスクイメージの DataVolume 設定ファイル

blank-image-datavolume.yaml

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi
```

2.24. CDI のスクラッチ領域の用意

2.24.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.24.2. スクラッチ領域について

Containerized Data Importer (CDI) では、仮想マシンイメージのインポートやアップロードなどの、一部の操作を実行するためにスクラッチ領域 (一時ストレージ) が必要になります。このプロセスで、CDI は、宛先 DataVolume (DV) をサポートする PVC のサイズと同じサイズのスクラッチ領域 PVC をプロビジョニングします。スクラッチ領域 PVC は操作の完了または中止後に削除されます。

CDIConfig オブジェクトにより、 **scratchSpaceStorageClass** を CDIConfig オブジェクトの **spec:** セクションに指定して、スクラッチ領域 PVC をバインドするために使用する StorageClass を定義することができます。

定義された StorageClass がクラスターの StorageClass に一致しない場合、クラスターに定義されたデフォルト StorageClass が使用されます。クラスターで定義されたデフォルトの StorageClass がない場合、元の DV または PVC のプロビジョニングに使用される StorageClass が使用されます。

注記

CDI では、元の DataVolume をサポートする PVC の種類を問わず、**file** ボリュームモードが設定されているスクラッチ領域が必要です。元の PVC が **block** ボリュームモードでサポートされる場合、**file** ボリュームモード PVC をプロビジョニングできる StorageClass を定義する必要があります。

手動プロビジョニング
ストレージクラスがない場合、CDI はイメージのサイズ要件に一致するプロジェクトの PVC を使用します。これらの要件に一致する PVC がない場合、CDI インポート Pod は適切な PVC が利用可能になるまで、またはタイムアウト機能が Pod を強制終了するまで **Pending** 状態になります。

2.24.3. スクラッチ領域を必要とする CDI 操作

タイプ	理由
レジストリーのインポート	CDI はイメージをスクラッチ領域にダウンロードし、イメージファイルを見つけるためにレイヤーを抽出する必要があります。その後、raw ディスクに変換するためにイメージファイルが QEMU-img に渡されます。
イメージのアップロード	QEMU-IMG は STDIN の入力を受け入れません。代わりに、アップロードするイメージは、変換のために QEMU-IMG に渡される前にスクラッチ領域に保存されます。
アーカイブされたイメージの HTTP インポート	QEMU-IMG は、CDI がサポートするアーカイブ形式の処理方法を認識しません。イメージが QEMU-IMG に渡される前にアーカイブは解除され、スクラッチ領域に保存されます。
認証されたイメージの HTTP インポート	QEMU-IMG が認証を適切に処理しません。イメージが QEMU-IMG に渡される前にスクラッチ領域に保存され、認証されます。
カスタム証明書の HTTP インポート	QEMU-IMG は HTTPS エンドポイントのカスタム証明書を適切に処理しません。代わりに CDI は、ファイルを QEMU-IMG に渡す前にイメージをスクラッチ領域にダウンロードします。

2.24.4. CDI 設定での StorageClass の定義

CDI 設定で StorageClass を定義し、CDI 操作のスクラッチ領域を動的にプロビジョニングします。

手順

- **oc** クライアントを使用して **cdiconfig/config** を編集し、**spec: scratchSpaceStorageClass** を追加または編集してクラスターの StorageClass に一致させます。

```
$ oc edit cdiconfig/config
```

```
API Version:  cdi.kubevirt.io/v1alpha1
kind: CDIConfig
metadata:
  name: config
...
spec:
  scratchSpaceStorageClass: "<storage_class>"
...
```

2.24.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

追加リソース

- StorageClass およびこれらがクラスターで定義される方法についての詳細は、「Dynamic provisioning」セクションを参照してください。

2.25. DATAVOLUME の使用による仮想マシンイメージのブロックストレージへのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスターにインポートできます。Container-native Virtualization は DataVolume を使用してデータのインポートおよび基礎となる PersistentVolumeClaim (PVC) の作成を自動化します。

注意

ディスクイメージを PVC にインポートする際に、ディスクイメージは PVC で要求されるストレージの全容量を使用するように拡張されます。この領域を使用するには、仮想マシンのディスクパーティションおよびファイルシステムの拡張が必要になる場合があります。

サイズ変更の手順は、仮想マシンにインストールされるオペレーティングシステムによって異なります。詳細は、該当するオペレーティングシステムのドキュメントを参照してください。

前提条件

- CDI でサポートされる操作マトリックスに応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように StorageClass を定義するか、またはCDI スクラッチ領域を用意します。

2.25.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.25.2. ブロック PersistentVolume について

ブロック PersistentVolume (PV) は、raw ブロックデバイスによってサポートされる PV です。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仮想マシンや、独自のストレージサービスを実装する仮想マシ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 PersistentVolumeClaim (PVC) 仕様で **volumeMode: Block** を指定してプロビジョニングされます。

2.25.3. ローカルブロック PersistentVolume の作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック PersistentVolume (PV) を作成します。次に、このループデバイスを PV 設定で **Block** ボリュームとして参照し、これを仮想マシンイメージのブロックデバイスとして使用できます。

手順

1. ローカル PV を作成するノードに **root** としてログインします。この手順では、**node01** を例に使用します。

2. ファイルを作成して、これを null 文字で設定し、ブロックデバイスとして使用できるようにします。以下の例では、2Gb (20 100Mb ブロック) のサイズのファイル **loop10** を作成します。

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. **loop10** ファイルをループデバイスとしてマウントします。

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

1. ループデバイスがマウントされているファイルパスです。
2. 前の手順で作成したファイルはループデバイスとしてマウントされます。

4. マウントされたループデバイスを参照する **PersistentVolume** 設定を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> 1
  capacity:
    storage: <2Gi>
  volumeMode: Block 2
  storageClassName: local 3
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - <node01> 4
```

1. ノード上のループデバイスのパス。
2. ブロック PV であることを指定します。
3. オプション: PV に StorageClass を設定します。これを省略する場合、クラスターのデフォルトが使用されます。
4. ブロックデバイスがマウントされたノード。

5. ブロック PV を作成します。

```
# oc create -f <local-block-pv10.yaml> 1
```

1. 直前の手順で作成された PersistentVolume のファイル名。

2.25.4. DataVolume を使用した仮想マシンイメージのブロック PersistentVolume へのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスターにインポートできます。Container-native Virtualization は DataVolume を使用してデータのインポートおよび基礎となる PersistentVolumeClaim (PVC) の作成を自動化します。その後、仮想マシン設定で DataVolume を参照できます。

前提条件

- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- データソースにアクセスするために必要な認証情報と共にイメージがホストされる **HTTP** または **s3** エンドポイント
- 少なくとも 1 つ以上の利用可能なブロック PV。

手順

1. データソースに認証情報が必要な場合、**endpoint-secret.yaml** ファイルを編集し、更新された設定をクラスターに適用します。
   a. 選択するテキストエディターで **endpoint-secret.yaml** ファイルを編集します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <endpoint-secret>
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" (1)
  secretKey:   "" (2)
```

(1) オプション: キーまたはユーザー名 (base64 エンコード)

(2) オプション: シークレットまたはパスワード、base64 エンコード

   b. シークレットを更新します。

```
$ oc apply -f endpoint-secret.yaml
```

2. インポートするイメージのデータソースを指定する **DataVolume** および **volumeMode: Block** を作成して、利用可能なブロック PV が使用されるようにします。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <import-pv-datavolume> (1)
spec:
  storageClassName: local (2)
  source:
     http:
       url:
<http://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora
```

```
-Cloud-Base-28-1.1.x86_64.qcow2> 3
      secretRef: <endpoint-secret> 4
  pvc:
    volumeMode: Block 5
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi>
```

1. DataVolume の名前。
2. オプション: ストレージクラスを設定するか、またはこれを省略してクラスターのデフォルトを受け入れます。
3. インポートするイメージの **HTTP** ソース。
4. データソースに認証が必要である場合にのみ必要です。
5. ブロック PV にインポートするために必要です。

3. 仮想マシンイメージをインポートするために DataVolume を作成します。

```
$ oc create -f <import-pv-datavolume.yaml> 1
```

1. 直前の手順で作成されたファイル名 DataVolume。

2.25.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.26. 新規ブロックストレージ DATAVOLUME への仮想マシンディスクのクローン作成

DataVolume 設定ファイルでソース PVC を参照し、新規ブロック DataVolume に仮想マシンディスクの PersistentVolumeClaim (PVC) のクローンを作成できます。

前提条件

- CDI でサポートされる操作マトリックスに応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように StorageClass を定義するか、またはCDI スクラッチ領域を用意します。

2.26.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

2.26.2. ブロック PersistentVolume について

ブロック PersistentVolume (PV) は、raw ブロックデバイスによってサポートされる PV です。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仮想マシンや、独自のストレージサービスを実装する仮想マシ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 PersistentVolumeClaim (PVC) 仕様で **volumeMode: Block** を指定してプロビジョニングされます。

2.26.3. ローカルブロック PersistentVolume の作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック PersistentVolume (PV) を作成します。次に、このループデバイスを PV 設定で **Block** ボリュームとして参照し、これを仮想マシンイメージのブロックデバイスとして使用できます。

手順

1. ローカル PV を作成するノードに **root** としてログインします。この手順では、**node01** を例に使用します。
2. ファイルを作成して、これを null 文字で設定し、ブロックデバイスとして使用できるようにします。以下の例では、2Gb (20 100Mb ブロック) のサイズのファイル **loop10** を作成します。

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. **loop10** ファイルをループデバイスとしてマウントします。

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

1 ループデバイスがマウントされているファイルパスです。

2 前の手順で作成したファイルはループデバイスとしてマウントされます。

4. マウントされたループデバイスを参照する **PersistentVolume** 設定を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> 1
  capacity:
    storage: <2Gi>
  volumeMode: Block 2
  storageClassName: local 3
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - <node01> 4
```

1 ノード上のループデバイスのパス。

2 ブロック PV であることを指定します。

3 オプション: PV に StorageClass を設定します。これを省略する場合、クラスターのデフォルトが使用されます。

4 ブロックデバイスがマウントされたノード。

5. ブロック PV を作成します。

```
# oc create -f <local-block-pv10.yaml> 1
```

1 直前の手順で作成された PersistentVolume のファイル名。

2.26.4. 新規 DataVolume への仮想マシンディスクの PersistentVolumeClaim のクローン作成

既存の仮想マシンディスクの PersistentVolumeClaim (PVC) のクローンを新規 DataVolume に作成できます。その後、新規 DataVolume は新規の仮想マシンに使用できます。

注記

Volume が仮想マシンとは別に作成される場合、DataVolume のライフサイクルは仮想マシンから切り離されます。仮想マシンが削除されても、DataVolume もその関連付けられた PVC も削除されません。

前提条件

- 使用する既存の仮想マシンディスクの PVC を判別します。クローン作成の前に、PVC に関連付けられた仮想マシンの電源を切る必要があります。
- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。
- ソース PVC と同じか、またはこれよりも大きい 1 つ以上の利用可能なブロック PersistentVolume (PV)。

手順

1. 関連付けられた PVC の名前および namespace を特定するために、クローン作成に必要な仮想マシンディスクを確認します。
2. 新規 DataVolume の名前、ソース PVC の名前および namespace、利用可能なブロック PV を使用できるようにするために **volumeMode: Block**、および新規 DataVolume のサイズを指定する DataVolume オブジェクトの YAML ファイルを作成します。
例:

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <cloner-datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source-namespace>" 2
      name: "<my-favorite-vm-disk>" 3
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 4
    volumeMode: Block 5
```

1. 新規 DataVolume の名前。
2. ソース PVC が存在する namespace。
3. ソース PVC の名前。
4. 新規 DataVolume のサイズ。十分な領域を割り当てる必要があります。そうでない場合には、クローン操作は失敗します。サイズはソース PVC と同じか、またはそれよりも大きくなければなりません。
5. 宛先がブロック PV であることを指定します。

3. DataVolume を作成して PVC のクローン作成を開始します。

```
$ oc create -f <cloner-datavolume>.yaml
```

注記

DataVolume は仮想マシンが PVC の作成前に起動することを防ぐため、PVC のクローン作成中に新規 DataVolume を参照する仮想マシンを作成できます。

2.26.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	□ TAR	□ TAR

✓ サポートされる操作

□ サポートされない操作

* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

2.27. 仮想マシンのライブマイグレーション

2.27.1. ライブマイグレーションについて

ライブマイグレーションは、仮想ワークロードまたはアクセスに支障を与えることなく、実行中の仮想マシンインスタンスをクラスター内の別のノードに移行するプロセスです。これは、他のノードに移行する仮想マシンインスタンスを選択する場合は手動プロセスで実行でき、仮想マシンインスタンスに **LiveMigrate** エビクションストラテジーがあり、これが実行されるノードがメンテナンス状態の場合には自動プロセスで実行できます。

重要

仮想マシンでは、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) のライブマイグレーションが必要です。

追加リソース:

- 仮想マシンインスタンスの別のノードへの移行
- ノードのメンテナンスモード
- ライブマイグレーションの制限

2.28. ライブマイグレーションの制限およびタイムアウト

ライブマイグレーションの制限およびタイムアウトは、移行プロセスがクラスターに負担をかけないようにするために適用されます。**kubevirt-config** 設定ファイルを編集してこれらの設定を行います。

2.28.1. ライブマイグレーションの制限およびタイムアウトの設定

更新された key:value フィールドを **openshift-cnv** namespace にある **kubevirt-config** 設定ファイルに追加することによってライブマイグレーションの制限およびタイムアウトを設定します。

手順

- **kubevirt-config** 設定ファイルを編集し、必要なライブマイグレーションパラメーターを追加します。以下の例は、デフォルト値を示しています。

```
$ oc edit configmap kubevirt-config -n openshift-cnv
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-config
  namespace: kubevirt
  labels:
    kubevirt.io: ""
data:
  feature-gates: "LiveMigration"
  migrations: |-
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    bandwidthPerMigration: 64Mi
    completionTimeoutPerGiB: 800
    progressTimeout: 150
```

2.28.2. クラスター全体のライブマイグレーションの制限およびタイムアウト

表2.5 マイグレーションパラメーター

パラメーター	説明	デフォルト
parallelMigrationsPerCluster	クラスターで並行して実行されるマイグレーションの数。	5
parallelOutboundMigrationsPerNode	ノードごとのアウトバウンドマイグレーションの最大数。	2

パラメーター	説明	デフォルト
bandwidthPerMigration	各マイグレーションの帯域幅 (MiB/s)。	64Mi
completionTimeoutPerGiB	マイグレーションが (メモリー GiB あたりの秒単位の) この時間内に終了しなかった場合、マイグレーションは取り消されます。たとえば、6GiB メモリーを持つ仮想マシンインスタンスは、4800 秒内にマイグレーションを完了しない場合にタイムアウトします。**Migration Method** が **BlockMigration** の場合、移行するディスクのサイズは計算に含められます。	800
progressTimeout	マイグレーションは、メモリーコピーの進展が (秒単位の) この時間内に見られない場合に取り消されます。	150

2.29. 仮想マシンインスタンスの別のノードへの移行

Web コンソールまたは CLI のいずれかで仮想マシンインスタンスのライブマイグレーションを手動で開始します。

2.29.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションの開始

実行中の仮想マシンインスタンスをクラスター内の別のノードに移行します。

注記

Migrate Virtual Machineアクションはすべてのユーザーに対して表示されますが、仮想マシンの移行を開始できるのは管理者ユーザーのみとなります。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。
2. この画面からマイグレーションを開始できます。 これにより、1 つの画面で複数のマシンに対してアクションを実行することがより容易になります。 または、**Virtual Machine Details**画面から仮想マシンを停止することもできます。 この場合、選択された仮想マシンの総合的な詳細情報を確認できます。

    - Options メニュー  をクリックし、**Migrate Virtual Machine**を選択します。

- 仮想マシン名をクリックし、**Virtual Machine Details**画面を開き、**Actions → Migrate Virtual Machine** をクリックします。

3. **Migrate** をクリックして、仮想マシンを別のノードに移行します。

2.29.2. CLI での仮想マシンインスタンスのライブマイグレーションの開始

クラスターに **VirtualMachineInstanceMigration** オブジェクトを作成し、仮想マシンインスタンスの名前を参照して、実行中の仮想マシンインスタンスのライブマイグレーションを開始します。

手順

1. 移行する仮想マシンインスタンスの **VirtualMachineInstanceMigration** 設定ファイルを作成します。**vmi-migrate.yaml** はこの例になります。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
spec:
  vmiName: vmi-fedora
```

2. クラスターにオブジェクトを作成します。

```
$ oc create -f vmi-migrate.yaml
```

VirtualMachineInstanceMigration オブジェクトは、仮想マシンインスタンスのライブマイグレーションをトリガーします。このオブジェクトは、手動で削除されない場合、仮想マシンインスタンスが実行中である限りクラスターに存在します。

追加リソース:

- 仮想マシンインスタンスのライブマイグレーションのモニター
- 仮想マシンインスタンスのライブマイグレーションの取り消し

2.30. 仮想マシンインスタンスのライブマイグレーションのモニター

Web コンソールまたは CLI のいずれかで仮想マシンインスタンスのライブマイグレーションの進捗をモニターできます。

2.30.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションのモニター

移行期間中、仮想マシンのステータスは **Migrating** になります。このステータスは **Virtual Machines** 一覧に表示されるか、または移行中の仮想マシンの **Virtual Machine Details**画面に表示されます。

手順

- Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。

2.30.2. CLI での仮想マシンインスタンスのライブマイグレーションのモニター

仮想マシンの移行のステータスは、**VirtualMachineInstance** 設定の **Status** コンポーネントに保存されます。

手順

- 移行中の仮想マシンインスタンスで **oc describe** コマンドを使用します。

```
$ oc describe vmi vmi-fedora
```

```
...
Status:
  Conditions:
    Last Probe Time:       <nil>
    Last Transition Time:  <nil>
    Status:                True
    Type:                  LiveMigratable
  Migration Method:  LiveMigration
  Migration State:
    Completed:                true
    End Timestamp:            2018-12-24T06:19:42Z
    Migration UID:            d78c8962-0743-11e9-a540-fa163e0c69f1
    Source Node:              node2.example.com
    Start Timestamp:          2018-12-24T06:19:35Z
    Target Node:              node1.example.com
    Target Node Address:      10.9.0.18:43891
    Target Node Domain Detected:  true
```

2.31. 仮想マシンインスタンスのライブマイグレーションの取り消し

仮想マシンインスタンスを元のノードに残すためにライブマイグレーションを取り消すことができます。

Web コンソールまたは CLI のいずれかでライブマイグレーションを取り消します。

2.31.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションの取り消し

仮想マシンインスタンスのライブマイグレーションは、Options メニュー  を **Workloads → Virtual Machines** 画面の各仮想マシンで使用するか、または **Virtual Machine Details** 画面の **Actions** メニューから取り消すことができます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。

2. この画面から移行を取り消すことができます。 これにより、1つの画面で複数のマシンに対してアクションを実行することがより容易になります。 または、**Virtual Machine Details**画面から仮想マシンを停止することもできます。 この場合、選択された仮想マシンの総合的な詳細情報を確認できます。

- Options メニュー をクリックして、**Cancel Virtual Machine Migration**を選択します。
- 仮想マシン名をクリックして **Virtual Machine Details**画面を開き、**Actions → Cancel Virtual Machine Migration** をクリックします。

3. **Cancel Migration** をクリックして仮想マシンのライブマイグレーションを取り消します。

2.31.2. CLI での仮想マシンインスタンスのライブマイグレーションの取り消し

マイグレーションに関連付けられた **VirtualMachineInstanceMigration** オブジェクトを削除して、仮想マシンインスタンスのライブマイグレーションを取り消します。

手順

- ライブマイグレーションをトリガーした **VirtualMachineInstanceMigration** オブジェクトを削除します。 この例では、**migration-job** が使用されています。

```
$ oc delete vmim migration-job
```

2.32. 仮想マシンのエビクションストラテジーの設定

LiveMigrate エビクションストラテジーは、ノードがメンテナンス状態になるか、ドレイン (解放) される場合に仮想マシンインスタンスが中断されないようにします。このエビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションが別のノードに対して行われます。

2.32.1. LiveMigration エビクションストラテジーでのカスタム仮想マシンの設定

LiveMigration エビクションストラテジーはカスタム仮想マシンでのみ設定する必要があります。共通テンプレートには、デフォルトでこのエビクションストラテジーが設定されています。

手順

1. **evictionStrategy: LiveMigrate** オプションを、仮想マシン設定ファイルの **spec** セクションに追加します。この例では、**oc edit** を使用して **VirtualMachine** 設定ファイルの関連するスニペットを更新します。

```
$ oc edit vm <custom-vm> -n <my-namespace>
```

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: custom-vm
spec:
  terminationGracePeriodSeconds: 30
  evictionStrategy: LiveMigrate
  domain:
    resources:
      requests:
...
```

2. 更新を有効にするために仮想マシンを再起動します。

```
$ virtctl restart <custom-vm> -n <my-namespace>
```

2.33. ノードのメンテナンスモード

2.33.1. ノードのメンテナンスモードについて

ノードがメンテナンス状態になると、ノードにはスケジュール対象外のマークが付けられ、ノードからすべての仮想マシンおよび Pod がドレイン (解放) されます。**LiveMigrate** エビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションは、サービスが失われることなく実行されます。このエビクションストラテジーはデフォルトで共通テンプレートから作成される仮想マシンで設定されますが、カスタム仮想マシンの場合には手動で設定される必要があります。

エビクションストラテジーのない仮想マシンインスタンスは、ノードで削除され、別のノードで再作成されます。

重要

仮想マシンでは、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) のライブマイグレーションが必要です。

追加リソース:

- 仮想マシンのライブマイグレーション
- 仮想マシンのエビクションストラテジーの設定

2.34. ノードのメンテナンスモードへの設定

2.34.1. ノードのメンテナンスモードについて

ノードがメンテナンス状態になると、ノードにはスケジュール対象外のマークが付けられ、ノードからすべての仮想マシンおよび Pod がドレイン (解放) されます。**LiveMigrate** エビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションは、サービスが失われることなく実行されます。このエビクションストラテジーはデフォルトで共通テンプレートから作成される仮想マシンで設定されますが、カスタム仮想マシンの場合には手動で設定される必要があります。

エビクションストラテジーのない仮想マシンインスタンスは、ノードで削除され、別のノードで再作成されます。

重要

仮想マシンでは、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) のライブマイグレーションが必要です。

Web コンソールまたは CLI のいずれかでノードをメンテナンス状態にします。

2.34.2. Web コンソールでのノードのメンテナンスモードへの設定

Options メニュー (**Compute → Nodes** 一覧にある各ノードにある) を使用するか、または **Node Details** 外面の **Actions** コントロールを使用してノードをメンテナンスモードから再開します。

手順

1. Container-native Virtualization コンソールで **Compute → Nodes** をクリックします。
2. この画面からノードをメンテナンスモードに設定できます。 これにより、1 つの画面で複数のノードに対してアクションを実行することがより容易になります。 または、**Node Details** 画面からノードをメンテナンスモードに設定することもできます。 この場合、選択されたノードの総合的な詳細情報を確認できます。

   - Options メニュー をクリックし、**Start Maintenance** を選択します。
   - ノード名をクリックし、**Node Details** 画面を開いて **Actions → Stat Maintenance** をクリックします。
3. 確認ウィンドウで **Start Maintenance** をクリックします。

ノードは **liveMigration** エビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションを行い、このノードはスケジュール対象外となります。このノードの他のすべての Pod および仮想マシンは削除され、別のノードで再作成されます。

2.34.3. CLI でのノードのメンテナンスモードへの設定

ノード名、およびこれをメンテナンスモードに設定する理由を参照する **NodeMaintenance** カスタムリソース (CR) オブジェクトを作成し、ノードをメンテナンスモードに設定します。

手順

1. ノードメンテナンス CR 設定を作成します。この例では、**node02-maintenance.yaml** という CR を使用します。

```
apiVersion: kubevirt.io/v1alpha1
kind: NodeMaintenance
metadata:
  name: node02-maintenance
spec:
  nodeName: node02
  reason: "Replacing node02"
```

2. **NodeMaintenance** オブジェクトをクラスターに作成します。

```
$ oc apply -f <node02-maintenance.yaml>
```

ノードは **liveMigration** エビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションを実行し、これがスケジュール対象外になるようにノードにテイントを設定します。このノードの他のすべての Pod および仮想マシンは削除され、別のノードで再作成されます。

追加リソース:

- メンテナンスモードからのノードの再開

2.35. メンテナンスモードからのノードの再開

ノードを再開することにより、ノードをメンテナンスモードから切り替え、再度スケジュール可能な状態にします。

Web コンソールまたは CLI のいずれかでノードをメンテナンスモードから再開します。

2.35.1. Web コンソールでのメンテナンスモードからのノードの再開

Options メニュー  (**Compute → Nodes** 一覧にある各ノードにある) を使用するか、または **Node Details** 外面の **Actions** コントロールを使用してノードをメンテナンスモードから再開します。

手順

1. Container-native Virtualization コンソールで **Compute → Nodes** をクリックします。
2. この画面からノードを再開できます。 これにより、1 つの画面で複数のノードに対してアクションを実行することがより容易になります。 または、**Node Details** 画面からノードを再開することもできます。 この場合、選択されたノードの総合的な詳細情報を確認できます。

   - Options メニュー  をクリックし、**Stop Maintenance**を選択します。
   - ノード名をクリックし、**Node Details** 画面を開いて **Actions → Stop Maintenance** をクリックします。

3. 確認ウィンドウで **Stop Maintenance** をクリックします。

ノードはスケジュール可能な状態になりますが、メンテナンス前にノード上で実行されていた仮想マシンインスタンスはこのノードに自動的に戻されません。

2.35.2. CLI でのメンテナンスモードからのノードの再開

ノードの **NodeMaintenance** オブジェクトを削除することによって、メンテナンスモードからノードを再開し、これを再度スケジュール可能な状態にします。

手順

1. **NodeMaintenance** オブジェクトを見つけます。

```
$ oc get nodemaintenance
```

2. オプション: **NodeMaintenance** オブジェクトを検査し、これが正しいノードに関連付けられていることを確認します。

```
$ oc describe nodemaintenance <node02-maintenance>
```

```
Name:         node02-maintenance
Namespace:
Labels:
Annotations:
API Version:  kubevirt.io/v1alpha1
Kind:         NodeMaintenance
...
```

```
Spec:
  Node Name:  node02
  Reason:     Replacing node02
```

3. **NodeMaintenance** オブジェクトを削除します。

```
$ oc delete nodemaintenance <node02-maintenance>
```

2.36. TLS 証明書の手動更新

Container-native Virtualization コンポーネントの TLS 証明書はインストール時に作成され、1 年間有効になります。これらの証明書は期限切れになる前に手動で更新する必要があります。

2.36.1. TLS 証明書の更新

Container-native Virtualization の TLS 証明書を更新するには、**rotate-certs** スクリプトをダウンロードし、実行します。このスクリプトは、GitHub の **kubevirt/hyperconverged-cluster-operator** リポジトリーから入手できます。

重要

証明書を更新すると、以下の操作の影響が確認されます。

- 移行がキャンセルされる。
- イメージのアップロードがキャンセルされる。
- VNC およびコンソールの接続が閉じられる。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていることを確認します。このスクリプトは、アクティブセッションをクラスターに対して使用し、**openshift-cnv** namespace の証明書を更新します。

手順

1. GitHub から **rotate-certs.sh** スクリプトをダウンロードします。

```
$ curl -O https://raw.githubusercontent.com/kubevirt/hyperconverged-cluster-
operator/master/tools/rotate-certs.sh
```

2. スクリプトが実行可能であることを確認します。

```
$ chmod +x rotate-certs.sh
```

3. スクリプトを実行します。

```
$ ./rotate-certs.sh -n openshift-cnv
```

TLS 証明書は更新され、1 年間有効になります。

2.37. VIRTIO ドライバーの既存の WINDOWS 仮想マシンへのインストール

2.37.1. VirtIO ドライバーについて

VirtIO ドライバーは、Microsoft Windows 仮想マシンが Container-native Virtualization で実行されるために必要な準仮想化デバイスドライバーです。サポートされるドライバーは、Red Hat Container Catalog の **container-native-virtualization/virtio-win** コンテナーディスクで利用できます。

container-native-virtualization/virtio-win コンテナーディスクは、ドライバーのインストールを有効にするために SATA CD ドライブとして仮想マシンに割り当てられる必要があります。仮想マシン上での Windows のインストール時に VirtIO ドライバーをインストールすることも、既存の Windows インストールに追加することもできます。

ドライバーのインストール後に、**container-native-virtualization/virtio-win** コンテナーディスクは仮想マシンから削除できます。

「Installing Virtio drivers on a new Windows virtual machine 」も参照してください。

2.37.2. Microsoft Windows 仮想マシンのサポートされる VirtIO ドライバー

表2.6 サポートされるドライバー

ドライバー名	ハードウェア ID	説明
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	ブロックドライバー。**Other devices** グループの **SCSI Controller** として表示される場合があります。
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	エントロピーソースドライバー。**Other devices** グループの **PCI Device** として表示される場合があります。
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	ネットワークドライバー。**Other devices** グループの **Ethernet Controller** として表示される場合があります。VirtIO NIC が設定されている場合にのみ利用できます。

2.37.3. VirtIO ドライバーコンテナーディスクの仮想マシンへの追加

Container-native Virtualization は、Red Hat Container Catalog で利用できる Microsoft Windows の VirtIO ドライバーをコンテナーディスクとして配布します。これらのドライバーを Windows 仮想マシンにインストールするには、仮想マシン設定ファイルで **container-native-virtualization/virtio-win** コンテナーディスクを SATA CD ドライブとして仮想マシンに割り当てます。

前提条件

- **container-native-virtualization/virtio-win** コンテナーディスクを Red Hat Container Catalog からダウンロードします。コンテナーディスクがクラスターにない場合は Red Hat レジストリーからダウンロードされるため、これは必須ではありません。

手順

1. **container-native-virtualization/virtio-win** コンテナーディスクを **cdrom** ディスクとして Windows 仮想マシン設定ファイルに追加します。コンテナーディスクは、クラスターにない場合はレジストリーからダウンロードされます。

```
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2 (1)
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk
```

(1) Container-native Virtualization は、**VirtualMachine** 設定ファイルに定義される順序で仮想マシンディスクを起動します。**container-native-virtualization/virtio-win** コンテナーディスクの前に仮想マシンの他のディスクを定義するか、またはオプションの **bootOrder** パラメーターを使用して仮想マシンが正しいディスクから起動するようにできます。ディスクに **bootOrder** を指定する場合、これは設定のすべてのディスクに指定される必要があります。

2. ディスクは、仮想マシンが起動すると利用可能になります。

- コンテナーディスクを実行中の仮想マシンに追加する場合、変更を有効にするために CLI で **oc apply -f <vm.yaml>** を使用するか、または仮想マシンを再起動します。
- 仮想マシンが実行されていない場合、**virtctl start <vm>** を使用します。

仮想マシンが起動したら、VirtIO ドライバーを割り当てられた SATA CD ドライブからインストールできます。

2.37.4. VirtIO ドライバーの既存 Windows 仮想マシンへのインストール

VirtIO ドライバーを、割り当てられた SATA CD ドライブから既存の Windows 仮想マシンにインストールします。

注記

この手順では、ドライバーを Windows に追加するための汎用的なアプローチを使用しています。このプロセスは Windows のバージョンごとに若干異なる可能性があります。特定のインストール手順については、お使いの Windows バージョンについてのインストールドキュメントを参照してください。

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。
2. Windows ユーザーセッションにログインします。
3. **Device Manager** を開き、**Other devices** を拡張して、**Unknown device** を一覧表示します。

a. **Device Properties** を開いて、不明なデバイスを特定します。デバイスを右クリックし、**Properties** を選択します。

b. **Details** タブをクリックし、**Property** リストで **Hardware Ids** を選択します。

c. **Hardware Ids** の **Value** をサポートされる VirtIO ドライバーと比較します。

4. デバイスを右クリックし、**Update Driver Software** を選択します。

5. **Browse my computer for driver software**をクリックし、VirtIO ドライバーが置かれている割り当て済みの SATA CD ドライブの場所に移動します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。

6. **Next** をクリックしてドライバーをインストールします。

7. 必要なすべての VirtIO ドライバーに対してこのプロセスを繰り返します。

8. ドライバーのインストール後に、**Close** をクリックしてウィンドウを閉じます。

9. 仮想マシンを再起動してドライバーのインストールを完了します。

2.37.5. 仮想マシンからの VirtIO コンテナーディスクの削除

必要なすべての VirtIO ドライバーを仮想マシンにインストールした後は、**container-native-virtualization/virtio-win** コンテナーディスクを仮想マシンに割り当てる必要はなくなります。**container-native-virtualization/virtio-win** コンテナーディスクを仮想マシン設定ファイルから削除します。

手順

1. 設定ファイルを編集し、**disk** および **volume** を削除します。

```
$ oc edit vm <vm-name>
```

```
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk
```

2. 変更を有効にするために仮想マシンを再起動します。

2.38. VIRTIO ドライバーの新規 WINDOWS 仮想マシンへのインストール

前提条件

- 仮想マシンからアクセスできる Windows インストールメディア (ISO のデータボリュームへのインポート および仮想マシンへの割り当てを実行)。

2.38.1. VirtIO ドライバーについて

VirtIO ドライバーは、Microsoft Windows 仮想マシンが Container-native Virtualization で実行されるために必要な準仮想化デバイスドライバーです。サポートされるドライバーは、Red Hat Container Catalog の **container-native-virtualization/virtio-win** コンテナーディスクで利用できます。

container-native-virtualization/virtio-win コンテナーディスクは、ドライバーのインストールを有効にするために SATA CD ドライブとして仮想マシンに割り当てられる必要があります。仮想マシン上での Windows のインストール時に VirtIO ドライバーをインストールすることも、既存の Windows インストールに追加することもできます。

ドライバーのインストール後に、**container-native-virtualization/virtio-win** コンテナーディスクは仮想マシンから削除できます。

「VirtIO ドライバーの既存の Windows 仮想マシンへのインストール」も参照してください。

2.38.2. Microsoft Windows 仮想マシンのサポートされる VirtIO ドライバー

表2.7 サポートされるドライバー

ドライバー名	ハードウェア ID	説明
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	ブロックドライバー。**Other devices** グループの **SCSI Controller** として表示される場合があります。
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	エントロピーソースドライバー。**Other devices** グループの **PCI Device** として表示される場合があります。
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	ネットワークドライバー。**Other devices** グループの **Ethernet Controller** として表示される場合があります。VirtIO NIC が設定されている場合にのみ利用できます。

2.38.3. VirtIO ドライバーコンテナーディスクの仮想マシンへの追加

Container-native Virtualization は、Red Hat Container Catalog で利用できる Microsoft Windows の VirtIO ドライバーをコンテナーディスクとして配布します。これらのドライバーを Windows 仮想マシンにインストールするには、仮想マシン設定ファイルで **container-native-virtualization/virtio-win** コンテナーディスクを SATA CD ドライブとして仮想マシンに割り当てます。

前提条件

- **container-native-virtualization/virtio-win** コンテナーディスクを Red Hat Container Catalog からダウンロードします。コンテナーディスクがクラスターにない場合は Red Hat レジストリーからダウンロードされるため、これは必須ではありません。

手順

1. **container-native-virtualization/virtio-win** コンテナーディスクを **cdrom** ディスクとして Windows 仮想マシン設定ファイルに追加します。コンテナーディスクは、クラスターにない場合はレジストリーからダウンロードされます。

```
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2 (1)
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk
```

(1) Container-native Virtualization は、**VirtualMachine** 設定ファイルに定義される順序で仮想マシンディスクを起動します。**container-native-virtualization/virtio-win** コンテナーディスクの前に仮想マシンの他のディスクを定義するか、またはオプションの **bootOrder** パラメーターを使用して仮想マシンが正しいディスクから起動するようにできます。ディスクに **bootOrder** を指定する場合、これは設定のすべてのディスクに指定される必要があります。

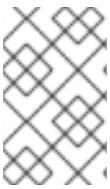
2. ディスクは、仮想マシンが起動すると利用可能になります。

   - コンテナーディスクを実行中の仮想マシンに追加する場合、変更を有効にするために CLI で **oc apply -f <vm.yaml>** を使用するか、または仮想マシンを再起動します。
   - 仮想マシンが実行されていない場合、**virtctl start <vm>** を使用します。

仮想マシンが起動したら、VirtIO ドライバーを割り当てられた SATA CD ドライブからインストールできます。

2.38.4. Windows インストール時の VirtIO ドライバーのインストール

Windows のインストール時に割り当てられた SATA CD ドライバーから VirtIO ドライバーをインストールします。



注記

この手順では、Windows インストールの汎用的なアプローチを使用しますが、インストール方法は Windows のバージョンごとに異なる可能性があります。インストールする Windows のバージョンについてのドキュメントを参照してください。

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。

2. Windows インストールプロセスを開始します。
3. **Advanced** インストールを選択します。
4. ストレージの宛先は、ドライバーがロードされるまで認識されません。**Load driver** をクリックします。
5. ドライバーは SATA CD ドライブとして割り当てられます。**OK** をクリックし、CD ドライバーでロードするストレージドライバーを参照します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。
6. 必要なすべてのドライバーについて直前の 2 つの手順を繰り返します。
7. Windows インストールを完了します。

2.38.5. 仮想マシンからの VirtIO コンテナーディスクの削除

必要なすべての VirtIO ドライバーを仮想マシンにインストールした後は、**container-native-virtualization/virtio-win** コンテナーディスクを仮想マシンに割り当てる必要はなくなります。**container-native-virtualization/virtio-win** コンテナーディスクを仮想マシン設定ファイルから削除します。

手順

1. 設定ファイルを編集し、**disk** および **volume** を削除します。

```
$ oc edit vm <vm-name>
```

```
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk
```

2. 変更を有効にするために仮想マシンを再起動します。

2.39. 仮想マシンログの表示

2.39.1. 仮想マシンのログについて

ログは、OpenShift Container Platform ビルド、デプロイメントおよび Pod について収集されます。Container-native Virtualization では、仮想マシンのログは Web コンソールまたは CLI のいずれかで仮想マシンランチャー Pod から取得されます。

-f オプションは、リアルタイムでログ出力をフォローします。 これは進捗のモニターおよびエラーの確認に役立ちます。

ランチャー Pod の起動が失敗する場合、**--previous** オプションを使用して最後の試行時のログを確認します。

警告

ErrImagePull および **ImagePullBackOff** エラーは、誤ったデプロイメント設定または参照されるイメージに関する問題によって引き起こされる可能性があります。

2.39.2. CLI での仮想マシンログの表示

仮想マシンランチャー Pod から仮想マシンログを取得します。

手順

- 以下のコマンドを実行します。

```
$ oc logs <virt-launcher-name>
```

2.39.3. Web コンソールでの仮想マシンログの表示

関連付けられた仮想マシンランチャー Pod から仮想マシンログを取得します。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines**をクリックします。
2. 仮想マシンをクリックし、**Virtual Machine Details**パネルを開きます。
3. **Overview** タブで、**POD** セクションの **virt-launcher-<name>** Pod をクリックします。
4. **Logs** をクリックします。

2.40. OPENSHIFT CONTAINER PLATFORM DASHBOARD を使用したクラスター情報の取得

OpenShift Container Platform Web コンソールから **Home > Dashboards > Overview**をクリックしてクラスターについてのハイレベルな情報をキャプチャーする OpenShift Container Platform ダッシュボードにアクセスします。

OpenShift Container Platform ダッシュボードは、個別のダッシュボード **カード** でキャプチャーされるさまざまなクラスター情報を提供します。

2.40.1. OpenShift Container Platform ダッシュボードページについて

OpenShift Container Platform ダッシュボードは以下のカードで構成されます。

- **Details** は、クラスターの詳細情報の概要を表示します。

ステータスには、**ok**、 **error**、**warning**、**in progress**、および **unknown** が含まれます。リソースでは、カスタムのステータス名を追加できます。

- クラスター
- プロバイダー
- バージョン

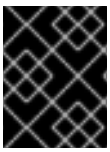
- **Cluster Inventory** は、リソースの数および関連付けられたステータスの詳細を表示します。これは、問題の解決に介入が必要な場合に役立ちます。以下についての情報が含まれます。
  - ノード数
  - Pod 数
  - 永続ストレージボリューム要求 (Persistent Storage Volume Claim)
  - 仮想マシン (Container-native Virtualization がインストールされている場合に利用可能)
  - クラスター内のベアメタルホスト。これらはステータス別に一覧表示されます (**metal3** 環境でのみ利用可能)。
- **Cluster Health** では、関連するアラートおよび説明を含む、クラスターの現在の健全性についてのサマリーを表示します。Container-native Virtualization がインストールされている場合は、Container-native Virtualization の全体的な健全性も診断されます。複数のサブシステムが存在する場合は、**See All** をクリックして、各サブシステムのステータスを表示します。
- **Cluster Capacity** チャートは、管理者が追加リソースがクラスターで必要になるタイミングを把握するのに役立ちます。このチャートには、現在の消費量を表示する内側の円が含まれ、外側の円には、以下の情報を含む、リソースに対して設定されたしきい値が表示されます。
  - CPU 時間
  - メモリー割り当て
  - 消費されるストレージ
  - 消費されるネットワークリソース
- **Cluster Utilization** は指定された期間における各種リソースの容量を表示します。これは、管理者がリソースの高い消費量の規模および頻度を理解するのに役立ちます。
- **Events** は、Pod の作成または別のホストへの仮想マシンの移行などのクラスター内の最近のアクティビティーに関連したメッセージを一覧表示します。
- **Top Consumers** は、管理者がクラスターリソースが消費される状況を把握するのに役立ちます。リソースをクリックし、指定されたクラスターリソース (CPU、メモリー、またはストレージ) の最大量を消費する Pod およびノードを一覧表示する詳細ページに切り替えます。

2.41. OPENSHIFT CONTAINER PLATFORM クラスターモニタリング、ロギング、および TELEMETRY

OpenShift Container Platform は、クラスターレベルでモニターするための各種のリソースを提供します。

2.41.1. OpenShift Container Platform クラスターモニタリングについて

OpenShift Container Platform には、Prometheus オープンソースプロジェクトおよびその幅広いエコシステムをベースとする事前に設定され、事前にインストールされた自己更新型のモニタリングスタックが同梱されます。これはクラスターのモニタリング機能を提供し、クラスター管理者に問題の発生を即時に通知するアラートのセットと Grafana ダッシュボードのセットを提供します。クラスターモニタリングスタックは、OpenShift Container Platform クラスターのモニタリング用のみにサポートされています。



重要

今後の OpenShift Container Platform の更新との互換性を確保するために、指定されたモニタリングスタックのオプションのみを設定することがサポートされます。

2.41.2. クラスターロギングコンポーネント

クラスターロギングコンポーネントは Elasticsearch、Fluentd、Kibana (EFK) に基づいています。コレクターの Fluentd は、OpenShift Container Platform クラスターの各ノードにデプロイされます。これはすべてのノードおよびコンテナーのログを収集し、それらを Elasticsearch (ES) に書き込みます。Kibana は、ユーザーおよび管理者が集計されたデータを使って高度な視覚化およびダッシュボードを作成できる中央の Web UI です。

現時点で、5 種類のクラスターロギングコンポーネントがあります。

- logStore: これはログが保存される場所です。現在の実装は Elasticsearch です。
- collection: これは、ノードからログを収集し、それらをフォーマットし、logStore に保存するコンポーネントです。現在の実装は Fluentd です。
- visualization: これは、ログ、グラフ、チャートなどを表示するために使用される UI コンポーネントです。現在の実装は Kibana です。
- curation: これは期間に基づいてログをトリミングするコンポーネントです。現在の実装は Curator です。
- event routing: これは、OpenShift Container Platform イベントをクラスターロギングに転送するコンポーネントです。現在の実装はイベントルーターです。

クラスターロギングについての詳細は、「OpenShift Container Platform クラスターロギング」のドキュメントを参照してください。

2.41.3. Telemetry について

Telemetry は厳選されたクラスターモニタリングメトリクスのサブセットを Red Hat に送信します。これらのメトリクスは継続的に送信され、以下について記述します。

- OpenShift Container Platform クラスターのサイズ
- OpenShift Container Platform コンポーネントの健全性およびステータス
- 実行されるアップグレードの正常性およびステータス
- OpenShift Container Platform のコンポーネントおよび機能についての使用情報 (一部の制限された情報)

- クラスターモニタリングコンポーネントによってレポートされるアラートについてのサマリー情報

Red Hat では、リアルタイムでクラスターの健全性をモニターし、お客様に影響を与える問題に随時対応するためにこのデータの継続的なストリームを使用します。またこれにより、Red Hat がサービスへの影響を最小限に抑えつつつアップグレードエクスペリエンスの継続的な改善に向けた OpenShift Container Platform のアップグレードの展開を可能にします。

このデバッグ情報は、サポートケースでレポートされるデータへのアクセスと同じ制限が適用された状態で Red Hat サポートおよびエンジニアリングチームが利用できます。接続クラスターのすべての情報は、OpenShift Container Platform をより使用しやすく、より直感的に使用できるようにするために Red Hat によって使用されます。この情報のいずれもサードパーティーと共有されることはありません。

2.41.3.1. Telemetry で収集される情報

Telemetry によって収集される主な情報には、以下が含まれます。

- クラスターごとに利用可能な更新の数
- 更新に使用されるチャネルおよびイメージリポジトリー
- 更新中に発生するエラーの数
- 実行中の更新の進捗情報
- クラスターごとのマシン数
- CPU コアの数およびマシンの RAM のサイズ
- etcd クラスターのメンバー数、および現在 etcd クラスターに保存されているオブジェクトの数
- マシンタイプ (インフラまたはマスター) ごとに使用される CPU コアおよび RAM の数
- クラスターごとに使用される CPU コアおよび RAM の数
- クラスターごとの OpenShift Container Platform フレームワークコンポーネントの使用
- OpenShift Container Platform クラスターのバージョン
- クラスターにインストールされている OpenShift Container Platform フレームワークコンポーネントの健全性、状態、およびステータス。 たとえば、クラスターバージョン Operator、クラスターモニタリング、イメージレジストリー、およびロギング用の Elasticsearch がこれらのコンポーネントに含まれます。
- インストール時に生成される一意でランダムな識別子
- Amazon Web Services などの OpenShift Container Platform がデプロイされているプラットフォームの名前

Telemetry は、ユーザー名、パスワード、またはユーザーリソースの名前またはアドレスなどの識別情報を収集しません。

2.41.4. CLI のトラブルシューティングおよびデバッグコマンド

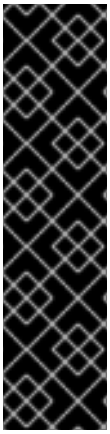
oc クライアントのトラブルシューティングおよびデバッグコマンドの一覧については、「OpenShift Container Platform CLI ツール」のドキュメントを参照してください。

2.42. RED HAT サポート向けの CONTAINER-NATIVE VIRTUALIZATION データの収集

サポートケースを作成する際、ご使用のクラスターについてのデバッグ情報を Red Hat サポートに提供していただくと Red Hat のサポートに役立ちます。

must-gather ツールを使用すると、仮想マシンおよび Container-native Virtualization に関連する他のデータを含む、 OpenShift Container Platform クラスターについての診断情報を収集できます。

迅速なサポートを得るには、OpenShift Container Platform と container-native virtualization の両方についての診断情報を提供してください。



重要

Container-native Virtualization 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。

2.42.1. must-gather ツールについて

oc adm must-gather CLI コマンドは、以下のような問題のデバッグに必要となる可能性のあるクラスターからの情報を収集します。

- リソース定義
- 監査ログ
- サービスログ

--image 引数を指定してコマンドを実行する際にイメージを指定できます。イメージを指定する際、ツールはその機能または製品に関連するデータを収集します。

oc adm must-gatherを実行すると、新しい Pod がクラスターに作成されます。データは Pod で収集され、**must-gather.local**で始まる新規ディレクトリーに保存されます。このディレクトリーは、現行の作業ディレクトリーに作成されます。

2.42.2. Container-native Virtualization データの収集について

oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、Container-native Virtualization に関連する機能およびオブジェクトが含まれます。

- Hyperconverged Cluster Operator namespace（および子オブジェクト）
- Container-native Virtualization リソースに属するすべての namespace (およびそれらの子オブジェクト)

- すべての Container-native Virtualization カスタムリソース定義 (CRD)
- 仮想マシンを含むすべての namespace
- すべての仮想マシン定義

must-gatherを使用して Container-native Virtualization データを収集するには、Container-Native Virtualization イメージ **--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-gather-rhel8** を指定する必要があります。

2.42.3. 特定の機能に関するデータ収集

oc adm must-gather CLI コマンドを **--image** または **--image-stream** 引数と共に使用して、特定に機能についてのデバッグ情報を収集できます。**must-gather** ツールは複数のイメージをサポートするため、単一のコマンドを実行して複数の機能についてのデータを収集できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。
- OpenShift Container Platform CLI（**oc**）がインストールされていること。

手順

1. **must-gather** データを保存するディレクトリーに移動します。
2. **oc adm must-gather** コマンドを 1 つまたは複数の **--image** または **--image-stream** 引数と共に実行します。たとえば、以下のコマンドは、デフォルトのクラスターデータと Container-native Virtualization に固有の情報の両方を収集します。

```
$ oc adm must-gather \
 --image-stream=openshift/must-gather \ 1
 --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8 2
```

1. デフォルトの OpenShift Container Platform must-gather イメージ
2. Container-native Virtualization must-gather イメージ

3. 作業ディレクトリーに作成された **must-gather** ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

1. **must-gather-local.5421342344627712289/** を実際のディレクトリー名に置き換えてください。

4. 圧縮ファイルを Red Hat カスタマーポータル 上のサポートケースに添付します。

第3章 CONTAINER-NATIVE VIRTUALIZATION 2.1 リリースノート

3.1. CONTAINER-NATIVE VIRTUALIZATION 2.1 リリースノート

3.1.1. Container-native Virtualization について

3.1.1.1. Container-native Virtualization の機能

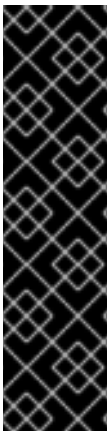
Container-native Virtualization は OpenShift Container Platform のアドオンであり、仮想マシンのワークロードを実行し、このワークロードをコンテナーのワークロードと共に管理することを可能にします。

Container-native Virtualization は、Kubernetes カスタムリソースを使って新規オブジェクトを OpenShift Container Platform クラスターに追加し、仮想化タスクを有効にします。これらのタスクには、以下が含まれます。

- Linux および Windows 仮想マシンの作成と管理
- 各種コンソールおよび CLI ツールの使用による仮想マシンへの接続
- 既存の仮想マシンのインポートおよびクローン作成
- ネットワークインターフェースコントローラーおよび仮想マシンに割り当てられたストレージディスクの管理
- 仮想マシンのノード間でのライブマイグレーション

機能強化された Web コンソールは、これらの仮想化されたリソースを OpenShift Container Platform クラスターコンテナーおよびインフラストラクチャーと共に管理するためのグラフィカルポータルを提供します。

3.1.1.2. Container-native Virtualization のサポート



重要

Container-native Virtualization 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。

3.1.2. 新機能および変更された機能

3.1.2.1. Web コンソールの強化

- OpenShift Container Platform ダッシュボードは、クラスターについてのハイレベル情報を

キャプチャーします。OpenShift Container Platform Web コンソールから、**Home → Dashboards → Overview** をクリックしてダッシュボードにアクセスします。Web コンソールのプロジェクトの概要には、仮想マシンが一覧表示されていないことに注意してください。仮想マシンは **Cluster Inventory** ダッシュボードカードに一覧表示されるようになります。

3.1.2.2. その他の改善点

- Container-native Virtualization のインストール後に、MAC プールマネージャーが自動的に起動します。MAC アドレスを指定せずにセカンダリー NIC を定義すると、MAC プールマネージャーは一意の MAC アドレスを NIC に割り当てます。

注記

特定の MAC アドレスでセカンダリー NIC を定義する場合、MAC アドレスがクラスター内の別の NIC と競合する可能性があります。

3.1.3. 解決済みの問題

- 以前のバージョンでは、Web コンソールを使用して、既存の仮想マシンと同じ名前を持つ仮想マシンテンプレートを作成すると、操作に失敗していました。これにより、**Name is already used by another virtual machine** というメッセージが出されました。この問題は Container-native Virtualization 2.1 で修正されています。(BZ#1717802)

- 以前は、**bridge** モードで仮想マシンを作成し、**cloud-init** ディスクを使用する場合、仮想マシンは再起動後にそのネットワーク接続を失いました。この問題は Container-native Virtualization 2.1 で修正されています。(BZ#1708680)。

3.1.4. 既知の問題

- 仮想マシンの **masquerade** バインディングメソッドは、RHEL 7 コンピュートノードを含むクラスターでは使用できません。(BZ#1741626)

- Container-native Virtualization のインストール時に **KubeVirt HyperConverged Cluster Operator Deployment** カスタムリソースを作成する場合、YAML ファイルは間違った値で表示されます。ファイルは以下の例のようになります。

```
apiVersion: hco.kubevirt.io/v1alpha1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  BareMetalPlatform: 'false' 1
```

1 **'false'** という語の周りにある単一引用符が適切ではありません。このファイルを編集し、**Create** をクリックする前に行に **BareMetalPlatform: false** と表示されるようにします。引用符が削除されないと、デプロイメントは成功しません。(BZ#1767167)

- Web コンソールの **Disks** タブでディスクを仮想マシンに追加する場合、**kubevirt-storage-class-default** ConfigMap に volumeMode が設定されたかどうかに関わらず、追加されたディスクに **Filesystem** volumeMode が常にあります。(BZ#1753688)

- 移行後、仮想マシンには新規の IP アドレスが割り当てられます。ただし、コマンドの **oc get vmi** および **oc describe vmi** は古くなった IP アドレスを含む出力を依然として出力します。(BZ#1686208)
  - 回避策として、以下のコマンドを実行して正しい IP アドレスを表示します。

```
$ oc get pod -o wide
```

- 仮想マシンウィザードは、管理者権限を持たないユーザーの場合は読み込まれません。この問題は、ユーザーがネットワーク割り当て定義を読み込む権限がないために発生します。(BZ#1743985)
  - 回避策として、ネットワークの割り当て定義を読み込むパーミッションを持つユーザーを指定します。
    1. 以下の例を使用して **ClusterRole** および **ClusterRoleBinding** オブジェクトを YAML 設定ファイルに定義します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: cni-resources
rules:
- apiGroups: ["k8s.cni.cncf.io"]
 resources: ["*"]
 verbs: ["*"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: <role-binding-name>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cni-resources
subjects:
- kind: User
  name: <user to grant the role to>
  namespace: <namespace of the user>
```

    2. **cluster-admin** ユーザーとして以下のコマンドを実行し、定義した **ClusterRole** および **ClusterRoleBinding** オブジェクトを作成します。

```
$ oc create -f <filename>.yaml
```

- **Virtual Machines Console** タブに移動する際に、コンテンツが表示されないことがあります。回避策として、シリアルコンソールを使用します。(BZ#1753606)

- ブラウザーからの Container-native Virtualization Operator のすべてのインスタンスの一覧表示を試行すると、404 (page not found) エラーが出されます。(BZ#1757526)
  - 回避策として、以下のコマンドを実行します。

```
$ oc get pods -n openshift-cnv | grep operator
```

- 一部のリソースは、Container-native Virtualization の削除時に不適切に保持される状態が生じます。Container-native Virtualization を再インストールするためにこれらのリソースを手動で削除する必要があります。(BZ#1712429), (BZ#1757705)
  - 回避策として、「Removing leftover resources from container-native virtualization 2.1 uninstallation」の手順に従ってください。
- 仮想マシンが Guaranteed CPU を使用する場合、ラベル **cpumanager=true** がノードに自動的に設定されないためにスケジュールされません。回避策として、**CPUManager** エントリーを **kubevirt-config** ConfigMap から削除します。次に、ノードに **cpumanager=true** のラベルを手動で付けてからクラスター上で Guaranteed CPU を持つ仮想マシンを実行します (BZ#1718944)。
- ノードの CPU モデルが異なるとライブマイグレーションに失敗します。ノードに同じ物理 CPU モデルがある場合でも、マイクロコードの更新によって導入される違いにより同じ状況が生じます。これは、デフォルト設定が、ライブマイグレーションと互換性のないホスト CPU パススルー動作をトリガーするためです。(BZ#1760028)
  - 回避策として、以下の例のように **kubevirt-config** ConfigMap にデフォルトの CPU モデルを設定します。

注記

ライブマイグレーションをサポートする仮想マシンを起動する前に、この変更を行う必要があります。

1. 以下のコマンドを実行して、編集するために **kubevirt-config** ConfigMap を開きます。

```
$ oc edit configmap kubevirt-config -n openshift-cnv
```

2. ConfigMap を編集します。

```
kind: ConfigMap
metadata:
  name: kubevirt-config
data:
  default-cpu-model: "<cpu-model>" 1
```

1 **<cpu-model>** を実際の CPU モデルの値に置き換えます。すべてのノードに **oc describe node <node>** を実行し、**cpu-model-<name>** ラベルを確認してこの値を判別できます。すべてのノードに存在する CPU モデルを選択します。

- Container-native Virtualization のアップグレードプロセスは、Operator Lifecycle Manager (OLM) の中断により失敗する可能性があります。この問題は、Container-native Virtualization Operator の状態を追跡するために宣言型 API を使用することに関連する制限によって生じます。インストール時に自動更新を有効にすることにより、この問題が発生するリスクが低くなります。(BZ#1759612)
- Container-native Virtualization は、**oc adm drain** または **kubectl drain** のいずれかを実行してトリガーされるノードのドレイン (解放) を確実に特定することができません。これらのコマンドは、Container-native Virtualization がデプロイされているクラスターのノードでは実行しないでください。ノードは、それらの上部で実行されている仮想マシンがある場合にドレイン (解放) を実行しない可能性があります。現時点の解決策として、ノードをメンテナンス状態にする方法があります(BZ#1707427)

- **virtctl image-upload** を実行して **qcow2** 形式で大規模な仮想マシンディスクをアップロードすると、アップロードが通常に処理され、完了する場合でも、データ送信後に EOF (end-of-file) エラーが報告される場合があります。(BZ#1754920)
  以下のコマンドを実行して、指定された Pod でアップロードのステータスを確認します。

```
$ oc describe pvc <pvc-name> | grep cdi.kubevirt.io/storage.pod.phase
```