OpenShift Container Platform 3.11

Container-native Virtualization ユーザーガイド

Container-native Virtualization ユーザーガイド

Last Updated: 2021-04-15

OpenShift Container Platform 3.11 Container-native Virtualization ユーザーガイド

Container-native Virtualization ユーザーガイド

法律上の通知

概要

CNV の使用方法

目次

第1章 CONTAINER-NATIVE VIRTUALIZATION の使用

1.1. 製品概要

1.1.1. Container-native Virtualization の導入

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。仮想マシンは、Containerized Data Importer (CDI) コントローラーを使用してインポートされるディスクイメージから作成することも、OpenShift Container Platform 内でゼロから作成することもできます。

Container-native Virtualization は 2 つの新たなオブジェクトを OpenShift Container Platform に導入します。

- **Virtual Machine** (仮想マシン): OpenShift Container Platform の仮想マシンです。
- **Virtual Machine Instance** (仮想マシンインスタンス): 実行される仮想マシンのインスタンスです。

Container-native Virtualization アドオンを使用すると、仮想マシンは Pod で実行され、仮想マシンに標準的な Pod と同じネットワークおよびストレージ機能を持たせることができます。

既存の仮想マシンディスクは永続ボリューム (PV) にインポートされます。この永続ボリューム (PV) は、Persistent Volume Claim (永続ボリューム要求、PVC) を使用して Container-native Virtualization 仮想マシンからアクセスできるようになります。OpenShift Container Platform では、仮想マシンオブジェクトは、PV に保存される永続データに影響を与えることなく、変更したり、置き換えたりすることができます。

重要

現時点で Container-native Virtualization はテクノロジープレビュー機能です。Container-native Virtualization についての Red Hat サポートの詳細は、「Container-native Virtualization - Technology Preview Support Policy」を参照してください。

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。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についての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

1.2. WEB コンソールの操作

1.2.1. 仮想マシンの管理

1.2.1.1. インタラクティブなウィザードを使用した仮想マシンの作成

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** を変更することはできますが、これを削除することはできません。

Bootable Disk は、仮想マシンに割り当てられているディスクがない場合、**PXE** ソースからプロビジョニングされる仮想マシンには必須ではありません。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** として 1 つを選択する必要があります。

手順

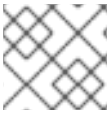
1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. **Create Virtual Machine**をクリックし、**Create with Wizard** を選択します。
3. すべての必須の Basic Settings に入力します。**Template** を選択すると、これらのフィールドへの入力が自動的に行われます。
4. **Next** をクリックして Networking 画面に進みます。デフォルトで **nic0** NIC が割り当てられます。
   a. (オプション) **Create NIC** 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。

1.2.1.2. YAML 設定ファイルを使用した仮想マシンの作成

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。

1.2.1.3. 仮想マシンの編集

Web コンソールで YAML を直接編集するか、または **Virtual Machine Overview**画面から仮想マシンの一部の値を編集できます。

Virtual Machine Overview画面から編集する場合、仮想マシンは **オフ** である必要があります。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Edit** をクリックして、編集可能なフィールドを選択可能にします。
4. **Flavor** は **Custom** のみに変更できます。これにより、**CPU** および **Memory** の追加フィールドが提供されます。
5. **Save** をクリックします。

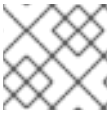
操作の処理後に、更新された値が表示されます。

1.2.1.4. 仮想マシンの YAML の編集

仮想マシンの YAML 設定は、Web コンソール内で直接編集できます。

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。
    a. (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。
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。

1.2.1.5. 仮想マシンのイベントの表示

実行中の仮想マシンのストリームイベントは、Web コンソールの **Virtual Machine Details**画面から確認できます。

▮▮ ボタンはイベントストリームを一時停止します。
▶ ボタンは一時停止されたイベントストリームを継続します。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Events** をクリックして、仮想マシンのすべてのイベントを表示します。

1.2.1.6. 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. 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。
    a. 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。

1.2.2. 仮想マシンの制御

1.2.2.1. 仮想マシンの起動

仮想マシンは、**Workloads → Virtual Machines** 一覧でそれぞれの仮想マシンの ⋮ ボタンを使用して起動できます。

同じ制御操作は、Virtual Machine Details画面の Actions コントロールを使用して実行できます。

手順

1. サイドメニューから Workloads → Virtual Machine をクリックします。
2. 仮想マシンの ⋮ ボタンをクリックし、Start Virtual Machineを選択します。
    a. 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 をクリックして Start Virtual Machineを選択します。
3. 確認のポップアップウィンドウで、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、仮想マシンは 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、Importing の状態になります。この状態は、イメージのサイズによって数分の時間続く場合があります。

1.2.2.2. 仮想マシンの停止

実行中の仮想マシンは、Workloads → Virtual Machines 一覧でそれぞれの仮想マシンの ⋮ ボタンを使用して停止できます。

同じ制御操作は、Virtual Machine Details画面の Actions コントロールを使用して実行できます。

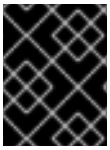
手順

1. サイドメニューから Workloads → Virtual Machine をクリックします。
2. 仮想マシンの ⋮ ボタンをクリックし、Stop Virtual Machineを選択します。
    a. 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 クリックして Stop Virtual Machineを選択します。
3. 確認のポップアップウィンドウで、Stop をクリックし、仮想マシンを停止します。

1.2.2.3. 仮想マシンの再起動

実行中の仮想マシンは、Workloads → Virtual Machines 一覧でそれぞれの仮想マシンの ⋮ ボタンを使用して再起動できます。

同じ制御操作は、Virtual Machine Details画面の Actions コントロールを使用して実行できます。



重要

仮想マシンのステータスが Importing の間は仮想マシンを再起動しないでください。この状態で再起動すると、仮想マシンのエラーが発生します。これは既知の問題です。

手順

1. サイドメニューから Workloads → Virtual Machine をクリックします。
2. 仮想マシンの ⋮ ボタンをクリックし、Restart Virtual Machineを選択します。

a. 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** クリックして **Restart Virtual Machine**を選択します。

3. 確認のポップアップウィンドウで、**Restart** をクリックし、仮想マシンを再起動します。

1.2.3. 仮想マシンコンソールへのアクセス

1.2.3.1. 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 Console|Serial Console ドロップダウン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 **シリアルコンソール** セッションを仮想マシンに対して手動で開きます。

1.2.3.2. 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. サイドメニューから **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。

1.2.3.3. 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 **シリアルコンソール** に接続します。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 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。

1.2.4. 仮想マシン NIC の管理

1.2.4.1. 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. サイドメニューから **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**、および **MAC ADDRESS** の詳細を入力します。
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。

1.2.4.2. 仮想マシンからの NIC の削除

NIC を仮想マシンから削除すると、NIC の割り当てが解除され、永続的に削除されます。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
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. 削除する NIC の ⋮ ボタンをクリックし、**Delete** を選択します。
5. 確認のポップアップウィンドウで、**Delete** をクリックして NIC の割り当てを解除し、これを削除します。

1.2.5. 仮想マシンディスクの管理

1.2.5.1. 仮想マシンのディスクの作成

Web コンソールから追加のストレージディスクを作成し、これを仮想マシンに割り当てます。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Disks** をクリックして、仮想マシンにすでに割り当てられているディスクを表示します。

4. **Create Disk** をクリックして、新規スロットを一覧に作成します。
5. 新規ディスクの **NAME**、**SIZE**、およびオプションの **STORAGE CLASS** の詳細を入力します。
6. ✓ ボタンをクリックしてディスクを保存し、これを仮想マシンに割り当てます。

1.2.5.2. 仮想マシンからのディスクの削除

ディスクを仮想マシンから削除すると、ディスクの割り当てが解除され、永続的に削除されます。

手順

1. サイドメニューから **Workloads → Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Disks** をクリックして、仮想マシンにすでに割り当てられているディスクを表示します。
4. 削除するディスクの ⋮ ボタンをクリックし、**Delete** を選択します。
5. **Confirm** をクリックし、ディスクの割り当てを解除し、これを削除します。

1.2.6. 仮想マシンテンプレートの管理

1.2.6.1. 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

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は、仮想マシンの作成時にテンプレートを参照できます。

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。

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** を変更することはできますが、これを削除することはできません。

Bootable Disk は、仮想マシンに割り当てられているディスクがない場合、**PXE** ソースからプロビジョニングされる仮想マシンには必須ではありません。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 として 1 つを選択する必要があります。

手順

1. サイドメニューから **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 に入力します。
4. **Next** をクリックして Networking 画面に進みます。デフォルトで **nic0** 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。テンプレートから作成される仮想マシンでは、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 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。

1.2.6.2. 仮想マシンテンプレートの YAML の編集

仮想マシンテンプレートの YAML 設定は、Web コンソール内で直接編集できます。

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。

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 Template**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。

a. (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。

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。

1.2.6.3. 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。

Workloads → Virtual Machines Templates 一覧でテンプレートの ⋮ ボタンを使用するか、または **Virtual Machine Templates Details**画面の **Actions** コントロールを使用して仮想マシンテンプレートを削除します。

手順

1. サイドメニューから **Workloads → Virtual Machine Templates** をクリックします。
2. 削除するテンプレートの ⋮ ボタンをクリックし、**Delete Template** を選択します。
   a. または、テンプレート名をクリックして **Virtual Machine Template Details**画面を開き、**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。

1.3. CLI 操作

1.3.1. 作業を開始する前に

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

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 で使用する **oc** コマンドが記載されています。

表1.1 oc コマンド

コマンド	説明
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>	特定のリソースの詳細を表示します。
oc create -f <config>	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 process -f <config>	テンプレートを設定ファイルで処理します。テンプレートには、作成時に生成されるか、またはユーザーによって設定される「パラメーター」とテンプレートを記述するメタデータが含まれます。
oc apply -f <file>	設定をファイル名または標準入力 (stdin) 別のリソースに適用します。

OpenShift Container Platform クライアントの具体的な情報については、『OpenShift Container Platform CLI リファレンスガイド』を参照するか、または **oc --help** コマンドを参照してください。

1.3.1.2. Virtctl 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 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1.2 Virtctl クライアント

コマンド	説明
virtctl start <vm>	仮想マシンを起動し、仮想マシンインスタンスを作成します。
virtctl stop <vmi>	仮想マシンインスタンスを停止します。
virtctl restart <vmi>	仮想マシンインスタンスを再起動します。
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 <...>	クライアントマシンからクラスターに仮想マシンディスクをアップロードします。

1.3.1.3. 適切な OpenShift Container Platform プロジェクトであることの確認

シェルまたは Web コンソールを使用してオブジェクトを変更する前に、適切なプロジェクトを使用していることを確認します。シェルでは、以下のコマンドを使用します。

コマンド	説明
oc projects	利用可能なすべてのプロジェクトを一覧表示します。現在のプロジェクトには、アスタリスクのマークが付けられます。
oc project <project_name>	別のプロジェクトに切り替えます。
oc new-project <project_name>	新規プロジェクトを作成します。

Web コンソールで、**Project** 一覧をクリックし、該当するプロジェクトを選択するか、または新規プロジェクトを作成します。

1.3.2. 仮想マシンのネットワークの設定

1.3.2.1. ゲスト IP アドレスの表示

QEMU ゲストエージェントを仮想マシンにインストールして、Linux 仮想マシンに割り当てられている IP アドレスを確認できます。ゲストエージェントが実行されている場合、コマンドラインで VMI ステータスをチェックし、各インターフェースの仮想マシン IP アドレスを確認できます。

手順

1. 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
```

2. 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
```

3. VMI の IP アドレス情報を表示します。

```
$ 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
```

注記

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 -o yaml** を実行して表示できます。

1.3.2.2. 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。

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: VM
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

1.3.3. 仮想マシンおよびディスクイメージのインポートおよびアップロード

1.3.3.1. ローカルディスクイメージの新規 PVC へのアップロード

virtctl image-upload を使用して、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 OpenShift Container Platform クラスターにアップロードできます。これにより、アップロードの完了後に仮想マシンに関連付けられる PVC が作成されます。

前提条件

- RAW または QCOW2 形式の仮想マシンディスクイメージ。これは **xz** または **gzip** を使用して圧縮できます。
- **kubevirt-virtctl** はクライアントマシンにインストールされている必要があります。
- クライアントマシンは、OpenShift ルーターの証明書を信頼できるように設定されている必要があります。

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. 既存の passthrough ルートを削除します。

```
$ oc delete route -n cdi cdi-uploadproxy-route
```

3. re-encryption termination を使用してセキュリティー保護されたルートを作成します。

```
$ oc get secret -n cdi cdi-upload-proxy-ca-key -o=jsonpath="{.data['tls\.crt']}" | base64 -d > ca.pem

$ oc create route reencrypt cdi-uploadproxy-route -n cdi --service=cdi-uploadproxy --dest-ca-cert=ca.pem
```

4. **virtctl image-upload** コマンドを使用して VM イメージをアップロードし、選択したパラメーターが含まれていることを確認します。以下は例になります。

```
$ virtctl image-upload --uploadproxy-url=https://$(oc get route cdi-uploadproxy-route -n cdi -o=jsonpath='{.status.ingress[0].host}') --pvc-name=upload-fedora-pvc --pvc-size=10Gi --image-path=/images/fedora28.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。

5. 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
```

次に、PVC にバインドする仮想マシンオブジェクトを作成できます。

1.3.3.2. DataVolume を使用した既存の仮想マシンイメージのインポート

DataVolume オブジェクトは、基礎となる PVC に関連付けられたインポート、クローンおよびアップロード操作のオーケストレーションを提供します。DataVolume は KubeVirt と統合し、それらは仮想マシンが PVC の作成前に起動するのを防ぎます。

注意

ディスクイメージを PersistentVolumeClaim 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、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

- 仮想マシンディスクには RAW または QCOW2 形式を使用でき、**xz** または **gz** を使用して圧縮できます。
- ディスクイメージは、**HTTP** または **S3** エンドポイントのいずれかで利用可能にする必要があります。

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** または **S3** ファイルサーバーを特定します。正しい形式の完全な URL が必要になります。

   - http://www.example.com/path/to/data
   - **s3://bucketName/fileName**

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、これ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: ""  # <optional: your key or user name, base64 encoded>
  secretKey:    "" # <optional: your secret or password, base64 encoded>
```

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

3. VM 設定ファイル (オプションで **secretRef** パラメーターが含まれる) を編集します。以下のサンプルでは、Fedora イメージを使用しています。

```
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
          secretRef: "" # Optional
    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: {}
```

4. 仮想マシンを作成します。

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

仮想マシンおよび DataVolume が作成されます。CDI コントローラーは、正しいアノテーションを使用して基礎となる PVC を作成し、インポートプロセスを開始します。インポートが完了すると、DataVolume ステータスは **Succeeded** に切り替わり、仮想マシンの起動が可能になります。

DataVolume プロビジョニングはバックグラウ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。仮想マシンを起動できますが、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 ステータスをモニターします。

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

1 VirtualMachine 設定ファイルで指定された 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>
```

1.3.3.3. 仮想マシンディスクの PVC へのインポート

仮想マシンディスクのインポートプロセスは CDI コントローラーによって処理されます。PVC が特殊な **cdi.kubevirt.io/storage.import** アノテーションを使って作成される場合、コントローラーは PV に割り当てられる有効期限の短いインポート Pod を作成し、仮想ディスクイメージを PV にダウンロードします。

OpenShift Container Platform インベントリーファイルの **openshift_docker_insecure_registries** 属性を使ってレジストリーに insecure (非セキュア) とマークする場合でも、Containerized Data Importer を使用してセキュアでないレジストリーからイメージをインポートすることはできません。

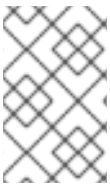
注意

ディスクイメージを PersistentVolumeClaim 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、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

- 仮想マシンディスクには RAW または QCOW2 形式を使用でき、**xz** または **gzip** を使用して圧縮できます。
- ディスクイメージは、**HTTP** または **S3** エンドポイントのいずれかで利用可能にする必要があります。



注記

ローカルにプロビジョニングされるストレージの場合、 PV は PVC の前に作成される必要があります。これは、PV が動的に作成される OpenShift Container Storage の場合には不要です。

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** または **S3** ファイルサーバーを特定します。いずれかの形式の完全な URL が必要になります。
   - **http://www.example.com/path/to/data**
   - **s3://bucketName/fileName**
     この URL を、PVC 設定ファイルの **cdi.kubevirt.io/storage.import.endpoint** アノテーションの値として使用します。

     例: **cdi.kubevirt.io/storage.import.endpoint: https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-Cloud-Base-28-1.1.x86_64.qcow2**
2. ファイルサーバーに認証情報が必要な場合、**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: ""  # <optional: your key or user name, base64 encoded>
  secretKey:    "" # <optional: your secret or password, base64 encoded>
```

a. PVC 設定ファイルの **cdi.kubevirt.io/storage.import.secret** アノテーションで使用する **metadata.name** の値を保存します。
例: **cdi.kubevirt.io/storage.import.secret: endpoint-secret**

3. **endpoint-secret.yaml** をクラスターに適用します。

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

4. PVC 設定ファイルを編集し、必要なアノテーションが組み込まれていることを確認します。以下に例を示します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: "example-vmdisk-volume"
  labels:
    app: containerized-data-importer
  annotations:
    cdi.kubevirt.io/storage.import.endpoint:
"https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-Cloud-Base-28-1.1.x86_64.qcow2" (1)
    cdi.kubevirt.io/storage.import.secret: "endpoint-secret" (2)
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

(1) インポートイメージ URL のエンドポイントアノテーション

(2) 認可シークレットのエンドポイントアノテーション

5. **oc** CLI を使用して PVC を作成します。

```
$ oc create -f <pvc.yaml> (1)
```

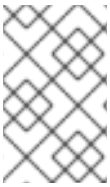
(1) PersistentVolumeClaim ファイル名。

ディスクイメージが PV に正常にインポートされた後に、インポーター Pod の有効期限は切れ、PVC を OpenShift Container Platform 内の仮想マシンオブジェクトにバインドできます。

次に、仮想マシンオブジェクトを作成し、PVC にバインドします。

1.3.3.4. dataVolumeTemplate を使用した既存 PVC のクローン作成および仮想マシンの作成

既存の仮想マシンの 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。仮想マシン仕様の **dataVolumeTemplate** を参照すると、**ソース** 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

- 既存の仮想マシンディスクの PVC。関連付けられた仮想マシンの電源はオフにする必要があります。そうでないと、クローンプロセスは PVC が利用可能になるまでキューに入れられます。

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 DataVolume 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 **<vm-dv-clone>** は **<my-favorite-vm-disk>** (**<source-namespace>** namespace にある) のクローンを作成し、仮想マシンで **<favorite-clone>** ボリュームとして参照される、**2Gi <favorite-clone>** DataVolume を作成します。
以下に例を示します。

```
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"
```

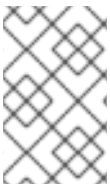
1 作成する仮想マシン。

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-dvt>.yaml
```

1.3.3.5. 既存の仮想マシンディスクの PVC クローンの作成

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



注記

DataVolume が仮想マシンと別に作成される場合、DataVolume のライフサイクルは仮想マシンとは切り離されます。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

- 既存の仮想マシンディスクの PVC。関連付けられた仮想マシンの電源はオフにする必要があります。そうでないと、クローンプロセスは PVC が利用可能になるまでキューに入れられます。

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 DataVolume を確認します。
2. 以下のパラメーターを指定する DataVolume オブジェクトの YAML ファイルを作成します。

metadata: name	新規 DataVolume の名前。
source: pvc: namespace	ソース PVC が存在する namespace。
source: pvc: name	ソース PVC の名前。

storage	新規 DataVolume のサイズ。十分な領域が割り当てられていることを確認します。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。
	
	以下に例を示します。
	
	apiVersion: cdi.kubevirt.io/v1alpha1
	kind: DataVolume
	metadata:
	  name: cloner-datavolume
	spec:
	  source:
	  pvc:
	  namespace: "<source-namespace>"
	  name: "<my-favorite-vm-disk>"
	  pvc:
	  accessModes:
	  - ReadWriteOnce
	  resources:
	  requests:
	  storage: 2Gi

3. DataVolume を作成して PVC のクローンを開始します。

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

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。

1.3.4. 空のディスクイメージの新規作成

1.3.4.1. DataVolume マニフェストを使用した空のディスクイメージの作成

空のディスクを使用して、ストレージ容量を拡大するか、または新規データパーティションを作成できます。新規の空のディスクイメージは、DataVolume マニフェストファイルを使って **PersistentVolumeClaim** で作成できます。

前提条件

- Container-native Virtualization 1.4
- 1 つ以上の利用可能な **PersistentVolume**

手順

1. DataVolume マニフェストファイルを作成します。

```
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. 空のディスクイメージを作成するために DataVolume マニフェストをデプロイします。

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

1.3.4.2. PVC マニフェストを使用した空のディスクイメージの作成

空のディスクを使用して、ストレージ容量を拡大するか、または新規データパーティションを作成できます。新規の空のディスクイメージは、PVC マニフェストファイルを使って **PersistentVolumeClaim** で作成できます。

前提条件

- Container-native Virtualization 1.4
- 1 つ以上の利用可能な **PersistentVolume**

手順

1. PVC マニフェストファイルを作成します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: "blank-image-pvc"
  labels:
    app: containerized-data-importer
  annotations:
    cdi.kubevirt.io/storage.import.source: "none"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
  # Optional: Set the storage class or omit to accept the default
  # storageClassName: local
```

2. 空のディスクイメージを作成するために PVC マニフェストをデプロイします。

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

1.3.5. 仮想マシンの管理

1.3.5.1. CLI での新規の仮想マシンの作成

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。

関連する PVC **claimName** をボリュームとして参照して、仮想マシンディスクを仮想マシンに割り当てます。

注記

現時点で、ReplicaSet は Container-native Virtualization ではサポートされていません。

ボリュームタイプおよび設定ファイルの例については、「リファレンス」セクションを参照してください。

表1.3 ドメイン設定

設定	説明
core	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) で単位を指定します。
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1.4 ボリューム設定

設定	説明
name	ボリュームの名前。DNS_LABEL 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。

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。

OpenShift Container Platform クライアントで仮想マシンを作成するには、以下を実行します。

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

仮想マシンは **Stopped** 状態で作成されます。仮想マシンを起動し、仮想マシンインスタンスを実行します。

1.3.5.2. 仮想マシンおよび仮想マシン PVC の削除

仮想マシンを削除する際に、これが使用する PVC のバインドは解除されます。この PVC を異なる仮想マシンにバインドする予定がない場合には、これも削除します。

-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。

```
$ oc delete vm fedora-vm
```

```
$ oc delete pvc fedora-vm-pvc
```

1.3.6. 仮想マシンの制御

1.3.6.1. 仮想マシンの制御

仮想マシンの現在の状態に応じて、これを起動したり、停止したり、再起動したりできます。

virtctl クライアントユーティリティーを使用して、仮想マシンの状態を変更し、仮想マシンへの仮想コンソールセッションを開き、仮想マシンポートをサービスとして公開します。

virtctl 構文は **virtctl <action> <vm_name> <options>** です。

-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制御できます。

例:

```
$ virtctl start example-vm
```

```
$ virtctl restart example-vm
```

```
$ virtctl stop example-vm
```

oc get vm はプロジェトの仮想マシンを一覧表示します。**oc get vmi** は実行中の仮想マシンインスタンスを一覧表示します。

1.3.7. 仮想マシンコンソールへのアクセス

1.3.7.1. VMI 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

- アクセスする仮想マシンインスタンスは実行中である必要があります。

手順

1. **virtctl** を使ってシリアルコンソールに接続します。

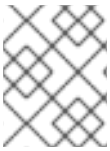
```
$ virtctl console <VMI>
```

1.3.7.2. VNC を使用した VMI 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** を使用して、実行中の仮想マシンインスタンスへのグラフィカルコンソールを開くことができます。これは **virt-viewer** パッケージでインストールされます。

前提条件

- **virt-viewer** がインストールされている必要があります。
- アクセスする仮想マシンインスタンスは実行中である必要があります。



注記

リモートマシン上で **SSH** 経由で **virtctl** を使用する場合、この手順を機能させるには、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っ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
```

1.3.7.3. SSH 経由での仮想マシンインスタンスへのアクセス

SSH を使用して仮想マシンにアクセスできますが、まずポート 22 を仮想マシンで公開する必要があります。

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実行中である必要があります。

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
```

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
```

3. ノードの **ipAddress** および手順 2 で見つけたポートを使用して、SSH 経由で仮想マシンインスタンスにログインします。

```
$ ssh username@<node IP> -p 32551
```

1.3.8. イベント、ログ、エラー、およびメトリクス

1.3.8.1. イベント

OpenShift Container Platform イベントは、プロジェクト内の重要なライフサイクル情報のレコードであり、リソースのスケジュール、作成、および削除に関する問題のモニターおよびトラブルシューティングに役立ちます。

プロジェクトの **イベント** を取得するには、以下を実行します。

```
$ oc get events
```

イベントはリソース説明に組み込むこともできます。これは OpenShift Container Platform クライアントを使用して取得できます。

```
$ oc describe <resource_type> <resource_name>
$ oc describe vm <fedora-vm>
$ oc describe vmi <fedora-vm>
$ oc describe pod virt-launcher-fedora-vm-<random>
```

リソース説明には、設定、スケジュールおよびステータスの詳細も含まれます。

1.3.8.2. ログ

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。仮想マシンのログは、仮想マシンランチャー Pod から取得されます。

```
$ oc logs virt-launcher-fedora-vm-zzftf
```

-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イメージの参照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。

1.3.8.3. メトリクス

OpenShift Container Platform メトリクスは、クラスターのノード、コンポーネント、およびコンテ

ナーのメモリー、CPU、およびネットワークパフォーマンスの情報を収集します。収集される具体的な情報は、メトリクスサブシステムが設定される方法によって異なります。メトリクスの設定についての詳細は、『OpenShift Container Platform クラスター設定ガイド』を参照してください。

oc CLI コマンド **adm top** は Heapster API を使用して、クラスターの Pod およびノードの現在の状態についてのデータをフェッチします。

Pod のメトリクスを取得するには、以下を実行します。

```
$ oc adm top pod <pod_name>
```

クラスター内のノードのメトリクスを取得するには、以下を実行します。

```
$ oc adm top node
```

OpenShift Container Platform Web コンソールは、一定の期間におけるメトリクス情報をグラフィカルに表示できます。

1.4. MICROSOFT WINDOWS 仮想マシンの VIRTIO ドライバーの管理

1.4.1. Microsoft Windows 仮想マシンの VirtIO ドライバー

VirtIO ドライバーは、Microsoft Windows 仮想マシンが Container-native Virtualization で適切に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 **virtio-win** コンテナーディスクで利用できます。

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られる必要があります。VirtIO ドライバーは、仮想マシンへの Windows のインストール時にインストールするか、または既存の Windows インストールに追加できます。

ドライバーのインストール後に、**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。

1.4.2. 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インストールするか、または Windows 仮想マシンに 追加するには、 **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。

手順

- **cnv-tech-preview/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: cnv-tech-preview/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。**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。

仮想マシンが作成され、起動された後に、VirtIO ドライバーは仮想マシン上の Windows のインストール時に割り当てられた SATA CD ドライブからインストールするか、または既存の Windows インストールに追加することができます。

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

1.4.3. 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 VirtIO ドライバーを仮想マシンにインストールします。最低でも、サポートされているストレージドライバーをインストールし、 Windows インストールのストレージの宛先を選択する必要があります。

前提条件

- VirtIO ドライバーコンテナーディスクが SATA CD ドライブとして割り当てられている仮想マシン。
- 仮想マシンがアクセスできる Windows インストールメディア。

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 ドライバー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。

Windows のインストール後に、仮想マシン設定ファイルから VirtIO ドライバーコンテナーディスクを削除できます。

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

VirtIO ドライバーを既存の Windows 仮想マシンにインストールします。

前提条件

- VirtIO ドライバーコンテナーディスクが SATA CD ドライブとして割り当てられている Windows 仮想マシン。

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。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 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、**不明なデバイス** 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する必要がある場合があります。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、ドロップダウンリストで **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、ドライバー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. 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。
8. 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。

ドライバーのインストール後に、仮想マシン設定ファイルからディスクを削除できます。

1.4.5. 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、**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。**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. Web コンソールで仮想マシン設定を編集するか、または選択するエディターで設定ファイルを編集し、**disk** および **volume** を削除します。

```
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: cnv-tech-preview/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。

1.5. 高度な仮想マシン設定

1.5.1. 仮想マシンのネットワークソースとしての Open vSwitch ブリッジの使用

Container-native Virtualization を使用すると、仮想マシンインスタンスを、ノード上で設定される Open vSwitch ブリッジに接続できます。

前提条件

- OpenShift Container Platform 3.11 以降を実行するクラスター

手順

1. クラスターのホストネットワークを準備します (オプション)。
ホストネットワークにボンディングなどの追加の設定変更が必要な場合は、『Red Hat Enterprise Linux ネットワークガイド』を参照してください。

2. すべてのクラスターホストでインターフェースおよびブリッジを設定します。
各ホストで、必要なネットワークに接続されたインターフェースを選択します。次に、Open vSwitch ブリッジを作成し、ブリッジのポートとして選択したインターフェースを指定します。

   この例では、ブリッジ **br1** を作成し、これをインターフェース **eth1** に接続します。このブリッジはすべてのノードで設定される必要があります。これがノードのサブセットでのみ利用可能な場合は、VMI の **nodeSelector** 制約が有効であることを確認してください。

注記

`eth1` への接続はインターフェースがブリッジに割り当てられると失われるため、別のインターフェースがホスト上になければなりません。

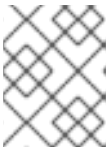
```
$ ovs-vsctl add-br br1
$ ovs-vsctl add-port br1 eth1
$ ovs-vsctl show
```

```
8d004495-ea9a-44e1-b00c-3b65648dae5f
    Bridge br1
        Port br1
            Interface br1
                type: internal
        Port "eth1"
            Interface "eth1"
    ovs_version: "2.8.1"
```

3. クラスター上にネットワークを設定します。
L2 ネットワークはクラスター全体のリソースとして処理されます。ネットワークをネットワーク割り当て定義 YAML ファイルで定義します。ネットワークは **NetworkAttachmentDefinition** CRD を使用して定義できます。

NetworkAttachmentDefinition CRD オブジェクトには、Pod とネットワーク間の割り当てについての情報が含まれます。以下の例では、Open vSwitch ブリッジ **br1** への割り当てがあり、トラフィックは VLAN 100 にタグ付けられています。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vlan-100-net-conf
spec:
  config: '{
      "cniVersion": "0.3.1",
      "type": "ovs",
      "bridge": "br1",
      "vlan": 100
    }'
```



注記

"vlan" はオプションです。これが省略される場合、VMI はトランクを経由して割り当てられます。

4. 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。
ネットワークが直前に作成された **NetworkAttachmentDefinition** に接続されるように指定します。このシナリオでは、 **vlan-100-net** は **vlan-100-net-conf** という **NetworkAttachmentDefinition** に接続されます。

```
networks:
- name: default
  pod: {}
- name: vlan-100-net
  multus:
    networkName: vlan-100-net-conf
```

VMI を起動すると、**eth0** インターフェースはデフォルトのクラスターネットワークに接続し、**eth1** は、VMI を実行するノードでブリッジ **br1** を使用して VLAN 100 に接続します。

1.5.2. MAC アドレスを指定した PXE ブート

PXE ブートまたはネットワークブートは Container-native Virtualization でサポートされています。

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、新規ホストのデプロイ時にこれを使用して、PXE サーバーから必要な OS イメージを選択できます。

「リファレンス」セクションには、PXE ブートの設定ファイルテンプレートがあります。

前提条件

- OpenShift Container Platform 3.11 以降を実行するクラスター
- PXE ブートを許可する設定済みのインターフェース

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** 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",
      "type": "ovs",
      "bridge": "br1"
    }'
```

注記

この例では、VMI はトランクポート経由で Open vSwitch ブリッジ **<br1>** に割り当てられます。

   b. Open vSwitch ブリッジ **<br1>** を作成し、これをインターフェース **<eth1>** に接続します。これは、PXE ブートを許可するネットワークに接続されます。

```
$ ovs-vsctl add-br br1
$ ovs-vsctl add-port br1 eth1
$ ovs-vsctl show
8d004495-ea9a-44e1-b00c-3b65648dae5f
    Bridge br1
        Port br1
            Interface br1
                type: internal
        Port "eth1"
            Interface "eth1"
    ovs_version: "2.8.1"
```

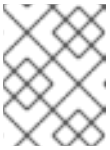
注記

このブリッジはすべてのノードで設定される必要があります。これがノードのサブセットでのみ利用可能な場合、 VMI の **nodeSelector** 制約が有効であることを確認してください。

2. 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. OS プロビジョニングの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. ネットワークが直前に作成された **NetworkAttachmentDefinition** に接続されるように指定します。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
   ```

3. 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
```

4. 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
```

5. 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
```

6. 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。

7. VMI にログインします。

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

8. 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指定された MAC アドレスがあ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
```

1.5.3. 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。

前提条件

- OpenShift Container Platform 3.11 以降を実行するクラスター

手順

VMI 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、**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、VMI 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、VMI 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 VMI に適用されます。

1.5.4. 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できます。

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によってノード上の VMI の密度を増やすことは可能です。

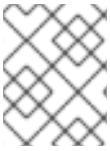
前提条件

- OpenShift Container Platform 3.11 以降を実行するクラスター

手順

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。

1.6. クラスターのメンテナンスタスク

1.6.1. 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、期限が切れる前に手動で更新する必要があります。

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。

- 移行はキャンセルされます
- イメージのアップロードはキャンセルされます
- 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、クラスターに対するアクティブなセッションを使用し、**kubevirt-hyperconverged** 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 kubevirt-hyperconverged
```

TLS 証明書は更新され、1 年間有効になります。

1.7. リファレンス

1.7.1. 仮想マシンウィザードのフィールド

Name	パラメーター	説明
Name		仮想マシンの名前。英数字の文字のみを使用でき、最長長さは 63 です。
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 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にある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**
Operating System		クラスターの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。
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	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汎用的な設定。
	highperformance	仮想マシンは、高パフォーマンスの負荷に対して最適化されたより効率的な設定を持ちます。
Start virtual machine on creation		このチェックボックスを選択し、作成時に仮想マシンを自動的に起動します。

Name	パラメーター	説明
cloud-init		このチェックボックスを選択し、cloud-init fields を有効にします。

1.7.2. 仮想マシンテンプレートウィザードのフィールド

Name	パラメーター	説明
Name		仮想マシンテンプレートの名前。英数字の文字のみを使用でき、最長長さは 63 です。
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 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にある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**
Operating System		クラスターの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。
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	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汎用的な設定。

Name	パラメーター	説明
	highperformance	仮想マシンは、高パフォーマンスの負荷に対して最適化されたより効率的な設定を持ちます。
cloud-init		このチェックボックスを選択し、cloud-init fields を有効にします。

1.7.3. Cloud-init フィールド

Name	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	ユーザーのパブリックキー。これは、仮想マシンの **~/.ssh/authorized_keys** にコピーされます。
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ことのできるテキストボックスに置き換えます。

1.7.4. ネットワークフィールド

Name	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、そのセッションで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。
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。

1.7.5. ストレージフィールド

Name	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。
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使用する利用可能なディスクの一覧。仮想マシンの Provision Source が **URL** または **Container** である場合に **rootdisk** に固定されます。

1.7.6. 仮想マシンのアクション

表1.5 アクション

アクション	利用可能になる場合の状態	説明
Start Virtual Machine	**Off**	仮想マシンを起動します。
Stop Virtual Machine	**Running** または **Other**	仮想マシンを停止します。
Restart Virtual Machine	**Running** または **Other**	実行中の仮想マシンを再起動します。
Delete Virtual Machine	**All**	クラスターから仮想マシンを永続的に削除します。

1.7.6.1. Container-native Virtualization でサポートされる Microsoft Windows 仮想マシンの VirtIO ドライバー

表1.6 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。

ドライバー名	ハードウェア ID	説明
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。

1.7.6.2. 仮想マシンのストレージボリュームのタイプ

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ネットワークボリュームを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) はいかなる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。これは、仮想マシンデータのセッション間での永続化を可能にします。 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。

1.7.6.3. テンプレート: PVC 設定ファイル

pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: "example-vmdisk-volume"
  labels:
   app: containerized-data-importer
  annotations:
    kubevirt.io/storage.import.endpoint: "" # Required. Format: (http||s3)://www.myUrl.com/path/to/data
    kubevirt.io/storage.import.secretName: "" # Optional. The name of the secret containing credentials
for the data source
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

1.7.6.4. テンプレート: VM 設定ファイル

vm.yaml

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt-vm: fedora-vm
  name: fedora-vm
spec:
  running: false
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/domain: fedora-vm
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
        machine:
          type: ""
        resources:
          requests:
            memory: 1Gi
      terminationGracePeriodSeconds: 0
      volumes:
      - cloudInitNoCloud:
          userData: |-
            #cloud-config
            password: fedora
            chpasswd: { expire: False }
        name: cloudinitdisk
      - name: containerdisk
        persistentVolumeClaim:
          claimName: example-vmdisk-volume
status: {}
```

1.7.6.5. テンプレート: Windows VMI 設定ファイル

windows-vmi.yaml

```
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
```

1.7.6.6. テンプレート: VM 設定ファイル (DataVolume)

example-vm-dv.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:
"https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-
Cloud-Base-28-1.1.x86_64.qcow2"
  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
```

1.7.6.7. 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:
"https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-
Cloud-Base-28-1.1.x86_64.qcow2" # Or S3
        secretRef: "" # Optional
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: "1G"
```

1.7.6.8. 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"
```

1.7.6.9. テンプレート: PXEブート用 VMI 設定ファイル

vmi-pxe-boot.yaml

```
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: {}
```